Expertise
and insight
for the future

Khoa Pham

Implementing Application with Modern Web Technologies and Microservices

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

04 May 2020

Author	Khoa Pham
Title	Implementing Application with Modern Web Technologies and Microservices
Number of Pages	55 pages
Date	04 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department ICT

This thesis aimed for student research work in applying modern web technologies to support everyday problems in real life. Particularly, initiated with the idea of reducing the food waste in supermarkets, the thesis created a platform for supermarkets and many kinds of organizations to connect with each other simply. However, this thesis concentrated on developing on the technical side with the aim to trial a large-scale application with numerous technologies combined all together for the higher efficiency in the web industry nowadays.

The technologies were chosen in this project including the TypeScript as the programming language, React and Redux for front-end development, Node.js, Express, and MongoDB for back-end development, and Docker and Kubernetes for the microservice architecture. The project also used Google Cloud Platform for the deployment. Aiming for researching, this project tried to reach new experiments to find out benefits and obstacles while building an application in a more macro-model.

In conclusion, this project has been accomplished with such a lot of effort to widen knowledge as well as experience in developing a demo large-scale web application. Despite the time restriction, this project has completed adequate features as a real-life application. Finally, from the technical field, this thesis may contribute to the research and development in expanding those kinds of techniques in various systems.

Keywords	TypeScript, React, Redux, Node.js, Microservices, Docker, Kubernetes

Contents

List of Abbreviations

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command-line Interface
DOM	Document Object Model
ES6	ECMAScript 6
GCP	Google Cloud Platform
GCR	Google Container Registry
GKE	Google Kubernetes Engine
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JSX	JavaScript XML
NPM	Node Package Manager
ODM	Object Data Modeling
RDBMS	Relational Database Management System
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security

1 Introduction

With 1.3 billion tons of food production wasted globally per year, food waste has become one of the most attentive problems nowadays [1]. Particularly, the yearly 400 million kilos of eatable food in Finland are thrown away [2]. Therefore, this thesis is initiated from that idea and solves the problem with a web application named Food Supporter. Generally, the Food Supporter application is designed with the purpose to solve the aforementioned problem, especially in supermarkets by helping them in connecting to as many organizations as possible to consume the edible food instead of throwing it away.

Food Supporter will create a platform where supermarkets and different kinds of organizations including non-profit organizations, restaurants, and cafeterias can contact each other conveniently. This platform is a web application built mostly with the most modern web technologies for the most efficient and powerful performance developed in full-stack.

Ideated from those, this thesis as Food Supporter will mainly focus on the technical side with an all in one application starting from the frontend, going through the backend, and completing with the microservices. The technologies which will be used for this application are TypeScript programming language, front-end technologies using React and Redux, backend technologies using Node.js, Express, and MongoDB, and microservice architectures with Docker and Kubernetes.

In the imagination of all the users (all supermarkets, non-profit organizations, restaurants, and cafeterias in the whole of Finland) joining into the system, this project is considered a large-scale application. That is why it is necessary to apply the above technologies for this thesis as technical research in developing new technologies into this web application.

Besides, the Food Supporter application is aimed to respond as many queries as possible with an extremely large database, yet still ensures the high efficiency in workload. By applying a complex combination of distinct technologies to solve a real-life problem, this project is realistic and reachable enough in the research field or even developing more in the future.

2 Theoretical Background

2.1 TypeScript

TypeScript, which is an open-source project developed by Microsoft in 2012, is considered a high-end programming language from JavaScript. Or, it, in another explanation, is said to be another kind of advanced JavaScript [3]. TypeScript is designed by Anders Hejlsberg, a founder of C# programming [4].

As a superset of JavaScript, programmers can use and develop simply TypeScript through any JavaScript file by replacing the extension .js into .ts. This language is added static typing options and object-oriented modules, including ECMAScript with the latest updated version [3]. Also, TypeScript supports tools for large-scale JavaScript applications with cross-platform required a large number of codes such as Web browsers, Backend, Desktop, Mobile Apps, and Cloud [5] [6].

In 2014, Google cooperated with Microsoft for using TypeScript to improve the Angular 2 framework. Since then, this language has become more popular for developers these days. And regarding Stack Overflow's 2019 Developer Survey, TypeScript was ranked in 3rd place of the most favorite programming languages as illustrated in figure 1. [3]

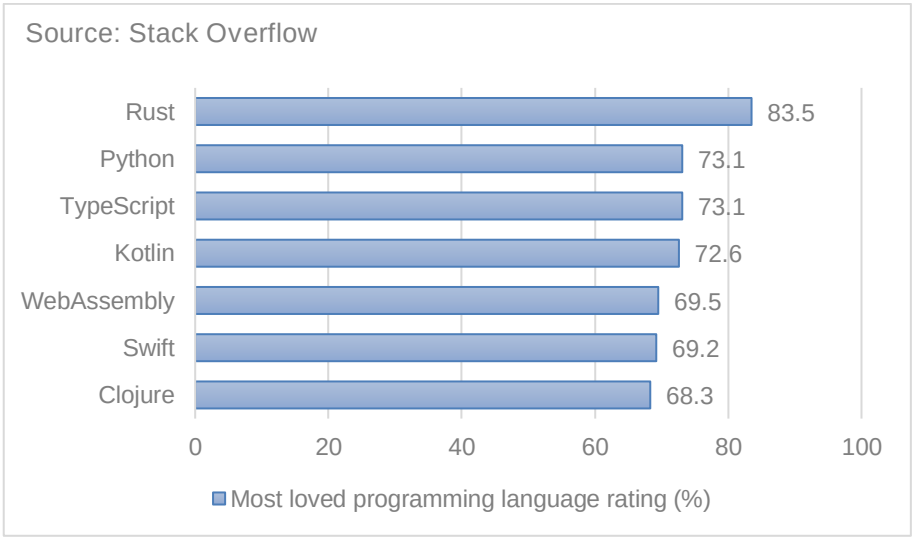

Figure 1. The most favorite Programming languages via Stack Overflow's Developer Survey in 2019 [11].

The nature of TypeScript is JavaScript, yet it is developed as a superset of JavaScript. TypeScript is compiled into JavaScript code so that it can run anywhere that supports JavaScript code. This also means it is possible to write TypeScript code on a JavaScript-based code [5]. (See figure 2 below) Nonetheless, it is unable to run TypeScript in IDEs as JavaScript, but TypeScript could be compiled to JavaScript by a compiler through the **npm** package with Node.js, which programmers need to setup manually themselves [3].

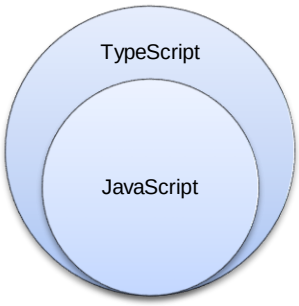

Figure 2. TypeScript in relation to JavaScript.

Optimized from JavaScript, TypeScript is created static typing together with the available dynamic typing in JavaScript as apart. With static typing, a type needs to be set into a variable before using it. By using this, it seems to be more convenient for programmers to create variable declarations and make some attempts to get different type results despite the compile-time error, meanwhile, it has been the biggest challenge in JavaScript. [3]

Built from JavaScript, TypeScript has all kinds of basic JavaScript data-typing, such as Number, String, Boolean, Symbol, Array, Object, etc. However, there are other types in TypeScript as Enum, Tuple, Any, Void. Moreover, in helping developers with an effective programming language, TypeScript has been created with an option in which developers can also define new types on their own by using **type** or **interface** as in listing 1. [7]

```
type Season = 'spring' | 'summer' | 'autumn' | 'winter';
type Age = number;
interface User {
      id: string;
      name: string;
      age: number;
}
```

Listing 1. Code example of user-defined type.

About how functions operate, a result would be returned in TypeScript depending on its type (for example, Number, String, etc.) while any functions declared Void or Any would return a value instead. In the listing 2 are examples of these circumstances. [7]

```
const hello = (name: string): void => {
     console.log(name);
};
const multiple = (a: number, b: number): number => {
     return a * b;
};
```

Listing 2. Code example of functions in TypeScript.

TypeScript also supports object-oriented programming with Inheritance, Abstract, Constructor, Implement, Interface, etc. [7]

Table 1. JavaScript and TypeScript in comparison [8].

	JavaScript	**TypeScript**
Data-typing	Dynamic	Static / Dynamic
Automatically converted types support	Yes	No (Mostly)
Checked types at	Runtime	Compile-time
Error message at	Runtime	Compile-time

Developed from JavaScript basis and got huge advantages as summarized in table 1, TypeScript is targeted on developers for easy lives by improving JavaScript to make codes cleaner, more innocent, and more helpful with type annotation or structural typing [5]. By using TypeScript, programmers can access any existing JavaScript libraries. However, they need to define the type of files to let it work. This is somehow imperfect for TypeScript's programmers [3] [5].

Besides that, 2 different kinds of typing, which are static and dynamic ones which are used in one is considered as a plus in TypeScript [4] [5]. A new update of ECMAScript is another benefit of TypeScript as well. Developers can work in any versions without any worries about the newer or older ones. Therefore, it could lead TypeScript to be more trending in the future [5].

The compiler is one of the reasons that made TypeScript become one of the most-loved programming languages for developers last year and helps JavaScript to become a trending language of programmers these days [4]. The open-source Apache license together with development for cross-platform are benefits that can't be denied as well [6].

However, there could be a limitation in TypeScript's feature when developers aim to older versions. It means the TypeScript compiler cannot translate completely all features used from the latest versions down to older ones. There might be some features missing anyway. [7]

Besides such a lot of benefits that TypeScript has delivered to developers, one more thing made TypeScript become a popular language is TypeScript works well for full-stack programming. This means it matches both front-end frameworks and back-end servers [9]. Firstly, for front-end working interactive environments, React is one of the most well-known frameworks and the favored choice for frontend coders because its components are defined and used frequently in TypeScript [8].

Meanwhile in another perspective, React is supposed to be not only a framework but also a library that could render View in MVC pattern [3]. TypeScript supports the React/JSX syntax only in the release of 1.6. For the new JSX syntax, the extension **.tsx** would be taken over from **.ts**. And templates used in React are combined into the normal JavaScript with a similar HTML-syntax [4].

To create a new React application with a transpiler as well as a bundler, developers need to configure files into the app by the CLI. This progress is called Create React App. The React package installation includes a library of JavaScript to create UI called **react**, a React package to work with DOM called **react-dom**, and the performance of Create React App for scripting and configuring called **react-scripts**. [3]

Also, there are two sorts of components: function and class components declared in React with properties and TSX rendering. Though, operating with React in TypeScript, programmers should follow restrictions enforced by TypeScript. For instance, tag names should be typed and nested correctly, or do not skip any irrelevant properties to a function component. [8]

As mentioned earlier, TypeScript grows to trend in web development since it is generated in not only front-end frameworks, such as Angular, React, Vue but also back-end servers like Node.js [9]. In a simple way to describe, TypeScript with node package can play a role in transpiling **.ts** to **.js** [10].

Consequently, a protection layer is created to check and interfere with type from TypeScript for the transpiling process. Further, TypeScript has numerous typing repository contributed from the community known as DefinitelyTyped. By using the prefix **@types/**, any kinds of typing are available via npm then. [10]

2.2 Front-end

2.2.1 React

Since React appeared, it has not only been a front-end framework built-in JavaScript but also become a powerful JavaScript library, which is compatible with mobile applications, server, client, or possibly VR applications as well [13]. The founder of React is Jordan Walke, a Facebook software engineer. It started being used on Facebook in 2011 and on Instagram in 2012, React was released officially as an open-sourced library for developers in 2013 [12].

In recent years, there are such a lot of projects in technical sites that used React to develop because of its huge benefits for developing applications. Those companies could be listed as Microsoft, PayPal, Tesla, Dropbox, Twitter, Disney, Netflix, Uber, or even Salesforce excluding Facebook – the foundation one [13]. React was ranked 2nd position in the most popular Web Frameworks (slightly overtook Angular compared to the previous year) as in figure 3 and became 1st one for the most loved Web framework in 2019 by developers via Stack Overflow as in figure 4 [11].

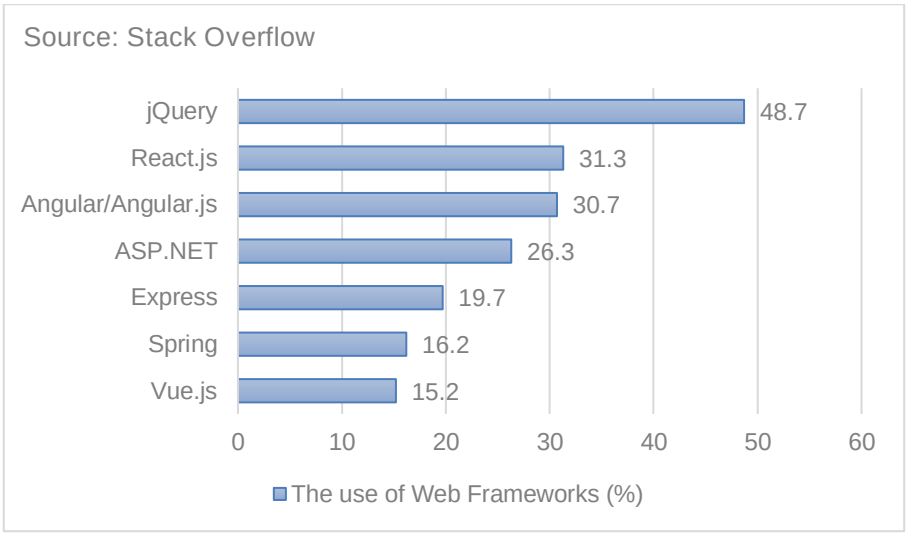

Figure 3. The most popular Web Frameworks via Stack Overflow's Developer Survey in 2019 [11].

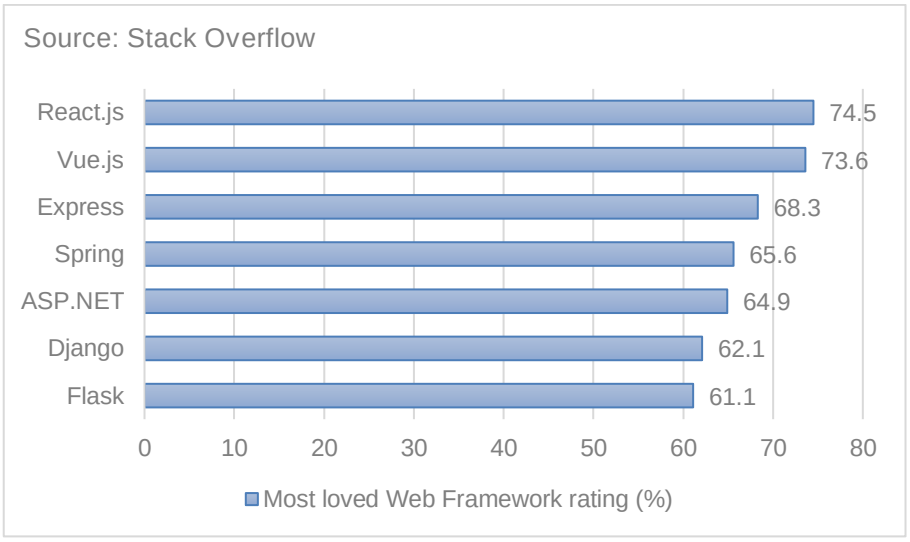

Figure 4. The most loved Web Frameworks via Stack Overflow's Developer Survey in 2019 [11].

As seen in the chart above, in comparison with other frameworks (Angular or Vue), React knocked down JavaScript fans quickly by its new techniques [13]. For example, React had very good movements in 2017 with React Fiber or React Hooks in 2019 [12]. There are some most interesting marks from React that could attract more developers to use including the efficient Virtual DOM, component-oriented programming, and the document-model abstraction.

Firstly, one of the most considerable reasons that React got success is Virtual DOM. Most web development techniques normally use DOM, which needs more resources by fetching and uploading data then reloading pages. This progress may be somehow more maintainable, scalable, and performed in the system. [14]

However, React solved that problem with an extremely complicated algorithm to create a Virtual DOM. Virtual DOM works more effectively and quickly than the DOM by forcing the in-memory portrayal of DOM. So, with any interactions from users, or fetching data, React will make a comparison between the current and the newer ones, then catch the new page instead of doing the whole process as the DOM. Consequently, applications used React may reach 60 frames per second despite the mobile devices. [14]

Moreover, React is structured by many blocks that play different roles in one layout. These blocks that could be anything built a user interface in React (eg. buttons, text boxes, forms, etc.) are called components. Written on JavaScript and oriented towards components, React with this ideal orientation is supposed to be the most beneficial for programmers fully when it combines all in one to target specific parts on a page. [14]

It is even more ideal when the components can be reused to React [15]. This means it has changed a little bit of the programming way by oriented towards components instead of separate tasks containing HTML, CSS, and JavaScript. It can reduce the limitation of different technologies. And it is also easier for developers to modify the content of each part [14].

The last advantage from React also comes from prior ideas mentioned earlier. That is the reduction for the document model with a super light user interface representation. Therefore, with this meaning, React approaches compliant standards and decreases the inconsistency in devices or browsers as well as perceives performance by the rendering of components. [14]

Furthermore, React is also created without any limitations on any presentation patterns. This means React works well with all kinds of models including MVC (Model – View – Controller), MVP (Model – View – Presenter), or even MVVM (Model – View – View-Model). Mostly, React does its task as rendering View in most models. [15]

Before getting knowledge about how React works, the first thing anyone needs to keep in mind is that React is an extremely powerful library that helps developers to customize the user interface flexibly with JavaScript and XML optionally. If JavaScript is designed to best fit with browsers, Node.js will play a role as a mediator between JavaScript applications and the system (as the local personal computer or a server) via the CLI. Additionally, Node.js often works together with **npm** to get the highest efficiency. [14]

At first glance, the definition of React in the book Pro React is thought to be **an engine** due to its power and interactions in the UI. React uses the Virtual DOM to manipulate the representation with fewer resources than the normal DOM by comparing differences between the recently changed page and the previous one, then carrying tasks out [14]. This improvement has led React with more users in the web development industry.

Then, React directs the user interface with specific and targeted content blocks named **components** to lower tasks in the whole UI. These components are reusable, extendible, and maintainable. The components could be created as a single element or nested into smaller parts as subcomponents so that programmers can easily develop their applications as their architectures. [14]

Furthermore, React works with the optional JSX, which is convenient. By adding the HTML code into JavaScript, React becomes so simple for Java-based developers. The minor problem in React is that it requires a translator to transpile the JSX into JavaScript. [14]

Particularly, a normal project used React includes a container of all necessary JavaScript modules called source folder, the main content page in index.html (commonly known as a landing page) where its duty is how the user interface should be shown, a package.json with all information of the project, and a module packager which is usually used to transform the JSX and others related. [14]

To work with React, programmers can create React elements by createElement, then render them into browser with ReactDOM. Then, as mentioned earlier, the most important definition in React is components with 2 types: a function and a class component. [12]

Function component should be a JavaScript or ES6 function, return a React element and might get props if needed while Class component is a bit more complex than Function component since it needs an initialization method, life-cycle, a render function leading to a React element, and maintain state data as well. [12]

Next, about how to let data work in components, two other definitions used are props and state. Props are used to push data to components from above and unchangeable whereas the state belongs to a component, works internally in a component, and could be changed with any events from the user. Additionally, there are two kinds of component attributes including stateful and pure components. [14]

Finally, the last part from React defined here is Hooks. Added in React 16.8, Hooks have solved existed problems for large scale applications like “wrapper hell”, the confusing classes, or even big component. React Hooks allow developers to use state and lifecycles without an ES6 Class as usual. [16]

Some advantages from Hooks that can be listed are minimized components, concise code, and even the use of states between components [16]. Nevertheless, there are some rules to remember when using Hooks. For example, Hooks must be called on top of a function, cannot be in the loops, conditions, or nested functions, or they are used only in functional components [17].

2.2.2 Redux

Since React had appeared in the web development technology, many people thought it would be the best fit for front-end tasks to render View in the MVC model. Nonetheless, the larger the project is, the more difficult the state management becomes. This is the reason why Redux was created and joined the web industry [18].

While React aims for an efficient user interface with extreme sophistication and high interactions, Redux, in contrast, is used widely in front-end development due to its reputation in state management [19]. In normal large-scale applications, there are so many issues relating to the state of an application, such as responses from servers, fetched data, or optimistic updates [20].

As a consequence, those lead to bigger problems with a conflicted and inconsistent state. Eventually, the application state is too easy to lose control leading to a slow-down the application generally. The main reason why those problems are too hard to solve because we are mixing two unusual concepts: Mutation and Asynchronicity. Also, Redux helps in trying to prevent state Mutation from unpredictable actions, yet still, keep Asynchronicity performance [20].

As developed later than Flux, which is an architecture pattern in supporting React for solving the problem above, Redux was also created for the pretty similar purpose, and even considered as an alternative solution. So Redux commonly works with React to get the best efficiency [18].

Redux can get connections with React via bindings that are used to connect data between the Redux store and React View component. Thus, Redux state management is carried out simply with most frameworks, such as Angular, Backbone, Ember, etc. To sum up for an overall view, if React is responsible for the application view, Redux will establish the application particularly. [18]

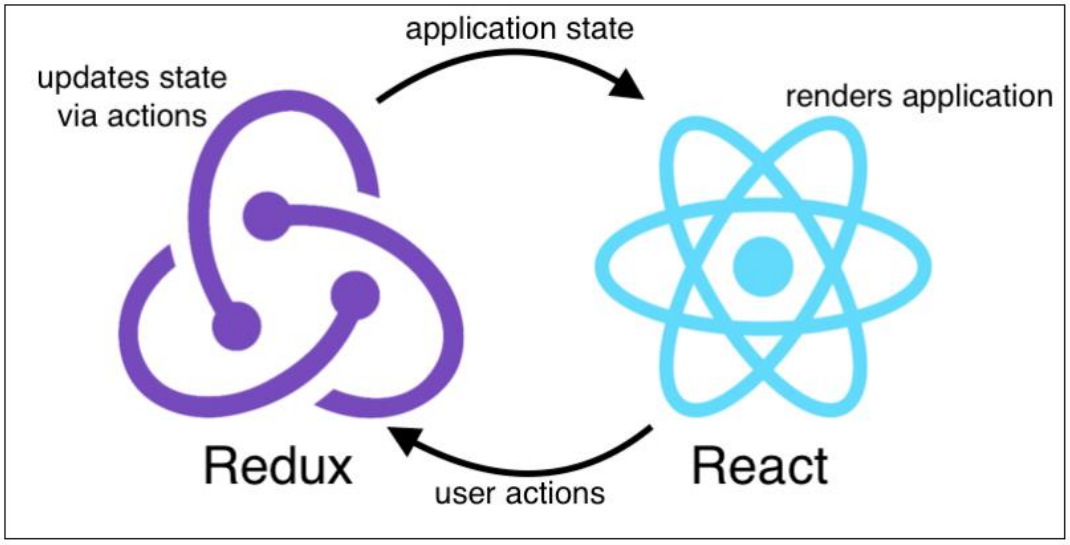

Figure 5. Interaction between Redux and React [20].

Redux has three fundamental principles including the single store of truth, the read-only state, and state changes made with pure functions only [20] via the flow: View → Action

→ Reducer → Store [18]. Figure 6 below is a diagram to show more specifically about how Redux works generally [18].

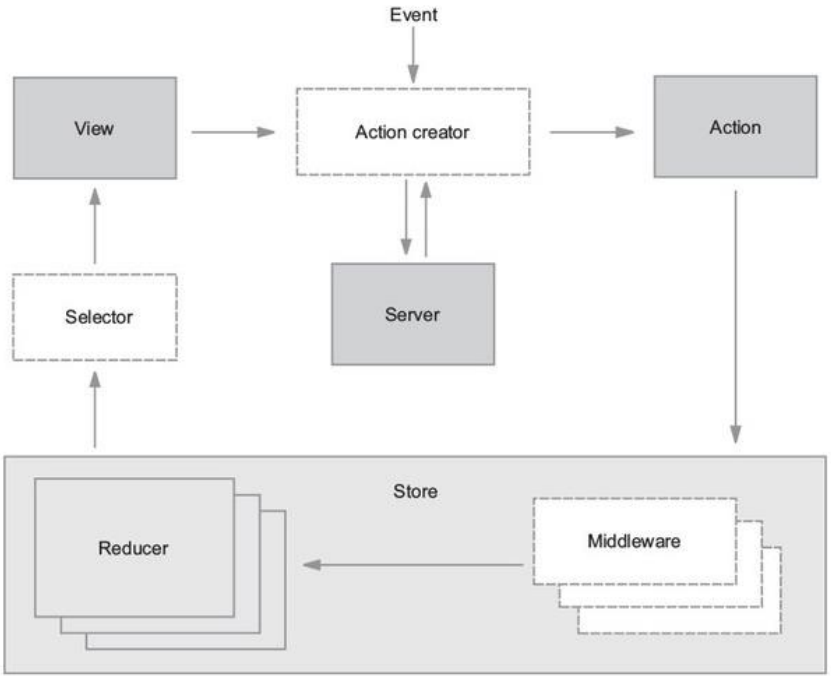

Figure 6. Redux workflow [18].

In general, Redux is an ideal solution for managing state, especially in large-scale applications. Yet, it is only ideal for large projects which meet numerous troubles with the state, not for the simpler ones. The advice from the creator is also considering your applications or projects before using Redux [18].

2.3 Back-end

2.3.1 Node.js

Node.js is a platform working on the V8 JavaScript Runtime environment – an extremely fast JavaScript transpilator on the Chrome browser. Also, Node.js can be used to develop many kinds of applications including web applications or even server-side or client-side applications. Nowadays there have been more companies switching their services to Node.js including PayPal, LinkedIn, eBay, or even Walmart. [21]

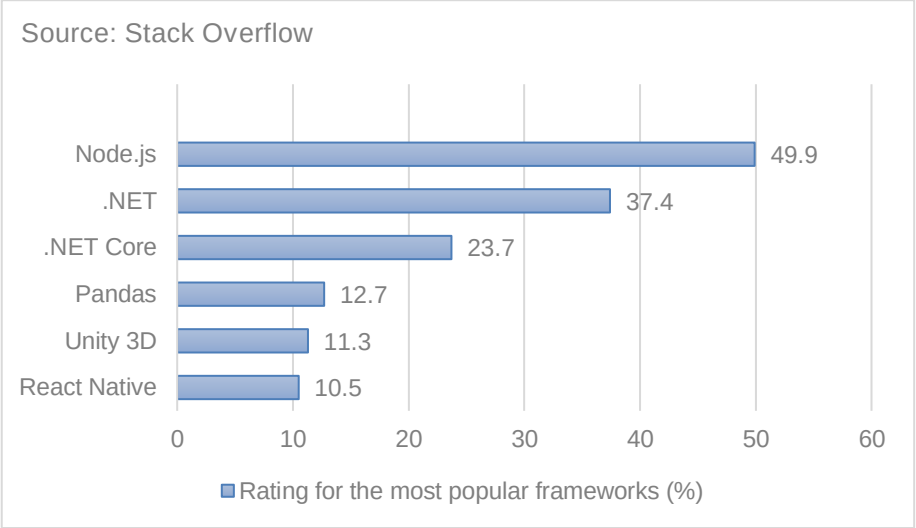

Figure 7. The most popular frameworks in different fields via Stack Overflow's Developer Survey [11].

Built with asynchronous functions and the single-thread event-driven architecture, Node.js is perfect for scalable network applications. Moreover, Node.js is designed to solve complicated and big issues when expanding applications. It operates the combined server-side based on JavaScript and the asynchronous non-blocking I/O mechanism. [21]

Node.js is considered an enticing solution in modern web architecture instead of the predecessors such as Java, Python, PHP, Ruby, etc [21]. Meanwhile, there are a few reasons for the thing that Node.js has become the most prevailing platform in the web development industry these days [22].

Firstly, it is simple and convenient because Node.js is written on JavaScript, also introduced with the ES6. So, there is no need to learn another technology or language [22]. Secondly, the aforementioned principles of Node.js have created itself efficiently to get high execution speed and scalability [21]. The third reason is the huge community support of Node.js might be useful for developers anyhow [22].

However, there might be some limitations that are unavoidable with Node.js. For example, Node.js is not supposed simply to be a replacement of Apache: The simpler the Node.js code is, the less convenient the work is. It means developers have to access

and program themselves at a low level of the HTTP or any other protocols. Meanwhile, there are many frameworks in the community to help in solving this trouble by supporting the higher level for programmers such as Express, Rails, Meteor, etc [21].

Or, another disadvantage of Node.js is that it is still being developed. This could lead to some features which might be changed in the future [10]. Otherwise, Node.js has grown rapidly as one of the most effective technologies these days. Before Node.js has appeared in the web technology era, backend developers had to decide which language on the server-side was suitable for their projects, such as Java, PHP, Python, or even Ruby, whereas, frontend developers still used JavaScript commonly in the client-side [21].

Yet, since the Node.js was designed in JavaScript-based programming language, it solved conflicted problems above between the client-side and server-side in programming. Since then, developers could start programming as full-stack work, from the first step for the user interface to the final step on the database works without learning another language. [21]

By using one language only for the whole system, the code is more concise in general, the same data formats and tools are applied for both server-side and client-side programming as well as the testing process and tools are similar in both sides [21]. Furthermore, Node.js is built on a JavaScript engine which is developed by Google called V8 engine for higher speed and more efficient performance [22].

However, as mentioned earlier, based on JavaScript language, Node.js works in a single thread environment with the non-blocking I/O mechanism to get better performance on the system [10]. Particularly, the traditional web development uses the blocking I/O to call data back. This has caused threads waiting in a queue, and the slowdown in server happening. For small applications, that could be minor issues, yet with the bigger ones, that is certainly major issues [22].

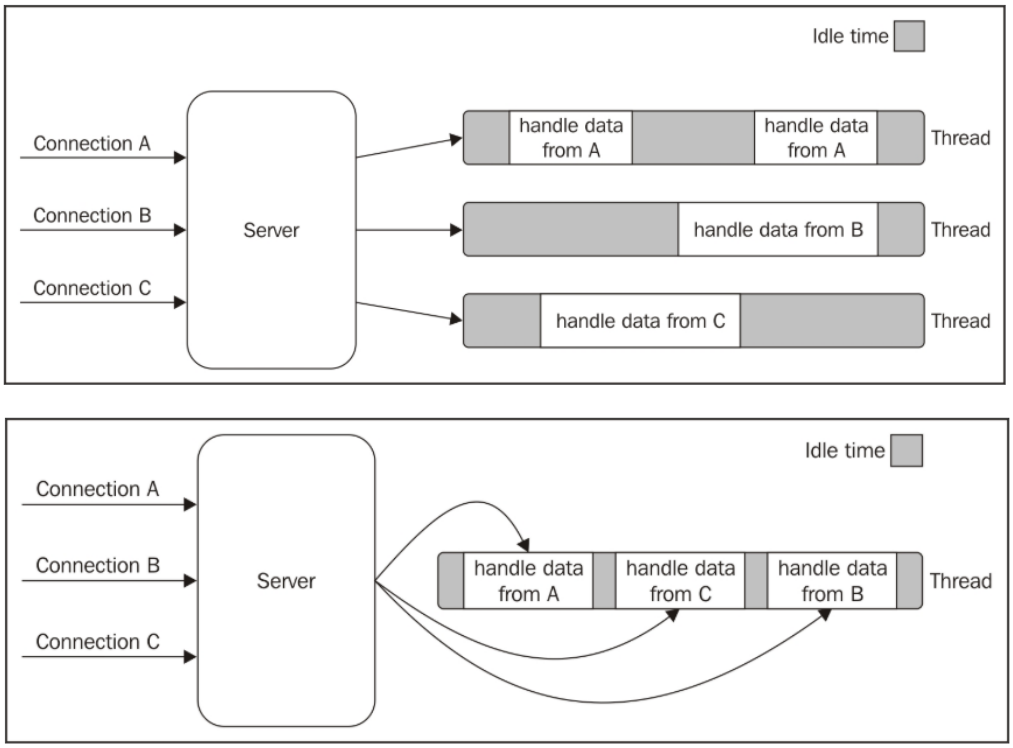

Figure 8. The comparison of the blocking I/O with non-blocking I/O [22].

On the contrary, Node.js is designed on the non-blocking I/O instead of the normal I/O to reduce the waiting line of processing threads, hence an event loop with asynchronous functions is created to carry out queueing tasks effectively. This single-thread event-driven model has made Node.js more powerful itself for backend works [21]. Figure 9 will show specifically how the Node.js manages the task line [22].

Besides that, Node.js is the ideal solution for microservice architecture, which is the best fit for agile projects in management. The microservices, in a simple definition, are a kind of software architecture with modules split into smaller parts called microservices and put in discrete servers. And Node.js is an extraordinary platform for microservice implementation. [21]

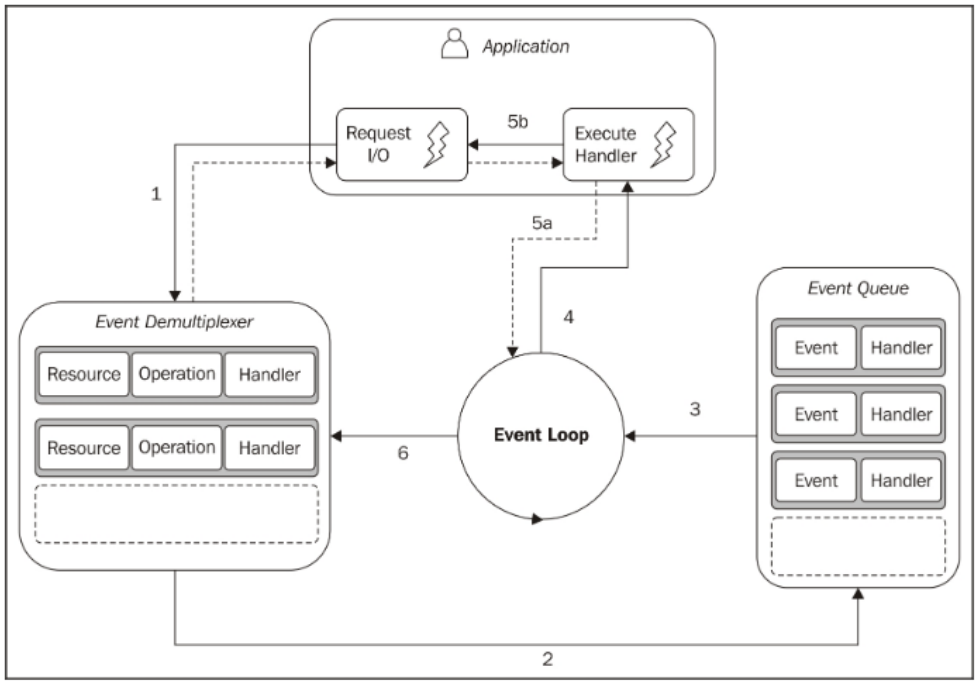

Figure 9. The execution of Event Loop with Node.js [22].

Together with the development of Node.js, there are some management system tools created to manage JavaScript libraries for Node.js. There are 2 most popular tools mentioned here to compare: NPM and Yarn.

NPM is a tool for creating and managing JavaScript libraries as well as dependencies. NPM works as an open-source repository for packages and modules, a CLI interacting with other online platforms, such as servers and browsers, and management tools while working with Node.js. Also, Yarn is another JavaScript package manager designed by Facebook, Google, and some other companies to solve problems in NPM. [21]

For NPM, package installation is slow and asynchronous, or issues related to security when NPM allows packages to run code through installation. Therefore, Yarn is said to be fixed those problems: It is faster, more secured, and more reliable than NPM. Nonetheless, built with the same features as NPM and using the same repository of NPM, but on top of the repository, Yarn is recommended to use due to more advantages. [21]

In the development of web technology, many web and API frameworks are popular to support Node.js developers in speeding up their works, for example, Express, Sails,

Meteor, Hapi, LoopBack, Koa, etc. These frameworks are designed to increase capabilities and get rid of the complexities of the default system; hence it is more convenient and simpler for developers' works [23].

2.3.2 Express

Express is a minimal and flexible framework built on the Node.js platform with sturdy features in web or mobile development. The express framework is created by TJ Holowaychuk. It started ideas from the Sinatra framework based on the Ruby language. Therefore, Express has become so popular with its huge benefits, such as package support or higher performance while programming [24].

With Express, two concepts made it reputable: the minimalization and flexibility. The minimalization attribute lets developers do easier work, yet more effectiveness whereas the flexibility simplifies the execution with receiving HTTP requests from the client-side then gets back HTTP responses. Due to the better performance, Express is considered best fit with the high-traffic web applications and a diverse ecosystem providing middleware system flexibly. [24]

About the operation, Express supports HTTP protocols and middleware to create a robust and usable API. Main features of Express include the implementation of middleware to return HTTP requests, defining router with different actions via the HTTP and URL, and allowing to return HTML through parameters. In general, with Express, developers can build their applications as a full-stack work since Express is one of the best frameworks for Node.js in the modern web development era. [24]

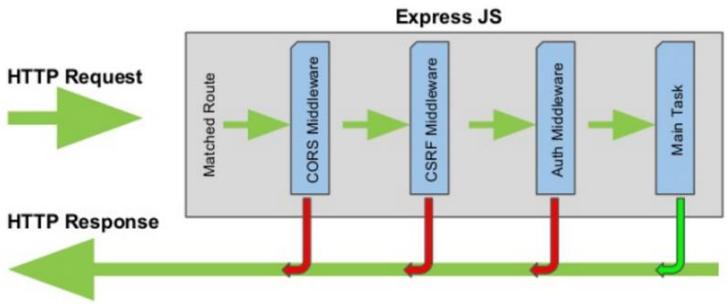

Figure 10. The Express workflow with Route [25].

2.3.3 MongoDB

MongoDB is an open-source document-oriented database, which stores data in another kind of database model comparing to the traditional database with tables. Unlike other predecessors in managing databases using the RDBMS, MongoDB is distinctive because of the NoSQL (abbreviated to None-Relational SQL or Not-Only SQL). [26]

The NoSQL is created to solve existing problems of the traditional SQL, such as speed, features, or even scalability. With NoSQL, it dismisses the database unity and transaction for higher performance as well as scalability. MongoDB allows developers to store database in JSON with a flexible schema. Compared to the SQL, it replaces table with the collection, row with the document, column with the field, joins with embedded documents. [26]

Developed with a new technique, there are pretty numerous benefits from using MongoDB. Firstly, collections in MongoDB are more diverse in sizes and more flexible in storing the database. Next, the horizontal scalability in MongoDB is so simple by adding nodes into the cluster. [26]

Thirdly, the cache of a data query is recorded on memory (RAM), so the flow of the next query is faster without reading from the drive. Last but not least, its performance is better and more powerful than other RDBMS due to the accessible speed. [26]

However, MongoDB still has some existing disadvantages. For instance, it might cause some flaws without the binding data, or it requires more memories for saving the database via the key-value. Overall, MongoDB is suitable for real-time applications that need to respond quickly, or a bigger database requires a large number of requests. [26]

2.4 Microservices

2.4.1 Docker

Docker is an open-source platform providing tools and services for users to contain, deploy, and run applications in distinct environments quickly. With Docker, users can create

separate and independent environments to launch and develop applications. This progress is called a container. To deploy applications onto servers, developers only need to run a Docker container, then it will launch the application immediately. [27]

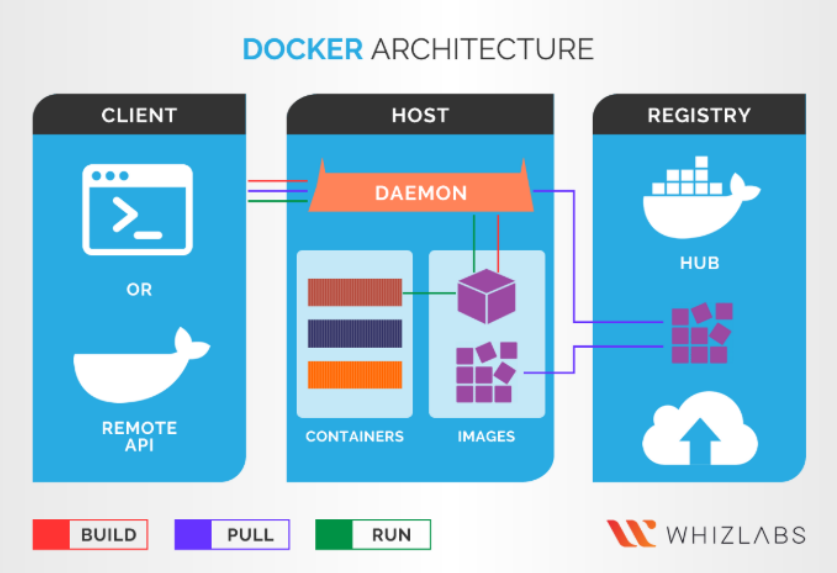

Figure 11. Docker architecture [28].

Docker includes several important elements: image, container, Docker run, Docker Engine, Docker Hub, Docker Daemon, etc. as in figure 11. For example, image is a read-only file which contains necessary specifications of the application where it will run while the container is an image instance to perform tasks on containers via Docker API or Docker CLI, Docker Client is a tool to let users interact with Docker host, or Docker Daemon is responsible on listening requests from Docker Client to manage its related objects such as container, image, etc. [27]

Docker is mostly recommended for deployment on microservice architecture, flexible scalability, or applications requiring building once, but running on several servers. Therefore, its advantages to be listed are swifter than normal virtual machines in which can start and stop in a few seconds, flexible on different servers, easy to configure work environment. [27]

2.4.2 Kubernetes

Kubernetes created by Google is an open-source platform that automates management, scalability, and deployment of applications via container. It abstracts some manual processes relating to deploying and scaling containerized applications. Nowadays, containerization via Docker is applied widely in many applications, especially in the production environment. [29]

The orchestration of Kubernetes lets users develop scalable applications with more containers. Kubernetes is compatible with large-scale applications that need to expand afterward. It manages the enforcement of containers through YAML to do with a manifest. Kubernetes mainly manages the Docker host and structure container cluster. Eventually, Kubernetes is optimal since it is scalable, available, portable, and more secure. [29]

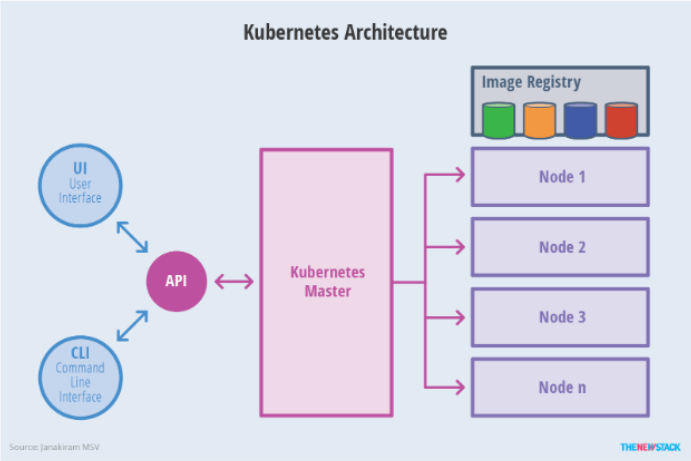

Figure 12. Kubernetes Architecture [29].

3 Implementation

3.1 Project Objective

To reduce the amount of food waste every day in supermarkets, this application builds a platform for two different kinds of users – supermarkets, and other organizations including non-profit organizations, restaurants, and cafeterias, to find and contact each other quickly and easily. In this way, supermarkets will join the Food Supporter application to give away expiring food while other organizations (including the non-profit organizations, restaurants, and cafeterias) will join also the application to receive the given-away food from supermarkets and use for their purposes.

To get the best effect, one type of user only can see the information and contact with another type of user. For example, supermarket users only can look for, see, and get contact with organization users, and vice versa. A supermarket user cannot find, view, or connect to other supermarket users, and similarly for organization users.

The Food Supporter requires some important features as the request feature in which users can create their requests to get rid of or receive the food waste, the chat feature in which two kinds of users can contact each other to discuss food transportation or others after seeing the requests. However, to use those features, users need to register at first. Via the Food Supporter, users also can find the partners' information to get contacts with each other easier.

These are the requirements of the application:

- The homepage can show the application introduction, news feed, features, and a list of partners joining the system.
- Only registered users can view the request page and send messages on the chat page.
- Unregistered users can log in and sign up, or log out in case the users have logged into the system.
- Users can register and sign in as Supermarket Users or Organization Users (including the non-profit organization, restaurant, or cafeteria).
- Users can view and edit their profiles.

- Users can view and create requests on the request page.
- There are two types of requests: offer and support. Offer requests are applied for supermarket users to give away food, and support requests are for organization users to ask for the food they need.
- The requests should have adequate information about what they would like to offer or need supports, place, and time to pick up, or a food list.
- Users can edit and delete all requests they have created.
- Supermarket users can view all requests of all organization users and vice versa.
- Users can send messages and chat in real-time with their partners to discuss requests or pickup arrangements.
- Users can switch to different pages on the application via the navigation bar.

3.2 Project Architecture

The application architecture includes several components or containers, which will be containerized with Docker and use Docker Compose for all to run concurrently during development. This approach can bring a lot of benefits by having the development environment match closely with production, especially when moving forwards to deployment with Kubernetes architecture. If the application runs in containers, it will work the same on the server.

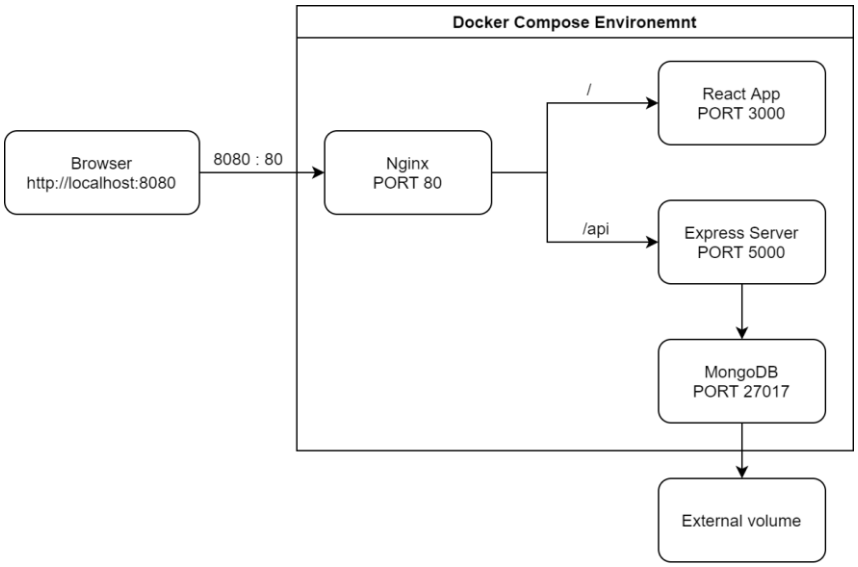

Figure 13. Application Architecture with Docker Compose for development environment.

Figure 13 illustrates all the components of the application and how it works on the development environment. When accessing the application from the browser, the user will first visit an Nginx web server. The server works as a router, it will then decide whether the browser is trying to access the React App or the Express server and route the request to the correct component.

The Express server functions both as RESTful APIs and WebSocket server thanks to the Socket.io library. If the user is going to send messages or chat, the Socket.io will handle the request, otherwise, the Express controllers will process. Next, the server will store or retrieve data from the MongoDB database server if needed, which will then save them to external volume for persistency.

3.3 Project Structure and Setup

3.3.1 Project Structure

The project divides the main components of the application into different folders. There is a separate folder named **k8s** containing all the configuration files for Kubernetes. The Docker Compose and Cloud Build CI/CD configuration files are also placed in the root directory of the project as displayed in figure 14.

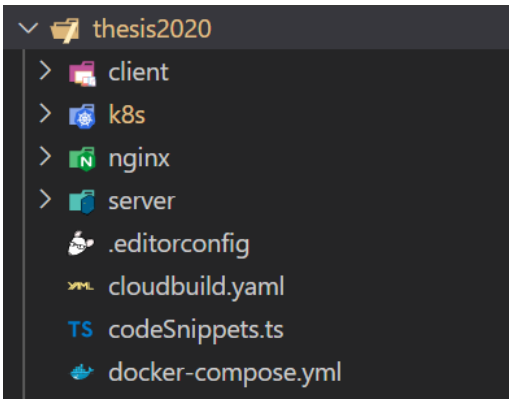

Figure 14. Project structure.

3.3.2 Project Setup

The frontend of the application is a React application created using the **create-react-app** library with the TypeScript template.

```
yarn create react-app client --template typescript
```

Figure 15. The command line to create a React application.

The backend is a Node.js application initialized interactively with the **yarn init** command. Then, dev dependencies are added to the application.

```
yarn add typescript @types/node nodemon concurrently --dev
tsc --init
```

Figure 16. The command lines for adding dev dependencies and tsconfig.json file.

Node.js engine works on the V8 JavaScript Runtime environment, so **typescript** package is needed to compile **.ts** files to **.js**. **@types/node** is a package containing the type definitions of Node.js. Next, **nodemon** is a tool for developing Node.js application by restarting automatically the application when it recognizes changes. **concurrently** is used to run contemporaneously many commands. **tsc --init** is the command to create and add the tsconfig.json file into projects to configure the TypeScript compiler.

```
"scripts": {
  "start": "node dist/index.js",
  "build": "tsc",
  "dev": "concurrently -k -p \"[{name}]\" -n \"TSC,Node\" -c \"yellow.bold,green.bold\"
    \"tsc -w\" \"nodemon dist/index.js\""
},
```

Figure 17. Scripts for commands in package.json file.

Next, the scripts above are added to the package.json file for creating custom commands and running the application. The **build** and **start** commands are to set up and run the application after it has been built. And the **dev** command is used during the development,

it automatically detects files changed in the directory then compiles and restarts the application.

After both front-end and back-end applications have been initialized, they are containerized with Docker by adding Dockerfile.dev files to both **/client** and **/server** directories, which describe the containerization steps for the application.

thesis2020 > client > Dockerfile.dev > ...

```
FROM node:13-alpine
WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn install
COPY ./ ./
CMD ["yarn", "start"]
```

thesis2020 > server > Dockerfile.dev > ...

```
FROM node:13-alpine
WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn install
COPY ./ ./
RUN yarn run build
CMD ["yarn", "run", "dev"]
```

Figure 18. Dockerfile.dev files for client and server containers.

The containerization process initiates by creating a container with the node:13-alpine image pulled from the Docker Hub server, then setting up the working directory for the application in the container, then copying package.json and yarn.lock file to the working directory and install all application dependencies. The next steps are copying source code to the container and creating a default start-command for the container. A difference in the Dockerfile.dev in server versus client is that it has an additional step for building the application before adding the start-command. This is the result that the React App **start** command has compiled TypeScript to JavaScript under the hood while the Node.js needs to run the compiler manually.

Nginx is a lightweight and powerful web server that is used for routing traffic to client and server containers. The Nginx configuration can be customized with the nginx.conf file. The Nginx routing configuration is shown in figure 19 below.

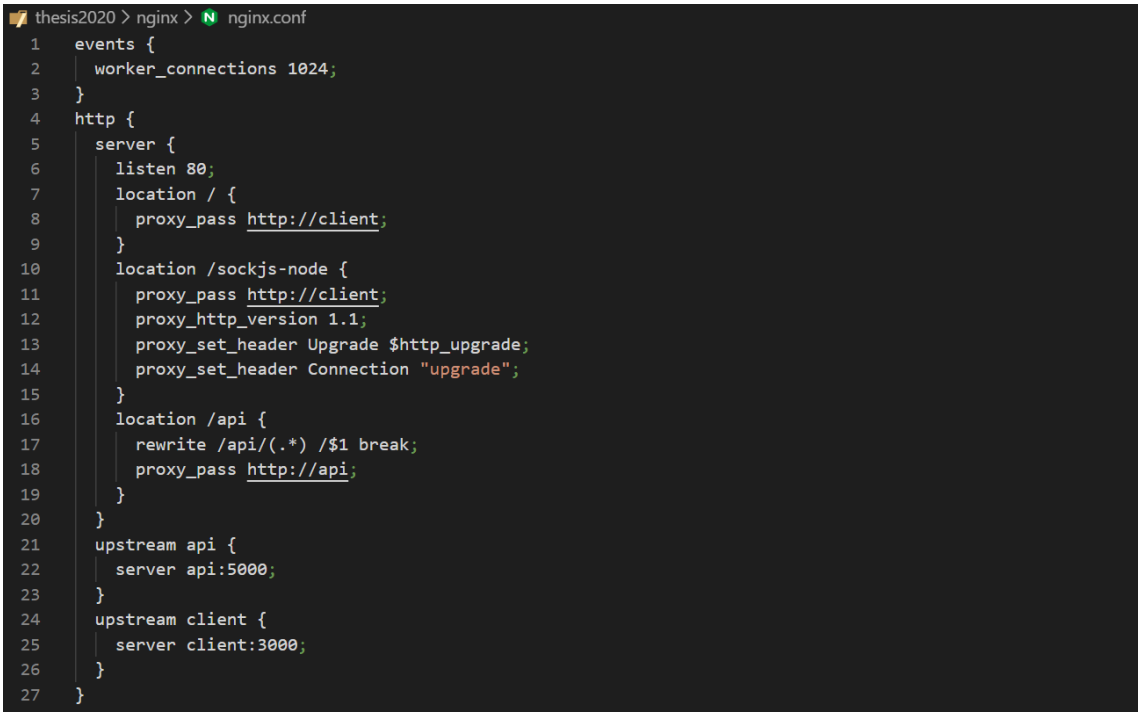

Figure 19. Nginx configuration file.

Then, the custom configuration file is copied to nginx container which is pulled from Docker Hub by the Dockerfile.dev added into folder **/nginx**.

Figure 20. Dockerfile.dev file for Nginx.

The next step is creating a docker-compose.yml file to manage Docker containers. Docker Compose is a tool for specifying and executing many contemporary Docker containers. For the Compose, the whole application's services are configured with a YAML file. So then, with only one command, it could create and activate all services together at the same time from the configuration file.

thesis2020 > docker-compose.yml

```yaml
version: "3"                        # Use Compose file version 3 format
services:                           # Contains configuration that is applied to containers
  api:                              # Configuration for server container
    build:                          # Build configuration for server
      dockerfile: Dockerfile.dev
      context: ./server
    volumes:                        # Mount local volume with container volume
      - /app/node_modules
      - ./server:/app
  client:                           # Configuration for React App container
    tty: true                       # Use interactive mode for running React App
    build:                          # Build configuration for React App
      dockerfile: Dockerfile.dev
      context: ./client
    volumes:                        # Mount local volume with container volume
      - /app/node_modules
      - ./client:/app
  nginx:                            # Configuration for nginx
    restart: always                 # Restart the server if crashed
    build:                          # Build configuration for React App
      dockerfile: Dockerfile.dev
      context: ./nginx
    ports:                          # Map port 8080 of localhost to port 80 of nginx
      - "8080:80"
```

Figure 21. Docker Compose configuration file.

In the docker-compose.yml file in figure 21, firstly it is necessary to define which Compose file format version is used, as this project using version 3. Next, define all the services that will run simultaneously. Each service has a built configuration that defines the location of Dockerfile and the folder used to build the container, and a volume configuration to mount the host directory with the container.

3.4 Front-end Implementation

3.4.1 React Application Structure

The front-end uses a template created by **create-react-app** so it has the setup of a React Application, such as the node_modules folder with all dependencies modules, the public folder with static files, the src folder with source code, and the general configuration files for React App in the root directory.

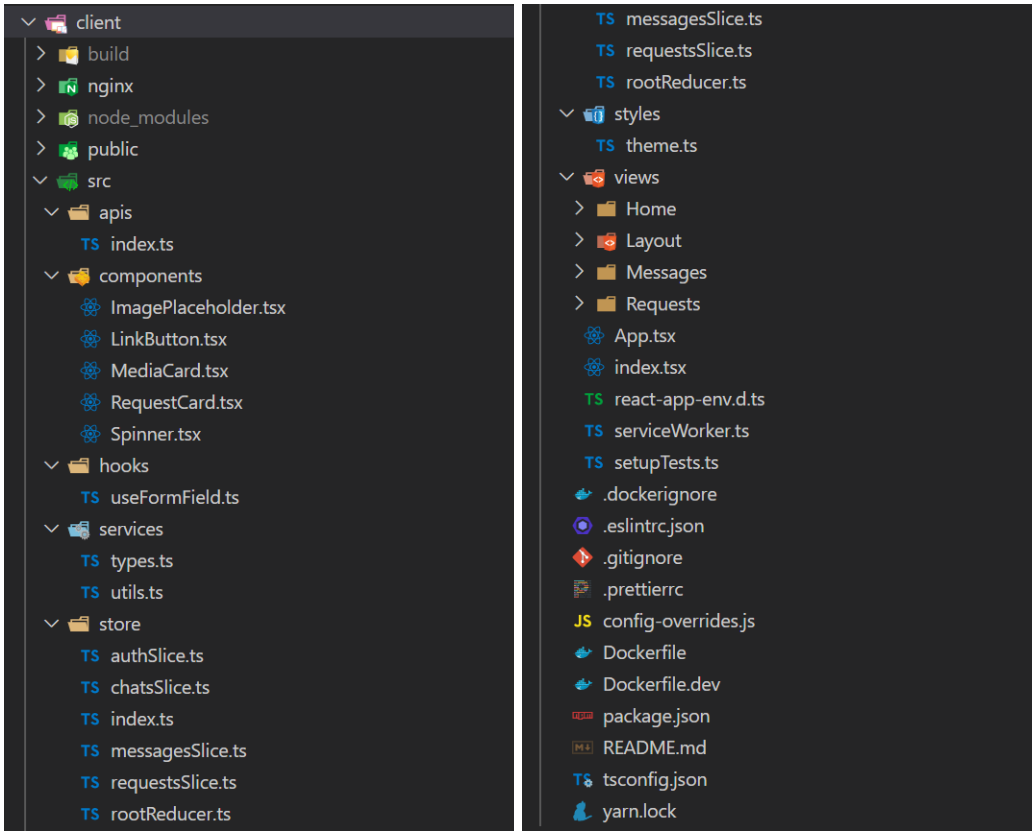

Figure 22. React application folder structure.

3.4.2 Tools and Libraries

Material-UI

Material-UI is an open-source project providing a large assemblage of React components that follow Google's Material Design. It is not only gorgeous but simple for using and customizing as well. With over 56-thousand star and used by 205 thousand repositories via GitHub till April 2020 [30], Material-UI is one of the best UI libraries for React. This project uses Material-UI to implement the user interface rapidly and focus on developing features.

thesis2020 > client > src > components > Spinner.tsx > ...

```tsx
import React from 'react';
import { makeStyles, Theme, createStyles } from '@material-ui/core/styles';
import { CircularProgress, Box } from '@material-ui/core';

interface Props {
  size?: number;
}

const Spinner = ({ size = 64 }: Props) => {
  const classes = useStyles(size);
  return (
    <Box className={classes.root}>
      <CircularProgress />
    </Box>
  );
};

const useStyles = makeStyles((theme: Theme) =>
  createStyles({
    root: {
      background: 'transparent',
      width: (size: number) => size,
      height: (size: number) => size,
      borderRadius: '100%',
      border: (size: number) => `${size / 10}px solid ${theme.palette.primary.light}`,
      borderBottomColor: 'transparent',
      display: 'inline-block',
      animation: '$clip 0.75s 0s infinite linear',
      animationFillMode: 'both',
    },
    '@keyframes clip': {
      '0%': { transform: 'rotate(0deg) scale(1)' },
      '50%': { transform: 'rotate(180deg) scale(0.8)' },
      '100%': { transform: 'rotate(360deg) scale(1)' },
    },
  }),
);
```

Figure 23. The Spinner component that is built using Material-UI.

This Spinner component is built on top of the Material-UI CircularProgress component with a custom style. The makeStyles and createStyles functions help in creating useStyles hook from Object-like CSS styles, which is called inside the component then injects the CSS selectors to className of JSX Element.

Redux Toolkit

Redux Toolkit is an official subjective toolset developed by the Redux team for better efficiency in Redux development and become an official Redux template for Create-React-App since 18.02.2020 [31]. Redux Toolkit changed completely the improved formula of the Redux application following an opinionated structure.

As the previous structure, two folders are existing: the **src/reducers** for reducer logics and **src/actions** for action creators. And they are set discretely with other feature components. Redux Toolkit encourages using the new approached way – feature folder. That is all reducer logics and action creators which are in the same file named slice, and in a folder with components having similar features. This helps the feature development effectively, code-line reduction, and avoids the modification of many files in different folders when needed to update one or another feature. [32]

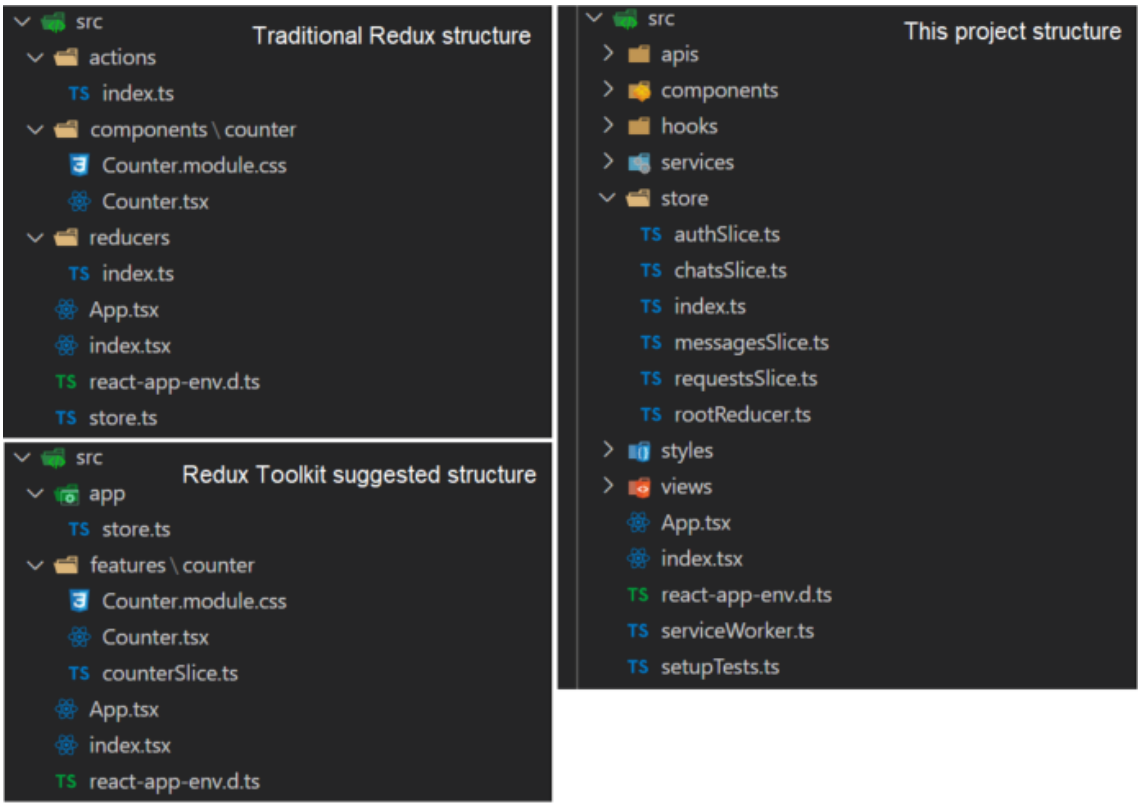

Figure 24. Traditional Redux, new Redux Toolkit, and Food Supporter structures.

However, this structure tends to be more complicated when applying in the situation that a Redux state or action creator is used in many different features. Hence, this project aims for benefits taken while using Redux Toolkit functionalities, but the store, slices, and all related Redux modules are placed in a separate folder with the name **store**. The slice implementation will be discussed more specifically in feature implementation sections.

Axios

Axios is an HTTP client based on Promise, which is used to support the implementation of API applications simply to more complexly, and used on both browsers and Node.js server.

thesis2020 > client > src > apis > TS index.ts > ...

```ts
import axios from 'axios';

const axiosInstance = axios.create({
  baseURL: '/api',
});

export const setTokenHeader = (token?: string) => {
  if (token) {
    axiosInstance.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  } else {
    delete axiosInstance.defaults.headers.common['Authorization'];
  }
};

export default axiosInstance;
```

Figure 25. The axios instance configuration file.

This module creates a new instance of Axios with a custom baseURL to the API server. This is convenient for data fetching from a server and avoidance of the configuration writing each call. Also, this module provides a helper function called setTokenHeader, which is triggered whenever a user logs in or logs out to add or remove the authentication token to the request header.

3.4.3 Homepage

The homepage is the first-page user opens the application. Its layout consists of 2 major components: the Carousel and the list of organization partners. The Carousel is an automatic slideshow switching images every five seconds, which is for the marketing purpose of the application, and appears news. And the partner list is a list of cards showing the information of supermarket users or organization users with logo, photos, name, and description.

Figure X shows how the homepage looks like with an unregistered user. All the data here is generated automatically and randomly from the API server for prototype purposes.

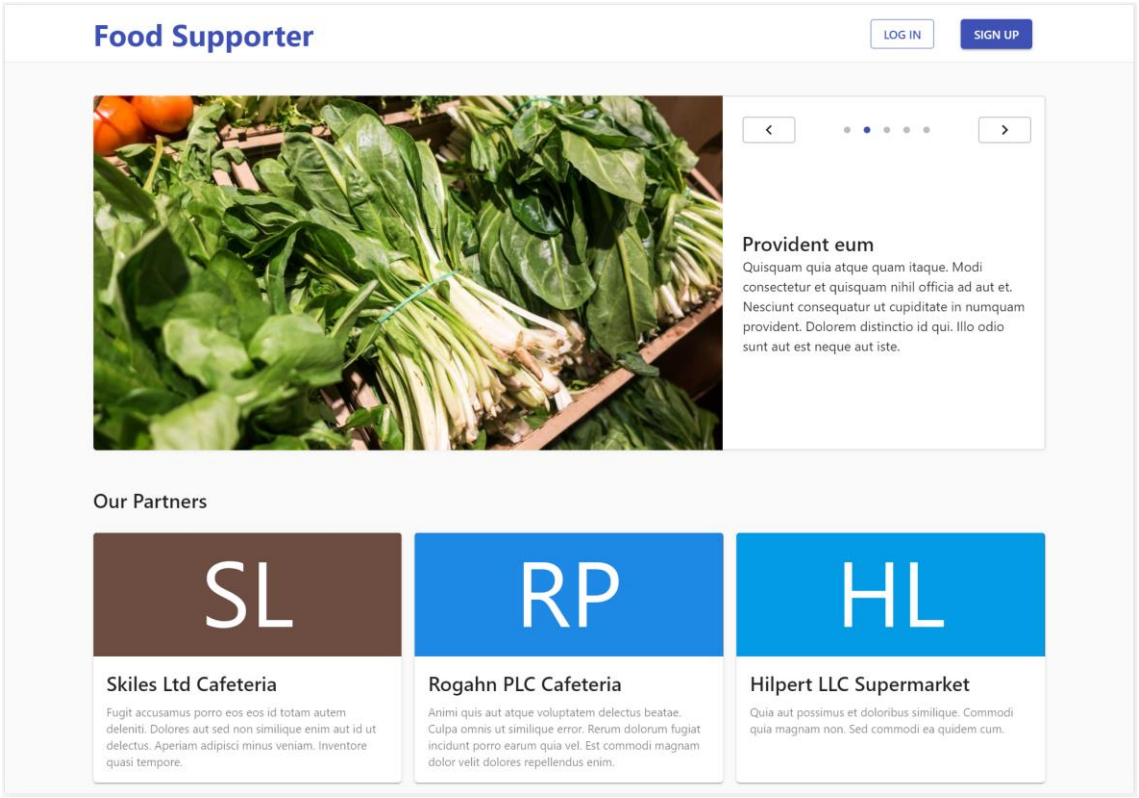

Figure 26. Homepage view.

3.4.4 Routing and Authentication

About the application routing, the unauthorized user can access only Home in the navigation bar (homepage), yet, after registering and logging in, the user is allowed to access all pages which appear in the navigation.

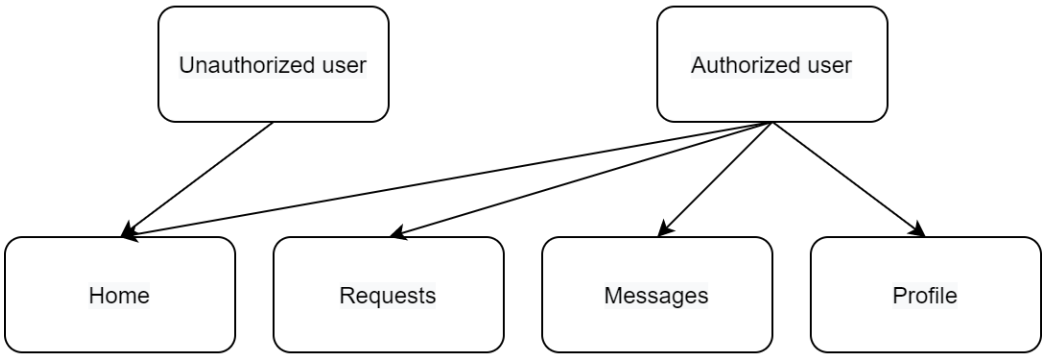

Figure 27. The routing logic of the application.

Before signing in, the user only can see two buttons: Log in and Sign up in the navigation bar. Then, after logging into the system, those two buttons are replaced with the Log out button, and the Menu bar is shown to help the user in navigating to other pages.

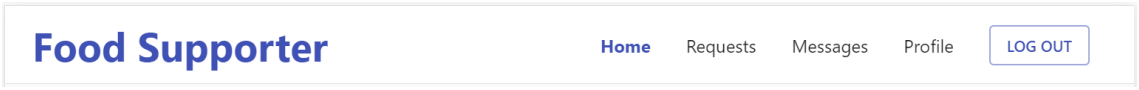

Figure 28. The view of the navigation bar after logging in.

Deepened little into the routing logic, it is created with the react-router-dom library and placed in the main App component.

thesis2020 > client > src > App.tsx > ...

```
13  const App = () => {
14    const dispatch = useDispatch();
15    const isLoggedIn = useSelector((state: RootState) => state.auth.isLoggedIn);
16
17    useEffect(() => {
18      if (validToken()) {
19        dispatch(initialAuth());
20      } else {
21        dispatch(logOut());
22      }
23    }, [dispatch]);
24
25    return (
26      <Router>
27        <Layout>
28          <Switch>
29            <Route exact path="/" component={Home} />
30            {isLoggedIn && (
31              <>
32                <Route exact path="/requests" component={Requests} />
33                <Route path="/requests/new" component={NewRequest} />
34                <Route path="/messages" component={Messages} />
35                <Route path="/profile" component={Profile} />
36              </>
37            )}
38            <Redirect to="/" />
39          </Switch>
40        </Layout>
41      </Router>
42    );
43  };
```

Figure 29. The main App component.

The useSelector hook of the react-redux library is used to get the login state of the user from the Redux store. The user is only able to access the requests, messages, or profile page when signing in, otherwise, it will redirect to the home page. The useEffect hook

runs once the app is loaded. If a user has a valid token saved on the browser localStorage, the initialAuth action creator would be called and dispatched to fetch user data, or the auth state would be reset by dispatching the logout action.

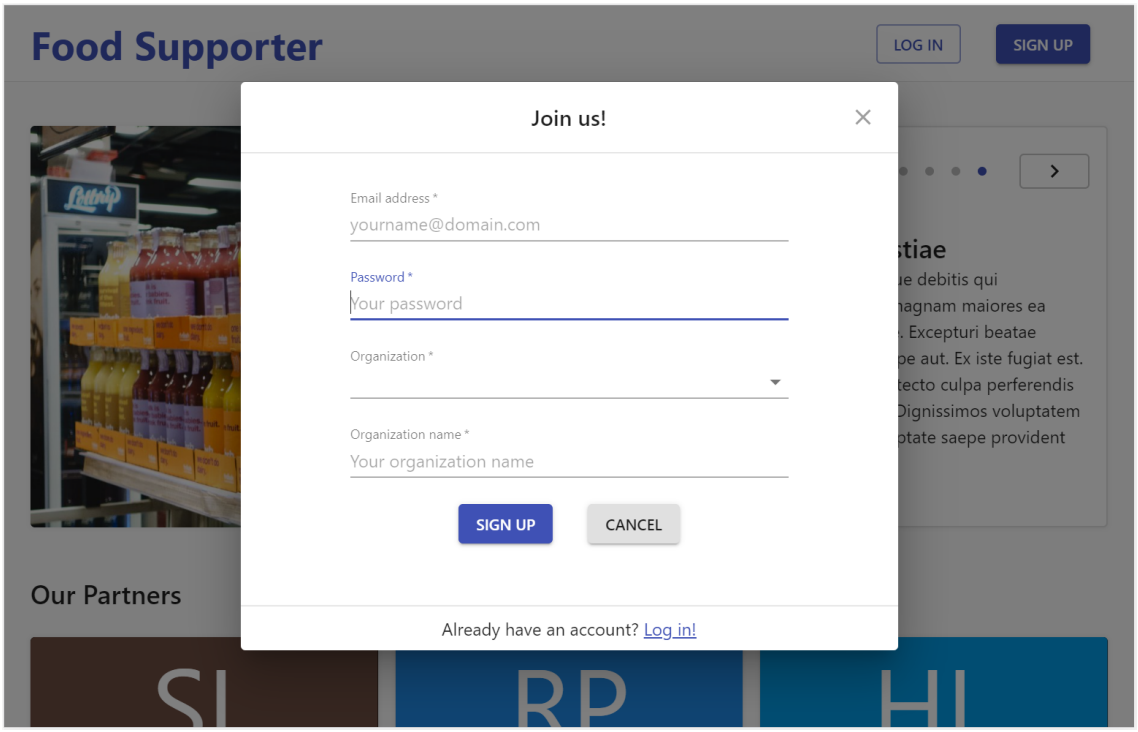

Figure 30. User register dialog.

The user can sign in or sign up in the appeared dialog (as figure 30) when clicking onto the Login or Signup buttons. Or, the user can switch between the login and signup through a link at the bottom of the dialog.

thesis2020 > client > src > store > TS authSlice.ts > ...

```ts
export interface AuthState {
  user: User | null;
  isLoggedIn: boolean;
  isLoading: boolean;
  error: string | null;
}

const initialState: AuthState = {
  user: null,
  isLoggedIn: false,
  isLoading: false,
  error: null,
};

const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    authStart: (state): AuthState => ({ ...state, isLoading: true, error: null }),
    authSuccess: (state, action: PayloadAction<User>): AuthState =>
      ({ ...state, isLoading: false, isLoggedIn: true, error: null, user: action.payload }),
    authFail: (state, action: PayloadAction<string>): AuthState =>
      ({ ...state, isLoading: false, error: action.payload }),
    authReset: (): AuthState => initialState,
  },
});

export const { authStart, authSuccess, authFail, authReset } = authSlice.actions;

export default authSlice.reducer;
```

Figure 31. The auth slice.

The authentication state of users is stored in the auth state of the Redux store. It contains user information, login state, authentication loading state, and error message in case the authentication is failed. For the execution principle of an authentication task, the createSlice function from Redux Toolkit gets an assemblage of the reducer functions, slice name, and an initial state value, then, automatically generates a slice reducer along with action creators and action types relatively. The authSlice Reducers and action creators describe an authentication process.

The authentication progress begins with the authStart transforming the loading state into true after the user fills in the form and chooses login or signup buttons. If the authentication happens successfully, the token responded from the server will be added to the request header to authorize the user with the server and saved to browser localStorage. After that, the authSuccess will save user information into the auth state and take over the sign-in status to true and the loading stops. Unless the authentication is successful, the authFail will store the error message returned from the server into the auth state and

loading stops. Lastly, the authReset will reset the auth state to initial state value when the user signs out.

3.4.5 Food Request

As mentioned earlier, this is the most important feature in solving the food waste problem. Via Food Request feature, supermarket users can post the request as an offer to get rid of their food waste whereas organization users can post the request as food support to ask for help in receiving food. However, the offer requests are viewed by the organization users only, and vice versa. Therefore, none of the users in the same type can view or contact each other.

The request page comprises two tabs, in which the first tab shows all the offer or support requests coming from other kinds of users, and the second one is where the user can view their own requests.

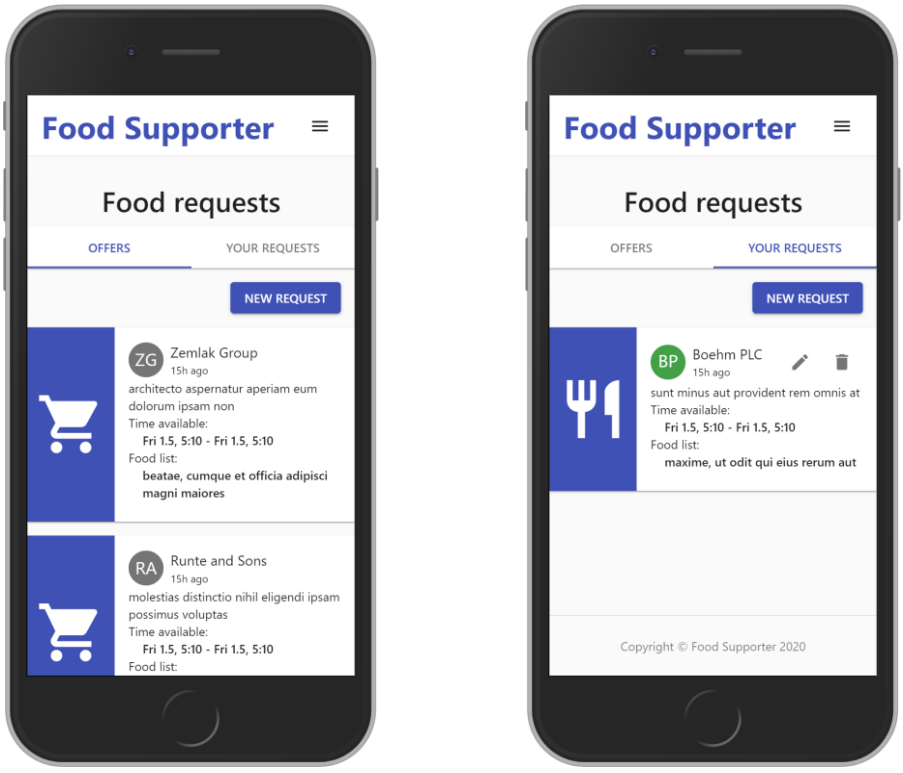

Figure 32. Request page on the mobile view of a restaurant user.

Each request is a card-component with necessary information such as the title or purpose of the quest, the available time for pickup, place, or a list of food. Users can edit and remove the requests created on their own.

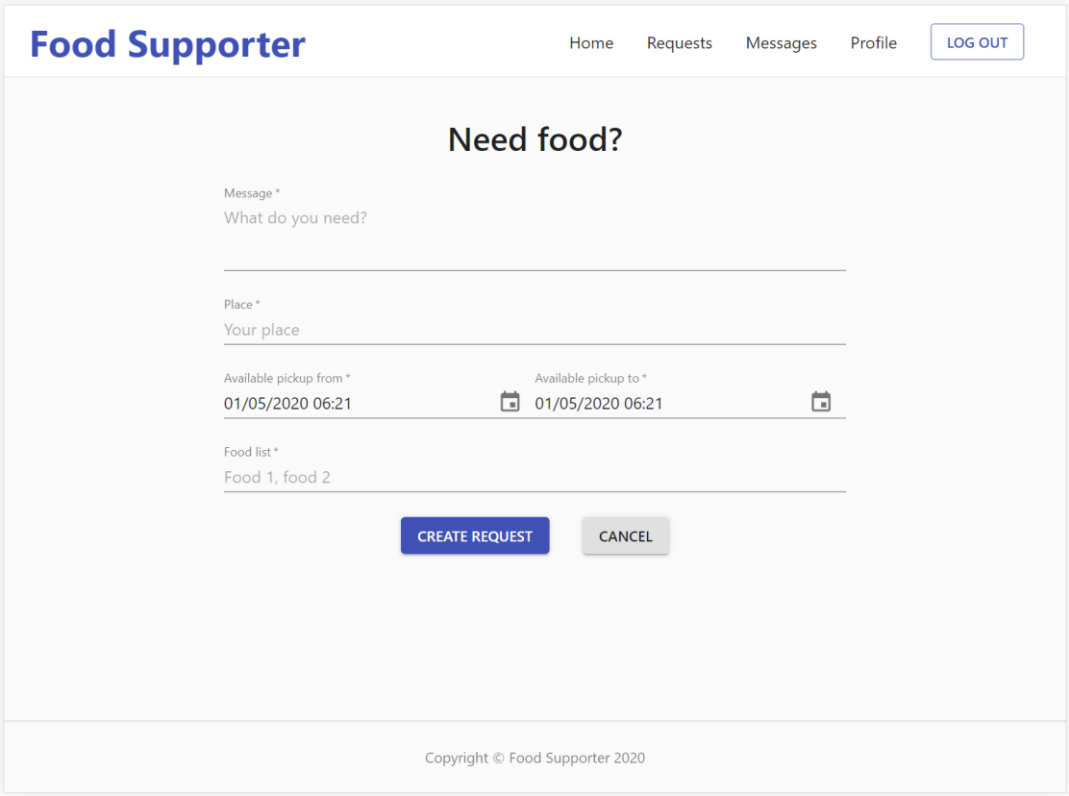

Figure 33. Create new request page.

The Create new request page appears when the user clicks onto the New Request button. Here user can make a new request via the customized form for distinctive users.

```
thesis2020 > client > src > hooks > TS useFormField.ts > ...
3   export interface FormFieldProps<T> {
4     value: T;
5     error: string;
6     handleChange: (value: T) => void;
7     setError: (error: string) => void;
8   }
9
10  export const useFormField = <T>(initialValue: T): FormFieldProps<T> => {
11    const [value, setValue] = useState<T>(initialValue);
12    const [error, setError] = useState<string>('');
13    return { value, error, setError, handleChange: setValue };
14  };
```

Figure 34. The useFormField custom hook.

While using the React Hook, defining state variables can be duplicated many times with one similar type of data. That causes the redundant and unclean code. Consequently, the useFormField custom hook in Figure 33 is created to abstract the form field state management. This hook uses a generic type for the value, so it could be reused easily for any form field and any type of data.

3.4.6 Real-time Messaging

The message feature is designed to help users in contacting and discussing each other quickly about the food requests. If the request feature is the most important role in this project, the message feature is like a catalyst to encourage reducing food waste by connecting each other.

The message page includes two main components: ChatList and ChatArea. ChatList is a list of conversations with partners with whom the user gets contact. Every conversation is a ChatItem component showing information as logo, name of partners, the content of the last message, and last message time. All message content of a conversation will be loaded to ChatArea when the user chooses the conversation on the ChatList. Messages are input via an input field in the MessageInput component inside the ChatArea, then sent by clicking the send button next to or using the enter key on the keyboard.

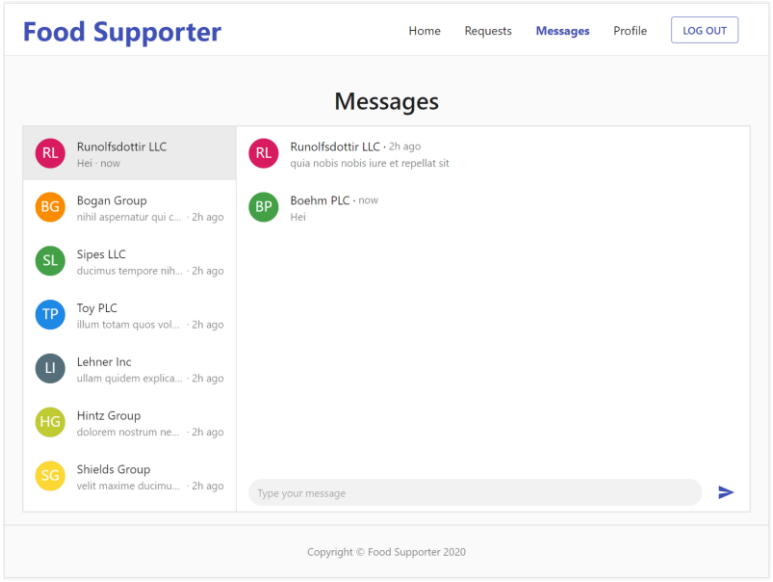

Figure 35. Real-time Messaging page.

The Real-time Messaging feature used for this application is developed on the Web-Socket technology with the Socket.io library. To use this library on the frontend, the **socket.io-client** package is required to be installed by running the command **yarn add socket.io-client**.

Figure 36. Socket.io client set up in the MessageInput component.

The WebSocket connection is set up in the MessageInput component, which is the last in the component hierarchy to avoid unnecessary re-rendering of child components. The socket.io client connects to the socket.io server via the path **/api/chatlive**. This connection does not need to be recomputed, so it is wrapped inside the React useMemo hook to memoize its value.

The socket subscribes to the event **message** from the server. When receiving messages and the new chat conversation, it will update the current active chat conversation, chat list, and message into the Redux state through the dispatch function created from the useDispatch hook. The socket will remove all listeners and close the connection when the user switches into other pages.

```
thesis2020 > client > src > views > Messages > MessageInput.tsx > ...
42    const [inputMessage, setInputMessage] = useState('');
43    const onSubmit = (
44      e: React.FormEvent<HTMLFormElement> | React.MouseEvent<HTMLButtonElement, MouseEvent>,
45    ) => {
46      e.preventDefault();
47      e.stopPropagation();
48      if (inputMessage.length) {
49        socket.emit('message', {
50          content: inputMessage,
51          from: currentUserId,
52          to: activeChat.to.id,
53          chatId: activeChat.id,
54        });
55        setInputMessage('');
56      }
57    };
```

Figure 37. The onSubmit function to send message.

The input message from the user is stored in the inputMessage state. When clicking on the send button or pressing the enter button on the keyboard, the onSubmit event is triggered. And, in case the inputMessage is not empty, it will be published onto the server through the socket connection, then set the input field to be empty.

3.4.7 Debugging

Redux Toolkit has a default Redux Devtool set up under the hood, hence, it is unnecessary to set up manually as the traditional Redux development. The Redux Devetool helps in debugging more conveniently and easing to jumping forward and backward between states. Also, the action types generated automatically from the createSlice as logic **sliceName/reducerKey** are beneficial to the debug.

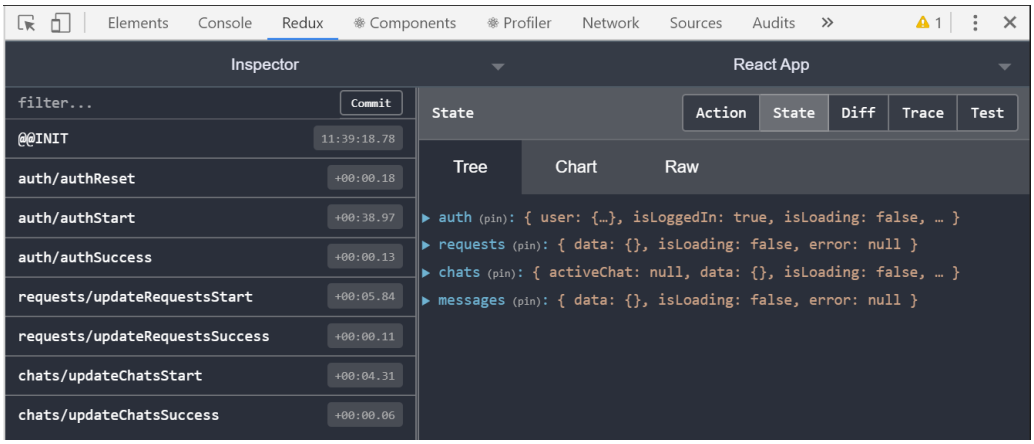

Figure 38. Redux devtool on Google Chrome browser.

3.5 Back-end Implementation

3.5.1 Database

In this section, the database of this project will be explained specifically. Due to the scalability and flexibility, MongoDB was chosen to use for this project's database. The schemas design will be the first step of the progress before deepening into the database implementation.

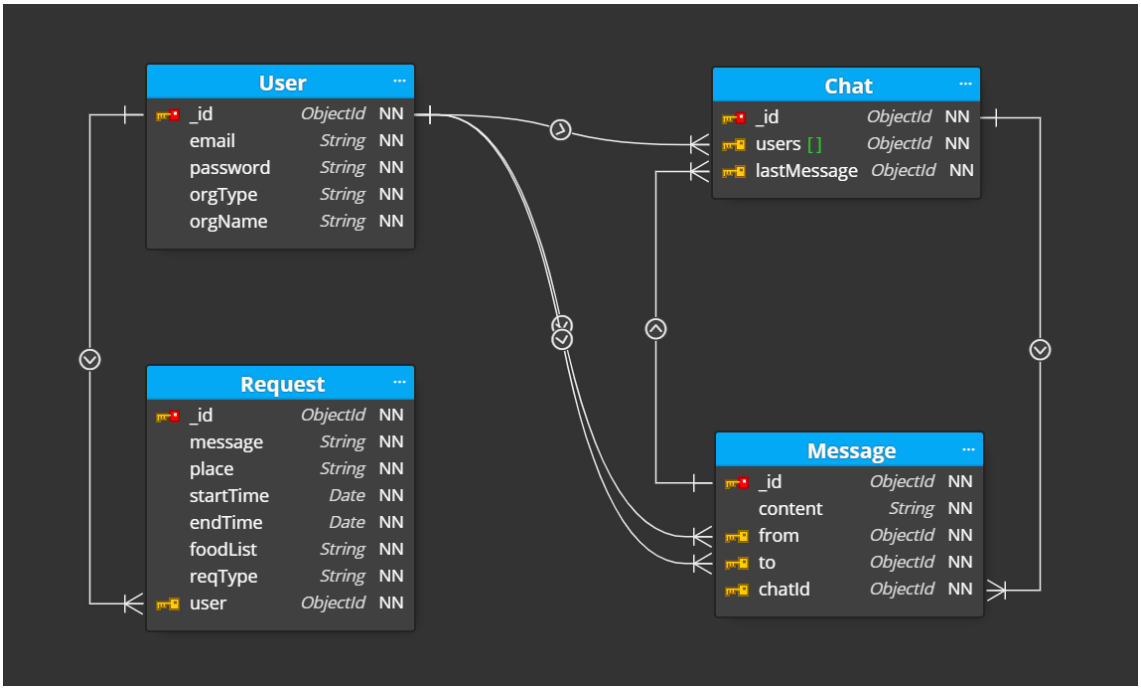

Figure 39. MongoDB Schema Design for Food Supporter.

The database of this project consists of 4 collections: User, Request, Chat, and Message. A user can create many requests, so the request will keep user property as a reference key. Also, a user can create a lot of chat conversations with other users. Besides that, a chat can comprise many messages. That is why the chat is put in a separate collection and keeps the reference to participant users in the **users** array. The chat contains the reference to the last message to access data as well. A message document will store the information of the sender, receiver, message content, and reference to the chat via the chatId property.

Next, the development environment with MongoDB can be set up easily ad quickly by the service declaration for MongoDB using the Mongo image from Docker Hub in the docker-compose.yml file.

thesis2020 > docker-compose.yml

```
1   version: "3"
2   services:
3     nginx: ···
10    api:
11      build: ···
14      volumes: ···
17      environment:
18        - MONGO_USERNAME=mongo_username
19        - MONGO_PASSWORD=mongo_password
20        - MONGO_HOST=mongo
21        - MONGO_DATABASE=mongo
22        - MONGO_PORT=27017
23    client: ···
32    mongo:
33      image: mongo
34      restart: always
35      volumes:
36        - mongodbdata:/data/db
37      environment:
38        - MONGO_INITDB_ROOT_USERNAME=mongo_username
39        - MONGO_INITDB_ROOT_PASSWORD=mongo_password
40  volumes:
41    mongodbdata:
42      external: true
```

Figure 40. Docker Compose file with mongo service.

MongoDB service in the docker-compose.yaml is declared besides services for the client, server, and Nginx. To ensure the persistency and prevent data from losing when the database stops or is deleted, the data is stored to the external volume **mongodbdata** on the host machine. The external volume can be created by the command **docker volume create mongodbdata**.

The root username and password which are used to access the database can be configured through the environment variables declared with the **environment** key. For security reasons, the environment variables are only applied for local development and differentiate with the production. Then, the database username, password as well as other database credentials used to connect the API server to the database are saved to its environment variables to avoid that information being used directly on the source code.

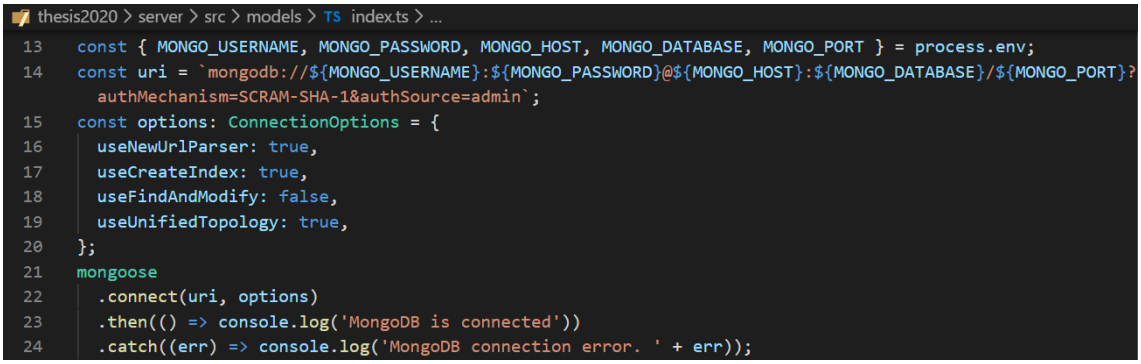

Figure 41. Database connection setup with Mongoose on API server.

The API server connects to the database via Mongoose, which is an ODM library for MongoDB and Node.js. Mongoose provides a strict modeling environment to manage data, validate the schema model, transform JavaScript objects to store into the database whilst still maintaining flexibility as a MongoDB strength.

3.5.2 RESTful APIs and WebSocket

The server is built on the Node.js, Express, and Socket.io, hence, it can support both RESTful APIs requests and WebSocket connection. The back-end implementation initiates by setting up the Express server.

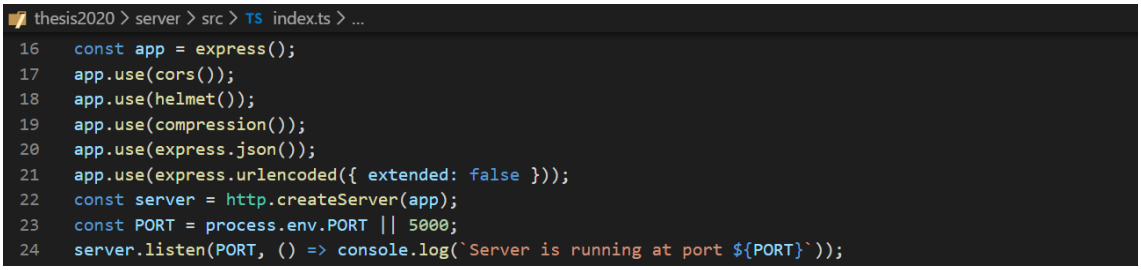

Figure 42. Express server setup.

The Express server is firstly configured with the middleware that supports further implementation. Then, it is wrapped up in the HTTP server instead of using its own server directly as a typical Express application. The set-up adds that stage to support the Socket.io later. Lastly, the server listens on port 5000 by default if PORT is not defined in the environment variables.

The server handles a lot of distinctive REST APIs in serving different purposes and executes distinct data.

thesis2020 > server > src > TS index.ts > ...

```
26  app.use('/public', publicController);
27  app.use('/auth', authController);
28  app.use('/profile/', auth, userController);
29  app.use('/requests/', auth, requestController);
30  app.use('/chats/', auth, chatController);
31  app.use('/chatlive/', auth, chatLiveController);
32  app.get('/*', (req, res) => res.send('Welcome to Food Help APIs!'));
33  app.use(errorHandler);
```

thesis2020 > server > src > middleware > TS error.ts > ...

```
3  interface HttpException extends Error {
4    status: number;
5  }
6  export const errorHandler = (err: HttpException, req: Request, res: Response, next: NextFunction) =>
7    res.status(err.status || 500).json({ error: err.message || 'Server internal error!' });
```

Figure 43. The RESTful APIs routing and custom error handler.

The publicController is responsible for solving requests from the Homepage whereas the authController is for the login or sign-up requests. These controllers do not require the requests to be authorized. And other controllers are on duty of the user signing in, so requests should be validated through the auth middleware. The errorHandler middleware will handle and send an error message back to the frontend in case an error occurs during request processing.

thesis2020 > server > src > models > TS User.ts > ...

```
60  userSchema.methods.generateJwt = async function (next): Promise<string> {
61    try {
62      return jwt.sign({ id: this.id }, process.env.SECRET_KEY as string, { expiresIn: '8h' });
63    } catch (err) {
64      return next(err);
65    }
66  };
```

Figure 44. Adding generate JWT function to User schema.

JSON Web Token or JWT is used for keeping the server secured and protecting the sensitive information of the user. The JWT is created and returned to the frontend when a user signs in or signs up. The JWT will be expired in eight hours. Only requests sent to the server with the valid JWT are considered as authorized and can access secured data from the server.

thesis2020 > server > src > middleware > TS auth.ts > ...

```ts
import jwt from 'express-jwt';
import { Request } from 'express';
export interface AuthRequest extends Request {
  auth?: { id: string };
}
export default jwt({ secret: process.env.SECRET_KEY as string, userProperty: 'auth' });
```

Figure 45. The auth middleware function.

The **express-jwt** is applied as the authentication middleware which validates the JWT sent to the server. If a request is valid through the process, the decoded token including userId will be added to the **auth** property of the request for the controllers to handle it in the following steps.

thesis2020 > server > src > TS index.ts > ...

```ts
import socketIo from 'socket.io';
const io = socketIo(server, { path: '/chatlive' });
io.on('connection', (socket) => socketController(socket, io));
```

Figure 46. Socket.io server setup.

The Socket.io is set up by using the prior installed Express server and operates on the **/chatlive** route.

thesis2020 > server > src > controllers > TS chatLiveController.ts > ...

```ts
export const socketController = (socket: Socket, io: Server) => {
  socket.on('disconnect', () => console.log('user disconnected'));
  socket.on('message', async (data: Data) => {
    try {
      if (!data.content || !data.from || !data.to) throw new Error('Invalid request!');
      if (data.chatId) {
        const message = await db.Message.create(data);
        const [populatedMessage, updatedChat] = await Promise.all([
          populateMessage(message),
          updateChatLastMessage(data.chatId, message.id),
          data.isNewChat ? updateUserWithChats(data.to, data.chatId) : null,
        ]);
        return io.emit('message', { message: populatedMessage, chat: updatedChat });
      }
      const chat = await db.Chat.create({ users: [data.from, data.to] });
      const message = await db.Message.create({ ...data, chatId: chat.id });
      const [populatedMessage, updatedChat] = await Promise.all([
        populateMessage(message),
        updateChatLastMessage(chat.id, message.id),
      ]);
      return io.emit('message', { message: populatedMessage, chat: updatedChat });
    } catch (err) {
      return io.emit('error', { error: err.message });
    }
  });
};
```

Figure 47. The socket controller for handling real-time messages.

The socketController handles the message transmission via the WebSocket connection. Message data, after receipt and validation, will be added to the Messages collection. On the other hand, if the conversation does not yet exist, a new conversation is created in the Chat collection. Afterward, the created message and updated chat conversation are responded to the client.

3.6 Kubernetes Orchestration

3.6.1 Production Containerization

Due to the different aspects of development and production, the containerization strategy for services on production is designed for the most optimized performance and scalability. The server containerization does not change much, yet the client does because of the nature of the React application.

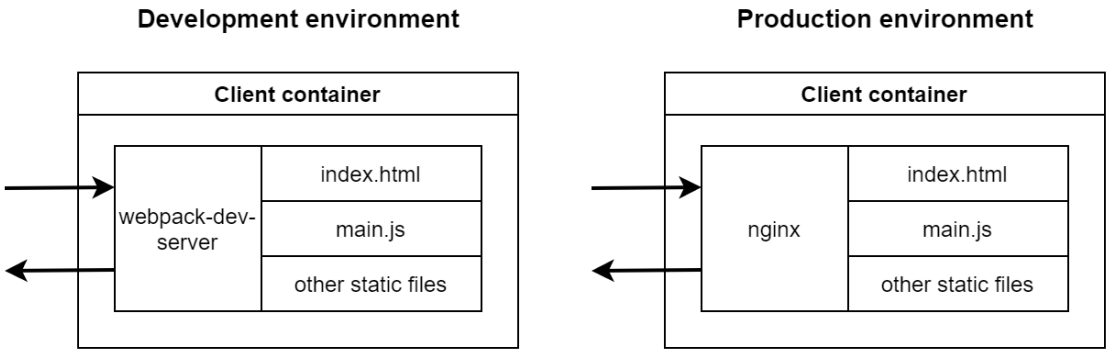

Figure 48. The difference in client containerization on development and production.

Since React uses the webpack-dev-server supporting the development in numerous features which become exceeding and lower the performance on production, React provides a production to generate static files. Then, a web server is needed to serve the static files. Nginx comes to solve this issue as a lightweight and powerful web server altogether with the multi-step Docker builds.

thesis2020 > client > Dockerfile > ...

```
FROM node:13-alpine AS builder
WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn install
COPY ./ ./
RUN yarn build

FROM nginx
EXPOSE 3000
COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/build /usr/share/nginx/html
```

Figure 49. The multi-step Docker builds for client containerization.

The first step in the multi-step Docker builds is similar to development build, but ends by running **yarn build** to create the production build with static files. The next stage is to generate a container from the Nginx image, copy the Nginx configuration file and the built files from the previous one to the Nginx container, and expose the container by port 3000. WebSocket connection support is also enabled in the Nginx configuration.

thesis2020 > client > nginx > default.conf

```
server {
  listen 3000;
  location / {
    root /usr/share/nginx/html;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
    # enable WebSockets
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $host;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
  }
}
```

Figure 50. The Nginx configuration file for React application with WebSocket enabled.

3.6.2 Kubernetes Cluster

A Kubernetes cluster is an assemblage of node machines running the containerized applications. A Kubernetes cluster consists of at least a master node and a worker node. The responsibility of the master node is to keep the cluster in maintaining the desired state which is described in the Kubernetes configuration files. The worker node, meanwhile, is the real workplace by containing and running applications.

Kubernetes can be used on the local machine to develop with the Docker Desktop application by a built-in single-node Kubernetes cluster.

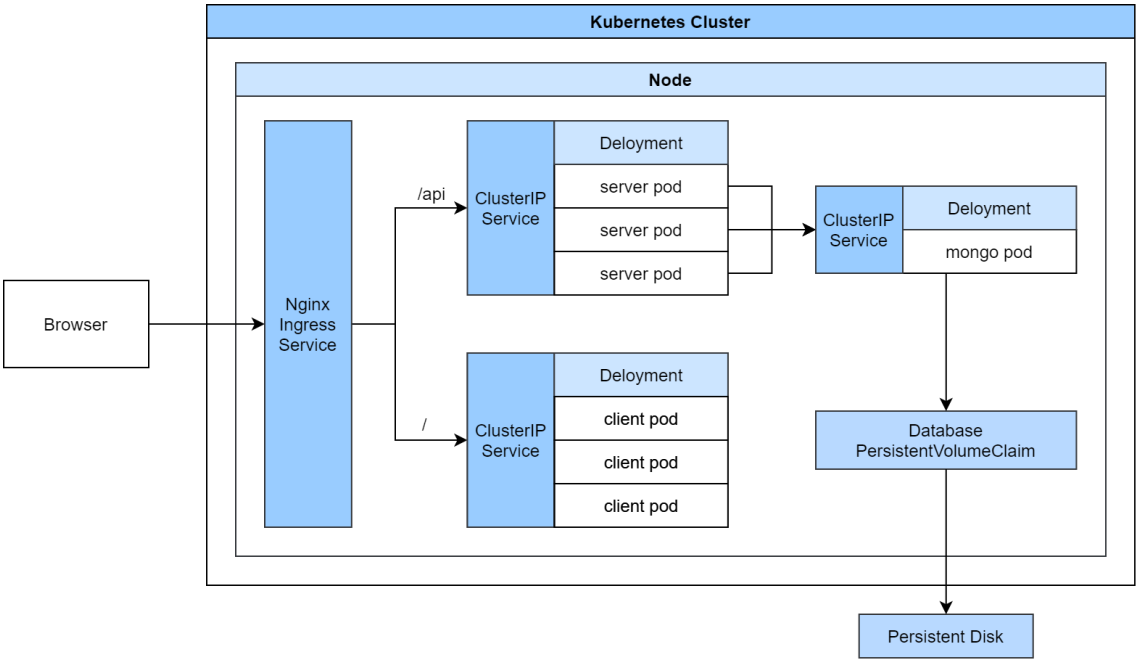

Figure 51. The architecture on the Kubernetes cluster with a worker node.

3.6.3 Kubernetes Configuration Files

Working with Kubernetes requires an assemblage of the Kubernetes configuration files, which uses Kubernetes API objects to declare and describe the desired state of the cluster. At a minimum, each containerized application needs two Kubernetes objects to show workloads be run and the network service itself. Additionally, storage resources, as well as the network load balancing for the whole system, also need to declare with other objects.

The figure below presents the Kubernetes configuration file for a client application that comprises Deployment and ClusterIP Service objects. The Deployment object depicts which image is necessary for running containers, the port that the container reveals, and the number of pod replicas to be deployed. The ClusterIP Service object exposes the set of pods created by the Deployment object with the cluster internal network.

thesis2020 > k8s > client-deployment-service.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      component: web
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
        - name: client
          image: gcr.io/thesis2020-04/client
          ports:
            - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: client-cluster-ip-service
spec:
  type: ClusterIP
  selector:
    component: web
  ports:
    - port: 3000
      targetPort: 3000
```

Figure 52. The Kubernetes configuration file for the client application.

3.6.4 Load Balancing

Besides having the ClusterIP Service to display the application pods with the internal network inside the cluster, the application should be accessed from the outside world. This is done with the Ingress object by exposing a group of ClusterIP Service objects with the external network and balancing the network traffic between the in and out of the cluster.

This project uses the Nginx Ingress, an open-source project from the Kubernetes community in the load balancing. The Nginx Ingress supports WebSocket connection by default, the only requirement is to increase the proxy read and send timeout to maintain the connection. Besides, to proceed with the horizontal scaling with the replica extent of the WebSocket server, it is required to enable the sticky session to ensure a particular user connecting to the same target server during the session.

thesis2020 > k8s > ingress-service.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-service
  annotations:
    kubernetes.io/ingress.class: nginx                          # Use Nginx Ingress
    nginx.ingress.kubernetes.io/rewrite-target: /$1             # Solve react router routing issue
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"      # Increase the proxy read
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"      # and send timeout to 3600s
    nginx.ingress.kubernetes.io/upstream-hash-by: "$remote_addr"  # Enable sticky session
spec:
  rules:
    - http:
        paths:
          - path: /?(.*)
            backend:
              serviceName: client-cluster-ip-service
              servicePort: 3000
          - path: /api/?(.*)
            backend:
              serviceName: server-cluster-ip-service
              servicePort: 5000
```

Figure 53. Nginx Ingress service configuration file.

3.7 Deployment

3.7.1 Cloud Services

This section will illustrate the deployment process of this project in which many cloud services and tools are applied and combined all together to support in the progress. The production is deployed onto GKE with diverse tools supporting by GCP. Figure 4 below displays cloud services used in the deployment and the way they connect altogether.

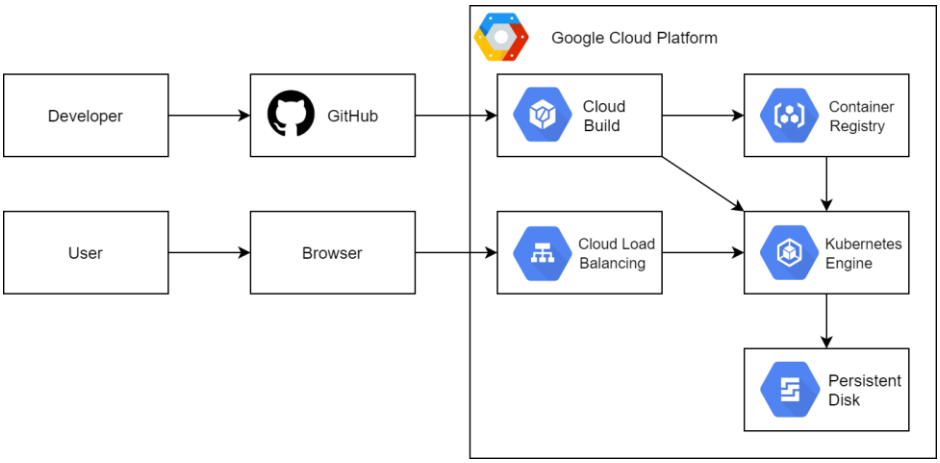

Figure 54. Integration of Google Cloud Platform services.

GCP is one of the top cloud service providers, vying with AWS and Microsoft Azure. Kubernetes architecture, which is developed by Google, is supported incredibly with GKE. Other supporting services play an important role as the whole powerful infrastructure for microservice architecture. For instance, Cloud Build is supplied for building a simple but efficient CI/CD pipeline, GCR is the private and secured Docker registry to let images stored on the GCP, Cloud Load Balancing distributes network traffic to all instances of the applications, limits risks from the overload of an instance while others are excess of resources or Persistent Disk offers the reliability, quick and flexibility in storage for virtual machine instances. [33]

3.7.2 CI/CD Pipeline

To automate the deployment, Google Cloud Build is applied to carry out a CI/CD pipeline in this project. Due to the prototype goal and scope of this thesis project, this pipeline is designed in simplified stages including building, testing, and deploying directly to production without going through a staging environment for quality assurance.

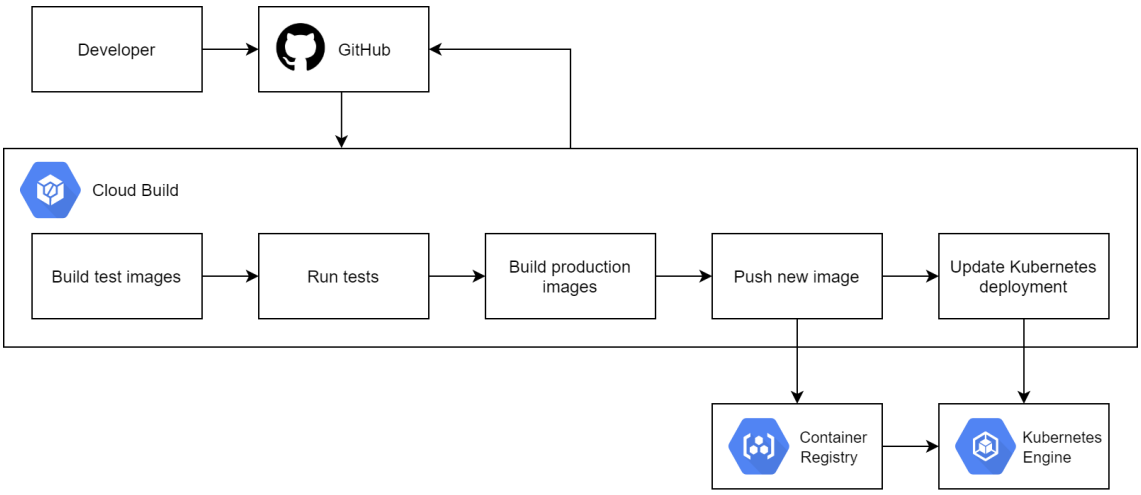

Figure 55. The CI/CD Pipeline.

The pipeline begins when developers commit and push their changes onto the GitHub repository. At that time, Cloud Build is automatically triggered and the build process will start. The initial steps are building test images and running tests. If all the images pass the tests, the optimized production images will be built and pushed to GCR. Next, Cloud Build updates the Kubernetes deployment. GKE receives a request for using new

images, then pulls the images from GCR. In case any stage from the process falls, Cloud Build will stop the process and feed the failed process back to GitHub. Otherwise, if it succeeds, the feedback will be informed to GitHub as a success.

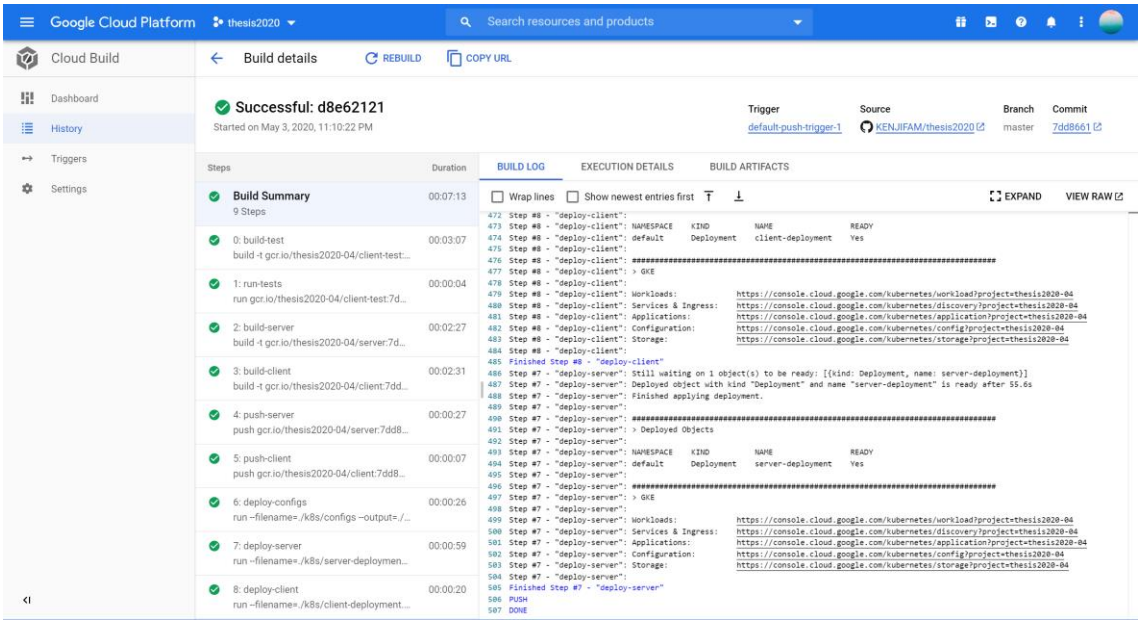

Figure 56. Result of the CI/CD pipeline on Cloud Build.

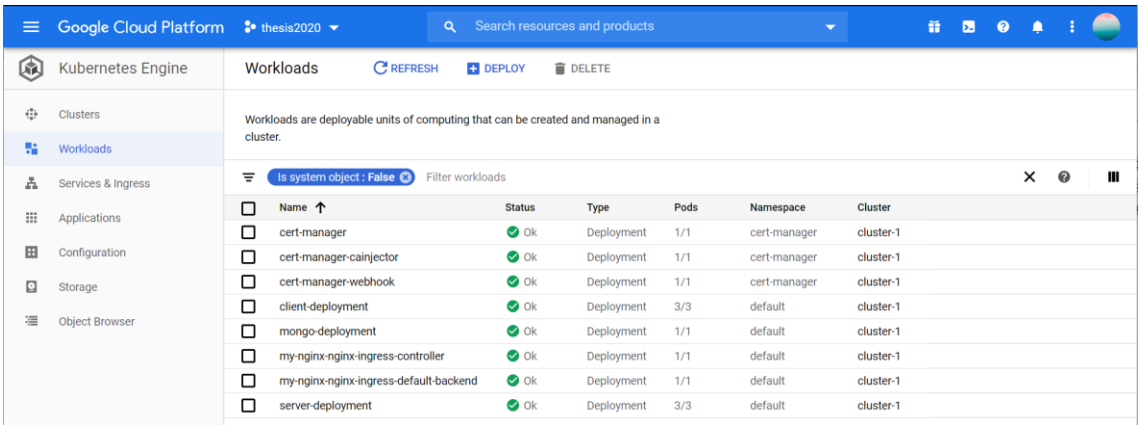

Figure 57. Containers run on a GKE cluster after being successfully deployed.

After Cloud Build runs the pipeline, the result and logs of the deployment process could be viewed via the GCP Console. And from the GKE dashboard, the cluster state and management as well as the resources could be observed. The dashboard is also useful for debugging, troubleshooting issues, monitoring, and supporting the DevOps workflow.

3.7.3 SSL Certificate

For providing privacy and security for data transmission, TLS protocol and SSL certificate are an obligatory requirement in the production. Cert-manager, a native Kubernetes certificate management controller, is used for issuing and auto-renewing SSL certificate before expiring. This project uses the SSL certificate from Let's Encrypt, a nonprofit Certificate Authority providing certificates. [34]

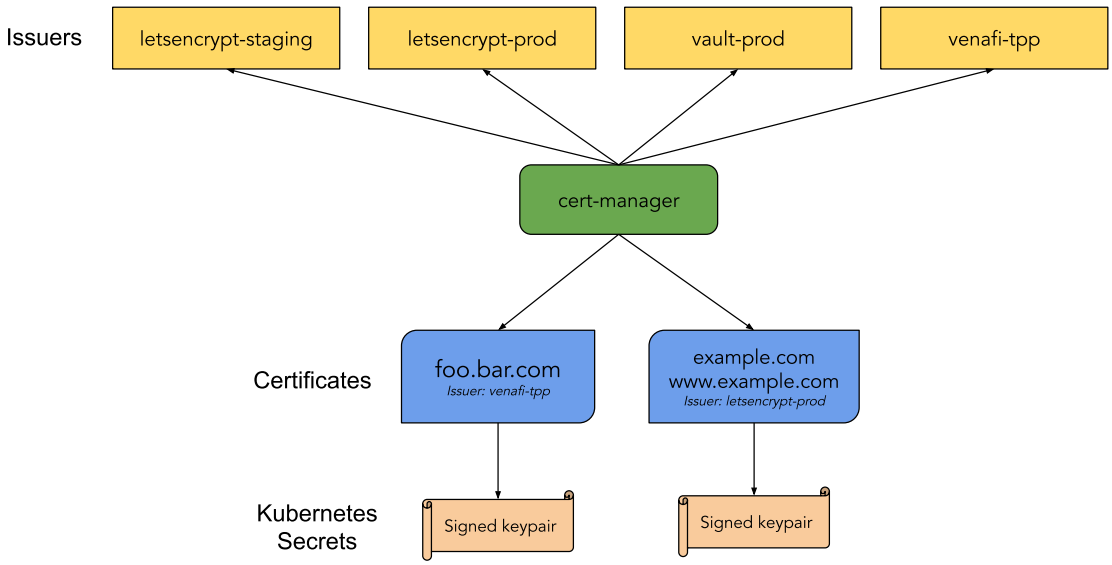

Figure 58. Overview of Cert-manager's operation [34].

4 Results and Discussions

In this section of the thesis, the reflection of the project will be discussed critically for the learning process as well as the feasibility of this thesis as the research on the technical side. Some challenges notwithstanding, the project aim has been reached with full of features following the technical requirements. The Food Supporter application now could be found as a published site thanks to the Google Cloud Platform.

Working through this project is fairly challenging with the trials on some new technologies combined with other prevailing ones. It took quite much time than other parts to work through Real-time Messaging because of the first time experiment on the WebSocket technology. Besides that, the configuration of the association between the WebSocket and the microservice with Docker and Kubernetes caused the difficulties in debugging and deploying on some servers, but it has been done finally and become a valuable experience with those.

Another obstacle while doing this project also came from the ambitious objective of this thesis to trial an all-in-one application as an assumption of developing as a large-scale application. Therefore, the thing that requires all technologies integrated altogether with keeping everything on track was a self-challenge for each stage including the new trial of the new Redux Toolkit released in February 2020, or the Kubernetes in managing containers and scaling the whole system. Yet, this project might not be a large-scale demo application without the aforementioned hindrances.

About further developments for this project, the Request feature should be supplemented with an updated status to ensure the possibilities of the requests such as “reserved”, “cancelled”, or “finished”. Moreover, to get better viability, the Message feature also needs to be improved. Particularly, the inbox connection should be linked directly to each request after a user accepts a request, hence, users can only contact each other via each request, not personally anymore. And, the last task which should be developed in the future is the database scalability of the microservice with the larger database and more servers.

Last but not least, for the scope of the Food Supporter, in reality, the practical of this project as the research on applying modern web technologies for a large-scale model could have much applicability to other applications in diverse fields. And, this project also has contributed an idea for food waste solution which might be developed more realistically in the future.

5 Conclusion

The objective of the thesis is to create an application that is ideated from a reduction in food waste problem using modern web technologies consisting of the TypeScript programming language, React and Redux for the frontend, Node.js, Express, and MongoDB for the backend, then Docker and Kubernetes in microservice. By focusing on technical development, this thesis is considered as the research on developing a web application for large-scale systems for full-stack developers.

The thesis as Food Supporter application has been done fully required features as listed above such as the Request, Message, and Registration features. Moreover, the result of this thesis is shown distinctly via the published site on Google Cloud Platform, and this document lists as a learning diary about how the Food Supporter works.

In summary, not so good as expected because of lacking time, the thesis still has completed researching how to build a larger model application with deepening multi-techniques altogether and figured out the harmony with advantages and disadvantages while applying many technologies in just a web application. And, as further as the web industry is developing, this thesis has been done as the basic research in those aforementioned issues, nonetheless, the opportunity and possibility for this thesis to improve deeper to break out of other new technologies, for example, could be more in the future.

References

1 UN report: one-third of world's food wasted annually, at great economic, environmental cost [online] UN News; 11 September 2013.
URL: https://news.un.org/en/story/2013/09/448652. Accessed 30 April 2020.

2 Food waste reduction by developing legistration [online] LUKE; September 2018.
URL: https://valtioneuvosto.fi/documents/1927382/2116852/9-2018-Food+waste+reduction+by+developing+legislation/713f019b-8b05-43c6-bfbd-5423706fadde?version=1.0. Accessed 30 April 2020.

3 Moiseev Antoon, Fain Yakov. TypeScript Quickly [e-book] Manning Publication; March 2020.
URL: https://learning.oreilly.com/library/view/typescript-quickly/9781617295942/. Accessed 30 March 2020.

4 Rozentals Nathan. Mastering TypeScript 3 – Third Edition [e-book] Packt Publishing; February 2019.
URL: https://learning.oreilly.com/library/view/mastering-typescript-3/9781789536706/. Accessed 30 March 2020.

5 Dubois Sebastien, Georges Alexis. Learning TypeScript 3 by Building Web Applications [e-book] Packt Publishing; November 2019.
URL: https://learning.oreilly.com/library/view/learn-typescript-3/9781789615869/. Accessed 30 March 2020.

6 Jansen H. Remo. Learning TypeScript [e-book] Packt Publishing; September 2015.
URL: https://learning.oreilly.com/library/view/learning-typescript/9781783985548/. Accessed 30 March 2020.

7 Freeman Adam. Essential TypeScript: From Beginner to Pro [e-book] Apress; August 2019.
URL: https://learning.oreilly.com/library/view/essential-typescript-from/9781484249796/. Accessed 30 March 2020.

8 Cherny Boris. Programming TypeScript [e-book] O'Reilly Media, Inc.; May 2019.
URL: https://learning.oreilly.com/library/view/programming-typescript/9781492037644/. Accessed 30 March 2020.

9 Neal David, Use TypeScript tp Build a Node API with Express [online] Okta; 15 November 2018.
URL: https://developer.okta.com/blog/2018/11/15/node-express-typescript. Accessed 30 March 2020.

10 Wilson Jim. Node.js 8 the Right Way [e-book] Pragmatic Bookshelf; January 2018.
URL: https://learning.oreilly.com/library/view/nodejs-8-the/9781680505344/. Accessed 30 March 2020.

11 Developer Survey Results 2019 [online] StackOverflow; 2019.
URL: https://insights.stackoverflow.com/survey/2019. Accessed 30 March 2020.

12 Porcello Eve, Banks Alex. Learning React, 2nd Edition [e-book] O'Reilly Media, Inc.; July 2020.
URL: https://learning.oreilly.com/library/view/learning-react-2nd/9781492051718/. Accessed 9 April 2020.

13 Carlos Santana Roldan. React Cookbook [e-book] Packt Publishing; August 2018.
URL: https://learning.oreilly.com/library/view/react-cookbook/9781783980727/. Accessed 9 April 2020.

14 Cássio de Sousa Antonio. Pro React [e-book] Apress; December 2015.
URL: https://learning.oreilly.com/library/view/pro-react/9781484212608/. Accessed 9 April 2020.

15 Chiarelli Andrea. Beginning React [e-book] Packt Publishing; July 2018.
URL: https://learning.oreilly.com/library/view/beginning-react/9781789530520/. Accessed 9 April 2020.

16 Bugl Daniel. Learn React Hooks [e-book] Packt Publishing; October 2019.
URL: https://learning.oreilly.com/library/view/learn-react-hooks/9781838641443/. Accessed 9 April 2020.

17 Tutorial [online] React.
URL: https://reactjs.org/tutorial/tutorial.html. Accessed 9 April 2020.

18 Garreau Marc and Faurot Will. Redux in Action [e-book] Manning Publications; June 2018.
URL: https://learning.oreilly.com/library/view/redux-in-action/9781617294976/. Accessed 14 April 2020.

19 Lee James, Wei Tao, Mukhiya Kumar Suresh. Redux Quick Start Guide [e-book] Packt Publishing; February 2019.
URL: https://learning.oreilly.com/library/view/redux-quick-start/9781789610086/. Accessed 14 April 2020.

20 Bugl Daniel. Learning Redux [e-book] Packt Publishing; August 2017.
URL: https://learning.oreilly.com/library/view/learning-redux/9781786462398/. Accessed 14 April 2020.

21 Herron David. Node.js Web Development – Fourth Edition [e-book] Packt Publishing; May 2018.
URL: https://learning.oreilly.com/library/view/nodejs-web-development/9781788626859/. Accessed 18 April 2020.

22 Casciaro Mario, Mammino Luciano. Node.js Design Patterns – Second Edition [e-book] Packt Publishing; July 2016.
URL: https://learning.oreilly.com/library/view/nodejs-design-patterns/9781785885587/. Accessed 18 April 2020.

23 Shipton Lizzie. The Best NodeJS Frameworks for 2020 [online] RapidAPI; 16 October 2019.
URL: https://rapidapi.com/blog/best-nodejs-frameworks/. Accessed 18 April 2020.

24 Brown Ethan. Web Development with Node and Express, 2nd Edition [e-book] O'Reilly Media, Inc.; November 2019.
URL: https://learning.oreilly.com/library/view/web-development-with/9781492053507/. Accessed 23 April 2020.

25 Santiago Antonio. Using async/await in ExpressJS middlewares [online] Acuriousanimal; 15 February 2018.
URL: https://www.acuriousanimal.com/blog/2018/03/15/express-async-middleware. Accessed 23 April 2020.

26 Hows David, Membrey Peter, Plugge Eelco. MongoDB Basisc [e-book] Apress; December 2014.
URL: https://learning.oreilly.com/library/view/mongodb-basics/9781484208953/. Accessed 25 April 2020.

27 Serdar Yegulalp. What is Docker? The spark for the container revolution [online] InfoWorld; 19 April 2019.
URL: https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html. Accessed 27 April 2020.

28 Neeru Jain. A Quick Introduction to Docker Fundamentals [online] WhizLabs; 1 August 2019.
URL: https://www.whizlabs.com/blog/docker-fundamentals/. Accessed 27 April 2020.

29 Janakiram MSV. Kubernetes: An Overview [online] TheNewStack; 7 November 2016.
URL: https://thenewstack.io/kubernetes-an-overview/. Accessed 29 April 2020.

30 Mui-org. Material-UI [online] GitHub.
URL: https://github.com/mui-org/material-ui. Accessed 30 April 2020.

31 Reduxjs/cra-template-redux. Initial Release [online] GitHub; 18 February 2020.
URL: https://github.com/reduxjs/cra-template-redux/releases/tag/v1.0.0. Accessed 30 April 2020.

32 What is Redux Toolkit? [online] Redux.
URL: https://redux.js.org/redux-toolkit/overview. Accessed 1 May 2020.

33 Get started with Google Cloud [online] Google Cloud.
URL: https://cloud.google.com/docs. Accessed 1 May 2020.

34 Welcome to cert-manager [online] Cert-Manager.
URL: https://cert-manager.io/docs/. Accessed 1 May 20