# System Implementation

## Architecture

## Product Search and Detail Display

## Image Storage and Handling

## Password Security

## Purchase Order Processing

## Customers’ Ratings and Reviews

## Concurrency Control

**Why We Choose Spring Boot for the Backend**

In school, we have learned two different programming languages: Python and Java. And we even study Django as backend framework which is based on Python. After looking through the Internet, we find out Spring Boot is also a good choice for us since it is a server-side Java framework.

Before implementation, we make comparison between these two frameworks in order to choose the most suitable one. Frist of all, in terms of performance, we found Spring Boot is faster in running the code as it is written in java comparing to Django. Since Google recommends fast websites and it also affects a website’s ranking, this is one of the most crucial factors we choose Spring Boot.

Secondly, we found that Django could only handle one request at once. However, Spring Boot is able to handle multiple requests at once. Using Spring Boot as backend will shorten the waiting time of users. In other words, it will promote the brilliant experience of users.

What’s more, as we know, Django and Spring Boot are both open-source technologies. According to statistics, Spring Boot has 25.8K GitHub forks and 39.8K GitHub stars, while Django has 42,000 ratings and 18,000 forks. From above, Spring Boot seems more popular than Django, which means we could get more reference and help benefiting from this. (Moreover, the responsible person is more familiar with Java.)

So, these are the reasons why we choose Spring Boot as the framework of our backend.

Spring Boot is ideal for building web applications due to its streamlined setup and auto-configuration features, which simplify the development process and enable developers to quickly build and deploy scalable and maintainable web applications.

For microservices, Spring Boot's modular architecture and lightweight design make it an excellent choice. It allows for the creation of independent microservices that can be easily scaled up or down to meet changing demands.

Finally, Spring Boot is also commonly used for batch processing, which involves processing large amounts of data at once. Its support for multiple data sources, job scheduling, and retry capabilities make it an ideal tool for this type of application.

**Pagination**

Pagination, also known as paging, is the process of dividing a document into discrete pages.

<https://en.wikipedia.org/wiki/Pagination>

Advantages:

The client can cache the entire dataset, and pagination can be implemented without the need for any server-side processing. This can significantly reduce the number of requests sent to the server, which can improve performance.

Since the client is responsible for managing the pagination, the user experience can be improved, as there is no delay between page changes.

Disadvantages:

If the dataset is too large, client-side pagination may not be practical, as it can result

We choose to apply client-side pagination due to the small size of the dataset. The performance is more important as the mobile app user interface requires high responsiveness.

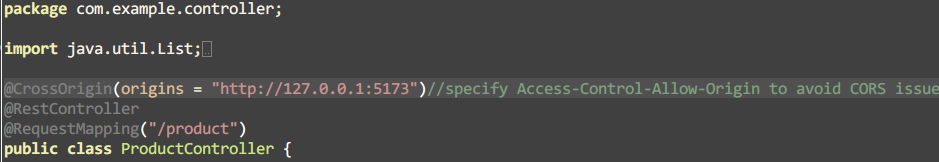
Our solution is the hybrid of both server-side and client-side pagination.

For the product list page, we use server-side pagination. Because the mass products may consume much bandwidth. Only few items are displayed at once in server-side pagination mode, and as the change of pages, the data of products are retrieved on demand.

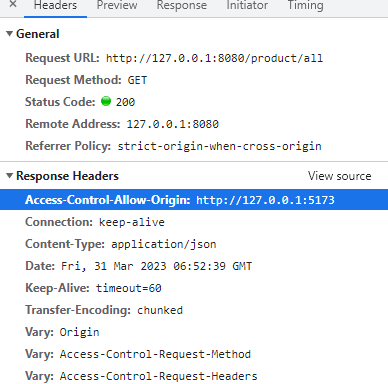
For the search page, we assume much smaller cardinality of data set is returned. Another assumption is performance is highly required when the user is searching and filtering by brand simultaneously. Therefore, we decide to apply client-side pagination in the search page. The entire list of searching results is returned first, the paging and filtering are done by client-side program subsequently. It is believed that using client-side pagination here increases the responsiveness without costing too much data traffic.

**CORS**

Two different servers



*ProductController*

****

According to Spring official documentation, this @CrossOrigin annotation enables cross-origin resource sharing only for this specific method.

<https://spring.io/guides/gs/rest-service-cors/>



*without adding the control header*

**Search Performance (instantly display result)**

Although typing one more word will send another request, which consumes a lot more data traffic than sending the request till the user tabs “Search” button, we still want to keep the function due to high performance and availability. We reckon when the user is searching instead of browsing, there must be some intention or aim. Therefore, the sooner the product is found or result is displayed, the sooner requirements or intention is satisfied or resolved. Such functionality greatly reduces the probability of causing anxiety when a customer cannot find a certain item in a short period of time.

**Fuzzy Search**

It’s a common situation that users sometimes type a misspelt word, either deliberately or unconsciously.

Moreover, a user may not know the exact product name he/she wants. The user has some keywords or

“Xiaomi Nova” flexibility is provided

*Levenshtein distance* is exploited for evaluating the similarity between two strings.

In addition, product names are decomposed for more accurate searching by keywords.

However, the algorithm is inaccurate to some extent given a short keyword. Instead of providing fewer but more accurate results, we decided to return more but slightly inaccurate results. For example, “aple” (misspelt “apple”), “xao” (misspelt “xiao”)

This is where availability and effectiveness conflict. To compromise,

**Personalized Recommendation**

[**https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine**](https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine)

The algorithm is based on *Jaccard index*

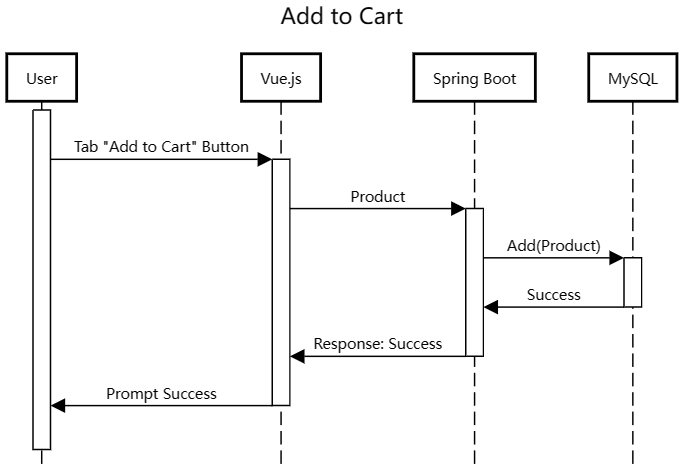
similarity index of two users

recommendation engine algorithm

The top 3 products that a customer may like according to the probability calculated from formula 2.

It is assumed that a customer may like the product which is favored by other highly similar customers.

A like or dislike is a direct index which determines whether a customer likes a product or not. However, we still want to estimate how likely a customer would prefer a product without existing feedback from that customer. We are inclined to provide several recommendations to the customer … So that increase the likelihood for the customer to purchase a product.

****