



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Phuc Le

# Development of an eCommerce web- site for Ngoc's MaxiNutri Company

School of Technology  
2022

## ABSTRACT

Author	Phuc Le
Title	Development of an eCommerce Website for Ngoc's MaxiNutri Company.
Year	2022
Language	English
Pages	44
Name of Supervisor	Mikael Jakas

---

Technology has evolved rapidly in recent years, especially in web development and technology. Many new tools and technologies have enhanced developers' experience, and building a website now is more popular and much easier than it used to be. In 2013, Facebook announced React, and it has become a part of the MERN stack (MongoDB, ExpressJS, React, NodeJS) – one of the most popular stacks to build a web application in the world.

The thesis aimed to learn the concepts and functionalities of the MERN stack and implement it with an online shop application for Ngoc's MaxiNutri company. Each technology will be explained in detail, along with the assistive packages and libraries to build the application, for example, Bootstrap 5, JWT, React Route DOM v6, and Redux.

The development process was unified, and all the parts of the web application were written in JavaScript. The user interface was created with React, and the database will be using an object-based NoSQL database MongoDB and interact with the server using NodeJS and ExpressJS.

The result of the thesis is a fully functional eCommerce web application for Ngoc's MaxiNutri.

---

Keywords <sup>1</sup>	MERN, React, Redux, NodeJS, ExpressJS, MongoDB, JWT, e-commerce, online store
-----------------------	---

---

## **ACKNOWLEDGMENTS**

I would like to express my gratitude to the people who assisted and supported me in completing this thesis which is also a milestone.

Firstly, I would like to thank my supervisor, Mr. Mikael Jakas, who provided me with feedback to complete my project.

Secondly, I would also like to thank my parent, the owner of Ngoc's Max-iNutri, for assisting me in providing information about the company and what should be implemented and feedback.

Finally, I would like to thank my friends for assisting me in providing feedback on the demo of my application and supporting me with my questions about the technologies I used in this application.

Tampere, May 21, 2022

Phuc Le

# CONTENTS

## ABSTRACT

ACKNOWLEDGMENTS.....	3
1 INTRODUCTION .....	8
2 FINAL PRODUCT DEMONSTRATION.....	10
3 THEORETICAL BACKGROUND .....	14
3.1 MERN.....	14
3.2 Front-end .....	14
3.2.1 ReactJS.....	15
3.2.2 Bootstrap .....	16
3.3 Back-end.....	17
3.3.1 NodeJS .....	17
3.3.2 ExpressJS .....	18
3.4 Database .....	19
3.4.1 MongoDB.....	19
3.4.2 Mongoose.....	20
4 PROJECT IMPLEMENTATION.....	21
4.1 Development Environment Setup.....	22
4.1.1 Version Control.....	22
4.1.2 Node Package and Framework Installation.....	23
4.2 Application Logic .....	25
4.2.1 Front-end Logic.....	27
4.2.2 Back-end Logic .....	32
4.2.3 Database Implementation.....	37
4.3 Application Deployment .....	40
5 CONCLUSIONS.....	42
REFERENCES .....	43

## LIST OF FIGURES AND TABLES

Figure 1 Final product - homepage route.....	10
Figure 2 Final product - single product route. ....	11
Figure 3 Final product - cart route.....	11
Figure 4 Final product - user profile route .....	12
Figure 5 Final product - admin tool route example.....	12
Figure 6 Final product - Login and Register route.....	13
Figure 7 Full-stack MERN architecture. ....	14
Figure 8 Angular vs. React vs. Vue download trends. ....	15
Figure 9 ReactJS component example using JSX and Hooks. ....	16
Figure 10 Using Bootstrap 5 inside React component. ....	17
Figure 11 Simple NodeJS server. ....	18
Figure 12 Simple ExpressJS server.....	19
Figure 13 MongoDB compass user interface.....	20
Figure 14 Object mapping between NodeJS and MongoDB via Mongoose. .....	20
Figure 15 The user architecture of the project. ....	21
Figure 16 The admin architecture of the project.....	21
Figure 17 Git workflow. ....	23
Figure 18 Structure of the application. ....	26
Figure 19 Reducer for login. ....	27
Figure 20 Action for the login. ....	28
Figure 21 Submit handler where login function is called. ....	28
Figure 22 Reducer for Add to Cart.....	29
Figure 23 Action for Add to Cart.....	29
Figure 24 Main index.js file inside front-end folder.....	30
Figure 25 The use of Bootstrap in the Footer component.....	31
Figure 26 The use of Bootstrap in the ProductCarousel component.....	31
Figure 27 Token authentication by JWT .....	34
Figure 28 Admin user authentication. ....	35
Figure 29 Error handler function. ....	35
Figure 30 Using Postman to test the GET /api/products route. ....	36
Figure 31 Application's user model. ....	37

Figure 32 Example of a user object in MongoDB.....	38
Figure 33 Connection to a database. ....	38
Figure 34 Initial user data. ....	39
Figure 35 Function used to create the initial data. ....	40
Figure 36 Project UI inside Heroku. ....	41
Table 1 List of all routes made in the project. ....	32

**LIST OF ABBREVIATIONS**

<b>API</b>	Application program interface
<b>CSS</b>	Cascading Style Sheets
<b>HTML</b>	Hypertext Markup Language
<b>JSON</b>	JavaScript Object Notation
<b>HTTP</b>	Hypertext Transfer Protocol
<b>URL</b>	Uniform Resource Locator
<b>JSX</b>	JavaScript XML
<b>XML</b>	Extensive Markup Language
<b>MERN</b>	MongoDB, ExpressJS, ReactJS, NodeJS
<b>SEO</b>	Search Engine Optimization
<b>VDOM</b>	Virtual DOM
<b>SPA</b>	Single Page Application
<b>JWT</b>	JSON Web Token

# 1 INTRODUCTION

The name of this thesis is “Develop an eCommerce Website for Ngoc’s MaxiNutri Company.” Ngoc’s MaxiNutri was established in 2012 with only a single and exclusive product line: “Shake Yogurt.” After five years of establishment and development, the establishment has developed other nutritious milk lines such as corn milk, fresh cow’s milk, lotus seed milk, aloe vera ginseng, and bubble milk tea. With the production motto: “Vì sức khỏe người tiêu dùng (For the health of consumers),” the conscience and responsibility of the food producer, and the highly closed production machinery system, Ngoc’s MaxiNutri chooses the freshest input materials and fully legal on food safety to produce a delicious and nutritious bottle of drinks to serve customers.

Ngoc’s MaxiNutri always wants and is pleased to serve their customers safe and clean milk. They always believe that with the conscience and enthusiasm of the establishment owner and staff, along with a closed machine system, Ngoc’s MaxiNutri will bring the highest value of nutritious dairy products to meet the need and beliefs of the customers. Now, when the customer demand increases rapidly, and Ngoc’s MaxiNutri products have been distributed across Viet Nam, they need to expand their business model, reaching new customers by selling online. This project aims to develop an online store for the company.

The objectives of the thesis were to explain and demonstrate the concept of the MERN stack and build an application using the MERN stack, which can be used in real life. The thesis is divided into six parts; at the beginning of the idea, the project’s outcome and functions are shown, then the leading technologies used in the project are discussed in depth. After that, the application requirements and project setup and implementation are discussed, and finally, the conclusion.

A fully functional and ready-to-use online shop for Ngoc’s MaxiNutri was built and deployed via Heroku. The thesis can be used as a tutorial about



the MERN stack application, targeting people who want to learn more about MERN.

## 2 FINAL PRODUCT DEMONSTRATION

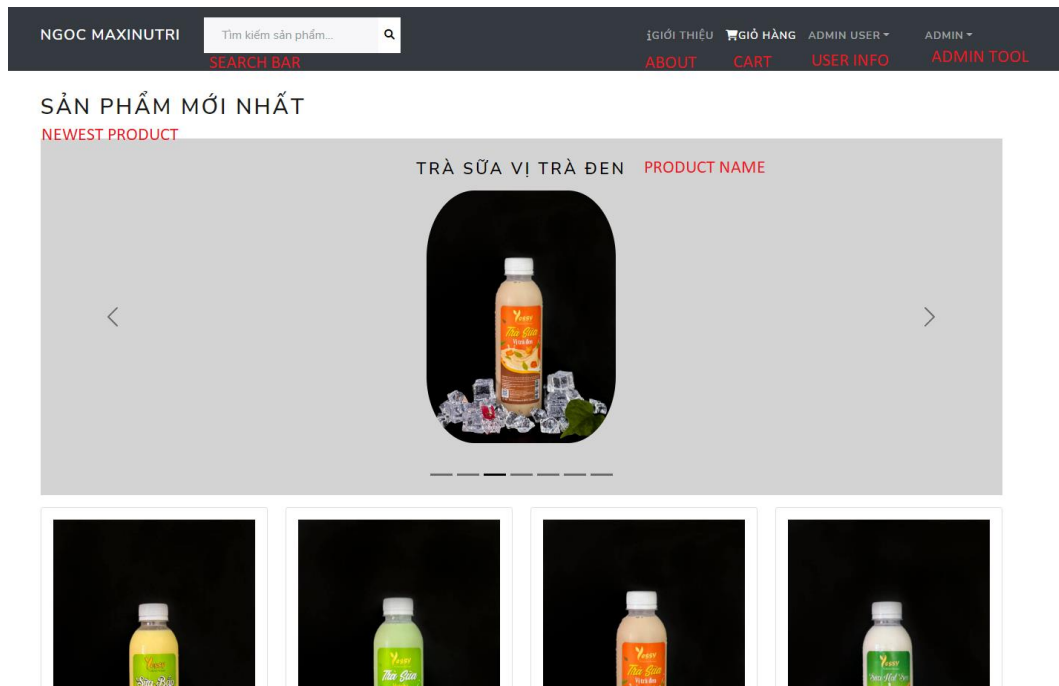


Figure 1 Final product - homepage route.

Figure 1 shows the website's homepage; as we can see, the website is in Vietnamese because, for the moment, the products are only sold in Vietnam. Therefore, the English translations will be provided.

In this web application, users can use the following functions:

- See the information for all the products. (Figure 1)
- See the information for a specific product. (Figure 2)
- Search function.
- Add their favorite products to the cart. (Figure 3)
- Pay for the product.
- See the order history and change the information. (Figure 4)
- Receive the delivery status. (Figure 4)
- Log in, log out, and register as a new user. (Figure 6)

The admin tool was implemented, and the admin has the same right as the average user, in addition, the admin can:

- Create, edit, and delete the product. (Figure 5)

- See, edit, and delete the user's information (except the user's password).
- Update the delivery progress

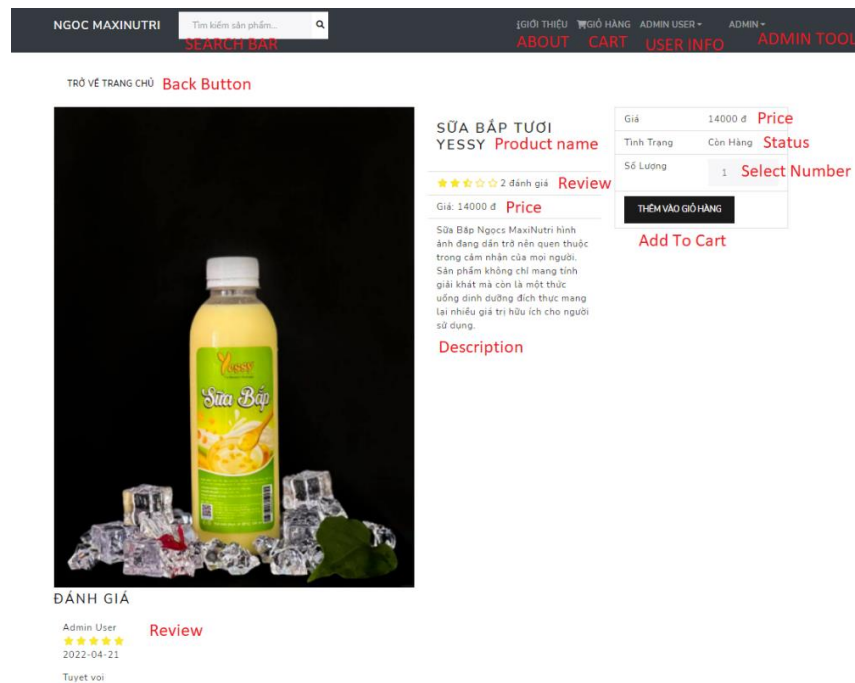


Figure 2 Final product - single product route.

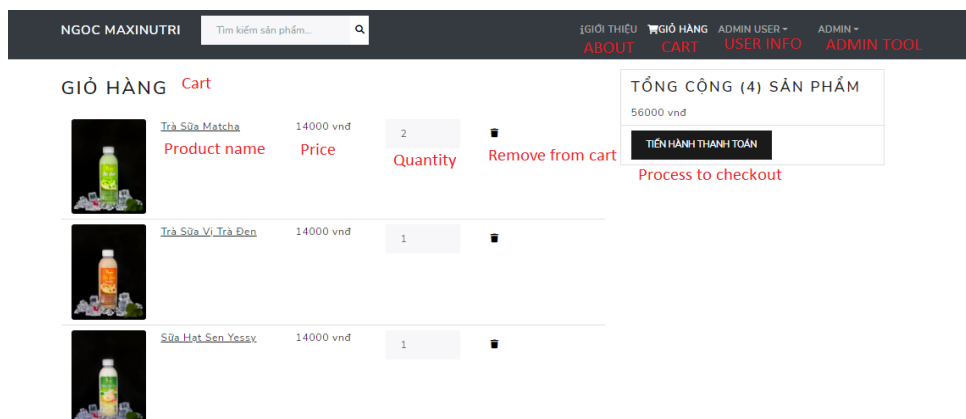


Figure 3 Final product - cart route.



### ĐĂNG NHẬP

Địa chỉ Email

Mật Khẩu

[ĐĂNG NHẬP](#)

Người Dùng Mới? [Đăng Kí](#)

Login Form

### ĐĂNG KÍ

Tên người dùng

Địa chỉ Email

Mật Khẩu

Xác Nhận Mật Khẩu

[ĐĂNG KÍ](#)

Đã có tài khoản? [Đăng Nhập](#)

Register Form

Figure 6 Final product - Login and Register route.

Figures 2 to figure 6 shows the most web application functions. The application using MERN stack JavaScript was developed successfully. React and React Bootstrap was used for the front end and styling, NodeJS and ExpressJS were used to create the back end, and MongoDB was used to store the database.

The prototype built is stable, but before being deployed in the production mode, there are still some areas for improvement that need to be tested more by quality assurance. Moreover, new features such as English support, logging in with Gmail and Facebook, and setting more helpful information could be implemented. If appropriately developed, Ngoc's MaxiNutri products will be more accessible by boosting sales.

### 3 THEORETICAL BACKGROUND

In computer science, the term 'stack' is an abstract data type that serves as a collection of elements. In web development, 'stack' refers to a combination of different technologies related to Front-end Development, Back-end Development, Database, Cloud, and so on. There are many web stacks globally, and the MERN stack is one of the most popular stack options. Therefore, the MERN stack has as many supports as possible in this thesis. In this chapter, the MERN stack will be described and explained why it was used in this project. Furthermore, other technologies that help us handle the application's state and authentication are discussed and examined. /1/

#### 3.1 MERN

MERN stack is a JavaScript stack used for easier and faster deployment of the full-stack web application. MERN stands for MongoDB, ExpressJS, ReactJS and NodeJS. Inside this stack, ExpressJS and NodeJS will make up the middle tier. ReactJS will handle the front-end (The first layer), while MongoDB takes the database and storage. Figure 7 below shows the architecture of a Full-stack MERN web application. /2/

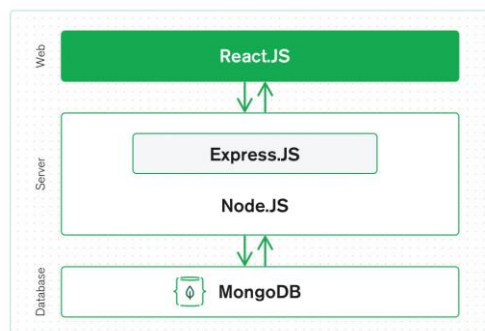


Figure 7 Full-stack MERN architecture. /2/

#### 3.2 Front-end

The front-end or client-side development is the feature or UI that users can directly interact with. In this project, several front-end technologies were used and are explained below.

### 3.2.1 ReactJS

React is one of the most well-known open-source, efficient, component-based, easy-to-use JavaScript libraries for building user interfaces. Facebook created it on May 29, 2013, and it is maintained by Meta and a community of individual developers and companies.

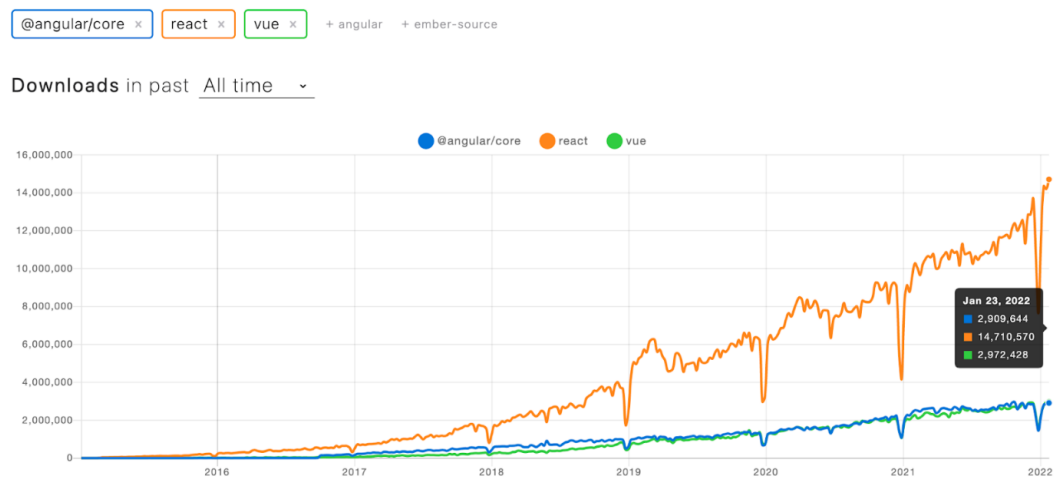


Figure 8 Angular vs. React vs. Vue download trends. /3/

As Figure 8 shows, the number of developers using React is almost four times that of developers using Vue and Angular.

ReactJS is a component-based JavaScript library. These components must be reusable and formed in the 'src' folder with the Camel Case naming convention. React DOM library can render a particular element in the DOM. The component element, value, and function can be passed through 'props.' Because of this, ReactJS only rerenders the changes that have been made in the component, but not the whole page, which is better for the performance.

Since ReactJS 16.8, 'Hooks' was implemented. Hooks let developers manage the React state and lifecycle features from function components. State management is now much easier with this new feature because the user has a simpler code that implements similar functionalities faster and more effectively.

ReactJS also uses JSX, or JavaScript Syntax Extension, to extend the JavaScript syntax. JSX is a combination of HTML and JavaScript. In JSX, we

can structure component rendering using HTML. React was also used to reuse all the components in the project. Figure 9 below shows an example of a ReactJS part with Hooks using JSX. /3/

```
1  import React, { useState } from 'react';
2
3  const TestState = () => {
4    const [test, setTest] = useState('test');
5    return <div>{test}</div>;
6  };
7
8  export default TestState;
```

Figure 9 ReactJS component example using JSX and Hooks.

### 3.2.2 Bootstrap

Bootstrap is a free and open-source CSS framework directed at front-end web development, responsive and mobile-first. Bootstrap contains HTML, CSS, and JavaScript-based design templates for many features, such as Buttons, Form, Navigation Bar, Background, and so on. Bootstrap 5 was used with Bootwatch in this project to make the process more convenient and reduce the amount of time spent on styling. /4/ Figure 10 below shows an example of using React Bootstrap inside React; the example shows the creation of a pre-styled button without any separate CSS file; as we see, the syntax and the styling are handy with this styling method.



```
1  import React from 'react'
2  import { Button } from "react-bootstrap"
3
4  const TestBootstrap = () => {
5    return (
6      <Button variant="primary">Click Me</Button>
7    )
8  }
9
10 export default TestBootstrap
```

Figure 10 Using Bootstrap 5 inside React component.

### 3.3 Back-end

Back-end development or server-side development focuses on the database, website architecture, logic, and everything that happens behind the screen. It can be purchased from the store, log in, log out methods, and so on. Back-end development also helps communicate with a database, in our case MongoDB.

#### 3.3.1 NodeJS

NodeJS is the most popular open-source and cross-platform JavaScript runtime environment that can execute JavaScript code outside the web browser.

NodeJS can help developers create a dynamic website by writing command-line tools and server-side scripting before sending the page to the user's web browser. By NodeJS, the term "JavaScript everywhere" is represented since the developer can write front-end code, back-end code, and even databases with one programming language only: JavaScript, making the development process more efficient and handier.

Ryan Dahl created NodeJS in 2009, about thirteen years after introducing the first server-side JavaScript environment. Since then, NodeJS has earned a good reputation in the tech industry. It plays a significant role in the web development stack, and many companies have been using it as an

essential part, such as Netflix, Nasa, Trello, PayPal, LinkedIn, and so on. /5/

NodeJS also provides tools such as NPM that help manage third-party libraries such as Mongoose, Express, and JWT. To make the development much faster and more efficient. Figure 11 below shows an example of a NodeJS web server.

```
1  const http = require("http");
2
3  const host = 'localhost';
4  const port = 8000;
5
6  const requestListener = function (req, res) {};
7
8  const server = http.createServer(requestListener);
9  server.listen(port, host, () => {
10    console.log(`Server is running on http://${host}:${port}`);
11  });
```

Figure 11 Simple NodeJS server.

### 3.3.2 ExpressJS

ExpressJS represents the “E” part of the MERN stack and is essential. Express is a minimal and flexible NodeJS web application framework that boosts the features of the single-page application, websites, hybrids, or public HTTP APIs. Since NodeJS is prevalent, many libraries have been developed to ease server-side development with NodeJS like Socket.io, Koa.js, Meteor.js, Nest.js, and so on. Nevertheless, ExpressJS is the most popular one for this purpose so far.

With the help of ExpressJS, an application can be built much faster than pure NodeJS. ExpressJS makes routing for requests made by clients with the GET, POST, PUT, DELETE method, and middleware-like authentication simpler. Figure 12 below is an example of a simple server built with express used to show ‘Hello World’ using ES6 syntax. /6/

```

1  import express from 'express';
2  const app = express();
3  const port = 3000;
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!');
7  });
8
9  app.listen(port, () => {
10   console.log(`Example app listening on port ${port}`);
11 });
12

```

Figure 12 Simple ExpressJS server.

In this thesis, ExpressJS developed different functions such as getting product and user information, registering new users, logging in, and creating a new order.

### 3.4 Database

“A database is an organized collection of structured information or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS and the associated applications are referred to as a database system, often shortened to the just database.” /7/

#### 3.4.1 MongoDB

MongoDB represents the “M” part of the MERN stack and stores the database for this thesis project. MongoDB is a document-oriented database program classified as a NoSQL database program. In MongoDB, JSON-like documents with schemas are used. In this thesis, MongoDB compass was used because with this application; we can access our database much faster without having to log in to the web application. Figure 13 below shows the interface of MongoDB compass; we can see that MongoDB auto-generates the ‘\_id’ for each child of the view; they also provide methods like ‘find,’ ‘findById,’ ‘save’, which is handy and make it easier to communicate with the database. /8/

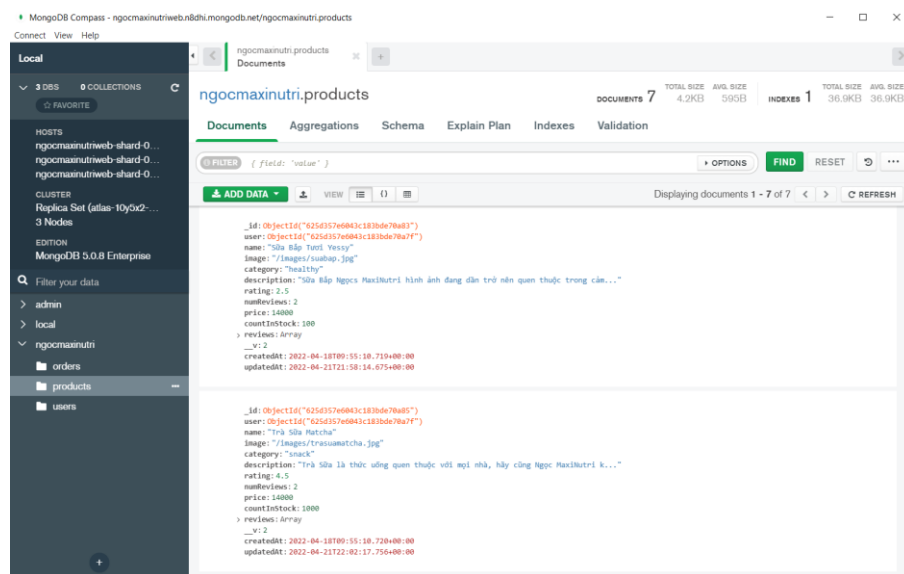


Figure 13 MongoDB compass user interface.

### 3.4.2 Mongoose

Mongoose is a JavaScript object data modeling library for MongoDB and NodeJS. It manages relationships between data and provides schema validation and representation between objects in MongoDB /9/. Figure 14 below shows the object mapping between NodeJS and MongoDB via Mongoose.

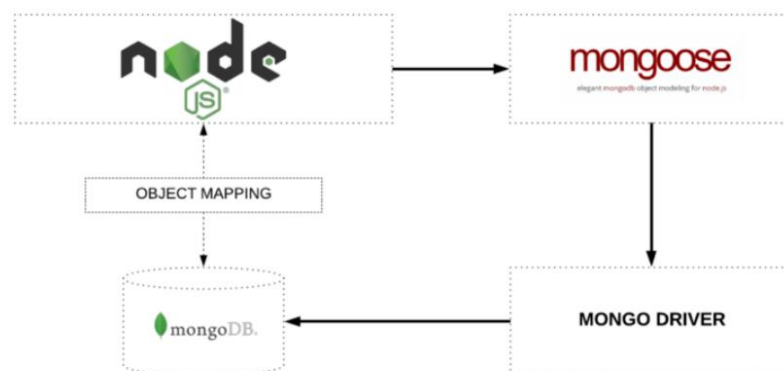


Figure 14 Object mapping between NodeJS and MongoDB via Mongoose.

## 4 PROJECT IMPLEMENTATION

Ngoc's MaxiNutri eCommerce web application uses the MERN stack that users can use to buy products from Ngoc's MaxiNutri online. Figure 15 below shows the average user architecture of the project.

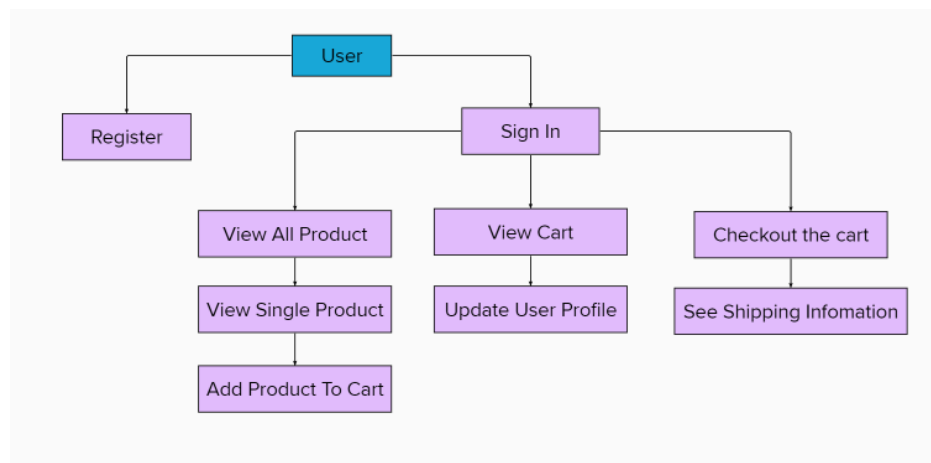


Figure 15 The user architecture of the project.

As described in Figure 15, several functions need to be implemented. There is a dashboard where users must sign up to use all the services. Otherwise, they can only view all the products and a specific product. When a user logs in to the application, they can perform features such as view their profile, update their name, email, and password, and see all their previous orders and the shipping status. By creating an account, users can also review the product in the store. The architecture of the admin account is shown in Figure 16 below.

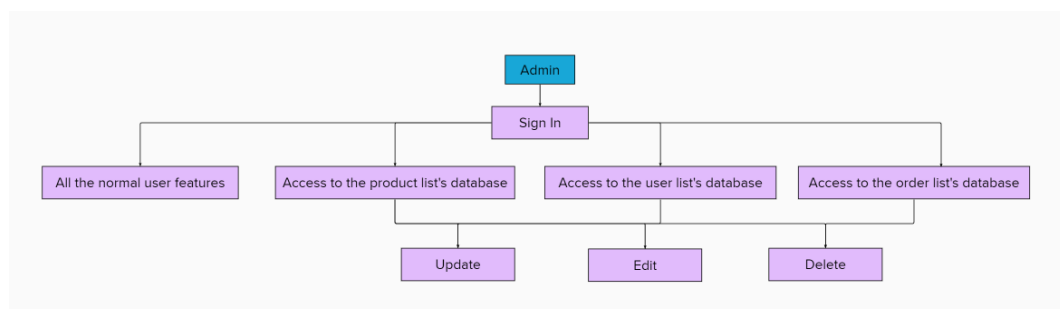


Figure 16 The admin architecture of the project.

## 4.1 Development Environment Setup

Choosing the right tools and a suitable environment is always important when dealing with big projects. A good combination between the code editor and other tools can increase work efficiency and productivity and save time. Therefore, choosing good tools is one of the keys to making the work easier.

In this project, Visual Studio Code was used as a code editor. Visual Studio Code (or VS Code) is a lightweight source code editor made by Microsoft for Windows, Linux, and macOS. VS Code supports many features for debugging, syntax highlighting, code refactor, Git, and auto-complete code, making the coding faster and much more accessible. /10/

The VS code's extensions that were helpful for this project are:

- 'ES7+ React/Redux/React-Native snippets' for auto-complete code and Snippet for React JavaScript.
- 'ES7+ React/Redux/React-Native snippets' for formatting the code to my chosen format.
- 'Auto Rename Tag' for automatically renaming paired XML tags.
- 'Bracket Pair Colorizer' for colorizing matching brackets.
- 'Material Icon Theme' for files/folders icon in Visual Studio Code.

### 4.1.1 Version Control

Version Control or Source Control is the tracking and managing of changes to software code. Furthermore, the Version Control System is the programming tool that helps software teams manage the source code changes over time. In the Version Control, differences are identified by a 'code,' which is the 'version number' of the source code. For example, an initial setup for a project is the 'version number 1', and when the project's contents change, the version control tool will determine that and set it as 'version number 2'. With Version Control Tool, users can compare different versions, drawbacks from an old version, or let multiple developers work on the same project. Many other version control systems such as Git, Apache, Azure DevOps,

and so on. In this project, Git was used because it is one of the most popular version control systems globally and is user-friendly. /11/

With Git, developers will work on a repository clone stored in the cloud. After doing some work, the developer will create a commit and push that commit to the cloud, creating another version of the original file. The Git Cloud that we used in this project is GitLab because, among Bitbuckets and GitLab, it is one of the most considerable Git Clouds in the world. Figure 17 demonstrates how Git works.

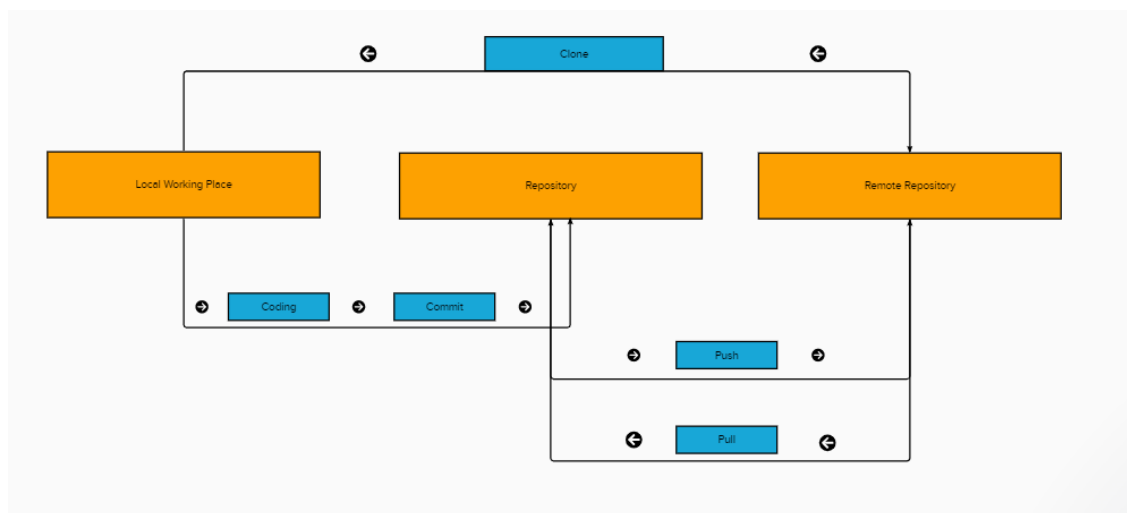


Figure 17 Git workflow.

#### 4.1.2 Node Package and Framework Installation

Some packages needed to be set up in this project, and the critical setup for those packages are listed below.

- NodeJS version 16.13.0 was installed from the official website. The application was initialized with the command 'npm init'. After that, the application's name, version, description, and author were set.
- MongoDB can be used online or locally using the MongoDB Compass.

- A React Application version 17.0.2 was created with the "create-react-app front-end" command.
- Mongoose can be installed with the "npm i mongoose" command.
- Nodemon is a tool used to monitor the server-side and was installed with the command "npm i nodemon".
- Bcrypt is a library that helps users hash passwords, so we do not save the original password to the database but the hash version. Bcrypt was installed with the command "npm i bcrypt".
- Redux is a library that helps the user manage and centralize the application state. Redux was installed with the command "npm i redux".
- React-router-dom is an npm package that enables developers to implement dynamic routing in a web app using ReactJS. React-router-dom was installed with the command "npm i react-router-dom".
- Axios is an npm package that provides promise base HTTP clients like PUT, GET, DELETE, and so on. for the browser and NodeJS. Axios was installed with the command "npm i axios".
- React-helmet is a document head manager for ReactJS. With this extension, we can customize the title of a route. React-helmet was installed with the command "npm i react-helmet".
- React Developer Tool and Redux Developer Tool are used to managing ReactJS's state and can be downloaded on the official page.
- Multer is a middleware for handling uploading files in our project. Multer was installed with the command "npm i multer".
- Express-async-handler is a middleware for handling exceptions inside async express routes and passing them to the error handlers. We are using this middleware in all our back-end functions, so we do not have to give the "try-catch" or ".catch" every time in our back-end development. This middleware was installed with the command "npm i express-async-handler".
- Jsonwebtoken extension can be used to generate a token for login. Jsonwebtoken was installed with the command "npm i Jsonwebtoken".



- Dotenv is a module that loads environment variables from a “.env” file into “process.env”. This module was installed with the command “npm i dotenv”.

Those are the essential packages and frameworks for this project. The less important ones are mentioned when discussing the implementation of the logic.

## 4.2 Application Logic

After installing the tools and having been, we come to the coding part. Figure 18 shows the application structure; as we can see, files are divided into two main folders: the front-end and the back-end. The front-end folder consists of all the user's UI directly interacts with. The main code for the front end is in the 'src' folder; the 'build' folder contains the final static build product and the auto-generated 'node\_modules'. The back-end folder contains all the logic and is split into multiple folders: 'config' handles the connection with database, 'controller' manage the functions for the application, 'data' holds the dummy data that we first push to the database, 'middleware' includes all the middleware for the application, 'models' is the place where the models for the database is stored, 'routes' for different routes created, and server.js is the central place where everything is connected. The role of each folder will be discussed later in this section.

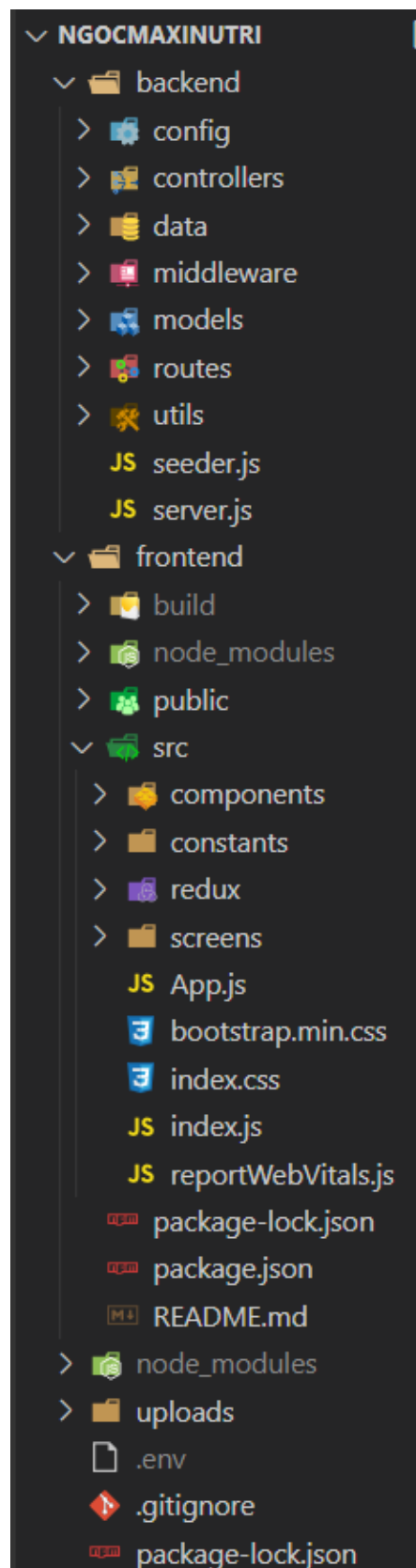


Figure 18 Structure of the application.

### 4.2.1 Front-end Logic

Every screen or route is divided into the “screens” folder, and every support element, such as Header, Footer, or Search Box, is divided into the “components” folder. By dividing everything into components, it is much easier to maintain and speed up the whole process instead of writing everything into one giant “HTML” file.

This project uses Redux to manage the state between components. Redux is an open-source JavaScript library for managing and centralizing the application state. /12/ With the everyday React state management, it is difficult to pass a variable to multiple layers of components. Still, with Redux, states are given into global storage (in our case, forms are stored in App.js), and we have access to this store everywhere in our application, making the process handier. Redux is used a lot in this application, and the document was discussed in detail with two examples of Redux in the project, “Log in” and “Add to Cart”.

Bootstrap is also discussed in detail since most components use Bootstrap for styling.

```
export const userLoginReducer = (state = {}, action) => {
  switch (action.type) {
    case USER_LOGIN_REQUEST:
      return { loading: true };
    case USER_LOGIN_SUCCESS:
      return { loading: false, userInfo: action.payload };
    case USER_LOGIN_FAIL:
      return { loading: false, error: action.payload };
    case USER_LOGOUT:
      return {};
    default:
      return state;
  }
};
```

Figure 19 Reducer for login.

```

export const login = (email, password) => async (dispatch) => {
  try {
    dispatch({
      type: USER_LOGIN_REQUEST,
    });

    const config = {
      headers: {
        'Content-Type': 'application/json',
      },
    };

    const { data } = await axios.post(
      '/api/users/login',
      { email, password },
      config
    );

    dispatch({
      type: USER_LOGIN_SUCCESS,
      payload: data,
    });

    localStorage.setItem('userInfo', JSON.stringify(data));
  } catch (error) {
    dispatch({
      type: USER_LOGIN_FAIL,
      payload:
        error.response && error.response.data.message
        ? error.response.data.message
        : error.message,
    });
  }
};

```

Figure 20 Action for the login.

```

const submitHandler = (e) => {
  e.preventDefault();

  dispatch(login(email, password));
};

```

Figure 21 Submit handler where login function is called.

Figures 19, 20, and 21 above show the reducer and action for logging in. When the user signs up, the login function is called, passing the email and

password to the login action, then the HTTP POST request to `/api/users/login` is made and gives the email and password to the login route in the back-end folder. If the process is successful, the login data, including id, name, email, admin check, and token, will be embedded under the "data" variable and saved to the store as "userInfo". If the process fails, the server's error message will be passed into the payload and shown in the "Message" component.

```
case CART_ADD_ITEM:
  const item = action.payload;

  const existItem = state.cartItems.find((x) => x.product === item.product);

  if (existItem) {
    return {
      ...state,
      cartItems: state.cartItems.map((x) =>
        x.product === existItem.product ? item : x
      ),
    };
  } else {
    return {
      ...state,
      cartItems: [...state.cartItems, item],
    };
  }
}
```

Figure 22 Reducer for Add to Cart.

```
export const addToCart = (id, qty) => async (dispatch, getState) => {
  const { data } = await axios.get(`/api/products/${id}`);

  dispatch({
    type: CART_ADD_ITEM,
    payload: {
      product: data._id,
      name: data.name,
      image: data.image,
      price: data.price,
      countInStock: data.countInStock,
      qty,
    },
  });

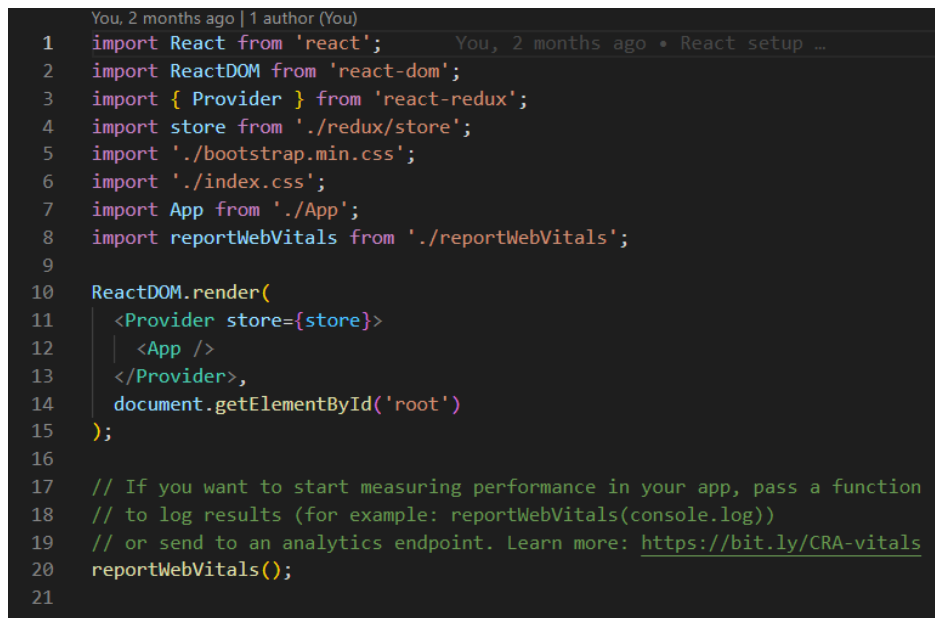
  localStorage.setItem('cartItems', JSON.stringify(getState().cart.cartItems));
};
```

Figure 23 Action for Add to Cart

Figures 19, 20, and 21 above show the reducer and action for adding to the cart. When the user clicks the "Add to cart" button, the page will navigate to the cart route, which has the product's id and quantity. First, the data of that

specific product is fetched from the server. After that, the “CART\_ADD\_ITEM” action is called; it will find wherever the product has been in the cart or not. If the product has not been in the cart, a new product will be added; otherwise, find it in the cart and update it. After the process, the final cartItems are updated to local storage.

Bootstrap 5, together with Bootswatch, is used almost in every front-end component in this project. Bootswatch is a free theme for Bootstrap; with this library setup, all the styling will be set to a theme, synchronizing everything. Bootswatch theme was downloaded from the official website and passed into the front-end's index.js file, as shown in Figure 24.



```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import { Provider } from 'react-redux';
4 import store from './redux/store';
5 import './bootstrap.min.css';
6 import './index.css';
7 import App from './App';
8 import reportWebVitals from './reportWebVitals';
9
10 ReactDOM.render(
11   <Provider store={store}>
12     <App />
13   </Provider>,
14   document.getElementById('root')
15 );
16
17 // If you want to start measuring performance in your app, pass a function
18 // to log results (for example: reportWebVitals(console.log))
19 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();
21
```

Figure 24 Main index.js file inside front-end folder

In Figure 24, on line 5, the “bootstrap.min.css” downloaded from Bootswatch is imported. Also, on line 11, we can see that the Redux store is imported, gaining the accessibility for the whole application to global variables.

Regarding Bootstrap in this project, the component “ProductCarousel” and “Footer” were considered.

```

const Footer = () => {
  return (
    <footer>
      <Container>
        <hr />
        <Row>
          <Col md={4} sm={12} className='text-center py-3'>
            Copyright &copy; Phuc Le Vinh      You, 3 weeks ago • update the name ...
          </Col>

          <Col md={8} className='text-center py-3'>
            <Row>
              <Col sm={12}>THÔNG TIN LIÊN HỆ</Col>
              <Col sm={12}>
                <strong>Địa Chỉ</strong>: 29/11 Nguyễn Văn Khỗi, phường 11, Quận
                Gò Vấp, Thành phố Hồ Chí Minh
              </Col>
              <Col sm={12}>
                <strong>Di động</strong>: 0938 192 499 - 0933 21 89 66 (Ms. Đài)
              </Col>
              <Col sm={12}>
                <strong>Email</strong>: suachuangoc@gmail.com
              </Col>
            </Row>
          </Col>
        </Row>
      </Container>
    </footer>
  );
};

export default Footer;

```

Figure 25 The use of Bootstrap in the Footer component.

```

const ProductCarousel = ({ products }) => {
  return (
    <Carousel pause='hover' variant='dark'>
      {products.map((product) => (
        <Carousel.Item key={product._id} className='carousel-item'>
          <Link to={` /product/${product._id}`}>
            <Image src={product.image} alt={product.name} fluid />
            <Carousel.Caption className='carousel-caption'>
              <h2>{product.name}</h2>
            </Carousel.Caption>
          </Link>
        </Carousel.Item>
      ))}
    </Carousel>
  );
};

```

Figure 26 The use of Bootstrap in the ProductCarousel component.

In Figure 25 and Figure 26, the concept of Bootstrap 5 combined with Bootswatch and “react-bootstrap” is clearly shown. In that Figure, Bootstrap was used by applying the built-in classes to elements and components, such as Row, Col, and Carousel.

#### 4.2.2 Back-end Logic

The combination of NodeJS, ExpressJS, Mongoose, and other helper libraries makes the whole application run in the back-end.

Routing or router in this web application is a mechanism where HTTP requests are routed to the code that handles them. We determine what should happen when a user visits a specific page in the route./13/ In this application, courses are divided into four categories based on their use: orderRoutes, productRoutes, uploadRoutes, and userRoutes.

Table 1. List of all routes made in the project.

Route	Purpose	Access
POST /api/users/login	Authorization user and get a token	Public
POST /api/users	Register a new user	Public
GET /api/users/profile	Get user profile	Private
PUT /api/users/profile	Update user profile	Private
GET /api/users	Get all users	Admin
DELETE /api/users/:id	Delete user	Admin
GET /api/users/:id	Get user by id	Admin
PUT /api/users/:id	Update user by id	Admin
GET /api/products	Fetch all products	Public
GET /api/products/:id	Fetch single product	Public
DELETE /api/products/:id	Delete single product	Admin
POST /api/products/	Create single product	Admin



PUT /api/products/:id	Update single product's content	Admin
POST /api/products/:id/review	Create new review	Private
POST /api/orders	Create new order	Private
GET /api/orders/:id	Get order by ID	Private
GET /api/orders/:id/pay	Update order to paid	Private
GET /api/orders/myorders	Get logged in user order	Private
GET /api/orders/	Get all orders	Admin
GET /api/orders/:id/deliver	Update order to delivered	Admin

All the routes made for this web application are shown in Table 1; the route is split into three colors for three purposes: blue for the user route, purple for the products, and orange for the order routes. The basic definition of the functions of the HTTP route that this project use are as follows:

- GET: Obtain information about something.  
Example: GET /api/users/profile means Obtaining the user profile's information.
- POST: Add or push information to the back end.  
For example, POST /api/users/login means to push the information entered to the server to log in to that user's account.
- PUT: Replace or update the information in the database.  
Example: PUT /api/users/profile means to update the user profile like name, email, or password.
- DELETE: Delete information from the database.  
Example: DELETE /api/users/:id means to delete the user account.

Authentication is one of the main functions of this application. Authentication is the process of determining someone or something. Authentication technology in this project provides access control to the system for the customer

by checking to see if a customer's credentials and token match with the database /14/. There are many kinds of authentication, such as the face, voice, fingerprint, or two factors. Still, the most common form of authentication in this application is being used via a login form, where the user can enter their email and password to log in.

The authentication of this application is handled by "JSON web token" as a middleware that is passed to the route requires login. When the user succeeds in logging into the application, the user's id generates the token. When that user calls any API that JWT protects, that token will be decoded and checked to see if that user has permission to perform that API call. The authentication for the user is shown below in Figure 27 in this application.

```
const protect = expressAsyncHandler(async (req, res, next) => {
  let token;

  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith('Bearer')
  ) {
    try {
      // You, last month * make the register, user profile screen
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      req.user = await User.findById(decoded.id).select('-password');

      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Không được xác thực, token lỗi');
    }
  }

  if (!token) {
    res.status(401);
    throw new Error('Không được xác thực, token lỗi');
  }
});
```

Figure 27 Token authentication by JWT

In Figure 27, firstly, we check if there is an authorization request using "Bearer," and after that, use jwt. To verify the method to decode the token, in this step, the token will be solved with a secret jwt word saved in our environment variable. Finally, the findById method provided by Mongoose is used to see if there is any user id in the database that match the token. If the id is found, the next() function is called and continues to the next step.

Otherwise, the error will be shown if the token is wrong or does not exist in the database.

This application also has an authentication for the admin user, which is shown in figure 28 below.

```
const admin = (req, res, next) => {
  if (req.user && req.user.isAdmin) {
    next();
  } else {
    res.status(401);
    throw new Error('Không được xác thực là tài khoản admin');
  }
};
```

Figure 28 Admin user authentication.

In Figure 28 above, the admin is checked by seeing if the “isAdmin” property is contained in the request header.

Two functions are used to respond to the error created, as shown in Figure 29 below.

```
const notFound = (req, res, next) => {
  const error = new Error('Not Found - ${req.originalUrl}');
  res.status(404);
  next(error);
};

const errorHandler = (err, req, res, next) => {
  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
  res.status(statusCode);
  res.json({
    message: err.message,
    stack: process.env.NODE_ENV === 'production' ? null : err.stack,
  });
};

export { notFound, errorHandler };
```

Figure 29 Error handler function.

The function will return the status code and error message in those functions.

Postman was used to testing all the back-end routes before using that API call in the front-end.

“Postman is an application used for API testing. It is an HTTP client that test HTTP request, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.” /15/

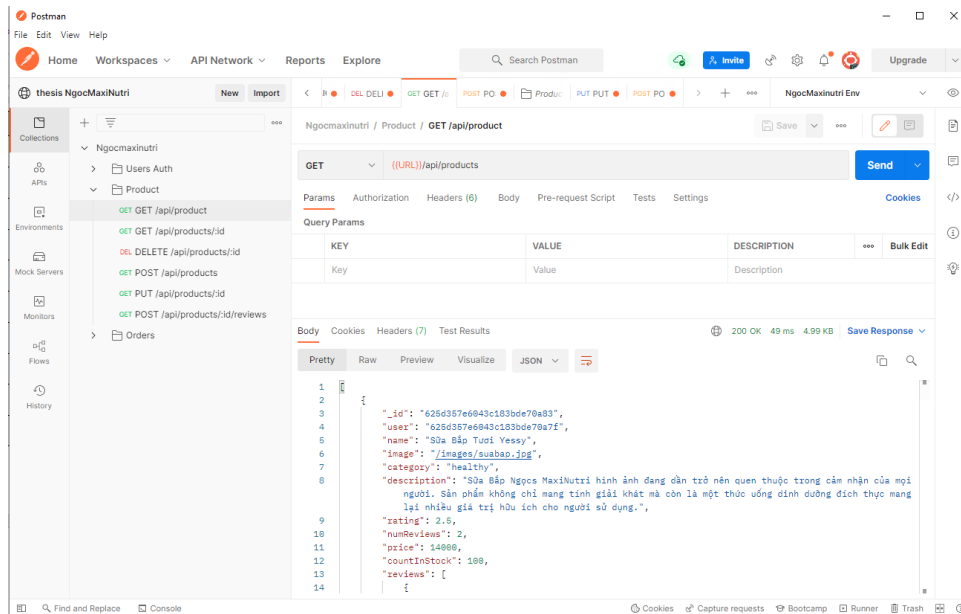


Figure 30 Using Postman to test the GET /api/products route.

In Figure 30, the UI of the Postman application is shown; here, we can see the method is GET with the input is “{{URL}}/api/products”, the “URL” is port “https://localhost:5000” as we are running the back-end on port 5000. The response is shown at the bottom of the Figure on the Body tab; the “Pretty” option will format JSON and XML responses so we can view them more straightforward; the “Raw” view is a bunch of text without any space or line breaks, which is the actual response from the server. Postman also provides other options such as “Cookies”, “Headers” and “Test results” so users can have more information about the API that they are testing. In this project, authentication can also be checked with Postman using the “Authorization” option from the menu, which is handy for private APIs that require a login method beforehand.

### 4.2.3 Database Implementation

This application uses MongoDB and Mongoose for the database section. To save new data to MongoDB, we need to create models to define the architecture of the specific document that will be stored in the database. This application has three models, one for the order, one for the product, and one for the user. Figure 31 below shows a part of the user model as an example in this application.

```
import mongoose from 'mongoose';
import bcrypt from 'bcryptjs';

const userSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    isAdmin: {
      type: Boolean,
      required: true,
      default: false,
    },
  },
  {
    timestamps: true,
  }
);
```

Figure 31 Application's user model.

In Figure 31, a Mongoose schema named `userSchema` was formed and represented the structure of the user, including name, password, email, and admin check. In `userSchema`, clarification of the type of the object and the “required” can also be set, making the database in a better format.

When data is pushed to MongoDB, the key or “\_id” as named by MongoDB will automatically generate, as shown in Figure 32 below.

```
_id: ObjectId("625d357e6043c183bde70a7f")
name: "Admin User"
email: "admin@example.com"
password: "$2a$10$a07Np6qkvWm6ck4jItTx5uA/QTjdJ8bGvbqcJsy2zqx70hEKk0bsq"
isAdmin: true
__v: 0
createdAt: 2022-04-18T09:55:10.690+00:00
updatedAt: 2022-04-18T09:55:10.690+00:00
```

Figure 32 Example of a user object in MongoDB.

After creating an account in MongoDB and going into the project, a unique Mongo URL is generated from the MongoDB website. A copy of that URL is passed into the environment variable file and used to connect to the database. Figure 33 below shows how to connect to the database in this project.

```
import mongoose from 'mongoose';

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URL, {
      useUnifiedTopology: true, //option to remove warning
    });

    console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
  } catch (error) {
    console.error(`Error: ${error.message}`.red.underline.bold);
    process.exit(1);
  }
};

export default connectDB;    You, 2 months ago • adding colors lib ...
```

Figure 33 Connection to a database.

An async operation was used since everything needs to be asynchronous when dealing with the database. The method used for connection is the `mongoose.connect()` provided by the `mongoose`. The `MONGO_URL` can be seen also from the `.env` file.

Before importing the initial data for the application, some initial data has been created based on our models. Figure 34 below shows the data that is prepared for users.

```
import bcrypt from 'bcryptjs';

const users = [
  {
    name: 'Admin User',
    email: 'admin@example.com',
    password: bcrypt.hashSync('123456', 10),
    isAdmin: true,
  },
  {
    name: 'Phuc Le',
    email: 'phuc@example.com',
    password: bcrypt.hashSync('123456', 10),
  },
  {
    name: 'Tram Nguyen',
    email: 'tram@example.com',
    password: bcrypt.hashSync('123456', 10),
  },
];

export default users;
```

Figure 34 Initial user data.

As shown in Figure 34 above, each object follows the standard of the user model we created, and the password is hashed with bcryptjs, a library that helps hash the password. Using a set algorithm, hashing a password means turning the original password into a scrambled representation of itself using the combination of the password and the provided key. /16/ The key this application uses is the number “10” The result is shown in Figure 32 above; the original password is hashed to another one, making the database safer for users.

After we have the initial data, a method named “importData” is called to push our data to the database. This function is shown in Figure 35.

```

const importData = async () => {
  try {
    await Order.deleteMany();
    await Product.deleteMany();
    await User.deleteMany();

    const createdUsers = await User.insertMany(users);
    const adminUser = createdUsers[0]._id;
    const sampleProducts = products.map((product) => {
      return { ...product, user: adminUser };
    });

    await Product.insertMany(sampleProducts);

    console.log('Data Imported!'.green.inverse);
    process.exit();
  } catch (error) {
    console.error(`${error}`.red.inverse);
    process.exit(1);
  }
};

```

Figure 35 Function used to create the initial data.

In Figure 35, first, we clear the database with “.deleteMany()”; after that, push all the users we have with “.insertMany()”, then the admin id is attached to all the sample products we have in the database and push to MongoDB.

### 4.3 Application Deployment

To deploy this application, Heroku was used. Heroku is a cloud service platform used for application development and deployment; with Heroku, deployment is now much simple than before. Since the Heroku platform manages hardware and servers, users only need to focus on their applications and do not need the infrastructure that supports them. /17/ Figure 36 shows this project in Heroku.



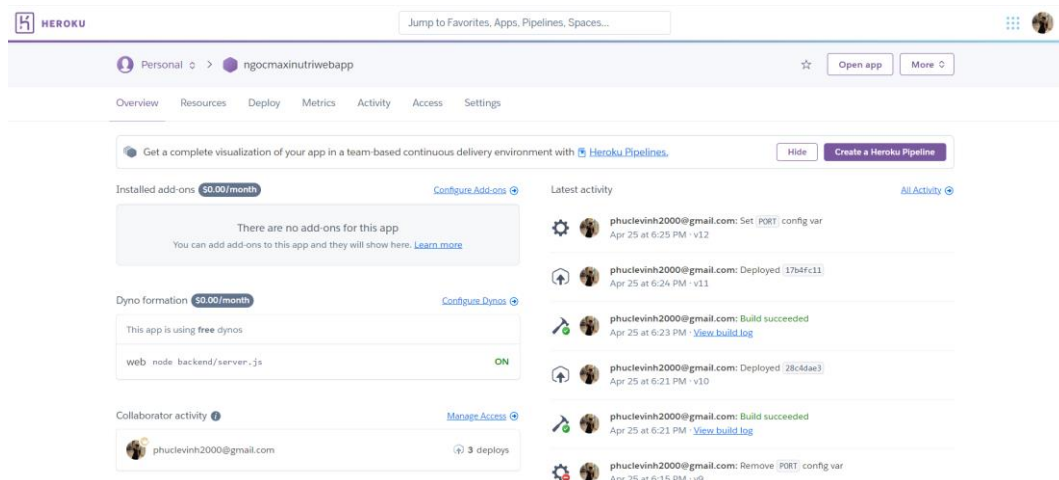


Figure 36 Project UI inside Heroku.

As we can see from Figure 36 above, the UI of Heroku is quite like other Git systems such as GitHub or GitLab. The application was deployed to Heroku with the npm script command “NPM\_CONFIG\_PRODUCTION=false npm install --prefix front-end && npm run build --prefix front-end”.

After pushing the project to Heroku, the environment variables needed to be defined again via the “Setting” option in the Heroku task pane since we have put our environment variable in the file “.env” to “.gitignore”.

After completing everything, now the application is deployed via the domain <https://ngocmaxinutri.herokuapp.com/>; with the business upgrade, the “herokuapp” can be removed. Furthermore, the eCommerce application was completed.

## 5 CONCLUSIONS

The thesis aimed to study and build a full-stack web application that can strengthen my coding skills and make a place for my family company to sell our products online. The project was well implemented in approximately four months, from the end of January 2022 until May 2022.

With this application, customers can order products online, have the tracking status, and the owner can easily update the product and manage users, orders, and products.

The web application is easy to use but is not easy to implement, especially for a junior developer with basics in Embedded systems. One of the most challenging parts was working with the MongoDB database and model schema. Thanks to Bootstrap, no CSS was a need.

Although the application has been completed, some implementations can still be made for the user experience and the UI. Firstly, implementing the Dark Mode, embedding a password manager to track the password are needed, updating the filter for the product, and creating a Google advertisement to promote the brand. Finally, an automation test for the web application is required so that no manual testing needs to be done.

## REFERENCES

- /1/ Roznovsky, Alexander Roznovsky. Choosing a technical stack for web application development. Accessed 8.5.2022.  
<https://light-it.net/blog/choosing-a-technology-stack-for-web-application-development/>.
- /2/ MongoDB. MERN Stack Explained. Accessed 8.5.2022.  
<https://www.mongodb.com/mern-stack>
- /3/ ReactJS. React – A JavaScript library for building user interfaces. Accessed 8.5.2022.  
<https://reactjs.org/>.
- /4/ Alexandre Ouellette. 2021. What is Bootstrap: A Beginner's Guide. Accessed 8.5.2022.  
<https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/>.
- /5/ Tutorials Point. Node.js – Introduction. Accessed 9.5.2022.  
[https://www.tutorialspoint.com/nodejs/nodejs\\_introduction.html](https://www.tutorialspoint.com/nodejs/nodejs_introduction.html).
- /6/ Express. Hello, world example. Accessed 9.5.2022.  
<https://expressjs.com/en/starter/hello-world.html>
- /7/ Oracle. What is Database? Accessed 15.5.2022.  
<https://www.oracle.com/database/what-is-database/>.
- /8/ David Taylor. 2022. What is MongoDB? Introduction, Architecture, Features & Example. Accessed 9.5.2022.  
<https://www.guru99.com/what-is-mongodb.html>
- /9/ Ado Kukic and Stanimira Vlaeva. 2022. MongoDB & Mongoose: Compatibility and Comparison. Accessed 9.5.2022.  
<https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/>.
- /10/ Visual Studio Code. Documentation. Accessed 10.5.2022.  
<https://code.visualstudio.com/docs>
- /11/ Bit Bucket. What is version control? Accessed 10.5.2022  
<https://www.atlassian.com/git/tutorials/what-is-version-control#:~:text=Version%20control%2C%20also%20known%20as,to%20source%20code%20over%20time.>

- /12/ ReduxJs. Getting Started with Redux. Accessed 16.05.2022  
<https://redux.js.org/introduction/getting-started>
  
- /13/ DivPusher. What is routing? Accessed 18.05.2022  
<https://divpusher.com/glossary/routing/#:~:text=Routing%20or%20router%20in%20web,user%20visits%20a%20certain%20page.>
  
- /14/ Mary E. Shacklett. Authentication. Accessed 20.05.2022  
<https://www.techtarget.com/searchsecurity/definition/authentication>
  
- /15/ Gustavo Romero. What is Postman API test? Accessed 20.05.2022  
<https://www.encora.com/insights/what-is-postman-api-test>
  
- /16/ Samuel Gibbs. Passwords and hacking: the jargon of hashing, salting, and SHA-2 explained. Accessed 21.05.2022  
<https://www.theguardian.com/technology/2016/dec/15/passwords-hacking-hashing-salting-sha-2#:~:text=When%20a%20password%20has%20been,key%2C%20using%20a%20set%20algorithm.>
  
- /17/ Konstantin Rusev. What is Heroku, and what is it used for? Access 21.05.2022  
<https://mentormate.com/blog/what-is-heroku-used-for-cloud-development/>