HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

# GRADUATION THESIS

## Apply Microservice Architecture in Building Ecommerce Competitor Monitoring System

**DAO NGOC THANH**
thanh.dn163693@sis.hust.edu.vn

**Field : Information Technology**
**Major : Computer Science**

**Thesis advisor** : Ph.D Vu Thi Huong Giang

Signature of advisor

**Major** : Information Technology
**Department** : Software Engineering Department

**Hanoi, 5-2021**

# GRADUATION THESIS ASSIGNMENT

1. Student's information :
   Name : Dao Ngoc Thanh.
   Phone : +84 968 037 260  Email: thanh.dn163693@sis.hust.edu.vn
   Class : ICT-K61
   Affiliation : Hanoi University of Science and Technology.
   Duration : 11/02/2021 - 31/05/2021.

2. Objectives of the thesis :

   - Study the microservice architecture and the requirement of ecommerce competitor tracking and analytic system.
   - Adopt technologies needed and utilities microservice architecture to build an web application that implements the requirements.
   - Build the CI/CD workflow for all the services in projects.

3. Disclosure of interests/Commitments :
   I – Dao Ngoc Thanh – hereby warrants that the Work and Presentation in this thesis are performed by myself under the supervision of Ph.D Vu Thi Huong Giang. All results presented in this thesis are truthful and are not copied from any other work. All references in this thesis - including images, tables, figures, and quotes - are clearly and fully documented in the bibliography. I will take full responsibility for even one copy that violates school regulations.

   *Hanoi, date month year 2021*
   Author

   *Dao Ngoc Thanh*

4. Comment of thesis advisor:
   . . . . . .
   . . . . . .
   . . . . . .
   . . . . . .

   *Hanoi, date month year 2021*
   Thesis Advisor

   *Ph.D Vu Thi Huong Giang*

# Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Ph.D Vu thi Huong Giang for the continuous support of my graduation thesis, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my engineer study.

I would like to thank all professors of Hanoi University of Science and Technology who have taught me since the day I entered the university five years ago. They are always willing to support students, and that really impresses me. Their expert knowledge inspired me a lot to study more and follow my passion.

For my friends and lecturers during my 5-year span in Hanoi University of Science and Technology, I am humble and grateful for the knowledge I learn from each and all of you, the connections and the memories I had and will cherish for the lifetime. This University has given me more than I ever asked for and it shapes the person I am today.

Finally, I would like to thank my parents, who have always stood by me, taken care for me, and given me a lot of motivation during my student life so I have enough strength and confidence to follow my passion.

# *Apply Microservice Architecture in Building Ecommerce Competitor Monitoring System*


## Abstract

Now a day, more and more products sold in ecommerce platform such as Shopee, Tiki, Lazada ..., the competition for an ecommerce store increases. More and more online shops pop up selling similar or the same products. It make commerce becomes a highly competitive environment and to be success in such environment, having a nice looking shop is no longer enough, seller need to consistently monitoring what their competitors are doing in order to stay ahead of the competitor.

Monitoring the competition can take up a lot of sellers time, if they do it manually. So that, they need a tool which can help they automatic this kind of tasks. Now, there are a few tools like that exists in the market and when I research about such tools, I see these tools are quite hard to use, because they have too many features which are not focus only on competitor monitoring. Moreover, the service price of such tools are also quite high.

For above reasons, I chose to build a web application that provide for sellers in ecommerce platform (Shopee) a tools set that focus on help sellers monitoring their competitors in a fast and convenient way. My application delivered the most essential use cases for competition monitoring such as competitor tracking and sentiment analytic. I follow microservice architecture to implement this application in order to utilize the benefits of the architecture, such as the flexibility to use the right tool for the right task, increased resilience, continuous delivery and so on.

# Contents


1 Introduction 1

1.1 Motivation . . . . . 1

1.2 Objectives and scope . . . . . 2

1.3 Approaches . . . . . 3

1.4 Structure of thesis . . . . . 4

2 Requirements analysis 5

2.1 Similar Applications . . . . . 5

2.2 Feature overview . . . . . 6

2.2.1 General usecase diagram . . . . . 6

2.2.2 Usecase diagram for "Product monitoring" . . . . . 7

2.2.3 Usecase diagram for "Shop monitoring" . . . . . 8

2.2.4 Usecase diagram for "Users management" . . . . . 9

2.2.5 Usecase diagram for "Data management" . . . . . 10

2.3 Usecases specifications . . . . . 11

2.3.1 Use case specification for "Add product by url" . . . . . 12

2.3.2 Use case specification for "Add product by file" . . . . . 13

2.3.3 Use case specification for "View product tracking data" . . . . . 14

2.3.4 Use case specification for "View product sentiment analytic data" . . . 15

2.3.5 Use case specification for "Update product data" . . . . . 16

2.3.6 Use case specification for "Remove product(s)" . . . . . 16

3 Adopted Technology and Framework 17

3.1 Front-end Technologies . . . . . 17

3.1.1 React and NextJS . . . . . 17

3.1.2 Ant Design . . . . . 18

3.1.3 Redux . . . . . 18

3.1.4 Tailwind css . . . . . 19

3.2 Back-end Technologies . . . . . 20

3.2.1 Golang and Gin . . . . . 20

3.2.2 Python, Flask and Scikit-learn . . . . . 20

3.2.3 Mongo . . . . . 22

3.2.4 REST API . . . 23
3.3 Deployment Technologies . . . 23
3.3.1 Docker . . . 23
3.3.2 Docker Compose . . . 24
3.3.3 Github actions . . . 25

4 Application Development and Deployment 26
4.1 Architecture Design . . . 26
4.2 Detail Design . . . 28
4.2.1 User interface . . . 28
4.2.2 Use case design . . . 29
4.2.3 Component design . . . 31
4.2.4 Database design . . . 33
4.3 Application Construction . . . 36
4.3.1 Library and Tools . . . 36
4.3.2 Achievements . . . 37
4.3.3 Main screens and features . . . 37
4.4 Test . . . 40
4.4.1 Test for "Add product to track list" usecase . . . 40
4.4.2 Test for "Update product data" usecase . . . 41
4.4.3 Test for "Delete product from track list" usecase . . . 41
4.5 Deployment . . . 42

5 Main Contributions 43
5.1 Develop a competitor monitoring tool for shopee seller . . . 43
5.1.1 Problem . . . 43
5.1.2 Solution . . . 44
5.1.3 Result . . . 45
5.2 Microservice architecture . . . 52
5.2.1 Problem . . . 52
5.2.2 Solution . . . 53
5.3 CI/CD Pipelines for Microservices . . . 54
5.3.1 Problem . . . 54
5.3.2 Solution . . . 55

6 Conclusions and Future Works 58

References 59

Appendices 60

A Test cases 60

# List of Figures


2.1 General usecase diagram . . . . . 6
2.2 Usecase diagram for "Product monitoring" . . . . . 7
2.3 Usecase diagram for "Shop monitoring" . . . . . 8
2.4 Usecase diagram for "User management" . . . . . 9
2.5 Usecase diagram for "Data Management" . . . . . 10

3.1 mongo document example . . . . . 22
3.2 Docker architecture . . . . . 24

4.1 General Architecture . . . . . 27
4.2 Frontend architecture . . . . . 28
4.3 Sequence Diagram for usecase "Add Product to track list by url" . . . . . 29
4.4 Sequence Diagram for usecase "Update product data" . . . . . 30
4.5 Frontend service component design . . . . . 31
4.6 core service component design . . . . . 32
4.7 Entity Relation Diagram . . . . . 33
4.8 Data management screen with small screen size . . . . . 38
4.9 Data management screen with medium screen size . . . . . 38
4.10 User management (user list) screen . . . . . 39
4.11 Data management (user edit) screen . . . . . 39

5.1 Activities diagram for "Add Product to monitoring track list" . . . . . 45
5.2 UI for adding product to track list . . . . . 46
5.3 UI for adding shop to track list . . . . . 46
5.4 UI for product feature navigation . . . . . 47
5.5 UI for shop feature navigation . . . . . 47
5.6 UI for product price tracking data display . . . . . 47
5.7 UI for product price title data display . . . . . 48
5.8 UI for product ratings tracking data display . . . . . 48
5.9 UI for related products data display . . . . . 49
5.10 UI for product review analytic data display . . . . . 49
5.11 UI for product review tags analytic data display . . . . . 50

5.12 UI for shop actives vouchers tracking data display . . . . . 50
5.13 UI for shop followers tracking data display . . . . . 51
5.14 UI for shop product quantity tracking data display . . . . . 51
5.15 UI for shop ratings tracking data display . . . . . 51
5.16 Monolithic architecture vs Microservice architecture . . . . . 52
5.17 General architecture . . . . . 53
5.18 CI/CD flow . . . . . 54
5.19 Core Service Dockerfile . . . . . 55
5.20 Core Service core.compose.yml file . . . . . 56
5.21 Core Service cd.yml file . . . . . 57

# List of Tables


2.1 List of use cases . . . . . 11
2.2 Use case "Add product to track list by url" specification . . . . . 12
2.3 Use case "Add product to track list by file" specification . . . . . 13
2.4 Use case "View product tracking data" specification . . . . . 14
2.5 Use case "View product sentiment analytic data" specification . . . . . 15
2.6 Use case "Update product data" specification . . . . . 16
2.7 Use case "Remove product(s)" specification . . . . . 16

4.1 User interface specification . . . . . 28
4.2 Detail database design . . . . . 33
4.3 Libraries and tools in project . . . . . 36
4.4 Application specification . . . . . 37
4.5 Test cases of "Add product to track list" usecase . . . . . 40
4.6 Test cases of "Update product data" usecase . . . . . 41
4.7 Test cases of "Delete product from track list" usecase . . . . . 41
4.8 Domain to application . . . . . 42
4.9 Host machine specification . . . . . 42

*The only way to go fast, is to go well.*

Robert C. Martin, Clean Architecture

# Chapter 1

# Introduction

In the first chapter, I would like to introduce the overview topic of this thesis, the reasons that led to my decision to choose the problems to be solved in my graduation research. Moreover, this part contains the purpose and the solution of the topic, as well as the introduction of the report outline.

## 1.1 Motivation

Marketplace ecommerce requires sellers to always be vigilant with their next move. There is little room for differentiation on a website like Shopee or Tiki.There are several reasons why successful sellers and brands monitor their competitors. I will touch on the most popular reasons for brands and sellers to use automation when tracking competitors below.

The first one is to **react to price changes**. Monitoring the price of your competitors is critical. It enables you to tailor your product prices appropriately and remain competitive. Price is one of the most vital factors when shopping online and if you are in a competitive niche with many sellers selling the same product – price is the single biggest determinant of conversion rate.

The second one is to **react to vouchers and promotions by competitors**. Voucher surprises from your competitors will have an impact on your business. Anytime they release a new voucher, it is likely to directly affect the cost of the product to consumers. With that being quite clear, having immediate knowledge of when a competitor has created new voucher activities like new promotions and vouchers will help prepare brands and sellers for a counter-strategy.

Next, monitoring competitor help sellers to **understand their investments into product listing optimization**. As you are gradually assessing the activities of your competitors, you become more mindful and aware of their product listing strategies. You can track their changes

to see what they're testing and imitate the product listing changes that you see to be valuable. Not only will you be able to avoid your competitors' mistakes but you can also adapt and enhance some of your competitors' optimization strategies which could boost your gains in your marketing investments.

Finally, it also helps sellers to **understand the growth and strategies used to increase followers**. By looking into your competitors' data, you can leverage their own strategies and use it to increase your followers and enable your brand to grow. One way to boost your strategy is to take pieces of the tactics that your competitors are doing and combine it with your own work of action. For instance, Shopee has its own feed where sellers can post content just like what you'd normally do in social media. As you monitor your competitors, you can check out which one of their posts gets the most engagement.

In short, when it comes to competitors, sellers can gain a significant advantage if they can monitor what their competitors are doing it and use that to help guide their own pricing and marketing strategy. Furthermore, monitor competitor's strategy also help sellers to identify the strengths and weaknesses of different approaches within specific categories. When seller to this,they will have access to data that allows they to evaluate the pros and cons of specific marketing and pricing tactics to ensure they're using the best approaches for their own ecommerce store.

For these reasons, in this thesis, I have developed an application that provide a set of productivity tools for seller who using e-commerce platform. It will help seller to obtain and analyze data about their field, business environment, competitors. This is a gradual and thorough process that helps businesses strategic in order to fill in competitive gaps and climb the ladder in their niche or category.

## 1.2 Objectives and scope

For the motivation mentioned in the above section, the objectives of conducting this thesis is to developing an web application for users with following modules:

- **User Management** provides features related to users which includes (i) login, (ii) register new account, (iii) CRUD user and (iv) user authorization.
- **Product** provides features related to product which includes (i) add/remove new product to track list, (ii) analyze sentiment of buyer to product, (iii) track product information (prices, title, ratings, related products), (iv) trigger update product data command.
- **Shop** provides features related to shop which includes (i) add/remove new shop to track

list, (ii) analyze sentiment of buyer to shop, (iii) track shop information (active vouchers, product quantity, followers, ratings), (iv) trigger update shop data command.

- **Data Management** provides features for super user to observe the general statistic about the whole application.

## 1.3 Approaches

Nowadays there are two main approaches when developing applications: web applications and mobile applications. Web apps have some undeniable advantages. They can run on every device which has a browser supporting HTML/CSS/JavaScript. It is not necessary for user to bother updating for web applications are able to update themselves without any user intervention. It do not require app store approval, so can be launched quickly. Moreover, the web applications are easier to maintain, because they have a common codebase regardless of mobile platform. For those reasons, I decided to choose web based approach. With this approach, we just need to focus on building a single platform, the web. Thus, the operation, repair and update are also easier and less expensive.

Next, after choosing a web application approach, the problem is how to build an interface compatible with many different types of screens and that interface needs to be friendly and improve user experience. Obviously, the web application has overcome most of the problems that a mobile application has, but the problem that web applications still need to solve is UI / UX. However, with the web application, it is not possible to limit the type of user display, as well as not knowing what kind of screen the user will use and therefore, the UI / UX design and construction will also be difficult. In order to deal with this disadvantage, I choose to design and build in a Web Responsive style. With a responsive design, we just need to add CSS snippets for each type of display. As a result, we can design and build an interface that is compatible, friendly with all kinds of screens and thereby also enhances UI / UX.

Nowadays, there are so many options about frameworks and libraries to choose when it comes to building visually appealing apps that rank high on user experience. I decided to use React because of several reasons. First of all, React is remarkably flexible. Once you have learned it, you can use it on a vast variety of platforms to build quality user interfaces. React is a library, NOT a framework. Its library approach has allowed React to evolve into such a remarkable tool. Next, React Has Facebook's Support/Resources. Finally, React Has Broader Community Support. Since 2015, React popularity has grown steadily.

The system as mentioned above required a number of modules need to be built, such as module of user management, product/shop, notification or user interface. I choose to use microservice architecture because of its benefits. Firstly, all the services can be deployed and updated independently, which gives more flexibility. A bug in one microservice has an impact

only on a particular service and does not influence the entire application. Second advantage of the microservices approach is that each element can be scaled independently. So the entire process is more cost- and time-effective than with monoliths when the whole application has to be scaled even if there is no need in it. Moreover, with microservice architecture, The engineering teams are not limited by the technology chosen from the start. They are free to apply various technologies and frameworks for each microservice.

## 1.4 Structure of thesis

The structure of the rest of the thesis is as follows.

Chapter 2 demonstrates requirement analysis of the project. This chapter contains three parts: (i) similar applications, (ii) functionality overview, (iii) the illustration of all use cases of the system by using use case diagrams.

Next, in Chapter 3, I will list and explain the technologies and frameworks adopted in the project. Specifically, I divide them into two groups: (i) front-end, (ii) back-end. In each part, I will give an overview about the technology or framework, then explain why I apply them in my application.

Chapter 4 will explain everything about the design of the system. In this chapter, I would like to give the overview of the architecture design and the UI design of the tool. In addition, database design would be illustrated.

Chapter 5 presents my main contributions to the application. Difficult problems arising during developing process and how I have solved them will be specified.

Finally, Chapter 6 is a conclusion of the thesis, which would be about the result after this project. Moreover, I would like to tell about the orientation of the future works for this system.

*Stay Hungry. Stay Foolish*

Steve Jobs

# Chapter 2

# Requirements analysis

The first chapter has presented why I conducted this project, the scopes and approaches that I took to fulfill the requirements and the structure of the thesis. In this chapter, I will discuss some similar applications in the market and their pros and cons, then give an overview of the application's functionalities. Some major use cases will also be specified.

## 2.1 Similar Applications

Today, There are several applications that can helper sellers monitor their competitors in ecommerce platform. The three most popular applications includes (i) splitdragon, (ii) powersell and (iii) sitegiant.

Splitdragon is a software tools for Shopee and Lazada sellers. Sellers can track their competitors' every move with Split Dragon competitor store and product tracking, set alerts to get notified anytime their competitor changes prices or introduces a new voucher so you can react sharply and stay competitive. However, Splitdragon does not have customer sentiment analytic about competitor feature and it's service price is quite high. The free plan of Splitdragon is only within 7 days with limited features.

Powersell is a multi-channel growth platform which includes many feature such as Platform integration, Product listing, Inventory management, Order management and on on. Competitor Monitoring is a current extension of Powersell. At the moment, Powersell is just support monitor competitor product with tracking prices, items sold and ratings.

SiteGiant is Malaysia's leading ecommerce enabler and was established back in 2013 to provide a range of ecommerce services, catered to both small and large enterprises in Malaysia. SiteGiant's ecommerce solution allows merchants to sell on multiple channels such as online marketplaces, Facebook, Instagram, WhatsApp and their own branded online store. Similar to

Powersell, monitoring competitor is just an add-on of SiteGiant so that, until now it just support track product prices.

## 2.2 Feature overview

### 2.2.1 General usecase diagram

The two main actors of the system are users and admins. User are sellers in ecommerce platform who want to monitor their competitor. They need to register an account and login in to use the functionalities of application. They are only allowed to monitoring the products or store that they added. Admin is the role with the highest level of access to application. Admins add/view all products or stores to be monitored and access all products and stores data in the system. They can add and delete other admin users, and can view the statistic of the whole application.

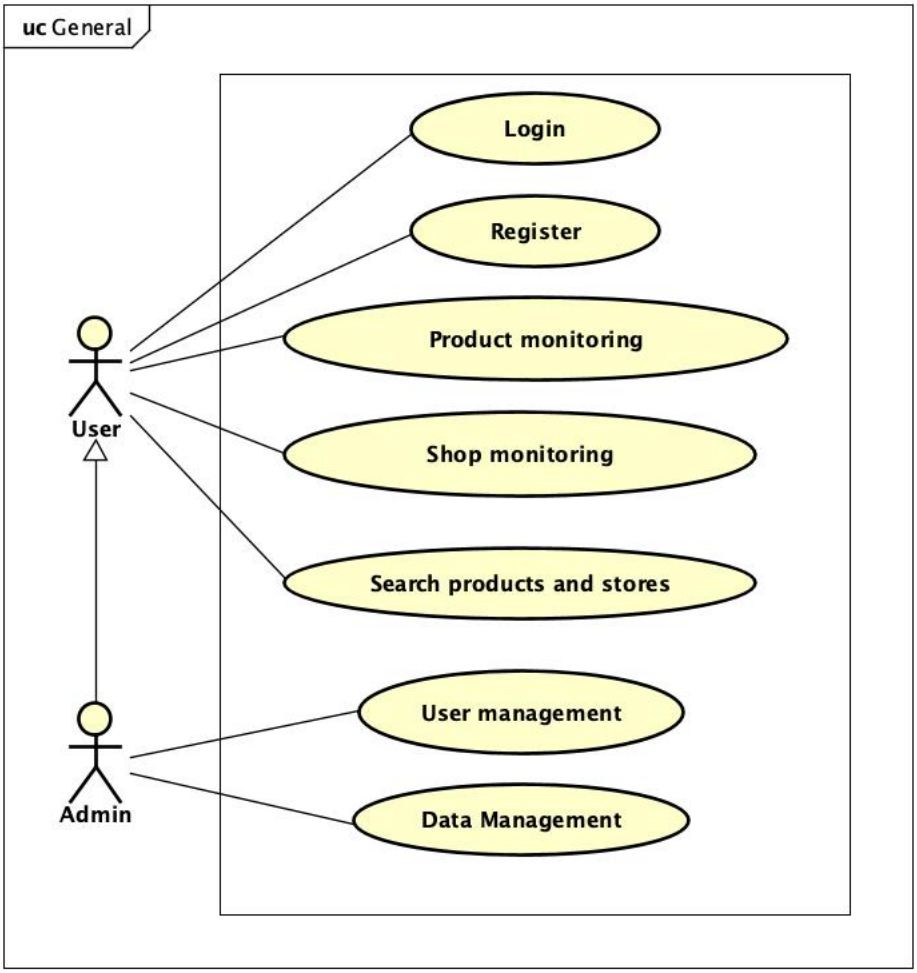


Figure 2.1: General usecase diagram

### 2.2.2 Usecase diagram for "Product monitoring"

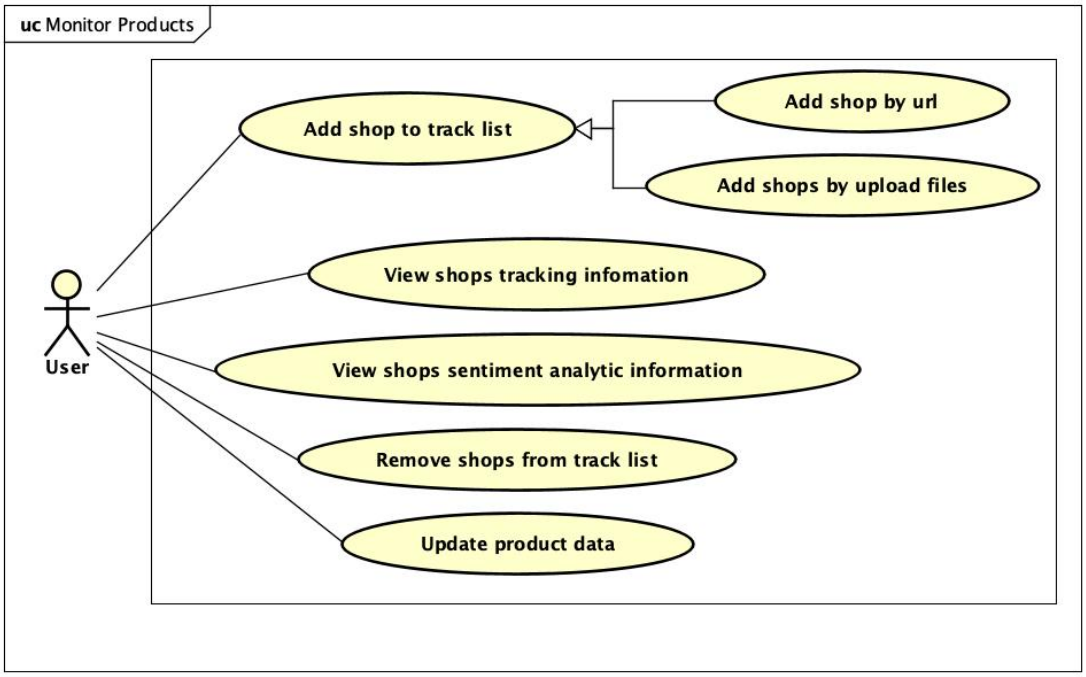


Figure 2.2: Usecase diagram for "Product monitoring"

The usecase "Product monitoring" is decomposed into smaller usecases in Figure 2.2. User can (i) add product to track list in two ways: by url and by upload csv file. In addition, they can (ii) view products tracking data, (iii) view products sentiment analytic data, (iv) remove product(s) from tracking list and user also can (v) trigger upload product data command.

### 2.2.3 Usecase diagram for "Shop monitoring"

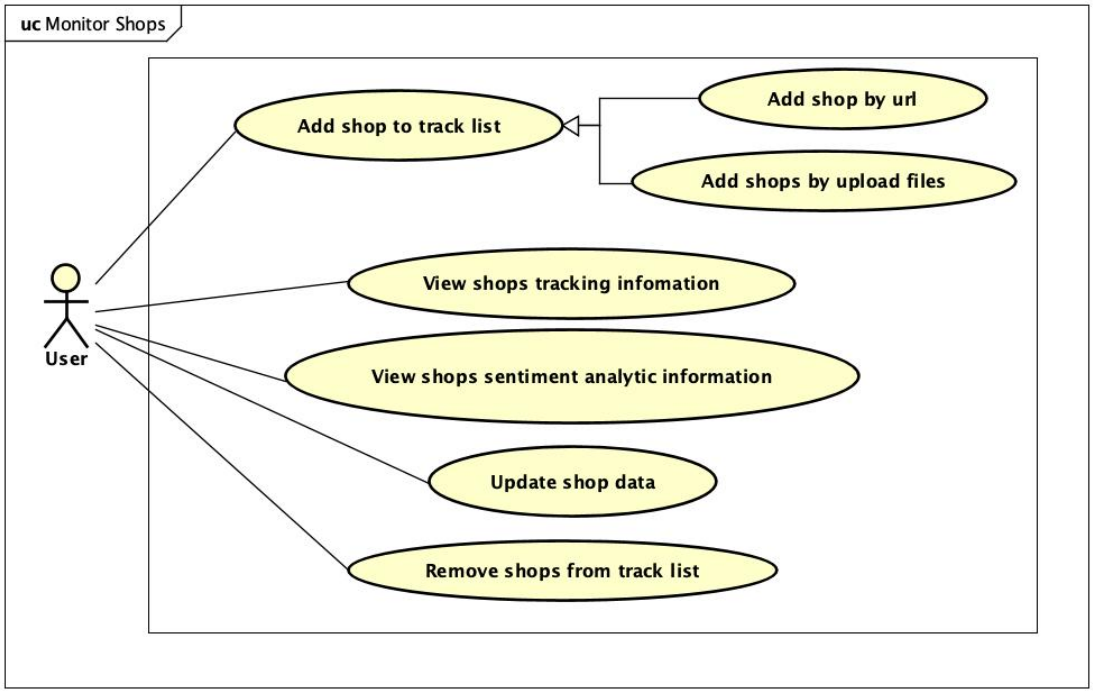


Figure 2.3: Usecase diagram for "Shop monitoring"

The usecase "Shop monitoring" is decomposed into smaller usecases in Figure 2.2. User can (i) add shop to track list in two ways: by url and by upload csv file. In addition, they can (ii) view shops tracking data, (iii) view shops sentiment analytic data, (iv) remove shop(s) from tracking list and user also can (v) trigger upload shop data command.

### 2.2.4 Usecase diagram for "Users management"

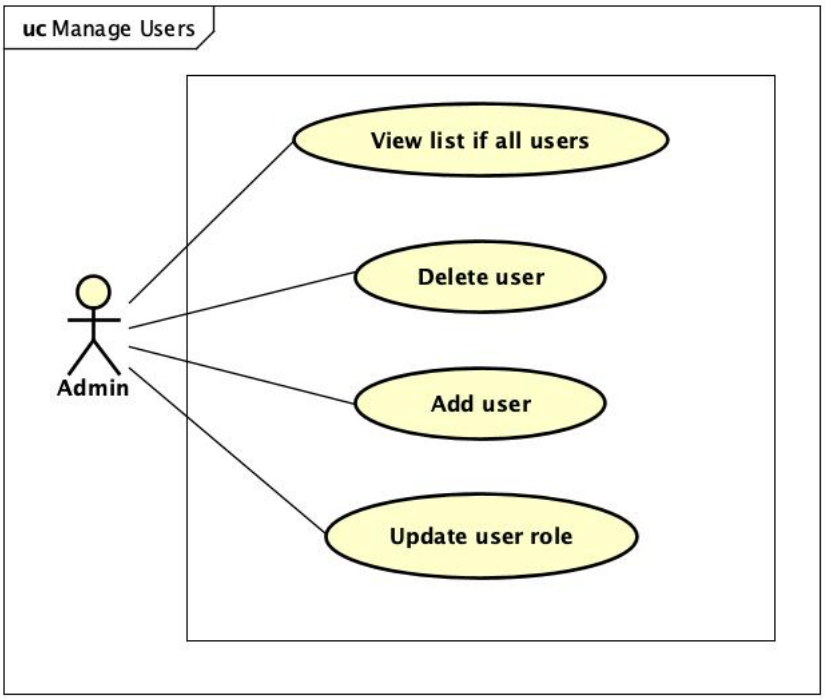


Figure 2.4: Usecase diagram for "User management"

The above usecase described what admin user can do to manage other users in the system. First, admin can view list of all users. They can select some users and delete those. Admin also can manually add new user and change role of a specific users.

### 2.2.5 Usecase diagram for "Data management"

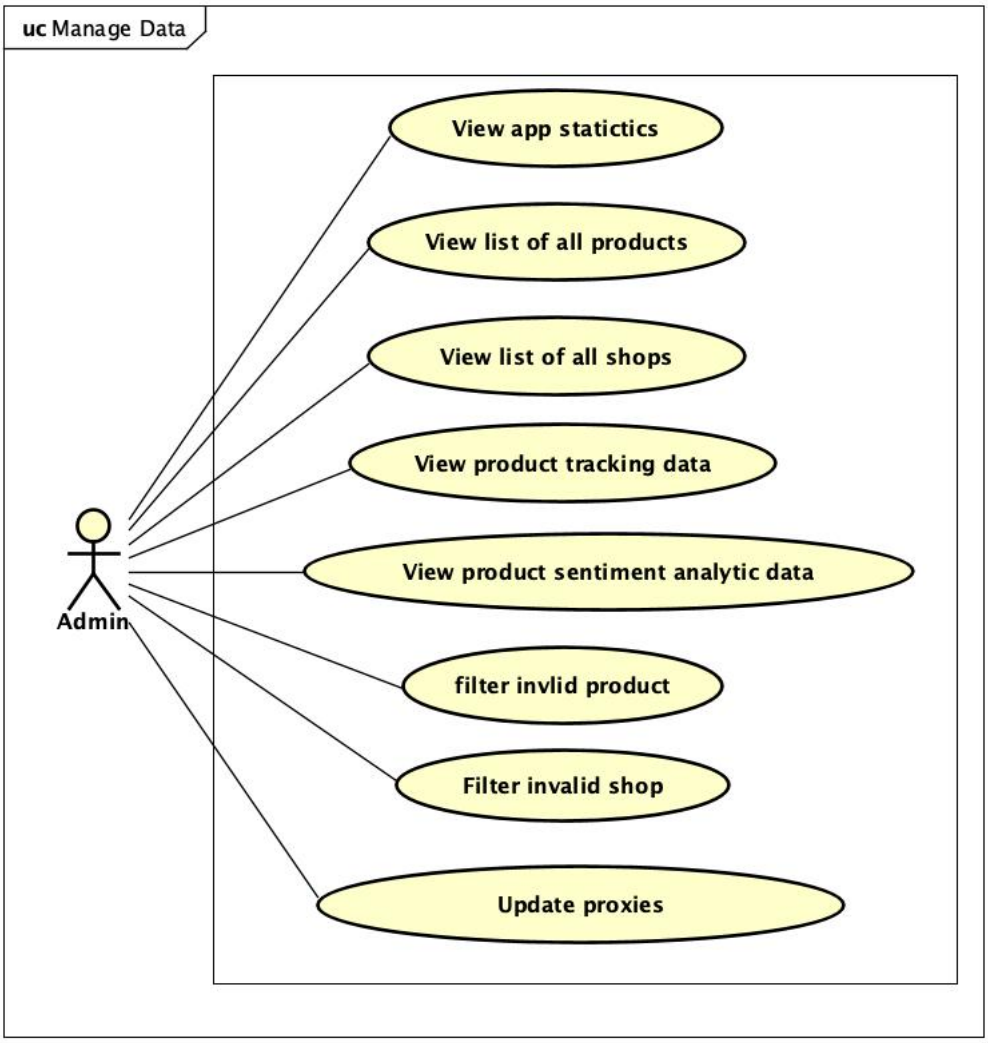


Figure 2.5: Usecase diagram for "Data Management"

The use case "Data management" is described in Figure 2.5. Only admin users are allowed to use this functions.

First, Admin can observe the application statistic like number of products, number of shops, number of proxies and so on. They can view list of all products in the system, the detail about tracking data and sentiment analytic data for each product. Admin also can view list of all shops in the system, the detail about tracking data and sentiment analytic data for each shops. Moreover, They can filter out the invalid products/stores and then trigger the update data command on these products/stores to refresh these data.

## 2.3 Usecases specifications

The following table is the list of all use cases of the system.

| General use case | Use case code | Use case name |
|---|---|---|
| | UC001 | Login by email and password |
| | UC002 | Register new account |
| | UC003 | search products and store |
| Product monitoring | UC004 | Add product to track list by url |
| | UC005 | Add product to track list by upload file |
| | UC006 | View product tracking data |
| | UC007 | View product sentiment analytic data |
| | UC008 | Remove product from track list |
| | UC009 | Update product data |
| Shop monitoring | UC010 | Add shop to track list by url |
| | UC011 | Add shop to track list by upload file |
| | UC012 | View shop tracking data |
| | UC013 | View shop sentiment analytic data |
| | UC014 | Remove shop from track list |
| | UC015 | Update shop data |
| User management | UC016 | View all users |
| | UC017 | Delete user(s) |
| | UC018 | Add new user |
| | UC019 | Update user role |
| Data management | UC020 | View app statistic |
| | UC021 | View list of all products |
| | UC022 | View list of all shops |
| | UC023 | View product tracking data |
| | UC024 | View shop tracking data |
| | UC025 | View product sentiment analytic data |
| | UC026 | View shop sentiment analytic data |
| | UC027 | Filter invalid products |
| | UC028 | Filter invalid shops |
| | UC029 | Update proxies |

Table 2.1: List of use cases

In the following part, there is specification of use cases of the system. Because of the limitation of the thesis, I would like to specify some main use cases only.

### 2.3.1 Use case specification for "Add product by url"

| **Use case code** | UC004 | **Use case name** | Add product by url |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User copy a product url on shopee.<br>2. User open "Competitor Product" screen.<br>3. User put the product url to the track new competitor product input.<br>4. User click Start button.<br>5. System call API to add the product to user track list and get the new product track list.<br>6. System display the new product track list and notify a success message. | | |
| **Alternative flow** | 6a. System display a "product already tracked" error message if the product is already in the track list.<br>6b. System display a "Invalid url" error message if the url invalid. | | |
| **Post-condition** | None | | |

Table 2.2: Use case "Add product to track list by url" specification

### 2.3.2 Use case specification for "Add product by file"

| **Use case code** | UC004 | **Use case name** | Add product by file |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User prepare a csv file which includes product url of shopee that user want to add.<br>2. User open "Competitor Product" screen.<br>3. User click "Upload csv" button .<br>4. User select prepared file.<br>5. User click "Start" button.<br>6. System call API to add the products to user track list and get the new product track list.<br>7. System display the new product track list and notify a success message. | | |
| **Alternative flow** | 7a. System display a "product already tracked" error message if there are any product already in the track list.<br>7b. System display a "Invalid url" error message if there are any url invalid. | | |
| **Post-condition** | None | | |

Table 2.3: Use case "Add product to track list by file" specification

### 2.3.3 Use case specification for "View product tracking data"

| **Use case code** | UC004 | **Use case name** | View product tracking data |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User open "Competitor Products" screen.<br>2. System call the API to get the product track list and display on screen.<br>3. User hover over the button on action column of the target product and click to "Product Tracking" option.<br>4. System call API to get the general information and the price tracking data of product and display it.<br>5. User click "TITLES" tab to see the title tracking data.<br>6. System call API to get the title tracking data of product and display it.<br>7. User click "RATINGS" tab to see the ratings tracking data.<br>8. System call API to get ratings tracking data of product and display it.<br>9. User click "RELATED" tab to see the related products tracking data.<br>10. System call API to get related products tracking data of product and display it.<br>11. User click "Visit in Shopee" to jump to shopee origin page of target product. | | |
| **Alternative flow** | 5a. User click to "Go back" button.<br>6a. System redirect user to product track list screen. | | |
| **Post-condition** | None | | |

Table 2.4: Use case "View product tracking data" specification

### 2.3.4 Use case specification for "View product sentiment analytic data"

| **Use case code** | UC004 | **Use case name** | View product sentiment analytic data |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User open "Competitor Products" screen.<br>2. System call the API to get the product track list and display on screen.<br>3. User hover over the button on action column of the target product and click to "Product Analytic" option.<br>4. System call API to get general information data of product and display it.<br>5. System call API to get sentiment analytic data of product ratings and display it.<br>6. System call API to get tags statistic data of product ratings and display it.<br>7. System call API to get ratings data of product and display it. | | |
| **Alternative flow** | 5a. User click to "Go back" button.<br>6a. System redirect user to product track list screen. | | |
| **Post-condition** | None | | |

Table 2.5: Use case "View product sentiment analytic data" specification

### 2.3.5 Use case specification for "Update product data"

| **Use case code** | UC004 | **Use case name** | Update product data |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User open "Competitor Products" screen.<br>2. System call the API to get the product track list and display on screen.<br>3. User hover over the button on action column of the target product and click to "Update" option.<br>4. System call API to update product data.<br>5. System call API to get the new product data and display it.<br>6. System display a success notification. | | |
| **Alternative flow** | 4a. System display a error notification if the API called fail. | | |
| **Post-condition** | None | | |

Table 2.6: Use case "Update product data" specification

### 2.3.6 Use case specification for "Remove product(s)"

| **Use case code** | UC004 | **Use case name** | Remove product(s) |
|---|---|---|---|
| **Actor** | User | | |
| **Pre-condition** | No | | |
| **Basic flow of events** | 1. User open "Competitor Products" screen.<br>2. System call the API to get the product track list and display on screen.<br>3. User select products to remove by click to checkbox in the first column on each row of table.<br>4. User click "DELETE" button.<br>5. User click "yes" button to confirm delete command.<br>6. System call API to delete products from track list and then display a success notification. | | |
| **Alternative flow** | 5a. User click "no" button to cancel delete command. | | |
| **Post-condition** | None | | |

Table 2.7: Use case "Remove product(s)" specification

*I'm not a great programmer; I'm just a good programmer with great habits.*

Martin Fowler, Refactoring: Improving the Design of Existing Code

# Chapter 3

# Adopted Technology and Framework

In the last chapter, I have mentioned the requirements for the application and specified them in diagrams. I will explain the technologies and frameworks adopted in the process of developing this application in this chapter. Within this thesis, they can be classified into three groups: front-end, back-end and deployment technologies.

## 3.1 Front-end Technologies

### 3.1.1 React and NextJS

React is a JavaScript library that aims to simplify development of visual interfaces. Developed at Facebook and released to the world in 2013, it drives some of the most widely used code in the world, powering Facebook and Instagram among many, many other software companies. Its primary goal is to make it easy to reason about an interface and its state in any point in time, by dividing the UI into a collection of components. React is used to build single-page web applications, among with many other libraries and frameworks that were available before React came into life. I choose react because of its three advance features includes Declarative, component-based and learn once, write anywhere.

Declarative, React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes. Declarative views make your code more predictable and easier to debug.

Component-Based, React build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere, React don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code. React can also render on the server using Node and power mobile apps using React Native.

Nextjs is a JavaScript framework created by Vercel [2]. It lets you build server-side rendering and static web applications using React. It's a great tool to build your next website. It has many great features and advantages, which can make Nextjs your first option for building your next web application. With Nextjs, I don't need any configuration of webpack [4] or similar to start using Nextjs. It comes with its configuration. All you need is to run npm run dev and start building your application.

### 3.1.2 Ant Design

Ant Design is a design system for enterprise-level products. It create an efficient and enjoyable work experience. Following the Ant Design specification, developer team at Ant Financial developed a React UI library antd that contains a set of high quality components and demos for building rich, interactive user interfaces, antd includes many powerful features:

- Enterprise-class UI designed for web applications.
- A set of high-quality React components out of the box.
- Written in TypeScript with predictable static types.
- Whole package of design resources and development tools.
- Internationalization support for dozens of languages.
- Powerful theme customization in every detail.

Over the past years, Ant Design has gained a lot of attention from the open source community with over 35k stars on GitHub. Although coming from China but thanks to the community, it's well documented in English.

### 3.1.3 Redux

Redux is a predictable state container designed to help write JavaScript apps that behave consistently across client, server, and native environments and are easy to test. While it's mostly used as a state management tool with React, developer can use it with any other JavaScript framework or library. It's lightweight at 2KB (including dependencies), so developer don't have to worry about it making application's asset size bigger. With Redux, the state of application is

kept in a store, and each component can access any state that it needs from this store. Redux have powerful features include:

- **Predictable**, Redux helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.
- **Centralized**, Centralizing application's state and logic enables powerful capabilities like undo/redo, state persistence, and much more.
- **Debuggable**, The Redux DevTools make it easy to trace when, where, why, and how application's state changed. Redux's architecture lets log changes, use "time-travel debugging", and even send complete error reports to a server.
- **Flexible**, Redux works with any UI layer, and has a large ecosystem of addons to fit your needs.

When using redux, the whole global state of app is stored in an object tree inside a single store. The only way to change the state tree is to create an action, an object describing what happened, and dispatch it to the store. To specify how state gets updated in response to an action, I need to write pure reducer functions that calculate a new state based on the old state and the action.

Redux helps I deal with shared state management, but like any tool, it has tradeoffs. It's not designed to be the shortest or fastest way to write code. It's intended to help answer the question "When did a certain slice of state change, and where did the data come from?", with predictable behavior. There are more concepts to learn, and more code to write. It also adds some indirection to my code, and asks me to follow certain restrictions. It's a trade-off between short term and long term productivity.

### 3.1.4 Tailwind css

Tailwind CSS is a highly customizable, low-level CSS framework that allows you to custom-build designs eliminating opinionated component styles that you would anyway want to override.

Beautiful custom user interfaces can be effectively built using CSS without actually putting in much coding effort. Being a utility-first framework, Tailwind CSS renders the leverage of styling every component distinctively, the way you desire. Adapting the "form follow function", the name of the utility class predicts the exact function of that class for ease in designing. Tailwind combines the most critical CSS properties which can be customized and extended according to project requirements.

There's a lot that goes on in web development. It can sometimes, thus, become an overly complicated task. Mapping the impact of styling might become tedious and time-consuming

hampering the progress of the application/website.

I choose to use Tailwind CSS because it eliminate all the above-mentioned issues. Tailwind CSS creates small utilities with a defined set of options enabling easy integration of existing classes directly into the HTML code. Custom styling can be provided to the components with the help of this framework.

## 3.2 Back-end Technologies

### 3.2.1 Golang and Gin

Golang or Go is an open source programming language that makes it easy to build simple, reliable, and efficient software. Based on the syntax of C, it comes with some changes and improvements to safely manage memory usage, manage objects, and provide static or strict typing along with concurrency. Go was created with the goal of

- The simplicity of use while increasing productivity
- Providing a high level of code efficiency
- Advanced performance for businesses.

I choose to use Go for the most backend service because Go has the same performance as C, but it is much easier to maintain than Java. Without the need for a virtual machine, Go boasts easier maintenance and no warming up period. These and many other characteristics are what make Golang stand out from its competitors such that fully concurrent and more stable, Accessible development tools and it's easy to grasp

Gin is a high-performance HTTP web framework written in Golang (Go). Gin allows you to build web applications and microservices in Go. It contains a set of commonly used functionalities (e.g., routing, middleware support, rendering, etc.) that reduce boilerplate code and make it simpler to build web applications. Due to its strong focus on extensibility, you can build a fitting solution for almost every web application scenario.

### 3.2.2 Python, Flask and Scikit-learn

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a

scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

I choose python as language to build sentiment analytic service because Python, with its rich technology stack, has an extensive set of libraries for artificial intelligence and machine learning. Here are some of them:

- Keras, TensorFlow, and Scikit-learn for machine learning
- NumPy for high-performance scientific computing and data analysis
- SciPy for advanced computing
- Pandas for general-purpose data analysis
- Seaborn for data visualization

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. I use Flask because it's light weight and fast to build web API.

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Scikit-learn comes loaded with a lot of features. Here are a few of them:

- **Supervised learning algorithms**, most of popular supervised machine learning algorithm is part of scikit-learn. Starting from Generalized linear models (e.g Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of scikit-learn toolbox.
- **Cross-validation**, There are various methods to check the accuracy of supervised models on unseen data using sklearn.
- **Unsupervised learning algorithms**, there is a large spread of machine learning algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.
- **Various toy datasets**, This came in handy while learning scikit-learn. I had learned SAS using various academic datasets (e.g. IRIS dataset, Boston House prices dataset). Having them handy while learning a new library helped a lot.

- **Feature extraction**, Scikit-learn for extracting features from images and text (e.g. Bag of words)

In sentiment analytic service, I use Scikit-learn to build a machine learning model to determine whether data (buyer comment) is positive or negative.

### 3.2.3 Mongo

MongoDB is an open-source document database built on a horizontal scale-out architecture. Founded in 2007, MongoDB has a worldwide following in the developer community.

Instead of storing data in tables of rows or columns like SQL databases, each row in a MongoDB database is a document described in JSON, a formatting language. Here's a simple JSON document describing a person information:

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

Figure 3.1: mongo document example

Document databases are extremely flexible, allowing variations in the structure of documents and allowing storage of documents that are partially complete. One document can have others embedded in it. Fields in a document play the role of columns in a SQL database, and like columns, they can be indexed to increase search performance.

I use mongo as database for all services because many reasons. First, The document data model is a powerful way to store and retrieve data that allows developers to move fast. Next, MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic. MongoDB also has a great user experience for developers who can install MongoDB and start writing code immediately. In addition, MongoDB has developed a large and mature platform ecosystem which means MongoDB can be used from all major languages, it has a worldwide community of developers and consultants, so it is easy to get help and it works on all types of computing platforms, both on-premise and in the cloud.

### 3.2.4 REST API

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding. REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular programming language to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

## 3.3 Deployment Technologies

### 3.3.1 Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

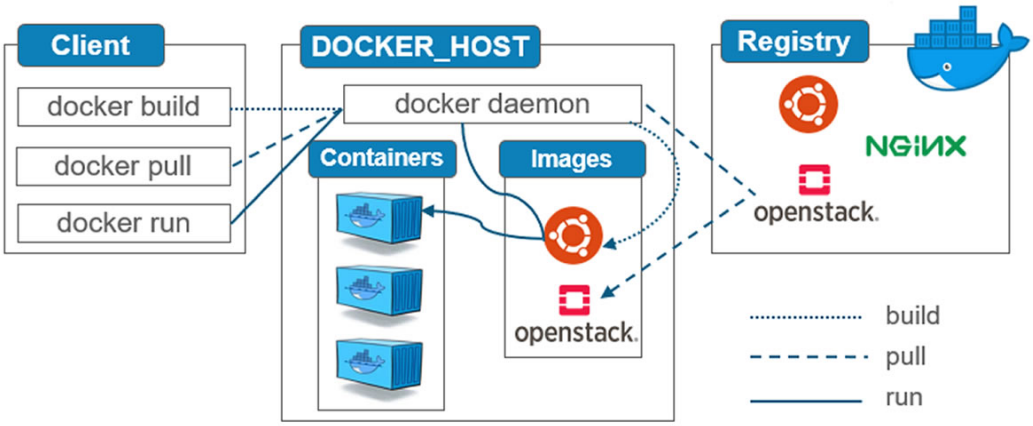


Figure 3.2: Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

I run all services of my application in docker containers because Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow I to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so I do not need to rely on what is currently installed on the host. I can easily share containers while I work, and be sure that everyone I share with gets the same container that works in the same way.

### 3.3.2 Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. Compose works in all environments: production, staging, development, testing, as well as CI workflows. Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker compose up and the Docker compose command starts and runs your entire app. You can alternatively run docker-compose up using the docker-compose binary.

I use docker-compose because of it's benefits:

- Single host deployment - This means I can run everything on a single piece of hardware
- Quick and easy configuration - Due to YAML scripts
- High productivity - Docker Compose reduces the time it takes to perform tasks
- Security - All the containers are isolated from each other, reducing the threat landscape

### 3.3.3 Github actions

GitHub Actions is a new feature of github which enables the user to create custom Software Development Life Cycle (SDLC) workflows in their GitHub repositories. It gives a privilege to the repository owner to write individual actions and then combine them to create a custom workflow of their choice for their project. GitHub Actions also provides the facility to build end-to-end Continuous Integration (CI) and Continuous Deployment (CD) capabilities directly in the repository.

However, the owner does not need to write the actions by themselves. GitHub has an inbuilt market place where people can find the actions created by other people, and reuse them if it fits their needs. Since GitHub actions is enabled on all repositories and organizations by default, extra efforts are not needed for getting started with it. Github actions offer advantages over other CI tools like Jenkins:

I found Actions easy to pick up as I'm familiar with other CI systems such as CircleCI and GitlabCI. What they share in common is the markup language YAML. GitHub Actions supports the three major operating systems, Windows, MacOS and Linux, and you can run any programming language supported by those. Apart from being triggered by pull requests and commits, actions allows you to respond to any GitHub event. It allows you to trigger certain GitHub Actions workflows (including open source actions) based on the creation of issues, comments, the joining of a new member to the repository and so on.

*Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.*

Albert Einstein

# Chapter 4

# Application Development and Deployment

In the third chapter, I have explained the technologies and frameworks adopted in the application. In this chapter, the process of developing and deploying the application will be described in details. It is divided into five parts: Architectures, Design, Development, Testing and Deployment.

## 4.1 Architecture Design

Many applications nowadays are built in monolithic architecture or in other words, they are built as a single unit. These applications generally have three main parts: the user interface, the data access layer, and the data store. The first and second ones are usually combined together into a single program on a single platform, tightly coupled together and depend on each other in order to run properly. In spite of the fact that this architecture is simple to develop and test, it is challenging for developers to maintain, scale vertically and adopt new technologies.

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

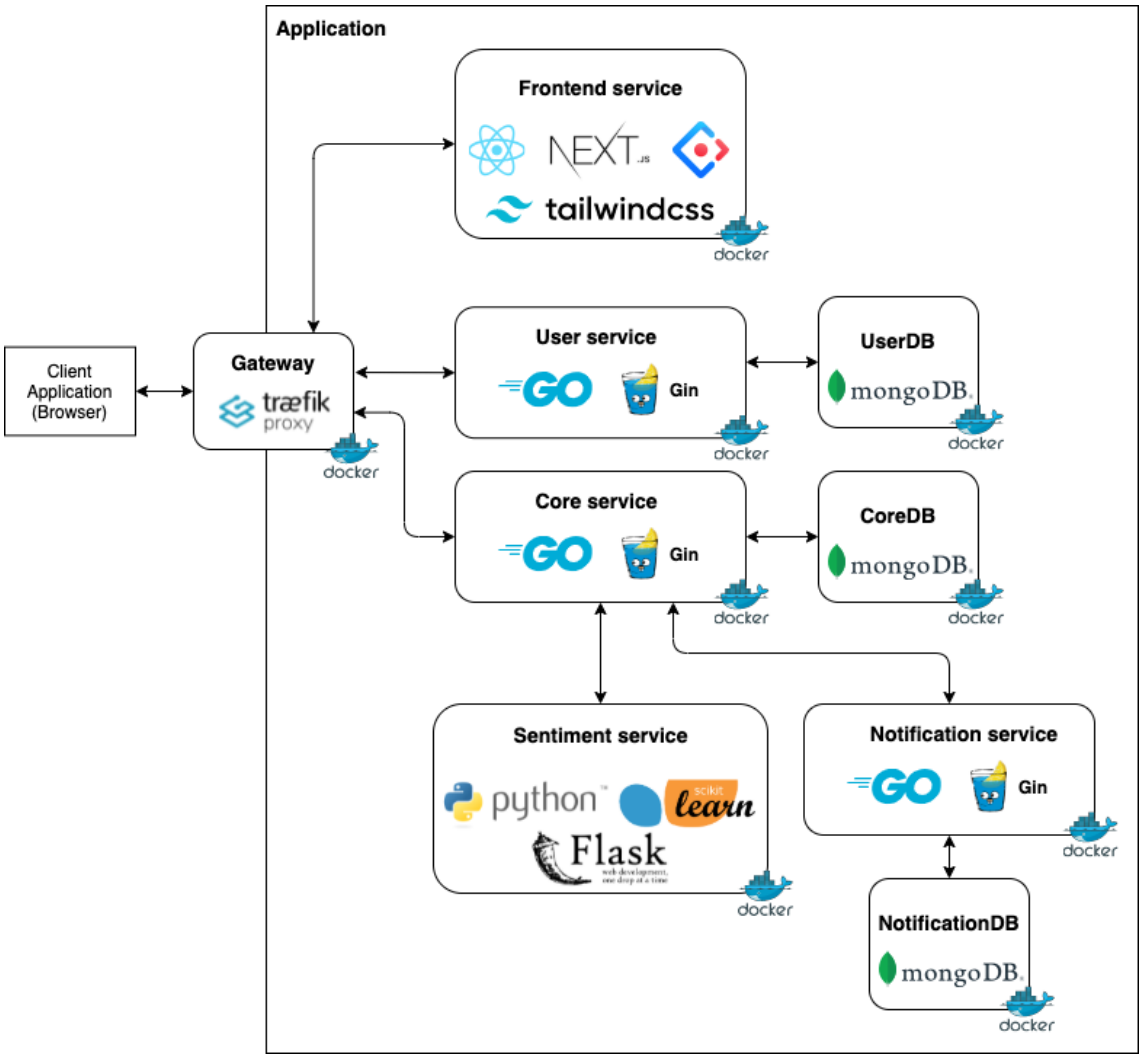


Figure 4.1: General Architecture

For my competitor monitoring application, I choose to follow the microservice architecture to build the application. There are five main services as specified in Figure 4.1 (i) user service providing registration, login and user management functions, (ii) frontend service serve the files that browser need to render the UI for user, (iii) core service providing all the API to handle all business logic in the app, (iv) notification service providing API for notification functions such an notify through email, get user notifications and (iv) sentiment analytic service providing API that determine whether comment is positive or negative.

In my architecture design, I have an API gateway that sits between the client apps and the microservices. It acts as a reverse proxy, routing requests from clients to services. It can also provide other cross-cutting features such as rate limiting, SSL termination or authentication. A service calls an API that another service exposes, using a HTTP protocol. I create a docker container for each individual microservice. Each of these containers will be responsible for one specific business functionality and it's also can then be independently maintained and deployed.

## 4.2 Detail Design

### 4.2.1 User interface

The frontend service provides the interface of the system, where users interact with the system. It is developed with Nextjs, a framework for web application development with server-side rendering as mentioned in section 3.1. Redux is also utilized to manage the application state in a simple and effective way.

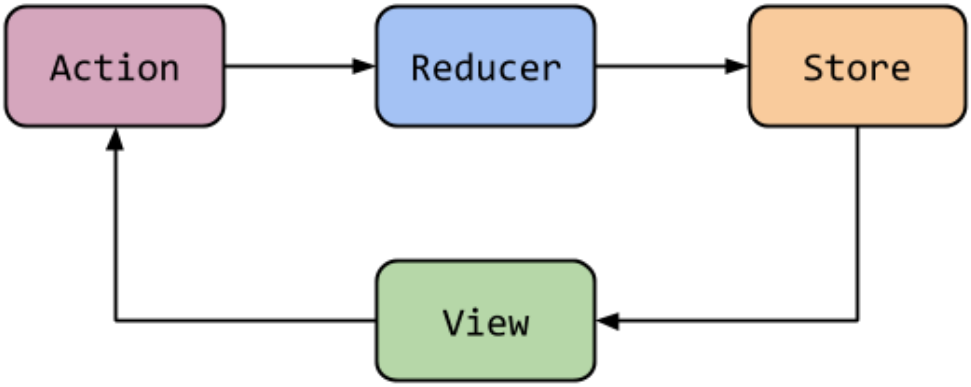


Figure 4.2: Frontend architecture

Applying Redux as a state management, I have separated the data flow and user interface. The arrow in the Figure 4.2 illustrate the data flow of application. This designed architecture centralized state of the application that can enable powerful capabilities like undo/redo, state persistence, and much more. As I have separated the concern, the application become flexible, debuggable and consistent web application. The detail specification of user interface is described as table below.

| | |
|---|---|
| Screen resolution | Appropriate for screen with resolution of 1360px or higher |
| Screen size | 14 inches or higher |
| Color | Fully support RGB color scheme, or RGBA color scheme. |
| Element des | Make use of borders, shadows with appropriate color. |
| Font | Noir Pro |
| Display error | Display alert |
| Language | English |

Table 4.1: User interface specification

### 4.2.2 Use case design

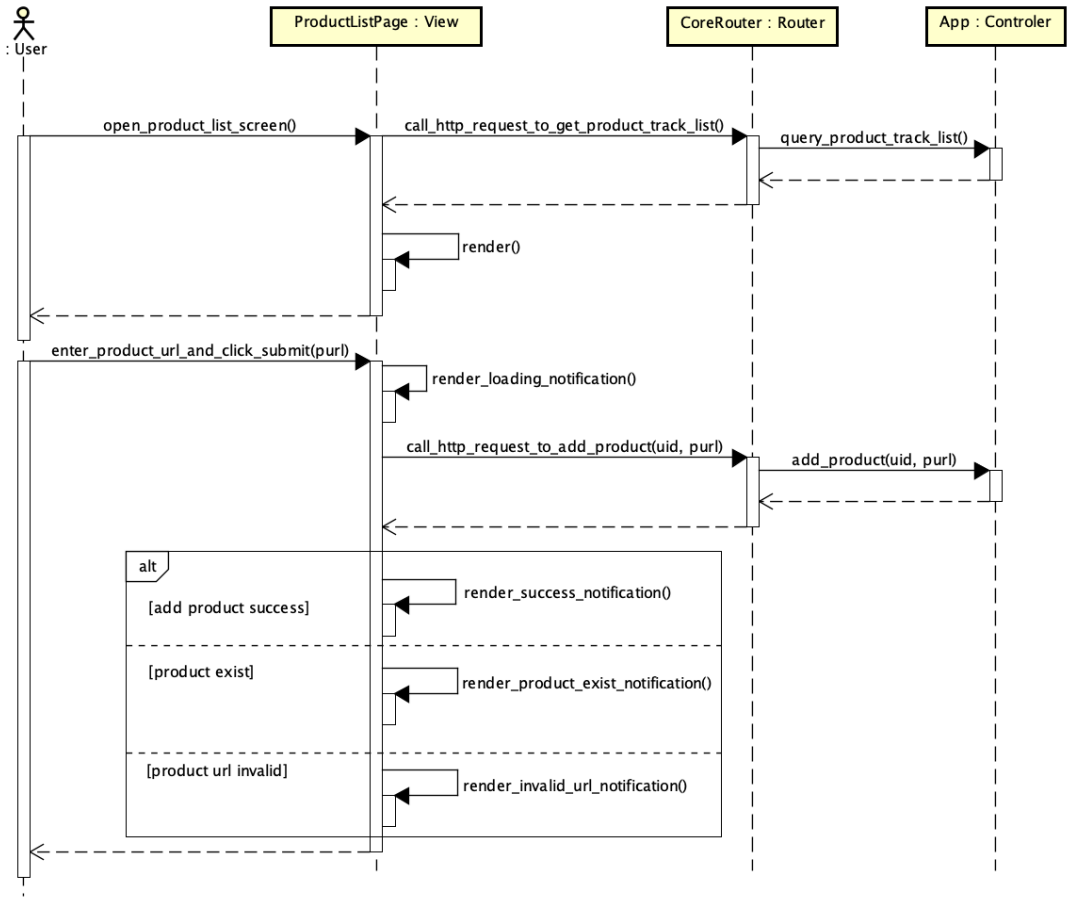


Figure 4.3: Sequence Diagram for usecase "Add Product to track list by url"

Figure 4.3 illustrates use case "Add product to track list by url" by a sequence diagram. It can be seen from the figure that there are two big steps.

First, User opens the product list page. This view will send a http request to get list of tracking products. The router will redirect the request to the corresponding controller. This controller is in charge of communicating with database to get the product list and return it to the view, which display the list to user.

Next, User enter the url of the new product that user want to track and click the submit button. The ProductListPage view component will send a http request again, but to add a new product to track list. The request will be routed to the controller which take user id, product url as input and then do some tasks for adding product to user track list such as validate input, check the product whether in list or not. When controller finish process the request, it returns a response to ProductListPage view. The view render notification depends on the response.

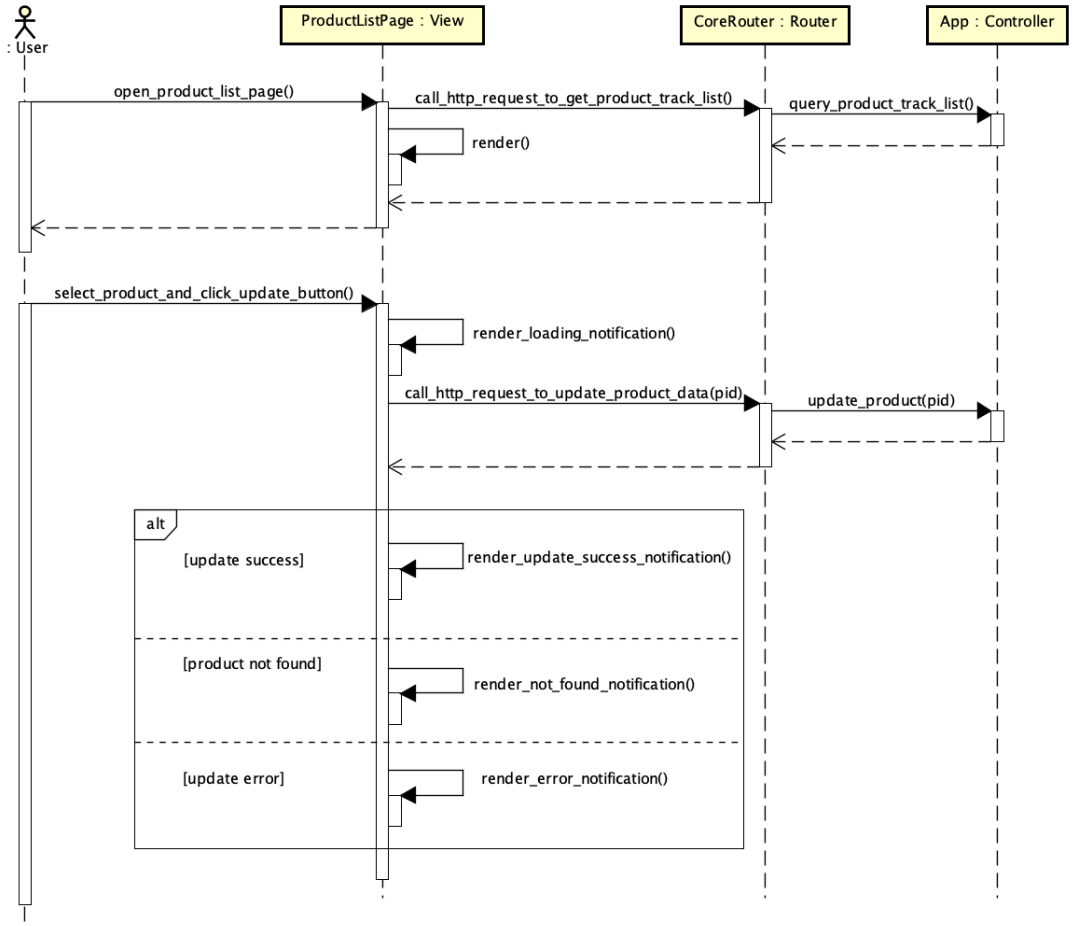


Figure 4.4: Sequence Diagram for usecase "Update product data"

Figure 4.4 illustrates use case "Update product" by a sequence diagram. It can be seen from the figure that there are two big steps.

The first step is similar with the sequence diagram of usecase "Add Product to track list by url". Next, User select a target product and click the update button. The ProductListPage view component will render a loading notification and send a http request to update the product data. The request will be routed to the controller which take product id as input and then do some tasks for update the product such as validate the product id and check the product whether crawled in database. When controller finish process the request, it returns a response to ProductListPage view. The view render notification depends on the response.

### 4.2.3 Component design

#### Frontend service component design

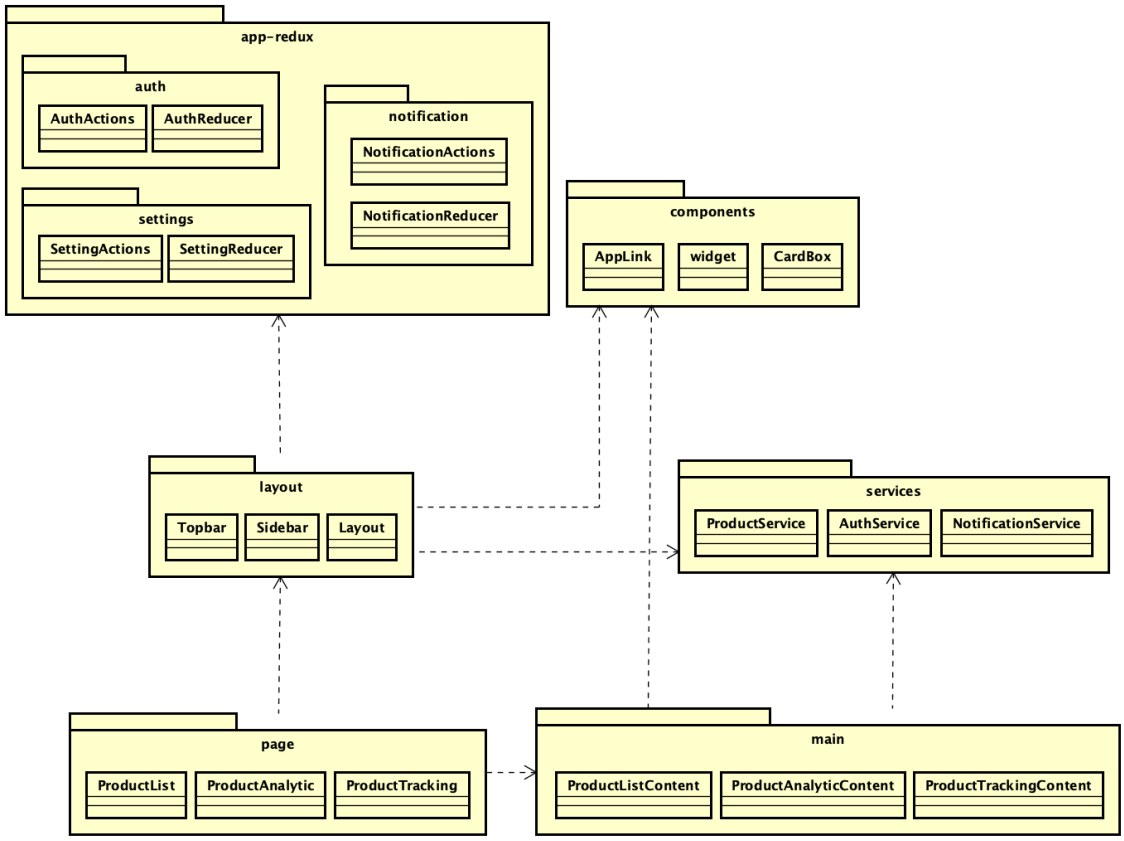


Figure 4.5: Frontend service component design

The above figure 4.5 demonstrates the package of frontend service. There are five six main packages include page, main, layout, services, components and app-redux. The components package has the React component that are reusable in different pages. The service package contains all code to access the APIs of my application, I have one class for each REST API controller. Layout package are used to wrap my template and provide them with the component which will be displayed by default in a specific layout. For example, I include the Footer, Header and navigation bar in a default layout. The pages package behaves as a sort of one-to-one static router for my application. Each class will match a route as described in NextJS documentation. I put all code that related to redux into app-redux package. I follow redux module pattern so in app-redux package, you can see some sub-package which are the modules of redux to manage state for different app domain. Finally, the main package will container the page content component, each class in page package will associate with a class component in main package to make up a complete page.

**core service component design**

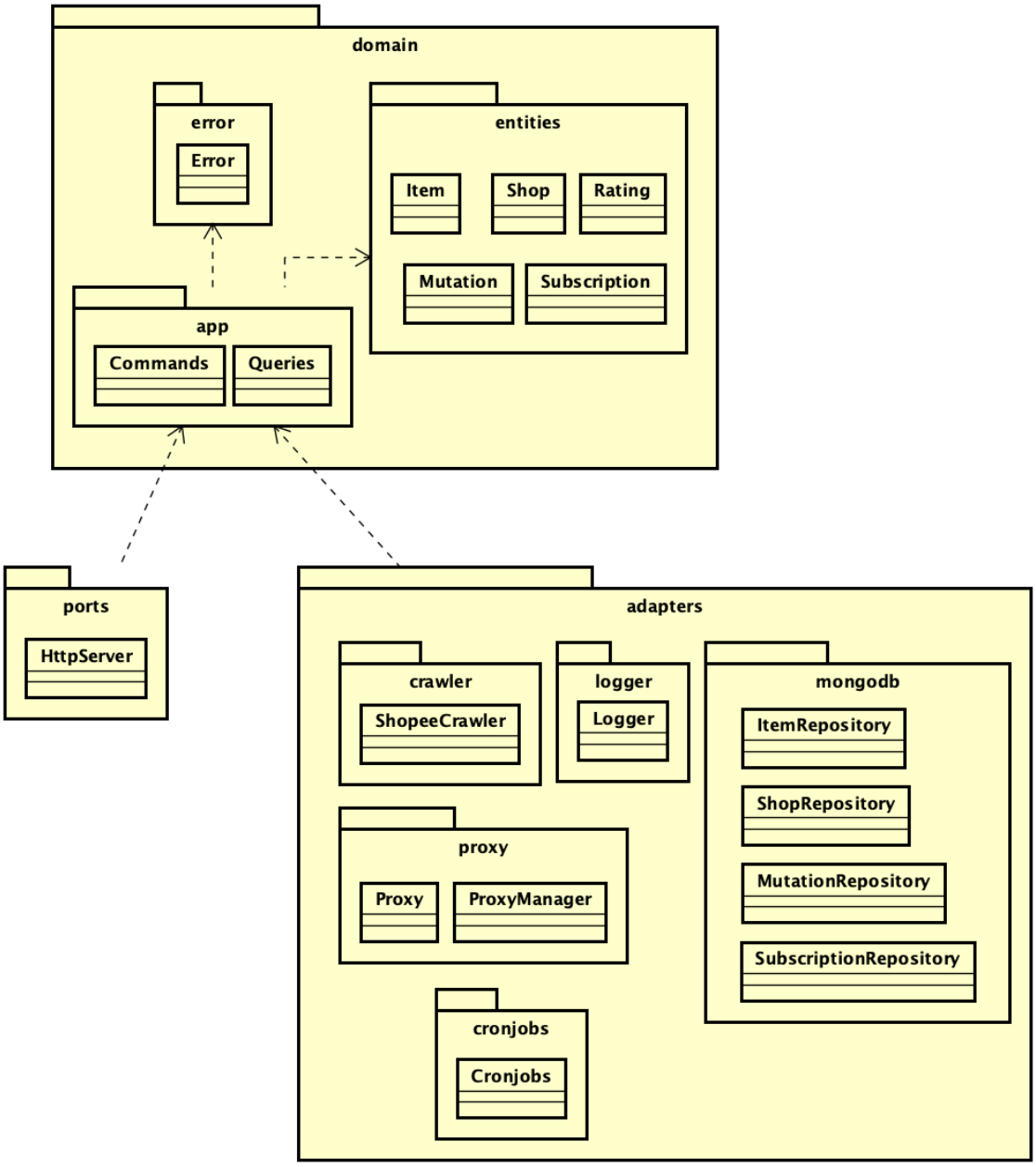


Figure 4.6: core service component design

I choose to build the core service follows clean architecture [3] and CQRS design pattern [1]. There are three main packages. Domain package contains code related to domain business logic of application, code in here never depends on UI, Database, Web Server, or any other external element. Adapters package container adapter which is how my application talks to the external world such as database, crawler or log library. A port is an input to my application, and the only way the external world can reach it. Port could be an HTTP or gRPC server, a CLI command, or a Pub/Sub message subscriber. You can see in figure 4.6 in core service, I have only one port to my application, which is an http server.

### 4.2.4 Database design

**Entity Relation diagram**, It can be seem from Figure 4.7 that the system has 9 entities. Each user associate with one subscription and have many notifications. Each subscription contains many item entities and shop entities, which indicate that user can track multiple items and shops at the same time. Item and shop entity can have many mutations represent for changes in time of this entity. An Item have many categories, ratings and also contains references to its self to show to similar items. A rating entity can have multiple tags.

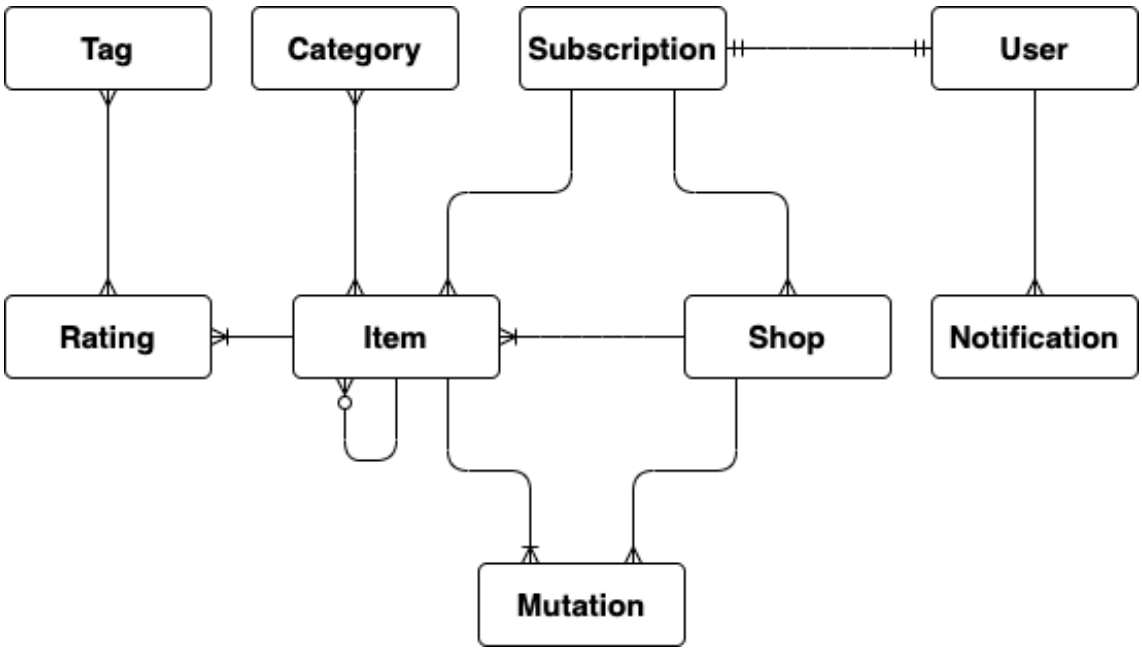


Figure 4.7: Entity Relation Diagram

From the entity relation diagram, I designed a database schema for NoSQL MongoDB. Each entity will correspond with a collection in mongo. The detail design for each collection are listed by tables below.

Table 4.2: Detail database design

| Collection | Field | Type | Description |
|---|---|---|---|
| User | _id | ObjectID | The primary key for document |
| | username | string | user name |
| | email | string | email of user |
| | avatar | string | file name for avatar of user in system |
| | password | string | hashed user password |
| | role | string | role of user |

| Continuation of Table 4.2 | | | |
|---|---|---|---|
| Collection | Field | Type | Description |
| Item | _id | ObjectID | The primary key for document |
| | shopid | string | shop id of item |
| | name | string | display name of item |
| | brand | string | brand name of item |
| | image | string | hash code for main image of item |
| | category_ids | Array[string] | list of category id of item |
| | origin | string | item origin url |
| | price | int64 | price of item before discount |
| | discount | int64 | discount for item |
| | sold_count | int | item sold count |
| | cmt_count | int | item comment count |
| | liked_count | int | item like count |
| | view_count | int | item like count |
| | rating_star | double | item rating star |
| | rating_count | Array[int] | item star count |
| | ctime | Date | document created time |
| | mtime | Date | document modified time |
| | lastcheck | Date | item last checked time |
| Subscription | _id | ObjectID | The primary key for document |
| | userid | string | user id corresponds with the subscription |
| | items | Array[string] | list of item id that user subscribed |
| | shops | Array[string] | list of shop id that user subscribed |
| Mutation | _id | ObjectID | The primary key for document |
| | type | string | mutation type |
| | ctime | Date | created time for the document |
| | other fields depends on mutation type | | |
| Tag | _id | ObjectID | The primary key for document |
| | description | string | tag description |
| Category | _id | ObjectID | The primary key for document |
| | name | string | category name |
| | image | string | category image |

| Continuation of Table 4.2 | | | |
|---|---|---|---|
| Collection | Field | Type | Description |
| Shop | _id | ObjectID | The primary key for document |
| | userid | string | shop owner user id |
| | avatar | string | shop avatar |
| | name | string | shop display name |
| | active_voucher | [Array]Voucher | shop active vouchers |
| | follower_count | int | shop follower count |
| | item_count | int | quantity of item in shop |
| | response_rate | int | shop response rate |
| | response_time | int | shop response time in mini second |
| | ctime | Date | document created time |
| | mtime | Date | document modified time |
| | lastcheck | Date | item last checked time |
| Notification | _id | ObjectID | The primary key for document |
| | created_at | Date | created time of document |
| | kind | string | notification kine |
| | status | string | notification status |
| | user_email | string | user email of notification owner |
| | user_id | string | user id of notification owner |
| Rating | _id | ObjectID | The primary key for document |
| | comment | string | comment of buyer for rating |
| | ctime | Date | created time of document |
| | itemid | string | id of item that rating create |
| | shopid | string | id of shop that rating create |
| | tag_ids | Array[string] | list of tag id for rating |
| | rating_star | int | rating star |

## 4.3 Application Construction

### 4.3.1 Library and Tools

The following table contains all libraries, frameworks and tools that I used to finish this project.

| Purpose | Library/Tool | URL |
|---|---|---|
| Code editor | Visual Studio Code | code.visualstudio.com |
| Package manager | npm<br>pip<br>Go module | www.npmjs.com<br>pip.pypa.io/en/stable<br>blog.golang.org/using-go-modules |
| Javascript library | ReactJS<br>ramda<br>recharts<br>redux<br>socketio client | reactjs.org<br>ramdajs.com<br>recharts.org<br>redux.js.org<br>github.com/socketio/socket.io-client |
| Golang library | Gin<br>Mongo driver<br>Crontab manager<br>colly | github.com/gin-gonic/gin<br>github.com/mongodb/mongo-go-driver<br>github.com/robfig/cron<br>github.com/gocolly/colly |
| Python library | Flask<br>scikit-learn<br>numpy | flask.palletsprojects.com<br>scikit-learn.org/stable<br>numpy.org |
| React UI library | Ant design | http://ant.design |
| DBMS | MongoDB | www.mongodb.com |
| React Framework | NextJS | nextjs.org |
| API Development Environment | insomnia | https://insomnia.rest |
| MongoDB Management tool | Robo 3T<br>MongoDB Compass | robomongo.org<br>www.mongodb.com/products/compass |

Table 4.3: Libraries and tools in project

### 4.3.2 Achievements

The application have been developed successfully with the features that acquire five module that i mentioned in section 1.2. seller can register an account in application an start monitoring their competitors with tracking and analytic screen. Admin can login to the app with admin account and observe the statistic of the whole system with dashboard interface.

The application specifications are described in Table 4.4

| Service | Specification |
|---|---|
| Core service | Source lines of code: 9532<br>docker image size: 19.6MB<br>Number of API: 33 |
| Frontend service | Source lines of code: 5499<br>docker image size: 381MB<br>Number of component: 52 |
| User service | Source lines of code: 1864<br>docker image size: 16.2MB<br>Number of API: 9 |
| Notification service | Source lines of code: 1292<br>docker image size: 18.2MB<br>Number of API: 6 |
| Sentiment analysis service | Source lines of code: 102<br>docker image size: 1.32GB<br>Number of API: 1 |

Table 4.4: Application specification

### 4.3.3 Main screens and features

In this section, I will illustrate some main functions of the competitor monitoring application such that (i) data management and (ii) user management. The other functions which can help sellers do the task of monitoring competitor will be presented in section 5.1

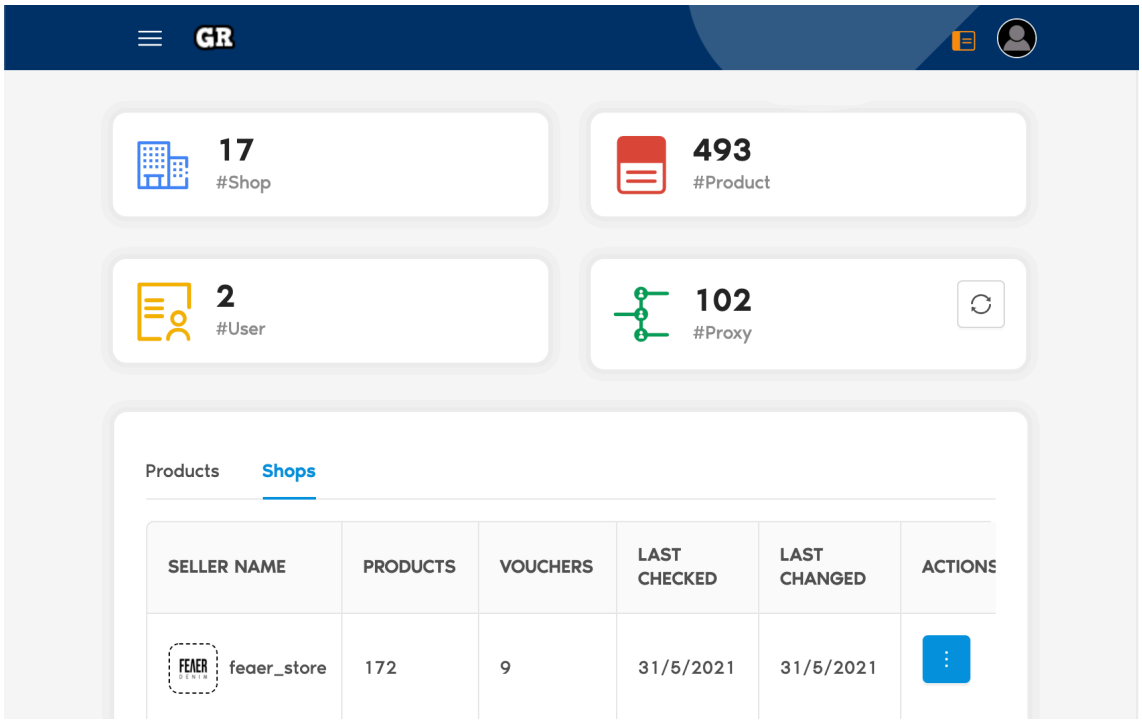


Figure 4.8: Data management screen with small screen size

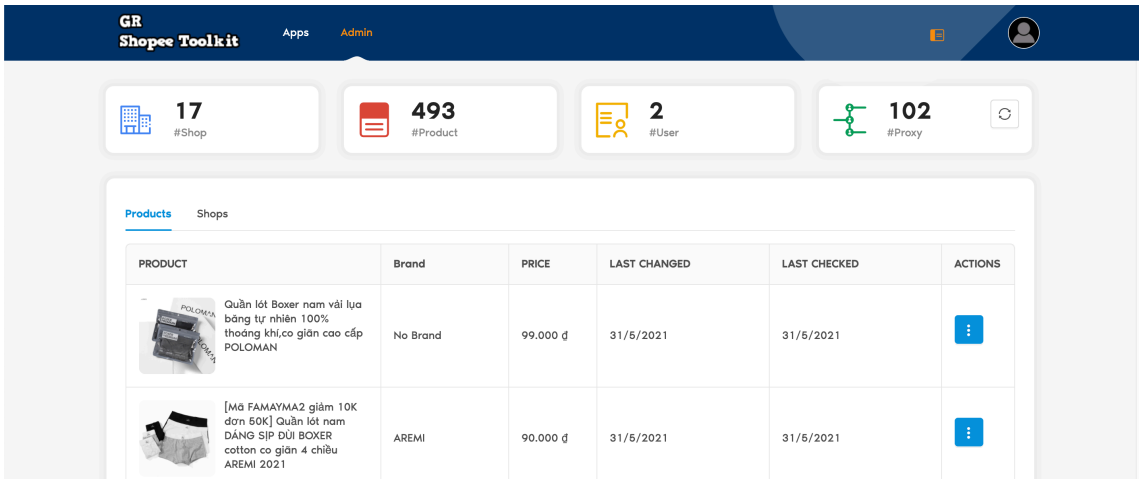


Figure 4.9: Data management screen with medium screen size

The figure 4.9 and 4.8 are the screen for data management of the application. Only admin user can access this screen. Admin can observe some statistic of the app such that the number of shop, product, user and the number of proxy used. In addition, admin also can see all of product and shop added by all user of app and select any product or shop to see the tracking and analysis data of it. The layout of app can support small screen by change the position of navigation bar style from always on the top to only display when click to the button on top left of the screen.

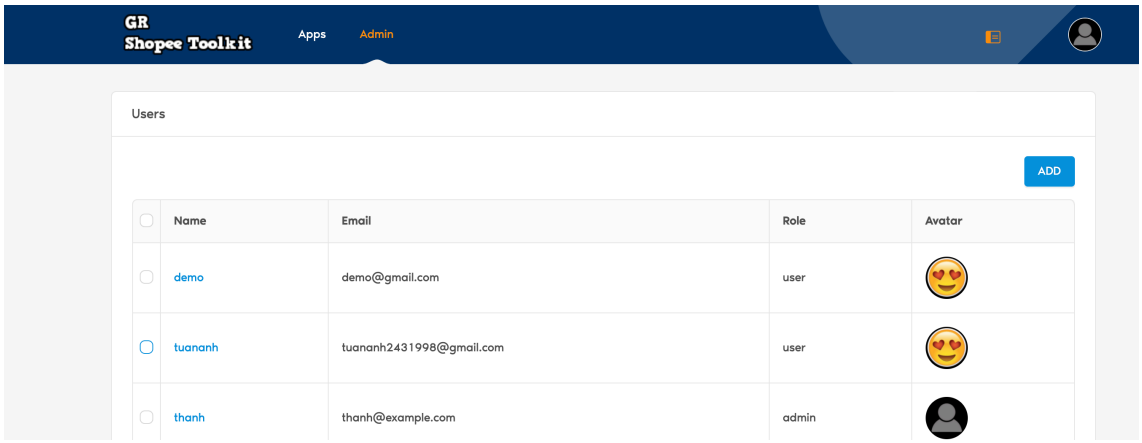


Figure 4.10: User management (user list) screen

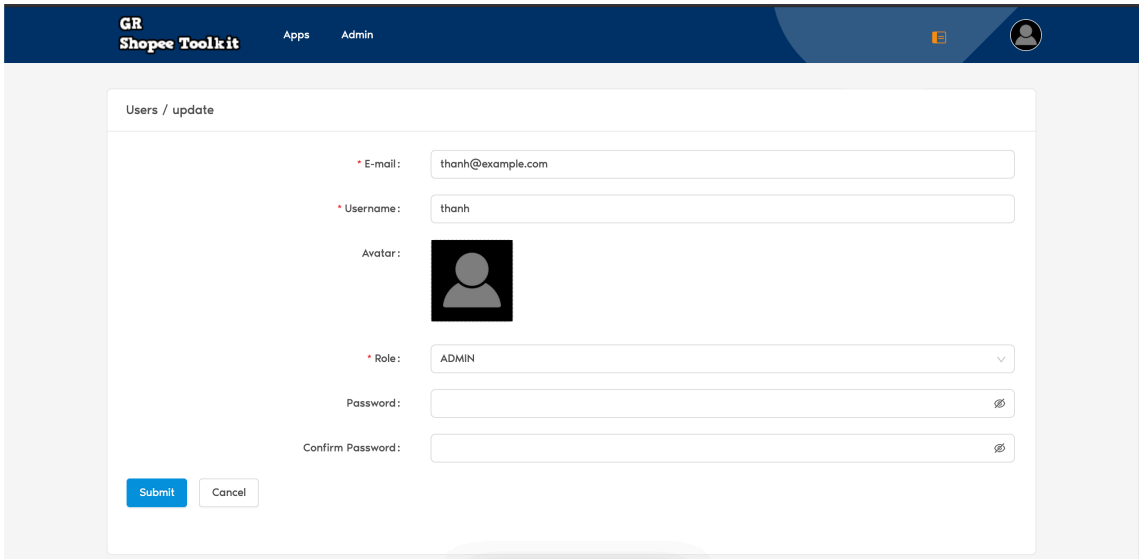


Figure 4.11: Data management (user edit) screen

The figure 4.10 and 4.11 are the screen for user management of the application. similar to data management screen, only admin user can access this screen. Admin can do the task of CRUD within this screen.

## 4.4 Test

### 4.4.1 Test for "Add product to track list" usecase

Pre-condition: User logged in to the application.

| Test case | Steps | Expected result | Result |
|---|---|---|---|
| Successfully add product to track list | Go to products list page | Display product track list of user | Pass |
| | Enter a valid product url that not exists in the list | Display the product url | Pass |
| | Click start button | The Start button change to loading state, then a success notification appear, the button change to default state. The product track list reload with new product | pass |
| Add product to track list by invalid url | Go to products list page | Display product track list view | Pass |
| | Enter a invalid product url | Display the product url | Pass |
| | Click start button | The Start button change to loading state, then a error notification appear with the invalid url message, the button change to default state. | pass |
| Add product that already exists on track list | Go to products list page | Display product track list view | Pass |
| | Enter a valid product url that already on track list | Display the product url | Pass |
| | Click start button | The Start button change to loading state, then a error notification appear with already exists message, the button change to default state. | pass |

Table 4.5: Test cases of "Add product to track list" usecase

### 4.4.2 Test for "Update product data" usecase

Pre-condition: User logged in to the application.

| Test case | Steps | Expected result | Result |
|---|---|---|---|
| Successfully update product data | Go to products list page | Display product track list of user | Pass |
| | Hover over the button on action column of the target product | Display the actions list | Pass |
| | Click update options | A loading notification appear on the top middle of screen, then a update success notification is displayed on top right of screen. After that, the product list reload with new data | pass |

Table 4.6: Test cases of "Update product data" usecase

### 4.4.3 Test for "Delete product from track list" usecase

Pre-condition: User logged in to the application.

| Test case | Steps | Expected result | Result |
|---|---|---|---|
| Successfully delete one product from track list | Go to products list page | Display product track list of user without delete button | Pass |
| | Click to check box of any product | Display the delete button | Pass |
| | Click delete button | A confirm popup is displayed | pass |
| | Click yes button | loading notification appear on the top middle of screen, then a delete success notification is displayed on top right of screen. After that, the product list reload without the deleted product | Pass |

Table 4.7: Test cases of "Delete product from track list" usecase

## 4.5 Deployment

Project have developed and deployed successfully. Until now, all the services of the competitor monitoring application have been deployed in a VPS of digital ocean with a domain mounted to this VPS.

| UI/Service | Domain |
|---|---|
| Main Application | insight.smartecommerce.tech |
| Gateway dashboard | traefik.smartecommerce.tech |
| Core service api | api.insight.smartecommerce.tech |
| Notification service api | notification.insight.smartecommerce.tech |
| Sentiment service api | sentiment.insight.smartecommerce.tech |
| User service api | user.insight.smartecommerce.tech |

Table 4.8: Domain to application

| Specification | value |
|---|---|
| Cloud provider | Digital Ocean |
| Resource type | Droplet |
| OS | Ubuntu 20.04 (LTS) x64 |
| Datacenter region | SGP1 (Singapore) |
| CPU | 2 intel vCPU |
| RAM | 4GB |
| Storage | 50 GB |

Table 4.9: Host machine specification

*Patience is a key element of success.*

Bill Gates

# Chapter 5

# Main Contributions

In chapter 4, I introduced the process of developing and deployment the application. I explained its architecture in both front-end and back-end layers. Use case and database design were also clarified. In this chapter, I will present my main contributions to the application development process, including: How I develop the competitor monitoring tool for shopee sellers, solutions to use microservices architectures and how to build a continuous delivery pipeline for all microservices.

## 5.1 Develop a competitor monitoring tool for shopee seller

### 5.1.1 Problem

It's unrealistic when a seller don't have any competition for their ecommerce stores on Tiki or Shopee. No matter how different an idea inspired seller to build their own ecommerce store, several online shops would already be selling similar or the same products. To be successful in ecommerce, a seller need to consistently track what their competitors are doing and what gets customers in order to help them a great deal in mitigating the risks. However, Monitoring competition can take up a lot of time, so seller need a tool to help them to do these task. As I mentions in section 2.1, this kind of applications in the market today, still have some limitations.

First, either of them have sentiment analytic of customer to competitor feature. This results in difficulties for users to gain an overview of the wider public opinion behind certain competitors. Next, these applications are try to cover many domain of ecommerce sellers supporter, so that it make these application are kind of difficult to use. Moreover, none of these applications have the feature that allow user to manual trigger update data for a specific product or shop. These data only updated in a fix periodic that depends on these applications.

### 5.1.2 Solution

To solve the limitations I mentions in sections 5.1.1, I will develop an competitor monitoring applications to tackle these problems.

First, I research about similar applications in market about what information sellers should monitor about their competitors. After researching, I see that there are two common marketplace competitor intelligence solutions: Product competitor monitoring and Shop competitor monitoring.

Product competitor monitoring refers to keeping an eye on a specific product of your competitor and being alerted anytime a change is made. For instance, if you are an online seller with a Coolmate shirt as your best-selling product, it would make sense to check into other bestselling Coolmate shirt on your marketplace and how they format their product listings and manage their pricing. By examining their product listing changes, you can have a good idea of how much effort/resources they're investing in their listings. Seeing how your competitors are writing their product listing gives you a better look on how to write your product listings as well.

Another valuable source of competitor intelligence is their shop. You can manually navigate to your competitors' stores and see the below:

- Active vouchers
- Followers
- Seller rating
- Product quantity
- Reviews

As with competitor product monitoring, it can be very time consuming to do this manually and most larger sellers and brands use automation to get competitor store change alerts.

Next, I define user scenarios that let sellers interact with my applications to monitor their competitors. The Figure 5.1 is an example of a main user scenario in my application. It's describe a process for user to add new product or shop to track list for monitoring.

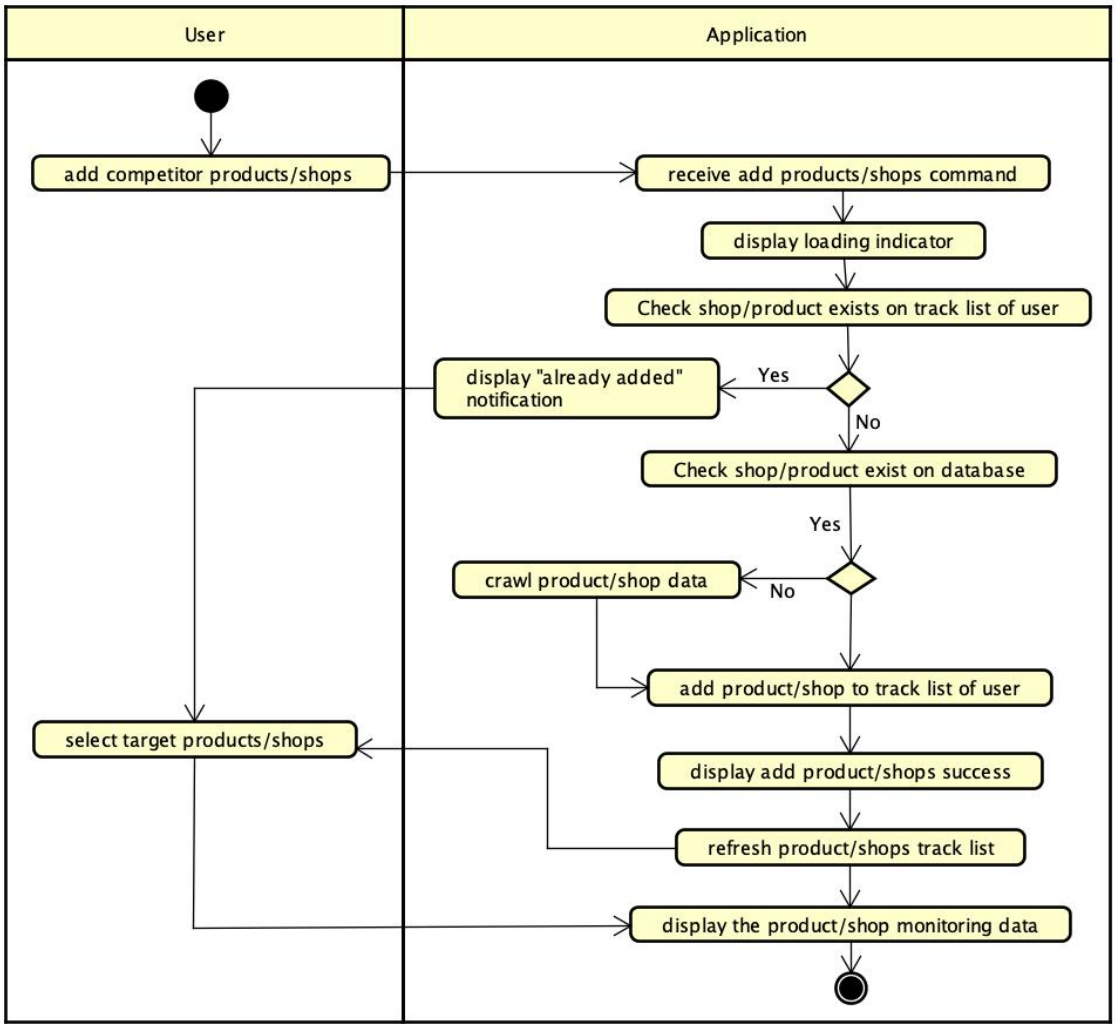


Figure 5.1: Activities diagram for "Add Product to monitoring track list"

Finally, when I have all user scenarios for my application. I start research all the technologies needed and apply them to develop a product that implement these scenarios.

### 5.1.3 Result

I have developed an visually web application that allow seller in shopee monitoring their competitor in fast and convenient way.

First, Seller need to register an account and login in, then they can add competitor product(s)/shop(s) to the track list in two ways. Sellers can add a product/store by inputting the competitor product/shop url to the input field or updating a csv file that contains a list url of products/shops by clicking "Upload CSV" button. Then, seller click "Start" button to add start the process.

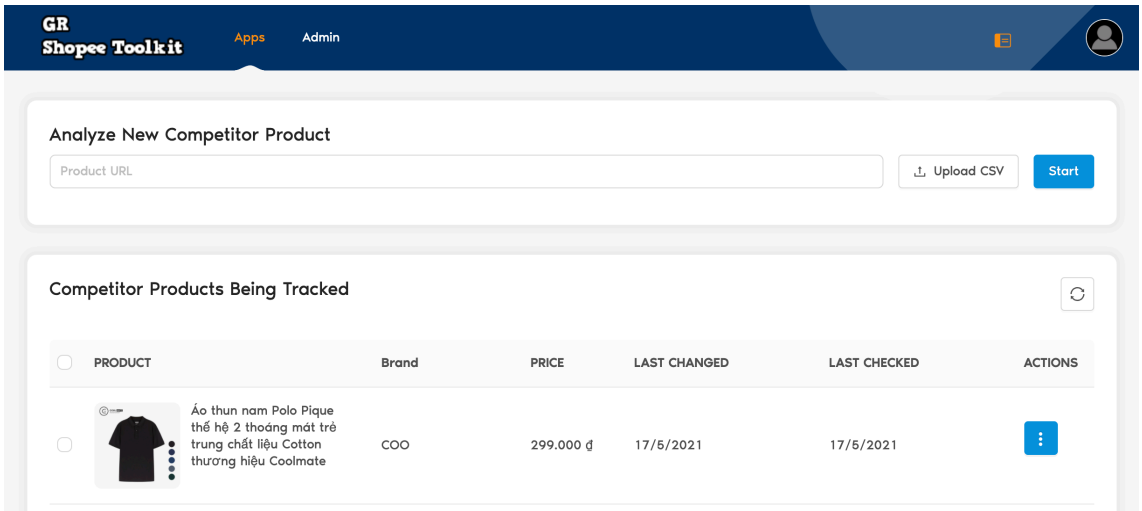


Figure 5.2: UI for adding product to track list

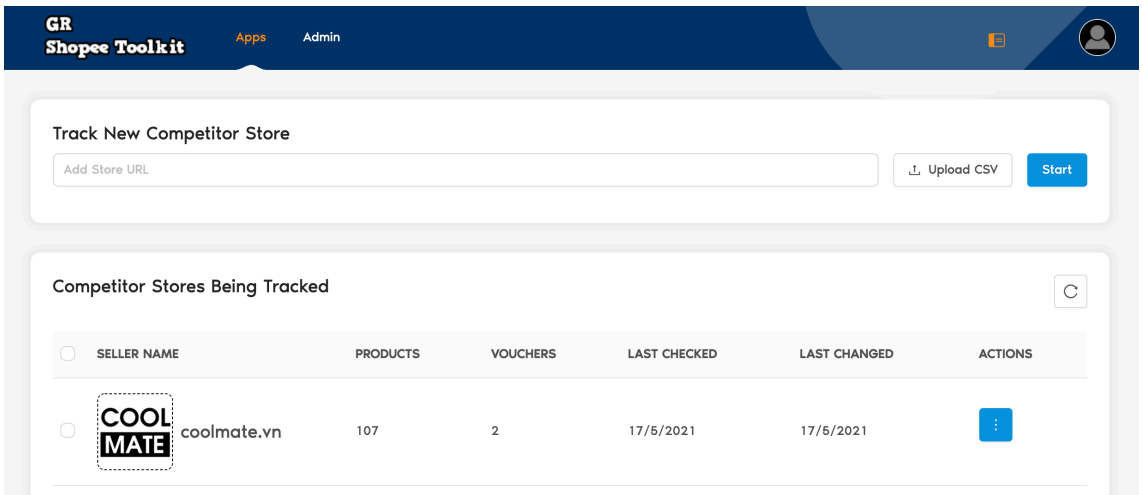


Figure 5.3: UI for adding shop to track list

Next, Seller can view the monitoring data of a product/shop in track list by hover the button in the "ACTIONS" columns as displayed in figure , and then click to "Tracking" button to see the tracking data of product and "Review analytic" to see the sentiment of buyer to product/shop by analytic of reviews. Seller can also update the product/shop data by click to the "Update" button

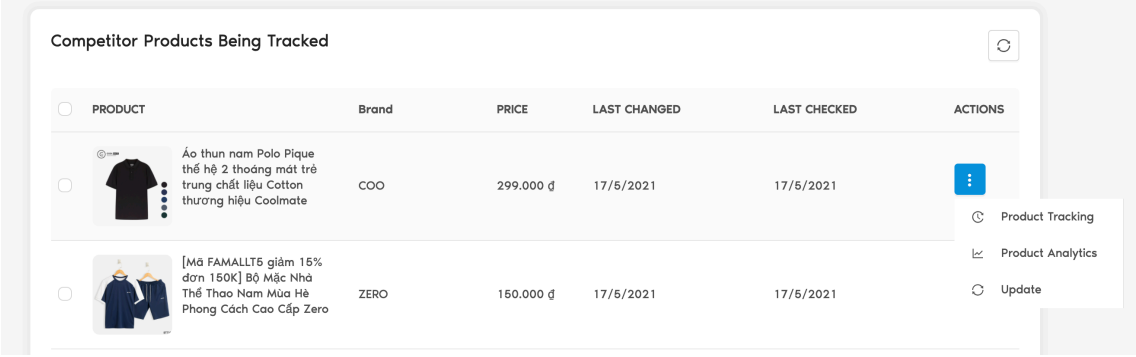


Figure 5.4: UI for product feature navigation

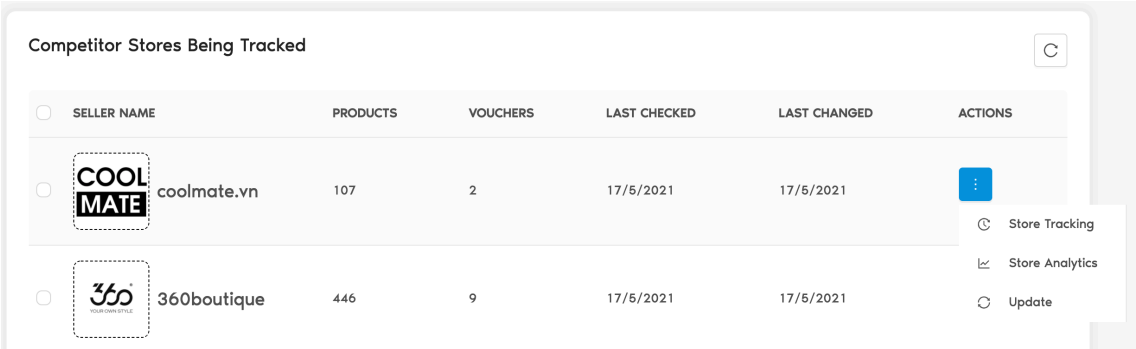


Figure 5.5: UI for shop feature navigation

In product tracking screen, some basic information of product are displayed on the left side and the tracking data is displayed on the right side, which includes data tracking for price, title, ratings and related products. The data will be updated continuously with an interval of time and seller can manually set that interval.

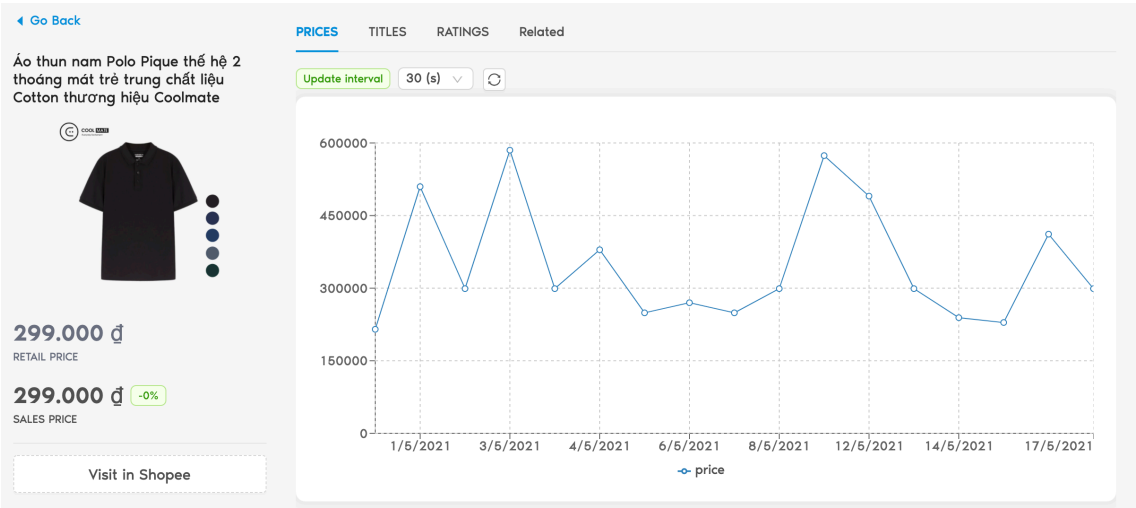


Figure 5.6: UI for product price tracking data display

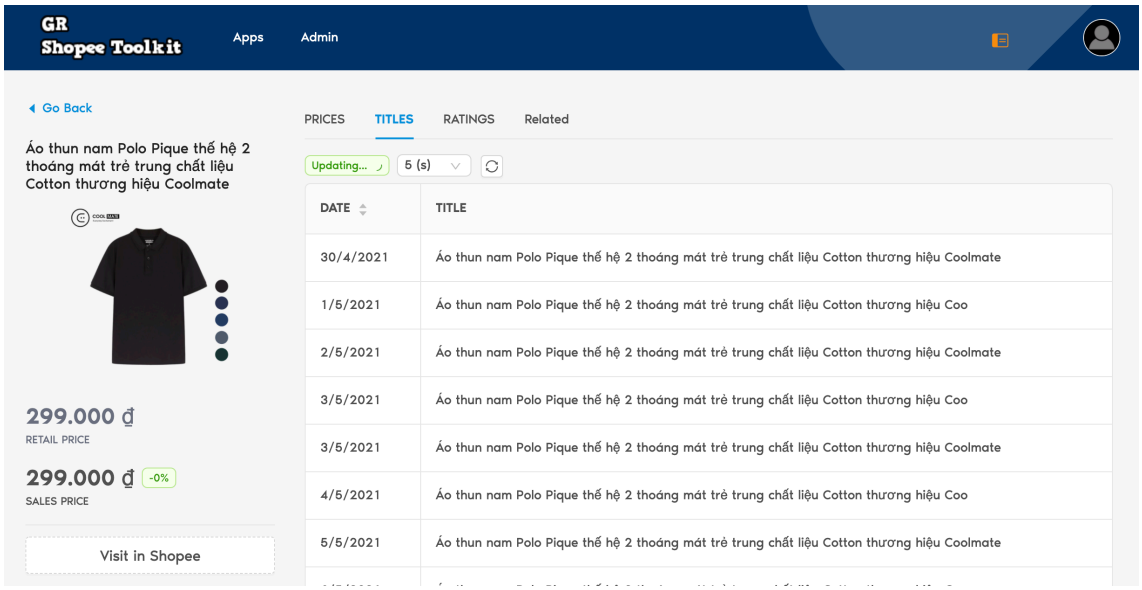


Figure 5.7: UI for product price title data display

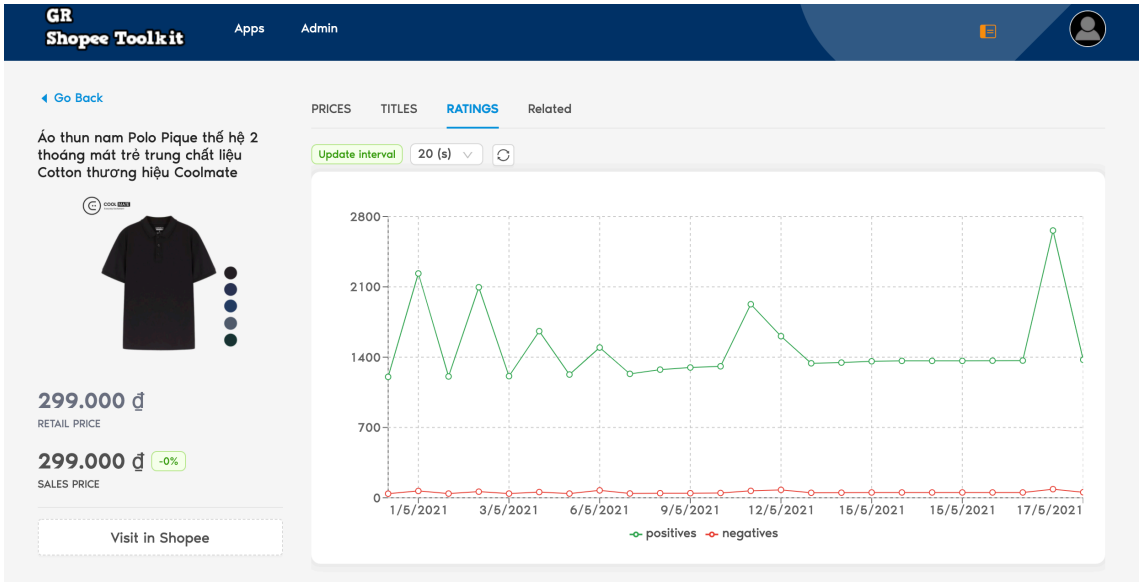


Figure 5.8: UI for product ratings tracking data display

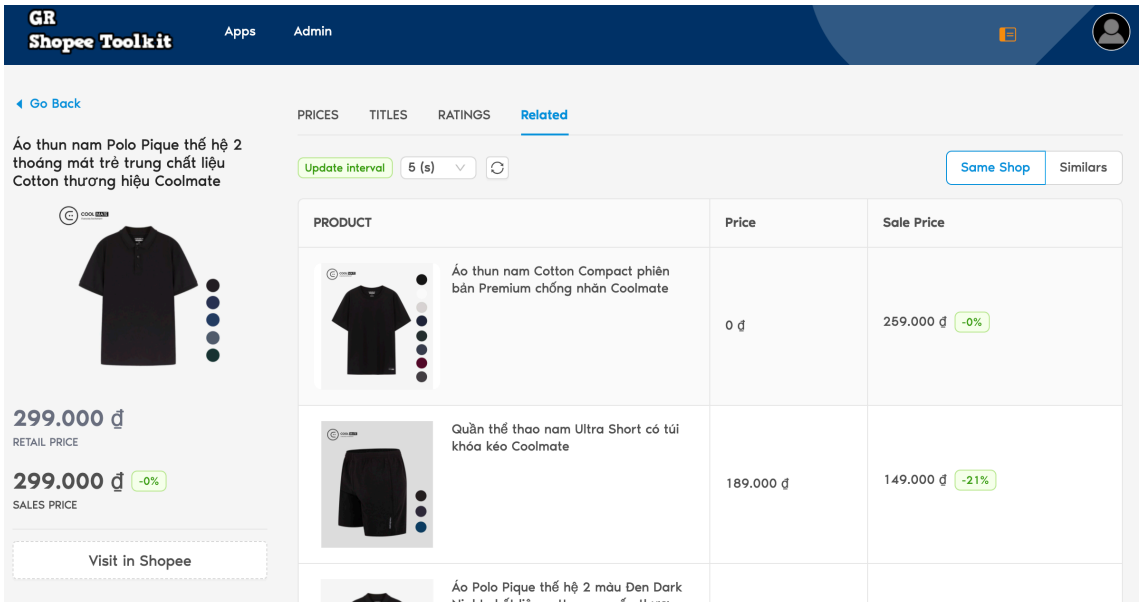


Figure 5.9: UI for related products data display

About product review analytic screen, seller can observe some statistic about product on the left side and the detail analytic of reviews on the right side. Seller can filter the review by categories such as star or tags of the review.

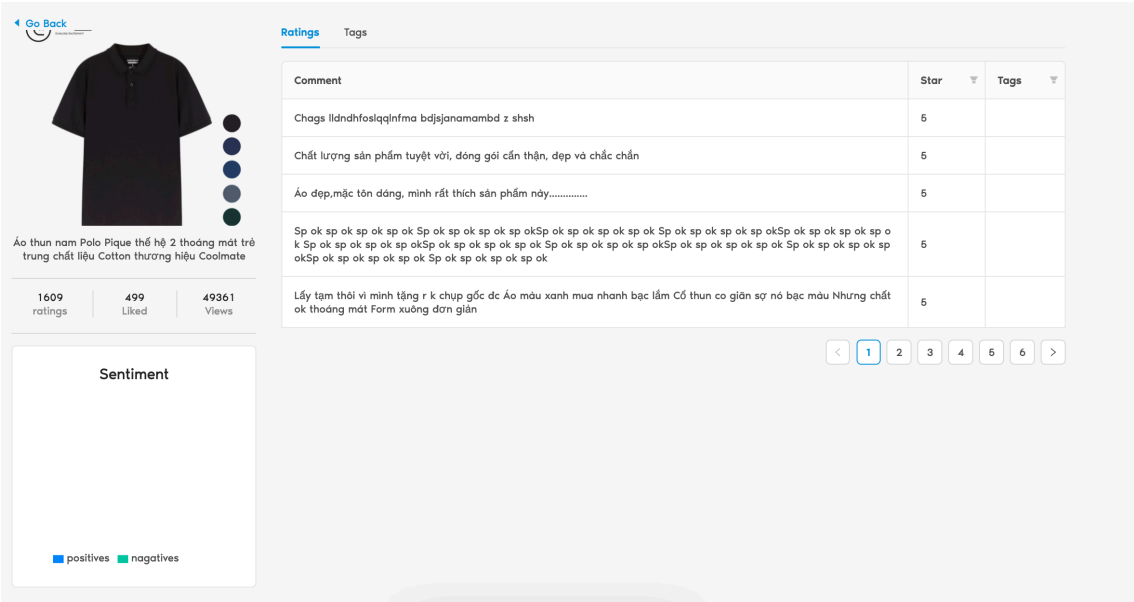


Figure 5.10: UI for product review analytic data display

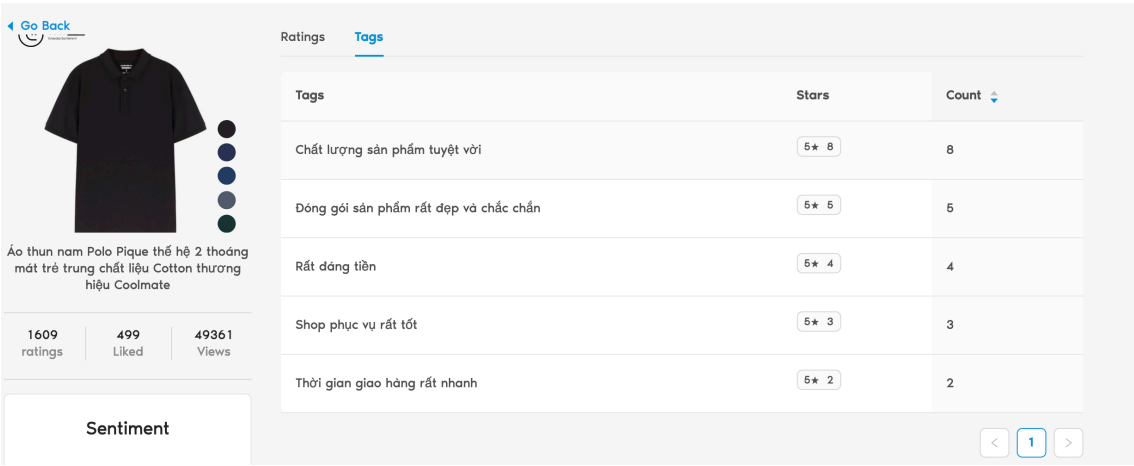


Figure 5.11: UI for product review tags analytic data display

In shop tracking screen, I organize the layout similar with the product tracking screen with the general information of shop in the left side and the monitoring data in the other side. About the tracking shop data, the application let seller monitor statistic about competitor shop includes active vouchers, followers, product quantity and seller ratings.

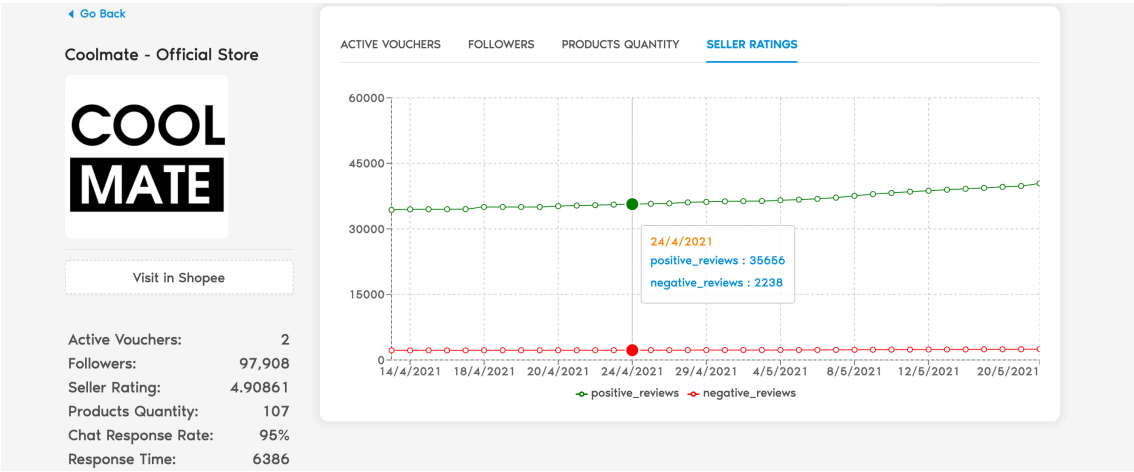


Figure 5.12: UI for shop actives vouchers tracking data display

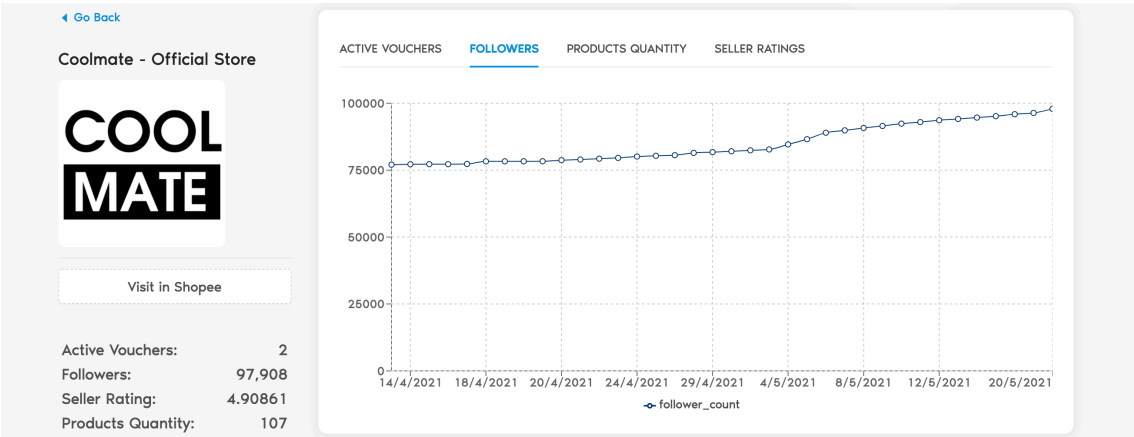


Figure 5.13: UI for shop followers tracking data display

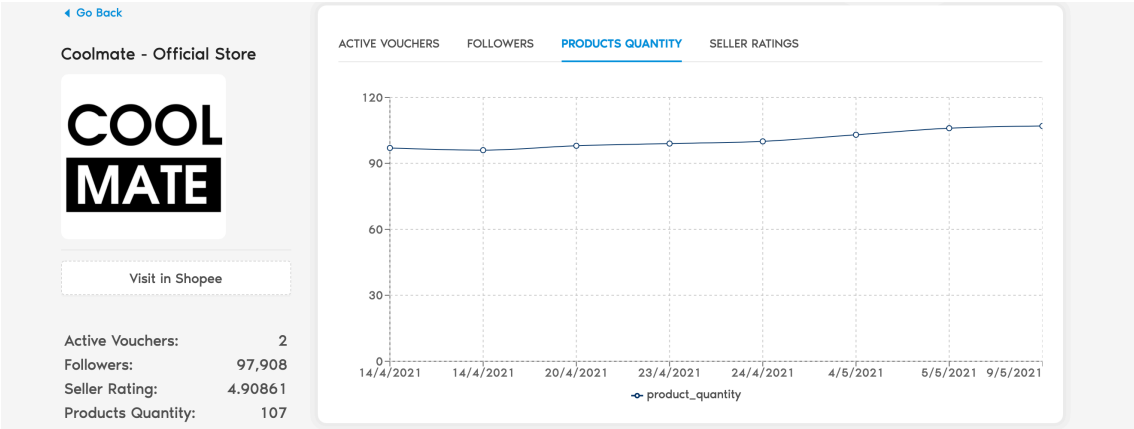


Figure 5.14: UI for shop product quantity tracking data display

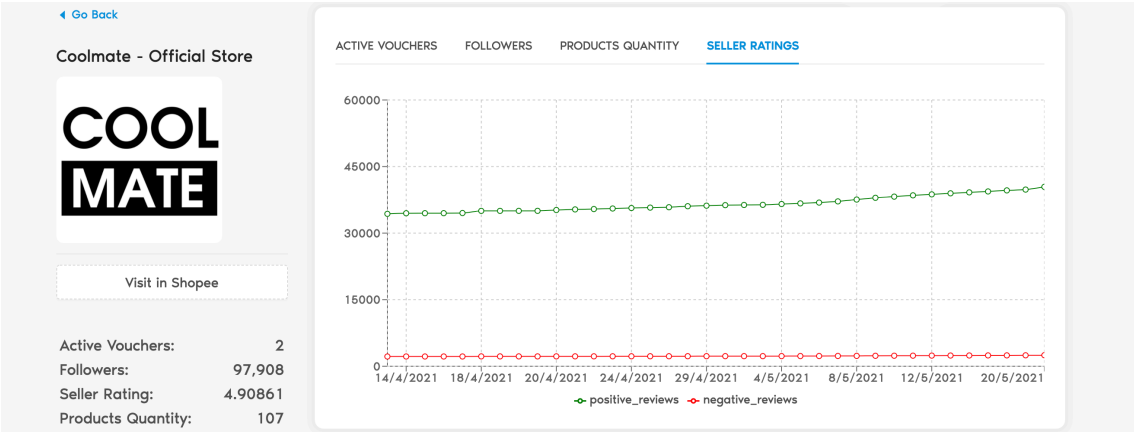


Figure 5.15: UI for shop ratings tracking data display

## 5.2 Microservice architecture

### 5.2.1 Problem

Nowadays, when it come to build an web application, it is a traditional model for software in which the structure is a single and indivisible unit. A monolith has one code base with multiple modules. The architecture has its advantages such that ease of development and monitoring, less latency and less complication about crosscutting concerns. However, like any solution, monolithic architecture has its drawbacks. Here's a list of some disadvantages. First, a large code base can be significantly harder to understand. Next, Every element is closely related and dependent on the others, so it is difficult to change to a new or advanced technology, language, or framework. In additions, Updating can be a challenge as it redeployment of the application.

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable, organized around business capabilities. The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

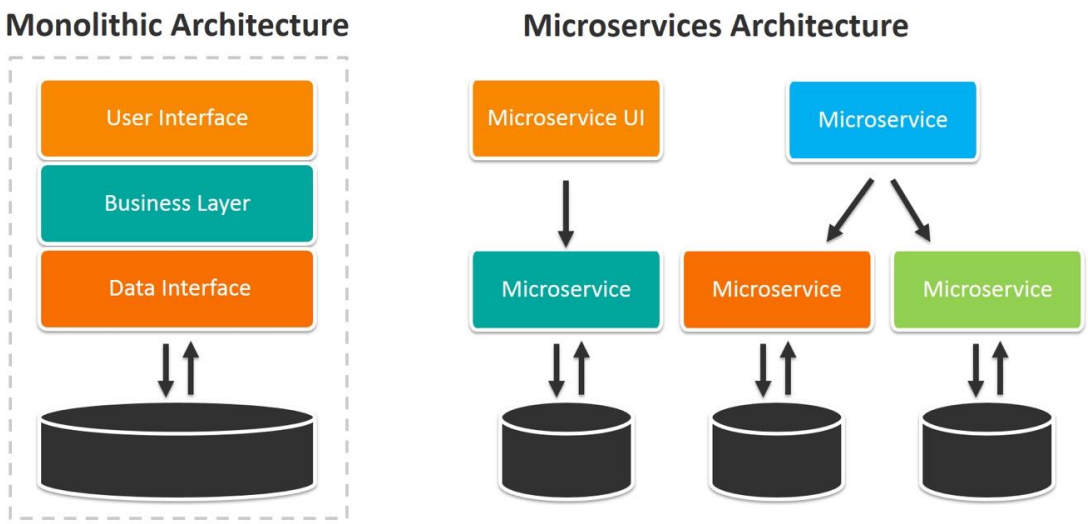


Figure 5.16: Monolithic architecture vs Microservice architecture

Because of above reasons, I decided to build the competitor monitoring application following the microservice architecture. The application will split into many services. Each service are built around business capabilities and independently deployable by fully automated deployment machinery.

### 5.2.2 Solution

Microservices should be designed around business capabilities, not horizontal layers such as data access or messaging. In addition, they should have loose coupling and high functional cohesion. Microservices are loosely coupled if you can change one service without requiring other services to be updated at the same time. A microservice is cohesive if it has a single, well-defined purpose, such as managing user accounts or serve the user interface. A service should encapsulate domain knowledge and abstract that knowledge from clients. For example, a client should be able to sent a notification without knowing the details of how the notification is managed.

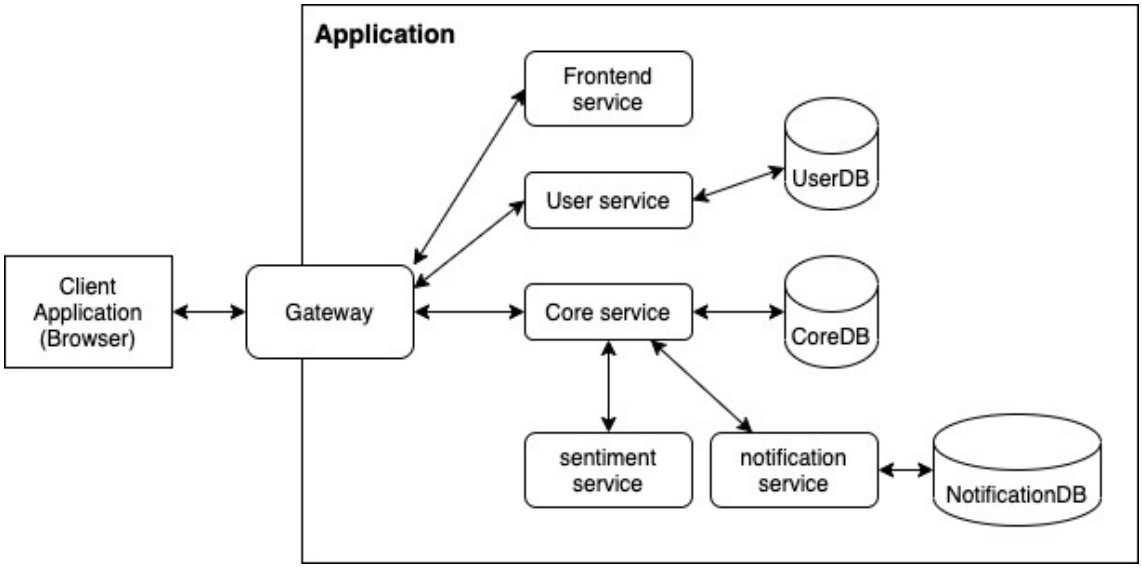


Figure 5.17: General architecture

In my application, there are five main services includes frontend service, user service, core service, sentiment service and notification service. As a practice when design micro-service system, I try to minimize the stateful service. Each service has its own database or volume if it is stateful and they are designed to run independently on different ports. We have only one endpoint. So that, all the traffic want to access our services need to go through a gateway. The gateway service then modify requests (if necessary) and forward those to target services. Almost services communicate with each other in Synchronous protocol (HTTP) and I choose JSON as for data exchange format.

## 5.3 CI/CD Pipelines for Microservices

### 5.3.1 Problem

In a traditional monolithic application, there is a single build pipeline whose output is the application executable. All development work feeds into this pipeline. If a high-priority bug is found, a fix must be published, which can delay the release of new features. You can mitigate these problems by having well-factored modules and using feature branches to minimize the impact of code changes. But as the application grows more complex, and more features are added, the release process for a monolith tends to become more brittle and likely to break.

Following the microservices philosophy, the changes for a new feature or a fix can be made in many services. The team that builds service "A" can release an update at any time, without waiting for changes in service "B" to be merged, deployed. Therefore, to achieve a high release velocity, your release pipeline must be automated and highly reliable to minimize risk. If you release to production one or more times daily , regressions or service disruptions must be rare. At the same time, if a bad update does get deployed, you must have a reliable way to quickly roll back or roll forward to a previous version of a service.

CI/CD, standing for Continuous Integration and Continuous Delivery or Development, is a development paradigm that is created to solve the above problem. CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository. The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline.

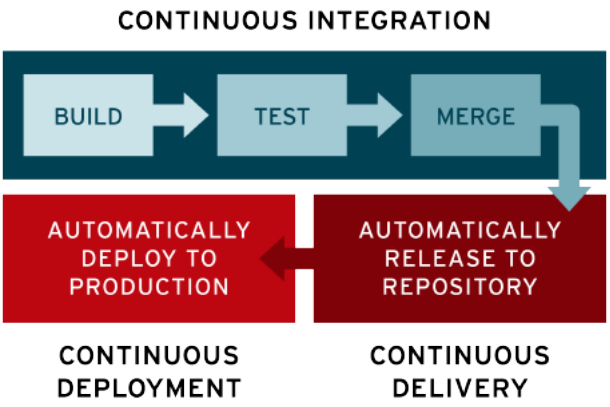


Figure 5.18: CI/CD flow

However, building an effective CI/CD pipeline can be a complex process with countless decisions that require a great deal of planning. You need to create or find a suitable tool to add to your development toolchain.

### 5.3.2 Solution

After research about some tools to configure and perform the entire CI/CD process such as Jenkins, Github actions, Gitlab CI, Travis CI and so on. I decided to choose Github actions because github actions enables me to create custom software development lifecycle workflows directly in my Github repository and it's easy to use and setup.

A typical CI/CD workflow of a service in my application deployed using github actions look like this

First, I need to dockerizing the existing service. I need to write a Dockerfile, an optional .dockerignore file and a [service name].compose.yml. Below is a Dockerfile and compose.yml for the core service.

```
FROM golang:1.16-alpine
WORKDIR /app
RUN apk --no-cache add ca-certificates
COPY src .
RUN go mod download
RUN CGO_ENABLED=0 GOOS=linux go build -ldflags "-w -s" -o appbin /app/cmd/apps/main.go

FROM scratch
WORKDIR /app
COPY --from=0 /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/
COPY --from=0 /app/appbin /app/appbin
COPY --from=0 /app/static /app/static
CMD ["/app/appbin"]
```

Figure 5.19: Core Service Dockerfile

```yaml
version: "3.8"

services:
  insight_coresvc:
    image: "daongocthanh/gr-coresvc:${IMAGE_TAG}"
    environment:
      - "RABBITMQ_URI=${RABBITMQ_URI}"
      - "MONGO_URI=${MONGO_URI}"
      - "MONGO_DB=${MONGO_DB}"
      - "QUEUE_UPDATE_ITEM=${QUEUE_UPDATE_ITEM}"
      - "QUEUE_UPDATE_ITEM_RATINGS=${QUEUE_UPDATE_ITEM_RATINGS}"
      - "GIN_MODE=release"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.coresvc.rule=Host(`api.insight.smartecommerce.tech`)"
      - "traefik.http.services.coresvc.loadbalancer.server.port=8000"
    links:
      - "fluentd"
    logging:
      driver: "fluentd"
      options:
        tag: "insight.core"
```

Figure 5.20: Core Service core.compose.yml file

Next, for github actions to work, I need to create a .github/workflows folder. I need to create the pipeline for service inside this folder. As a convention, I create cd.yml file for every services. Here is what we want from our workflow:

On every push, perform these actions in the given order

- Build the docker image for service and tag it a new version
- Login to docker hub
- Push the image to docker hub
- SSH to host machine to deploy the new version of service by applying service_name.compose.yml file

This is how my cd.yml for core service would look like (figure 5.21). Let's dissect this file. In line 3-6, I want to trigger our pipeline when someone pushes the code to repository on staging branch. Line 14, this is step 1 of my application. Here I want to get my source code. We can write my own code to pull my source code or reuse an open source. Line 16-21, I want to set the current time to the time output to use later. Line 23-28, I build a docker image of my code with the help of docker action, tag the image with new version and then push it to docker hub. In line 30-38, I want to copy the coresvc.compose.yml file from github repository to the folder /home/gr/gr in the host machine. And finally in line 40-55, I ssh to the host machine and run a bunch of command to deploy a new version for this core service.

```yaml
name: cd

on:
  push:
    branches:
      - staging

jobs:
  push:
    runs-on: ubuntu-latest
    if: github.event_name == 'push'

    steps:
      - uses: actions/checkout@v2

      - name: Get current time
        uses: 1466587594/get-current-time@v1
        id: current-time
        with:
          format: DD-M-YYYY-HH-MM-SS
          utcOffset: "+07:00"

      - name: Build && Push to Dockerhub
        uses: elgohr/Publish-Docker-Github-Action@master
        with:
          name: daongocthanh/gr-coresvc:${{ steps.current-time.outputs.format}}
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Prepare deploy antifacts
        uses: appleboy/scp-action@master
        with:
          host: ${{ secrets.VPS_HOST }}
          username: ${{ secrets.VPS_USERNAME }}
          key: ${{ secrets.VPS_SSH_KEY }}
          source: "coresvc.compose.yml"
          target: "/home/gr/gr"
          overwrite: 1

      - name: Deploy
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.VPS_HOST }}
          username: ${{ secrets.VPS_USERNAME }}
          key: ${{ secrets.VPS_SSH_KEY }}
          script: |
            cd /home/gr/gr
            export IMAGE_TAG=${{ steps.current-time.outputs.formattedTime }}
            export RABBITMQ_URI=${{ RABBITMQ_URI }}
            export MONGO_URI=${{ MONGO_URI }}
            export MONGO_DB=${{ MONGO_DB }}
            export QUEUE_UPDATE_ITEM=${{ QUEUE_UPDATE_ITEM }}
            export QUEUE_UPDATE_ITEM_RATINGS=${{ QUEUE_UPDATE_ITEM_RATINGS }}
            docker container ls | grep coresvc && docker stop insight_coresvc
            docker-compose -f coresvc.compose.yml up -d insight_coresvc
```

Figure 5.21: Core Service cd.yml file

*I have not failed. I've just found 10,000 ways that won't work.*

Nikola Tesla

# Chapter 6

# Conclusions and Future Works

I have presented about the web-based tool for support buyer in ecommerce platform (shopee) to monitoring their competitors. The system has been applied with the microservice architecture, using the ReactJS library and NextJs framework for UI building. All services that communicate with each other using the Restful API, which is the simple HTTP protocols that all common programming languages support with Golang and Python for the back-end. With the use of API application programming interfaces, the modules in the system are clearly separated, so it is easy to upgrade, repair or replace functions.

The project has reached some good results after a semester of learning new technologies, analyzing the requirement of the system as well as the desire of end user. The tool has the pretty good interface, user friendly and good user experiences. The website is built with web responsive so it is able to be used on different types of monitors. The system has been deployed to product environment, which means it is used by real user.

However, there are some disadvantages that I hope to improve in the near future that is the sentiment analysis feature of my application is still quite simple with basic information. In near future, I want to improve the sentiment analytic model to support more type of analysis such that Emotion detection or aspect-based. In addition, I also want to extend the crawler to support more ecommerce platform such that Tiki or Lazada.

# Bibliography

[1] M. Fowler. Cqrs. [Online]. Available: https://martinfowler.com/bliki/CQRS.html

[2] V. Inc. About - vercel. [Online]. Available: https://vercel.com/about

[3] R. C. Martin. The clean architecture. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[4] webpack. About - webpack. [Online]. Available: https://webpack.js.org

# Appendix A

# Test cases