Shopping cart web application development

Joonas Niinimäki

Thesis, AMK
May 2023
ICT
Information – and communications technology

jamk

Kuvailulehti

Joonas Niinimäki

Shopping cart web application development

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2023, 36 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikka. Opinnäytetyö AMK.

Julkaisun kieli: Englanti

Julkaisulupa avoimessa verkossa: Kyllä

Tiivistelmä

Tutkielma tutkii web applikaation suunnittelua ja kehittämistä, sekä siihen liittyvien hyvien kehittämisperiaatteiden soveltamista. Web ohjelmoinnin osuus teollisessa sovelluskehityksessä on nostanut suosiotaan viimeisen vuosikymmenen aikana ja yhä useampi tuote on jollain tavalla web ohjelmointia vaativa sovellus tai kehitysalusta.

Raportissa käydään läpi yksityiskohtaisesti, miten web sovellusta kehitetään, mitä asioita pitää huomioida niiden suunnittelussa, mitkä ovat hyvät käytänteet web ohjelmoinnissa, mitä teknologiaratkaisuja web ohjelmoinnissa on tarjolla ja mitä kipukohtia tai kehittämistarpeita web ohjelmoinnissa on teollisuudessa? Tavoitteena on vastata millainen moderni web applikaatio ratkaisu voisi olla.

Tutkimus hyödyntää soveltavaa kahta eri tutkimusmetodologiaa, joista ensimmäinen on soveltava insinööri tutkimusmetodologia sovelluskehittämisen prosessien tutkimista varten ja toinen metodologia on tapaustutkimus, jolla pyritään selvittämään mitä tarkoittaa laadukas web applikaatio kokonaisuudessaan ja mitkä ovat ne tekijät, jotka laadukkaiksi web applikaatioksi koetut kehitysperiaatteet muodostavat. Tulos saavutetaan ristiin vertailemalla kahta julkaistua web applikaatiota, joilla on yleisesti samankaltaiset käyttötapaus tarkoitukset ja tuloksiin perustuen rakentamalla tapaustutkimus matriisin. Lopullinen tulos saavutetaan analysoimalla tutkittavaa web applikaation kehitysvaiheita, sekä alan kirjallisuuteen perustuvaa lähdemateriaalia.

Tulokset osoittivat, että teesin projektin web applikaatio kykeni seuraamaan useita modernin web applikaation kehittämisen trendejä. Huomioitavana tekijöinä on käytettävissä olleen ajan rajoitteet, käytössä olevat resurssit ja kehittäjän kokemuksen määrä. Lopputuloksesta pystyi päättelemään, että web applikaation kehittäminen vie paljon aikaa ja tämä projekti kykeni tuottamaan vain osittain implementoidun mock-up prototyypin oikeasta sovelluksesta. Viimeisessä luvussa tutkija ehdottaa millä tavoin applikaatiota voisi jatkokehittää tulevaisuudessa ja mitä huomiointeja pitäisi pohtia, mikäli aikoo suunnitella web applikaatiota.

Avainsanat (asiasanat)

Web Ohjelmointi, Sovelluskehitys, Käytettävyys, Ohjelmistosuunnittelu, MERN

Muut tiedot (salassa pidettävät liitteet)

NaN

Description

Joonas Niinimäki

Shopping cart web application development

Jyväskylä: JAMK University of Applied Sciences, May 2023, 36 pages

ICT-Engineering. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

The thesis investigations web application design and development as well as application of good practice principles related to it. The share of web application development has increased in its popularity over the past decade and increasingly many products are at least in some ways require a web application or web development platform.

The report goes through a web application development process in detail, exploring how a web application is developed, what elements should be considered when designing a web application, what are good principles in web development, what kind of technology solutions are available in web development and what kind of shortcomings and development demands are in web development within the industry. It aims to answer what a modern web application solution could be like.

The study uses two different kinds of research methodologies, one of which is an applicative engineering methodology designed to study web development processes and then a case study research methodology to examine what a high quality web application means in totality and what are the variables that formulate the perceived high quality web applications by doing a cross-comparison examination with two other published web applications that have general use case purpose and building a case study matrix based on the results. The conclusive result will be accomplished by analyzing the development phases of web application development and examining the literature of the subject matter.

The research results indicated that the thesis project web application was able to follow many of the general trends of modern web application development. There are considerable factors in time constraints, available resources, and development experience. It became clear that web application development takes a lot of hours to develop, and this project was only able to produce a partially implemented mock-up for a prototype of a real application. In the closing chapter the author makes some suggestions on how to develop the application in the future and what considerations should be thought of whenever designing a web application.

Keywords/tags (subjects)

Web Development, Application development, Software Design, MERN

Miscellaneous (Confidential information)

NaN

Contents

Tables

Abbreviations

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
BaaS	Backend as a Service
CRUD	Create, Read, Update and Delete
DRY	Don’t Repeat Yourself
Express	A library for routing, handling and using HTTP requests
FaaS	Function as a Service
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
IaaS	Infrastructure as a Service
IP	Internet Protocol
MEAN	MongoDB, Express, Angular and NodeJS
MERN	MongoDB, Express, React and NodeJS
MEVN	MongoDB, Express, Vue and NodeJS
NodeJS	A framework for server development
NoSQL	Unstructured Query Language
P2P	Peer to peer
PaaS	Platform as a Service
PWA	Progressive Web Application
React	Javascript library used for many modern web applications
REST	Representational State Transfer
SaaS	Software as a Service
SQL	Structured Query Language
XML	Extensible Markup Language

1 Introduction

1.1 The project

This thesis work aims to investigate the design and development principles of web application development. The project in this report is a personal project that will be continued after the thesis work. The goal of the thesis is to develop a full stack proof of concept of the web application that will emulate the final product.

This report will explore different web development concepts and principles, as well as to investigate what could be viable solutions for the shopping cart web application example. The approach in this thesis work will be holistic case study with the aim to get an overall view of web application methodologies, principles, and functions. It will cover a full-stack development production, while some components will be mocking a web application rather than fully implemented functionality.

The rising availability of access to the internet and the usefulness of sharing data and augmenting lives through web applications has seen a major increase in web applications in the past decades Given the increase in popularity of web development applications in the recent years and the increasing need for all kinds of web applications from news sites to life augmenting tools like the shopping-cart have increased the demand for web development skills for any developer working in the software industry (Radwan 2022).

1.2 Development problem

The goal of the project is to determine how plausible it is to produce a web application proof-of-concept mock-up in a tight time limit and then how closely does this project resemble real world application equivalents. The paper aims to answer questions like what makes a high-quality web application? What features do shopping-cart web applications often have in common? What available resources and technology stack solutions are for this purpose? What limitations or struggles may there be during development? What the industrial trends currently indicate? What features were developed and what features could be added in the future? What does user accessibility mean in modern web applications? How to produce a mock-up of a real-world application in limited time and resources?

The paper does not cover web application testing and only touches on preliminary examination of web application cybersecurity topics trough the mandatory login page the project necessitated. The paper also does not explore performance issues, resourcing on industrial level production, maintenance, performance nor raw resource costs of an industrial level production web application. The thesis project will explore the theory of different kinds of available technology stack solutions used commonly in the industry, but it will not explore producing one other than with MERN stack solution. The project being developed for this paper is exclusively focused on features mocking and quality of code and feature functionality. It is for this reason that the chosen methodologies for this study are both qualitative methodologies, of which the first engineering study and the second method will be a case study.

Since there is no definitive answer to the development problems broad questions and ultimately very few standardized guidelines to the approach of designing and developing a web application, this paper explores and gives a testable example of a list of features, arguments for choosing them and a solution that a store-front type of web application would have so that in the future it is easier to explore and start designing web applications. The project explores both the front-end and back-end development by combining them to full-stack solution and the resulting end product should give an idea of what a real-world application requires and what is the purpose of those features and how they function.

2 Research methdologies

2.1 Chosen methodologies

For the chosen methodology in this thesis, there will be two qualitative methods that are used to investigate the web applications quality and modernity. Qualitative methods are empirical methods that explore relationships of data trough observation and textual data rather than quantitative numerical data (Dybå, Prikladnicki, Rönkkö, Seaman & Sillito 2011).

First method is an engineering study approach to the web application where the developed web application features will be compared with existing similar web applications. The second method is an empirical case study method which aims to qualify and quantify the case study results into a table graph. The methodologies are limited to the thesis project end-product features specifically and how closely does the end product compare and emulate other web applications of similar purpose. For example, a comparison analysis of what features does the thesis project have compared to a real world published web application and what benefits and shortcomings each web application solution has.

2.2 Rationale for the chosen research methodologies

The justification for choosing these methodologies specifically is that in software engineering, there are not that many ways mathematically measure the quality of a web application. By doing a case study and comparing other existing similar web applications, it gives some type of comparative grounds to allow for a metric system of sorts. In this way the existing number of features is measured against the developed web application in this thesis to quantify how far the development has gotten.

The basis for choosing this method specifically is because the project is an examination of a more widely pronounced phenomena within software development industry, and specifically the growing popularity of web-application development. The purpose is to expand knowledge and understanding of the phenomena by hyper focusing on the subject matter (Wohlin & May 2021). The benefits of using a case study are that it allows developing hypothesis trough the exploration of experimental research and by focusing on the topic it gives the ability to collect great deal of information on the phenomena (Runeson & Höst 2008). For the second parameter it utilizes a use case-based metrics for feature priorities which will affect how much the mock-up is able to produce. It is important to consider that different web applications are created for different purposes and therefore their comparison may not translate exactly identically. So, the results of the case study are relative.

The other problem with lacking mathematical measurement methods is that what constitutes to as quality can be highly subjective depending on the person, their priorities and even the level of their technical knowledge on web application development side. Average user will have some kind of idea on how a web application should work, but a developer may have some further insight into a web application that is not often considered. For this part of measurement, the thesis will do a cross comparison from different source sites to try to quantify a table and compare it against the web application developed for this thesis work.

In the case of the empirical method, it was chosen as the other methodology due to the ability to extract data by cross-examining already existing and produced web applications from the web app store and comparing them to product that is developed in this thesis paper (Wohlin, Höst and Henningsson 2003). The other reason for choosing this methodology is to enable constructing a controlled experiment and analyzing the results of the experiment trough introspective and retrospective analysis (Stol, Fitzgerald 2015). The latter was achieved by the construction of cross-examination matrix which is covered in further detail in the chapter 5.

3 Theoretical framework

3.1 Web development generalization

Since the popularization of web browsers by Marc Andreessen in 1993, a web browser called Mosaic revolutionized the availability and accessibility of browsing the world wide web and popularized the use of web browsers and the internet. From year 1993 having 130 websites, the amount had risen to as high as 100 000 websites by the start of 1996 in just a short span of few years (Science and History Museum 2020).

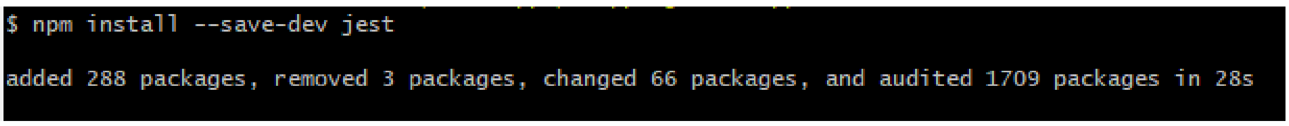
As increasingly of everyday services has been transferring to web - or mobile applications, it has created a need within the ICT industry for growing number of developers to transfer their development work into a type of web application development and solutions. In part new emerging technologies have eased up web development over the years and they have made web development faster and more efficient.

However, the increased usage and popularity of web browsers and websites has also increased demands for features, feature scopes and web security as examples. Cyberattacks like SQL injection, usage statistics to demand bigger load balancing to improve server performance rates and features that will consider different disabilities have all been absent from older websites from the 90’s and even early 2000’s, but modern web applications are more secure and consider the consumers usability a lot more than their predecessors used to.

The early websites were far more text based and featured few pictures (SQ 2020). They often did not have authentication, cookies, session controlling nor guards against cyberattacks trough websites (Chadd 2022). Modern web applications establish a secure connection, often require some sort of authenticating interaction from users and have far more balanced mix of text, pictures, link, and functionality.

3.2 Web application development

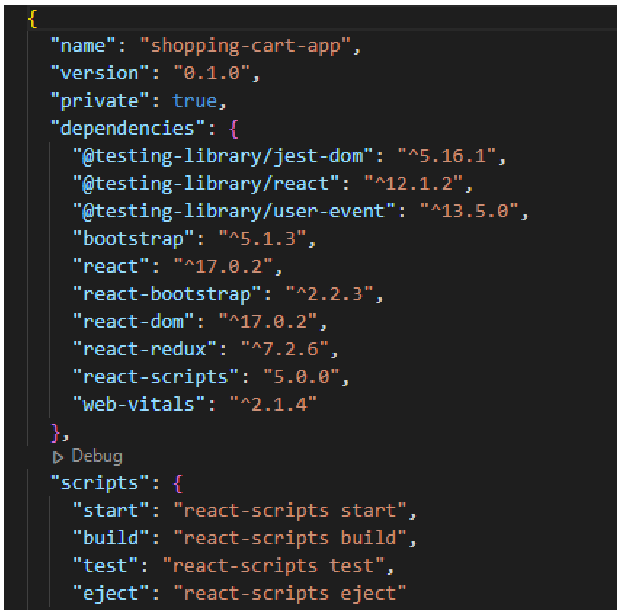
Developing a web application requires a certain level of base knowledge about the architecture of web applications and then a level of knowledge on the technology solutions within the project. There are standardized technology solutions for web development such as packet management tools npm and yarn (MDM Web Docs N.d.). These tools are used to initialize projects, install new software packages to the software, update, manage and delete packages if necessary.



```
$ npm install --save-dev jest

added 288 packages, removed 3 packages, changed 66 packages, and audited 1709 packages in 28s
```

Figure 1 An example of npm script to install Jest testing library to the project

Package managers do not do anything else for the project than install packages to of usable tools to the respective project package folders and in React the references to the node module packages can be found under package.json and package-lock.json folders. The former contains the projects package dependencies, package versioning and enables your projects build to be reproducible. The latter is a middleware responsible for validating and managing the versioning and maintaining the node module tree in which the source code for the packages exist.



```
{
  "name": "shopping-cart-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.1",
    "@testing-library/react": "^12.1.2",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^5.1.3",
    "react": "^17.0.2",
    "react-bootstrap": "^2.2.3",
    "react-dom": "^17.0.2",
    "react-redux": "^7.2.6",
    "react-scripts": "5.0.0",
    "web-vitals": "^2.1.4"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
```

Figure 2 package.json file within the root of the project

The update process for packages is maintained handily by first checking if there are anything to update with script *npm outdated* and then running another script *npm update* to finally update the packages trough the NPM package manager. An alternative and more advanced solution would be to install tools like npm-check-updates which after hoisting the package as a global variable with *npm install -g npm-check-updates*, it allows the execution of custom upgrading with *ncu* and then either *ncu –upgrade* or with shortened flag simply *ncu -u*.

According to a poll of 1502 respondents, in modern web application development roughly 77% of correspondent developers in various work positions reported their companies to be using microservices (Loukides & Swoyer 2020). More than third of the respondents reported to report their organizations to have been using microservices between 1 and 3 years as of 2022.

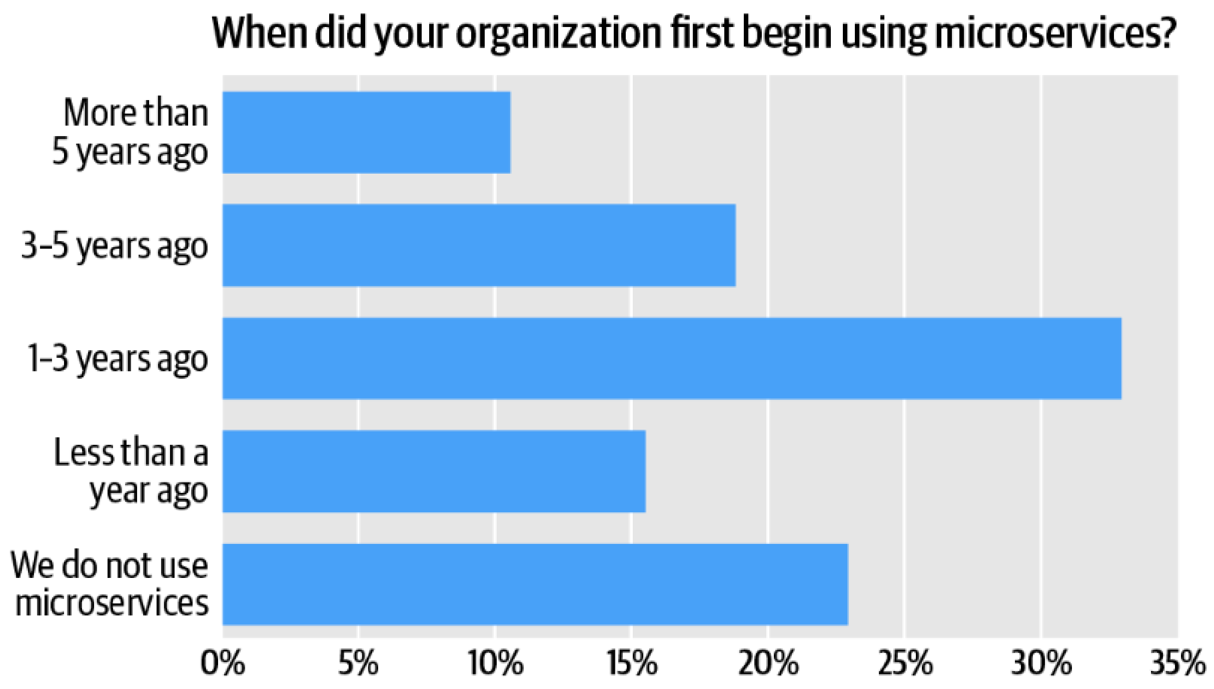


Figure 3 Microservice usage adaptation by 2020

Microservices architecture is the concept of providing an app, service or other where a single application is composed of independently deployable and loosely coupled smaller components or services.

They have their own independent technology stacks, can use their own programming languages, and communicate with each other over REST APIs, message brokers and event streaming. The benefits of microservices are that they are more flexible, allowing an easier way to update code or introduce new features to the service without having to touch the entire application (What are microservices? N.d.). Due to their loose coupling, they are more independently scalable with one another, and modularity allows for smaller chunks of handling data traffic, resource cost of scaling and reducing unnecessary clutter.

3.2.1 Modern technology stacks

As the count of web applications has increased by a multitude of factors over the years, many different kind of technology stacks, i.e., technology compositions or technology architectures have been invented to compete in the technology stack solution markets. When designers, architects or managers decide what kind of technology stacks they will opt in, they will weigh the different options and try to determine the best and most efficient technology solution for the job.

Some of the factors that need to be considered is the objectives and requirements of the application. In other words what are the use cases that the application aims to fulfill. A web application that needs to be light and nimble has a vastly different kind of use cases than an application that is not time sensitive. Another factor to consider is the applications cyber security level and trying to determine how much sensitive data the application will need to use and how to protect the data against attackers. If the application is very impersonal and will not need any data binding to known user, then it will not need to protect anything else than to prevent attackers installing malware to the application. However often times in modern web applications some kind of user authentication and data binding is required (Archison 2022).

Other factors are more about the logistical needs to the web application, such as how well can the applications architecture scale and what are the costs that need to be paid for the web application. The cost does not simply refer to monetary cost, although that is a part of it but also human resources in development time and maintenance, end-user-experience in the application workload and even the legality costs in case of a data-breach (What factors go into choosing a technology stack? N.d.). For example, a scalability cost to consider in modern web applications is the database solution.

Technology stack solutions examples that are still in popular wide use include LAMP, Django, RoR, Flutter and some variation of MEAN, MEVN, PERN or MERN. The oldest of them all is LAMP, and it has seen usage as one of the industry classics. It’s open-source and free availability, as well as being able to work on any operating system makes it an attractive technology stack for any web application that needs highly optimized, costly efficient and flexible solution (Best Web Development Stacks to Use in 2023 N.d.). Django is a Python based web development framework for rapid development and clean code, and it has gained popularity among some web developers due to its flexibility and ease of use (Song 2022).

RoR or as elongated Ruby on Rails is an open source and dynamic programming language, which is efficient for lightweight applications due to its flexibility. It can utilize HTML, CSS, JavaScript for UI design and XML or JSON for data transferring. Flutter has been described as revolutionary web stack for cross-platform development industry, which allows easier development for multitude of different types of platforms from desktop applications like for Windows and Mac OS, to mobile applications like Android and iOS (Best Web Development Stack 2023 2022).

PERN, MEVN, MEAN and MERN all belong to a similar technology stack solution and by far share the most commonalities between each other. PERN differs from others by using PostgreSQL as the database solution, instead of MongoDB. PostgreSQL is stricter when it comes to rules for data integrity and it utilizes the ACID principle in data validation, which allows PostgreSQL to offer fail-states and more reliable storage for data. Companies that manage sensitive data like banks, law enforcers or healthcare tend to favor it over MongoDB. On the other end of the spectrum are MEVN, MEAN and MERN, where V represents VueJS, A is for Angular, and R is for React. All of them are simply different JavaScript based frontend frameworks. VueJS is a lightweight alternative to Angular and is prized for its simplicity and adaptability. Angular is its heavier cousin that is focused on all-in-one framework solutions that is easier to test. React is arguably the most popular of them all due to its high customizability and cost-effectiveness (Best Web Development Stacks to Use in 2023 N.d.).

3.2.2 Architecture of a ReactJS web application

Javascript has been dominating web development for a couple of decades ever since the turn of the 2000's when big companies like Facebook and Google started using it in their projects due to the capabilities and functionalities of the programming language (Codeacademy 2022).

Those development needs of the early 2000's have sprung up a host of technology solutions to web development raging from libraries and frameworks like ReactJS, TypeScript and Angular, to data management tools such as React Redux, security tools like Babel or testing libraries such as Jest.

Because technology solutions like ReactJS are based on Javascript, it inherits many of the main principles of Javascript scripting language. Among the inherit attributes is the DOM principle, which builds the application in a hierarchy tree principle. In practice this means that any element that is rendered on a higher level in the DOM hierarchy is a parent element and every component that is rendered within the parent component is called a child component.

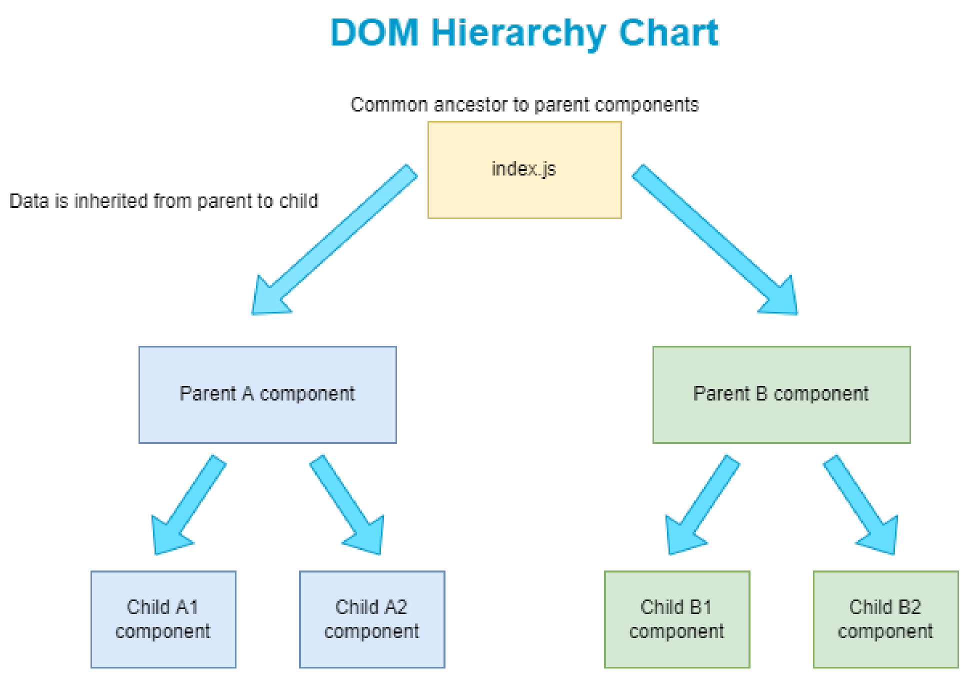


Figure 4 DOM hierarchy chart

In practice this means that the flow of data by default is one directional. The child inherits data from the parent and if there is a need for child to share data with another child, the data state must be set within a common ancestor. There are ways to send data from child to parent trough props, but a more convenient solution is to use React Redux technology.

React Redux is a technology solution that provides an easy to understand, easy to manage predictable state container for Javascript app solutions. Rather than doing callbacks and sending props manually, React Redux automates the process by sending the state of data to a common ancestor known as store, and then fetching the state of the data by using a useSelector. This allows the application to share data between components trough the common ancestor and able to overcome the manual labor required without such technology solutions.

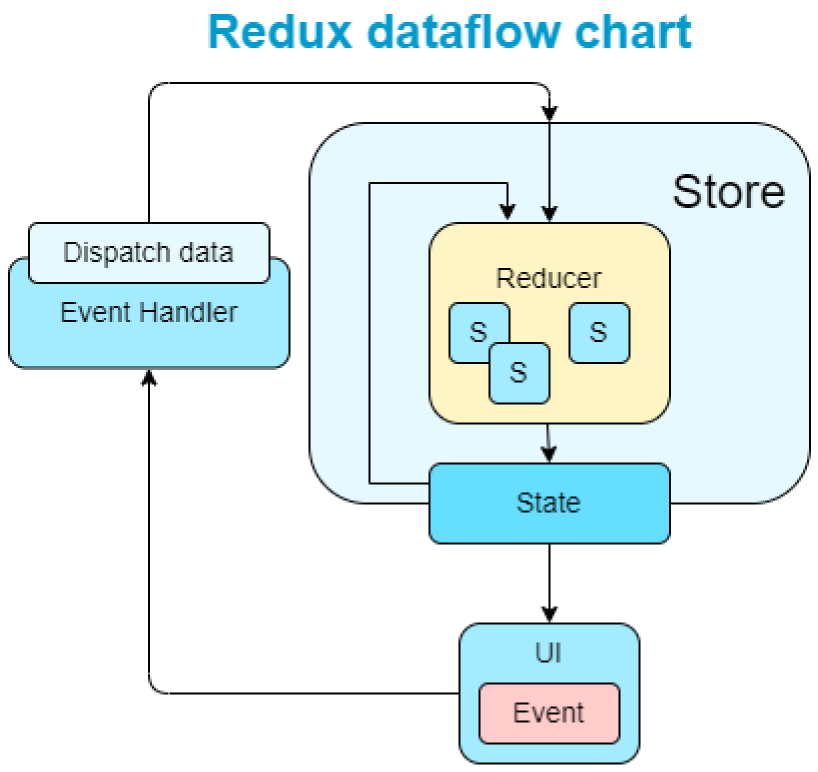


Figure 5 Dataflow example in React Redux

However, it is recommended by good coding practices to separate different kinds of functions, purposes, and roles of file types into their own separate folders (Chien. N.d.). So, for example a database schema is not at the same folder as the router files are located at.

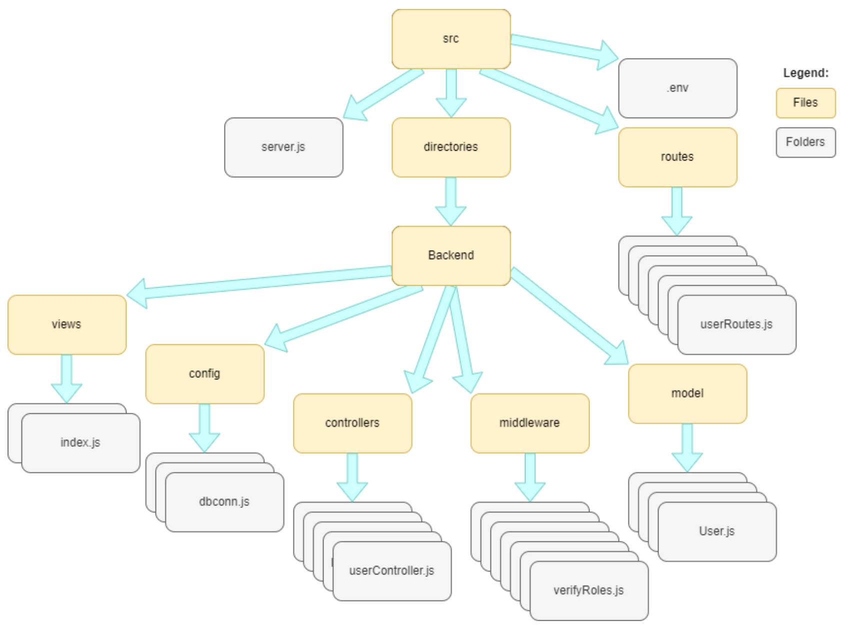


Figure 6 Backend architecture

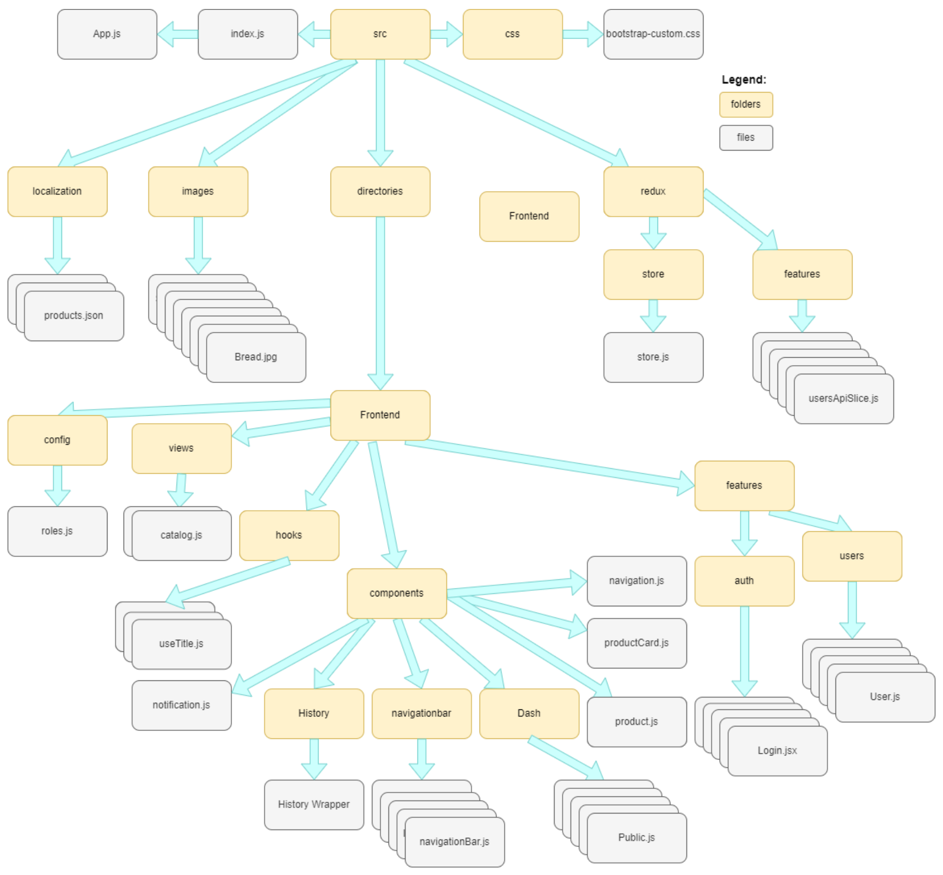


Figure 7 Frontend architecture

Once a rough sketch of file structure has been designed, it is time to create a mock-up of the UI/UX for the web application. This gives some sort of base concept as to how the web application should look like, what features it should have and it helps to orient the development process which is especially useful in the initial development phase when there is still no code that has been written.

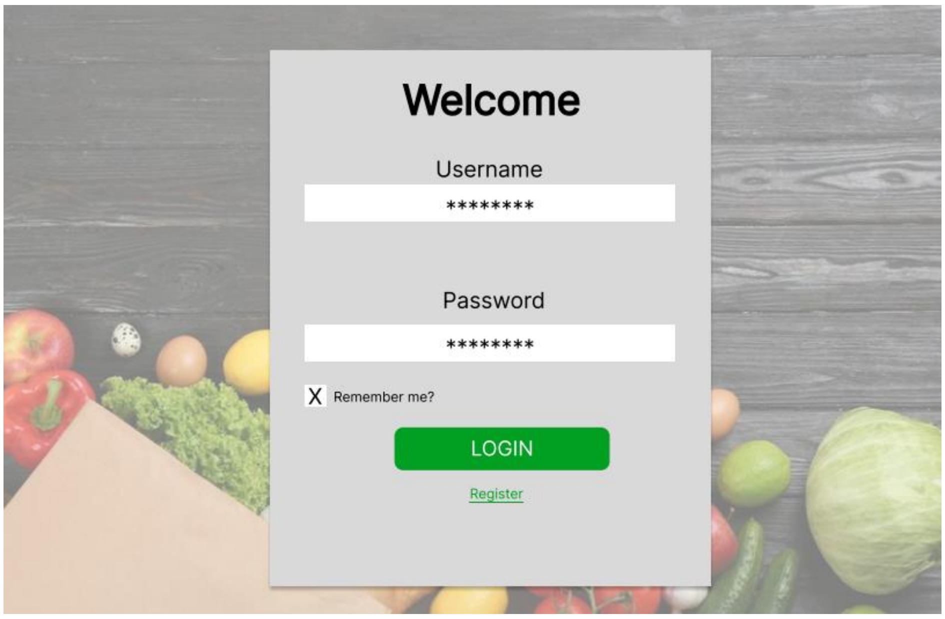


Figure 8 Login mock-up

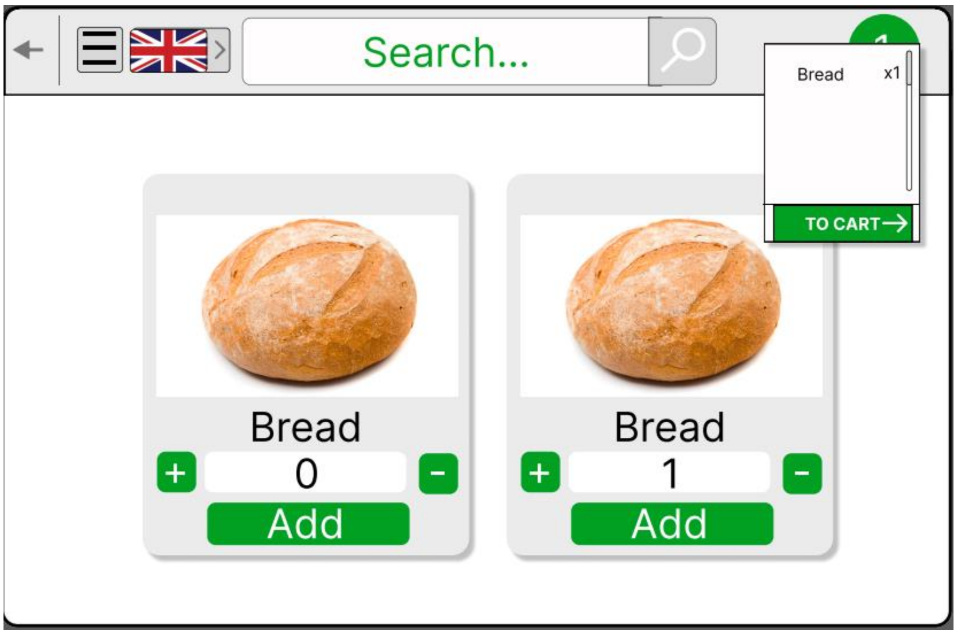
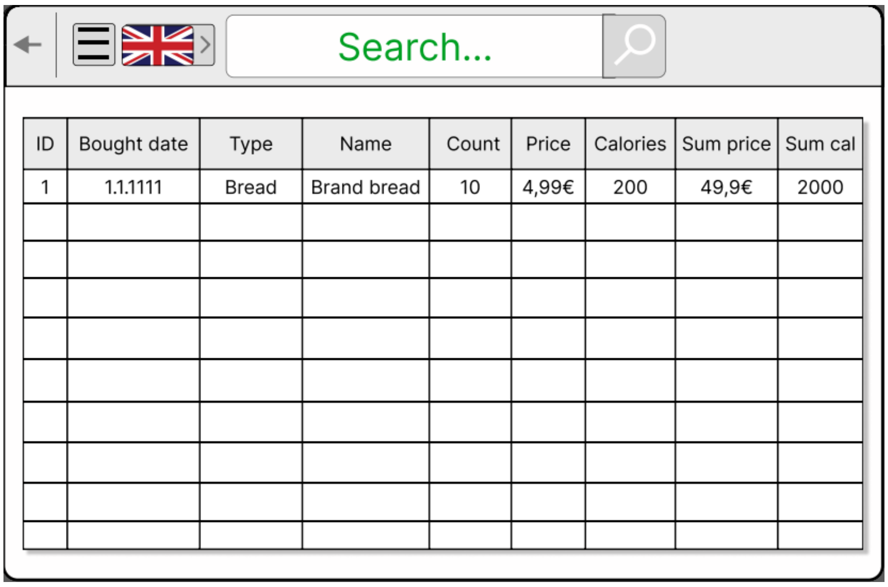


Figure 9 Shopping catalog mock-up



ID	Bought date	Type	Name	Count	Price	Calories	Sum price	Sum cal
1	1.1.1111	Bread	Brand bread	10	4,99€	200	49,9€	2000

Figure 10 Purchase history mock-up

3.2.3 MERN-Stack

MERN is an acronym from MongoDB, Express, React and NodeJS technology stack. It is a techno-logical option of a stack for a full stack application development and one of the more commonly seen stacks in modern day web application development. The base concept is simple, MongoDB works as an alternate to other database options and as a NoSQL database, it does not inherit most of the design principles behind SQL databases.

MongoDB is a non-relational database which means it does not create relationships between dif-ferent tables within the database. Instead, it utilizes so called documents, which are similar in for-mat to json-filetype format. Everything in data is stored into data objects and these objects are then polled from -, to - or updated based on the queries that the server sends on user input inter-action from the front end.

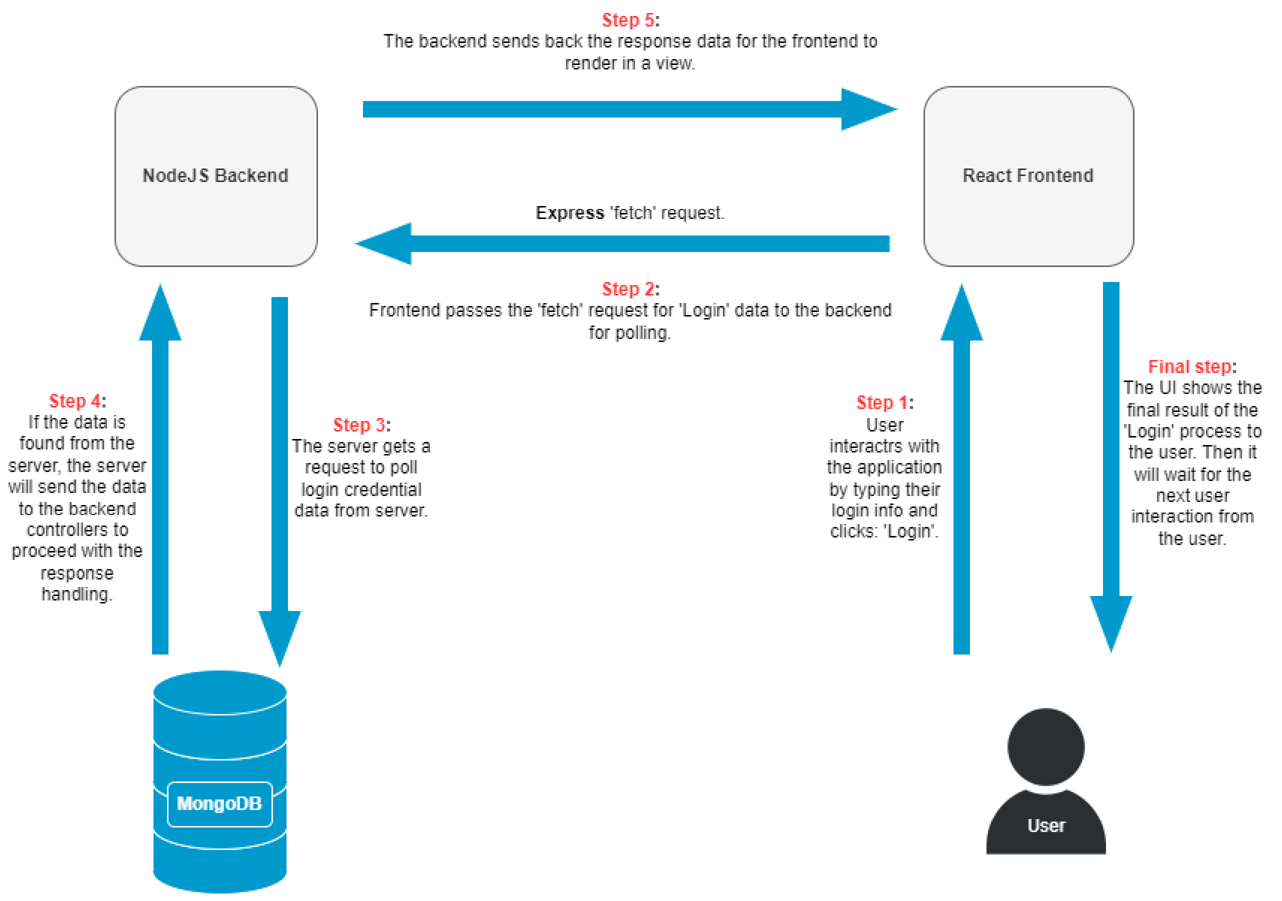

Step 5:
The backend sends back the response data for the frontend to render in a view.
NodeJS Backend
React Frontend
Express 'fetch' request.
Step 2:
Frontend passes the 'fetch' request for 'Login' data to the backend for polling.
Step 4:
If the data is found from the server, the server will send the data to the backend controllers to proceed with the response handling.
Step 3:
The server gets a request to poll login credential data from server.
Step 1:
User interactrs with the application by typing their login info and clicks: 'Login'.
Final step:
The UI shows the final result of the 'Login' process to the user. Then it will wait for the next user interaction from the user.
MongoDB
User

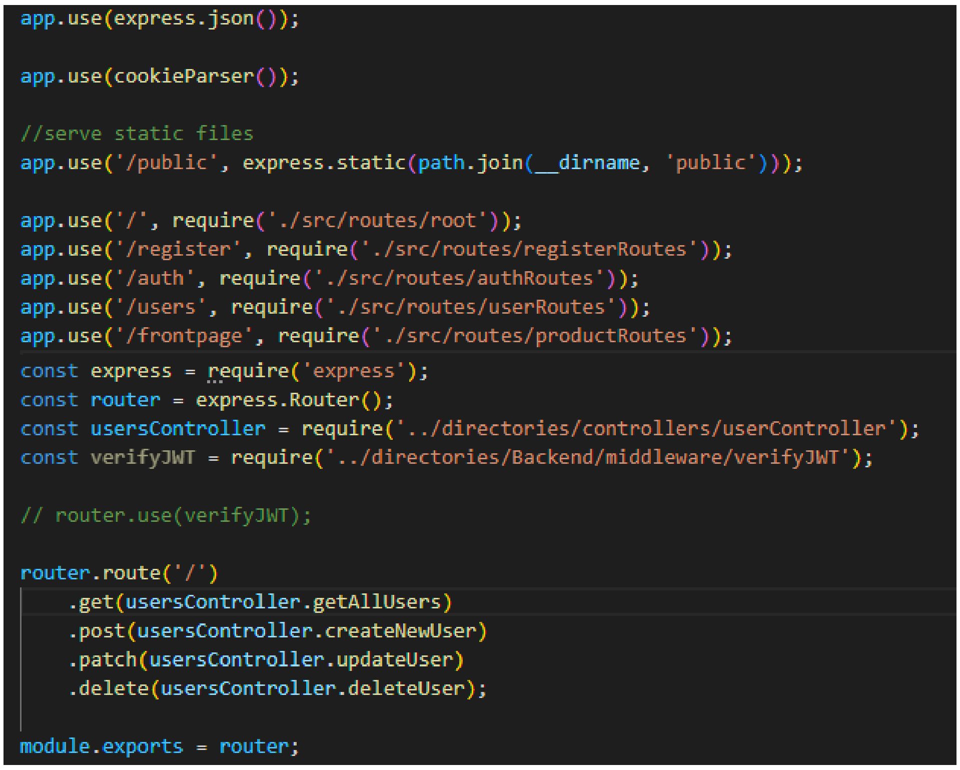
Figure 11 Sample of 'Login' process in the MERN stack

The benefit of MonboDB over other SQL and NoSQL database solutions its flexibility and scalability (What is MonboDB? N.d.). In other words, it is advertised to being good for larger enterprise sized datasets that are easy to replicate, apply load balancing to network traffic communication and data aggregation into clusters of data.

Compared to other database solutions, the disadvantages of a MongoDB database solution are problems with continuity due to its automated failover strategy whereby each node only have one master node and upon failure the MongoDB will default to converting other node into a new mas-ter. This limits the availability of database infrastructures. Other technological limitations are its write limits due to the beforementioned one master node architecture, data consistency issues due to not supporting full referential integrity using foreign-key constraints, and possible issues with database security as user authentication is not enabled by default in databases (Gillis March 2023).

Another disadvantage of MongoDB technology solution may be that it is difficult to do database joining if the use case requires some type of relationship. According to the MongoDB database, the designer should really think about the type of cardinality the database topology has (Zola 2014).

Express is another technology belonging to the MERN-stack technology solution package. NodeJS is a runtime environment designed to build server-side applications but what it will not provide is how to serve files, handling HTTP requests and methods. Express offers a solution and forms one of the main components of a MERN-stack. It is a NodeJS web application framework that will provide a broad selection of features for web application development (Express/Node introduction N.d.).



```
app.use(express.json());

app.use(cookieParser());

//serve static files
app.use('/public', express.static(path.join(__dirname, 'public')));

app.use('/', require('./src/routes/root'));
app.use('/register', require('./src/routes/registerRoutes'));
app.use('/auth', require('./src/routes/authRoutes'));
app.use('/users', require('./src/routes/userRoutes'));
app.use('/frontpage', require('./src/routes/productRoutes'));
const express = require('express');
const router = express.Router();
const usersController = require('../directories/controllers/userController');
const verifyJWT = require('../directories/Backend/middleware/verifyJWT');

// router.use(verifyJWT);

router.route('/')
    .get(usersController.getAllUsers)
    .post(usersController.createNewUser)
    .patch(usersController.updateUser)
    .delete(usersController.deleteUser);

module.exports = router;
```

Figure 12 Express routing for HTTP requests

Some of the advantage of Express over its other competitors is that it allows developers to structure their code freely and choose the sort of structure their code uses. Due to its structured nature, the disadvantages can be related to it if the data cannot be organized and structured in a way the server expects and the code can become difficult to understand. Sometimes this calls for denormalization of the code (Cosset 2017).

3.3 Web development in the industry

ICT-industry is quickly evolving industry that requires developers to adapt along with it and learn new technology solutions at a frequent phase. This is partly due to the exponential growth of the popularity and digitalization of the world overall. According to a new survey released by Netcraft on 28.2.2023, the amount of website has grown from just 32 594 in October 1995 to 202 009 862 in February 2023, an increase of 61 877,76% rounded increase in active host domain names in a little over of 28-years timespan (February 2023 Web Server Survey 2023).

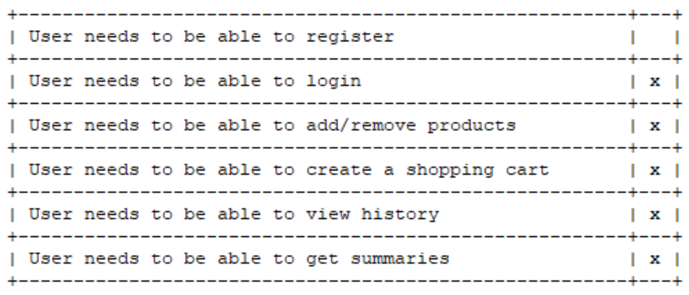
In a survey conducted by Tiago Bianchi in 30.1.2023 surveying the share of global mobile website traffic between 2015-2022 found that nearly 59,12% of all web application traffic came from mobile devices in the last quarter of 2022 (Bianchi 2023). The increase in PWA architecture web application has increased the demands for a more universally available web application technology solutions that allow the web applications to be accessed no matter what the device is and increasingly many sites incorporate P2P synchronization into their web applications to allow the users a more accessible user experience.

Another way modern web development trends have shifted is their shift from server-required architectural solutions to serverless solutions. Rather than the company or product owner hosting up the web application from their own servers, they buy a third-party hosting service to host their applications there. Some of the common competitors of these kinds of services are AWS Lambda, Google Cloud Functions or Microsoft Azure Functions (Makhov 2021). Many of them employ Cloud solutions and architecturally they either typically represent FaaS or BaaS solutions (Clark N.d.), but of course they can also take other forms like IaaS, PaaS, or SaaS respectively.

4 Developing the web application

4.1 Software development phases

When designing a web application, there are several different doctrines and principles that the developer can follow. In most of the models it is important to firstly decide what are the user stories that the developer hopes to serve. This helps to orient questions such as what the app is about and to whom is it for, and what are the goals of the app (Johnston 2019).



```
+--------------------------------------------------------+---+
| User needs to be able to register                      |   |
+--------------------------------------------------------+---+
| User needs to be able to login                         | x |
+--------------------------------------------------------+---+
| User needs to be able to add/remove products           | x |
+--------------------------------------------------------+---+
| User needs to be able to create a shopping cart        | x |
+--------------------------------------------------------+---+
| User needs to be able to view history                  | x |
+--------------------------------------------------------+---+
| User needs to be able to get summaries                 | x |
+--------------------------------------------------------+---+
```

Figure 13 User stories sample

After the user stories are defined, the developer should start thinking about the web applications topology and architecture. Since it was already covered earlier, the developed web application will be utilizing a MERN stack as the technology solution and it will be multipage solution. This only solves what technology and service solutions will be used but it does not have an input on the web application structure. There is no standardization on how a web application file structure should be organized but there are some general principles and best practice guidelines that are good to follow depending on the chosen programming language (How To Structure React Projects From Beginner to Advanced 2022).

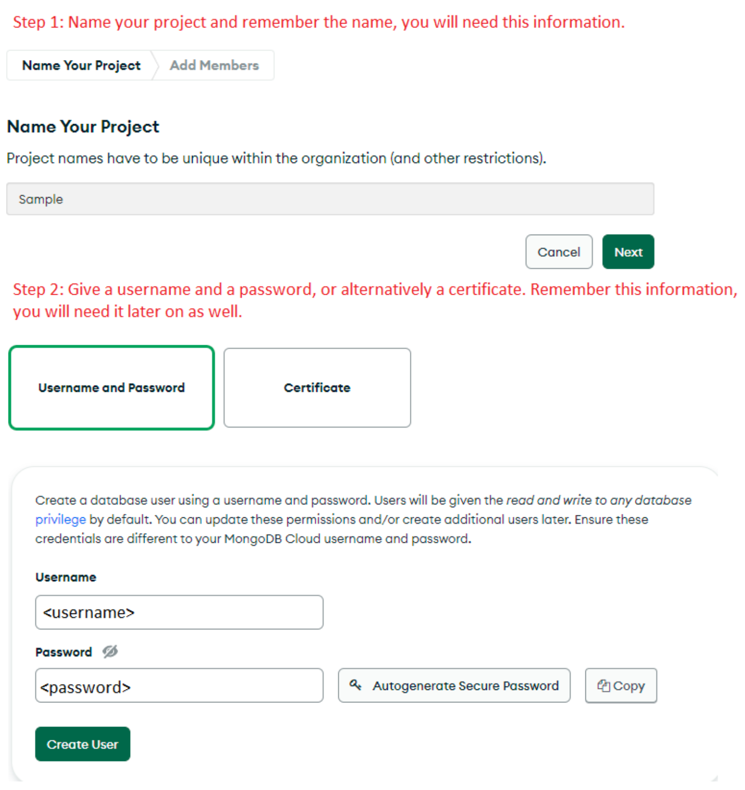
4.2 Setting up the development environment

Since the web application is known to utilize a MERN stack, which requires a full stack solution to be developed, there needs to be a backend and a frontend. This is because for a user to be able to view a purchase history, the data must be bound to the user because without this binding, how could the application tell which purchase history event belongs to which account. It is therefore necessary for the web application to have at least some sort of user and then a cart functionality.

When a frontend makes callbacks to the server for polling, the server must have an endpoint which does the call back handling, polling to server and returning the data state for the frontend to render (Figure 15).

There needs to be a server that builds up and maintains the server and an environment file which contains the . There needs to be a router which can tell the application how to route the network traffic appropriately on callbacks to the server and finally there needs to be the controllers themselves which hold the functionality and business logic of the backend. The rest is up to the developer to choose how to organize and structure their backend logic.

Next the web application will need a database and since the project uses MERN stack, it'll be a MonboDB database solution. When creating a MongoDB solution, the only relevant information you will need from the steps to setup your database environment is the name of the project, a mandatory username and a password or other credentials that are used to identify the connecting account.



Step 1: Name your project and remember the name, you will need this information.

Name Your Project | Add Members

Name Your Project

Project names have to be unique within the organization (and other restrictions).

Sample

Cancel | Next

Step 2: Give a username and a password, or alternatively a certificate. Remember this information, you will need it later on as well.

Username and Password | Certificate

Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username

<username>

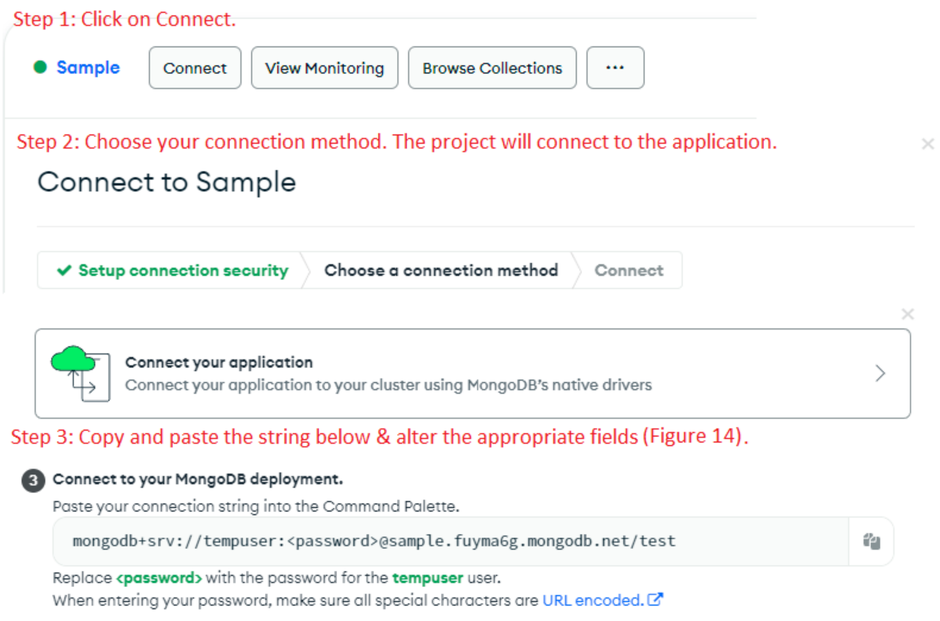
Password

<password>

Autogenerate Secure Password | Copy

Create User

Figure 14 Creating a MongoDB database


Step 1: Click on Connect.

Step 2: Choose your connection method. The project will connect to the application.

Connect to Sample

Connect your application

Connect your application to your cluster using MongoDB's native drivers

Step 3: Copy and paste the string below & alter the appropriate fields (Figure 14).

3 Connect to your MongoDB deployment.

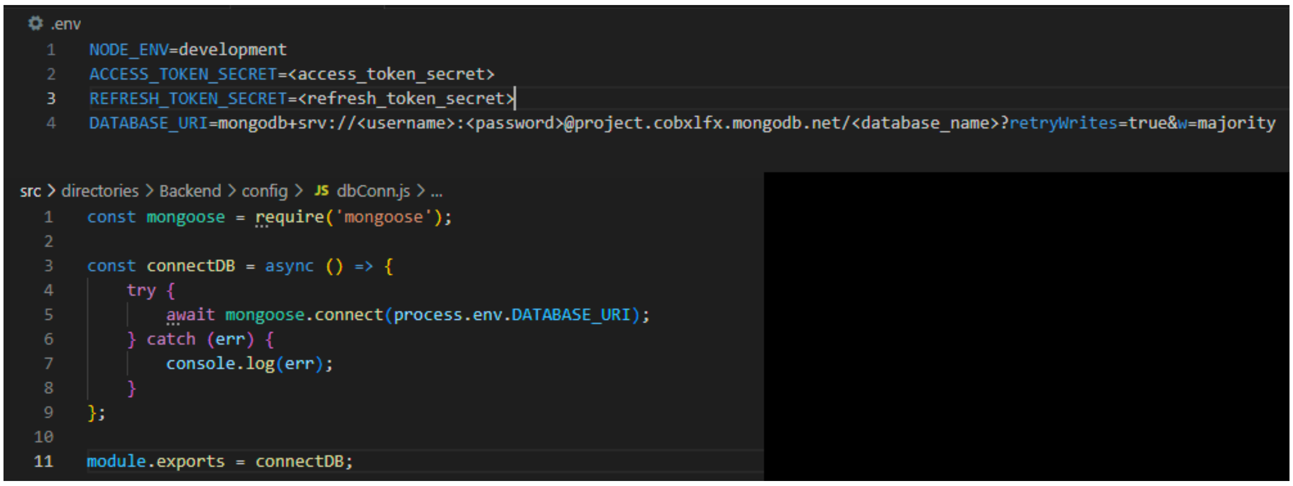
Paste your connection string into the Command Palette.

```
mongodb+srv://tempuser:<password>@sample.fuyma6g.mongodb.net/test
```

Replace <password> with the password for the tempuser user.
When entering your password, make sure all special characters are URL encoded.

Figure 15 Guide on MongoDB connection

After that the last few steps to setting up a development environment is to create a file which gives the database connector resources. It is the .env file and dbConn.js files in figure 9. To generate the token secrets, you need to open a terminal, write *node*, press enter and then enter the following script *require('crypto').randomBytes(64).toString('hex')* and run that script twice. Copy and paste the generated tokens into your .env file.


.env

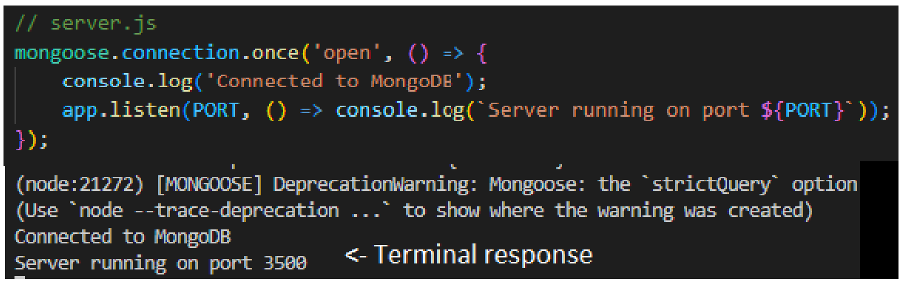
```
NODE_ENV=development
ACCESS_TOKEN_SECRET=<access_token_secret>
REFRESH_TOKEN_SECRET=<refresh_token_secret>
DATABASE_URI=mongodb+srv://<username>:<password>@project.cobxlfx.mongodb.net/<database_name>?retryWrites=true&w=majority
```

src > directories > Backend > config > JS dbConn.js > ...

```
const mongoose = require('mongoose');

const connectDB = async () => {
    try {
        await mongoose.connect(process.env.DATABASE_URI);
    } catch (err) {
        console.log(err);
    }
};

module.exports = connectDB;
```

Figure 16 .env and dbConn.js files

All there is left to do is to install appropriate packages by running the script *npm i* and creating a server file (Figure 20) and finally running the script *npm run dev*. If the server and connection was successful, the terminal should tell this.



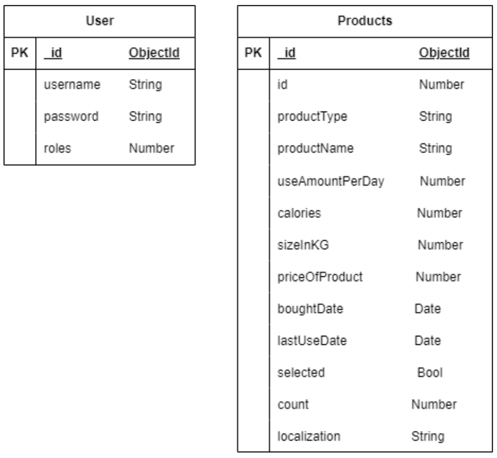
```
// server.js
mongoose.connection.once('open', () => {
    console.log('Connected to MongoDB');
    app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
});
```

```
(node:21272) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option
(Use `node --trace-deprecation ...` to show where the warning was created)
Connected to MongoDB
Server running on port 3500
```

Figure 17 Development environment deployed

4.3 First iteration

In the original concept the web application was supposed to simulate the backend using NoSQL solution and instead relying on json files to serve the data to the frontend. However early on during the development the need for a functional backend architecture became relevant due to needing a way to bind the user data with the shopping cart containing a list of product references. Otherwise, when viewing the purchase history component, how could the application tell whose purchase history should be rendered within the data grid. Another note on the database solution was that because the application is not meant to handle purchasing products, it will not need transaction data anywhere. All it needs is data of the products and simulated estimation of costs.



User		
PK	_id	ObjectId
	username	String
	password	String
	roles	Number

Products		
PK	_id	ObjectId
	id	Number
	productType	String
	productName	String
	useAmountPerDay	Number
	calories	Number
	sizeInKG	Number
	priceOfProduct	Number
	boughtDate	Date
	lastUseDate	Date
	selected	Bool
	count	Number
	localization	String

Figure 18 MongoDB original topology

The requirements for the backend were that it needs to be able to process a CRUD operation for basic login and product functionality. The login needed to be secure enough for the purposes of the mock-up to simulate what software security should look like. This meant that the password needed to be encrypted with salting. In real world applications mere salting would not be sufficient but since this thesis is not a lookup into software security but rather creating a mock-up of a shopping cart web application, it should not

4.4 Development iteration & refactoring

It was clear a backend and a database were necessary, and the database topology needed a cardinality relationship between a user and a product, hence the cart collection. The concept behind it is that whenever a user is created, an empty cart is also created. The benefits of this solution over others are that it allows for effortless data binding of user data to carts and thus allowing purchase history creation without needing extensive backend for data validation. The downside of this solution is that it creates unnecessary clutter to the database because if there is a user account that is created but it is unused, it will also create an unused cart which creates empty data unnecessarily. However, it would also be unrecommended to simply cull data from the server after a period of inactivity and in modern web applications even inactive user data is stored for a prolonged period of time in case the user wants to return one day.

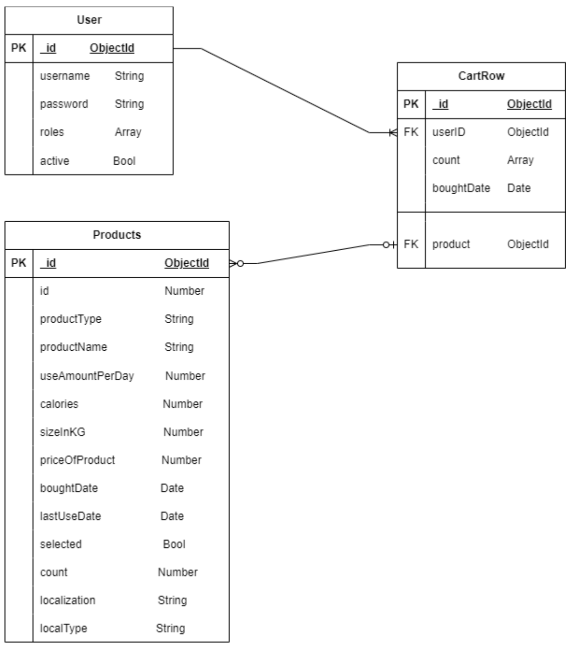
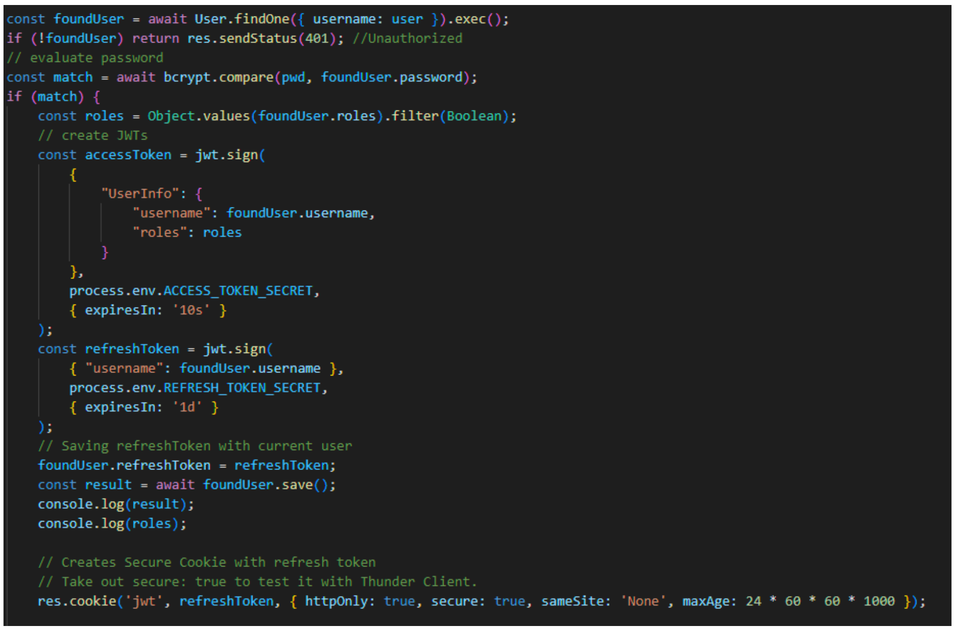


Figure 19 Updated MongoDB topology and cardinality

Another notable issue with the web application was that it had a lot of duplication in the codebase and while there are different schools of thought on the value of duplicate code, a consensus for duplication is that it is bad for your codebase. In other words, a high-quality code should try to rely on the DRY principle as much as possible and avoid unnecessary duplication (Silva. 2019). If the knowledge or logic looks like a duplicate and there is a way to avoid writing the same code to several locations at the same time, it is beneficial to simply extract the code outside from the function and simply do a callback when needed (Bongarts 2021).

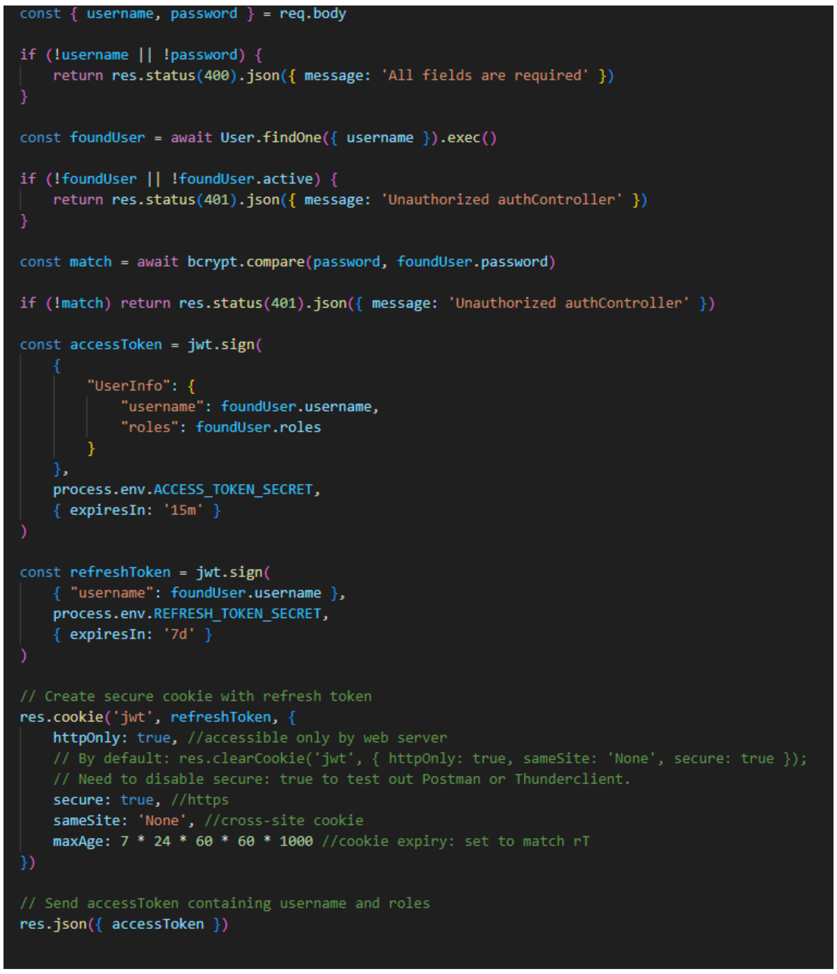
Having duplicate code makes the code lengthier which affects traceability and performance. It increases technical dept unnecessarily in cases where updates to the codebase are necessary and instead of making a fix to one centralized and distributed code block, the developer must make the same change to several different locations which makes it much harder -, slower - and resource costly to maintain. From a software security standpoint, it is also harmful because if the there is a new security vulnerability, fixing takes much longer and leaves the service vulnerable for exploitation for longer time periods than it needs to (Duplicate code: Everything you need to know N.d.).

Furthermore, there was a problem with data handling and lifespan throughout the web application. When the user logged in and switched the pages from one page to another, upon returning to the previous page, the data state was not stored and rather it was reset back to its default state. At worst this would mean the user also lost their login credentials data which caused the application to crash as it was unable to poll and get the login token back once more. To offset and work around this issue, the web application needed a persistent login logic which uses a refresh token that will persist over the session lifespan inside cookies, or it will expire and release resources if the user has been inactive on the web application for one week.



```
const foundUser = await User.findOne({ username: user }).exec();
if (!foundUser) return res.sendStatus(401); //Unauthorized
// evaluate password
const match = await bcrypt.compare(pwd, foundUser.password);
if (match) {
    const roles = Object.values(foundUser.roles).filter(Boolean);
    // create JWTs
    const accessToken = jwt.sign(
        {
            "UserInfo": {
                "username": foundUser.username,
                "roles": roles
            }
        },
        process.env.ACCESS_TOKEN_SECRET,
        { expiresIn: '10s' }
    );
    const refreshToken = jwt.sign(
        { "username": foundUser.username },
        process.env.REFRESH_TOKEN_SECRET,
        { expiresIn: '1d' }
    );
    // Saving refreshToken with current user
    foundUser.refreshToken = refreshToken;
    const result = await foundUser.save();
    console.log(result);
    console.log(roles);

    // Creates Secure Cookie with refresh token
    // Take out secure: true to test it with Thunder Client.
    res.cookie('jwt', refreshToken, { httpOnly: true, secure: true, sameSite: 'None', maxAge: 24 * 60 * 60 * 1000 });
```

Figure 20 Old authentication controller



```
const { username, password } = req.body

if (!username || !password) {
    return res.status(400).json({ message: 'All fields are required' })
}

const foundUser = await User.findOne({ username }).exec()

if (!foundUser || !foundUser.active) {
    return res.status(401).json({ message: 'Unauthorized authController' })
}

const match = await bcrypt.compare(password, foundUser.password)

if (!match) return res.status(401).json({ message: 'Unauthorized authController' })

const accessToken = jwt.sign(
    {
        "UserInfo": {
            "username": foundUser.username,
            "roles": foundUser.roles
        }
    },
    process.env.ACCESS_TOKEN_SECRET,
    { expiresIn: '15m' }
)

const refreshToken = jwt.sign(
    { "username": foundUser.username },
    process.env.REFRESH_TOKEN_SECRET,
    { expiresIn: '7d' }
)

// Create secure cookie with refresh token
res.cookie('jwt', refreshToken, {
    httpOnly: true, //accessible only by web server
    // By default: res.clearCookie('jwt', { httpOnly: true, sameSite: 'None', secure: true });
    // Need to disable secure: true to test out Postman or Thunderclient.
    secure: true, //https
    sameSite: 'None', //cross-site cookie
    maxAge: 7 * 24 * 60 * 60 * 1000 //cookie expiry: set to match rT
})

// Send accessToken containing username and roles
res.json({ accessToken })
```

Figure 21 New authentication controller

5 Results

5.1 Engineering method

This chapter is an analysis on the end-product of the thesis project to try to determine how well the project was able to fulfill modern web application requirements both in the codebase requirements and in the application UI standards and good practices. This chapter will answer questions like what is good UI/UX design? What considerations does the developer need to account for? What are some of the trends in modern web application best practices within the industry? How well has the thesis project been able to emulate and follow these guidelines?

Many modern UI/UX designs prefer a minimalist approach when it comes to the UI design philosophy and principles (Fleck 2021). This is partially due to usability reasons, as it is easier to navigate a simple and clear UI over complex and inconsistent UI's. Brand consistency, proper choice of color palette, spacing, typography and consistent actions and design are all some of the key aspects behind a successful web application over an unsuccessful one (Design Consistency Guide UI and UX Best Practices 2023). Utilizing proper fonts and having images consistently but neither too many images nor too much text to create information overload for cognition is advisable (Prokhorova 2022). In modern web applications it is a consensus among developers that simple UI is better than complex UI and the applications should be as fluid and intuitive to use as possible (What are the 10 Rules of Good UI Design? What is Good UI/UX Design? 2020).

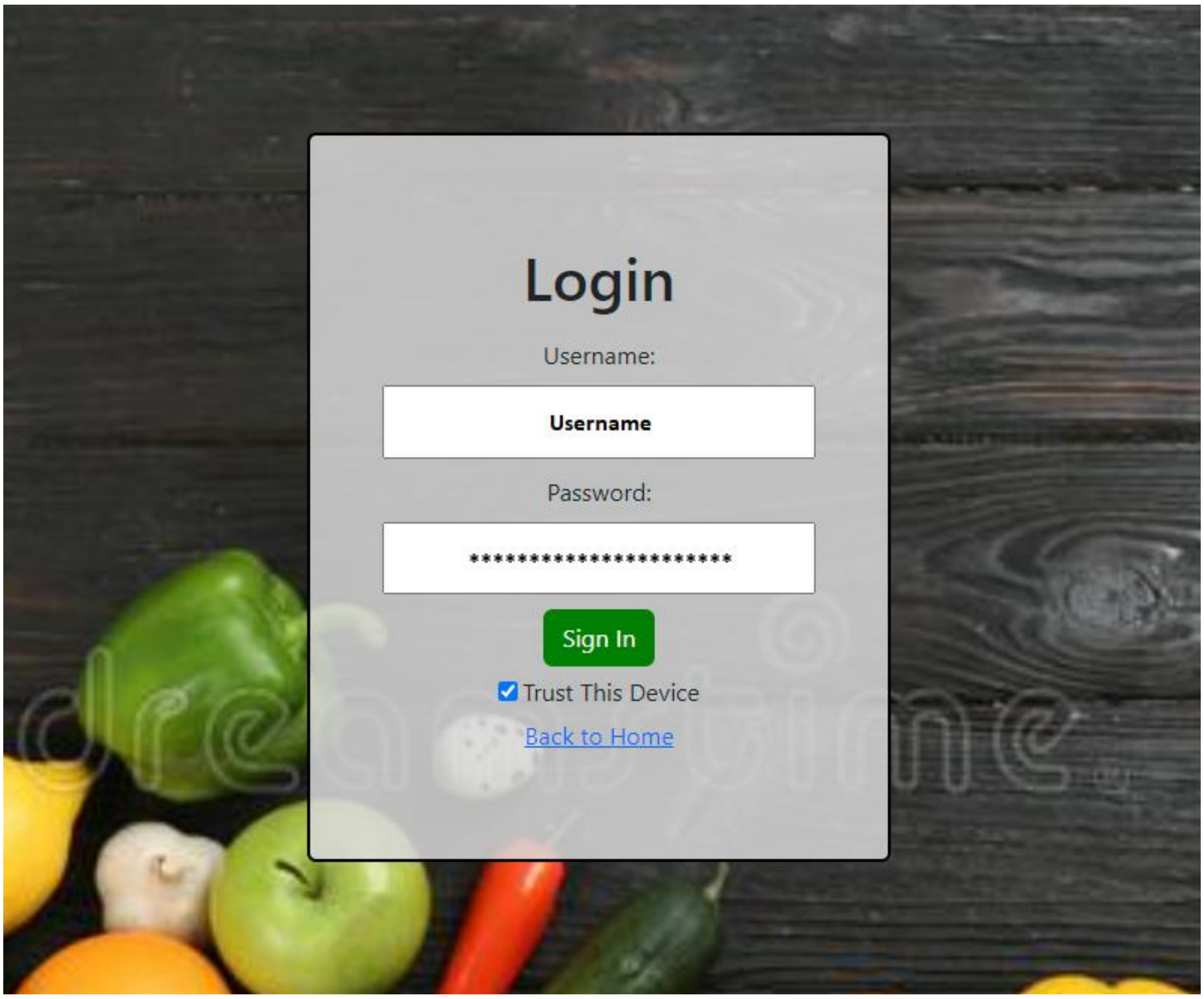


Figure 22 Login page of final version

First is the login page. The design principle of the login page was to create a simple form login page that takes a username and a password. The login feature was necessary for the project because without it, how the application could tell which user owns what shopping-cart items and whose history the application is supposed to show? The form itself is supposed to be simple, although it lacks some basic functionalities such as a register form.

At the release state, an admin must manually use Postman, ThunderClient or other tools to manually create a user to the database to be able to login. In a real-world industrial application, a register form would be some of the first features to create but since this thesis is a proof-of-concept project covering wide array of more generalized topics of interest, many basic features have not been fully implemented. Regardless the UI/UX design follows good practices and principles of modern web application style: Simple and clear.

While the user-to-cart binding was ultimately not fully implemented due to time constraints limiting proper testing, but the basic functionality is present and sufficient to give an example of how it can be implemented further into the application. The login feature itself is fully secure, it uses salting and SHA-256 decoding and encoding, then stores the jwt-token key to the cookies and carries the userdata this way over the application.

The main catalog page featuring all the different shopping items used mock data to simulate what it could look like. Much like advised by the articles above, it follows the same design simple and clean minimalist design philosophy throughout the applications lifecycle. The biggest critiques towards the projects UI/UX design is its overt simplicity such that in most modern web applications and even in this papers later chapter, the original and usual design is to have a separate cart-list that stores a list of objects into it and only makes POST requests to the server once user finally goes to buy the products. In the current project iteration, a POST request callback is fired whenever a user manipulates the virtual cart row data in any way. While this is not the end of the world, it is generally better to create an actual cart rather than a virtual one.

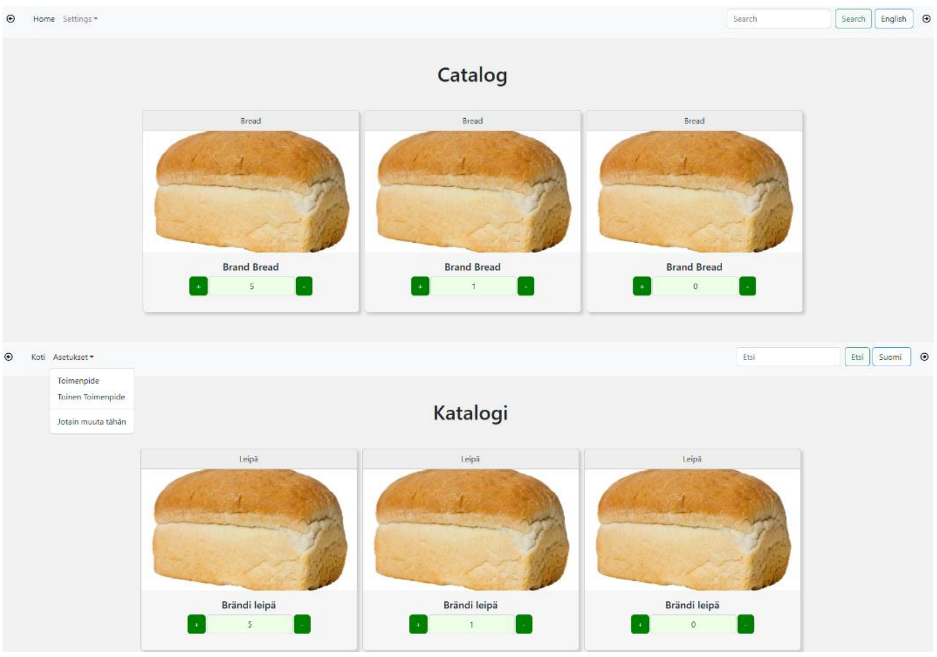
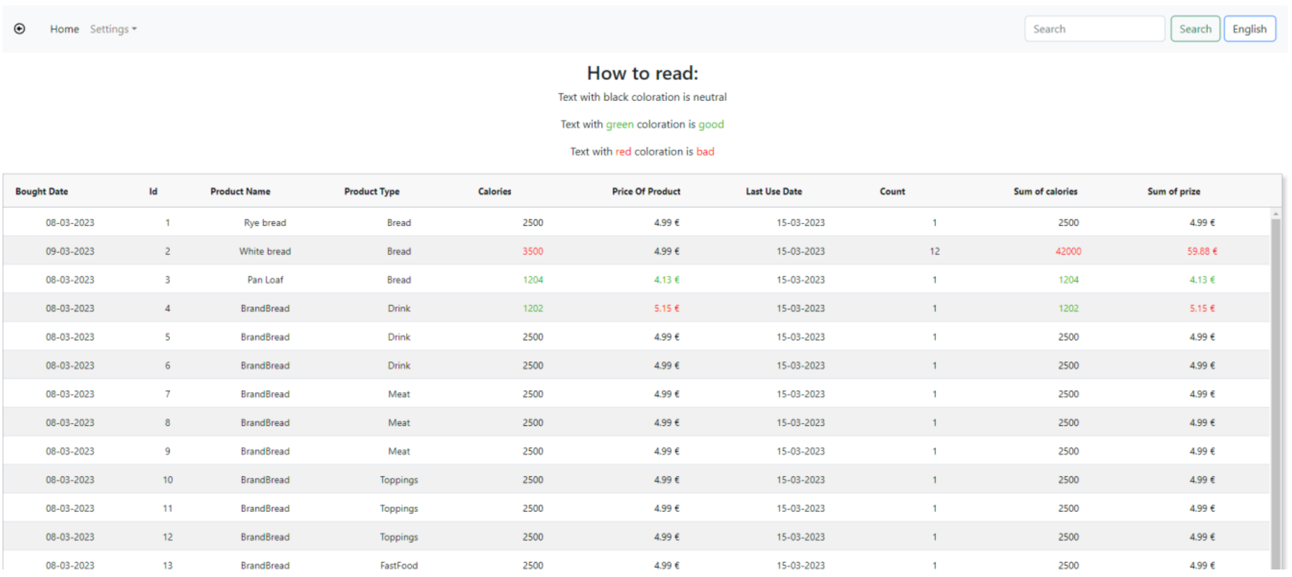


Figure 23 Sample of the catalog page

As seen from the figure 2. the application has some basic functionalities implemented while more features have been thought of but not yet fully implemented. The user can successfully create a shopping cart row for their items and the session will remember the state for a while even if the user logs-out accidentally or updates the page. Furthermore, the catalog page fully supports language translation feature by using i18l-language translation context library technology, which is important in any modern web application given we live in an increasingly connected, multifaceted and international world. In the thesis project sample, the project only supports English and Finnish translations, however, more can be easily implemented if necessary. Especially Arabic and Asian language selections are important due to their much more unique linguistic syntaxes. The author is unsure if the color choices are the best because the field where the user can manually write a number is light-green and the application has not accounted for color-blindness as a feature, even though from a usability perspective that is also another feature that should be implemented eventually.



How to read:

Text with black coloration is neutral

Text with green coloration is good

Text with red coloration is bad

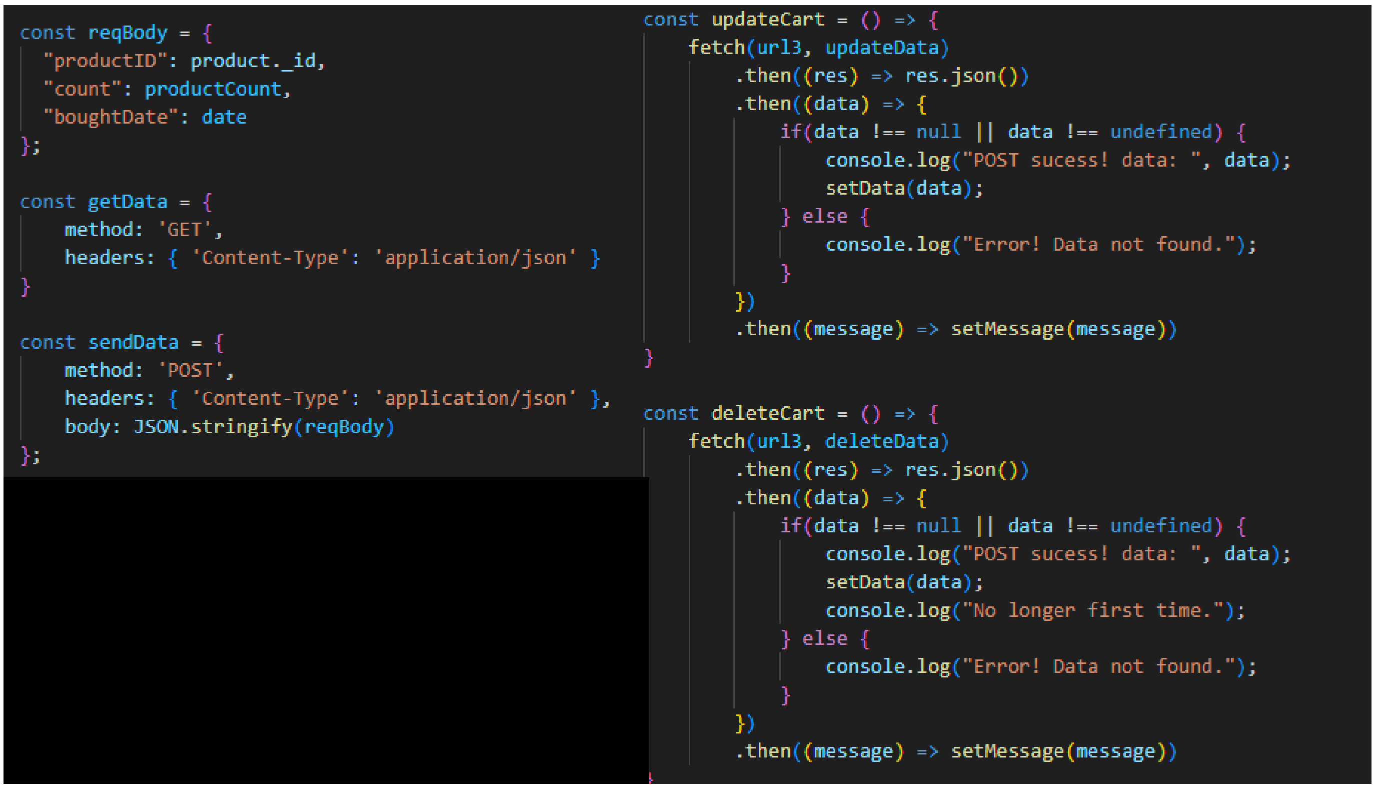
Bought Date	Id	Product Name	Product Type	Calories	Price Of Product	Last Use Date	Count	Sum of calories	Sum of prize
08-03-2023	1	Rye bread	Bread	2500	4.99 €	15-03-2023	1	2500	4.99 €
09-03-2023	2	White bread	Bread	3500	4.99 €	15-03-2023	12	42000	59.88 €
08-03-2023	3	Pan Loaf	Bread	1204	4.13 €	15-03-2023	1	1204	4.13 €
08-03-2023	4	BrandBread	Drink	1202	5.15 €	15-03-2023	1	1202	5.15 €
08-03-2023	5	BrandBread	Drink	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	6	BrandBread	Drink	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	7	BrandBread	Meat	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	8	BrandBread	Meat	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	9	BrandBread	Meat	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	10	BrandBread	Toppings	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	11	BrandBread	Toppings	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	12	BrandBread	Toppings	2500	4.99 €	15-03-2023	1	2500	4.99 €
08-03-2023	13	BrandBread	FastFood	2500	4.99 €	15-03-2023	1	2500	4.99 €

Figure 24 History page sample

Finally, the history page. The history page is a pooling of all the data stored within the application except the unnecessary data for the user such as localization indicators have obviously been filtered out from the final datagrid while summary fields using calculations such as $f(x) = calories * count = x$ and $f(y) = price * count$ have been added there to fulfill the use case needs for having a way to quickly determine when the user has either eaten too much or bought items that cost too much for the budget. The grid itself uses ag-grid community version library as the datagrid implementation and uses custom rules for the formatting of the cells. Above the datagrid is a simple legend to tell the user how to read it. A better implementation of the legend would be to hide it inside a question mark tooltip which would show the information on hover, but it was not implemented in this iteration.

Again, the design philosophy follows the same minimalistic design that other components do, and shares many of the same problems as well. For one this is especially prevalent sample of the problem of using red and green to indicated good and bad values, as covered in the paragraphs above from a usability perspective the application needs an alternative theme for the people who suffer from monochromacy. Another issue with using the ag-grid library is that currently at the time of this thesis, it does not support i18l-language translation context library well, which means that the contents and column titles will not support language translation feature. Furthermore, the columns themselves are off set from the data, which is fixable but not implemented in this project.

So, there are some clear design flaws and implementation flaws in the final project but for the purposes and goals of the thesis, most of the major features have been fully implemented well enough. Of course, there are some logic issues as covered earlier and some repeating code in the backend as well, but on the other hand there is a case to be made that sometimes consistent coding style is better in terms of traceability than unnecessary complexity for the sake of avoiding repetition when unnecessary (Bongarts 2021).



```
const reqBody = {
  "productID": product._id,
  "count": productCount,
  "boughtDate": date
};

const getData = {
    method: 'GET',
    headers: { 'Content-Type': 'application/json' }
}

const sendData = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(reqBody)
};
```

```
const updateCart = () => {
    fetch(url3, updateData)
        .then((res) => res.json())
        .then((data) => {
            if(data !== null || data !== undefined) {
                console.log("POST sucess! data: ", data);
                setData(data);
            } else {
                console.log("Error! Data not found.");
            }
        })
        .then((message) => setMessage(message))
}

const deleteCart = () => {
    fetch(url3, deleteData)
        .then((res) => res.json())
        .then((data) => {
            if(data !== null || data !== undefined) {
                console.log("POST sucess! data: ", data);
                setData(data);
                console.log("No longer first time.");
            } else {
                console.log("Error! Data not found.");
            }
        })
        .then((message) => setMessage(message))
```

Figure 25 Sample of duplicate code

5.2 Case study

In the case study section, we will compare two other already produced shopping-cart web applications to the thesis web application. For the chosen metrics there will be eleven features of various priorities and use case priorities. The resulting table is a matrix that demonstrates how closely the thesis mock-up has been able to follow some of the common good practices for feature development in modern web applications. For the chosen web applications, the chosen options for Mealime and Our Groceries. Both are popular PWA web applications especially designed for mobile devices for their real time mobility and management allowing for wide accessibility. Each feature was randomly selected and will be given one point by default if it has that feature and zero if they do not have them. Then they will be multiplied by feature priority multiplier in descending order where P1 = * 0,5 and P5 = *0,1 multiplier.

Table 1 Case study matrix

Table 1. Case study matrix				
		Application		
Features	Priority (Px = *0,x^{-1})	Web app	Mealime	Our Groceries
Login	P2	0,4	0,4	0,4
Settings	P4	0,2	0,2	0,2
Logout	P2	0,4	0,2	0
Adding products	P1	0,5	0,5	0,5
Creating a shopping list	P1	0,5	0,5	0,5
Browsing the shopping list	P1	0,5	0,5	0
Purchase history	P3	0,3	0	0
Summary	P3	0,3	0,3	0
Diary	P3	0	0	0
Health supervision	P4	0	0,2	0
Budgeting features	P4	0,2	0	0
Suggested Diets	P5	0	0,1	0
Summary:		3,3	3	1,6

In the case study analysis, the developed web application managed to get most points out of all three selected web applications in terms of feature count. It is relevant to note that these feature selections may not be indicative or representative of each applications quality as they are designed for vastly distinctive design and use case goals in mind. For example, the thesis web application is specifically designed as a tool for supervision of transactions and health, while Mealime is designed for dieting and Our Groceries as a lightweight quick-to-access, light-to-use design purpose. In a different matrix the results can vary a lot and the selection can be biased. For the purposes of this matrix however, I calculated the standard deviation s with the following formula.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Where s is standard deviation, N-1 is the number of degrees of freedom in the vector of deviations and $(x_i - \overline{x})^2$ is the deviation range from mean. The resulting standard deviation was s=0,018 rounded. Still, this should only be considered within the limitations of the parameters and does not necessarily indicate quality of compared applications overall.

5.3 Available resources & considerable limiting factors

The web application was developed privatively for the thesis purposes, and it did not have any additional resources available for utilizing. MongoDB offers a free plan for limited usage that could not be used in real world industrial level web application use cases, but it is perfect for simulating a mock of a program and test out the functionality to the database. Figma is a free-to-use mock-up modelling tool which allows for one-project-per-user mocking, although there are other competitors that offer similar services as well. At one point the web application was supposed to be a serverless solution served from Amazon's AWS S3 bucket but due to the limitations and high costs of the AWS service, the web application was not deployed to a serverless solution as other competitors to AWS did not seem to offer viable solutions for this level of project.

6 Consideration

6.1 Conclusion & synthesis

The web application was able to fulfill most of its designed goals within the time limit of the thesis project and in some cases, it even exceeded the original design goals and expectations. The web application successfully and securely manages login and then CRUD operations of the shopping cart and finally the history can poll the server for purchase history event data and successfully render it to the end user.

There are still bugs in the web application, some refactoring to be done especially for error message duplicate code and the frontend could use some basic functionality such as loading state rendering for data when waiting for a response from the server and more tools for the end user to influence shopping cart, purchase history or their own user settings as they are currently only handled manually by the admin and in a real world industrial level web application these are some of the basic functionalities that should exist in this type of a web application.

The web application also would need a more robust cybersecurity as salting is simply the most basic level of cybersecurity measures in any web application. The web application also does not feature any testing, nor is it designed for mobile application scaling which is ultimately the end goal for such an app. In the original concept the web application was meant to be produced by using React Native or other mobile application language solutions but due to the authors unfamiliarity with mobile application language solutions and tight resources and time constraints as is, the author opted for a more familiar programming language solution that will eventually migrate the codebase to the appropriate programming language. For a proof of concept, the developed web application is sufficient and proves that there is potential for a handy tool though it clearly needs further development.

6.2 What was learnt

Considering the author was relatively inexperienced, the project taught a lot. For one it became clearer how much work and time even a proof-of-concept application takes if done with care for code quality and foresight into the future of the application. The initial concept and goal of the application and the final result vary quite a lot which indicates the authors inexperience in software development. Learning how to properly create backend code, manage data and communicate between the server and the frontend were all valuable experiences to the author.

While the frontend was more familiar, even it taught new experiences and perspectives on higher-quality web application development. Managing a wider full stack solution all the way from design process and architecture to the details of code solution and code quality management and all the goals that were designed but did not have the time to investigate were all part of valuable lessons throughout the project. The thesis work gave a glance into the expansive world of web application development needs, what worked and what did not work, and it serves as a solid foundation to be built upon.

6.3 Development needs

Considering the future of the web application, the first steps would be to do the migration from MERN stack to another technology solution. Considering that the application is ultimately and primarily designed for a mobile application platform that can be accessed and used anywhere on the go from work to home for example, a desktop application is less than ideal. Before any kind of actual deployment, the application needs to have more extensive security guards to protect sensitive user data from hostile outside actors trying to do data breaches. This may also mean some further refactoring to the codebase in general due to having some security gaps present even in the salting process, such as showing the password unencrypted in the network console tab before sending it to the server rather than having the encryption at a lower level. The web application also needs proper testing in general since this project did not do any testing whatsoever and it is an important aspect of any real-world web application needs. It would also be beneficial to get feedback from a pilot user tester group to see what other development needs the web application would have.

6.4 Further development opportunities

As mentioned in the previous chapters, the web application should eventually migrate to using a mobile application programming language. However other suggested features for future development needs would be to have a similar feature to Mealime's suggested diets based on user selected parameters such as dieting goals, and the ability to create your own dishes and save up a grocery list to the dishes directly so that by clicking on a dish, it would automatically fetch all the ingredients for you and add them to your shopping cart. Another handy feature would be to utilize a mobile devices camera and AI to scan for the products date tag on the product and automatically set it as the last use date, and then give a notification to the end user once the last use date is coming near.

Furthermore, the same photo-detection AI feature could also be able to automatically set product prize and calories simply by taking a photo from the product or the store shelf of the product which has the price to automatically determine the price and summaries for the total price and calories on the individual shopping event. Other suggested features would be the ability to change the color theme for color blind modes and more language options other than English and Finnish.

The web application also needs a shared universal service state that is able to synchronize between various devices if linked together. This synchronization between different devices is part of any modern web application unless there is a good reason to disallow that feature, such as security risks and concerns. A shopping cart application, despite holding some sensitive data, does not hold anything that sensitive to not allow for this kind of feature from being developed.

Sources

Prokhorova, A. 8 best practices for UI card design. 14.11.2022. Medium.com. Referred: 3.5.2023. https://uxdesign.cc/8-best-practices-for-ui-card-design-898f45bb60cc

A Short History of the Internet. 3.12.2020. Science and Media Museum. An article about a condensed version of the history of the internet. Referred: 15.4.2022. https://www.scienceandmediamuseum.org.uk/objects-and-stories/short-history-internet.

Atchison, L. Identity, trust, and their role in modern applications. 7.4.2022. InfoWorld. Referred. 4.4.2023. https://www.infoworld.com/article/3665655/identity-trust-and-their-role-in-modern-applications.html

Bianchi, T. Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2022. N.d. Statista. Referred 14.3.2023. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/

Best Web Development Stack 2023. 7.7.2022. Dev. 7.7.2022. Updated 11.1.2023. Referred: 4.4.2023. https://dev.to/theme_selection/best-web-development-stack-2jpe

Best Web Development Stacks to Use in 2023. 13.1.2023. NobleDesktop. Referred: 4.4.2023. https://www.nobledesktop.com/classes-near-me/blog/best-web-development-stacks

Bongarts, M. When You SHOULD Duplicate Code. 6.9.2021. Medium. Referred: 21.3.2023. https://medium.com/@mariusbongarts/when-you-should-duplicate-code-b0d747bc1c67

Chadd, K. The history of cybersecurity. 24.11.2022. Avast. Referred: 27.4.2022. https://blog.avast.com/history-of-cybersecurity-avast.

Cosset, D. MongoDB: Normalization vs Denormalization. 13.8.2017. Referred: 14.3.2023. https://dev.to/damcosset/mongodb-normalization-vs-denormalization

Design Consistency Guide UI and UX Best Practices. 31.8.2023. Uxpin.com. Referred: 2.5.2023. https://www.uxpin.com/studio/blog/guide-design-consistency-best-practices-ui-ux-designers/

Duplicate Code: Everything you need to know. N.d. codegrip. Referred 21.3.2023. https://www.codegrip.tech/productivity/what-is-duplicate-code/#:~:text=Having%20duplicate%20code%20will%20make,the%20decreased%20productivity%20of%20developers.

Dybå, T. Prikladnicki, R. Rönkkö, K. Seaman, C. Sillito, J. Qualitative research in software engineering. 28.5.2011. Springer Link. Referred: 11.5.2023. https://link.springer.com/article/10.1007/s10664-011-9163-y

Express/Node introduction. N.d. MDN Web Docs. Referred 9.3.2023. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

February 2023 Web Server Survey. 28.2.2023. Netcraft. Referred: 14.3.2023. https://news.netcraft.com/archives/category/web-server-survey/

Fleck, R. 10 Fundamental UI Design Principles You Need to Know. 6.12.2021 Dribble.com. referred 2.5.2023. https://dribbble.com/resources/ui-design-principles

Chien, D. File Structure. N.d. Board Institute. Referred: 21.3.2023. https://mitcommlab.mit.edu/broad/commkit/file-structure/

Gillis, A. March 2023. MongoDB. Techtarget. Referred 9.2.2023. https://www.techtarget.com/searchdatamanagement/definition/MongoDB

How Websites And Web Design Have Changed Over 20 Years. 29.7.2020. SQ. Referred: 7.5.2022. https://www.sqdigital.co.uk/insights/how-websites-and-web-design-have-changed-over-20-years/#:~:text=The%20increase%20in%20the%20use,much%20more%20visually%20appealing%20websites.

How To Structure React Projects From Beginner To Advanced. 11.7.2022. Web Dev Simplied Blog. Referred: 20.3.2023. https://blog.webdevsimplified.com/2022-07/react-folder-structure/

JavaScript: How Did It Get So Popular? Published 4.9.2018, updated: 25.3.2022. CodeAcademy. Referred 7.5.2022. https://www.codecademy.com/resources/blog/javascript-history-popularity/.

Jonhston, J. How to build a web app: A beginner's guide (2022). 23.4.2019. budibase. Referred: 16.3.2023. https://budibase.com/blog/how-to-make-a-web-app/

Loukides, M. Swoyer, S. 15.7.2020. O'Reilly. Referred 9.3.2023. https://www.oreilly.com/radar/microservices-adoption-in-2020/

Makhov, V. Server vs. Serverless: Benefits and Downsides. 25.11.2021. Nordic APIS. Referred: 16.3.2023. https://nordicapis.com/server-vs-serverless-benefits-and-downsides/

Mariana, C. Top 10 Serverless Hosting Providers. N.d. Back4app. Referred: 16.3.2023. https://blog.back4app.com/serverless-hosting-providers/

Package Management Basics. N.d. MDN Web Docs. Referred. 5.6.2022. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management.

Radwan, A. What Makes Learning Web Development Worthwhile in the Coming Years? 19.12.2022. Nerd Level Tech. Referred: 11.5.2023. https://nerdleveltech.com/what-makes-learning-web-development-worthwhile-in-the-coming-years/

Runeson, P. Höst, M. Guidelines for conducting and reporting case study research in software engineering. 19.12.2008. Springer Link. Referred: 11.5.2023. https://link.springer.com/article/10.1007/s10664-008-9102-8

Silva, M. V. Don’t repeat yourself. 12.12.2019. Medium. https://medium.com/code-thoughts/dont-repeat-yourself-caa413910753

Song, C. How to choose the right tech stack for web development. 22.7.2022. Educative. Referred: 4.4.2023. https://www.educative.io/blog/choose-a-web-development-tech-stack

Stol, K-J. Fitzgerald, B. A Holistic Overview of Software Engineering Research Strategies. 18.5.2015. Date published in journal: 27.7.2015. IEEE Xplore. Referred: 11.5.2023 https://ieeex-plore.ieee.org/document/7167427

What are Microservices? N.d. IBM. Referred 9.3.2023. https://www.ibm.com/topics/microservices

What Are the 10 Rules of Good UI Design? What Is Good UI/UX Design? 17.4.2023. Medium.com. Referred: 3.5. 2023. https://medium.com/@OPTASY.com/what-are-the-10-rules-of-good-ui-design-what-is-good-ui-ux-design-3ecc0d575c8f

What factors go into choosing a technology stack? N.d. Aha! Referred: 4.4.2023. https://www.aha.io/roadmapping/guide/it-strategy/technology-stack

What is MongoDB? N.d. IBM. Referred: 9.3.2023. https://www.ibm.com/topics/mongodb

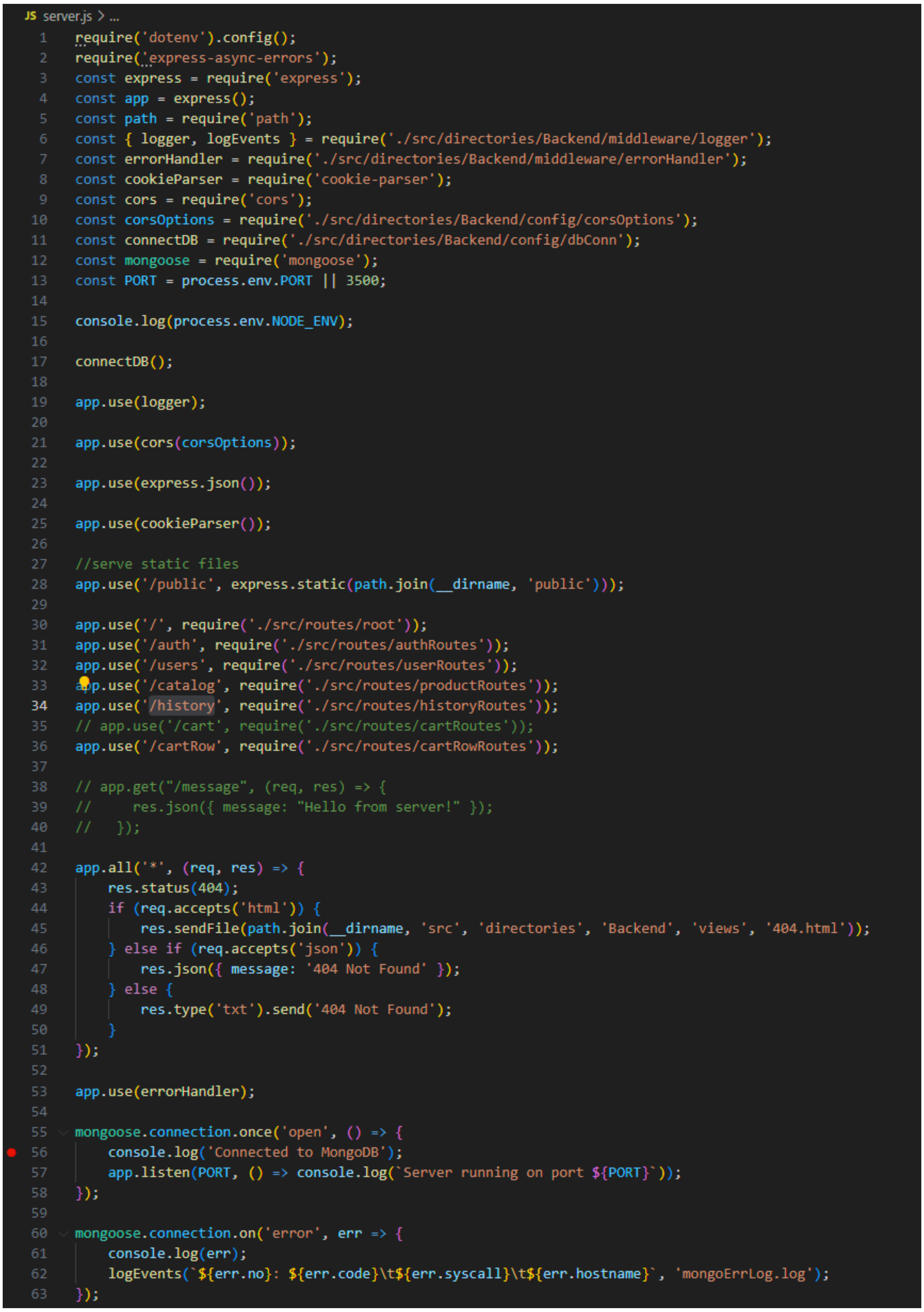
Wohlin, C. Höst, M. Case Study Research in Software Engineering – It is a Case, and it is a Study, but is it a Case Study? May 2021. Elsevier. Referred: 11.5.2023 https://www.sciencedi-rect.com/science/article/pii/S0950584921000033

Wohlin, C. Höst, M. Henningsson, K. Empirical Research Methods in Software Engineering. Pages 7-23. Published: 20.8.2003. LNCS. https://link.springer.com/chapter/10.1007/978-3-540-45143-3_2

Zola, W. 6 Rules of Thumb for MongoDB Schema Design. 11.7.2014. Updated: 2.11.2022. Referred: 14.3.2023. https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design

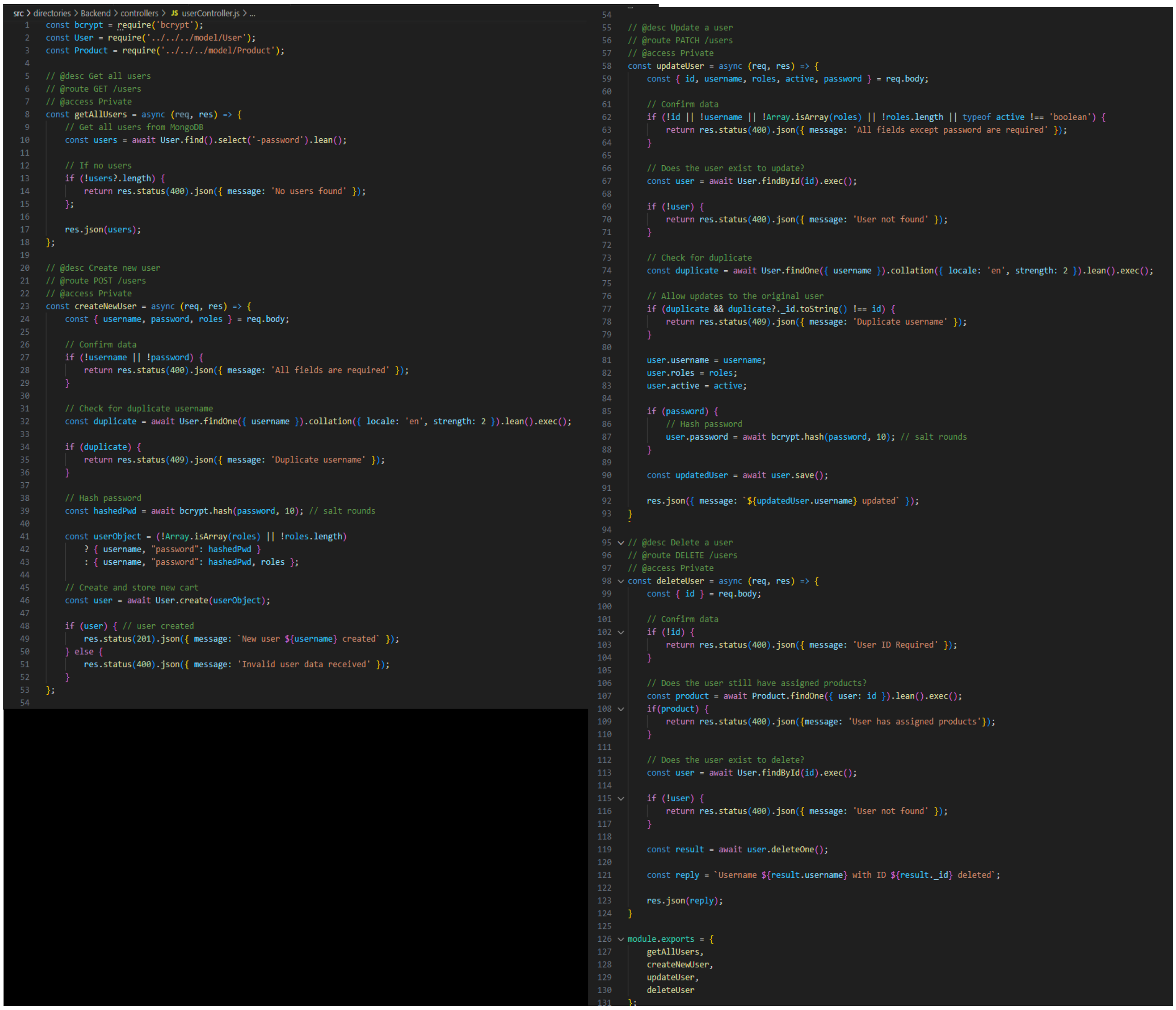
Attachments

Attachment 1. server.js



```
JS server.js > ...
require('dotenv').config();
require('express-async-errors');
const express = require('express');
const app = express();
const path = require('path');
const { logger, logEvents } = require('./src/directories/Backend/middleware/logger');
const errorHandler = require('./src/directories/Backend/middleware/errorHandler');
const cookieParser = require('cookie-parser');
const cors = require('cors');
const corsOptions = require('./src/directories/Backend/config/corsOptions');
const connectDB = require('./src/directories/Backend/config/dbConn');
const mongoose = require('mongoose');
const PORT = process.env.PORT || 3500;

console.log(process.env.NODE_ENV);

connectDB();

app.use(logger);

app.use(cors(corsOptions));

app.use(express.json());

app.use(cookieParser());

//serve static files
app.use('/public', express.static(path.join(__dirname, 'public')));

app.use('/', require('./src/routes/root'));
app.use('/auth', require('./src/routes/authRoutes'));
app.use('/users', require('./src/routes/userRoutes'));
app.use('/catalog', require('./src/routes/productRoutes'));
app.use('/history', require('./src/routes/historyRoutes'));
// app.use('/cart', require('./src/routes/cartRoutes'));
app.use('/cartRow', require('./src/routes/cartRowRoutes'));

// app.get("/message", (req, res) => {
//     res.json({ message: "Hello from server!" });
//   });

app.all('*', (req, res) => {
    res.status(404);
    if (req.accepts('html')) {
        res.sendFile(path.join(__dirname, 'src', 'directories', 'Backend', 'views', '404.html'));
    } else if (req.accepts('json')) {
        res.json({ message: '404 Not Found' });
    } else {
        res.type('txt').send('404 Not Found');
    }
});

app.use(errorHandler);

mongoose.connection.once('open', () => {
    console.log('Connected to MongoDB');
    app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
});

mongoose.connection.on('error', err => {
    console.log(err);
    logEvents(`${err.no}: ${err.code}\t${err.syscall}\t${err.hostname}`, 'mongoErrLog.log');
});
```

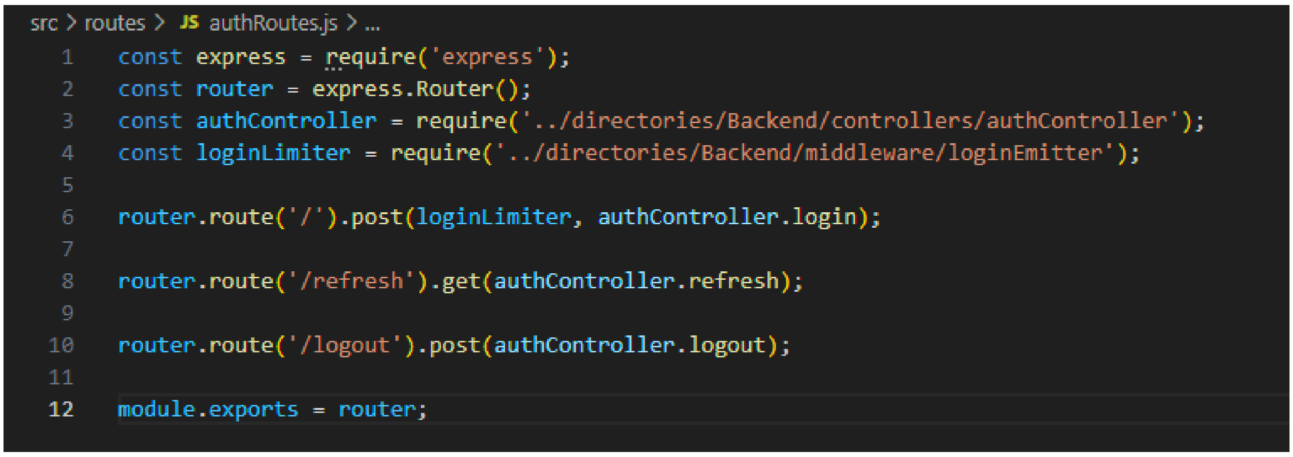
Attachment 2. Login controller



src > directories > Backend > controllers > JS userController.js > ...

```javascript
const bcrypt = require('bcrypt');
const User = require('../../../model/User');
const Product = require('../../../model/Product');

// @desc Get all users
// @route GET /users
// @access Private
const getAllUsers = async (req, res) => {
    // Get all users from MongoDB
    const users = await User.find().select('-password').lean();

    // If no users
    if (!users?.length) {
        return res.status(400).json({ message: 'No users found' });
    };

    res.json(users);
};

// @desc Create new user
// @route POST /users
// @access Private
const createNewUser = async (req, res) => {
    const { username, password, roles } = req.body;

    // Confirm data
    if (!username || !password) {
        return res.status(400).json({ message: 'All fields are required' });
    }

    // Check for duplicate username
    const duplicate = await User.findOne({ username }).collation({ locale: 'en', strength: 2 }).lean().exec();

    if (duplicate) {
        return res.status(409).json({ message: 'Duplicate username' });
    }

    // Hash password
    const hashedPwd = await bcrypt.hash(password, 10); // salt rounds

    const userObject = (!Array.isArray(roles) || !roles.length)
        ? { username, "password": hashedPwd }
        : { username, "password": hashedPwd, roles };

    // Create and store new cart
    const user = await User.create(userObject);

    if (user) { // user created
        res.status(201).json({ message: `New user ${username} created` });
    } else {
        res.status(400).json({ message: 'Invalid user data received' });
    }
};

// @desc Update a user
// @route PATCH /users
// @access Private
const updateUser = async (req, res) => {
    const { id, username, roles, active, password } = req.body;

    // Confirm data
    if (!id || !username || !Array.isArray(roles) || !roles.length || typeof active !== 'boolean') {
        return res.status(400).json({ message: 'All fields except password are required' });
    }

    // Does the user exist to update?
    const user = await User.findById(id).exec();

    if (!user) {
        return res.status(400).json({ message: 'User not found' });
    }

    // Check for duplicate
    const duplicate = await User.findOne({ username }).collation({ locale: 'en', strength: 2 }).lean().exec();

    // Allow updates to the original user
    if (duplicate && duplicate?._id.toString() !== id) {
        return res.status(409).json({ message: 'Duplicate username' });
    }

    user.username = username;
    user.roles = roles;
    user.active = active;

    if (password) {
        // Hash password
        user.password = await bcrypt.hash(password, 10); // salt rounds
    }

    const updatedUser = await user.save();

    res.json({ message: `${updatedUser.username} updated` });
}

// @desc Delete a user
// @route DELETE /users
// @access Private
const deleteUser = async (req, res) => {
    const { id } = req.body;

    // Confirm data
    if (!id) {
        return res.status(400).json({ message: 'User ID Required' });
    }

    // Does the user still have assigned products?
    const product = await Product.findOne({ user: id }).lean().exec();
    if(product) {
        return res.status(400).json({message: 'User has assigned products'});
    }

    // Does the user exist to delete?
    const user = await User.findById(id).exec();

    if (!user) {
        return res.status(400).json({ message: 'User not found' });
    }

    const result = await user.deleteOne();

    const reply = `Username ${result.username} with ID ${result._id} deleted`;

    res.json(reply);
}

module.exports = {
    getAllUsers,
    createNewUser,
    updateUser,
    deleteUser
};
```

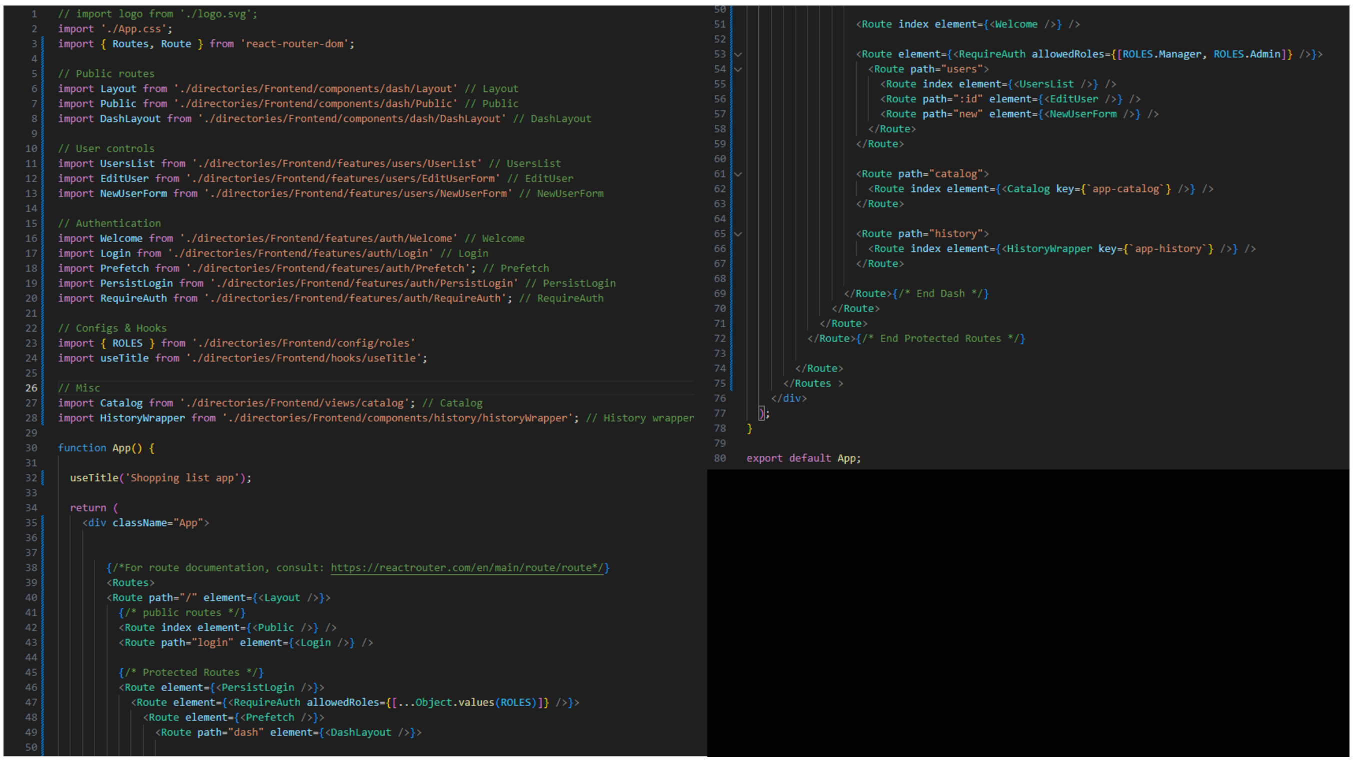
Attachment 3. Login router



src > routes > JS authRoutes.js > ...

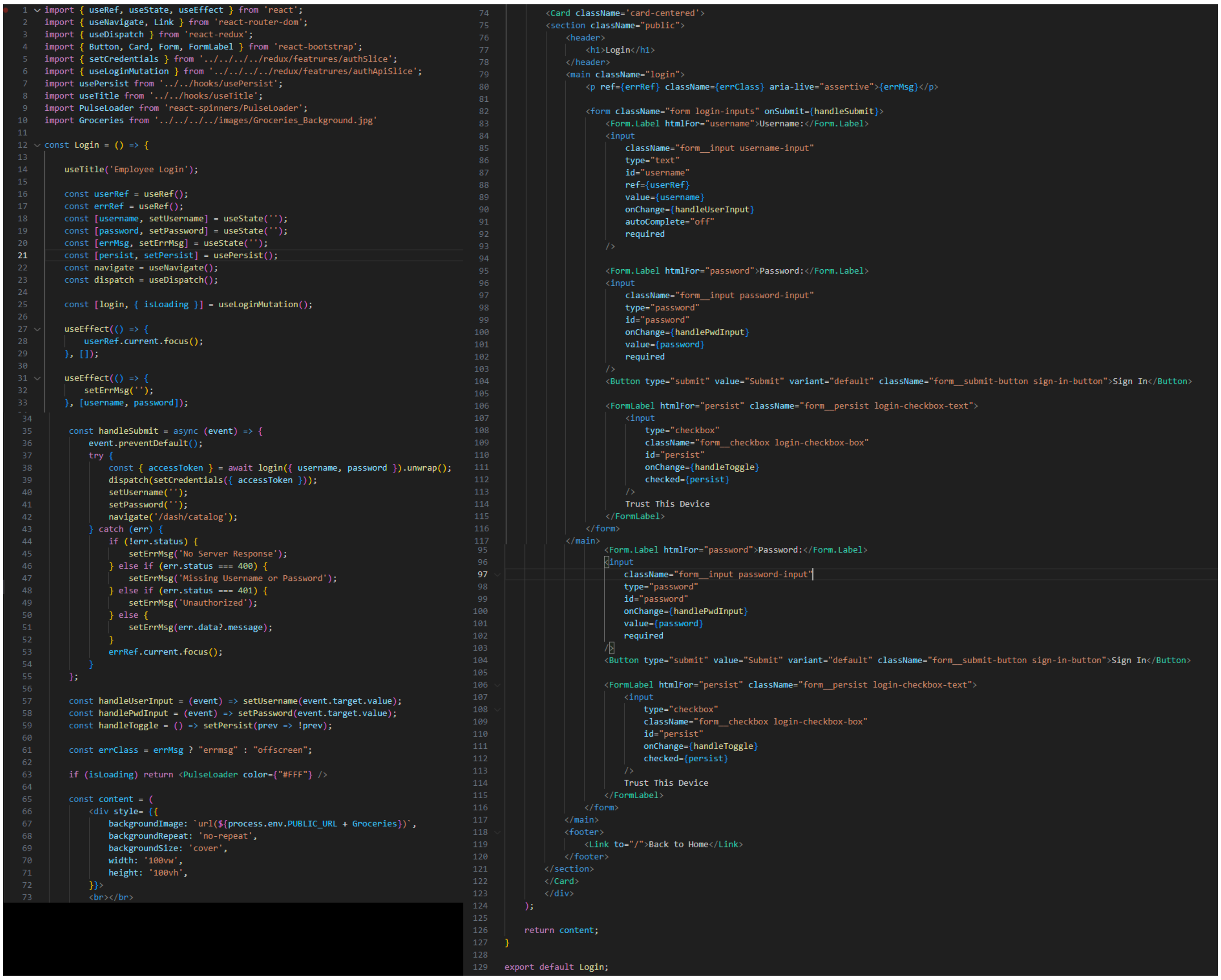
```javascript
const express = require('express');
const router = express.Router();
const authController = require('../directories/Backend/controllers/authController');
const loginLimiter = require('../directories/Backend/middleware/loginEmitter');

router.route('/').post(loginLimiter, authController.login);

router.route('/refresh').get(authController.refresh);

router.route('/logout').post(authController.logout);

module.exports = router;
```

Attachment 4. App.js



```javascript
// import logo from './logo.svg';
import './App.css';
import { Routes, Route } from 'react-router-dom';

// Public routes
import Layout from './directories/Frontend/components/dash/Layout' // Layout
import Public from './directories/Frontend/components/dash/Public' // Public
import DashLayout from './directories/Frontend/components/dash/DashLayout' // DashLayout

// User controls
import UsersList from './directories/Frontend/features/users/UserList' // UsersList
import EditUser from './directories/Frontend/features/users/EditUserForm' // EditUser
import NewUserForm from './directories/Frontend/features/users/NewUserForm' // NewUserForm

// Authentication
import Welcome from './directories/Frontend/features/auth/Welcome' // Welcome
import Login from './directories/Frontend/features/auth/Login' // Login
import Prefetch from './directories/Frontend/features/auth/Prefetch'; // Prefetch
import PersistLogin from './directories/Frontend/features/auth/PersistLogin' // PersistLogin
import RequireAuth from './directories/Frontend/features/auth/RequireAuth'; // RequireAuth

// Configs & Hooks
import { ROLES } from './directories/Frontend/config/roles'
import useTitle from './directories/Frontend/hooks/useTitle';

// Misc
import Catalog from './directories/Frontend/views/catalog'; // Catalog
import HistoryWrapper from './directories/Frontend/components/history/historyWrapper'; // History wrapper

function App() {

  useTitle('Shopping list app');

  return (
    <div className="App">


        {/*For route documentation, consult: https://reactrouter.com/en/main/route/route*/}
        <Routes>
        <Route path="/" element={<Layout />}>
          {/* public routes */}
          <Route index element={<Public />} />
          <Route path="login" element={<Login />} />

          {/* Protected Routes */}
          <Route element={<PersistLogin />}>
            <Route element={<RequireAuth allowedRoles={[...Object.values(ROLES)]} />}>
              <Route element={<Prefetch />}>
                <Route path="dash" element={<DashLayout />}>

                  <Route index element={<Welcome />} />

                  <Route element={<RequireAuth allowedRoles={[ROLES.Manager, ROLES.Admin]} />}>
                    <Route path="users">
                      <Route index element={<UsersList />} />
                      <Route path=":id" element={<EditUser />} />
                      <Route path="new" element={<NewUserForm />} />
                    </Route>
                  </Route>

                  <Route path="catalog">
                    <Route index element={<Catalog key={`app-catalog`} />} />
                  </Route>

                  <Route path="history">
                    <Route index element={<HistoryWrapper key={`app-history`} />} />
                  </Route>

                </Route>{/* End Dash */}
              </Route>
            </Route>
          </Route>{/* End Protected Routes */}

        </Route>
      </Routes >
    </div>
  );
}

export default App;
```

Attachment 5. login.js



```
import { useRef, useState, useEffect } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import { Button, Card, Form, FormLabel } from 'react-bootstrap';
import { setCredentials } from '../../../../redux/featrures/authSlice';
import { useLoginMutation } from '../../../../redux/featrures/authApiSlice';
import usePersist from '../../hooks/usePersist';
import useTitle from '../../hooks/useTitle';
import PulseLoader from 'react-spinners/PulseLoader';
import Groceries from '../../../../images/Groceries_Background.jpg'

const Login = () => {

    useTitle('Employee Login');

    const userRef = useRef();
    const errRef = useRef();
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [errMsg, setErrMsg] = useState('');
    const [persist, setPersist] = usePersist();
    const navigate = useNavigate();
    const dispatch = useDispatch();

    const [login, { isLoading }] = useLoginMutation();

    useEffect(() => {
        userRef.current.focus();
    }, []);

    useEffect(() => {
        setErrMsg('');
    }, [username, password]);

    const handleSubmit = async (event) => {
        event.preventDefault();
        try {
            const { accessToken } = await login({ username, password }).unwrap();
            dispatch(setCredentials({ accessToken }));
            setUsername('');
            setPassword('');
            navigate('/dash/catalog');
        } catch (err) {
            if (!err.status) {
                setErrMsg('No Server Response');
            } else if (err.status === 400) {
                setErrMsg('Missing Username or Password');
            } else if (err.status === 401) {
                setErrMsg('Unauthorized');
            } else {
                setErrMsg(err.data?.message);
            }
            errRef.current.focus();
        }
    };

    const handleUserInput = (event) => setUsername(event.target.value);
    const handlePwdInput = (event) => setPassword(event.target.value);
    const handleToggle = () => setPersist(prev => !prev);

    const errClass = errMsg ? "errmsg" : "offscreen";

    if (isLoading) return <PulseLoader color={"#FFF"} />

    const content = (
        <div style= {{
            backgroundImage: `url(${process.env.PUBLIC_URL + Groceries})`,
            backgroundRepeat: 'no-repeat',
            backgroundSize: 'cover',
            width: '100vw',
            height: '100vh',
        }}>
        <br></br>
        <Card className='card-centered'>
        <section className="public">
            <header>
                <h1>Login</h1>
            </header>
            <main className="login">
                <p ref={errRef} className={errClass} aria-live="assertive">{errMsg}</p>

                <form className="form login-inputs" onSubmit={handleSubmit}>
                    <Form.Label htmlFor="username">Username:</Form.Label>
                    <input
                        className="form__input username-input"
                        type="text"
                        id="username"
                        ref={userRef}
                        value={username}
                        onChange={handleUserInput}
                        autoComplete="off"
                        required
                    />

                    <Form.Label htmlFor="password">Password:</Form.Label>
                    <input
                        className="form__input password-input"
                        type="password"
                        id="password"
                        onChange={handlePwdInput}
                        value={password}
                        required
                    />
                    <Button type="submit" value="Submit" variant="default" className="form__submit-button sign-in-button">Sign In</Button>

                    <FormLabel htmlFor="persist" className="form__persist login-checkbox-text">
                        <input
                            type="checkbox"
                            className="form__checkbox login-checkbox-box"
                            id="persist"
                            onChange={handleToggle}
                            checked={persist}
                        />
                        Trust This Device
                    </FormLabel>
                </form>
            </main>
```

```
                    <Form.Label htmlFor="password">Password:</Form.Label>
                    <input
                        className="form__input password-input"
                        type="password"
                        id="password"
                        onChange={handlePwdInput}
                        value={password}
                        required
                    />
                    <Button type="submit" value="Submit" variant="default" className="form__submit-button sign-in-button">Sign In</Button>

                    <FormLabel htmlFor="persist" className="form__persist login-checkbox-text">
                        <input
                            type="checkbox"
                            className="form__checkbox login-checkbox-box"
                            id="persist"
                            onChange={handleToggle}
                            checked={persist}
                        />
                        Trust This Device
                    </FormLabel>
                </form>
            </main>
```

```
            <footer>
                <Link to="/">Back to Home</Link>
            </footer>
        </section>
        </Card>
        </div>
    );

    return content;
}

export default Login;
```