Contents

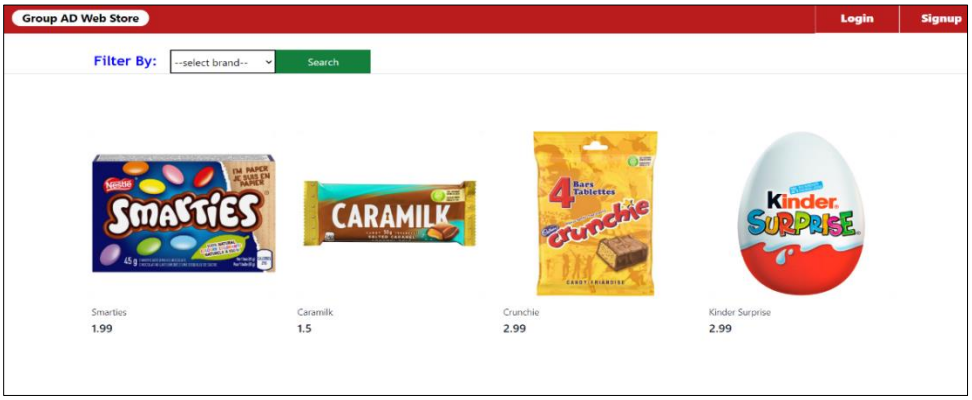
Introduction

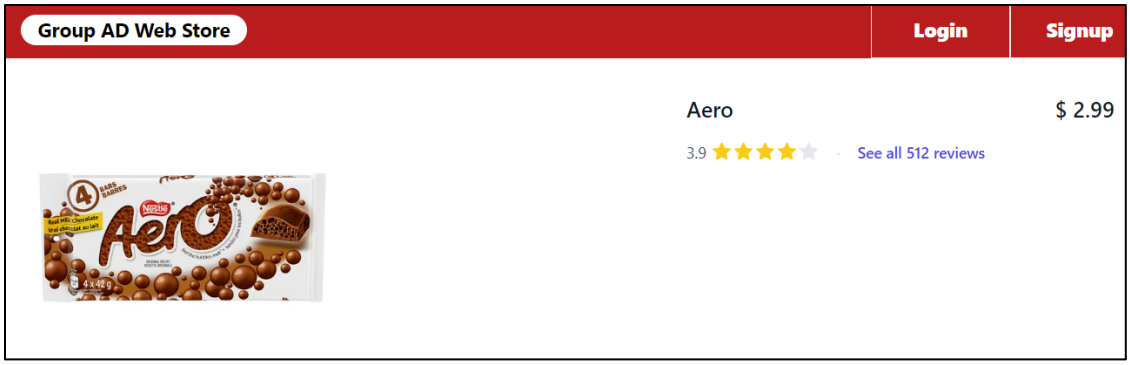
This project is E-commerce System called Snack bar. It provides online shopping services for various snacks. It has been developed using React, Spring Boot, MySQL, Vercel and Azure cloud service. MVC, Repository and Singleton pattern has been used for the system and Spring Boot Security framework has been applied for security.

Application access information

Link to website: https://web-store-alpha.vercel.app/

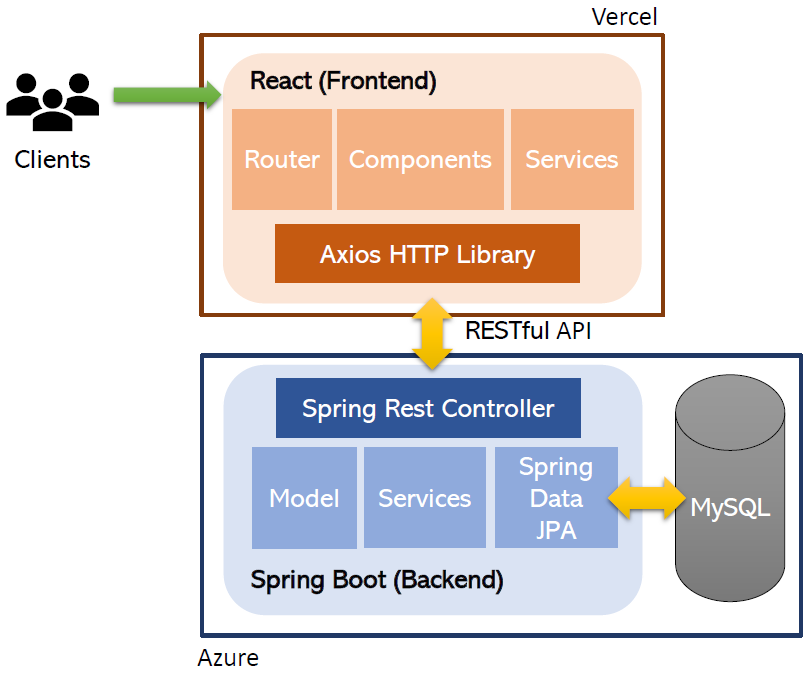
[Link to API: https://web-store-api-store-spring-api.azuremicroservices.io (/products is one basic endpoint)
Link to gitfront (gives access to private repo by link): https://gitfront.io/r/user-2256353/EEByoR8UWDoJ/web-store/ (If that link doesn't work: https://github.com/4413GroupAD/web-store Michelangelo will approve for access)
For access to Cloud DB, Email Jamo125@my.yorku.ca (Michelangelo), you need to be IP whitelisted + given credentials
NOTE: -There is one user with admin access, email: admin@google.com, pass: StrongPassword1!
you can also add one manually via the API, regular users can be created through the signup page.
-If cart or other page is not showing immediately, press refresh button please.
-If signing in doesn't work well with valid email and password after signing up, press login button a few times more please. It sometimes happens.]





Architecture Description

Overall architecture & System components



The frontend project is based on React project. It consists of routers, some web pages and services. It interacts with users using web pages and communicates to the backend using the Axios HTTP library. The backend project is based on Spring Boot in the Maven package. Model(entity) classes, Service classes and Spring JPA repository interfaces are included. To interact with database MySQL, it utilizes the Spring JPA repository. Spring Rest Controller is used to communicate with the frontend. For public website hosting, we used Azure server for the backend API and Vercel hosting service for the frontend side.

Use cases - Sequence diagrams

Scenario1 - Original use case

Name: Buying products through 'Snack Bar' website.
Actors: User who wants to buy products.
Description: The user comes to 'Snack Bar' website using URL address and look some products using catalog page. Signup or login if user already have account for the website. They click their cart to put their order. After checking the products, they proceed checkout. To receive products, they put their shipping address. To pay, they put billing address and credit card information. After checking for card's validation, the order is submitted to server.
Precondition: The website is currently being hosted in the given URL. Products on the website are not out of stock to be added to the cart. (Some products can be out of stock.) The user has a valid email address to sign up. The user has a valid credit card and addresses for payment.

Standard Procedure:

1. The user opens the ‘Snack bar’ website using the given URL.
2. The user clicks the login button to log in.
3. The user puts valid account information.
4. The user clicks home button to see products.
5. The user clicks the image of products that they want to buy.
6. The user adds products to their cart on the product detail page.
7. The user clicks the cart button to proceed the order.
8. The user clicks checkout button to proceed the order.
9. The user puts the shipping address, credit card numbers, expired month/year of credit card to pay.
10. The user clicks confirm order button to finish the order.

Extension:

3.1 If the use does not have account
    3.1.1 The user clicks create account button.
    3.1.2 The user put valid email, password and name.
        3.1.2.1 If the email or password is not valid
            3.1.2.1.1 The website shows error message
    3.1.3 The website shows sign up success message
3.2 If the use put invalid account information
    3.2.1 The website shows error message
10.1 If the information on a credit card is not valid
    10.1.1 The website shows the error message.

Scenario2 – admin use case

Name: Seeing website’s report on the website through the admin account.
Actors: Owner of the business who has admin authority.
Description: The admin comes to website using URL and logs in with admin account. The admin clicks several reports to see record of selling/visiting history of the website.
Precondition: The website is currently being hosted in the given URL. The owner has an admin account of the business in advance.

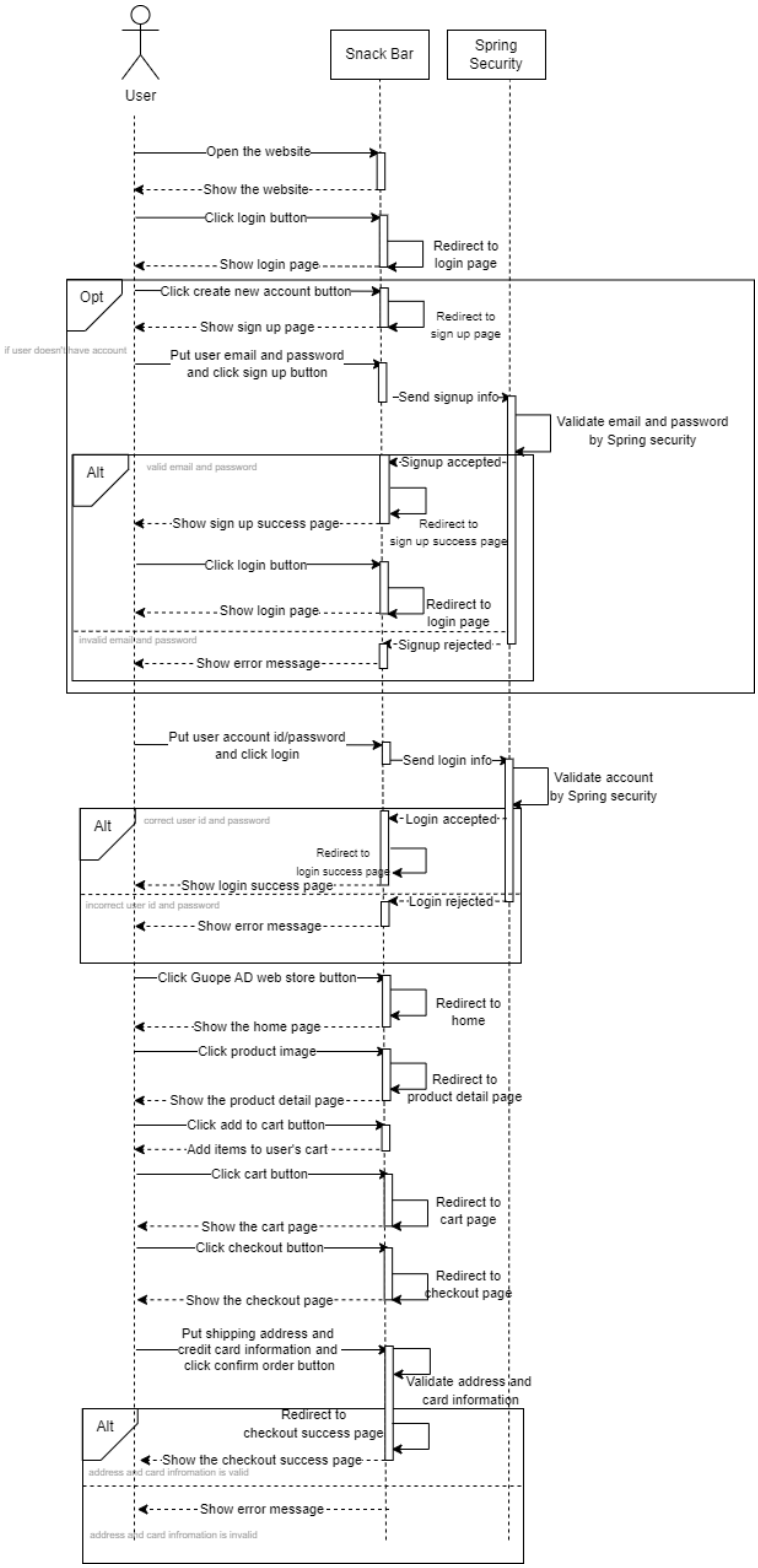
Standard Procedure:

1. The admin opens the ‘Snack bar’ website using the given URL.
2. The admin clicks the ‘login’ button.
3. The admin put admin account information to login.
4. The admin clicks the analytics button on the top bar.
5. The admin sees analytics reports.

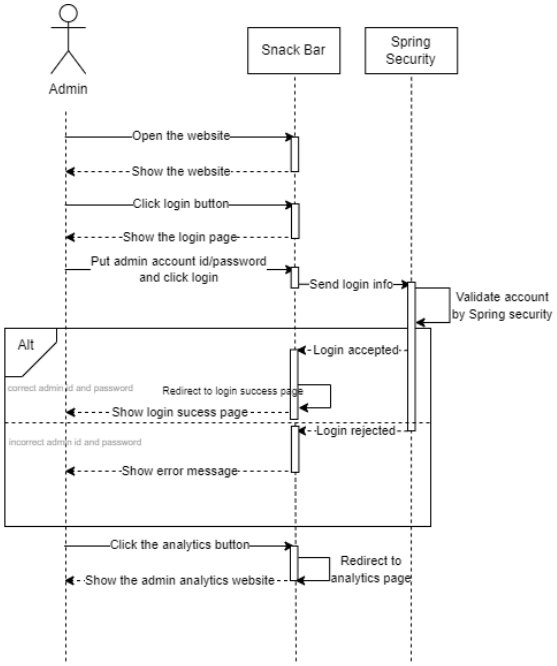
Extension:

3.1 If the email or password is not valid:
    3.1.1 The website shows error message.

3.2 If the email or password is not admin role:
    3.2.2 The ‘analytics’ button to go to the admin page doesn’t come up.



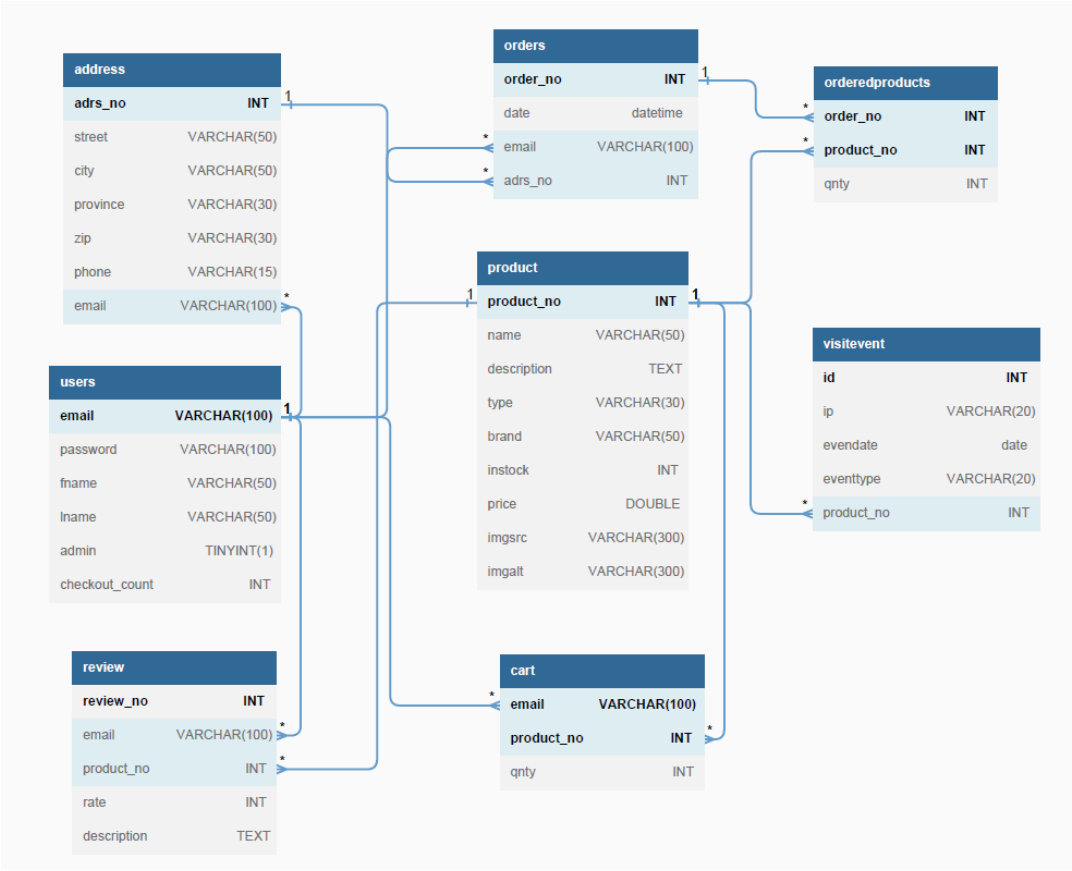
[Sequence diagram of Scenario 1]



[Sequence diagram of Scenario 2]

Design Description

Database Schema



Design Patterns

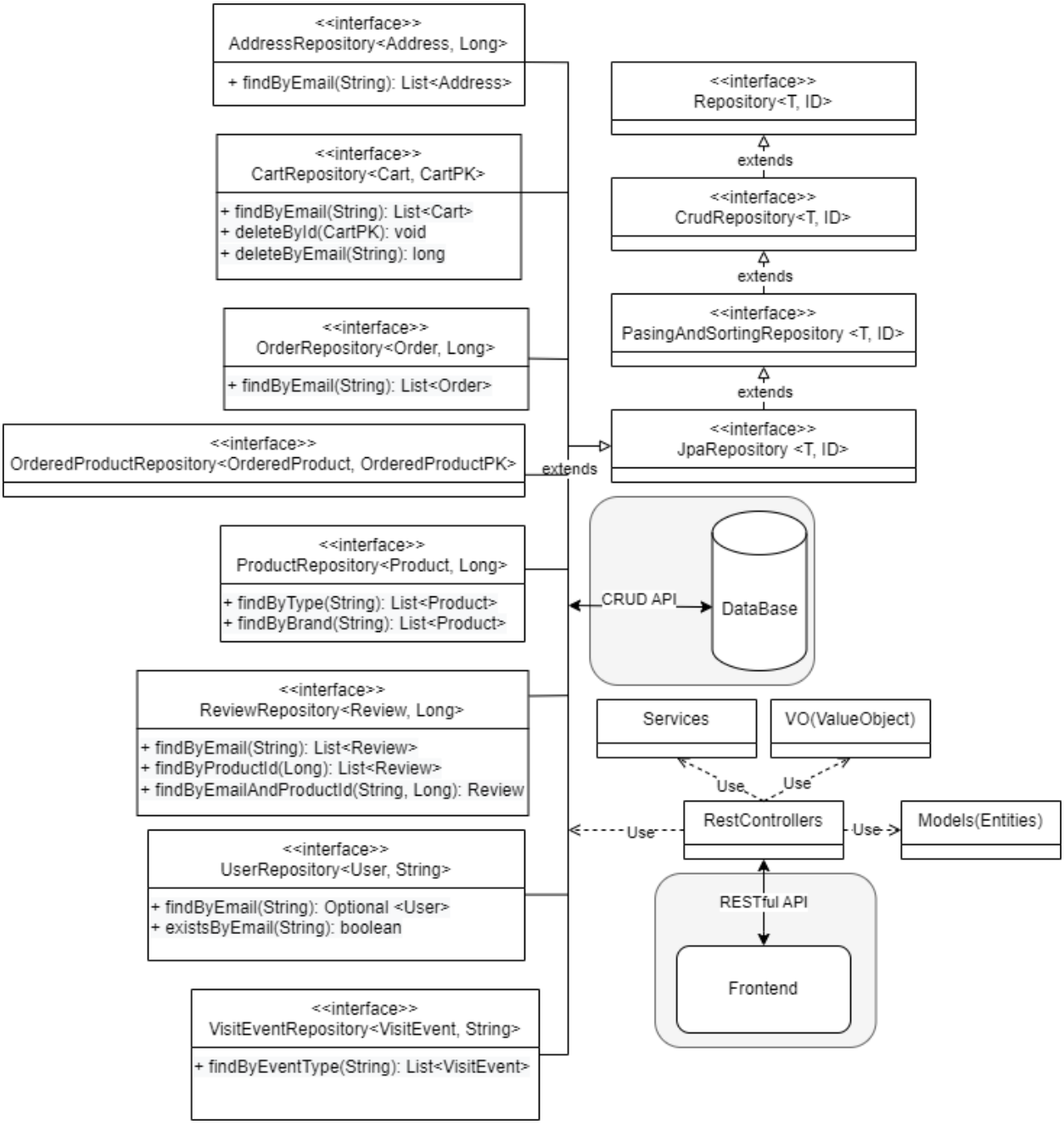
Patterns: MVC, Singleton, Repository Pattern

Main design decision: To separate user interfaces, data and controller logic, we decided to implement an MVC pattern. The interfaces and backend side interacts using RESTful API. For robust security, we decided to use Spring Boot for the backend application and applied the Spring Security framework. As a result of using Spring Boot, we could apply the Singleton pattern using Spring Boot Annotations. Rest controller, Bean, Repository and so on have been applied and they are used with @Autowired annotation. Also, we could easily apply the Repository pattern using Spring JpaRepository interfaces. Since JpaRepository extends CrudRepository and we have model components that act as entities, we could design our application using CRUD API to interact with Database.

Trade-offs: Since we decided to use Spring boot, we could utilize Repository and Singleton patterns, simply. Also, we don't need to separate components to interact with the frontend and database because controllers can interact with both sides through RESTful API and CRUD API using the repository. However, since Spring Boot tends to be a little stricter and more complicated than Nodejs we considered

as alternatives, we had to spend lots of time creating and setting up classes. For example, the entity class should follow the camel case naming rule, not using the underbar, and MySQL database should follow the underbar naming rule for Hibernate mapping. If they are not followed, we can't utilize the advantage of JpaRepository.

Class diagram for one component – repository classes



Implementation

Describe implementation decision & trade-offs

We don’t allow unsigned user to put items to cart. When they want to add products into their cart, we require login. It can make users feel a little bit inconvenient. But, we can get data from the user by forcing login for analytics. All validation checks proceeded on the backend side. By doing it, the frontend doesn’t need to care about validation check logic. When we handle the products’ quantity, we subtract the items when the users add them to their cart. Then, it doesn’t make any conflicts when they check out. But, if the items are not proceeded and remaining in the cart, it’s not available by other users.

Functionality test

- [x] TestCase1: Showing product lists at home.
- [x] TestCase2: Filter products by brand.
- [x] TestCase3: Login in login page.
- [x] TestCase4: Signup in signup page.
- [x] TestCase5: Showing product detail page.
- [x] TestCase6: Adding product into the cart.
- [x] TestCase7: Adding review in the cart page.
- [x] TestCase8: Placing order with valid credit card.

Limitation

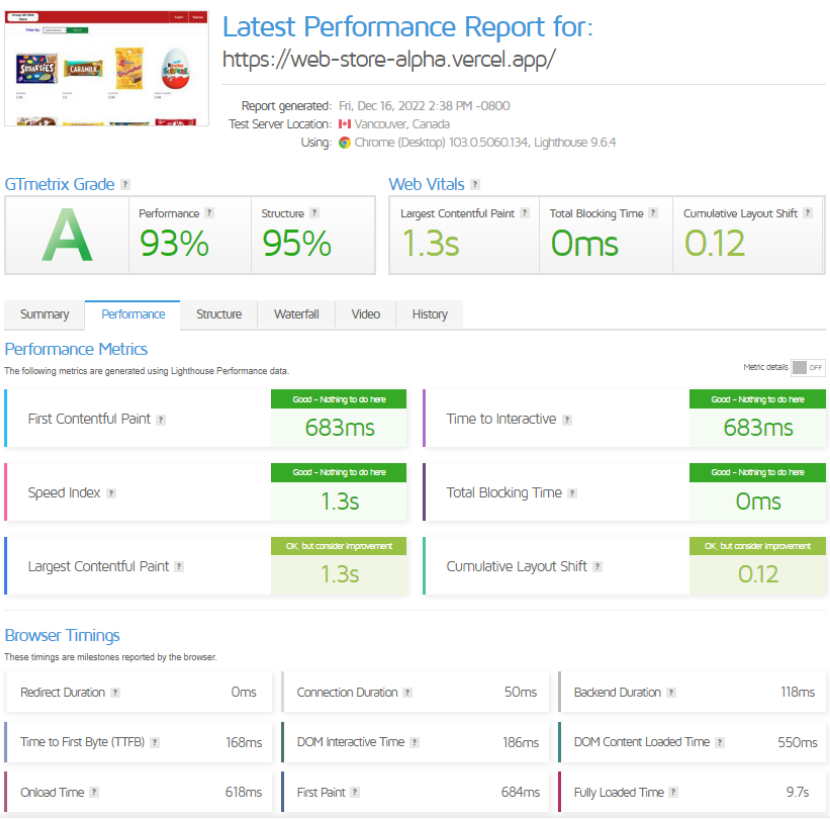
Regarding the test, there are some limitations. First, adding item button is available only in product detail page. So, user can’t add products on the home page. For the review part, we show every reviews, not individually based on each product, and we can see the popup to write reviews when we click write a review button but, actually, the backend implementation is not available now. So, we can only see the frontend implementation now for the review part. The website tends not to response immediately, especially in cart page. So, refresh button is needed to show the right data. Also, saving data such as created account’s data tends to be saved slowly, as well. So, we should wait a little bit to check signing up functionality. If you try login and signing up again after creating account immediately, it may show the wrong results.

Security Report

Spring Security Framework: We used Spring Security Framework to handle security of the project. When users sign up, out website encrypts their password using this framework and save as encrypted version into database, not exposing their real passwords. Spring Security in our project separates the role

(user/admin) and distinguishes users. We can see analytics button with admin account and the button is not available in users' accounts. Spring Security automatically catches users approaching to website without proper authority and force them to put their account information to provide authorization. We tested that cart and check out pages require login to unknown users. Except for these fundamental functionalities, other implementation for security was not applied. For example, there is no prevention for viruses, frauds, SQL injection and so on. So, tests for them were not conducted.

Performance Report by GTmetrix



Show Audits Relevant to: All | FCP | LCP | TBT | CLS

IMPACT	AUDIT	
Med	Properly size images	Potential savings of 730KB
Med-Low	Serve static assets with an efficient cache policy	Potential savings of 160KB
Low	Avoid large layout shifts CLS	5 elements found
Low	Reduce unused JavaScript LCP	Potential savings of 68.8KB
Low	Serve images in next-gen formats	Potential savings of 190KB
Low	Avoid enormous network payloads LCP	Total size was 1.02MB
Low	Avoid long main-thread tasks TBT	1 long task found
Low	Reduce JavaScript execution time TBT	82ms spent executing JavaScript
Low	Reduce initial server response time FCP LCP	Root document took 117ms
Low	Avoid serving legacy JavaScript to modern browsers TBT	Potential savings of 51B
Low	Avoid chaining critical requests FCP LCP	2 chains found

Conclusion

We built an e-commerce system using several frameworks based on RESTful API and CRUD API that we learned in the lectures. We could develop our understanding of APIs and get experience implementing them directly. Also, we could get advanced experience in not only Java, HTML, CSS, JS but also frameworks that are widely used in real software projects. By implementing design patterns, we could understand software designs more deeply and figure out the pros and cons of design patterns that we chose with some tests. In addition, we now understand the fundamental mechanism of the e-commerce system with public website hosting. As a result, we expect that we can utilize and apply these skills in real projects.