VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Duc Anh Le

E-COMMERCIAL FULL STACK WEB APPLICATION DEVELOPMENT

with React, Redux, NodeJS, and MongoDB

Technology and Communication
2023

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Duc Anh Le
Title	E-Commercial Full Stack Web Application Development
Year	2023
Language	English
Pages	78
Name of Supervisor	Harri Lehtinen

E-commerce is a fast-expanding industry, and businesses rely increasingly on their online presence to attract global customers. Businesses want full-stack web applications that can support their operations throughout the whole e-commerce lifecycle in order to meet the growing demand.

This thesis intends to create a web application that integrates front-end and back-end technology to streamline e-commerce processes. Utilizing multiple software frameworks and programming languages, the application creates a powerful platform. This research gives insights into the difficulty of designing full-stack e-commerce applications and suggests strategies for overcoming these obstacles.

This thesis utilizes the technologies React, Redux, and SASS for the front-end, and Node.js and Express for the back-end. The primary database system of the program is MongoDB. TypeScript is the primary programming language, with HTML/CSS support for web structure and design. The application allows authentication with Google Passport as a main method of authentication.

Overall, the full-stack web application developed for this thesis provides e-commerce enterprises with a means to better their online presence and increase sales.

Keywords	React, Redux, SASS, Node.js, Express.js, MongoDB, TypeScript, and Google Passport

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION OF THE THESIS

1.1 Background

The advent of e-commerce has brought about a significant transformation in contemporary business practices, and the persistent demand for e-commerce solutions indicates a sustained trend towards its adoption. The growing trend of businesses shifting towards online platforms and adopting digital transformation has accentuated the significance of dependable, secure, and effective e-commerce solutions. The thesis investigates the creation of a comprehensive web-based platform for commercial transactions, which involves the implementation of both client-side and server-side programming, as well as the integration of a database system. The objective of this thesis is to furnish a thorough manual for the creation of e-commerce websites, encompassing all aspects ranging from design to deployment, security, and performance enhancement.

1.2 Objectives and Processes

The process of developing a full-stack web application comprises several stages, each of which presents unique challenges and complexities. The initial phase of front-end development encompasses the development of an interface that is user-friendly, possesses responsive design, and features intuitive navigation. Conversely, the subsequent stage of back-end development is centered on the construction of resilient server-side applications and APIs. The phase of integrating the database is of utmost importance and necessitates meticulous planning for data modeling, storage, and retrieval. To ensure a smooth and secure user experience, it is imperative for an e-commerce web application to seamlessly integrate all of these components.

The thesis highlights the fundamental principles, tools, and technologies employed in the creation of a comprehensive full-stack e-commerce web application. A comprehensive survey of pertinent academic and industry literature, along with case studies, was undertaken to ascertain the optimal approaches and methodol-

ogies for creating e-commerce websites that are efficient, dependable, and secure. The thesis offers practical exposure to the creation and implementation of an e-commerce website, utilizing prevalent web development frameworks, content management systems (CMS), and hosting platforms.

1.3 Expectations

The outcomes of this research will be advantageous for individuals involved in software development, business ventures, and those with a keen interest in electronic commerce. It will provide valuable knowledge on the most effective techniques for creating and implementing comprehensive e-commerce web applications. The objective of the study is to provide a user-friendly approach that illustrates the systematic procedure of creating web applications. This includes the provision of practical instances, code excerpts, and visual representations to assist novices in the field of web development. Furthermore, the outcomes of this thesis will aid enterprises in optimizing their procedures, improving their marketing strategies, and delivering an improved digital shopping experience to their clientele.

In summary, this thesis aims to bridge the gap between theory and practice in web development and provide valuable insights for businesses looking to enhance their online presence. The practical examples and recommendations presented in this study can serve as a valuable resource for both novice developers and established companies seeking to stay ahead in the digital landscape.

2 TYPESCRIPT

2.1 Introduction

Microsoft has developed the programming language TypeScript, an open-source programming language that supersedes JavaScript. TypeScript seeks to improve the development experience for developers by adding static typing, class, and interface definitions to JavaScript.

> TypeScript is an open-source pure object-oriented programming language. It is a strongly typed superset of JavaScript which compiles to plain JavaScript. It contains all elements of the JavaScript. It is a language designed for large-scale JavaScript application development, which can be executed on any browser, any Host, and any Operating System. The TypeScript is a language as well as a set of tools. TypeScript is the ES6 version of JavaScript with some additional features.
>
> TypeScript cannot run directly on the browser. It needs a compiler to compile the file and generate it in JavaScript file, which can run directly on the browser. The TypeScript source file is in ".ts" extension. We can use any valid ".js" file by renaming it to ".ts" file. TypeScript uses TSC (TypeScript Compiler) compiler, which converts TypeScript code (.ts file) to JavaScript (.js file). (Sharma, S. 2020. Online)

TypeScript is a statically typed programming language that is a superset of JavaScript. It compiles to ordinary JavaScript and is compatible with all browsers, hosts, and operating systems. TypeScript provides advanced features, such as interfaces, classes, and modules that make it simpler for developers to write code that is maintainable and scalable.

The TypeScript syntax is very similar to that of JavaScript. It includes static typing and other features that aid in error detection at compile time. This makes it simpler to detect errors prior to sending code to production. TypeScript supports features such as decorators, which allow programmers to add metadata to their code.

One of the primary advantages of TypeScript is that it makes code more modular and manageable. By utilizing classes and interfaces, developers can better structure their code. This facilitates code maintenance and refactoring as the size of the codebase increases.

TypeScript is additionally extremely interoperable with other JavaScript frameworks and libraries. This implies that developers can use TypeScript with popular front-end frameworks such as React, Angular, and Vue.

TypeScript is a potent instrument for developers who wish to write maintainable, scalable, and error-free code. Its advanced features make it simpler for developers to write complex applications, and its interoperability with other frameworks and libraries enables its use in a variety of projects.

2.2 Advantages of TypeScript

TypeScript is a statically typed programming language that offers many advantages over dynamically typed languages, particularly for developers. Here are some of TypeScript's primary benefits:

TypeScript provides robust type checking capabilities that aid in identifying errors at compile time as opposed to runtime. This increases the reliability and quality of the code. TypeScript also enforces strict syntax standards, which can aid in reducing common programming errors.

Maintaining and updating code can be difficult, particularly as the complexity of a project increases. TypeScript offers enhanced documentation, code organization, and error handling to address this issue. Additionally, the intrinsic typing system better organizes code, making it simpler to read and comprehend.

Refactoring is an essential aspect of software development because it enables the enhancement of features and functionality without disrupting the existing code. TypeScript's streamlined syntax, modular structures, and helpful error messages make it simpler to refactor code without affecting other code elements.

Reading and comprehending code is frequently difficult, particularly when working with the code of others. TypeScript facilitates the process by enforcing strict syntax rules, clear variable declarations, and improved organization in general. The language also supports type validation and documentation, which improves the overall readability of code.

2.3 Disadvantages of TypeScript

TypeScript is a superset of JavaScript that facilitates the development of sophisticated web applications. Despite its many benefits, such as type checking, code organization, and enhanced developer tools, it has a number of disadvantages that can negatively impact the development process. In this section, we will discuss TypeScript's three primary drawbacks:

2.3.1 Slow Development Time

One of the major drawbacks of using TypeScript is that it can lengthen the development process. TypeScript's strict typing system necessitates that developers define the data types of each variable explicitly, which can be time-consuming and laborious. This can help prevent errors and improve code quality, but it can also slow down the development process, which can be frustrating for developers who must move rapidly.

The need to continually compile TypeScript code into JavaScript is an additional factor that contributes to longer development times. Because most web browsers do not inherently support TypeScript, it must be compiled into JavaScript before it can be executed. This additional stage increases the duration and complexity of the development process.

2.3.2 Complex Initial Setup

Setting up TypeScript can be difficult for developers unfamiliar with the language. Unlike traditional JavaScript, which only requires an HTML file and a JavaScript file, TypeScript setup requires the installation and configuration of several additional tools.

For instance, developers must first install Node.js and npm before installing the TypeScript compiler, which can be installed globally or locally. In addition, developers must frequently install and configure distinct development environments, such as Webpack, in order to support TypeScript.

For newcomers in particular, these additional setup procedures can be time-consuming and perplexing, which may deter developers from utilizing TypeScript altogether.

2.3.3 Limited Browser Compatibility

TypeScript has limited browser compatibility, which is a significant disadvantage. While the majority of modern web browsers support JavaScript, not all can run TypeScript out of the box. This can be problematic for developers who must ensure that their applications are compatible with a broad range of devices and platforms.

Moreover, TypeScript may not be compatible with all JavaScript libraries and frameworks, particularly older ones that have not been updated to support it. This can restrict the options available to developers, making it harder to acclimate to changes in the web development environment.

2.4 Conclusion

TypeScript provides developers with numerous advantages when it comes to improving code quality, increasing maintainability, enabling better refactoring, and improving readability and understanding of code. However, it also has several disadvantages that can slow down development time, make initial setup more complex, and limit browser compatibility. Ultimately, whether or not to use TypeScript depends on the specific needs of the project, your level of experience with the language, and your development goals.

TypeScript provides a number of implications for developers that can considerably enhance the development process. TypeScript enables programmers to write cleaner and more concise code by supplying type annotations. Using type annotations, developers can readily detect errors during the development phase, thereby reducing errors that occur at runtime. In addition, TypeScript provides improved code navigation and completion, and enables developers to write more robust

code by leveraging interfaces, classes, and inheritance. TypeScript also offers enhanced tooling support and compiler optimizations to ensure the code is quick and efficient. Overall, TypeScript provides numerous benefits for developers, and it can considerably enhance the quality of code by providing improved type checking and a more solid code structure.

3 THE MONGODB DATABASE MANAGEMENT SYSTEM

3.1 Overview

3.1.1 Definition

MongoDB is a NoSQL document-oriented database that allows for flexible and scalable data storage. It is designed to manage large volumes of unstructured and semi-structured data, making it ideal for modern web applications and big data processing. MongoDB also provides high availability and automatic scaling features, which ensure that applications can handle increasing amounts of data without compromising performance or reliability. Its dynamic schema allows for easy modification and evolution of data models as business needs change over time.

> MongoDB is an open source NoSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.
>
> MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.
>
> Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents. Documents are made up of key-value pairs -- MongoDB's basic unit of data. Collections, the equivalent of SQL tables, contain document sets. MongoDB offers support for many programming languages, such as C, C++, C#, Go, Java, Python, Ruby and Swift. (Gillis, A. S. & Botelho, B. 2023. online)

Additionally, MongoDB offers a flexible query language and powerful indexing capabilities that enable developers to retrieve data quickly and efficiently. These features make MongoDB a popular choice for building modern, scalable applications.

3.1.2 Advantages

Some advantages of using MongoDB include its ability to scale horizontally, its flexible data model, and its support for dynamic queries. Additionally, MongoDB's

document-oriented approach allows for easier integration with object-oriented programming languages. This makes it a popular choice for developers who want to build applications with complex data structures. Moreover, MongoDB's automatic sharding and replication features provide high availability and fault tolerance for mission-critical applications.

MongoDB is a choice that is suited for both small and large-scale projects due to its flexibility, which enables it to easily scale up in response to growing data requirements. In addition, its document-based paradigm makes the development process simpler by enabling developers to work with data in a manner that is more in line with how they naturally think about it. This, in turn, makes the process simpler.

3.2 Characteristics of MongoDB

3.2.1 Database Structure

MongoDB is a document-oriented NoSQL database that stores data in flexible, JSON-like documents. It uses a dynamic schema, allowing the storage of complex hierarchical data structures with ease. MongoDB is a popular choice for modern web and mobile applications due to its scalability, high availability, and ease of use. MongoDB's flexible data model allows for faster iteration and more efficient development. Additionally, its document-oriented approach enables developers to work with data in a way that closely mirrors their programming language of choice.

This makes it easier for developers to build and maintain applications, as they can work with data in a way that is natural and intuitive to them. MongoDB's scalability and high availability features also make it a popular choice for large-scale, mission-critical applications.

3.2.2 Data Model

The MongoDB data model is based on collections and documents, which allows for flexible and scalable data storage. It also supports various data types, including

arrays and embedded documents, making it ideal for handling complex data structures. Its flexible data model also allows for faster development and iteration times compared to traditional relational databases. Additionally, MongoDB's distributed architecture enables horizontal scaling and high availability, ensuring that applications can handle copious amounts of data and remain operational even in the event of hardware failure. This makes it a popular choice for modern web applications and big data processing.

Moreover, MongoDB's flexible data model allows for easy integration with various programming languages and frameworks, making it a versatile solution for developers. Its document-oriented approach also enables faster development cycles and easier data manipulation compared to traditional relational databases.

3.2.3 Query Language

Query Language of MongoDB allows users to retrieve and manipulate data stored in MongoDB databases using a variety of operators and commands. It supports a flexible and powerful syntax that enables complex queries to be executed efficiently. MongoDB's Query Language also allows users to perform aggregations and transformations on data, making it a versatile tool for data analysis and reporting.

Additionally, it can be integrated with programming languages such as Python and Java for seamless data processing. This flexibility and integration capabilities make MongoDB a popular choice for modern data-driven applications that require efficient and scalable data storage and retrieval. With its rich query language and robust features, MongoDB has become a leading NoSQL database solution in the industry.

3.3 Applications of MongoDB

3.3.1 Web Applications

Web applications with MongoDB offer a scalable and flexible solution for managing large amounts of data. MongoDB’s document-oriented database model allows for easy integration with web frameworks, making it a popular choice for modern

web development. In addition, MongoDB's dynamic schema allows for faster development cycles and easier data modeling compared to traditional relational databases. This makes it a great option for agile development teams that need to quickly iterate and adapt to changing requirements.

Furthermore, MongoDB's scalability and ability to handle large amounts of unstructured data make it ideal for applications that require high performance and flexibility. This is particularly useful in industries such as finance, healthcare, and e-commerce where data needs can be complex and constantly evolving.

3.3.2 Cloud Computing

Cloud Computing with MongoDB allows developers to easily deploy and scale their applications on cloud platforms, providing flexibility and cost-effectiveness. This has made MongoDB a preferred choice for cloud-based applications and services. MongoDB's ability to handle large amounts of unstructured data and its support for distributed architectures make it well-suited for cloud computing. Additionally, its document-oriented data model allows for faster development and deployment of applications in the cloud.

Moreover, MongoDB's flexible data model enables developers to easily modify their data structures as their applications evolve, making it an ideal choice for rapidly changing cloud environments. Finally, its built-in horizontal scaling capabilities allow for seamless growth of databases as demand increases.

3.3.3 Mobile Applications

MongoDB's flexible data model and ability to handle large amounts of unstructured data make it an ideal choice for mobile applications that require real-time data syncing and offline capabilities. Additionally, MongoDB's native support for geospatial queries allows developers to build location-based features into their mobile apps with ease. This makes it a popular choice for mobile apps in industries such as transportation, logistics, and healthcare. With MongoDB, developers can

create robust and scalable mobile applications that meet the demands of today's users.

With MongoDB, developers can create robust and scalable mobile applications that meet the demands of today's users. MongoDB's flexible data model and ability to handle large amounts of unstructured data also make it a suitable choice for IoT applications, where data is generated from various sources. Additionally, MongoDB's cloud-based Atlas service offers a fully managed solution for deploying and scaling mobile applications.

3.3.4 Big Data Analytics

This feature enables developers to analyze large amounts of location data and gain insights into user behavior, preferences, and patterns. MongoDB's flexible data model also makes it easy to integrate with other big data tools for even more powerful analytics capabilities. With this feature, developers can create personalized experiences for users based on their location data. Additionally, MongoDB's scalable infrastructure allows for efficient processing of large datasets, making it ideal for businesses with rapidly growing data needs.

This combination of location-based personalization and scalable infrastructure is particularly useful for businesses in the e-commerce and retail industries, as it enables them to provide targeted promotions and recommendations to customers in real-time. By leveraging MongoDB's capabilities, these businesses can improve customer engagement and drive sales growth.

3.4 Implementation of MongoDB in this thesis

3.4.1 Data Models

This thesis demonstrates two types of data: Product and User.

The Product data type includes several essential properties of a typical product: name, description, image link, categories to which the product belongs, variants

of the product in terms of colors, shapes, and other physical characteristics, and product sizes.

The User data type features some important properties of a user: email address, first name, last name, and the role of the user (normal user or administrator).

```
export type ProductDocument = Document & {
  name: string
  description: string
  img: string
  categories: string[]
  variants: string[]
  sizes: string[]
}
```

Figure 1. Product type definition

```
export enum Role {
  ADMIN = 'admin',
  USER = 'user',
}

export type UserDocument = Document & {
  email: string
  firstname: string
  lastname: string
  role: Role
}
```

Figure 2. User type definition

3.4.2 Data in the MongoDB Database System

In the MongoDB database, these two types of data are represented as follows:

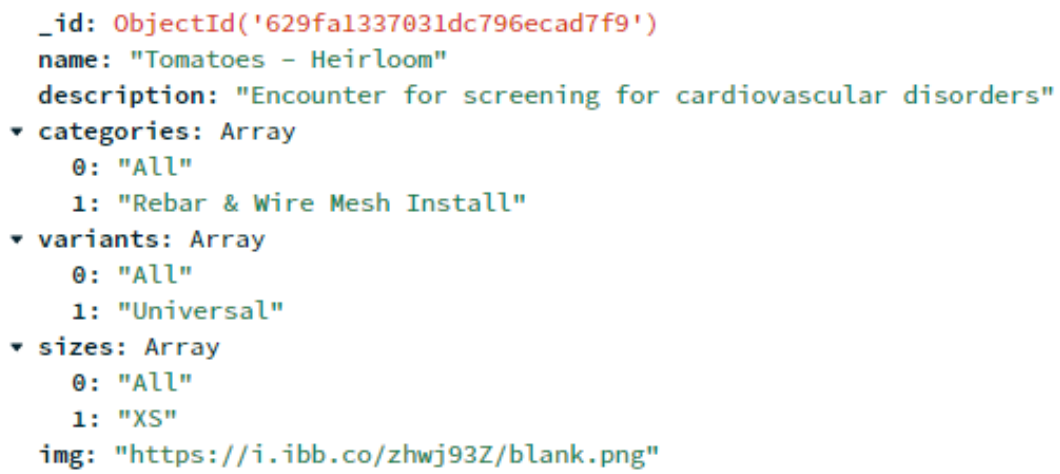

Figure 3. Product data type example in the database

Figure 4. User data type example in the database

In addition to the fields that have been defined in the data models above, MongoDB database system automatically generates extra fields, the _id and the __v fields. The _id field states the unique identifier for an object (in this case it can be either a Product or a User), and the __v field indicates the version number, which shows changes made on the object over time.

4 THE BACK-END SIDE

4.1 Introduction

4.1.1 Definition of NodeJS and ExpressJS

NodeJS is a server-side JavaScript runtime environment that allows developers to build scalable and high-performance applications. ExpressJS, on the other hand, is a popular NodeJS web application framework that simplifies the process of building web applications by providing a set of robust features and tools.

NodeJS and ExpressJS have revolutionized the way web applications are built, as they provide developers with an efficient and effective way to create scalable applications. The combination of NodeJS's fast processing capabilities and ExpressJS's streamlined framework has made it possible for developers to build web applications that can handle large amounts of traffic without compromising performance or reliability. As a result, NodeJS and ExpressJS have become the go-to tools for web developers looking to create high-quality, dynamic web applications.

NodeJS's key features include its event-driven architecture, non-blocking I/O model, and ability to handle multiple requests simultaneously. Additionally, ExpressJS offers a range of features including middleware support, routing capabilities, and support for templating engines. These features allow developers to create custom APIs and web applications quickly and efficiently. It is important to note that the popularity of NodeJS and ExpressJS is evidenced by their large and active communities, which provide extensive documentation, resources, and tutorials for developers of all skill levels. This has helped to further cement their position as essential tools for modern web development.

4.1.2 Benefits of Using NodeJS and ExpressJS

NodeJS and ExpressJS are both open-source technologies that are widely used for building scalable and high-performance web applications. They offer a great level

of flexibility, speed, and efficiency, making them ideal for building real-time applications and APIs.

Furthermore, the event-driven architecture of NodeJS and the middleware support and routing capabilities of ExpressJS enable developers to easily create complex web applications with minimal effort. Additionally, the non-blocking I/O model of NodeJS allows for efficient handling of large amounts of data, making it suitable for use in data-intensive applications. The active communities surrounding these technologies also provide valuable support and resources to developers, ensuring that they remain relevant and up to date with modern web development practices. Overall, the benefits of using NodeJS and ExpressJS make them indispensable tools for developers looking to create efficient and scalable web applications.

4.2 NodeJS

4.2.1 Overview of NodeJS

NodeJS is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side applications using JavaScript. It is built on top of the V8 JavaScript engine and provides an event-driven, non-blocking I/O model that makes it lightweight and efficient for building scalable network applications.

NodeJS features include a rich library of various JavaScript modules that simplifies web application development, the ability to handle multiple client requests simultaneously, and support for real-time web applications with two-way connections. Additionally, it has a large and active community that contributes to its continuous improvement and development.

NodeJS is used because it allows developers to use JavaScript on both the front-end and back-end of web applications, making it a versatile and efficient tool for building full-stack applications.

Moreover, NodeJS is known for its scalability and ability to handle a large number of concurrent connections, making it a popular choice for building real-time applications such as chat applications and online gaming platforms. Its non-blocking I/O model also allows for faster processing of data, improving the overall performance of web applications.

4.2.2 NodeJS in Web Development

NodeJS is used in web development as a server-side platform for building scalable and high-performance web applications. It allows developers to write server-side code in JavaScript, making it easier to develop both the front-end and back-end of web applications using a single programming language.

NodeJS offers several benefits in web development, including faster development cycles, improved scalability and performance, and a vast library of pre-built modules and packages that can be easily integrated into applications. Additionally, NodeJS supports non-blocking I/O operations, allowing for efficient handling of large volumes of data and concurrent connections.

NodeJS is also highly compatible with cloud-based architectures, making it an ideal choice for developing applications that can be easily deployed and scaled in the cloud. Furthermore, NodeJS has a large and active community of developers, providing ample resources and support for those working with the technology.

4.3 ExpressJS

4.3.1 Overview of ExpressJS

ExpressJS is a popular and widely used web application framework for Node.js that simplifies the process of building robust, scalable, and maintainable web applications. It provides a range of features and tools for handling HTTP requests, routing, middleware, and much more.

Some of the features of ExpressJS include support for various HTTP methods, easy integration with databases, and the ability to create RESTful APIs. Additionally, it

allows for the use of middleware functions to handle requests and responses. ExpressJS is used because it simplifies the process of building web applications and APIs in Node.js, making it faster and easier to develop robust server-side applications.

It also provides a wide range of features such as routing, templating engines, and error handling, which makes it a popular choice among developers. Furthermore, its lightweight nature and flexibility make it suitable for building both small and large-scale applications.

4.3.2 ExpressJS in Web Development

ExpressJS is a popular web application framework used for building web applications and APIs. It simplifies the process of building server-side applications by providing a set of robust features and tools for creating scalable and maintainable web applications.

ExpressJS provides a minimalist framework that allows developers to create web applications quickly and efficiently. Additionally, it offers flexibility in terms of integrating with other libraries and databases, making it a popular choice for building APIs and microservices.

ExpressJS is also known for its middleware architecture, which enables developers to add functionality to their applications without having to modify the core code. This makes it easier to maintain and scale applications over time.

4.4 Implementation of NodeJS and ExpressJS in this thesis

4.4.1 Understanding the CRUD Implementation of the Back-end

In the world of software development, the back-end portion of a project plays an essential role in improving its overall performance and functionality. It focuses on creating, managing, and processing data stored in databases. The project's back-end is responsible for all business logic, storage operations, and data processing.

This section will discuss the functionalities of the back-end portion of a project, focusing specifically on the Product and User data types.

4.4.2 Product Data Type

In this thesis, the product data type represents the various products offered by the business or organization. The back-end portion of a project that implements a product data type is responsible for CRUD operations.

1. Create: This operation entails the creation of a new product and its addition to the database. This is typically done when a new product is added to the product line.
2. Read: This operation retrieves information about a specific product from the database. It is utilized frequently when displaying product information on the store website.
3. Update: This operation entails modifying the database information of an existing product. It is typically performed when a product attribute, such as the price or description, must be modified.
4. Delete: This operation involves erasing a particular product from the database. Typically, this occurs when a product is discontinued or no longer for sale.

4.4.3 User Data Type

A project's user data type represents the individuals who interact with the organization, such as customers and employees. In a project, there are two types of users: Admin and Normal User. The back-end portion of a project that implements a user data type is responsible for CRUD operations.

1. Create: This operation involves the creation of a new user and the addition of their information to the database. This is typically done when a new customer registers on the company's website or a new employee is hired.

2. Read: This operation retrieves information about a specific user from the database. When a user is logged in and their profile information is displayed, this operation is frequently employed.
3. Update: This operation entails modifying the information of an existing user in the database. It is typically performed when a user wants to update their contact information.
4. Delete: This operation deletes a particular user from the database. Typically, this occurs when a user deletes their account or when an employee leaves the company.

4.4.4 Product Service Structure

The Product service structure provides a conceptualization of the CRUD implementation, as well as establishing the connection to the database through the Product data model (see Figures 5 and 6).

```
const createProduct = async (
  product: ProductDocument
): Promise<ProductDocument> => {
  return product.save()
}

const findProductById = async (productId: string): Promise<ProductDocument> =>
{
  const foundProduct = await Product.findById(productId)

  if (!foundProduct) {
    throw new NotFoundError(`Product ${productId} not found`)
  }

  return foundProduct
}

const findAllProducts = async (): Promise<ProductDocument[]> => {
  return Product.find().sort({ name: 1 })
}
```

Figure 5. Product service structure 1

```
const updateProduct = async (
  productId: string,
  update: Partial<ProductDocument>
): Promise<ProductDocument | null> => {
  const foundProduct = await Product.findByIdAndUpdate(productId, update, {
    new: true,
  })

  if (!foundProduct) {
    throw new NotFoundError(`Product ${productId} not found`)
  }

  return foundProduct
}

const deleteProduct = async (
  productId: string
): Promise<ProductDocument | null> => {
  const foundProduct = Product.findByIdAndDelete(productId)

  if (!foundProduct) {
    throw new NotFoundError(`Product ${productId} not found`)
  }

  return foundProduct
}
```

Figure 6. Product service structure 2

The codes demonstrate the required services for the Product data type:

1. **createProduct:** receives the information of a new product and saves that new product to the database.
2. **findProductById:** receives an argument as the ID of the product and returns the information of that product. Returns an error if the product cannot be found in the database.
3. **findAllProducts:** returns all the products available in the database.
4. **updateProduct:** receives two arguments, the ID of the product to be modified, and the new data of the product that is to replace the old data of such product. Returns an error if the product cannot be found.
5. **deleteProduct:** receive one argument which is the ID of the product to be removed from the database. Return an error if the product cannot be found in the database.

4.4.5 User Service Structure

The User service structure (Figure 7) gives an abstraction of how the CRUD implementation is performed regarding the database.

```
const createUser = async (user: UserDocument): Promise<UserDocument> => {
  return user.save()
}

const findAllUsers = async (): Promise<UserDocument[]> => {
  return User.find().sort({ firstname: 1 })
}

const findUserByEmail = async (email: string): Promise<UserDocument | any> =>
{
  const foundUser = User.findOne({ email })
  if (!foundUser) {
    throw new NotFoundError(`User with email ${email} not found`)
  }
  return foundUser
}

const deleteUser = async (userId: string): Promise<UserDocument | any> => {
  const foundUser = User.findByIdAndDelete(userId)
  if (!foundUser) {
    throw new NotFoundError(`User ${userId} not found`)
  }
  return foundUser
}
```

Figure 7. User service structure

The codes demonstrate the required services for the Product data type:

1. **createUser:** receives the information of a new user and saves that new user to the database.
2. **findAllUsers:** returns all users available in the database.
3. **findUserByEmail:** receives the email address of the user as the only argument, and returns the user associated with such email address. Returns an error message if the user cannot be found in the database.
4. **deleteUser:** receives the ID of the user that needs to be deleted. Performs the deletion if the user is found, or else returns an error message.

To the limitation of my knowledge, the CRUD implementation of User Update is not in place.

4.4.6 Product Controller Structure

The Product controller structure illustrates in detail how the back-end processes front-end data and converts it into Product service requests as mentioned above.

```
export const createProduct = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { name, description, img, categories, variants, sizes } = req.body
    const product = new Product({
      name,
      description,
      img,
      categories,
      variants,
      sizes,
    })
    await ProductService.createProduct(product)
    res.json(product)
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 8. createProduct controller method

This method destructurizes the new product data in the request sent from the front-end, processes it into correct format and sends it to the **createProduct** service in **Figure 5**. This method also checks for errors such as Validation Error and other errors.

```
export const updateProduct = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const update = req.body
    const productId = req.params.productId
    const updatedProduct = await ProductService.updateProduct(productId,
update)
    res.json(updatedProduct)
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 9. updateProduct controller method

The same with **createProduct** method as mentioned above, this method also processes the data and sends it to the **updateProduct** service in **Figure 6**. This will examine the pertinent errors to ensure the data's continued integrity as well.

```
export const deleteProduct = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    await ProductService.deleteProduct(req.params.productId)
    res.status(204).end()
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 10. deleteProduct controller method

This works the same way with **createProduct** and **updateProduct** methods above. This method provides processed data passed from the front-end request to the **deleteProduct** service in **Figure 6**.

```
export const findProductById = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    res.json(await ProductService.findProductById(req.params.productId))
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 11. findProductById controller method

This method will take in the request which asks for the Product ID and then look for that unique Product item in the database and return that Product item if found. Otherwise, it will display an error message indicating the unavailability of such Product item. This method will also notify if an error comes from a bad request.

```
export const findAllProducts = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    res.json(await ProductService.findAllProducts())
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 12. findAllProducts controller method

This method will return all the Product items available in the database.

4.4.7 User Controller Structure

The User controller structure illustrates in detail how the back-end processes front-end data and converts it into User service requests.

```
export const deleteUser = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    await UserService.deleteUser(req.params.userId)
    res.status(204).end()
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 13. deleteUser controller method

This method will receive the request which asks for the User ID and look for that user in the database. Once found, the method will carry out the deletion of such user with the deleteUser service in **Figure 7**.

```
export const createUser = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { email, firstname, lastname, role } = req.body
    const user = new User({
      email,
      firstname,
      lastname,
      role,
    })
    await UserService.createUser(user)
    res.json(user)
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 14. createUser controller method

This method will receive a request which contains the information of a new user, send to the **createUser** service as stated in **Figure 7** for the creation of a new user.

```
export const findAllUsers = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    res.json(await UserService.findAllUsers())
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 15. findAllUsers controller method

This method will return all the users available in the database.

```
export const findUserByEmail = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    res.json(await UserService.findUserByEmail(req.params.email))
  } catch (error) {
    if (error instanceof Error && error.name == 'ValidationError') {
      next(new BadRequestError('Invalid Request', error))
    } else {
      next(error)
    }
  }
}
```

Figure 16. findUserByEmail controller method

This method will receive a request which asks for an email associated with the user we want to find, send to the **findUserByEmail** service as stated in **Figure 7** and return that user if available. Error handling is also carried out.

4.4.8 Authentication at Back-end Level

This back-end level authentication will receive the login information from the front-end login request and verify the user with the support of JSON web token.

```
export default function authentication(
  req: Request,
  res: Response,
  next: NextFunction
) {
  try {
    const auth = req.headers.authorization || ''
    const token = auth.split(' ')[1]

    const verifiedToken = jwt.verify(token, JWT_SECRET)
    req.user = verifiedToken
    next()
  } catch (error) {
    console.log(error)
    next(error)
  }
}
```

Figure 17. Back-end authentication

JSON web token (JWT) is a compact and secure way of transmitting information between parties as a JSON object. It can be used for authentication, authorization, and exchanging information. There are three components: a header, a payload, and a signature.

The token's encryption algorithm, such as HMAC SHA256 or RSA, is specified in the token's header. The payload contains the transferred claims or data, such as username or role. The signature is used to confirm that the message has not been altered and that the sender is who they claim to be.

The server generates a JWT with the user's information, signs it with a secret key, and sends it to the client when the user logs in. The client stores the JWT and includes it in subsequent requests to the server's Authorization header. The server can then decode and validate the JWT to determine if the user is authorized to access the requested resource as illustrated in **Figure 17**.

JWT is a more secure and efficient method of authentication and authorization that eliminates the need to repeatedly query the database for user credentials.

4.4.9 Product and User router structures

After we define how the controllers for the Product and User data types, we now put them together in different types of HTTPS requests, each request will have its own URL and HTTPS method.

```
const userRouter = express.Router()

userRouter.post('/', createUser)
userRouter.get('/', findAllUsers)
userRouter.get('/:email', findUserByEmail)
userRouter.delete('/:userId', deleteUser)
```

Figure 18. userRouter method

```
import authentication from '../middlewares/authentication'

const productRouter = express.Router()

productRouter.get('/', authentication, findAllProducts)
productRouter.get('/:productId', findProductById)
productRouter.put('/:productId', updateProduct)
productRouter.delete('/:productId', deleteProduct)
productRouter.post('/', createProduct)
```

Figure 19. productRouter method

We use the GET method with the necessary authentication for the Read operation of CRUD. We utilize POST for the Create operation and PUT for the Update operation. The DELETE method must be used for Delete operation.

4.4.10 Login with Google Passport

```
const loginWithGoogle = () => {
  return new GoogleStrategy(
    {
      clientID: process.env.GOOGLE_CLIENT_ID,
    },
    async (parsedToken: any, googleID: any, done: any) => {
      try {
        let user = await
UserService.findUserByEmail(parsedToken.payload.email)
        if (!user) {
          user = {
            email: parsedToken.payload.email,
            firstname: parsedToken.payload.given_name,
            lastname: parsedToken.payload.family_name,
            role: isAdmin(parsedToken.payload.hd) ? Role.ADMIN : Role.USER,
          } as UserDocument
          const newUser = new User(user)
          await UserService.createUser(newUser)
        }
        done(null, user)
      } catch (error) {
        done(error)
      }
    }
  )
}
```

Figure 20. Logging in with Google Passport

This method of logging in with Google Passport support will assist in determining the various authorization levels of user accounts, such as administrators and regular users. In the context of this thesis, authorization will be based on the email address associated with the user's Google account during the login step.

If the user's email address ends in the integrify.io domain, the administrator role will be assigned. The remaining email addresses will be assigned as regular users.

4.4.11 Setting up the Main Application

Now that we have developed all the necessary APIs, it is time to integrate them together into a complete application for our server.

```
const app = express()

// Express configuration
app.set('port', process.env.PORT || 3000)

// Global middleware
app.use(apiContentType)
app.use(express.json())
app.use(cors())

// Use Google passport for login
app.use(passport.initialize())
passport.use(loginWithGoogle())
app.post(
  '/google-login',
  passport.authenticate('google-id-token', { session: false }),
  (req, res) => {
    const user = req.user as { email: string; role: string }
    const token = jwt.sign(
      {
        email: user.email,
        role: user.role,
      },
      JWT_SECRET,
      { expiresIn: '1h' }
    )
    res.json({ message: 'login done', token })
  }
)

// Set up routers
app.use('/api/v1/users', userRouter)
app.use('/api/v1/products', productRouter)

// Custom API error handler
app.use(apiErrorHandler)
```

Figure 21. Back-end main application

As is evident, the application will use Google Passport for authentication rather than standard email addresses and passwords. Users will only need one Google account to log in and gain access to Google Passport-supported services. This is crucial for passwordless authentication because it enhances security and the user experience. **Figure 21** shows how the back-end is implemented with Google Passport Authentication mechanism.

4.4.12 Running the Server

We will connect the back-end to MongoDB database and gain access to the Product and User data stored there. Simultaneously, the server will listen to the requests made from the front-end with appropriate APIs call and will reply with corresponding responses.

```
mongoose
  .connect(mongoUrl, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useFindAndModify: false,
    useCreateIndex: true,
  })
  .then(() => {
    logger.info('Connected to MongoDB')
  })
  .catch((err: Error) => {
    console.log(
      'MongoDB connection error. Please make sure MongoDB is running. ' + err
    )
    process.exit(1)
  })

if (process.env.NODE_ENV === 'development') {
  app.use(errorHandler())
}

// Start Express server
app.listen(app.get('port'), () => {
  console.log(
    '  App is running at http://localhost:%d in %s mode',
    app.get('port'),
    app.get('env')
  )
  console.log('  Press CTRL-C to stop\n')
})
```

Figure 22. Get the server up and running

5 THE FRONT-END SIDE

5.1 Introduction

5.1.1 Definition of Front-end Development

Front-end development refers to the process of creating a website's or application's user interface and visual design. It involves using languages such as HTML, CSS, and JavaScript to create visually appealing and functional interfaces that are easy to navigate and interact with for users. Front-end development is an essential part of web development as it is responsible for the look and feel of a website or application. It also plays a crucial role in enhancing user experience by ensuring that the interface is responsive and accessible across different devices.

Front-end developers work closely with designers and back-end developers to ensure that the website or application is visually appealing, functional, and easy to use. They also need to stay up to date with the latest technologies and trends in web development to create modern and innovative user interfaces. In addition, front-end developers may also be responsible for optimizing the website or application for search engines and ensuring that it is accessible to users with disabilities. This requires a deep understanding of web standards and best practices.

5.1.2 Overview of React, Redux, and SASS

React is a JavaScript library used for the development of user interfaces. It enables developers to create UI components that are reusable and efficiently update the interface when data changes.

Redux is a state management library that enables developers to predictably manage application state. It offers a centralized repository for all application data and operations, making it simple to manage complex state and update the interface accordingly.

SASS (Syntactically Awesome Style Sheets) is a CSS pre-processor that enables developers to compose more modular and efficient CSS code. It offers capabilities,

such as variables, mixins, and nesting, that simplify the styling process and reduce code duplication. Together, these technologies provide front-end developers with a potent toolkit for creating dynamic and responsive user interfaces.

5.2 React

React is a JavaScript library used to create user interfaces. Developers can make modular UI elements that can be used in multiple places, and the UI can be quickly refreshed when data changes. React was developed by Facebook and is widely used in web development. It allows for the creation of complex and interactive user interfaces with ease.

React is based on a component-based architecture, making it easy to reuse code and maintain applications. React is highly performant due to its virtual DOM implementation, which allows for efficient updates and rendering of components. Additionally, React's popularity has led to the creation of many third-party libraries and tools that can further enhance development productivity. It also has a large and active community, providing developers with ample resources and support.

React becomes an ideal choice for beginners who want to get started with programming quickly and efficiently. Additionally, its versatility and scalability make it a popular choice for building complex applications and software systems. Overall, React offers a straightforward and effective method for developing user interfaces, making it an ideal choice for web developers.

In the field of web development, React provides some features that improve performance, security, and scalability, as follows:

5.2.1 Declarative Approach

React employs a declarative approach to constructing user interfaces, in which the desired state of the application is described and React handles updating the UI.

- This approach facilitates easier debugging and testing, as well as enhanced performance due to React's ability to update only the necessary components when changes occur.
- Developers can concentrate on the logic of the application rather than manually manipulating the DOM, resulting in more efficient and predictable development.

5.2.2 Component-based

Component-based React enables the reuse of existing code and the construction of complex user interfaces from simple building blocks called components.

- React encourages component reusability by enabling the developer to create a library of reusable components that can be used across the entire website.
- This assists in reducing development time and enhancing code organization and allows for easier maintenance and updates of software systems by isolating and modularizing specific functionalities.

5.2.3 Virtual DOM

React uses a virtual DOM instead of the actual DOM to render updates. This allows React to make changes without requiring a page reload. Utilizing a virtual DOM enables the interface to be efficiently updated by rendering only the modified components.

- React's virtual DOM allows for efficient updates and rendering, resulting in improved performance compared to traditional methods. This makes React a popular choice for building high-performance web applications.
- Cross-site scripting attacks are prevented by React's virtual DOM, making it a secure option for web development. Moreover, React's one-way data binding reduces the risk of data manipulation and enhances security.
- The virtual DOM of React enables the efficient rendering of components, resulting in enhanced scalability and performance for web applications.

5.2.4 Server-Side Rendering

React supports server-side rendering, which generates the initial HTML on the server before sending it to the client. This significantly improves the website's performance, as the client can begin rendering without waiting for the initial HTML to load.

- Server-side rendering also improves search engine optimization (SEO) by providing search engines with fully rendered HTML content to crawl and index, rather than relying on JavaScript to generate the content dynamically on the client-side.

5.2.5 Optimization

React provides numerous tools and libraries to assist developers in writing optimized code. This results in an overall faster and more efficient website.

- Among these tools are React.memo, which memoizes components to prevent unnecessary re-renders, and the use of hooks, which enables improved state management and code organization.
- By utilizing these tools, developers can create high-performance websites with an enhanced user experience.

5.2.6 Performance Monitoring

React provides excellent performance monitoring tools that enable developers to identify performance issues and optimize a website's performance.

- These tools enable developers to monitor the performance of a website and make the necessary adjustments to enhance its speed and efficiency.
- Using the performance monitoring tools provided by React, developers can ensure that their website is operating efficiently and providing a positive user experience.

5.2.7 Community

There is a large and active community of developers who use React, so documentation, tutorials, and support are readily available.

- This makes it easier for developers to learn and troubleshoot issues they may encounter while using React. Additionally, the community regularly contributes new libraries and tools to enhance the development experience.
- This means that developers can easily find solutions to common problems and have access to a wide range of resources that can help them build better applications. Moreover, being part of such a vibrant community allows developers to stay up-to-date with the latest trends and best practices in React development.
- Moreover, the community also organizes conferences and meetups to share knowledge and network with other React developers. These events provide a great opportunity for developers to stay up-to-date with the latest trends and best practices in React development.

5.2.8 Integration

React is simple to integrate with other libraries and frameworks, making it a versatile option for a wide variety of projects.

This means that developers can easily incorporate React into their existing projects without having to completely overhaul their codebase. The flexibility of React also allows for easy collaboration between teams with different technology stacks.

5.3 Redux

Redux is a popular JavaScript state management library inspired by the Flux architecture. It offers a predictable state container that can manage and update an application's state. Redux is compatible with all front-end frameworks and libraries, such as React, Angular, and Vue. It centralizes and predictably updates the state, making it easier to analyze and debug the application. Redux also facilitates easier

data flow and state management across an application's various components, thereby enhancing the overall performance of the application. Redux is widely used by web developers because it provides a more efficient and reliable method for managing an application's state.

In addition, Redux facilitates testing and debugging of an application's state changes, making it a popular choice for large-scale projects. Its popularity has also resulted in the creation of numerous tools and libraries that enhance its functionality. Redux DevTools, which provides a visual representation of the state changes in real-time, and Reselect, which enables the efficient computation of derived data from the Redux store, are examples of these tools. These features make Redux a potent state management tool for modern web applications.

5.3.1 Ability to View Application's State

Redux is a robust library for state management that enables developers to create complex and scalable applications. The ability to view application state is one of the most important features of Redux for developers during the application development, debugging, and testing processes.

Using Redux DevTools, developers are able to monitor and debug the real-time state of their application. Redux DevTools provides a visual representation of the state tree, allowing developers to track the evolution of the application's state. This visibility into the application's state enables developers to identify problems, debug errors, and optimize performance.

Redux's application state view enables developers to implement efficient and effective testing strategies. By comparing the current state of the application to its previous states, developers can track the application's evolution over time and identify potential issues before they become significant problems.

In addition, the ability to view application state in Redux enables developers to optimize application performance. By monitoring the state tree, developers can identify redundant or inefficient code and improve the application's performance.

Overall, the ability to view application state in Redux is a crucial feature that enables developers to create more effective, scalable, and streamlined applications. Redux DevTools enables developers to easily monitor, debug, and optimize their applications, resulting in a better user experience and enhanced performance.

5.3.2 Reduction of Boilerplate Code

Redux is a JavaScript state container with predictable behavior. It is commonly employed in web and mobile applications to manage and store application state in a central location. Redux reduces the amount of boilerplate code needed to manage the state, which is one of its most notable benefits. The codebase can be simplified and enhance the performance of an application using Redux.

Managing application state in traditional web and mobile applications can be arduous and difficult. To manage the application's state, developers must write a significant amount of boilerplate code. This code can quickly become complex and difficult to maintain, making it difficult to debug and resolve issues.

Redux significantly reduces the state management code, resulting in code that is simpler to read, understand, and maintain. Redux accomplishes this by introducing a few straightforward concepts that simplify the code.

Redux begins by introducing a centralized store that holds the application's state. This implies that the entire application's state is stored in a single location, making it easier to manage and manipulate.

Redux then uses a simple set of rules to update the state of the application. Create actions that describe what should occur in the state. The store is then notified of these actions, which trigger a reducer function that modifies the state based on the action.

Finally, Redux provides a collection of tools and libraries that automate a number of repetitive state management tasks. Redux, for instance, offers libraries such as React-Redux that simplify the integration of Redux with React applications.

In conclusion, utilizing Redux to manage the state of web and mobile applications reduces boilerplate code. This facilitates codebase readability, comprehension, and maintenance. In addition, Redux offers a collection of tools and libraries that automate many of the repetitive tasks associated with state management, thereby reducing the need for boilerplate code.

5.3.3 Improved Debugging

Debugging is one of the primary advantages of utilizing Redux. Due to the dispersed state across various components in conventional web applications, debugging can be difficult. However, Redux provides a centralized state management system, making code debugging easier.

Redux makes debugging easier because all state changes take place in an individual location, the Redux store. Having only one source of truth makes it simple to determine where the code is failing and permits developers to monitor the application's state at all times.

In addition, some of the most effective debugging tools have been created specifically for Redux. The Redux DevTools Extension, for instance, provides a multitude of useful features, such as time-travel debugging, which enables developers to view all past actions and state changes in their application.

Additionally, developers can easily view and analyze an application's entire state history, allowing them to determine where a problem may have occurred. The use of middleware and time-travel debugging tools further enhances the developer's ability to diagnose and fix issues in web applications.

In conclusion, Redux improves the debugging process for web applications by providing centralized state management and essential tools. This makes it easier for developers to identify and resolve issues more quickly and create applications of higher quality.

5.3.4 Storing the State in a Single Object

Redux's ability to store state in a single object simplifies the management of application state, which is one of its primary advantages. By storing all application states in a specific location, it becomes easier to track, manage, and share this state among the application's modules and components.

Another advantage of this method is that it simplifies the implementation of time-travel debugging. Because Redux stores all state changes in a single object, it is simple to roll back to a previous state and debug any application issues that may have arisen.

Additionally, storing application state in a single object can reduce the number of API requests required. Instead of making repeated requests for data that has already been loaded, Redux can store this data in the state object and retrieve it when necessary.

Redux's ability to store state in a single object has the potential to vastly improve the efficiency and effectiveness of application state management. By reducing complexity and streamlining the debugging process, developers are able to concentrate on creating better applications more quickly.

5.4 SASS

SASS (Syntactically Awesome Style Sheets) is a preprocessor scripting language that streamlines the creation of CSS stylesheets. It adds features such as variables, nesting, mixins, and functions to CSS, making stylesheet scripting more organized and efficient. It is interpreted or compiled into CSS, enabling developers to write code that is more efficient and maintainable.

5.4.1 Improved Readability and Maintainability

SASS is a preprocessor for CSS that offers features to make coding stylesheets more efficient and organized. It introduces a set of features that improve the ability to maintain code in several ways.

- **Reusability:** With SASS, code fragments can be constructed that are reusable and can be shared across multiple stylesheets. This saves effort and makes the code easier to maintain.
- **Modular code:** SASS enables the writing of code that is modular and reusable across multiple project components. Codebase-wide functions, variables, and mixins can be generated. This eliminates redundant code and ensures uniformity throughout the codebase.
- **Reduced Code:** SASS enables more concisely and effectively written code. This reduces the quantity of code to be written, resulting in a faster and more responsive application.
- **Code Organization:** SASS enables you to have more logically and systematically organized code. This facilitates the maintenance and updating of stylesheets.
- **Variables:** Variables can be defined that can be utilized throughout the stylesheets with SASS. This facilitates code maintenance and increases its flexibility.
    - SASS permits the use of variables, so values that are repeated throughout the stylesheet can be designated to variables and utilized consistently throughout the document. This makes updating designs simpler and reduces the likelihood of making mistakes.
    - SASS enables the storing of values such as colors, font sizes, and spacing in variables. This makes it simpler to update these values across the codebase, as the variable needs to be changed only once as opposed to each instance of the value.
- **Mixins:** SASS enables the creation of mixins that are reusable throughout stylesheets. This facilitates code maintenance and ensures that designs are uniform.
    - Mixins are parts of code that can be written once and used multiple times. This makes the code simpler and more modular.
    - SASS permits the creation of mixins, which are reusable sections of code used to apply styles to various elements. This makes it simple

to implement consistent styles throughout the codebase without duplicating code.

- **Inheritance:** SASS supports inheritance, so styles can inherit the properties of other styles. This eliminates code redundancy and assures uniformity throughout the stylesheet.
    - SASS enables the sharing of styles between selectors using inheritance. This decreases the quantity of code that must be written and makes the code easier to maintain.
- **Comments:** Comments can be utilized in SASS to provide context and explanation for the code. This makes it simpler for other developers to comprehend the code's functionality.

5.4.2 Ability to Create Variables and Functions

SASS is a CSS pre-processor that enables developers to more efficiently and effectively write CSS code. The ability to create variables and functions is one of the primary benefits of SASS.

Variables enable programmers to assign a value to a name and reuse it throughout the code. This makes it easier to modify styles in the future, as the variable value only needs to be changed rather than manually updating each instance of the value.

SASS's functionality allows developers to group CSS styles into reusable functions. This can facilitate the streamlining and cleaning of code, making it easier to manage and more effective. SASS functions can accept arguments and return a value, allowing for the creation of more complex and dynamic styles.

In conclusion, variables and functions in SASS provide a more efficient method for writing CSS code, allowing developers to reuse styles and streamline workflow.

5.4.3 Ability to Nest CSS

The ability to nest CSS rules is one of the most useful features of SASS, as it allows selectors to be nested within each other and styles for child elements to be written within the block of the parent element.

SASS enables nesting of CSS selectors, making the code more readable and understandable. This also reduces the amount of code that must be written, as the parent selector is not repeated for each child selector. SASS permits the nesting of rules, allowing related styles to be grouped together. This establishes a visual hierarchy that makes it easier to determine which styles apply to particular elements.

Using nesting in SASS can make the code more organized and readable, as the nesting structure clearly indicates the HTML element hierarchy. It also simplifies styling, as developers can write styles for particular elements without worrying about specificity issues, as SASS generates the appropriate CSS selectors automatically.

5.5 Implementation of Front-end technologies in this thesis

5.5.1 The Overall Structure of the Front-end

The **index.tsx** file contains the **App.tsx** file, which is nested within the Redux store provider and theme provider by **React Context**. The Redux provides an efficient method for managing all the application's global states. The **Context** manages a simple status of the application, which in this case is the page theme. This file also contains the **Router** from React Router DOM, which defines all paths required for navigation and HTTPS requests.

The **App.tsx** file includes the **AppBar.tsx**, **Drawer.tsx**, and **Router.tsx** components. These components are responsible for rendering the navigation bar, displaying the menu drawer, and routing to various application pages.

The **Router.tsx** is responsible for defining all application routes and rendering the appropriate pages as React components based on the current URL. It would also handle any navigation logic required, such as redirecting the user to a different page or displaying an error message.

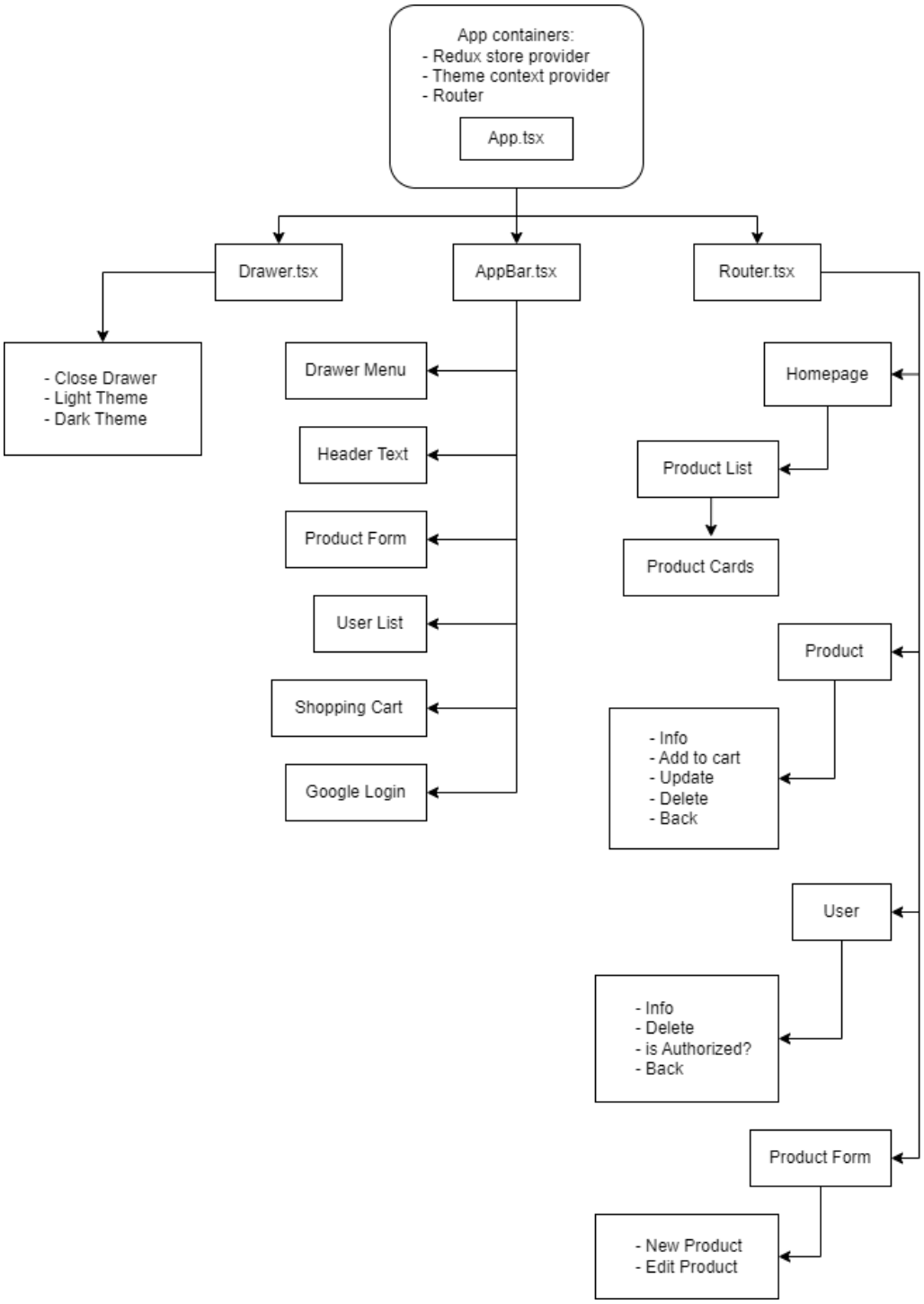

Figure 23. The structure of the front-end

5.5.2 HTTPS Requests to Back-end

The connections to the back-end are among the essential features of a successful front-end interface. The front-end provides a platform for displaying data from the back-end, as well as tools and functions for user interaction with the data. A successful connection to the back-end guarantees a streamlined workflow between users and the data warehouse.

```
const baseUrl = "http://localhost:5000/api/v1/users";

const getAllUsers = async () => {
  try {
    const res = await axios.get(baseUrl);
    return res.data;
  } catch (error) {
    console.log(error);
  }
};

const deleteAUser = async (userId: string) => {
  try {
    const res = await axios.delete(`${baseUrl}/${userId}`);
    return res.data;
  } catch (error) {
    console.log(error);
  }
};
```

Figure 24. Front-end User API

```
export const clientId =
  (process.env.GOOGLE_CLIENT_ID as string) ||
  "7367156687-ci6cn59gllt698sjpklf2c8v7a6lh4ji.apps.googleusercontent.com";

export const handleLogin = async (res: any) => {
  const tokenId = res.credential;
  const response = await axios.post(
    "http://localhost:5000/google-login",
    {},
    {
      headers: {
        Authorization: `Bearer ${tokenId}`,
      },
    }
  );
  const token = await response.data.token;
  const decodedToken = jwt_decode(token) as { [key: string]: any };
  return { token, user: decodedToken };
};
```

Figure 25. Front-end Login APIs

```
const baseUrl = "http://localhost:5000/api/v1/products";

const getAllProducts = async (token: string) => {
  try {
    const res = await axios.get(baseUrl, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    return res.data;
  } catch (error) {
    console.log(error);
  }
};

const createNewProduct = async (newProduct: Product) => {
  try {
    const res = await axios.post(baseUrl, newProduct);
    return res.data;
  } catch (error) {
    console.log(error);
  }
};

const editAProduct = async (productId: string, newProduct: Product) => {
  try {
    const res = await axios.put(`${baseUrl}/${productId}`, newProduct);
    return res.data;
  } catch (error) {
    console.log(error);
  }
};

const deleteAProduct = async (productId: string) => {
  try {
    const res = await axios.delete(`${baseUrl}/${productId}`);
    return res.data
  } catch (error) {
    console.log(error);
  }
};
```

Figure 26. Front-end Product APIs

When we have created all the necessary APIs to connect the front-end to the back-end APIs, we can define different user roles to determine which types of users will have access to particular APIs.

```
export const RBAC_RULES = {
  admin: {
    view: ["dashboard"],
    actions: [
      "products:get",
      "product:post",
      "product:edit",
      "product:delete",
      "user:findAll",
      "user:findByEmail",
      "user:delete",
    ],
  },
  user: { view: ["homepage"], actions: ["products:get"] },
};
```

Figure 27. Roles and their own authorizations

```
const check = (rules: any, role: any, action: any) => {
  const permissions = rules[role];

  if (!permissions) {
    return false;
  }

  const staticPermissions = permissions.view;
  if (staticPermissions && staticPermissions.includes(action)) {
    return true;
  }

  const dynamicPermissions = permissions.actions;
  if (dynamicPermissions) {
    const permissionCondition = dynamicPermissions.includes(action);
    if (!permissionCondition) {
      return false;
    }
    return true;
  }
  return false;
};

const Can = ({ role, perform, yes, no }: any) => {
  return check(RBAC_RULES, role, perform) ? yes() : no();
};
```

Figure 28. Methods to check roles and return corresponding authorizations

Now that we have covered all required APIs, we can move on to retrieving and saving data to the global state of the web application using such APIs. To accomplish this, we must define the global state of the application using Redux.

5.5.3 Redux and Global State Management of the Application

To store the application's global states, we must initially define a "store" as shown in **Figure 29**.

```
const store = configureStore({
  reducer: {
    user: userReducer,
    cart: cartReducer,
    product: productReducer,
    search: searchReducer,
    userList: userListReducer,
  },
});
```

Figure 29. Global store of the application

The store consists of multiple reducers, each of which defines an application component state in order to manage each application data type.

```
const initialState: Product[] = [];

const cartReducer = createSlice({
  name: "cart",
  initialState,
  reducers: {
    addToCart(state, action) {
      state.push(action.payload);
    },
    removeFromCart(state, action) {
      return state.filter((product) => product.id !== action.payload.id);
    },
  },
});
```

Figure 30. Shopping cart reducer

This shopping cart reducer manages the state of the shopping cart, allowing the user to add and remove items from the cart as well as alter existing items.

```
const searchReducer = createSlice({
  name: "search",
  initialState: "",
  reducers: {
    setSearchKeyword(state, action) {
      return action.payload;
    },
  },
});
```

Figure 31. Search bar reducer

This search reducer controls the state of the web application's search bar, enabling users to search for product items by name.

```
const initialState: User = {
  token: "",
  user: {},
};

const userReducer = createSlice({
  name: "user",
  initialState,
  reducers: {
    setUser(state, action) {
      return action.payload;
    },
  },
});

export const handleLoginSuccess = (res: any) => {
  return async (dispatch: any) => {
    const user = await handleLogin(res);
    dispatch(setUser(user))
  };
};
```

Figure 32. Single User reducer for current User login

This reducer assists the user in logging in and storing their login credentials so they can be used in various web application authorizations.

```
const initialState: User[] = [];

const userListReducer = createSlice({
  name: "userList",
  initialState,
  reducers: {
    setUsers(state, action) {
      return action.payload;
    },
    deleteUser(state, action) {
      return state.filter((user) => user.id !== action.payload);
    },
  },
});

export const getAllUsers = () => {
  return async (dispatch: any) => {
    const users = await userListService.getAllUsers();
    dispatch(setUsers(users));
  };
};

export const deleteAUser = (userId: string) => {
  return async (dispatch: any) => {
    await userListService.deleteAUser(userId);
    dispatch(deleteUser(userId));
  };
};
```

Figure 33. UserList reducer

This reducer saves all users available in the database, compiles them into a list, and displays it to the admin user responsible for deleting users of normal role.

Next, we will consider the Product reducer.

This reducer retrieves data from all Product service APIs and stores it in the front-end's global state. The ability to view all product items is accessible to all user roles, but the ability to add, delete, or modify product items is restricted to admin users.

```
const initialState: Product[] = [];
const productReducer = createSlice({
  name: "product",
  initialState,
  reducers: {
    setProducts(state, action) {
      return action.payload;
    },
    updateAProduct(state, action) {
      return state.map((product) =>
        product.id !== action.payload.id ? product : action.payload
      );
    },
    deleteAProduct(state, action) {
      return state.filter((product) => product.id !== action.payload);
    },
    createAProduct(state, action) {
      state.push(action.payload);
    },
  },
});

export const setAllProducts = (token: string) => {
  return async (dispatch: any) => {
    const products = await productService.getAllProducts(token);
    dispatch(setProducts(products));
  };
};
export const createProduct = (product: any) => {
  return async (dispatch: any) => {
    const createNew = await productService.createNewProduct(product);
    dispatch(createAProduct(createNew));
  };
};
export const editProduct = (productId: string, product: any) => {
  return async (dispatch: any) => {
    const editedProduct = await productService.editAProduct(productId,
product);
    dispatch(updateAProduct(editedProduct));
  };
};
export const deleteProduct = (productId: string) => {
  return async (dispatch: any) => {
    await productService.deleteAProduct(productId);
    dispatch(deleteAProduct(productId));
  };
};
```

Figure 34. Product reducer

5.5.4 index.tsx and App.tsx

In our project, **index.tsx** and **App.tsx** are the two files that serve as first building blocks of the web application, where it leads to all other pages and components.

```
const WithProvider = () => (
  <Provider store={store}>
    <ThemeProvider>
      <Router>
        <App />
      </Router>
    </ThemeProvider>
  </Provider>
);

ReactDOM.render(<WithProvider />, document.getElementById("root"));
```

Figure 35. index.tsx file

```
function App() {
  const { theme } = useContext(ThemeContext);

  const [drawerState, setDrawerState] = useState(false);
  const handleDrawerState = (state: boolean) => {
    setDrawerState(state);
  };

  return (
    <div className={`App ${theme}`}>
      <AppBar drawerState={drawerState} onClick={handleDrawerState} />
      <Drawer state={drawerState} onClick={handleDrawerState} />
      <Router />
    </div>
  );
}
```

Figure 36. App.tsx file

The Redux store, Theme provider, and Router are all global states that are implemented by the **index.tsx** file. Wherever users navigate inside the web application, these statuses will remain constant throughout the entire web application. All additional components and pages will appear to be located to the **App.tsx** file, which serves as the starting location of the web application.

5.5.5 Theme provider

```
export const ThemeContext = createContext<any>({ theme: "light", undefined });

export const ThemeProvider: React.FC<{}> = ({ children }) => {
  const [theme, setTheme] = useState("light");
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

Figure 37. Theme Provider

The complete web application's theme can be changed thanks to the Theme Provider, which functions as a Context Hook. The two themes are light and dark.

5.5.6 Router

```
const Router = () => {
  return (
    <Routes>
      <Route path="/" element={<Homepage />} />
      <Route path="/products/:id" element={<Product />} />
      <Route path="/user/" element={<User />} />
      <Route path="/form" element={<Form />} />
      <Route path="/form/products/:id" element={<Form />} />
    </Routes>
  );
};
```

Figure 38. Router

The Router component is accountable for directing the URL input to the appropriate API call. Each route will navigate to its proper React component.

5.5.7 Pages in the Application

There are four main pages in the application:

- **Homepage.tsx**: This page redirects to the component that displays the entire product catalog.
    - **ProductList.tsx**: This component contains all the products.

    - **ProductCard.tsx**: This element provides concise information about a single product item. The component only displays some information about the product so as to make the **ProductCard.tsx** component fit nicely in the list.
- **Product.tsx**: This page displays information about a particular product.
- **User.tsx**: This page displays the whole list of available users. Administrators can also delete standard users.
- **Form.tsx**: This page serves as a form for adding a new product item or updating an existing one.

The **AppBar.tsx** component is always present at the top of every web page on every page. It serves as the navigation bar, allowing users to navigate to various pages. It also includes the search dialog, which allows users to search for product items using a particular keyword, and the shopping cart, which contains the product items that users wish to purchase.

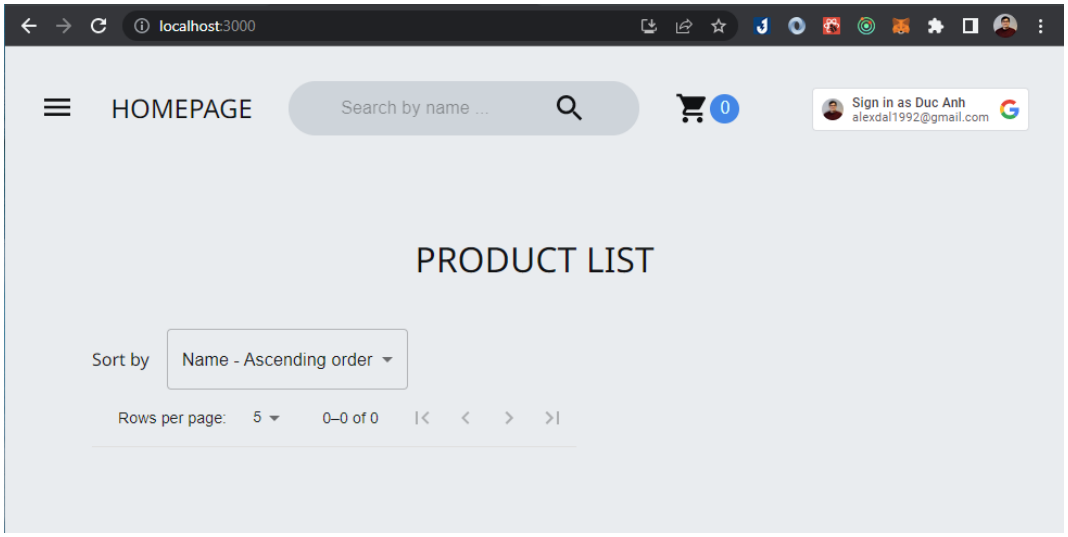

Figure 39. AppBar components at the top of the page

When the user is not logged in, the product list is not displayed.

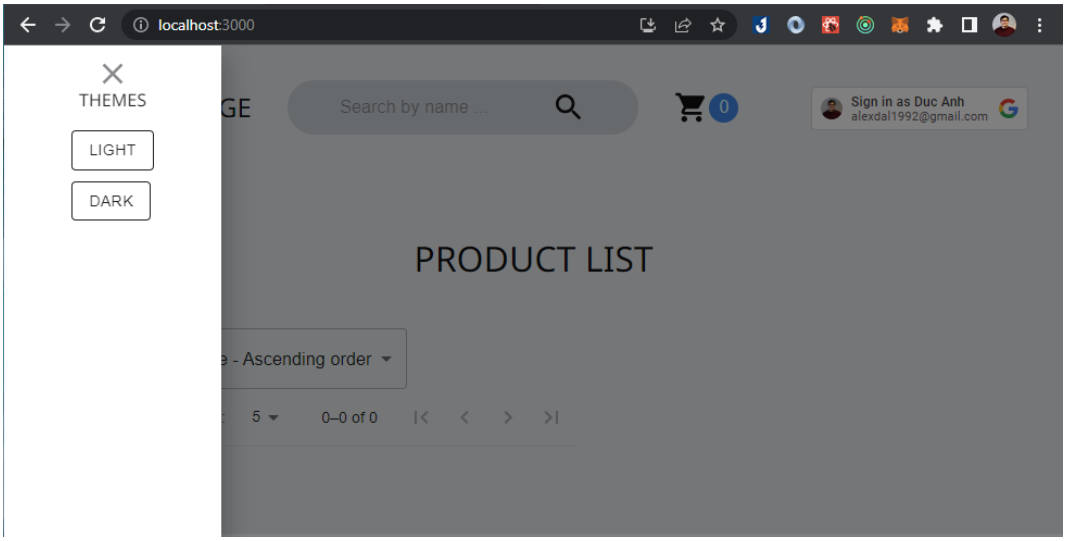

Figure 40. Light theme mode

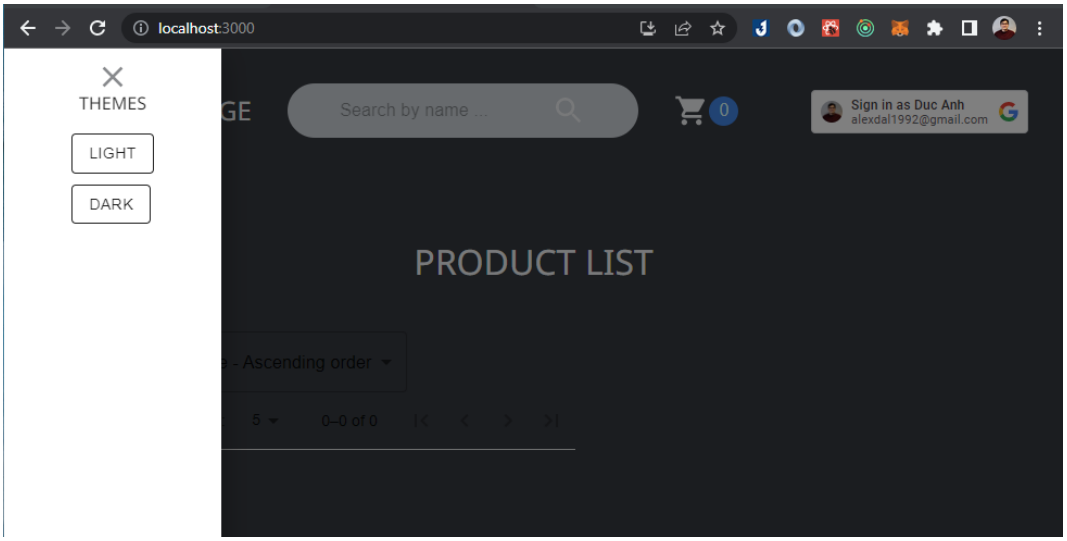

Figure 41. Dark theme mode

The web application permits Google users to log in with their Google credentials, eliminating the need to register separate accounts. The web application will assign the role of administrator to Google account emails associated with the **integrify.io** domain, while all other Google account emails will be assigned the role of normal user.

This is how the page looks like when a normal user is logged in:

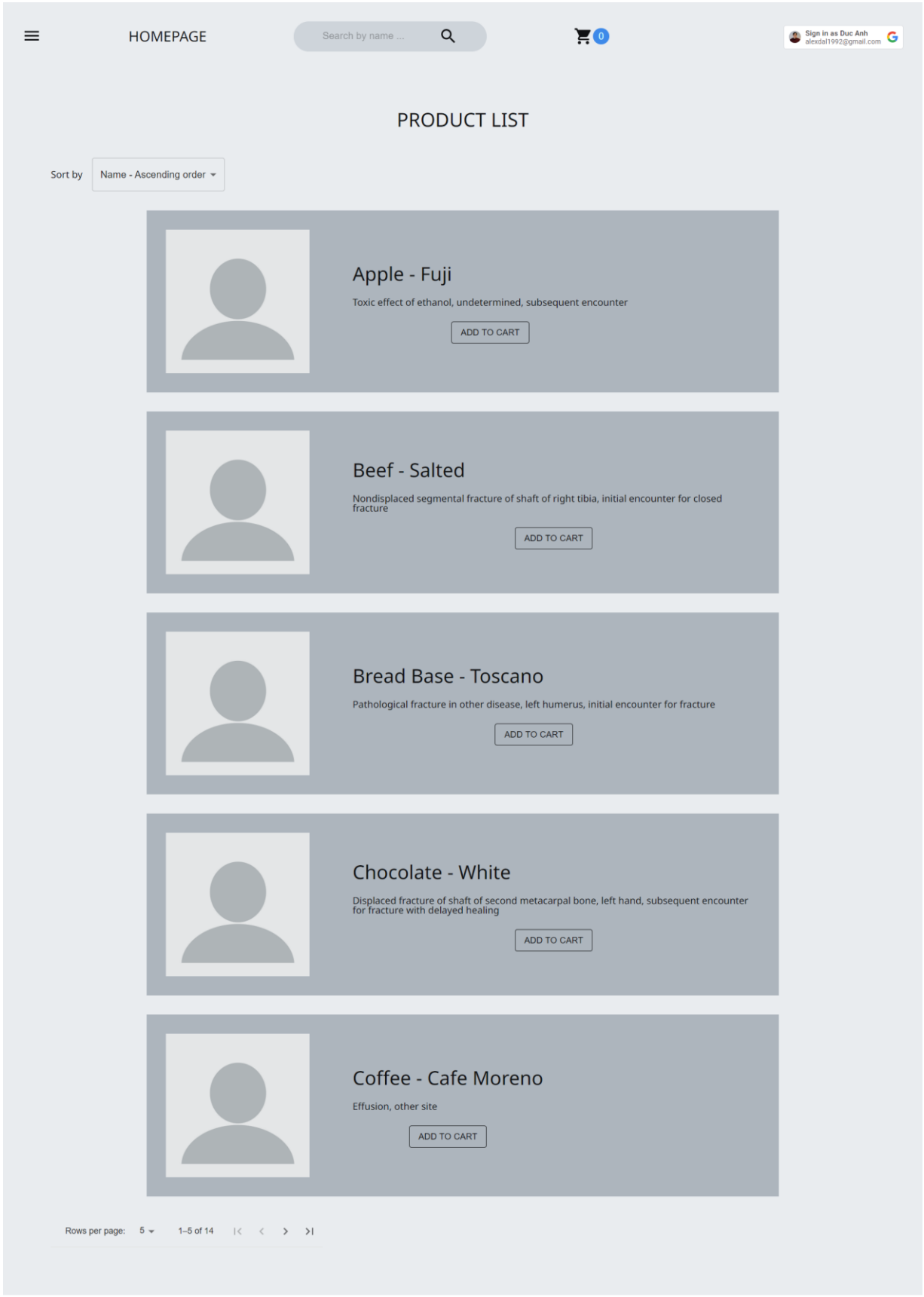

Figure 42. Page layout at Homepage

When you click on an item, you will be taken to a page containing the item's specific information. This page is also the product page.

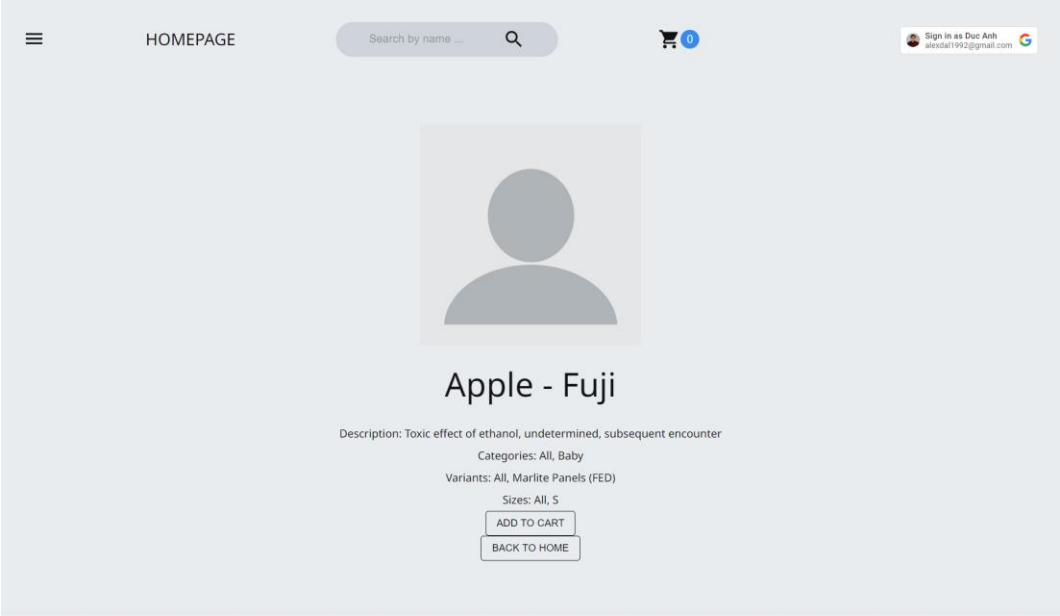

Figure 43. Specific information for an item

An item can be added to the shopping cart from either the homepage or the product page, and the cart will be updated everywhere.

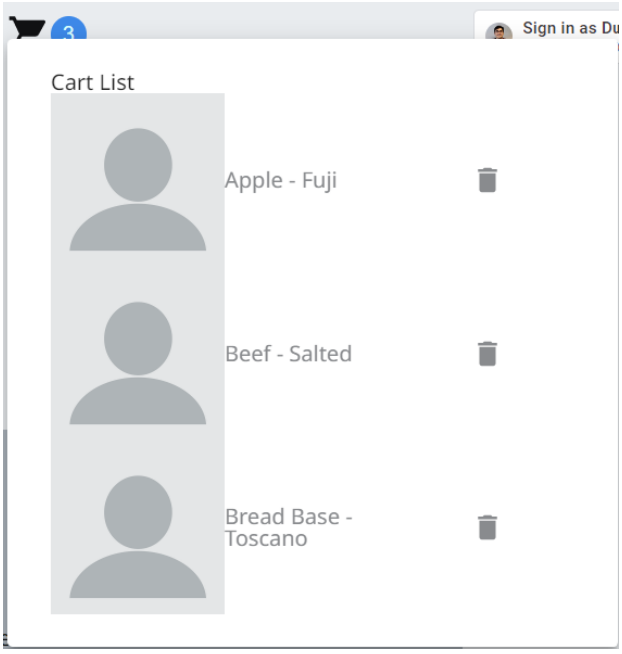

Figure 44. Shopping cart with products inside

When a product is searched with the search bar, it will show the product items with the names containing the search keyword. In this case, all the items whose names contain the letter “b” are searched.

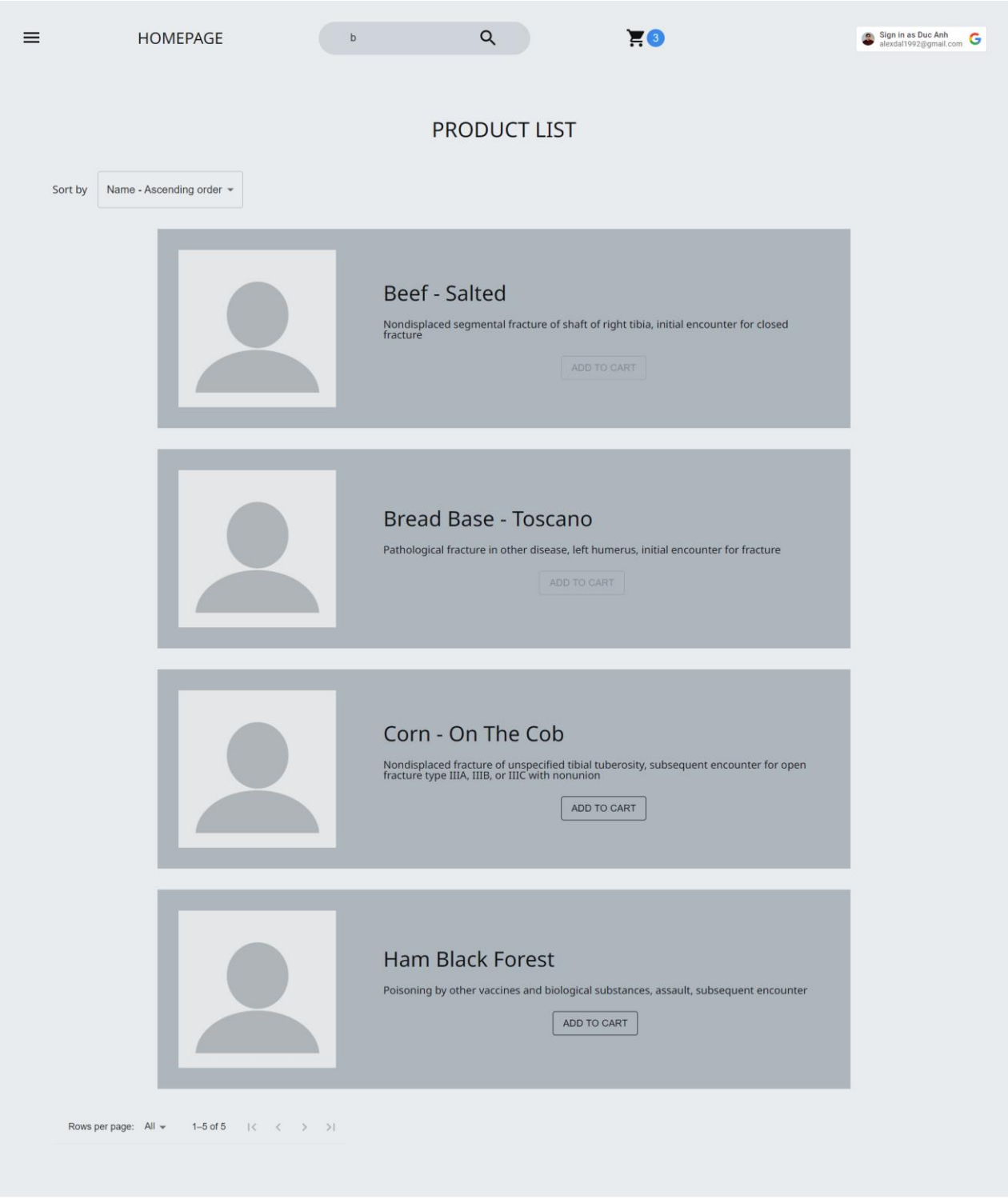

Figure 45. Search with keyword

Users can also sort the list of product items in different order:

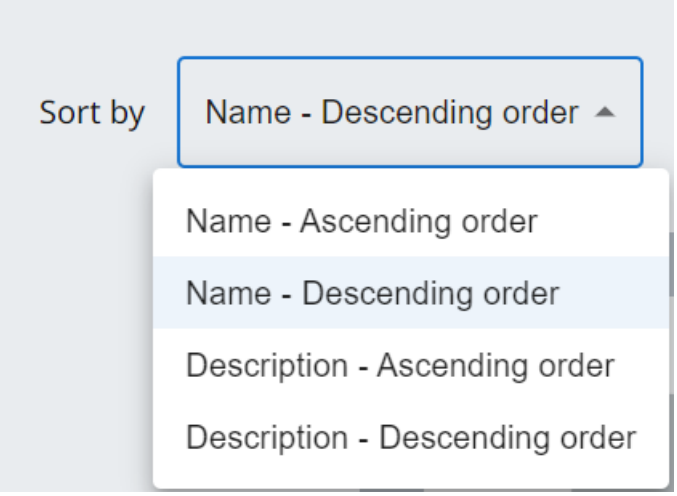

Figure 46. Sorting with different orders

Pagination on a website refers to the practice of separating lengthy content into multiple pages to enhance the user experience and readability. It involves dividing lengthy articles, blog posts, product lists, or search results into smaller, more digestible sections and providing links or icons for navigation. Pagination can increase the loading speed of a website and make it simpler for users to locate and consume the intended content. It is utilized frequently on e-commerce, news, blog, and other content-heavy websites.

The pagination component will divide the total number of product items across multiple pages. It permits users to view a limited number of items per page and navigate to other pages to view additional items.

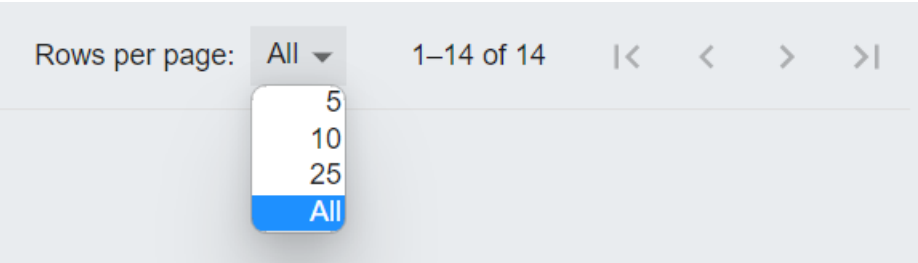

Figure 47. Pagination in the web application

Now we will observe what an admin user can do with the web application.

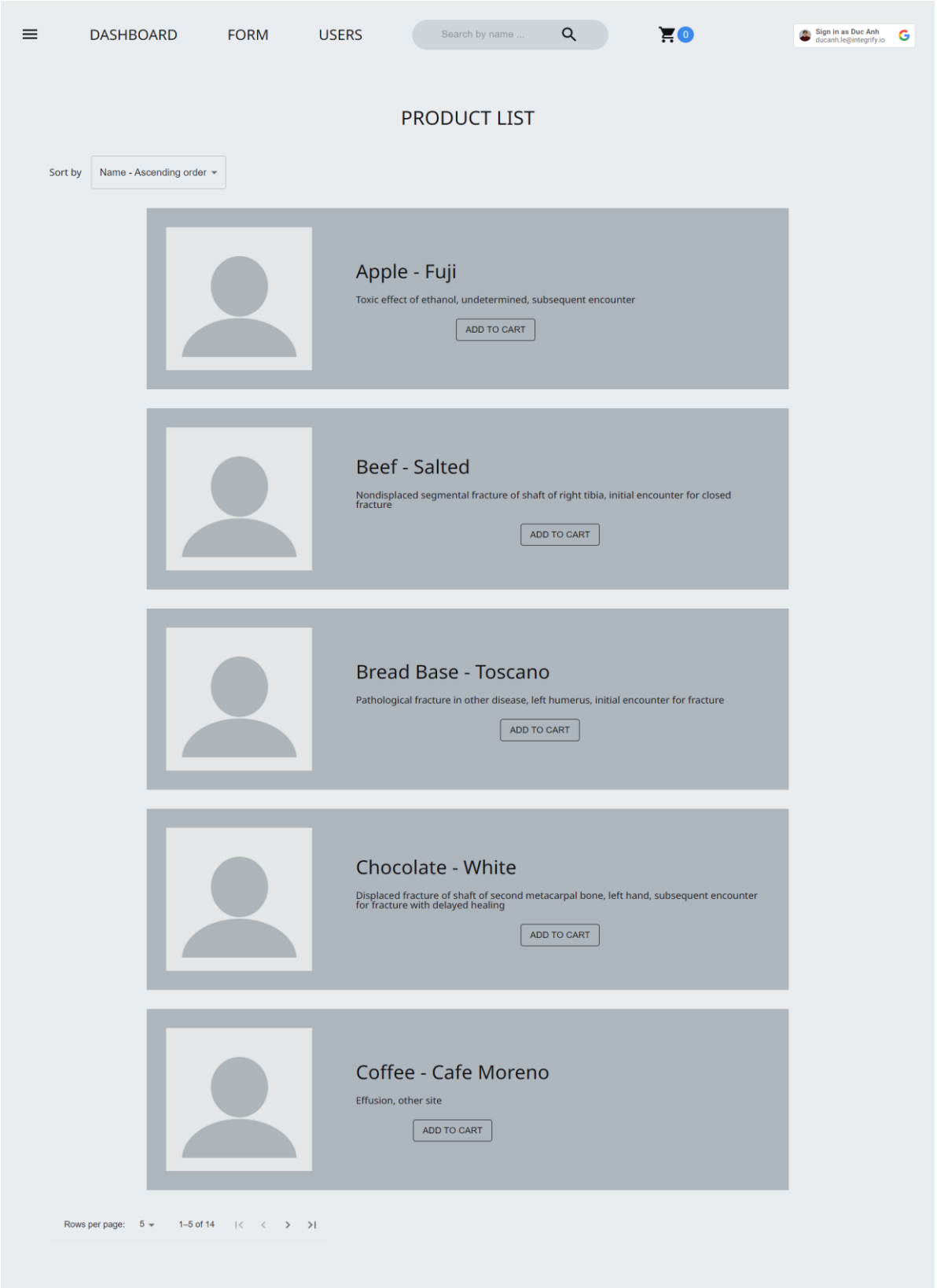

Figure 48. Homepage as an admin user

This time the home page looks different with additional pages, specially reserved for admin accounts only.

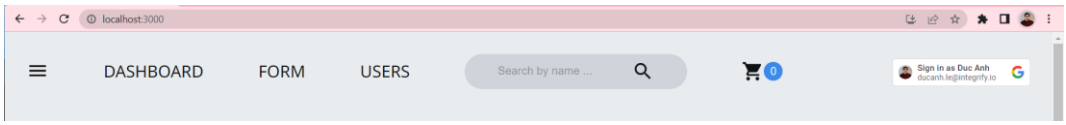

Figure 49. AppBar navigation for admin user

The Form page will allow admin users to create new products.

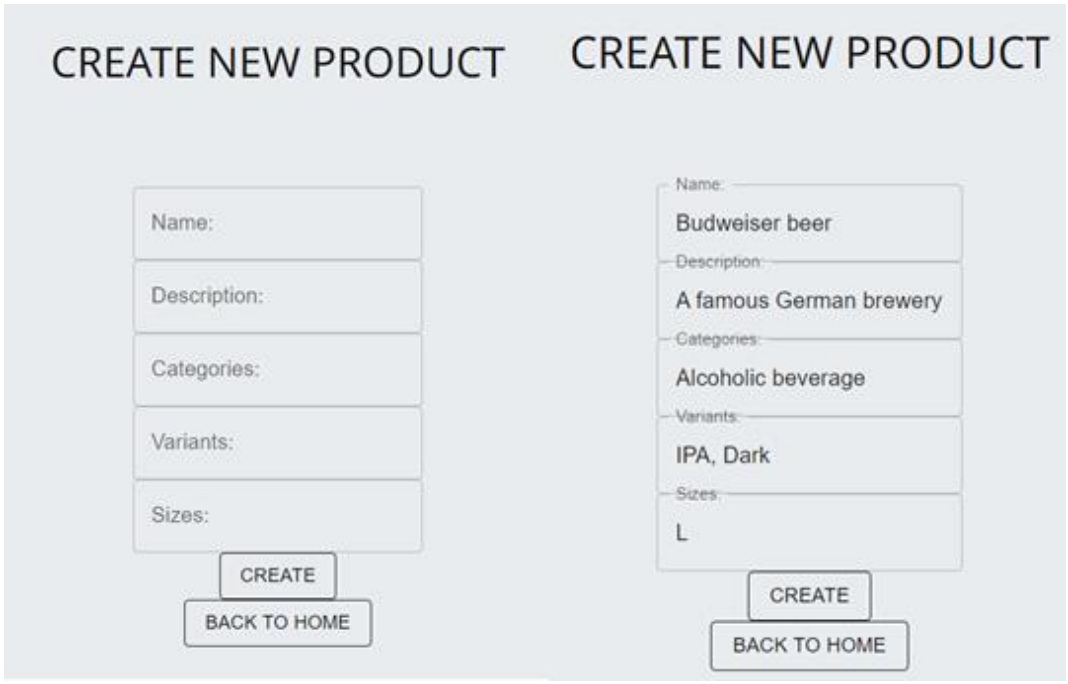

Figure 50. Product creation page

Let us try creating a new product and see how it will appear on the home page. After the creation of the product, we can see that the new product has been successfully added to the home page:

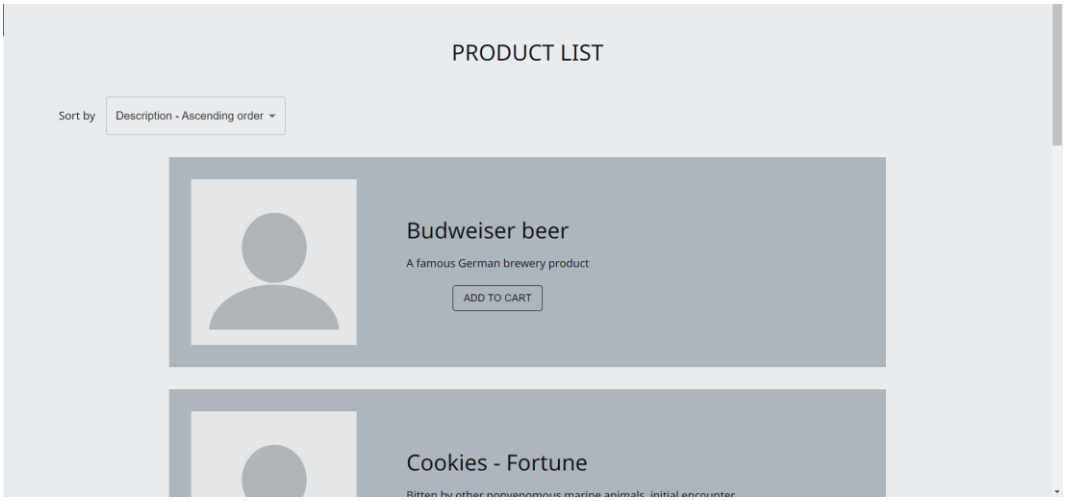

Figure 51. New product item on the home page

When we click on the new product item, it will direct us to the information page:

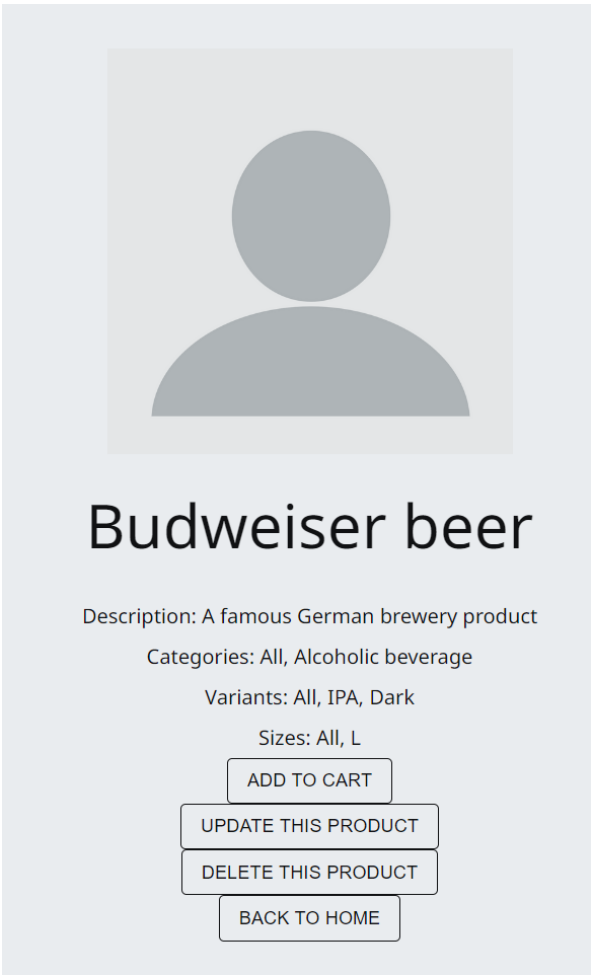

Figure 52. Information of the new product item

We can also update the product item with new data as an admin user:

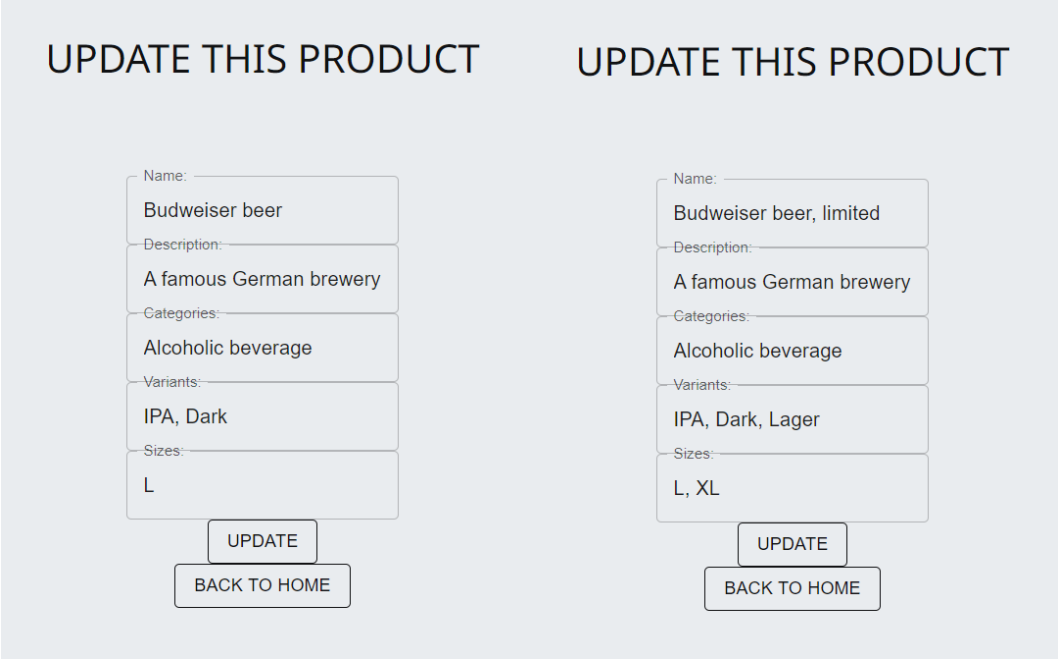

Figure 53. Modify product item information

Now the new data on our product has been successfully updated.

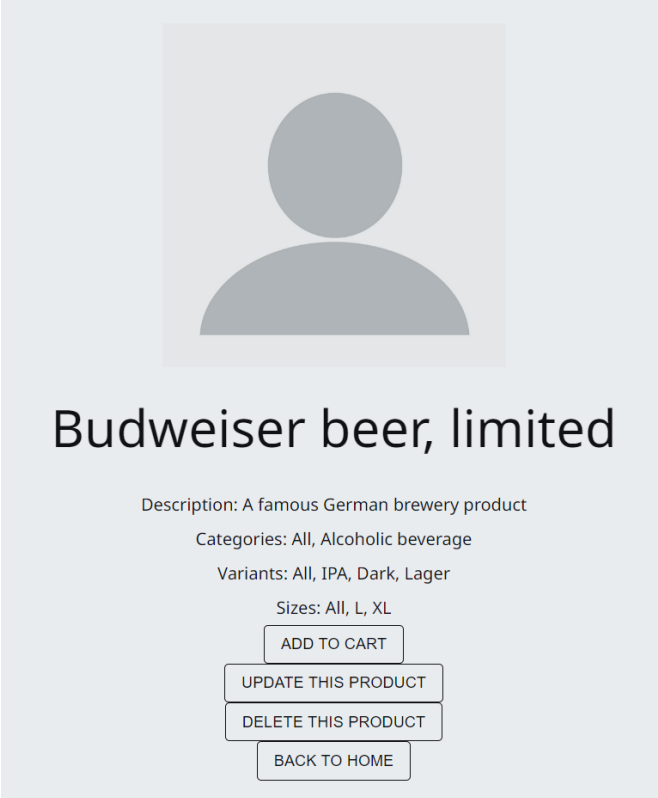

Figure 54. Product item with updated data

Another feature for admin users is that they can manage other accounts.

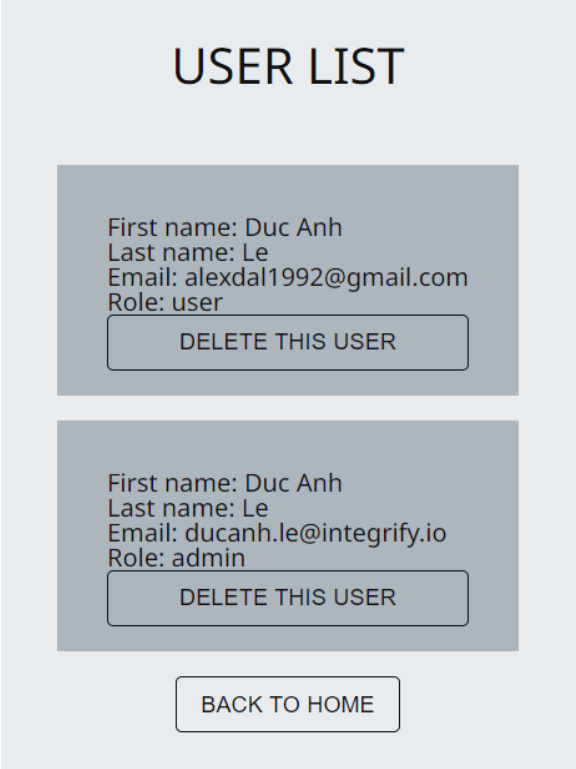

Figure 55. User list, only available to admin users

Once a user is deleted, that user information will be removed from the user list. The deleted user will then need to log in to his or her Google account again in order to gain access to our web application.

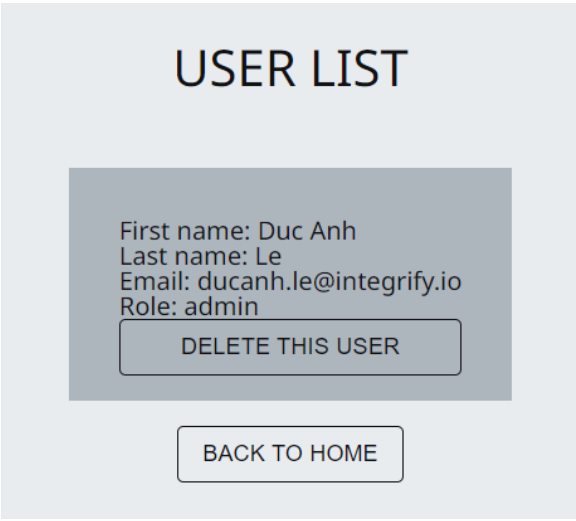

Figure 56. Updated user list after deletion of a user

6 AUTHENTICATION WITH GOOGLE PASSPORT

6.1 Introduction

In the current digital era, security is a top priority for both enterprises and individuals. The proliferation of online transactions, cloud computing, and social networking has increased the demand for secure and trustworthy authentication mechanisms. Google Passport offers a remedy to this problem by allowing users to authenticate and authorize requests with their Google account credentials.

Google Passport is an authentication and authorization service that allows developers to authenticate users and authorize API access using a singular set of credentials. It offers a unified authentication mechanism that enables a user to log in to multiple websites with the same Google account credentials. Google Passport enables developers to integrate authentication and authorization stages into their applications and websites.

It supports multiple authentication strategies, including OAuth 2.0, OpenID Connect, and third-party authentication providers like Facebook and Twitter, making it a flexible and versatile authentication platform. Google Passport complies with all data protection regulations and stores user data securely, ensuring the safety and preservation of user data. Many prominent technology companies and apps rely on Google Passport for user authentication and authorization. The system is widely utilized by developers around the globe. By utilizing Google Passport, businesses and developers can concentrate on providing innovative solutions without fretting about authentication and security.

6.2 Benefits of Authentication with Google Passport

Authentication with Google Passport is a popular and secure method for logging into multiple web services with a single set of logon credentials. Some of the benefits of authenticating with Google Passport are discussed next.

It is optimal for security to have a unique username and password for each online service used, but it can be difficult to manage. Users can forget their passwords or fall into the pitfall of using the same password for multiple accounts. Using Google Passport to link multiple online accounts with a single password provides an additional layer of security that can help mitigate these issues. This method can also expedite the login procedure and reduce the likelihood that your account will be compromised.

One of the greatest advantages of Google Passport is that it simplifies account access. Multiple websites can be logged onto without entering the credentials each time because they are stored in a central location. This makes logging in speedier and more convenient, ultimately saving you time.

With Google Passport, the user only needs to remember a single set of login credentials, making it much simpler to access multiple websites. There is no longer needed to remember multitudes of passwords for various accounts; instead, the user only needs to remember one password that grants access to Google Passport. Two-factor authentication adds an additional layer of security to the account by requiring a security code in addition to the user's login information.

Google Passport enhances the user experience by streamlining the registration procedure. Customers do not need to establish new accounts or remember complicated usernames and passwords to access multiple websites; they can simply use their Google Passport login credentials. This streamlined approach can provide consumers with a more efficient and satisfying experience, which can aid in retaining their loyalty.

Using Google Passport for authentication can significantly enhance online security, simplify account access, make passwords simpler to remember, and enhance the customer experience. In a world where online security threats are constantly growing, it is crucial to select an authentication method that ensures the safety of your data and credentials. Google Passport offers a secure and convenient method for doing so.

Authentication is a crucial security measure that helps prevent unauthorized access and deception. Google Passport is a popular authentication mechanism that enables users to sign in to multiple websites and applications without needing to recollect multiple login credentials. The steps required to create a Google Passport account, install the Passport application, link existing accounts to the application, and use the application for authentication are simple and straightforward.

7 CONCLUSIONS

Full-stack web application development necessitates a strategic approach involving meticulous planning, implementation, and testing. The development of this application utilizing MongoDB, TypeScript, NodeJS, ExpressJS, React, Redux, SASS, and Google Passport was a challenging but rewarding endeavor.

This application's development began with the identification of its functional requirements, which included creating user accounts, logging in, creating, and publishing content. In the initial phase of development, the database schema was created using MongoDB, and NodeJS and ExpressJS were configured to establish the server-side code. Next, we wrote the front-end components with TypeScript and integrated them with React and Redux. In consideration of the user interface and user experience, we also used SASS to create a custom appearance and feel for the application.

In the concluding phase of development, Google Passport was integrated for user authentication and authorization. This aspect of the application was essential for establishing secure communications between the server and client, thereby facilitating a seamless user experience.

During the application's development, we confronted a number of obstacles, particularly with the integration of the various technologies, user authentication procedure, and debugging and evaluating the application. We overcame these obstacles, however, by testing the application, identifying the problems, and implementing the required changes. Using a more robust framework, such as Angular or Vue.js, to streamline the development process would be a significant suggestion for enhancement.

The overall design of the application appears robust, and it provides a secure authentication mechanism that safeguards users against unauthorized access. In addition, TypeScript improves the maintainability of the application by providing a more structured codebase that makes it simpler to identify errors and vulnerabil-

ities. The techniques utilized in this development process have enormous potential for creating secure, scalable, and maintainable cross-platform web applications. These techniques have a promising future, and we anticipate additional innovations that will further facilitate the development process.

Overall, we are satisfied with the outcome of our efforts. However, there is always room for improvement, and we have identified a few areas where the application can be enhanced further. These areas include enhancing the user authentication process to improve security and the user experience, refining the code for improved performance and scalability, and adding more features to make the application more versatile and appealing to a larger audience.

In conclusion, the development of a full-stack web application is a complicated procedure requiring meticulous planning, implementation, and testing. The development process can be made more manageable by having a well-defined project plan, a clear comprehension of the functional requirements, and a talented team of developers. With careful attention to detail and a focus on the user experience, it is possible to effectively develop a full-stack web application that meets user needs and expectations.

REFERENCES

Richards, G., Francesco, Z.N. & Jan, V. 2015. Concrete Types for TypeScript. Accessed 14.04.2023. https://drops.dagstuhl.de/opus/volltexte/2015/5218/.

Sharma, S. 2020. TypeScript: A Superset of JavaScript. Accessed 27.05.2023. https://www.linkedin.com/pulse/typescript-superset-javascript-sunil-sharma/.

Gillis, A.S. & Botelho, B. 2023. MongoDB. Accessed 01.05.2023. https://www.techtarget.com/searchdatamanagement/definition/MongoDB.

Tutorials Point. 2023. MongoDB Tutorials. Accessed 01.03.2023. https://www.tutorialspoint.com/mongodb/index.htm.

Google Developers. 2023. Learn web development. Accessed 12.03.2023. https://web.dev/learn/.

W3Schools. n.d. What is React?. Accessed 21.03.2023. https://www.w3schools.com/whatis/whatis_react.asp.

Redux. 2023. Getting Started with Redux. Accessed 02.04.2023. https://redux.js.org/introduction/getting-started.

Luksza, R. n.d. How To Setup Redux with Redux Toolkit. Accessed 05.04.2023. https://www.softkraft.co/how-to-setup-redux-with-redux-toolkit/.

W3Schools. n.d. SASS Introduction. Accessed 07.04.2023. https://www.w3schools.com/sass/sass_intro.php.

Giraudel, K. 2023. SASS Guidelines. Accessed 11.04.2023. https://sass-guidelin.es/.

Simplilearn. 2023. Node.js Tutorial. Accessed 15.04.2023. https://www.simplilearn.com/tutorials/nodejs-tutorial.

Tutorials Point. 2023. ExpressJS Tutorial. Accessed 15.04.2023. https://www.tutorialspoint.com/expressjs/index.htm.

Geeks For Geeks. 2023. Express.js. Accessed 16.04.2023. https://www.geeksforgeeks.org/express-js/.

Ram, P. 2021. How to implement Google Authentication in Node JS using Passport JS. Accessed 27.04.2023. https://medium.com/@prashantramnyc/how-to-implement-google-authentication-in-node-js-using-passport-js-9873f244b55e.

Gathoni, M. 2022. NodeJS Google Authentication Using Passport and Express. Accessed 30.04.2023. https://www.makeuseof.com/nodejs-google-authentication/.