Newark Fulfillment Centers

- CS631 Project Report

Group: Weixin Tang & Dairui Zhang

1.Introduction

The requirement for our project was to design web-based applications for Newark Fulfillment Centers (hereinafter referred to as NFC). NFC provides B2B warehouse storage services for local small businesses. This means that these seller customers can store their products in the NFC warehouse. These products can be PPE, or pet supplies, clothes, and so on. At the same time, these seller customers can set the goods they store in the warehouse to be sold, and individual consumer users can buy them through NFC shopping sites.

The goal of the project assignment was to create web-based applications that include warehouse management, customer inventory management, customer portal management, and end user website.

Our group first discussed and analyzed the requirements document and jointly designed the database; then we divided the project into layers, discussed the division of labor, and finally completed their respective work. In the whole process, the division of labor among the team members was clear and the cooperation was smooth. At the same time, we also learnt new knowledge from each other.

The URLs are:

- https://seller.cs631njit.me/signup
- https://buyer.cs631njit.me/login
- https://manager.cs631njit.me/login

(* The links to access our project are hosted on Zhang's laptop. If you want to test the website, please contact us with dz9@njit.edu or wt2@njit.edu)

2. Database Deigns

In the part of database design, as we submitted in milestone, we designed the ER diagram and made ER mapping. However, in actual development, the database design has been simplified. Because we want to be as streamlined as possible so that we can develop faster.

2.1 ER-diagram in milestone

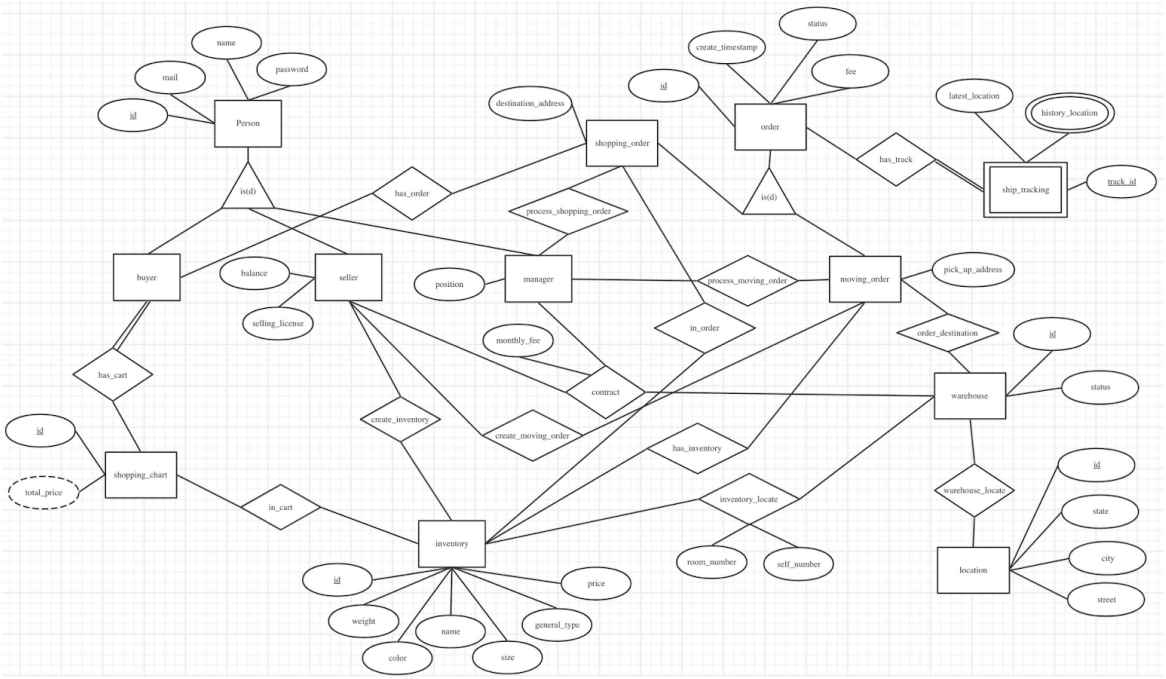


fig. ER-diagram

2.2 Database in actual development

Blow is the final version of our database. It includes 10 tables: Person, Warehouse, Order, Inventory, Order_Inventory, Order_Tracking, Location, Invoice, Cart, and Config.

```
-- ----------------------------
-- Table structure for Cart
-- ----------------------------
DROP TABLE IF EXISTS `Cart`;
CREATE TABLE `Cart` (
  `person_id` char(64) NOT NULL,
  `inventory_id` char(64) NOT NULL,
  `time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`inventory_id`,`person_id`)
USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Config
-- ----------------------------
DROP TABLE IF EXISTS `Config`;
CREATE TABLE `Config` (
  `id` char(64) NOT NULL,
  `value` double DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Location
-- ----------------------------
DROP TABLE IF EXISTS `Location`;
CREATE TABLE `Location` (
  `id` char(64) NOT NULL,
  `inventory_id` char(64) DEFAULT NULL,
  `warehouse_name` char(64) DEFAULT NULL,
  `room_number` int(11) DEFAULT NULL,
  `shelf_number` int(11) DEFAULT NULL,
  `length` double DEFAULT NULL,
  `width` double DEFAULT NULL,
  `height` double DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `layer_key`
(`warehouse_name`,`room_number`,`shelf_numb
er`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Inventory
-- ----------------------------
DROP TABLE IF EXISTS `Inventory`;
CREATE TABLE `Inventory` (
  `id` char(64) NOT NULL,
  `name` char(64) DEFAULT NULL,
  `type` char(32) DEFAULT NULL,
  `status` char(32) DEFAULT NULL,
  `length` double DEFAULT NULL,
  `width` double DEFAULT NULL,
  `height` double DEFAULT NULL,
  `weight` double DEFAULT NULL,
  `price` double DEFAULT NULL,
  `warehouse_id` char(64) DEFAULT NULL,
  `seller_id` char(64) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Invoice
-- ----------------------------
DROP TABLE IF EXISTS `Invoice`;
CREATE TABLE `Invoice` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `description` char(255) DEFAULT NULL,
  `employee_id` char(64) DEFAULT NULL,
  `seller_id` char(64) DEFAULT NULL,
  `money` int(11) DEFAULT NULL,
  `history_balance` double(11,0) DEFAULT
NULL,
  `time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=15
DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Order
-- ----------------------------
DROP TABLE IF EXISTS `Order`;
CREATE TABLE `Order` (
  `order_id` char(64) NOT NULL,
  `user_id` char(64) DEFAULT NULL,
  `order_type` char(32) DEFAULT NULL,
  `payment_method` char(32) DEFAULT NULL,
  `manager_id` char(64) DEFAULT NULL,
  `order_status` char(64) DEFAULT NULL,
  `fee` double DEFAULT NULL,
  `address` text,
  `time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Person
-- ----------------------------
DROP TABLE IF EXISTS `Person`;
CREATE TABLE `Person` (
  `person_id` char(64) NOT NULL,
  `type` char(32) DEFAULT NULL,
  `position` char(32) DEFAULT NULL,
  `balance` double(11,0) DEFAULT '0',
  `mail` char(64) DEFAULT NULL,
  `name` char(64) DEFAULT NULL,
  `password` char(64) DEFAULT NULL,
  PRIMARY KEY (`person_id`),
  UNIQUE KEY `layer_key` (`mail`,`type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Order_Inventory
-- ----------------------------
DROP TABLE IF EXISTS `Order_Inventory`;
CREATE TABLE `Order_Inventory` (
  `order_id` char(64) NOT NULL,
  `inventory_id` char(64) NOT NULL,
  PRIMARY KEY (`order_id`,`inventory_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Order_Tracking
-- ----------------------------
DROP TABLE IF EXISTS `Order_Tracking`;
CREATE TABLE `Order_Tracking` (
  `order_id` char(64) NOT NULL,
  `location` char(64) DEFAULT NULL,
  `time` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`order_id`,`time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- ----------------------------
-- Table structure for Warehouse
-- ----------------------------
DROP TABLE IF EXISTS `Warehouse`;
CREATE TABLE `Warehouse` (
  `name` char(64) NOT NULL,
  `status` char(10) DEFAULT NULL,
  PRIMARY KEY (`name`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This version of the database is significantly simpler than that of milestone. There are two main reasons. First, considering that the development of our group is in the form of remote cooperation, if the database is more complex, some small changes may lead to changes in many entities or relationships, which is not conducive to synchronous development; second, because the simplified design is more convenient for us to use during programming, and it does not need to span too many tables for data operation.

3. Mock up

3.1 Project structure

We combined the application requirements to analyze and designed this project into three layers:

- Management layer
- Seller layer
- Buyer layer.

The management layer is for managers of NFC company. In this layer of website, users (that is, managers of NFC company) can view, edit and send orders to customers, warehouses, inventory, employees and orders.

The seller layer is for the small businesses customers. In this layer, users (i.e. customers of small business) can submit store orders, edit or choose to sell their products, and manage their customers.

Finally, the buyer layer is for the end users. In this layer, individual consumers can search and buy their favorite products on the website, add shopping cart, submit orders and other operations.

For these three layers of web-based application, our group decided to co-operate based on division of labour. Because we are familiar with different programming languages, the first and third layers use nodejs express framework for development, and the second layer uses Python's Flask framework for development. Although we work in different languages, the results are very harmonious. The three-tier application uses the same database. In the final test, there is no conflict in the system. Each layer of website can correctly receive the results of database changes made by other layers.

3.2 wireframes

We made some mock-up to help us imagine what the website should look like. There are some wireframes or frontend pages. We also designed the logo for NFC since we want to make this brand more authentic.

fig.The mock-ups

4. Website Achievement

4.1 Management Layer

The management layer is for managers of NFC. Every manager can sign up and sign in to this application and start their management. After sign-in, your position and your name will show in the upper right corner. In this management system, you can easily manage customers, inventory, order, warehouse, and check all the employees of NFC company.

When your seller customer changed their name or Email address, you can change the information for them. You can also create a new customer who decided to use services provided by NFC.

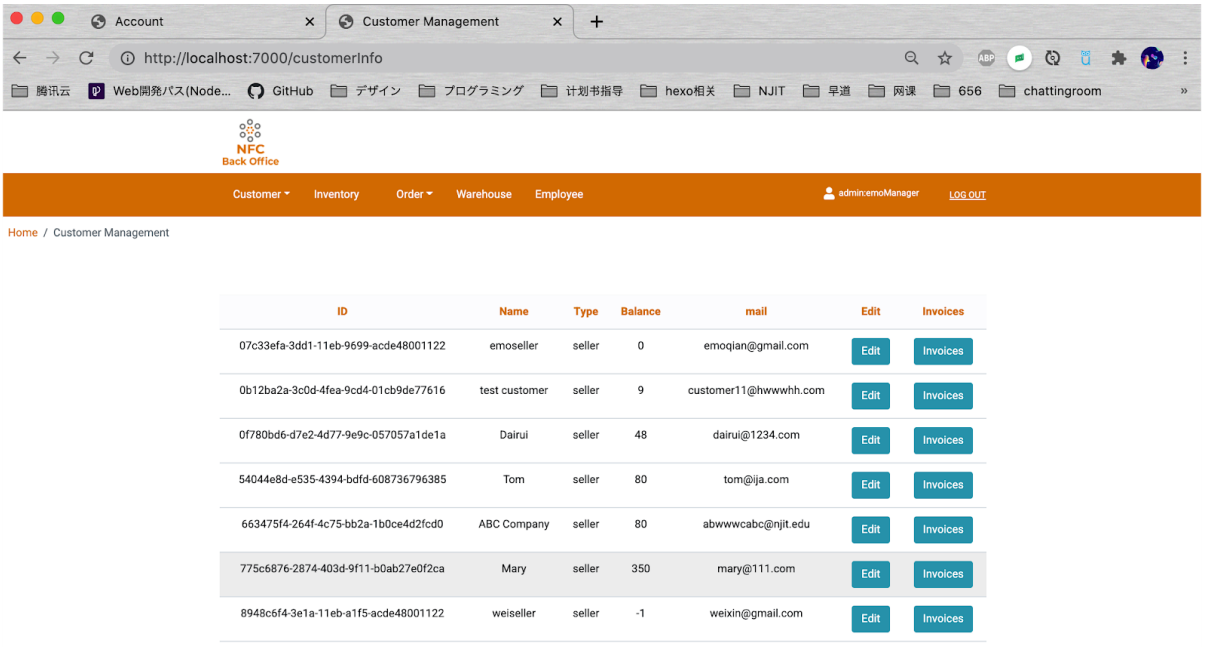


fig. Management layer-customer management page

When your seller customer submits a new storage order, you can see the new order on customer order management page. All the details will be shown and you can check what inventory they want to put into the warehouse by clicking the detail button.

Since the inventories have 4 kinds of status: restore, store, sell, and sold. All the new inventories' status will be restored. As a manager, you need to click the store button to put it on a suitable shelf for your customer. So by clicking this button, the system will calculate the size of this inventory and put it in right place automatically. You can see the store succeed alert then. After storing all the inventories of this order, you can refresh this order's status from created to complete. If the order has already been completed, the refresh status button will not be shown.

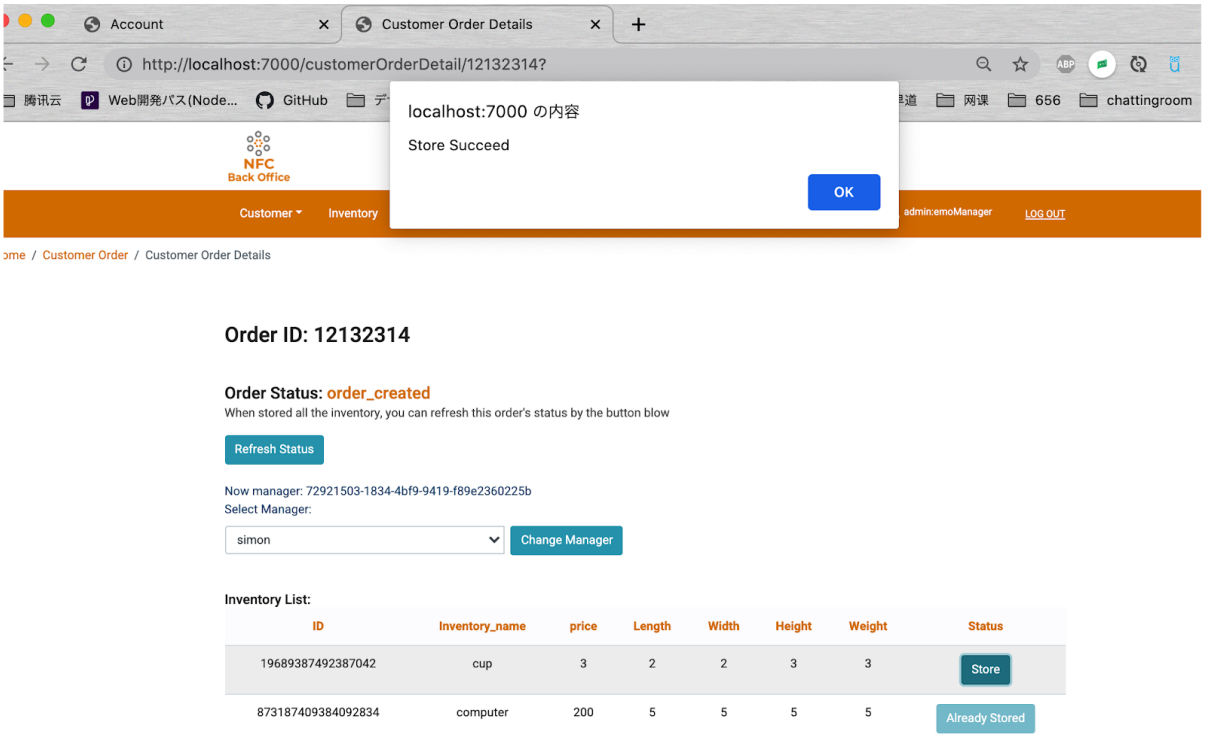


fig. Store the inventory

By the way, you can also check the end-users' orders.

Certainly, these services will not be free. As a manager, you need to send invoices to your customer. You can create invoices here by input amount and some description.

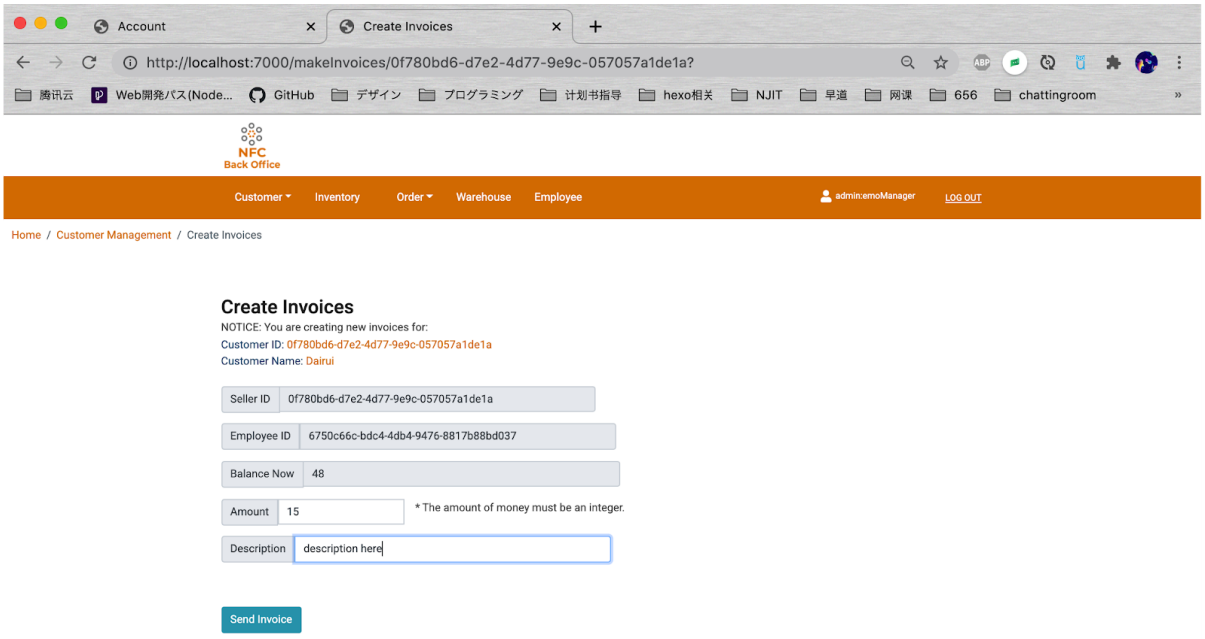


fig. Send the invoices

Then, you can manage NFC's warehouse and all the locations of this warehouse. To empty the location, you just need to edit location information and delete the inventory ID. Besides, you can check all the inventory on the inventory management page, and you can only filter inventories in a specific state. For example, to see all the inventories that are on selling or all the inventories have not been stored.

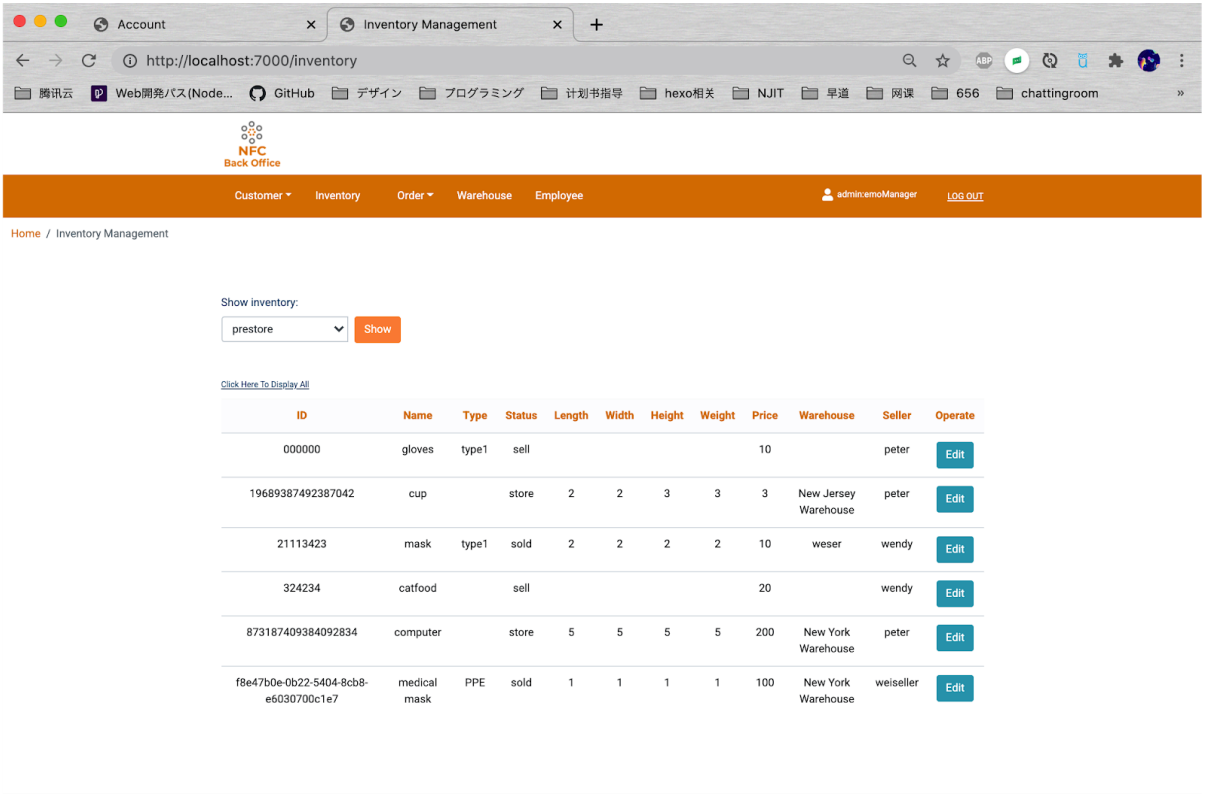
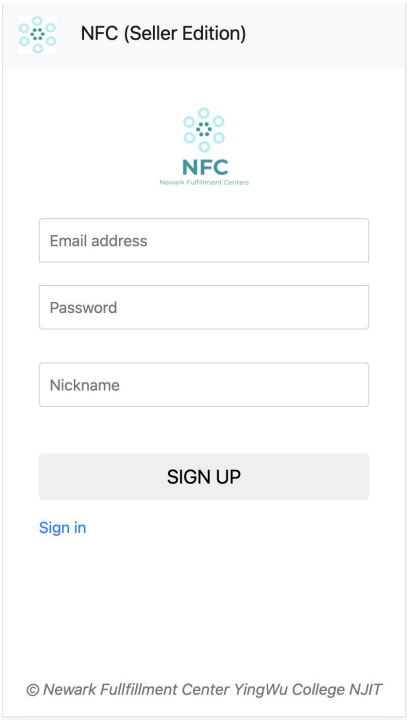


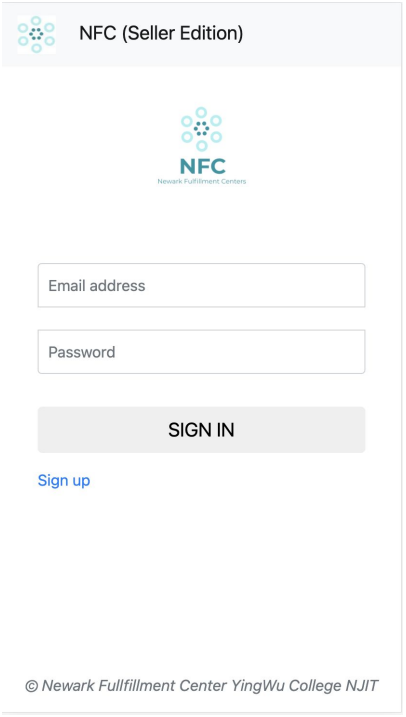
fig. inventory management

When the customer wants you to edit the price or the name of some inventories, you can change them on this page.

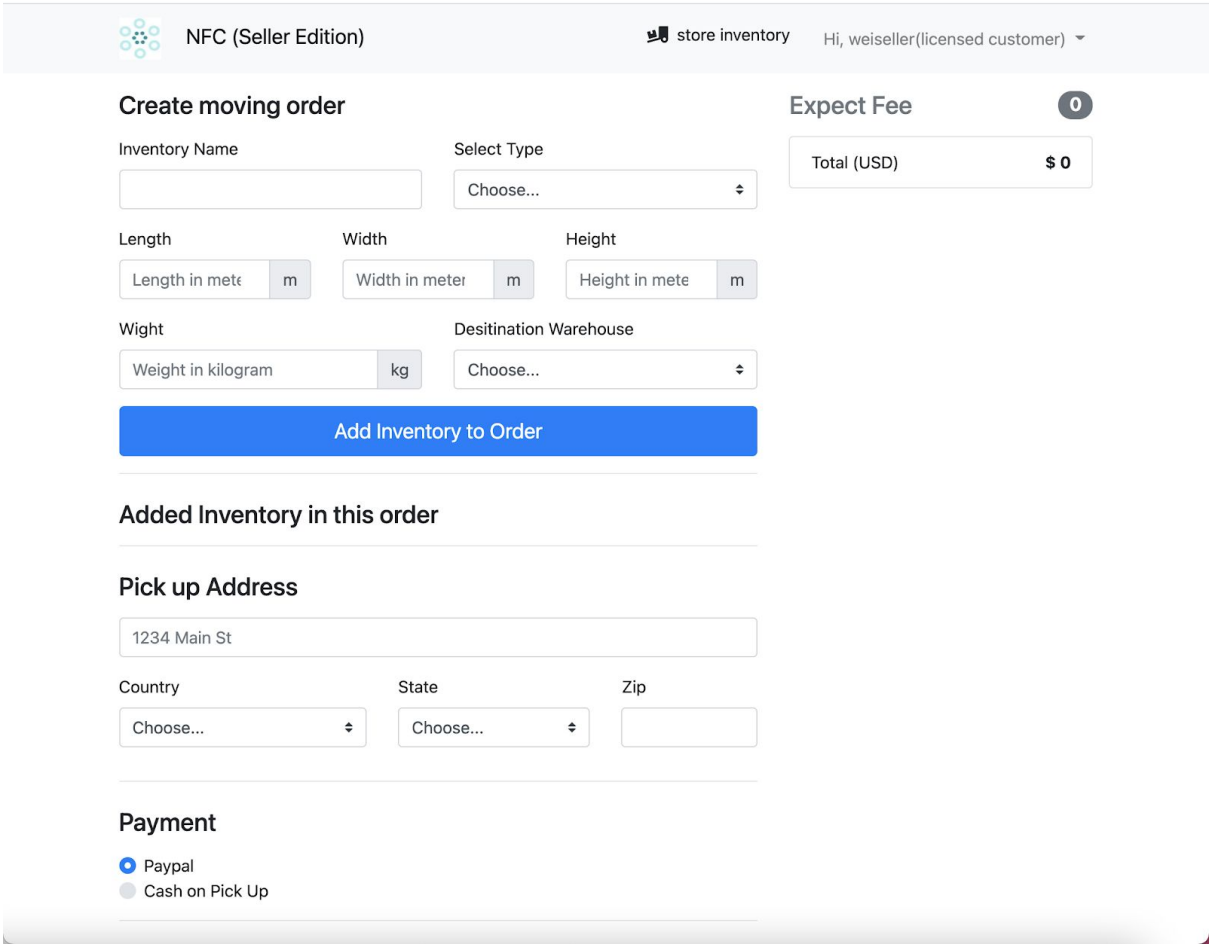
4.2 Seller Layer

The seller layer is for sellers to store inventory in the warehouse and sell it. The new seller also needs to sign up and sign in to get started.

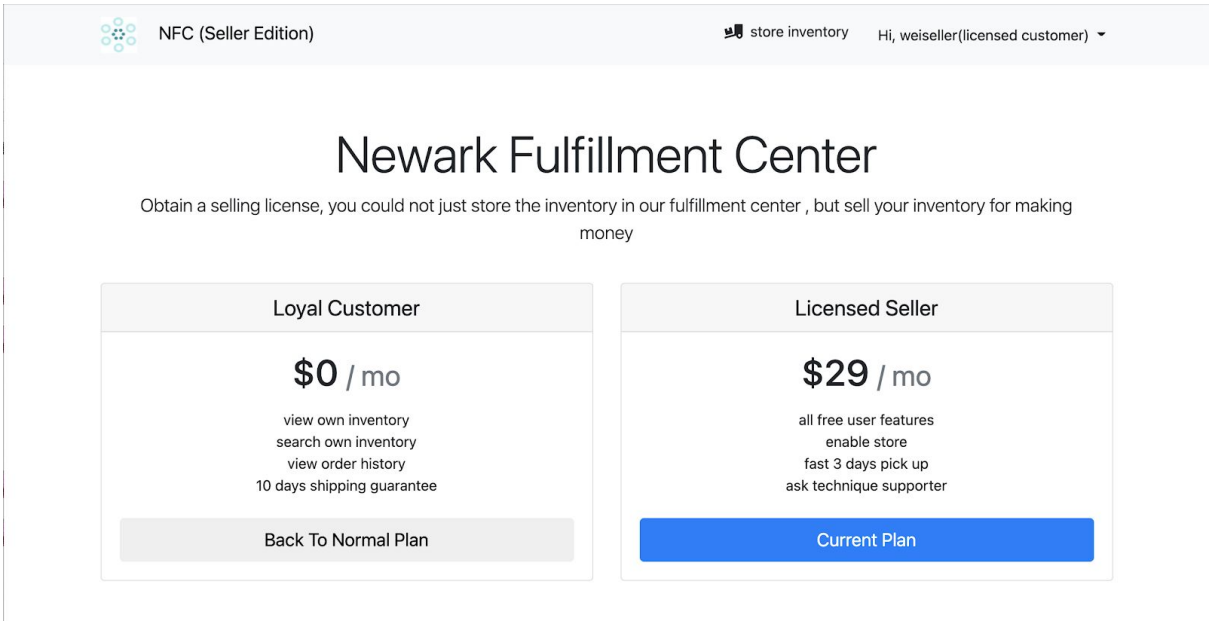




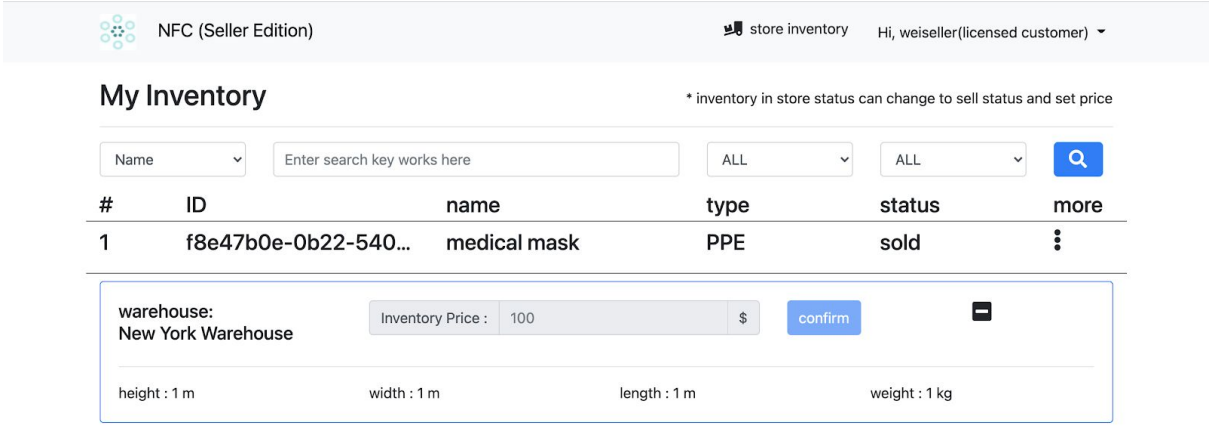
After sign in, the new user could create a shipping order to store inventory in the warehouse. For each inventory, users could give a name, select type, and choose which warehouse should be sent to. They also need to provide length, width, height, and weight, all those values will be used to calculate the shipping price. Besides, creating a shipping order also needs to provide a shipping address and select a payment method. The user can provide the country, state, zip code, and street location. If any of the information above is missed, the seller couldn't successfully submit the order.



After they submit the order, they can see their order and inventory. The new order will be shown with order created status and inventory will be prestore status. The sellers are still not available to set the inventory price and sell their inventory. They need to enable the store and wait for the manager to store the inventory in the warehouse which changes the inventory status from prestore status to store status. To enable the store, they need to change their account status from a loyal customer to a licensed customer. Then they could see more features in the inventory.

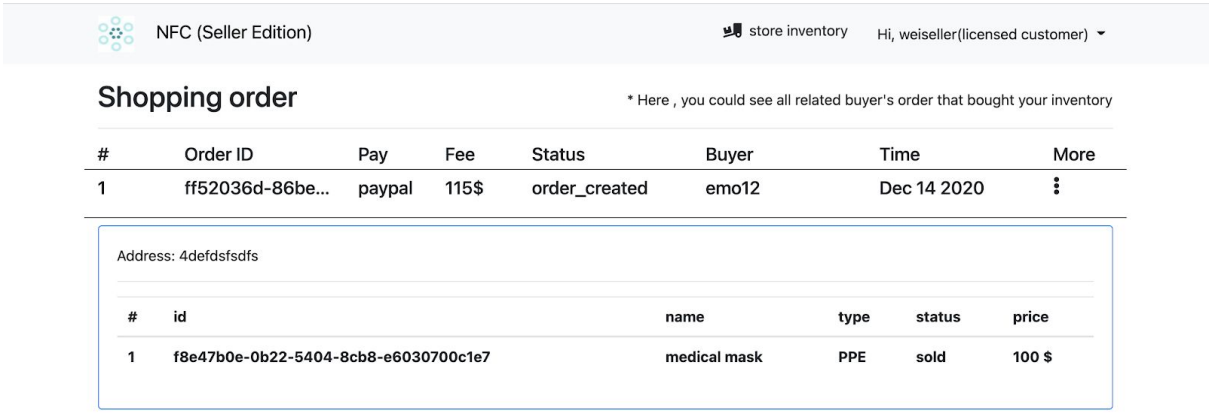


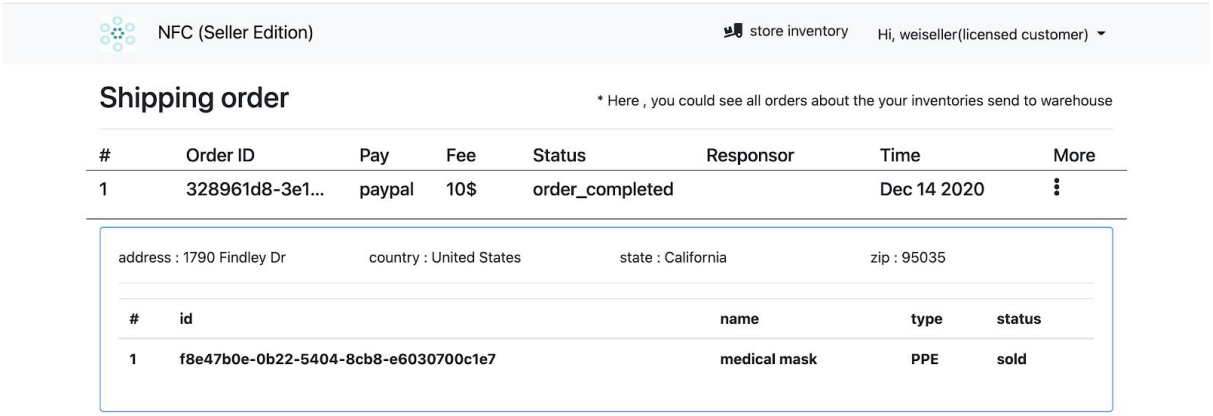
On the inventory page, sellers could search for the inventory based name, type, and status. They could click three dots to observe more details about the inventory. The little square with a minus symbol in the center is used for the seller to mark the inventory as available for sale, but it will only work only if the inventory is in sell or store status. The image below only shows it's sold status. Hence, it won't work here.



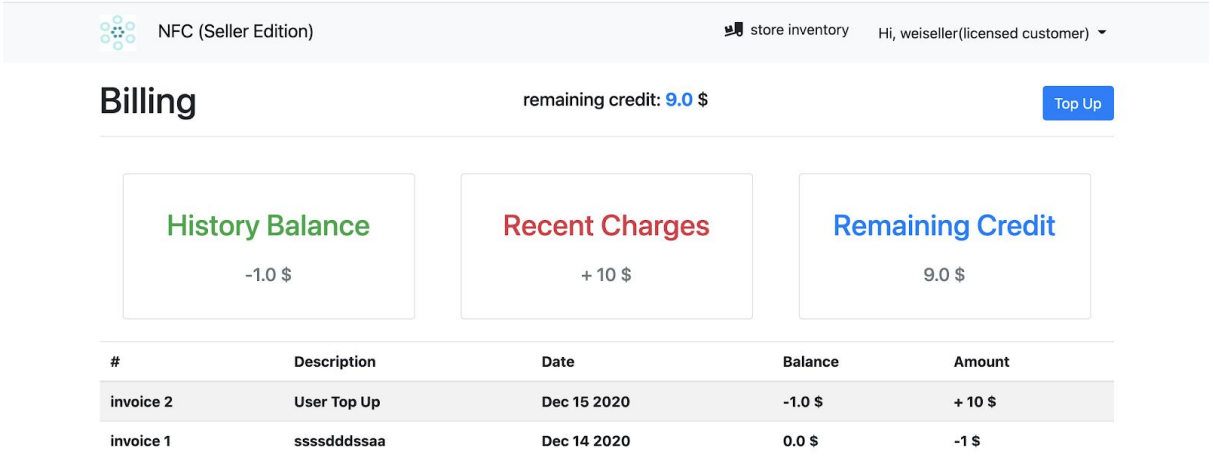
Since the website information is not synchronized. We did some prevention to prevent users from doing unexpected operations. If the inventory is not in sell status, the user couldn't change status or change the inventory price. A warning will pop up below the input area.

Furthermore, the user could not just see their shipping order, but he or she could see relevant shopping orders which contain inventory sold by him or herself.





Lastly, the user could see his or her own balance, and invoice. The balance is used for the seller to pay for inventory stored in the warehouse. So, the invoice only records the user top up and charges for store inventory.



4.3 End-user Layer

The end-user layer is similar to a normal online shopping site. You can sign in or sign up firstly. Each page has a function of checking user session. So if you click the navigation without sign-in, you will be redirected to the sign-in page automatically.

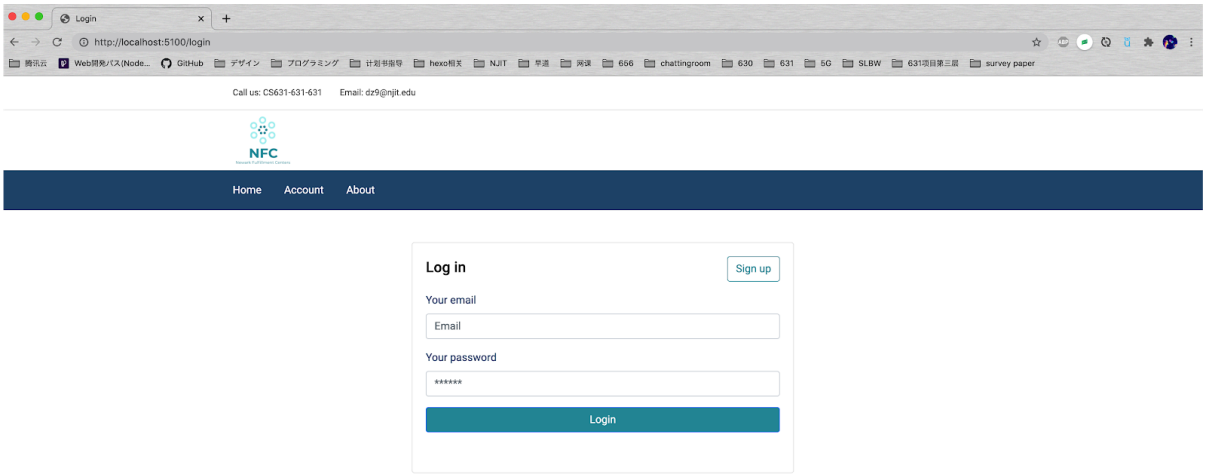


fig. sign-in/sign-up

After login, you can check your profile and history order on account page. On home page you can surf all the product which is on selling. You can also search for products by product name. When you find the product you want to buy, click the “add to cart” button.

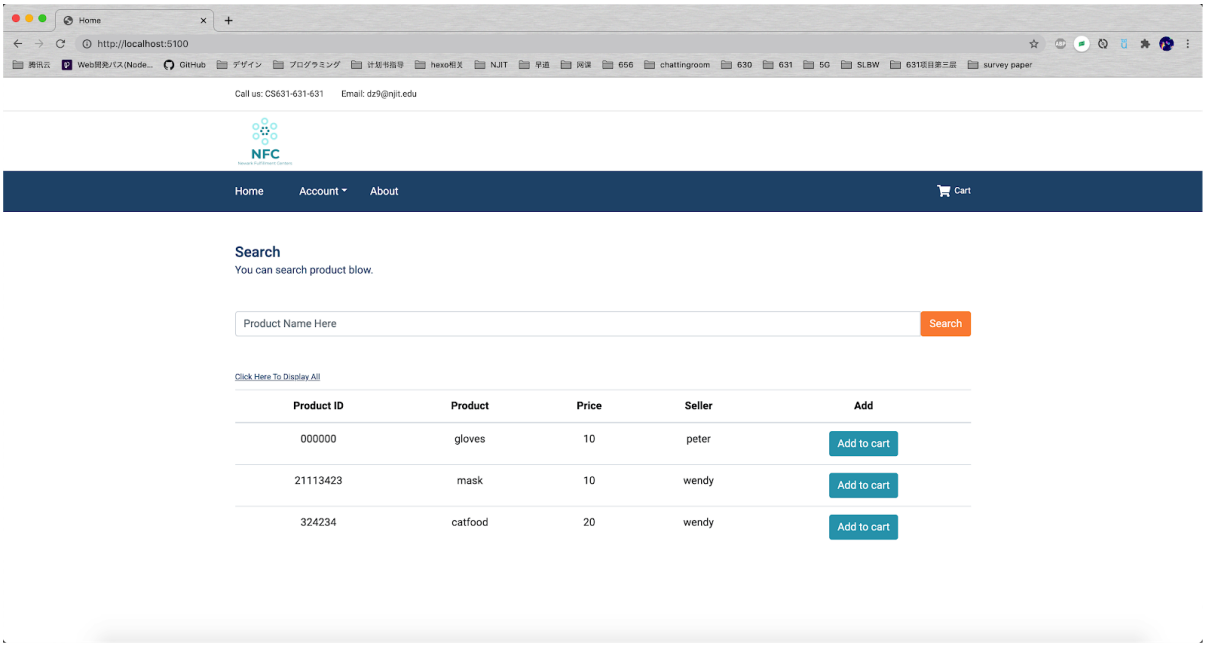


fig. view the products(inventories that status are “sell”)

The cart will calculate the total price and you just need to click the checkout button and you will come to this submit order page. Because this is a course project, we didn’t make the real payment function. When you click the submit order button, the system will think you have paid and submit this order for you.

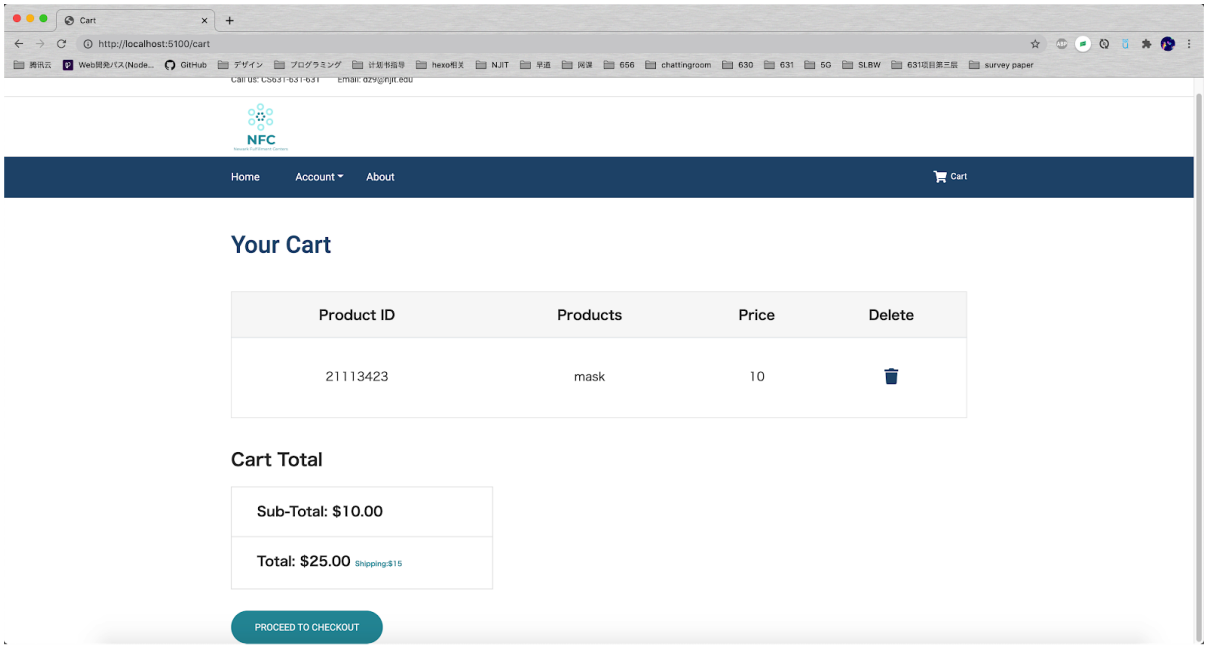


fig.cart

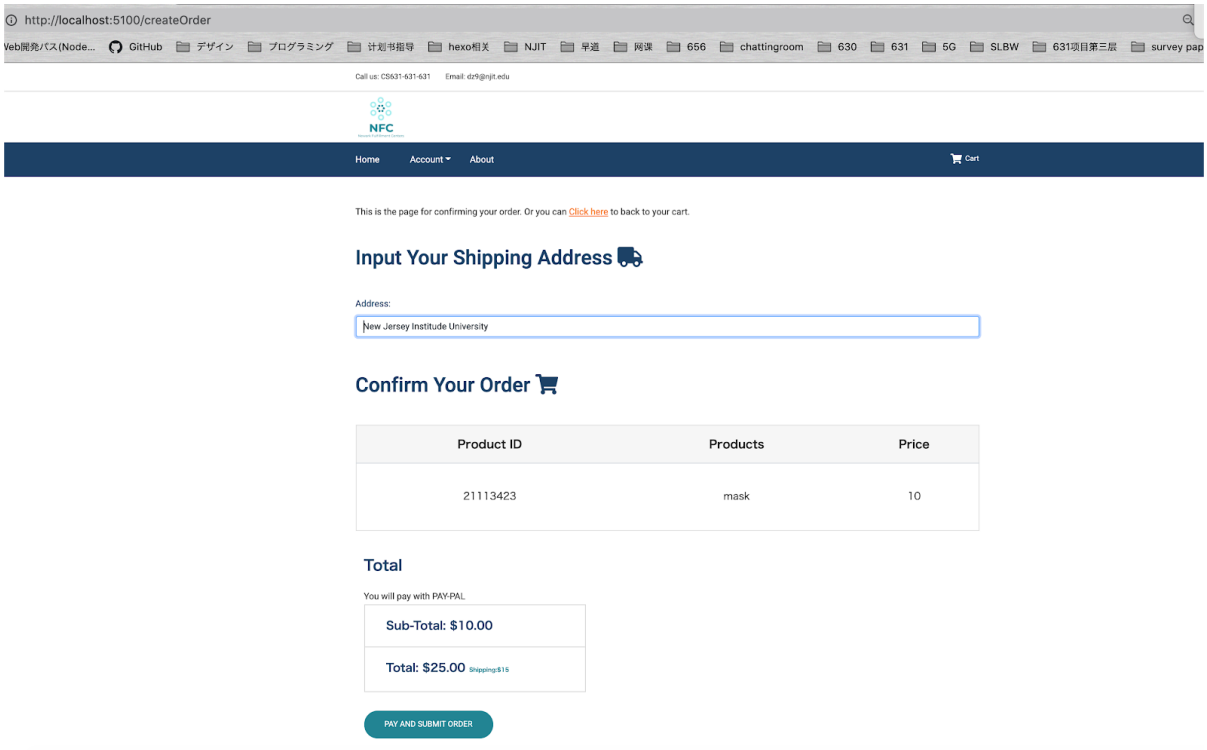


fig.submit order

5. Programming Details

5.1 Frontend

First, in the frontend, we used bootstrap4. It provides a responsive grid system and some pretty prebuilt components. I made responsive navigation and some buttons and tables by using it.

Secondly, we use jQuery to make some details. Such as the function which is checking whether passwords are the same in sign up, and the function which is to calculate the total price in the cart.

5.2 Server framework

5.2.1 Python-Flask

The seller layer is used as a flask as a backend. The main structure uses the singleton pattern to split the programming project into 4 parts. The main parts contain all the handlers used to handle a variety of seller's requests. The database class is used to handle all database operations. The tool class contains many convenient tools like parse JSON, generate universal id, etc. The config class contains all the configuration data like the database's host address, password, etc. The templates contain all the webpage which is used to render and send to the client-side. Overall, the whole structure is clear and simple.

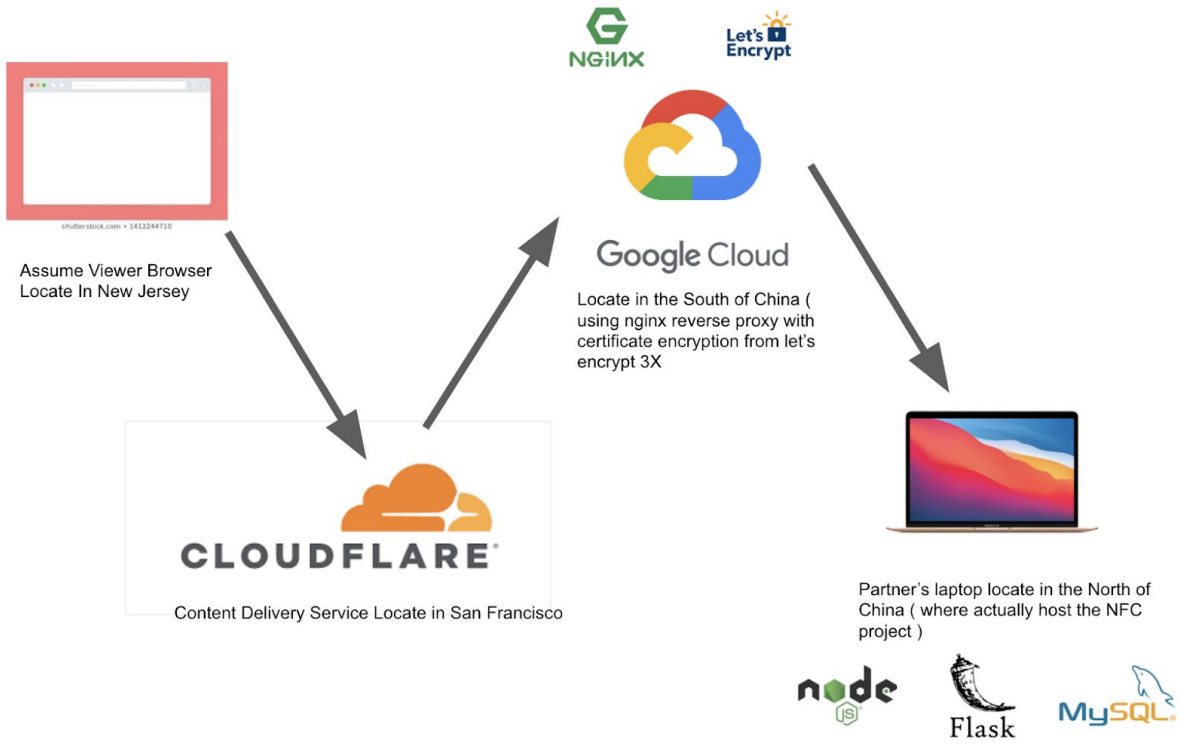
5.2.2 NodeJS-Express

We used the express framework of nodeJS as the server in the management layer and end user layer. It is responsible for communicating between client and database. Among them, some mid-ware is installed, such as node-uuid to generate uuid, and express-session to record user login information, and so on.
In the interaction with the client, post requests are received through app.post, and get requests are received through app.get. Besides, some web pages also use AJAX technology to transfer JSON data for interaction.

In the interaction between the server and the database, we use the Navicat software, so that we can see whether the database changes are in line with expectations more intuitively. Navicat's ability to export SQL files also facilitates our team's remote collaboration and ensures that our databases are consistent.

6. Difficulties Encountered and Solutions

The biggest trouble is when we try to deploy the project on the server. We find the node has trouble connecting to the server's MySQL database. We find the problem is MySQL connector API (application interface) for Node Js. We looked at some tutorials on StackOverflow, and we are still not able to solve this weird problem. So, we decided to just host this project on my partner's laptop. So, when the viewer tries to access our project. It first accessed Cloudflare's content delivery service located in San Francisco. Then, it redirects the data packets to a server provided by Google Cloud Platform and located in the South of China. Inside the server, it has let's encrypt 3X's certificate to encrypt all the data between browser and server. And, we use Nginx reverse proxy to redirect the data again. The packets finally arrive at Zhang's laptop in North of China where the programming project is located.



7. Conclusion

This report introduces our development process, project achievements, technical principles, and summarizes the main problems encountered in the project. In this project, we have gone through several stages: analysis, database design, prototype design, actual development and testing. In these stages, we have learned a lot. First of all, we have a deeper understanding of the importance of the database, whether it is the design of the database, or the selection of database software in the project development is crucial. Then, we have a practical experience of the development process of web app, and gained a lot of specific technical knowledge. Finally, as a team, we learned a lot of knowledge from each other. Hope we can sum up the experience of this project and apply it to the future study and work.