Cosmeticvue

Anqi Li 20082481

**Description：**

The basic functions of this web app are register, login and logout functions. Register and login icons always appear together when users do not login. Logout icon shows up when users logging or already logged in. Login function will store user’s ID, role and token which is create from the back end in session storage. ID and token are needed in some API and the navbar changes by roles. There are two role in the website (seller and customer), and they are provided different functions respectively.

Seller can check their own products on cosmetics management page after login successfully. In the cosmetics list page, seller can edit and delete released cosmetics. Besides, there is a page for adding new cosmetics. Whatever editing or adding, the input information should be validated in the front end before submitting. These three operations will interact with the cosmetics schema in MongoDB database.

Customer can view all of the cosmetics and order them. In cosmetic list page, there are name filter and brand filter. However, the order function needs authentication. It means that users should login as customer first. The order form automatically validate the input information and it integrates google map in shipping address area. The order operation connects to the transaction schema in MongoDB database.

The home page and cosmetics List page do not request authorization, but others need authorization. If a user try to access page requested authorization without token, web app will redirect to login page.

This web app use MEVN (MongoDB + Express + Vue.js + Node.js) framework. The front-end uses Vue.js technology and the back-end use Node.js and express technology. MongoDB as a document-oriented database handle the persistence of this application. Finally, the server deploys to heroku and the client deploys to the firebase.

**App Features:**

Get a list of cosmetics.

Get a list of cosmetics belong to a seller using a seller ID.

Add a cosmetic in JSON format to database using POST request.

Add an order in JSON format to database using POST request.

Filter all cosmetics and orders.

Delete an individual cosmetic using cosmetic ID.

Edit a cosmetic information via Put request using cosmetic ID.

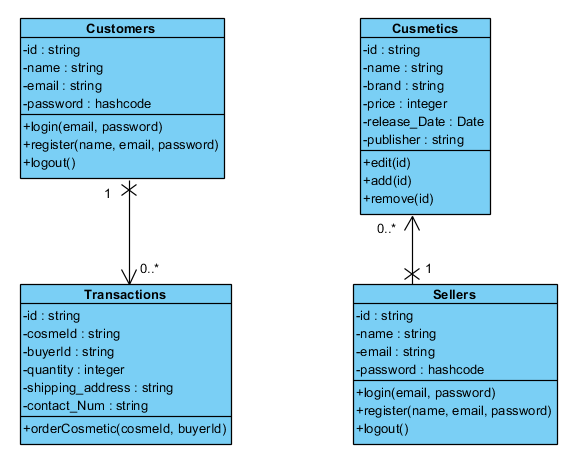
Register a seller/customer account via Post request.

Login a seller/customer account via POST request using email and password.

Logout an account to release the resources.

Show and store the location of shipping address in the Google Map and database.

**Class Diagram:**



**Use Cases:**

Use case 1: Register

Actor: seller, customer.

Scenario 1: Register.

1. User clicks the register icon.
2. System jump to register page.
3. User fills the register form and submit the form.
4. System creates a new account successfully and save it to the database.
5. System redirects to the home page.

Scenario 2: As above (1 to 5) but with the following modifications:

Alternative: Replicate name.

At step 3, the name entered by the user already exists in the system.

Allows user to re-input name.

Scenario 3: As above (1 to 5) but with the following modifications:

Alternative: Replicate email.

At step 3, the email entered by the user already exists in the system.

Allows user to re-input email.

Scenario 4: As above (1 to 5) but with the following modifications:

Alternative: Wrong format of email.

At step 3, the email entered by the user is not a email format.

Allows user to re-input email.

Use case 2: Login

Actor: seller, customer

Scenario 1: Login.

1. User clicks the login icon.
2. System jump to login page.
3. User select role, inputs email and password.
4. User clicks the button to login.
5. System match role, email and password successfully from database successfully.
6. System redirect cosmetic list page (customer) or cosmetic management page (seller).

Scenario 2: As above (1 to 6) but with the following modifications:

Alternative: Match failure.

At step 5, system fails to match email and password in customer schema or seller schema from the database.

Allows user to re-input and login again.

Use case 3: Order cosmetics from cosmetics list page

Actor: customer.

Scenario 1: Order a cosmetic.

1. Customer browses cosmetics on the cosmetics list page.
2. System displays all cosmetics.
3. Customer chooses a cosmetic to order.
4. System jump to order page.
5. Customer inputs quantity, shipping address and contact number, and submit the order.
6. System upload the order to the transaction schema in database.

Scenario 2: As above (1 to 6) but with the following modifications:

Alternative: Using filter to find cosmetics.

At step 3a, Customer inputs fuzzy word in name filter and/or brand filter.

Scenario 3: As above (1 to 6) but with the following modifications:

Alternative: Submit failure.

At step 5, customer does not input correct information in order form.

Allows customer to re-input and submit again.

Use case 4: Release cosmetics

Actor: seller.

Scenario 1: Release cosmetic.

1. Seller goes to the add cosmetics page.
2. Seller fills the detail of cosmetic.
3. Seller submits the form of cosmetic.
4. System uploads cosmetics to the cosmetics schema in the database.
5. System redirect to the cosmetics management page and loads all cosmetics belong to this seller.

Scenario 2: As above (1 to 5) but with the following modifications:

Alternative: Submit failure.

At step 5, seller does not input correct information in cosmetic form.

Allows seller to re-input and submit again.

Use case 5: Edit cosmetics

Actor: seller.

Scenario 1: Edit cosmetics released by seller.

1. Seller bowers cosmetics on cosmetic management page.
2. Seller selects a cosmetic to edit.
3. System jump to edit page.
4. Seller changes information of cosmetic and save it.
5. System updates cosmetic to the cosmetics schema in the database.
6. System redirect to the cosmetics management page and loads all cosmetics belong to this seller.

Scenario 2: As above (1 to 6) but with the following modifications:

Alternative: Save failure.

At step 4, seller does not input correct information in cosmetic form.

Allows seller to re-input and submit again.

Use case 6: Delete cosmetics

Actor: seller.

Scenario 1:Delete a cosmetic released on website.

1. Seller finds cosmetics on cosmetics management page.
2. Seller selects a cosmetic and deletes it.
3. System pops up remaining box.
4. Seller confirms to remove cosmetic.
5. System removes cosmetic from the cosmetics schema in the database.

Scenario 2: As above (1 to 5) but with the following modifications:

Alternative: Remove cancel.

At step 4, seller cancel the deletion.

Cosmetic still saves in the database.

Use case 7: Logout

Actor: seller, customer

Scenario 1: Logout

1. User clicks the logout icon.
2. System pops up remaining box.
3. User confirm to logout.
4. System release the resource of user.

Scenario 2: As above (1 to 4) but with the following modifications:

Alternative: Logout cancel.

At step 3, user cancel the logout.

User still in the login statue.

**Database Schemas:**

Cosmetics Schema :

name: {type: String, required: true},

brand: {type: String, required: tru},

price: {type: Number, required: true},

publisher: {type: String, required: true},

release\_date: Date

Customers Schema:

name: {type: String, required: true, unique: true},

email: {type: String, match:/^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+(\.[a-zA-Z0-9\_-])+/,

required: true, unique: true},

password: {type: String, required: true},

phoneNum: String,

address: String,

register\_date: Date

Sellers Schema:

name: {type: String, required: true},

email: {type: String, match:/^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+(\.[a-zA-Z0-9\_-])+/,

required: true, unique: true},

password: {type: String, required: true},

description: String,

register\_date: Date

Transactions Schema:

cosmeId: {type: String, required: true},

buyerId: {type: String, required: true},

quantity: {type: Number, min: 1, required: true},

shipping\_address: {type: String, required: true},

contact\_Num: {type: Number, required: true},

last\_date: Date,

status:{ type: String, enum: ['unpaid', 'paid', 'delivering', 'finished']}

**Heroku Link:**

https://cosmeticapi.herokuapp.com/

**Firebase Link:**

https://cosmeticvue-8d355.firebaseapp.com/

**GitHub Link:**

<https://github.com/SMARTBIGBOSS/cosmeticclient.git>

**YouTube Link:**

<https://youtu.be/0gvMq-pCJ9s>

**User experience approach:**

Understanding the user’s requirement firstly. Then I create use cases and user scenarios to outline the web app. After identifying the goals, design user interfaces to support the interaction between users and application. Developing and testing the web app, which is a highly iterative process.

**Developer experience approach:**

I use a video to show what can the website do and using a design document file to descript this website. Release notes and changelogs using Git during the development and upload source code to my GitHub account. Using README file to descript this website.

**Reference:**

<https://cn.vuejs.org/v2/guide/components-slots.html>

https://cn.vuejs.org/v2/guide/conditional.html

<https://www.w3schools.com/icons/fontawesome_icons_webapp.asp>

<http://www.w3school.com.cn/cssref/pr_background-size.asp>

<https://alligator.io/vuejs/vue-google-maps/>

<https://developers.google.com/maps/documentation/embed/guide>

<https://www.ltnow.com/when-your-map-says-this-page-cant-load-google-maps-correctly/>

<https://www.interaction-design.org/literature/topics/ux-design>

<https://blog.prototypr.io/a-common-product-ux-design-process-55af4ab5665e>