LetsGrocery: Online Shopping Website

Chenhao Wei, Jiahao Wang, Yue Wan, Wenxin Han, Jiaming Li
Master of Computer Science, University of Ottawa
CSI 5112 - Software Engineering

1 Introduction

Our website is called "LetsGrocery" and the target audiences are the merchant, as well as administrator, who will provide products of different categories (they can add, delete or edit the information of all products), check customers' orders, and reply to customers' questions, and customers, who are able to view the products of different categories, selected the items that they would like to buy and add them to the shopping cart, then check them out. Also, they are able to manage their account, they can set their information like address, view their previous orders, and ask and answer any questions in the chatbox. Our online shopping website consists of the following key pages: User Registration and Login Page, Category Page, Shopping Cart Page, Checkout Page, Customer Profile Page, Merchant Profile Page.

2 Database Functionalities

2.1 Which indexes did you create in Mongo? Why?

In our database "shop", we have four collections storing all data that we need.

The first collection is "product" which contains all information about products. For each product, we have an auto-generated ObjectId, a ProductName, a Category which is the category this product belongs to, a price, an inventory, a quantity which represents the quantity that users buy and is always reset to 1 after each customer checkout, a product description, and an image path.

The second collection is "order" which stores orders of all users. It has a unique Id, a userId showing which user it belongs to, an orderDate which is auto-generated by Date() when user places an order, a totalPrice which is calculated using unitprice and quantities, an orderAddress which is the address when checking out, a string of products, a string of unitPrice and a string of quantities. We meet some difficulties in serializing and deserializing arrays, so we use "," to separate each product in the string of products, unitPrice, and quantities. When the merchant or user wants to review or print order details, we use split() to convert the strings to array and generate order details.

The third collection is "user" which is a set of all users. It has an Id, username, password, phoneNumber, and address. The Id is auto-generated, however, the username and password are required when the user registered. They can change the password, and initialize and modify the phone numbers and the addresses on the Setting page, but the username must stay unique and unchanged. Since encryption and decryption of the passwords are not what we focus on in this course, passwords are stored as plain text so far. We know we should hash and store them in the database and check the hashed string when users log in, but we prefer focusing on the overall functionality of this shopping web, so we did not implement the encryption and decryption functions.

The fourth collection is "chatbox" which has all questions and answers. It has an Id, a question string, a reply string, and a postdate. When a user posts a question, the question is added to our database and the reply string is set as an empty string. If someone answers (maybe a merchant or a customer), we use HTTP PUT to update the reply string.

We have to special note that we do not have a category collection. Our TA deducted points in phase two because of this, but after careful thought by all of our team members, we decided to remain the same because we really do not need an extra collection to store the category. There are two situations in that we need to use category. The first is on the category page where users can view all products by category. The second is on the merchant dashboard where the merchant can view all products and modify them (add new products, change inventory, and delete products by name and by category). Those two situations all require getting a full list of products from the database to show products. We use the category value stored in each product to generate a category selector for users and merchants and it is working properly.

2.2 Are all CRUD operations working?

All CRUD operations work well if we need them.
For product:

- Create: merchant can create products (HTTP POST).
- Read: on the merchant dashboard page and category page, products can be shown properly using data from the database (HTTP GET).
- Update: merchant can change the inventory of a product (HTTP PUT).
- Delete: merchant can delete a product by its name or delete products by category (HTTP DELETE).

For order:

- Create: every time a user places an order, an order record is added to the database.
- Read: merchant can see all orders by all users. Each user can only see their own orders.
- Update: once an order is placed, users and the merchant are not able to update.
- Delete: users and the merchant are not able to delete an order as well.

For user:

- Create: a new user record is added to database when a user registers a new account.
- Read: when a user logs in, we need to read data from database to verify their username.
- Update: users can update their password, phone number and address after they log in.
- Delete: We do not support account deactivation at this moment.

For chatbox:

- Create: only customers can post new questions.
- Read: the merchant and customers can see a list of all questions and their replies.
- Update: customers and the merchant can add replies no matter the question has not been answered yet or not.
- Delete: users and the merchant are not able to delete a question.

3 Privacy

When a user inputs their username and password, HTTP GET is used trying to get a user record from the database. If the return value is not null (which means the user exists), and the username and password both match, the user logs in successfully. The user's personal data is stored and we use the username to select their history order records from the database. This way we can make sure that the user logged in can only see orders that belong to them.

As stated in the database part, we did not implement password encryption and decryption. The passwords are stored in the database as plain texts, and this is not safe. They should be stored as hashed values and each time a user tries to log in, the input password will be compared with the stored value, returns true if they match and the user logs in successfully. Only after the user logs in successfully, their history order can be retrieved from the database.

Since our project back-end connects to MongoDB and MongoDB requires a username and password to locate a specific database. Previously, our group stored the username and password in the form of a connection string in the back-end. Therefore, if we push our project to GitHub publicly, everyone can see our connection string and get our username and password which indicates that we might face the safety hazard. Based on this situation, our group encrypted the connection string so that others cannot get our username and password. We define the key and value in AWS, then specified the path in the parameter store. In order to actively fetch data from the database and then connect to the service. Our group set the key and value (which includes username and password) in the environment variable on AWS, so that people who access the AWS backstage can see the connection string. After the modification, we pushed the back-end code to GitHub, since only those who can access the AWS back-end can see the connection string, thus ensuring the privacy of our project.

4 Challenges

4.1 Challenges of working with Front-end

Flutter's innovative approach to UI website development brings the best out of native and cross-platform development. However, this combination can feel forced on a team who did not work with similar frameworks before like us. The first issue to deal with is actually learning Dart. Our team gradually gets used to particular Widget settings, editing, modes, managing controllers, and storyboards. So, our team was adopting Flutter has been ready to face certain difficulties in the process and has already overcome those difficulties.

4.1.1 Stateless and Stateful Widgets

While we were working on the first phase, we did not pay much attention to define which page should use stateless widgets and which need to use stateful widgets as long as the page can navigate to each other and that would be fine. Then, when we were trying to keep doing our second phase, we realized that the state of widgets are important since it can determine whether the data can be transferred. So our team spent long time redefined each page and fix them.

Take our Shopping Cart Page as an example, it contains leftview and rightview as the figure shown below. In the left side, we can see that users can add, edit, or remove the product and the price will change based on the quantity. Since we do not design a database for shopping cart, all the action will be done in front-end, then when the user checks out, the order information will be stored in our database. Therefore, there is no need to transfer data on the left side, and it is reasonable

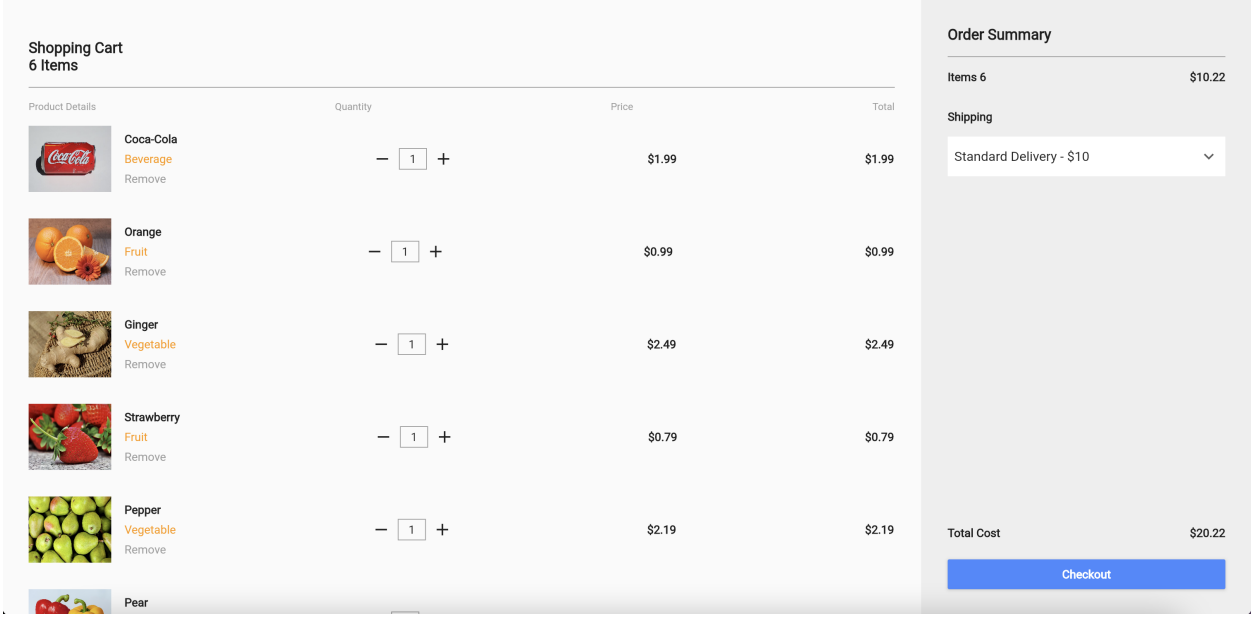

Figure 1: Shopping Cart Page

to use stateless widgets. However, for the right side, it contains checkout button which is used for saving customer's order to the database, it required data transferring. So, we have stateful widgets for the rightview.

4.1.2 Sidebar Navigating

Based on what we designed for the category page, we use side bar to display all categories on the left and when click on a category, it will show the products in this category on the right as the Figure 2 shows. Our group create a function called "generateCategorySelector", which contains a for loop to walk through all the categories and generate a list. So the category leftview shows the list that we get and if you select different categories, the right part will show you the products of the chosen category.

4.1.3 Making HTTP Request by Using JSON

While we were connecting our front-end and back-end, the challenge we were faced was making HTTP request to transmit data in JSON format. We need to represent different data models in JSON data format and send them to the back-end by POST or PUT. Moreover, we also adopted deserialization technique to map response of data into entities from the back-end.

4.1.4 Improving Data Storage and Representation Format

During the transition from Phase1 to Phase2, we needed to construct proper data models as templates to represent our data and enable our application to provide actual and various information from the back-end and more flexible. The challenge was to replace all the hard-coded part with predefined data model in an appropriate manner in our project and it required carefulness and awareness to every step in the transaction to make sure we match the data entities between front-end and back-end and did not miss out anything hard-coded or outdated.

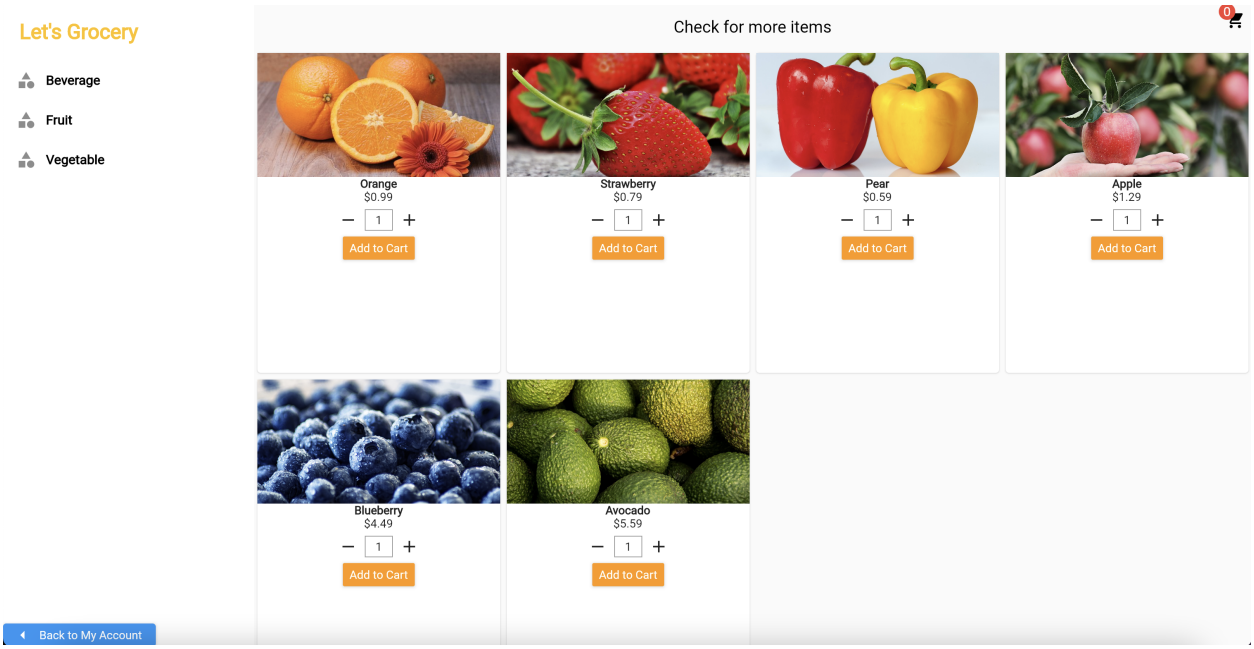

Figure 2: Category Page

4.2 Challenges of working with Back-end

4.2.1 Determine the Function of Each Layer

The MVC design pattern of ASP.Net structure was difficult to understand and we need to use such abstract idea to construct encapsulated web APIs for which the front-end application can call. Before establishing the actual communication between them, the responsibilities of each layer on the back-end should be determined including which to manipulate the database, which to define the data model and which to regulate the web APIs format, etc.

4.2.2 RESTful API Design

First we had to identify the needs for http requests in each module that can fit our expectations for the desired features. And correspondingly, we needed consult about the parameters required from each specific API in order to realize the functions.

4.3 Challenges of Deploying the Project

4.3.1 Connection using SSH

In AWS deployment, our team meet some challenges. The first issue is connection instance using an SSH client on Mac. After installing the SSH client successfully, we found the connection always failed. It is an important step to connect local computer and AWS platform.

4.3.2 Deploy failure in EC2 instance

After we created Docker file successfully, we could not build Docker images to utilize. We require a managed service for building docker images, which can automatically produce new up-to-date container images and publish them to AWS repositories.

4.3.3 IP address changed every time

The IP address always change each time when we launch instance because a dynamic IP address is default in AWS. It is difficult for us to connect the database. Therefore, we require a static IP address.

4.3.4 Cannot use Https

For our website, we cannot use Https because of without domain name. We require to pay if we need unique domain name. Currently, our website uses AWS automatic http URL instead of https URL.

5 Resolutions

5.1 Using Stateless and Stateful Widgets

When the application is running, stateless widgets always have the same stage, independent of hardware condition or user actions. Stateless widgets are used to provide reliable functionality that is not dependent on user input. The functionality does not require anything other than the execution of predetermined setups.

Stateful widgets respond to user input and can modify their state in response to actions taken by the user. Even when the application is running, the components in the widget change. They may react quickly to the actions of users, resulting in a responsive application.

So, a stateless widget is used for stable functionality like app icons, immutable graphic elements, and logos. A stateful widget works for anything that requires input such as registration form, menu, pop-up, payment page, etc.

5.2 HTTP Methods

There are four main HTTP methods that our team uses in the project and we test them based on Postman. Here is what each of them does:

- GET: Requests a representation of the specified resource. Requests using GET only retrieve data and they should have no other effect on the data.
- POST: Submits data to the specified resource. Our team uses this method to send data to the server, such as customer orders or product information.
- DELETE: Deletes the products or categories.
- PUT: Replaces all current representations of the target resource with the uploaded content.

5.3 AWS Deployment

For solve the issue we describe above, the first step is we watched the lecture videos very carefully and tried to find the professor's instruction to solve them. Fortunately, most of challenges are mentioned in the lecture video and we could follow the video step by step to resolve them. We also research some content via the Internet.

For IP address changes, our team create an elastic IP to associate with the current IP to maintain the EIP address all the time.

6 Conclusion

From what we did for this project, we learned how to use flutter to develop a website, how to use HTTP to transfer data, how to design and create data models, how the front-end and back-end connect to each other. We also improve our understanding of database connection and master the knowledge of Cloud Deployment. Besides, we realized the importance of teamwork and work distribution since five of us focus on different parts of project and we noticed that cooperation and collaboration are significant factors building a successful project.