Expertise
and insight
for the future

Quang Nhat Mai

E-commerce Application using MERN stack

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

30 August 2018

Author Title Number of Pages Date	Quang Nhat Mai E-commerce Application using MERN stack 36 pages + x appendices 30 August 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen

This thesis is about studying the basic components of MERN Stack technology such as: MongoDB, ExpressJS framework, ReactJS library, and NodeJS platform. Discussing the basic functions of an e-commerce web application such as sign up, sign in, showing dashboards, displaying store categories and products Using MERN Stack technology in conjunction with Braintree to build a web application to search for product stores and payment gateway. Develop administrative functions for the website such as: user management, store management, statistics, and reports.

Since then, this topic is declared to research and create an online product search website so that small grocery stores and groceries can post and manage their products right on the website. website system and branding. At the same time, the store can link to its own website (if any). For customers, they can quickly search for urgent products from their nearby stores. Based on the search results, customers can directly contact the store owner to discuss more about the product they are looking for.

Keywords	

Contents

List of Abbreviations

DOM	Document Object Model
W3C	World Wide Web Consortium.
ISO	International Organization for Standardization
NPM	Node Package Manager
API	Application Program Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
JS	JavaScript
JSX	JavaScript XML
XML	Extensive Markup Language

1 Introduction

It is true that technology has become an essential tool for online marketing nowadays. However, there are numerous small shops and grocery stores with mostly offline business model in Vietnam recently. With this commerce model, it will bring a lot of bad experiences for both buyers and sellers. For instance, the seller has the product want to offer but the buyer may not know it, or the buyer may urgently need to purchase something, but the store is out of stock. Moreover, online shopping helps customers to choose a wide range of products, prices and they can compare them to each other easily.

Encountering the inadequacies and the weaknesses of the offline business model, making a website application for searching and buying things for each shop is very necessary right now. Recently, there have been many e-commerce sites exported such as Amazon, e-bay or the stores that can sell products via social media channels like Facebook. However, customers still find it difficult to choose the products they want because of the large variety of products on these sites and not focus on specific things. Moreover, the sellers have to spend a high amount of money on marketing or paying for fees. From there disadvantages, implement an online e-commerce web application for small grocery stores helps retailers can manage products on their own systems and not depend on the 3rd party website. For the customers, they can quickly search the products if it is available and come to store to pick it up and they can contact directly to the shop owner to learn more about the products that they are looking for.

In order to make a website that can acquire the needs of both customers and retailers, MERN (MongoDB, Express.js framework, ReactJS library, NodeJS platform) is one of the powerful stacks that can help us to develop an e-commerce web application.

2 E-commerce

2.1 Definition

E-comm, EC for short (E-commerce) is a concept referring to transactions, purchase and sale of goods and services by the internet.

E-commerce was first known in the 1960s. After years of development, as mobile devices became popular, social media increasingly affirmed the power and the boom of the webpage. Launchers promote the rapid development of commerce (E-commerce).[1]

2.2 Types

Currently, there are many forms of e-commerce, including the following basic forms:

B2B (Business to Business): is a trade between companies, businesses and organizations. About 80% of e-commerce today falls into this category.

B2C (Business to Consumer): is an Internet-based business to directly exchange the goods and services it creates or distributes to consumers.

C2B (Consumer to Business): is a consumer who sells their products or services to a business or organization.

C2C (Consumer to Consumer): is when a consumer sells his goods or services to another consumer.

There are also G2C, G2B, etc., but used less often than these four basic forms.

2.3 Advantages

Global market: Clearly, when you open a physical store, you will only be able to deliver your goods and services in a small geographic area. E-commerce will help you solve that problem. E-commerce helps you reach the market quickly, expanding the market to the maximum level compared to direct sales, so that products and services are easily introduced, purchased and sold through retailers. and online market.

Always open: In e-commerce, running an online business is much easier, it's always open 24h / 7/365. For businesses, it's a great opportunity to increase sales opportunities all the time.

Budget savings: Compared with traditional forms of commercial business, all costs when e-commerce business are reduced: the cost of renting booths, salespeople and management is much more economical. . Naturally, when sellers save operating costs, they can offer more incentives and better discounts for their customers. At this time, the customer is the next beneficiary. Mutual benefit, isn't it great?

Inventory management: By using electronic tools to speed up the ordering, delivery, and payment processes, e-commerce businesses can save billions of operating costs and reduce amount of inventory.

Most accurate customer marketing: With access to customer data and the opportunity to track customers' buying habits, e-commerce businesses can quickly identify and market products and services. service. Service most suitable for consumers.

Work anywhere, buy anywhere: Running an e-commerce business allows you to not need to sit in the office, and buying does not force you to go to the supermarket. Everything the seller and the buyer needs is an internet-connected device and that's all.[2]

2.4 Challenges

Internet access required: When participating in the EC, to be able to buy and sell, you need a device connected to the internet. Currently, most people have internet access but, in many areas, it is still very limited.

Not enough to trust: Products and services that cannot be seen, touched, held or felt directly, are not allowed to try as a prudent buyer. Doubt in both buyers and sellers leads to many incomplete transactions, especially when they have dealt with untrusted partners before.

Limited payment methods: Currently, the most popular payment method in Vietnam when buying goods online is to receive and pay. Payment gateway in Vietnam is growing quite strong, but not reliable enough for users to use as the main payment method. Therefore, it also contributes to teething.

In addition, e-commerce business also faces many other challenges: technical, competitors, payment, etc.

3 MERN Stack

3.1 JavaScript

JavaScript is a scripting, object-oriented, cross-platform programming language. Objects of host environment can be connected to JavaScript and arrange ways to operate them.

Standard libraries for objects are contained by JavaScript, for such as Array, Date, Math, and the essence component of programming languages for instance managers, control framework and statements.

By adding objects, JavaScript could be protracted for many principles, such as:

- **Client-side JavaScript:** JavaScript is developed by implementing objects for controlling the browser and DOM. For instance, an application is granted by client-side extensions to influence components on an HTML page and answer to user behavior like mouse hovers, form input and page changeover.

- **Server-side JavaScript:** JavaScript is developed by implementing the supplementary objects required to run JavaScript on the server. For instance, an application is granted by this server-side extension to connect to a database, transfer data frequently from one request to other section of the application or execute application with another function file on the server.

In 1996, JavaScript was officially named ECMAScript. ECMAScript 2 was released in 1998 and ECMAScript 3 was released in 1999. It is continuously evolving into today's JavaScript, now works on all browsers and devices from mobile to desktop. Open standard language can be used by association to establish their own JavaScript applications. The ECMAScript Standard is one of the parts of the ECMA-262 specification.

ISO has approved the ECMA-262 standard at ISO-16262. The ECMAScript standard does not include descriptions for the DOM, it is standardized by the W3C.

The DOM specifies how your scripts display HTML objects. To get a advance anticipate of the distinctive innovations used when programming with JavaScript, check out the JavaScript technology analysis article. [3]

3.2 NodeJS

Node.js is an open source, a system application and furthermore is an environment for servers. Nodejs is an independent development platform built on Chrome's JavaScript Runtime that we can build network applications quickly and easily. Google V8 JavaScript engine is used by Node.js to execute code. Moreover, a huge proportion of essential modules are written in JavaScript

Node.js accommodate a built-in library which allows applications to serve as a Web-server left out demanding software like Apache HTTP Server, Nginx or IIS.

An event-driven, non-blocking I / O mechanisms (Input / Output) are implemented by Node.js. It optimizes application throughout and is extremely high extensible. Node.js use asynchronous in it functions. Therefore, Node.js processes and executes all tasks in the background (background processing).

products that have a lot of traffic are applying Node.js. Nonetheless, Node.js handle the application that need to spread expeditiously, develop innovation, or build Startup projects as rapidly as possible.[4]

Applications using NodeJS:

- WebSocket server
- Notification system
- Applications that need to upload files on the client.
- Other real-time data applications.

NodeJS Pros:

- Node.js is the exclusive application that with only a single thread, it can obtain and handle numerous connections. Building new threads for each query is not needed, therefore the structure expends the least amount of RAM and run rapidly. Secondly, Node.js produces the most of server property without generate latency with the JavaScript's non-blocking I/O.
- **JSON APIs**. JSON Web services can take advantages of that because of the event-driven, non-blocking I/O structures and JavaScript-enabled model.

- **Single page application**. NodeJS is very suitable with an application on a single page. Node.js has the capability to handle different requests concurrent and quick return. Node JS should be used in an application that does not have to reload the page, including users who makes a vast number of requests and need a quick procedure to show professionalism.

- **Shelling tools Unix**. Node.js usually uses Unix to work. They can handle multiple processes and return them for best performance. Programmers often use Node.js to build real Web applications like chat, feeds, etc.

- **Streaming Data**. Typical websites send HTTP requests and also receive responses. Node.js can handle many questions and feedback, so they are suitable if the developer wants to create an application on the page. In addition, Node.js also builds proxies to stream the data, this is to ensure maximum operation for other data streams.

- **Real-time Web Application**. Node.js is sufficient to develop real-time innovations like chat apps, social networking services like Facebook, Twitter because of the opening of mobile application.

NodeJS Cons:

- **Resource-intensive applications**. Node.js is written in C ++ & JavaScript, so when programmers need to handle applications that use a lot of file conversion, video encoding, decoding, etc., they should not be used Node.js. Programmers need to use it more carefully in this case.

- The final purpose of NodeJS is like other programming languages such as Ruby, PHP, .NET, Python, that is developing web application. Therefore, do not expect NodeJS to outperform other language for now. But with NodeJS the application can be developed successfully as expected. [5]

NodeJS should not be used when:

- **Build resource-intensive applications**: Do not use Node.js when creating a video converter application. Node.js often comes down to bottlenecks when working with large files.

- **An all-CRUD-only application**: Node.js is not faster than PHP when doing heavy I/O tasks. In addition, with the long-term stability of other webserver scripts, its CRUD tasks have been optimized. Node.js will come up with odd APIs and never be used.

- **Stability in the application**: Within 11 years of development (2009-2020), the current version of Node.js is already v14.2.0. Every API can be changed – in a way that is not backwards compatible.

- **Lack of knowledge about Node.js**: Node.js is extremely dangerous in this case, you will fall into a world full of difficulties. With most non-blocking/async APIs, not understanding the problem will cause an error that you do not even know where it came from. Moreover, when the Node.js community is not strong enough, and there will be less support from the community.

NodeJS should be used when:

- **Building RESTful API (JSON)**. You can use Node.js in building RESTful API (JSON). They handle JSON very easily, even more than JavaScript. API servers when using Node.js usually do not have to perform heavy processing, but the number of concurrent requests is high.

- **Applications that demand alternative connection protocols**, not just http. With TCP protocol backing, any custom protocol can be built easily.

- **Real-time applications**.

- **Stateful websites**. Every request on the invariable procedure is handled by Node.js, therefore building caching is simpler: store it to a comprehensive variable then all requests can approach the cache. The status of one client can be

saved and shared with other clients and do not have to go through external memory. [6]

3.3 Express.js

Express.js is a framework built on top of Nodejs. It provides powerful features for web or mobile development. Express.js supports HTTP and middleware methods, making the API extremely powerful and easy to use.

Express implements extra features to developer which help them get a better programming environment, not scaling down the speed of NodeJS.

Importantly, the well-known frameworks of NodeJS apply Express.js as a substance function, for instance: Sails.js, MEAN.[7]

3.4 MongoDB

MongoDB is an open source database; it is also the leading NoSQL (*) database currently used by millions of people. It is written in one of the most popular programming languages today. In addition, MongoDB is cross-platform data that operates on the concepts of Collections and Documents, providing high performance with high availability and ease of expansion.[8]

(*) NoSQL is a source database format that does not use Transact-SQL to access information, this database was developed on JavaScript Framework on JSON data type. With its introduction, it has overcome the disadvantages of RDBMS relational data model to improve operating speed, functionality, model scalability, cache ...

Furthermore, MongoDB is a cross-platform database, performing on Collection and Document approach, it produces sharp production, huge availability, and effortless scalability.

Commonly used terms in MongoDB:

- **_id:** Almost every document required this field. The _id field illustrates a exceptional value in the MongoDB document. The _id field can also be interpreted as the primary key in the document. If you add a new document, MongoDB will automatically generate a _id representing that document and be unique in the MongoDB database.

- **Collection:** A group of many documents in MongoDB. Collection can be interpreted as a corresponding table in the RDBMS (Relational Database Management System) database. Collection resides in a single database. Collections do not have to define columns, rows or data types first.

- **Cursor:** This is a pointer to the outcome set of a query. The client can emphasize over a cursor to get the result.

- **Database:** The location of the collections, similar to the RDMS database that contains the tables. Each Database has a separate file stored on physical memory. Some MongoDB owners may contain various databases.

- **Document:** A transcript belonging to a Collection. Documents, in turn, include name and value fields.

- **Field:** A name-value pair in a document. A document may not need all the fields. The fields are like columns in a relational database.

- **JSON:** Short for JavaScript Object Notation. Human readability is in the plain text format representing structured data. JSON currently supports a lot of programming languages.

- **Index:** Exclusive data structures used to save a small allocation of data sets for simple scanning. The index puts the value of a individual field or sets of fields, sorted by the value of these fields. Index effectively supports the analysis of queries. Without an index, MongoDB will have to scan all the documents of the set to choose the documents that pair the query. This scan is ineffective and requires MongoDB to progress a vast amount of data.

MongoDB Atlas is MongoDB's cloud database launched in 2016 on AWS, Microsoft Azure and Google Cloud Platform.

The data in each Cluster in the Atlas is stored by Replication mechanism, with 3 nodes: 1 master (primary) and 2 slaves (secondary).

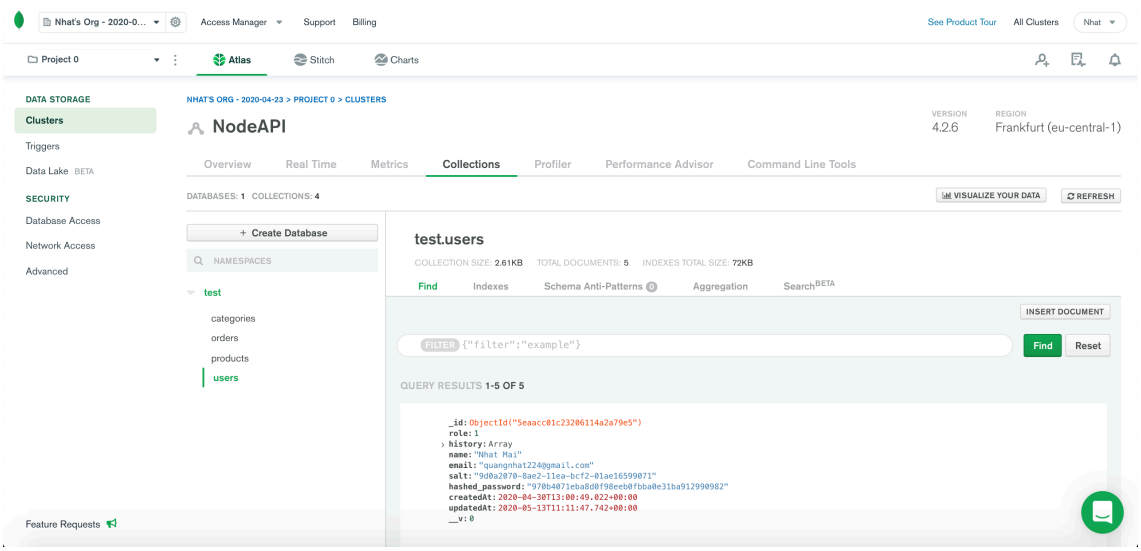


Figure 1. MongoDB Atlas screenshot.

3.5 ReactJS

3.5.1 Virtual-DOM

Virtual-DOM is a JavaScript object, each object contains all the information needed to create a DOM, when the data changes it will calculate the change between the object and the real tree, which will help optimize re-render DOM tree. It can be assumed that is a virtual model can handle client data.[9]

3.5.2 Component

React is built around components, not templates like other frameworks. A component can be created by the create Class function of the React object, the starting point when accessing this library.

ReactJS creates HTML tags unlike we normally write but uses Component to wrap HTML tags into stratified objects to render.

Among React Components, render function is the most important. It is a function that handles the generation of HTML tags as well as a demonstration of the ability to process via Virtual-DOM. Any changes of data at any time will be processed and updated immediately by Virtual-DOM.[10]

3.5.3 Props and State

Props: are not controlled by Component, actually stands for Properties.

```
1  import React from 'react';
2  import Menu from './Menu';
3  import '../styles.css';
4
5  const Layout = ({
6      title = "Title",
7      description = "Description",
8      className,
9      children
10 }) => (
11         <div>
12             <Menu />
13             <div className="jumbotron">
14                 <h2>{title}</h2>
15                 <p className="lead">{description}</p>
16             </div>
17             <div className={className}>{children}</div>
18         </div>
19     );
20
21 export default Layout;
```

Figure 2. Layout component.

The title = “Title” line creates a name attribute with the value "Title". It looks like a function call. It is true that props are passed to the component in the same way that an argument is passed to a function.

A component may also have a default props, so if the component does not pass any props, it will still be set.

State: private data is controlled by Component.

Like props, sate also holds information about the component. However, the type of information and how to handle it varies. State works differently than Props. The state is a component of the component, while props are passed in from the outside into the component. It should be noted that we should not update the state directly using this.state but always use setState to update. Update the state of the objects. Use setState to re-renders one component and all its children. This is great, because you don't have to worry about writing event handlers like any other language.[11]

3.5.4 Pros and Cons of ReactJS

Pros of ReactJS:

- Update data changes quickly.
- React is not a framework so it offloads the constraints of libraries together.
- Easy access to who understands JS.

Cons of ReactJS:

- ReactJS only serves the View tier, but the library size is on par with Angular while Angular is a complete framework.
- Incorporating ReactJS within common MVC frameworks demands reconfiguration.
- Hard to reach for beginners on website developing.[12]

3.6 MERN Stack in Website Development

3.6.1 Concept of Stack technology

The technical stack is a combination of technologies / frameworks / programming languages, etc. to create a complete software.

With current software, there are usually two parts: client side and server side, also known as frontend and backend. Therefore, people also split the backend stack, the frontend stack as well. We often use the first letter to name the technical stack: LAMP (Linux, Apache, MySQL, PHP), MEAN (MongoDB, Express, Angular, NodeJS).[13]

3.6.2 Concept of MERN Stack

The MERN stack is a complete open source combination, all JavaScript-related technologies are also the hottest today: MongoDB, Express.js, React / React Native, NodeJS. People use the MERN stack to build Universal React App.[14]

3.6.3 Highlights in MERN Stack

- Hot Reloading for React Components.
- The code snippets are divided by React Router.
- Multi-language support.
- Generate code support.
- Modular structure.
- Docker support.
- Can customize the MERN version for individual users. [8]

4 Funko Pop E-commerce Web Application (FunkoStore)

FunkoStore is an E-commerce Web Application using MERN stack that can help a toys retailer bring their products to the customers. Main function:

- Sign up and log in: Requires Users to register using their phone number or email
- Shopping cart: this feature helps users buy and check goods directly on the application
- Search: Users can search directly by typing in the search box for the product they want to see.
- Buy and pay: Customers who buy through the app can pay through many different payment gateways.

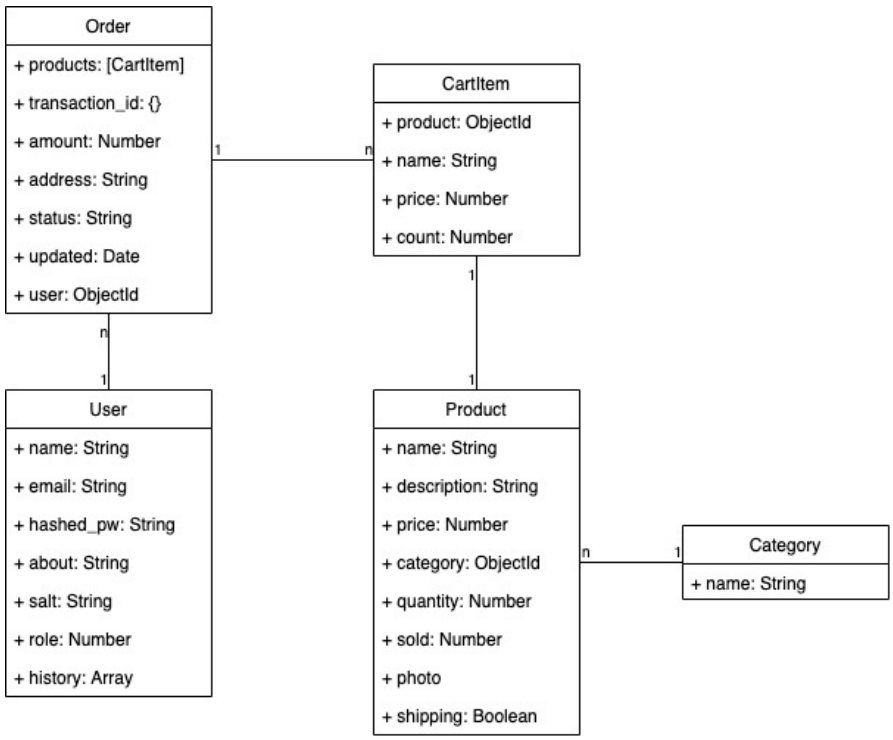


Figure 3. Simple Class Diagram of FunkoStore Application.

4.1 Home Page

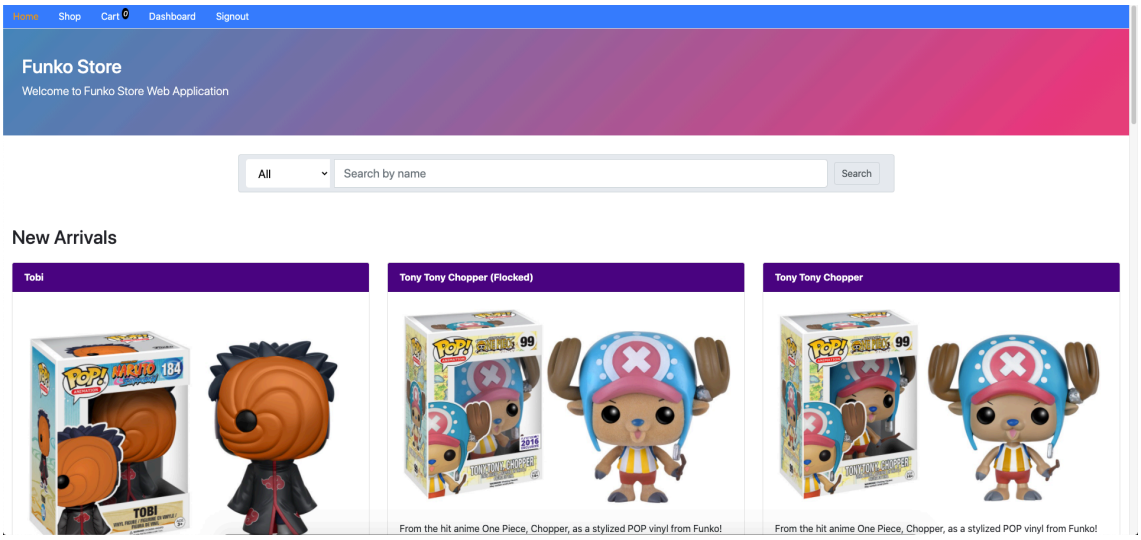


Figure 4. Main Home Page.

On the figure 4 can see the home page of the app. It includes the indispensable components of an online store like the function bar, the search bar and a list of products. Customers who visit the store can search for the products they want to buy and they can customize the filter on the toolbar to be able to buy the best products.

```
        <Layout
            title="Funko Store"
            description="Welcome to Funko Store Web Application"
            className="container-fluid"
        >
            <Search />
            <h2 className="mb-4">New Arrivals</h2>
            <div className="row">
                {productsByArrival.map((product, i) => (
                    <div key={i} className="col-4 mb-3">
                        <Card product={product} />
                    </div>
                ))}
            </div>

            <h2 className="mb-4">Best Sellers</h2>
            <div className="row">
                {productsBySell.map((product, i) => (
                    <div key={i} className="col-4 mb-3">
                        <Card product={product} />
                    </div>
                ))}
            </div>
        </Layout>
    );
};
```

Figure 5. Layout component of Home.js

The New Arrivals section list all the products those are recently added to the database in MongoDB. As it can be seen at figure 5, there are Best Sellers section beside the New Arrivals. React Hook is taking a big advantage in displaying these sorted products. The figure 6 below show how to use hook with our database.

At first, the productsBySell and productsByArrival are empty arrays, then the loadProductsBySell and loadProductsByArrival functions get the correct data from the database and change the state of those arrays.

```
const Home = () => {
    const [productsBySell, setProductsBySell] = useState([]);
    const [productsByArrival, setProductsByArrival] = useState([]);
    const [error, setError] = useState(false);

    const loadProductsBySell = () => {
        getProducts('sold').then(data => {
            if (data.error) {
                setError(data.error);
            } else {
                setProductsBySell(data);
            }
        });
    };

    const loadProductsByArrival = () => {
        getProducts('createdAt').then(data => {
            console.log(data);
            if (data.error) {
                setError(data.error);
            } else {
                setProductsByArrival(data);
            }
        });
    };

    useEffect(() => {
        loadProductsByArrival();
        loadProductsBySell();
    }, []);
```

Figure 6. React Hook in sorting products at Home page.

The getProducts fetch the API and take data from backend by each keyword. If data have any error it will return the err, or it will return the list of products.

The Search Bar used for the searching products by name and can be filtered out by categories. Here is the main method for it in Figure 7. If there is no error it will return a list of products that is searched before.

```
30    const searchData = () => {
31        // console.log(search, category);
32        if (search) {
33            list({ search: search || undefined, category: category }).then(
34                response => {
35                    if (response.error) {
36                        console.log(response.error);
37                    } else {
38                        setData({ ...data, results: response, searched: true });
39                    }
40                }
41            );
42        }
43    };
```

Figure 7. searchData function in Search Bar.

4.2 Login System

The login system of the app will ask the user to provide the username and password either they are customer or admin.

4.2.1 Sign Up

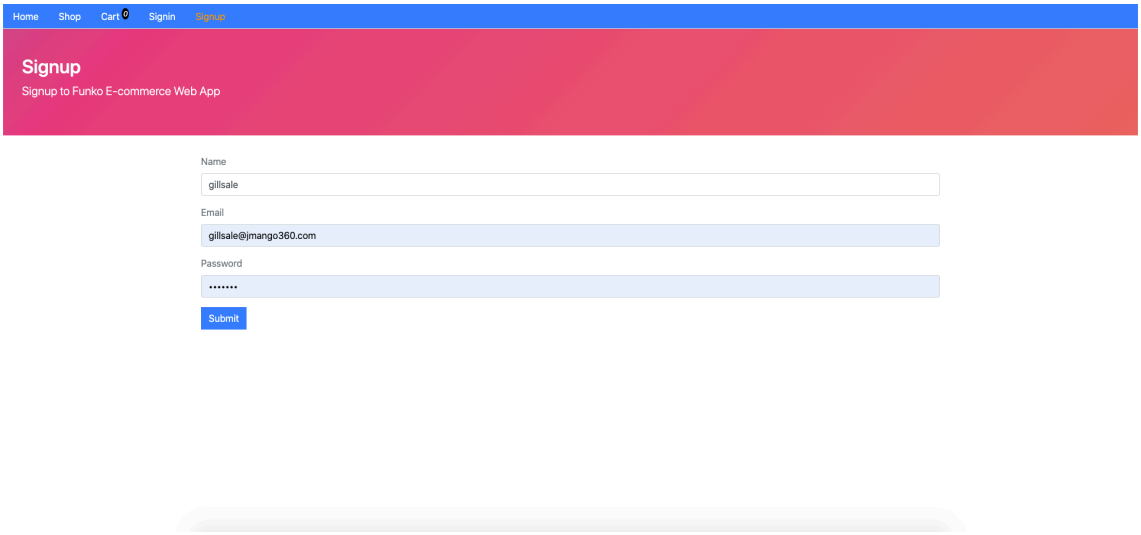


Figure 8. Sign -Up screen for the users.

Users login into the website need to sign in otherwise they have to register a new account. In figure 8, the sign-up screen only has 3 fields that require users to fill in is Name, Email and Password.

```
const Signup = () => {

    const [values, setValues] = useState({
        name: '',
        email: '',
        password: '',
        error: '',
        success: false
    });

    const { name, email, password, success, error } = values

    const handleChange = name => event => {
        setValues({ ...values, error: false, [name]: event.target.value })
    };

    const clickSubmit = event => {
        event.preventDefault();
        setValues({ ...values, error: false });
        signup({ name, email, password })
            .then(data => {
                if (data.error) {
                    setValues({ ...values, error: data.error, success: false })
                } else {
                    setValues({
                        ...values,
                        name: '',
                        email: '',
                        password: '',
                        error: '',
                        success: true
                    })
                }
            })
    };
```

Figure 9. JS code for Sign-Up function.

The empty strings are setting up for each field. After a user finished, they form and clicked submit button, the handleChange will setValues to each string and store all the data to the backend system. If there is an error occurred, none of data is stored. It can be explained in figure 9.

After a user sign up successfully. A green notification will display and tell them to sign in with their account like figure 10.

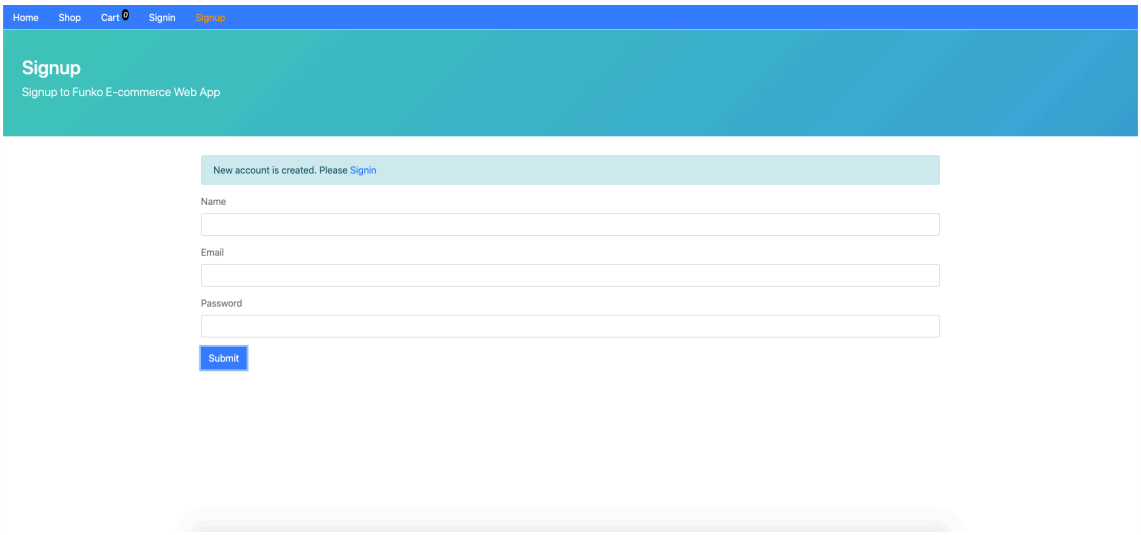


Figure 10. Successfully signed up.

4.2.2 Sign In

The Sign In is similar to the Sign-Up screen, only the email and password field are display. After the sign in is successful, the user will be redirected to the Dashboard page.

```
23    const clickSubmit = event => {
24        event.preventDefault();
25        setValues({ ...values, error: false, loading: true });
26        signin({ email, password }).then(data => {
27            if (data.error) {
28                setValues({ ...values, error: data.error, loading: false });
29            } else {
30                authenticate(data, () => {
31                    setValues({
32                        ...values,
33                        redirectToReferrer: true
34                    });
35                });
36            }
```

Figure 11. Authentication for Sign In users.

After typing email and password, these strings need to be read and authenticated, if there is no error then the login session is successful.

4.3 Dashboard

4.3.1 Admin Dashboard

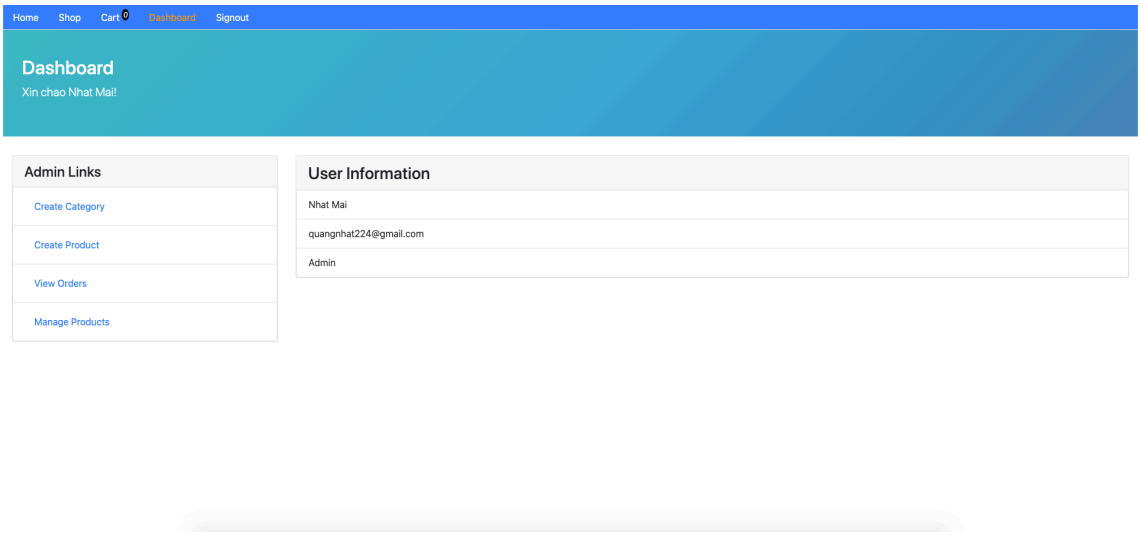


Figure 12. Admin Dashboard screen.

On the Admin Dashboard, the admin can create products, categories and manage them. These functions are very useful for a store owner.

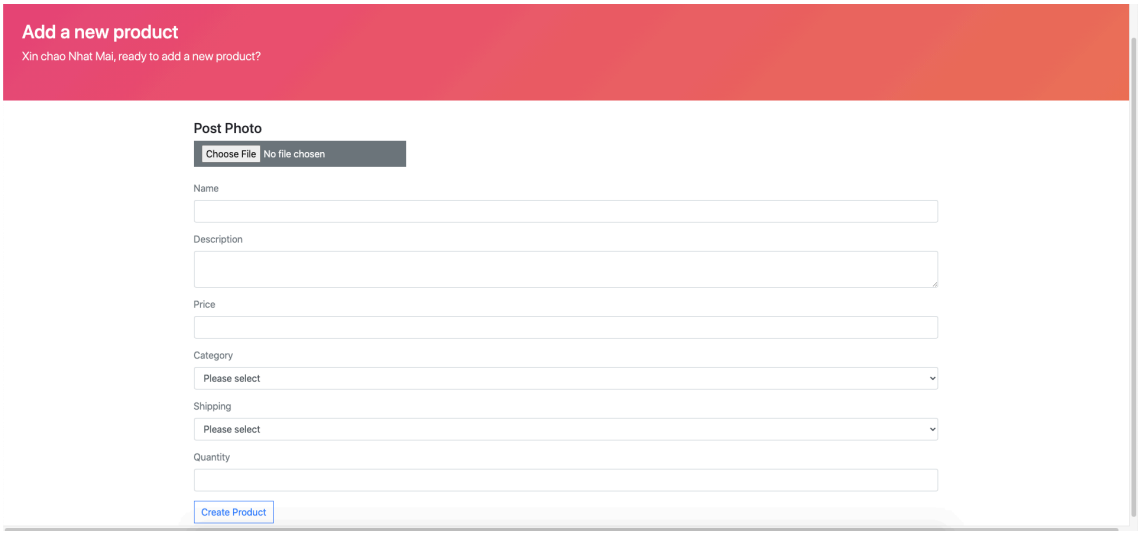
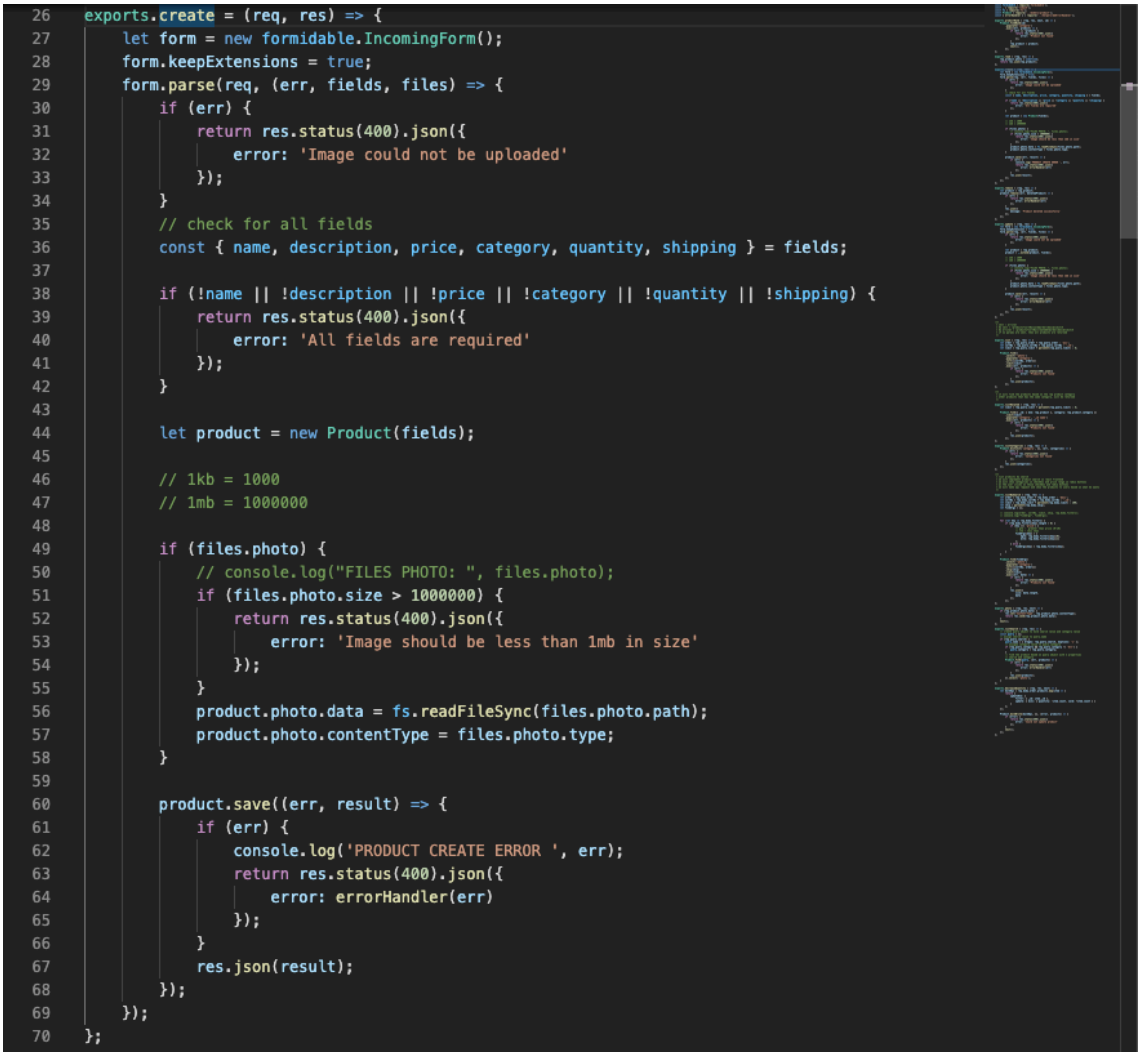


Figure 13. Add a new product screen.

In figure 13, the store owner can add the new products, as they can manage them. Similar with the sign up and sign in, all the strings are stored into MongoDB database.



```
exports.create = (req, res) => {
    let form = new formidable.IncomingForm();
    form.keepExtensions = true;
    form.parse(req, (err, fields, files) => {
        if (err) {
            return res.status(400).json({
                error: 'Image could not be uploaded'
            });
        }
        // check for all fields
        const { name, description, price, category, quantity, shipping } = fields;

        if (!name || !description || !price || !category || !quantity || !shipping) {
            return res.status(400).json({
                error: 'All fields are required'
            });
        }

        let product = new Product(fields);

        // 1kb = 1000
        // 1mb = 1000000

        if (files.photo) {
            // console.log("FILES PHOTO: ", files.photo);
            if (files.photo.size > 1000000) {
                return res.status(400).json({
                    error: 'Image should be less than 1mb in size'
                });
            }
            product.photo.data = fs.readFileSync(files.photo.path);
            product.photo.contentType = files.photo.type;
        }

        product.save((err, result) => {
            if (err) {
                console.log('PRODUCT CREATE ERROR ', err);
                return res.status(400).json({
                    error: errorHandler(err)
                });
            }
            res.json(result);
        });
    });
};
```

Figure 14. JS code for creating new product.

When admin want to create a new product, they have to fill out all the fields and upload a valid image for the products. After that, the new products will be processed to the database.

Moreover, the admin can view the orders they received from the customers. The order status can be updated as well.

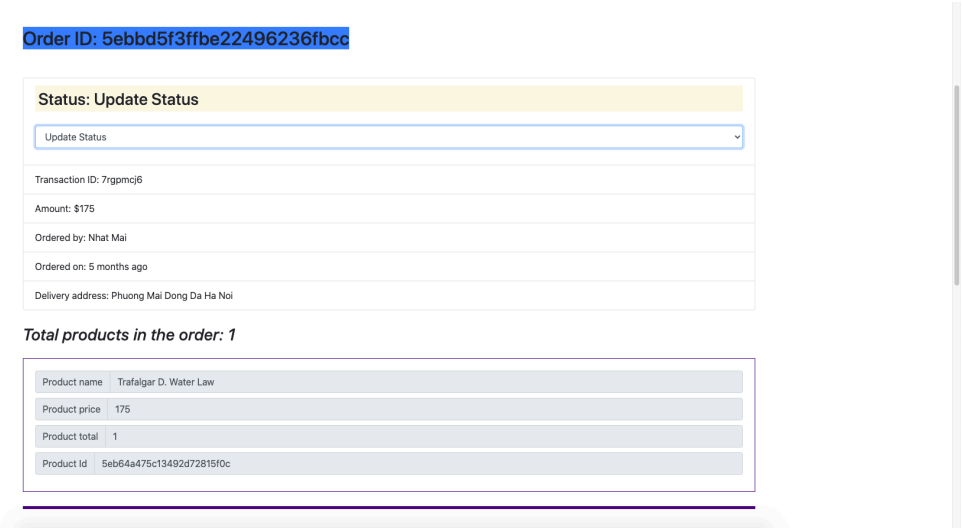


Figure 15. Order details screen.

All of the order details of customers made in history are recorded. In figure 15, admin can see all of the orders then the show owner can set the shipping status for each order individually.

```
75  export const listOrders = (userId, token) => {
76      return fetch(`${API}/order/list/${userId}`, {
77          method: 'GET',
78          headers: {
79              Accept: 'application/json',
80              Authorization: `Bearer ${token}`
81          }
82      })
83          .then(response => {
84              return response.json();
85          })
86          .catch(err => console.log(err));
87  };
```

Figure 16. Getting list of orders.

The figure 16 shows the way how to get the list of orders. The API is fetched then the admin gets the json data back from our database.

The userId have to be admin ID in order to make a request to API to call for history of orders.

4.3.2 User Dashboard

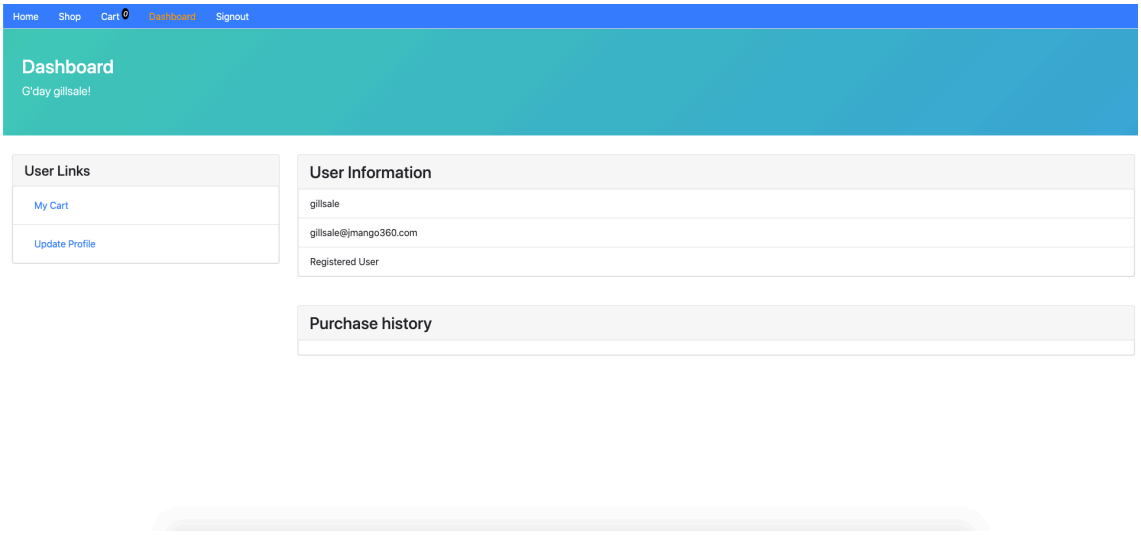


Figure 17. User Dashboard

Beside creating the dashboard for admin, user dashboard is important also. In figure 17, customer can view their cart, purchase history and also update their profile.

```
65  const purchaseHistory = history => {
66      return (
67          <div className="card mb-5">
68              <h3 className="card-header">Purchase history</h3>
69              <ul className="list-group">
70                  <li className="list-group-item">
71                      {history.map((h, i) => {
72                          return (
73                              <div>
74                                  <hr />
75                                  {h.products.map((p, i) => {
76                                      return (
77                                          <div key={i}>
78                                              <h6>Product name: {p.name}</h6>
79                                              <h6>
80                                                  Product price: ${p.price}
81                                              </h6>
82                                              <h6>
83                                                  Purchased date:{" "}
84                                                  {moment(
85                                                      h.createdAt
86                                                  ).fromNow()}
87                                              </h6>
88                                          </div>
89                                      );
```

Figure 18. Purchase history in User Dashboard

In the purchase history of customer, user can see their product name, price and purchased date of their goods which they have bought before by a list of items.

Moreover, in Update profile section, user can make changes of their profile like name, password. It could affect their data in our database.

4.4 Shop Page

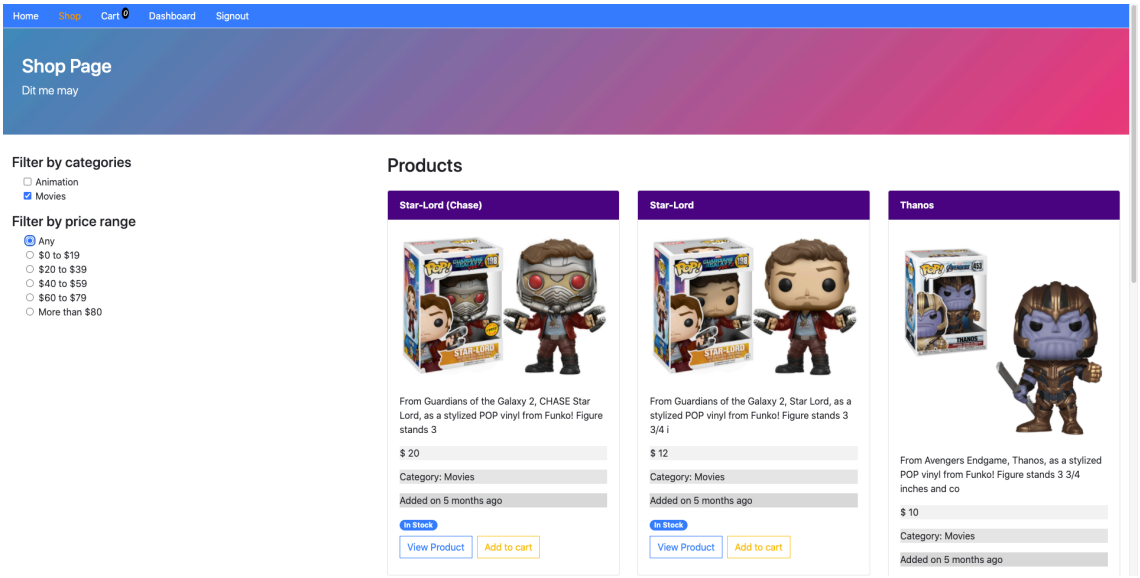


Figure 19. Shop Page

The main shop page that customers can select the products that they want to add to cart. User can scroll down the mouse to browse all the products, or filter by categories or filter by price range like Figure 19.

In each product, user can see it short description and their available status. Therefore, customer can consider about buying it or not.

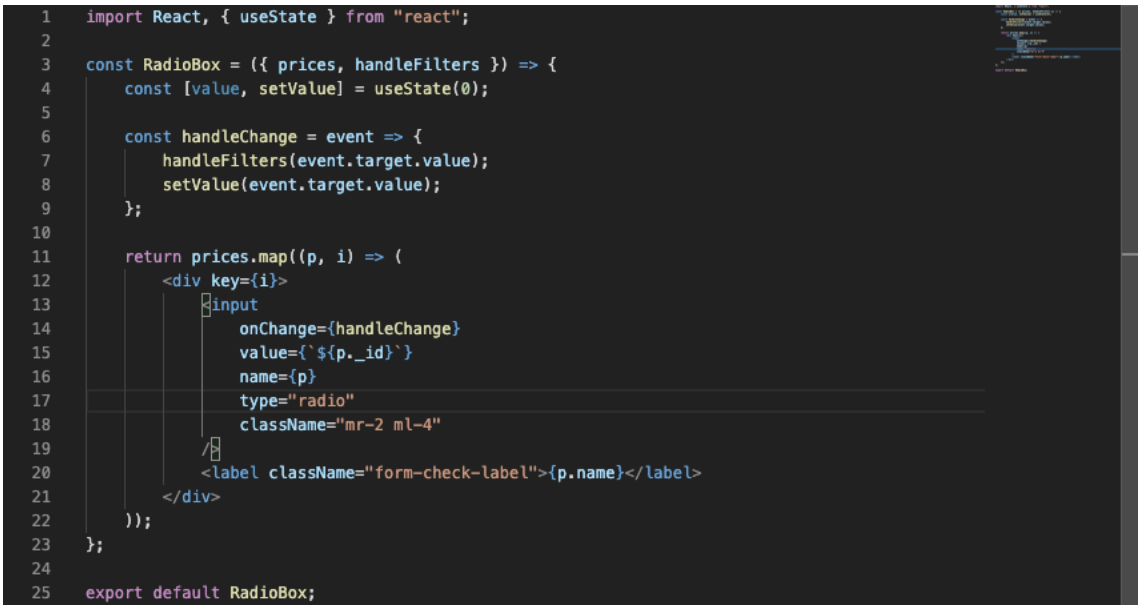
```jsx
import React, { useState } from "react";

const Checkbox = ({ categories, handleFilters }) => {
    const [checked, setCheked] = useState([]);

    const handleToggle = c => () => {
        // return the first index or -1
        const currentCategoryId = checked.indexOf(c);
        const newCheckedCategoryId = [...checked];
        // if currently checked was not already in checked state > push
        // else pull/take off
        if (currentCategoryId === -1) {
            newCheckedCategoryId.push(c);
        } else {
            newCheckedCategoryId.splice(currentCategoryId, 1);
        }
        // console.log(newCheckedCategoryId);
        setCheked(newCheckedCategoryId);
        handleFilters(newCheckedCategoryId);
    };

    return categories.map((c, i) => (
        <li key={i} className="list-unstyled">
            <input
                onChange={handleToggle(c._id)}
                value={checked.indexOf(c._id === -1)}
                type="checkbox"
                className="form-check-input"
            />
            <label className="form-check-label">{c.name}</label>
        </li>
    ));
};

export default Checkbox;
```

Figure 20. Filter by categories

When one or more categories need to be checked, handleToggle method is used like Figure 20. Therefore, all the categories can put in an array and send to the backend to get all the perks based on those categories.

This indexOf method will return the first index at which a given element can be found in the array. If it is not found in the state, then that will return -1.

newCheckedCategoryId give all the category that is in the state and currentCategoryId will tell us if it is already there.



```
import React, { useState } from "react";

const RadioBox = ({ prices, handleFilters }) => {
    const [value, setValue] = useState(0);

    const handleChange = event => {
        handleFilters(event.target.value);
        setValue(event.target.value);
    };

    return prices.map((p, i) => (
        <div key={i}>
            <input
                onChange={handleChange}
                value={`${p._id}`}
                name={p}
                type="radio"
                className="mr-2 ml-4"
            />
            <label className="form-check-label">{p.name}</label>
        </div>
    ));
};

export default RadioBox;
```

Figure 21. Filter by price range.

Implementation of handleChange on the radio boxes filter. First of all the event.target.value must be sent to the handleFilters. It is coming from the parent subcomponent in Shop page. Therefore, any time when user does something onChange on this input it will be sent to the parent component.

Various categories can be selected at the same time but with the price range only can select either one of them.

In Figure 22, it describes the product details on the left. Beside on the right side there is related products which have the same category with the product user clicked on it.

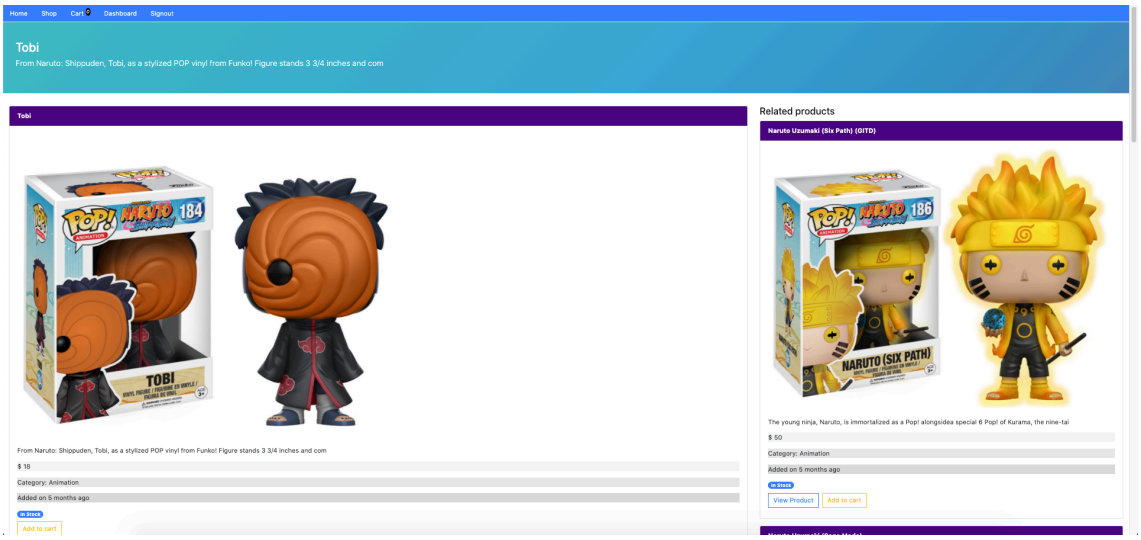


Figure 22. Product details.

4.5 Cart Page

Customers select all of the products they want to buy and add to the cart. All of these products will move to Cart. In Figure 23, there is a number 2 on the Cart Button on the navigation bar which means there are 2 items in Cart that have not paid yet.

On the left side there is list of products that customer added to cart. On the right is the delivery address field that required user to fill in. The total price of the products.

Below that is the payment gateway. The payment gateway has 2 selection for customers which are Card or PayPal.

In each product listed in Cart, user can remove them from the cart also if they do not want to buy it anymore.

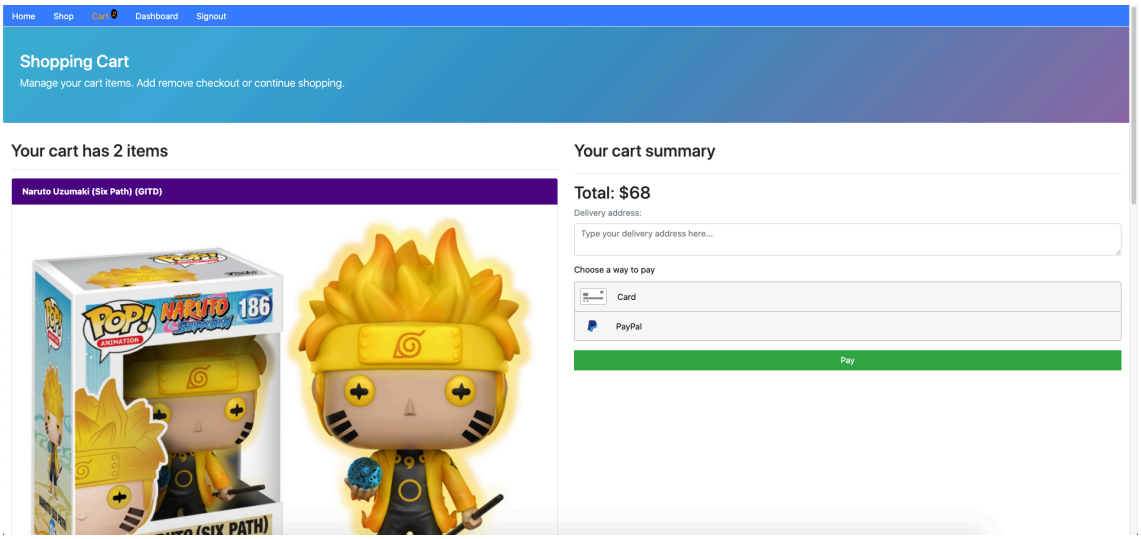


Figure 23. Cart Page.

4.5.1 Cart CRUD

When the user clicks Add to Cart button, the product must be saved to the local storage. Based on that, total products in cart can be calculated.

```
export const addItem = (item = [], count = 0, next = f => f) => {
    let cart = [];
    if (typeof window !== 'undefined') {
        if (localStorage.getItem('cart')) {
            cart = JSON.parse(localStorage.getItem('cart'));
        }
        cart.push({
            ...item,
            count: 1
        });

        cart = Array.from(new Set(cart.map(p => p._id))).map(id => {
            return cart.find(p => p._id === id);
        });

        localStorage.setItem('cart', JSON.stringify(cart));
        next();
    }
};
```

Figure 24. Add items to Cart.

When user clicked on the product twice or more than that, this function only updates the quantity of the product instead of actually duplicating the product.

The Array.from method to create a new array that will make sure that there is no duplicate. Build an Array from new Set and turn it back into array using Array.from so that later we can re-map it.

```
export const getCart = () => {
    if (typeof window !== 'undefined') {
        if (localStorage.getItem('cart')) {
            return JSON.parse(localStorage.getItem('cart'));
        }
    }
    return [];
};
```

Figure 25. Get Cart Items.

The getCart method will give the user all the items from the cart. This method can be exported to be used in menu.

```
export const updateItem = (productId, count) => {
    let cart = [];
    if (typeof window !== 'undefined') {
        if (localStorage.getItem('cart')) {
            cart = JSON.parse(localStorage.getItem('cart'));
        }

        cart.map((product, i) => {
            if (product._id === productId) {
                cart[i].count = count;
            }
        });

        localStorage.setItem('cart', JSON.stringify(cart));
    }
};
```

Figure 26. Update items in Cart.

In Figure 26 is the updateItem method. All of the items are already in the local storage with the name of “cart” would be put in the cart variable. After that each of them will be map through and try to match with the productId.

```
export const removeItem = productId => {
    let cart = [];
    if (typeof window !== 'undefined') {
        if (localStorage.getItem('cart')) {
            cart = JSON.parse(localStorage.getItem('cart'));
        }

        cart.map((product, i) => {
            if (product._id === productId) {
                cart.splice(i, 1);
            }
        });

        localStorage.setItem('cart', JSON.stringify(cart));
    }
    return cart;
};
```

Figure 27. Remove items in Cart.

The removeItem function is implemented to remove the product from the cart. This method is similar to update item. The splice method is used to take one out from the index. This method takes two argument. The first Is where to splice, the second is how many to splice. In Figure 27 only one item has to take out from the cart.

4.5.2 Payment Gateway

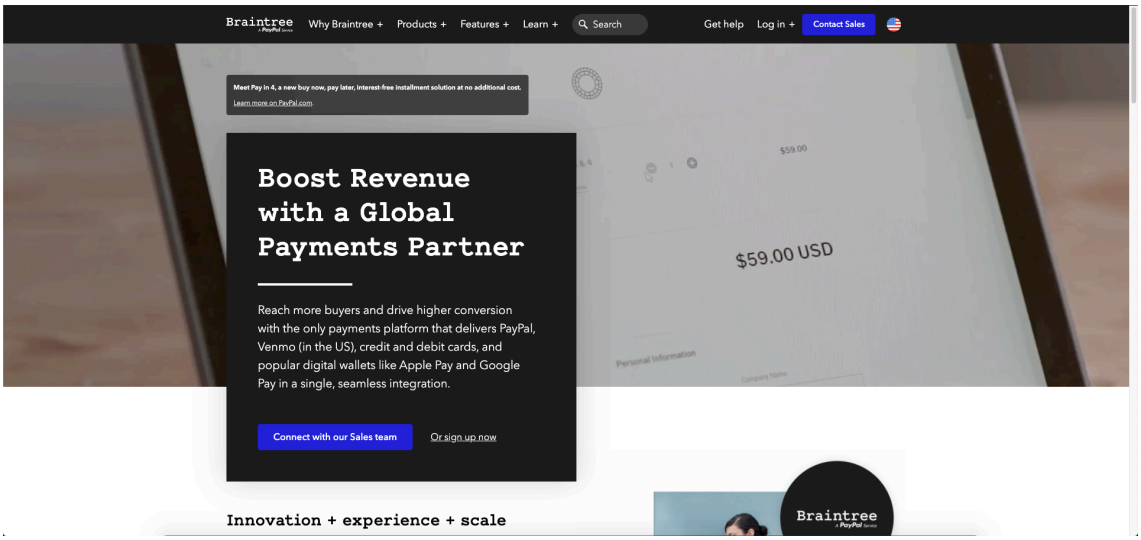


Figure 28. Braintree Homepage.

The most important feature of any e-commerce application is payment gateway. Braintree is one of the best payment processing option available in the market. If the store only wants to use the credit card system, then it could be losing a lot of customers

because many people nowadays want to use PayPal. Therefore, the application integrates both credit card and PayPal system. Braintree is used by some of the biggest name in the world such as Google, Dropbox, GitHub.

```
.env
1   PORT=8000
2   DATABASE=mongodb+srv://quangnhat224:vietanh216@nodeapi-rudc3.mongodb.net/test?retryWrites=true&w=
3   JWT_SECRET=dasjifjasfajfajkflasfljka
4   BRAINTREE_MERCHANT_ID=dtp5x43pwk3sh582
5   BRAINTREE_PUBLIC_KEY=yddwnrpf5v9fvz2k
6   BRAINTREE_PRIVATE_KEY=69fb66f364603169209dd6103393ffcb
```

Figure 29. Braintree token.

Braintree have to be installed individually in node API. Then the token in figure 29 can be requested in backend and it can be given to frontend. The .env file is required privately for the usage of environment variables.

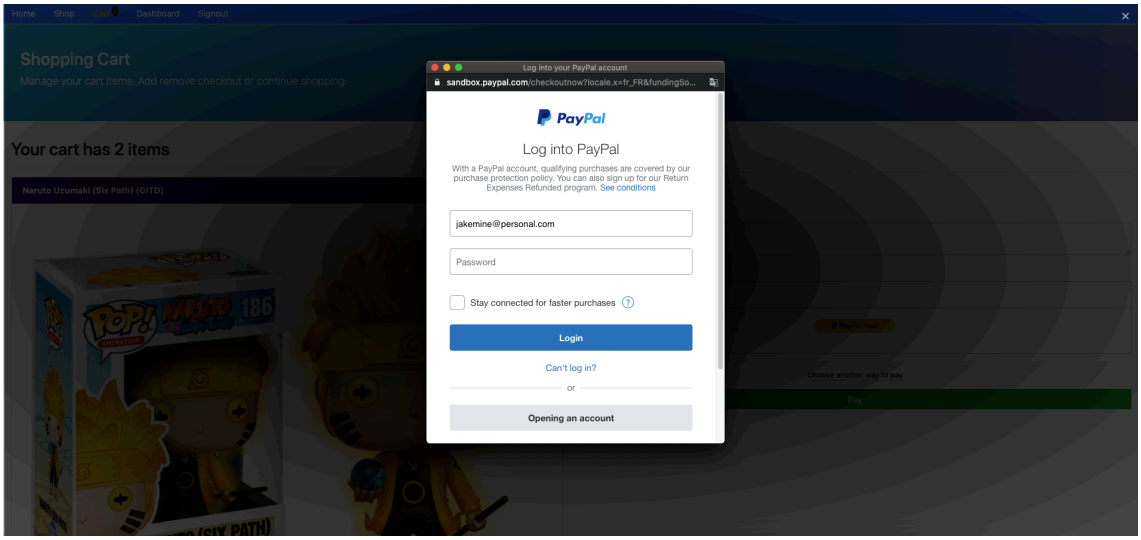


Figure 30. PayPal login screen.

After the token has be sent to frontend. Customers can login to their PayPal account and pay for the order they have made before. In Figure 30, the sandbox account can be created to test the PayPal function.

After paying for the order by PayPal or credit cards, users just finished their shopping and go back to the Home page.

5 Summary

The achievement of the thesis is researching the basic components of MERN stack technology: MongoDB, ExpressJS framework, ReactJS library and NodeJS platform. Using MERN stack technology in conjunction with Braintree to build an e-commerce web application with payment gateway.

The advantages are performing the basic functions of a product search website for customers, making it easy for customers to find categories that have the products they are looking for. Gives small stores a platform to store information and promote their products. Password data of accounts when logging in to the system is stored in a secure database. The management interface, statistics of the user and admin are easy to use for everyone.

The disadvantages are online chat functions between shop owners and customers are not yet supported as well as between shop owners and administrators. The current product search algorithm locates by the user location that is not really optimal, needs to be improved to speed up the search even more.

Since the purpose of the thesis is the e-commerce application, the understanding about MERN technologies and applying it to this app is the most important. Overcome current shortcomings, listen to customers' comments and making improvements, helping users have a great experience in the future.

References

1. E-commerce Definition – What is E-commerce? [Internet]. Shopify.com. Available from: https://www.shopify.com/encyclopedia/what-is-ecommerce

2. Advantages of E-commerce [Internet]. Thebalancesmb.com 2019 [cited 20 November 2019]. Available from: https://www.thebalancesmb.com/ecommerce-pros-and-cons-1141609

3. JavaScript [Internet]. Mozilla.org. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript

4. NodeJS Introduction [Internet]. Tutorialspoint.com. Available from: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

5. NodeJS Pros and Cons [Internet]. Mindinventory.com. Available from: https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/

6. NodeJS use cases [Internet]. Credencys.com. Available from: https://www.credencys.com/blog/node-js-development-use-cases/

7. Express.js Introduction [Internet]. Mozilla.org. Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

8. MongoDB [Internet]. Mongodb.com. Available from: https://docs.mongodb.com/manual/introduction/

9. Virtual-DOM [Internet]. Reactjs.org. Available from: https://reactjs.org/docs/faq-internals.html

10. Component [Internet]. Reactjs.org. Available from: https://reactjs.org/docs/components-and-props.html

11. Props and State [Internet]. Flaviocopes.com. Available from: https://reactjs.org/docs/components-and-props.html

12. Pros and Cons of ReactJS [Internet]. Javatpoint.com. Available from: https://www.javatpoint.com/pros-and-cons-of-react

13. Stack technology [Internet]. Stackshare.io. Available from: https://stackshare.io/stacks

14. MERN stack concept [Internet]. Mongodb.com. Available from: https://www.mongodb.com/mern-stack