



Final Project Report

HYPEGEAR

By

Mr. Nuttipong	Wattanaboonma	6588116
Mr. Pornpatchara	Wattananan	6588117
Miss Thitiwan	Keattitat	6588174
Miss Kirana	Teerachanatarn	6588197

**A Report Submitted in Partial Fulfillment of
the Requirements for**

ITCS212 Web Programming

**Faculty of Information and Communication Technology
Mahidol University
2024**

Table of content

	Page
Business Domain	1
Overview of HYPEGEAR	2-5
Phase I	
Task 1 Wireframe	
1.1. HomePage	6
Collection	7
Detail Page of Collection	8
1.2. Detail Page	9
Sale	10
Detail of sale page	11
Support	12
1.3. Search Page	13
Advance Search	14
1.4. Login Page	
Admin&User login page	15
User Registor's page	16
Admin or User log in already	17
Admin page	18
1.5 Product Management	19
Add page	20
Search for Edit page	21

Edit page	22
Edit success page	23
Delete page	24
Delete success page	25
1.6 User Account Management	26
Update	27
Delete	28
YES button	28
NO button	28
1.7 Team Page	29
1.8 Additional Page	
Our service page	30
Privacy Policy page	31
Terms & Conditions page	32
Task 2	
ERD Diagram	33
Details of Data Model	34-36
Phase II	
1 - Project Overview	37
1.1 Homepage	37
1.2 Register page	37
1.3 Login page	37

1.4 Search page	37
1.5 Advance Search page	38
1.6 Detail page	38
1.7 Admin list page	38
1.8 User list page	38
1.9 Product list page	38
1.10 Homepage and logo	39
1.11 Guess the new collection	39
1.12 Sale page	39
1.13 Support page	39
1.14 About us page	39
1,15 Our Service page	39
1.16 Privacy page	39
1.17 Term & conditions page	39
2 - Navigation Diagram of web Application	40
3 - Details of your web application and code	41
3.1 Home_page.html	41
3.2 Guess.html	44
3.3 Guess.js	45
3.4 Login.html	46
3.5 Login.js	49
3.6 Register.html	50
3.7 Register.js	52
3.8 Edit.html	54
3.9 Edit.js	55
3.10 Product.html	56
3.11 Product.js	57

3.12 hosting.js	60
3.13 Advance Search.html	63
3.14 Advance Search.js	65
3.15 Search_showproduct.html	68
3.16 Search_showproduct.js	69
3.17 nav.js	74
3.18 List_user.html	83
3.19 List_user.js	84
3.20 User_list.html	86
3.21 User_List.js	87
3.22 List_admin.html	90
3.23 List_admin.js	91
3.24 Admin_detail.html	92
3.25 Admin_detail.js	94
3.26 Admin_detail_add.html	96
3.27 Admin_detail_add.js	98
4 - Details of your web service and code	100
5 - Testing results of web services using Postman	108
5.1 Log-In	108
5.2 Register	109
5.3 Select All Admins	109
5.4 Select All Products	110
5.5 Select All Users	110
5.6 Select Admin by Admin ID	111
5.7 Select Users by Email	111
5.8 Select Product by product ID	112
5.9 Update Admin	112

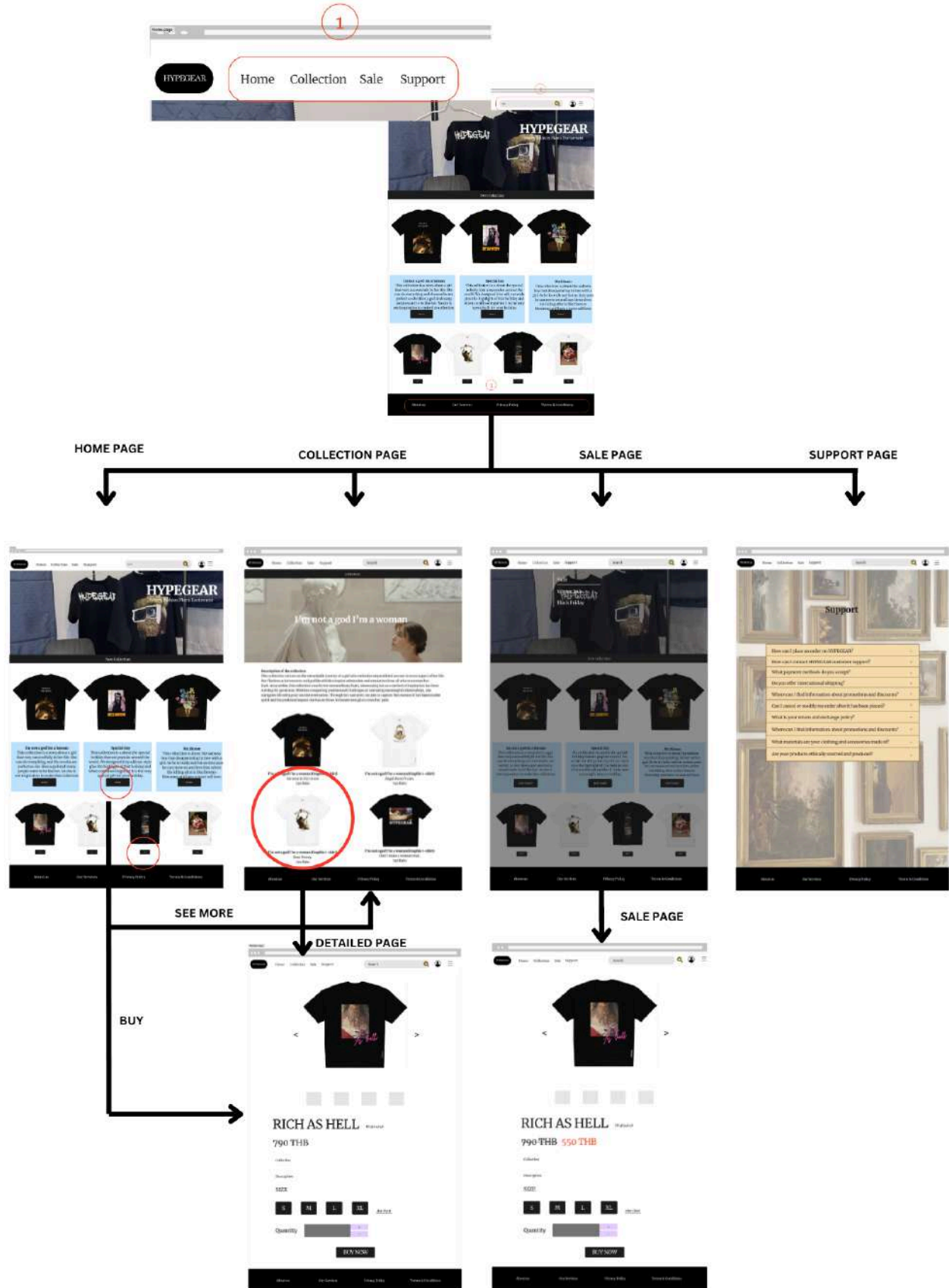
5.10 Update User	113
5.11 Update Product	113
5.12 Delete Admin by Admin ID	114
5.13 Delete User by Email	114
5.14 Delete Product by Product ID	115
5.15 Advance Search	115
5.16 Insert Admin	116
5.17 Insert Product	116
Reference	117

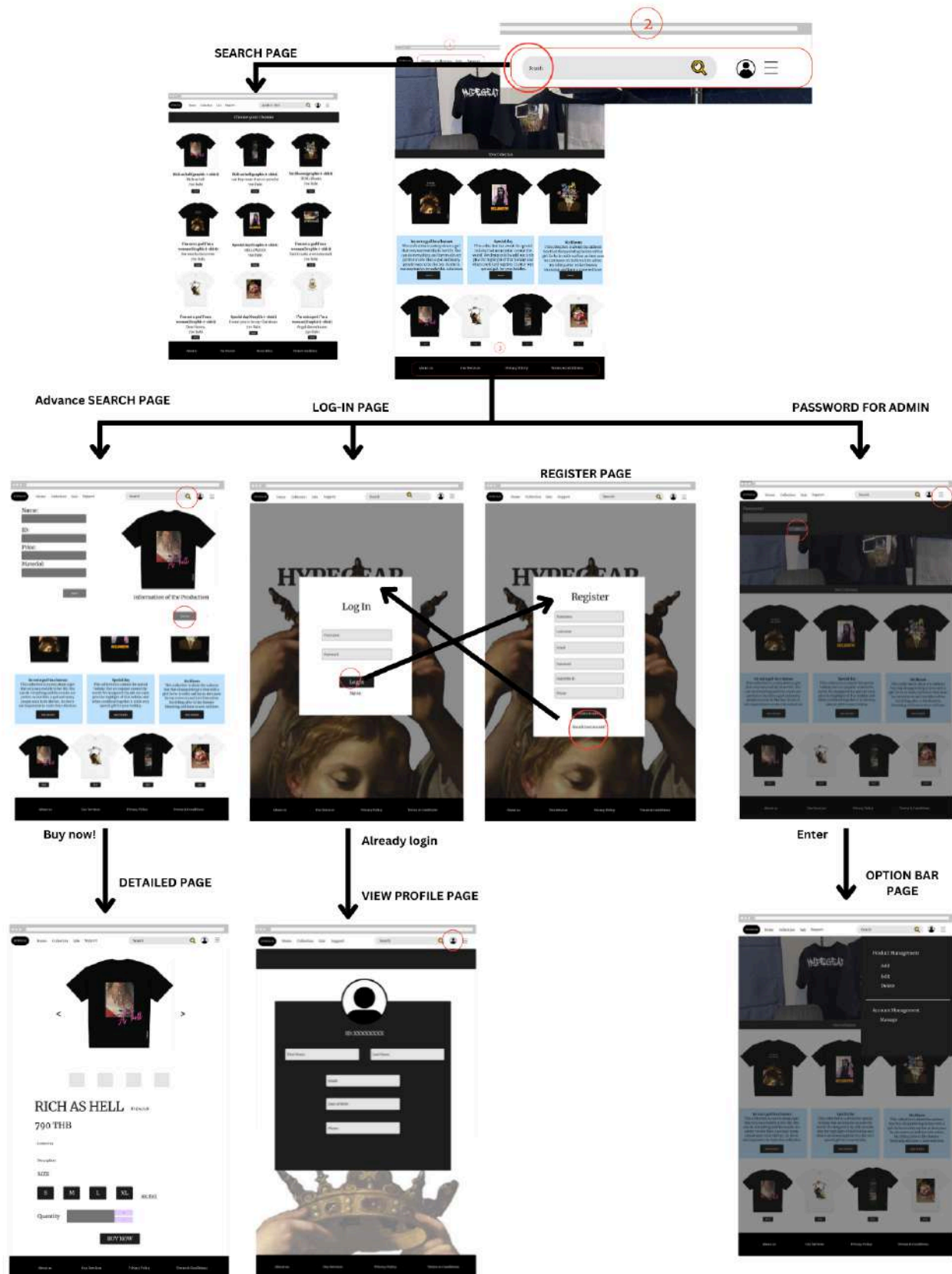
Business Domain: Fashion Retailer online store.

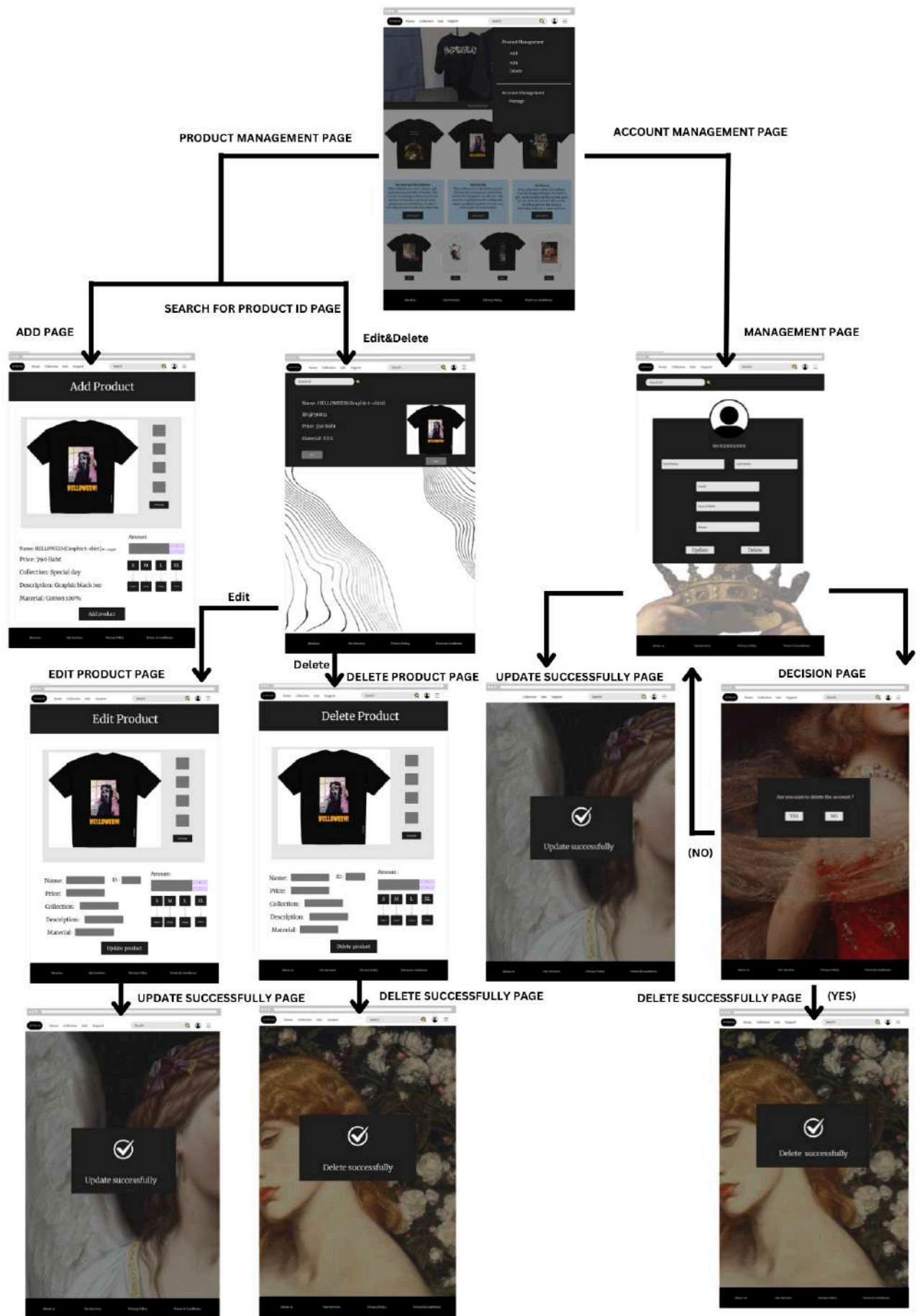
History of HYPEGEAR

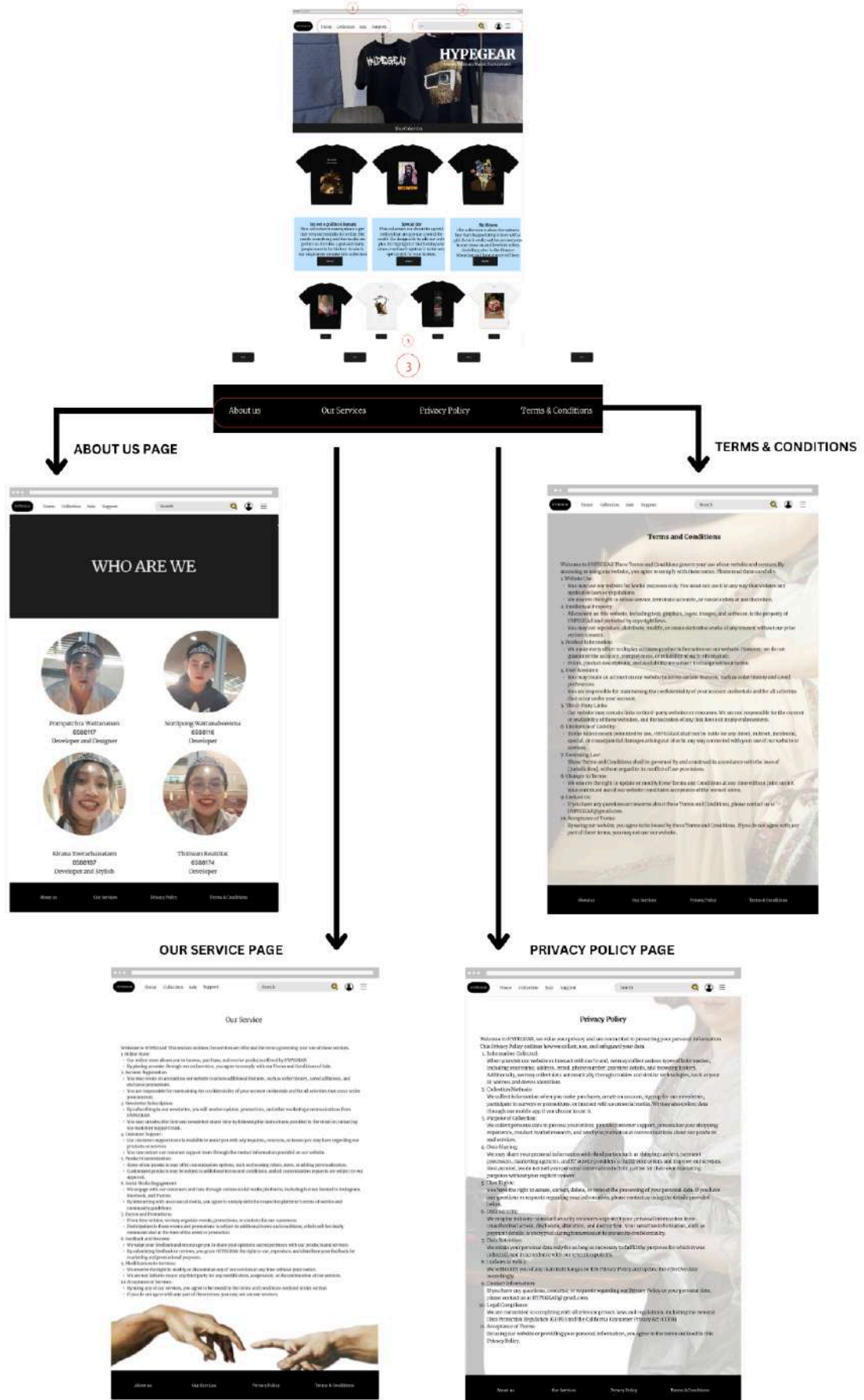
HYPEGEAR started with a love for art and fashion. It's a place where cool art meets awesome clothes. Think of it like wearing your creativity on your sleeve. From tees with cool designs to accessories that pop, HYPEGEAR is all about showing off your style in a unique way. We're inspired by the colorful vibe of the renaissance and street art. Our mission is simple to help you stand out and express yourself with what you wear. Whether you're into bold patterns or subtle statements, HYPEGEAR has something for everyone. Join us on our journey where fashion meets fun and creativity rules the runway. Welcome to HYPEGEAR, Where Fashion Meets Excitement.

Overview of HYPEGEAR







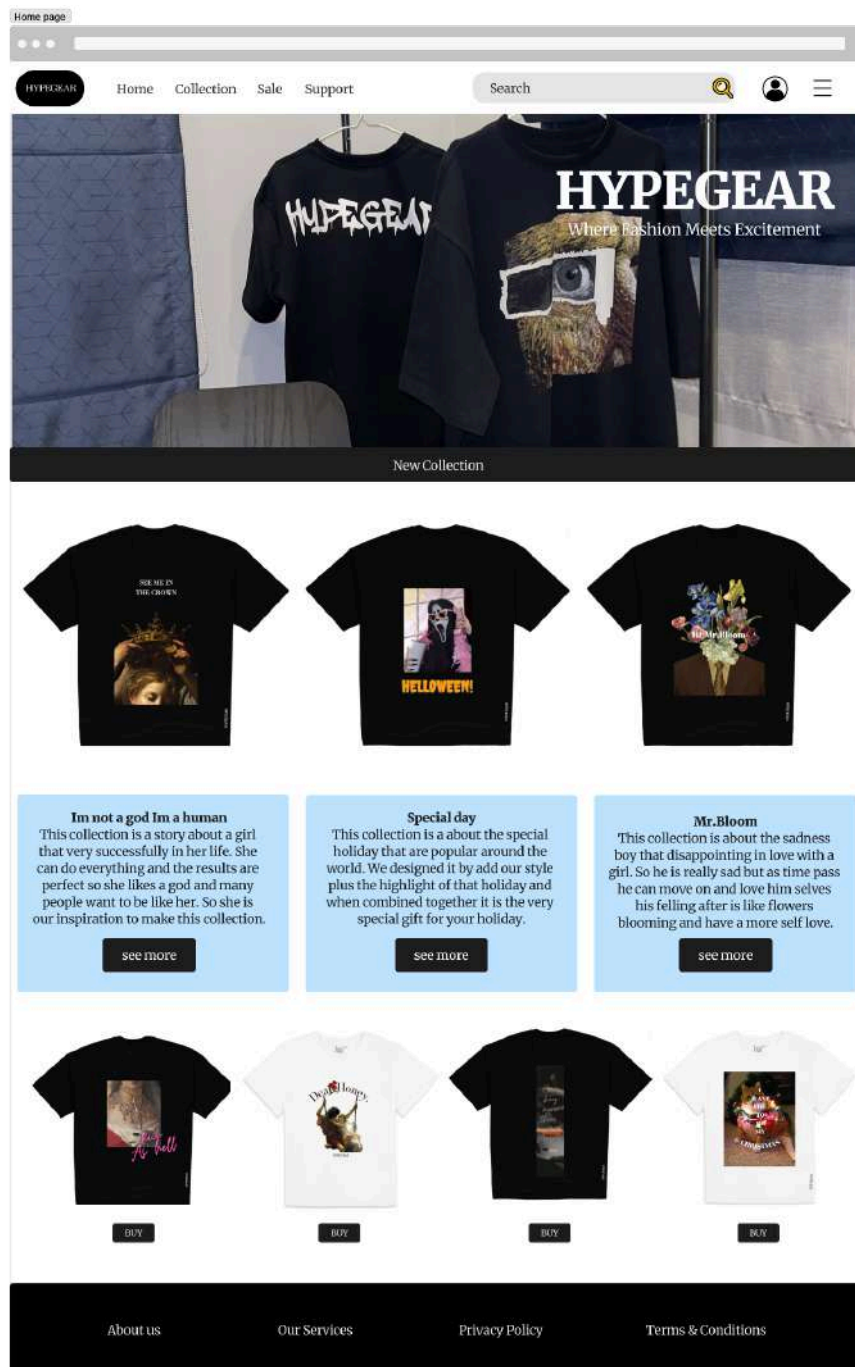


Phase I

Task 1: Wireframe

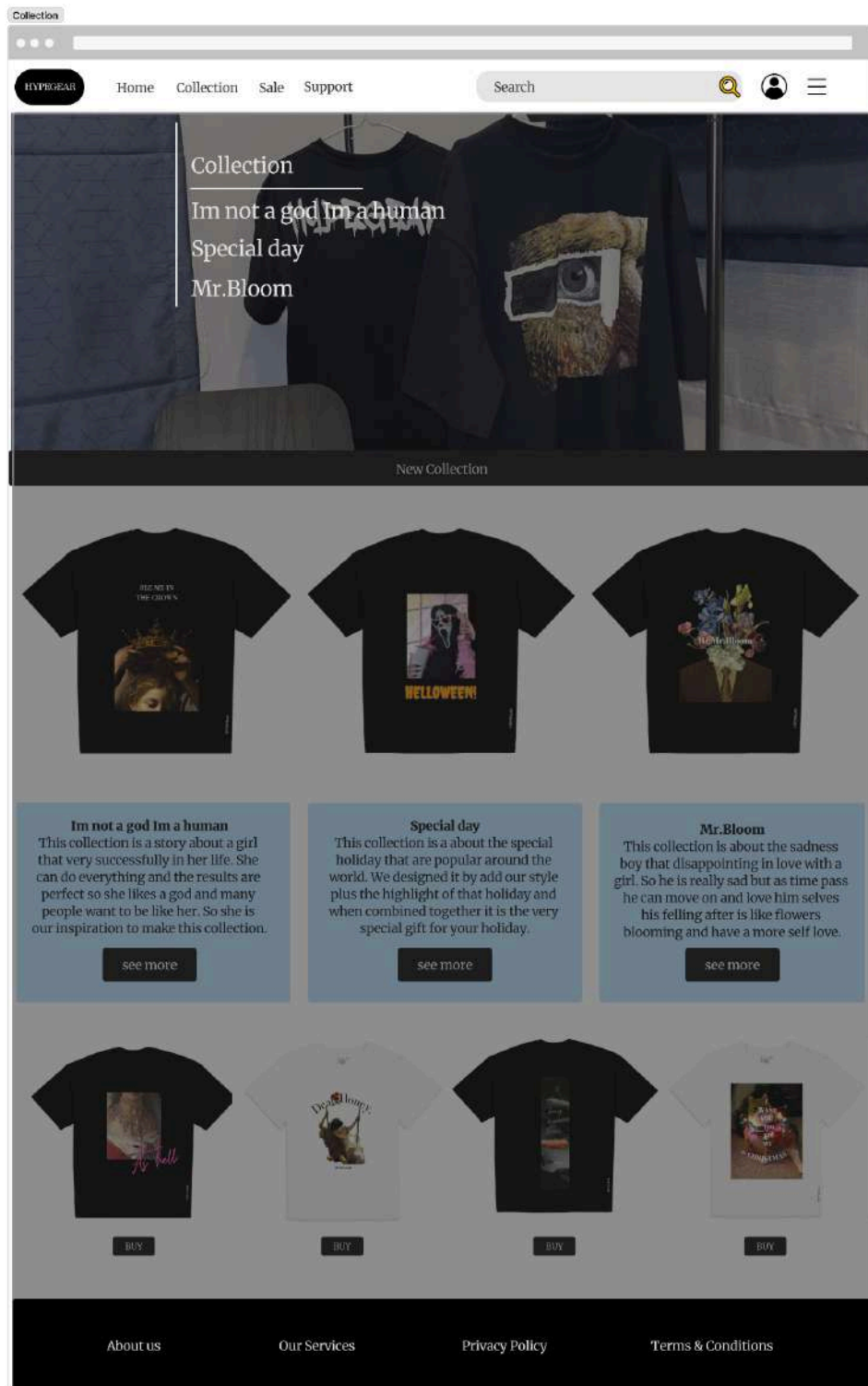
1. HomePage

This page is the first page that users can see. Users can go through Home, Collection, Sale, and Support at the navigation bar and can quickly search and filter search to find the right products. Users can click on the account icon to see their profile info. Users can see many new collections and recommended products on the homepage. And Admin can click the options bar to Add, Edit, Delete, and manage the products stocks.

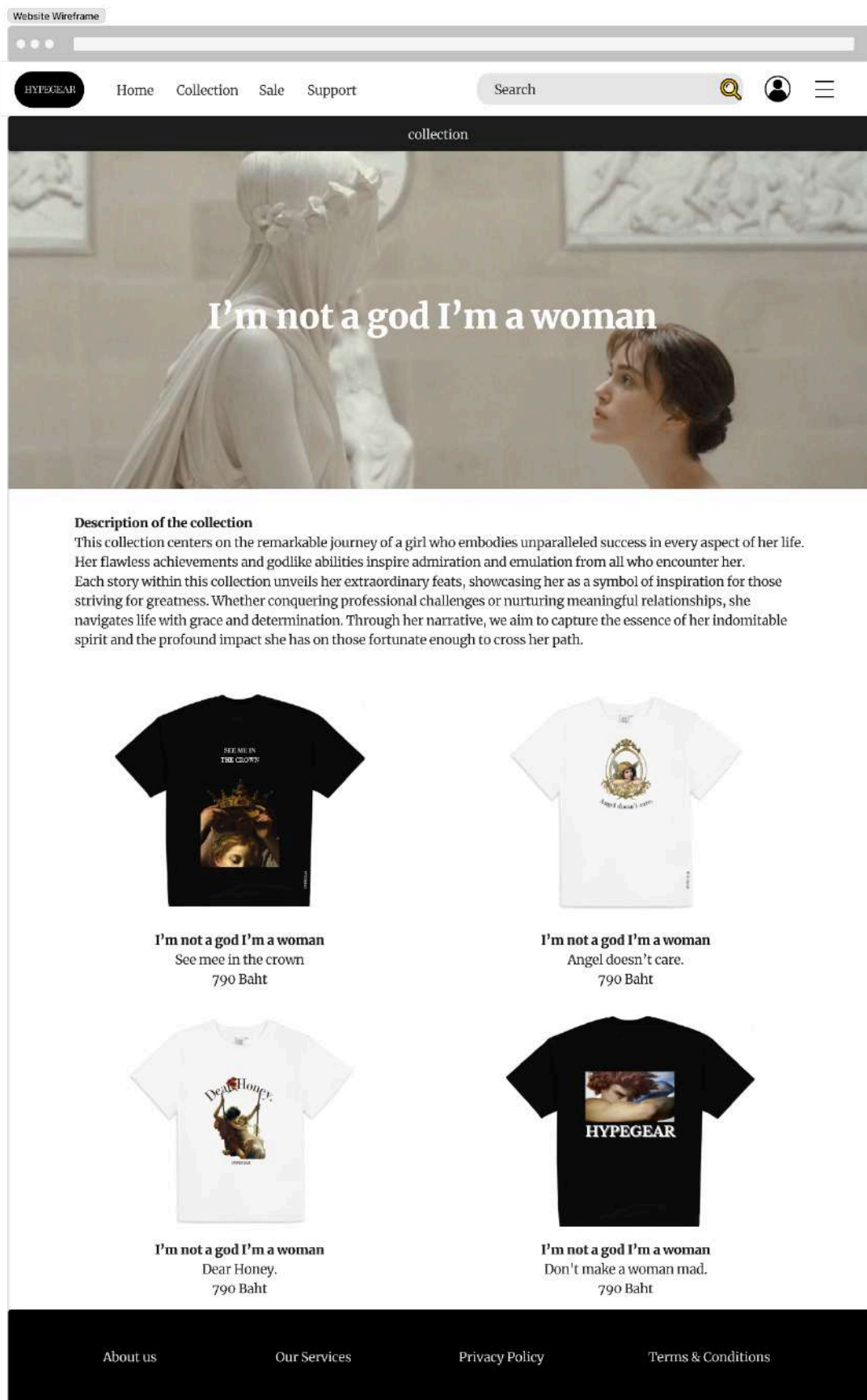


From the navigation bars :

Collection : If you click on each collection, you'll be taken to its own page. There, you can explore more about that collection. Each collection has its own special style and interesting story. Whether you like cool streetwear, simple looks, or colorful designs, you'll find something for you. We make it easy for you to find what you like. Just click, and you'll see all the details.



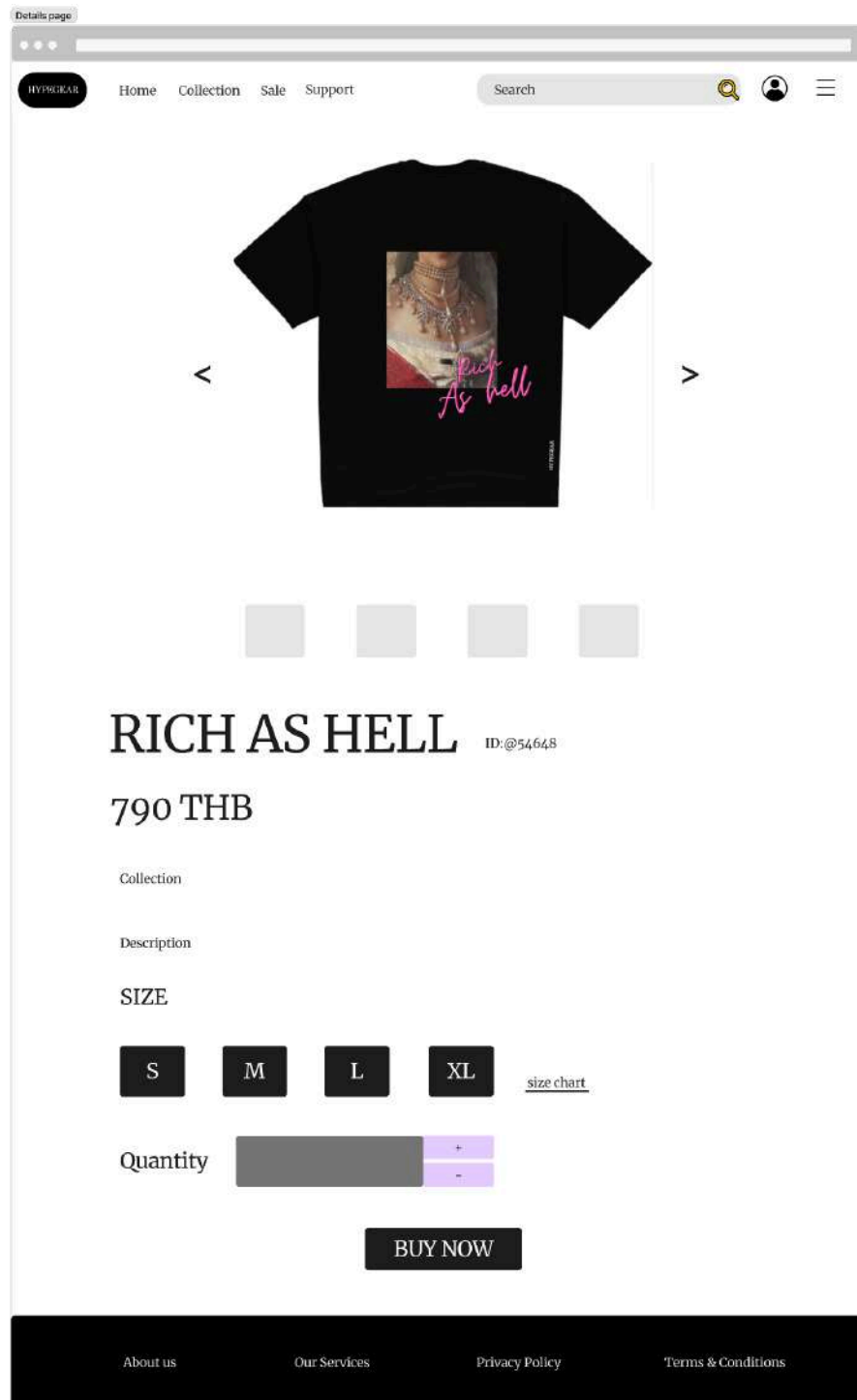
Detail Page of Collection:



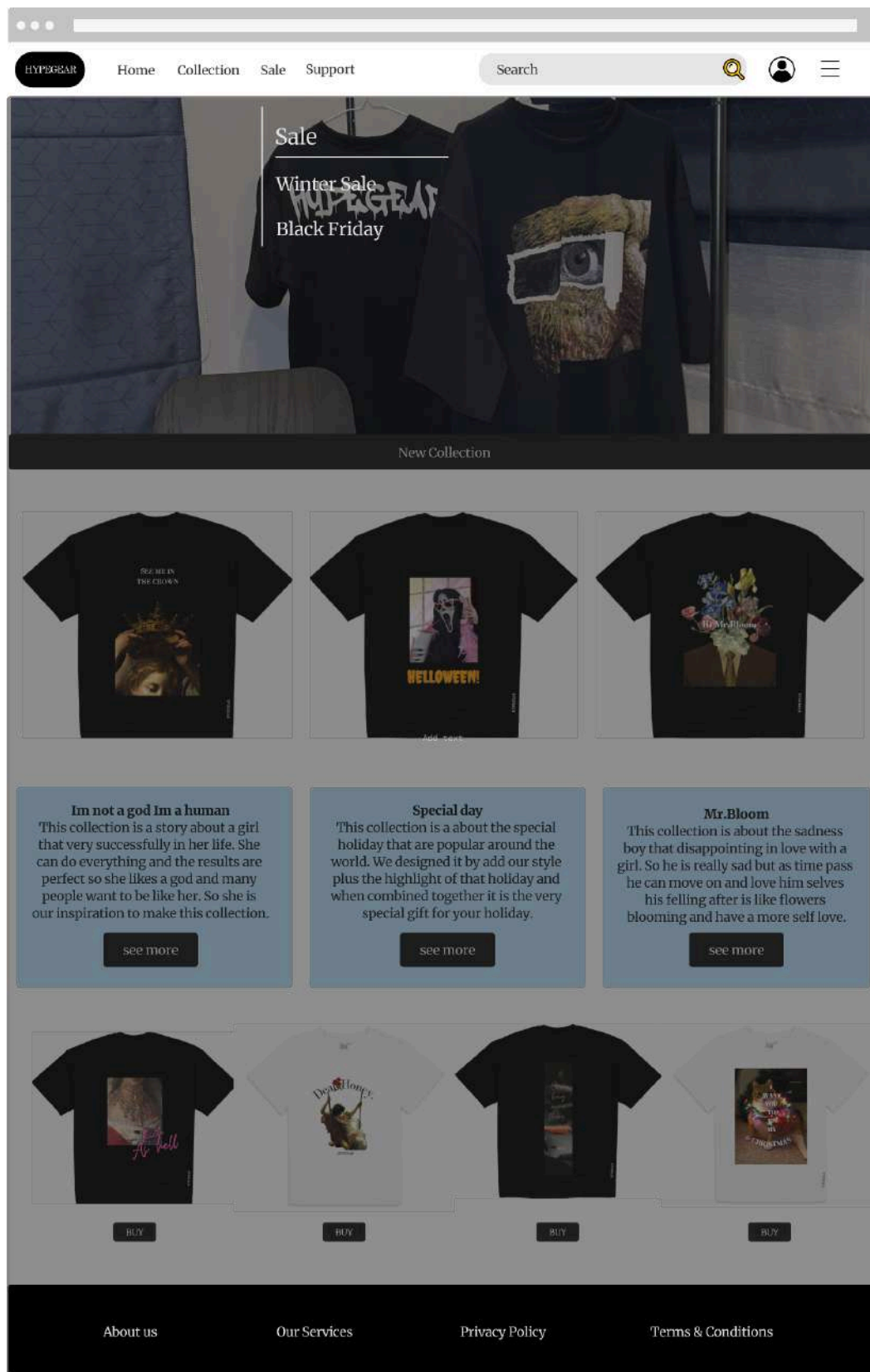
If you click on each product, it will show you a detailed page of the product.

2. Detail Page

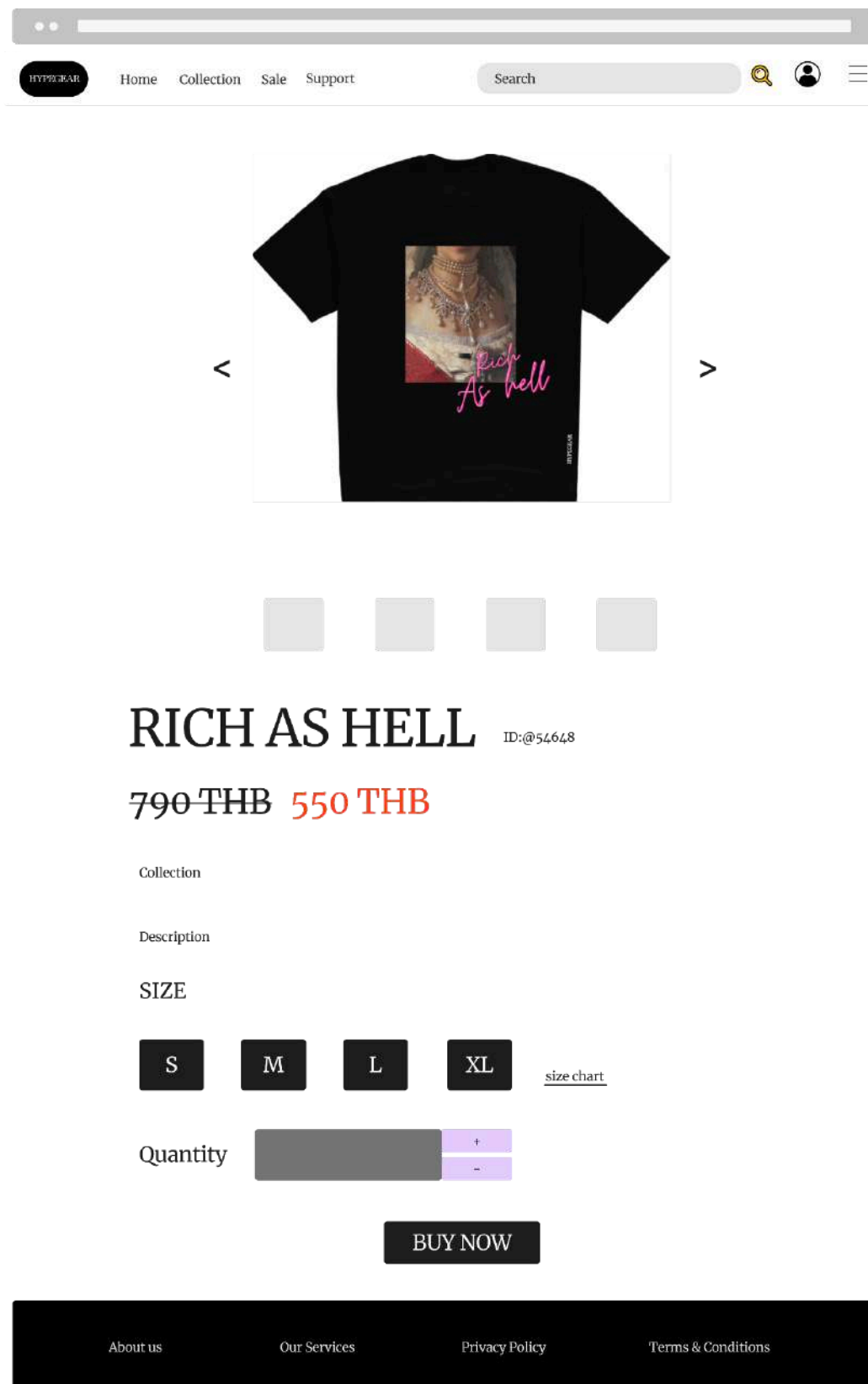
For the details page, it displays comprehensive product information, including multiple product images from different angles, product name, ID, price, collection, and description. It also features available sizes (S, M, L, XL) with a size chart for customer reference, following the quantity selector for ordering. A clear 'BUY NOW' button makes the website full and encourages simple purchases.



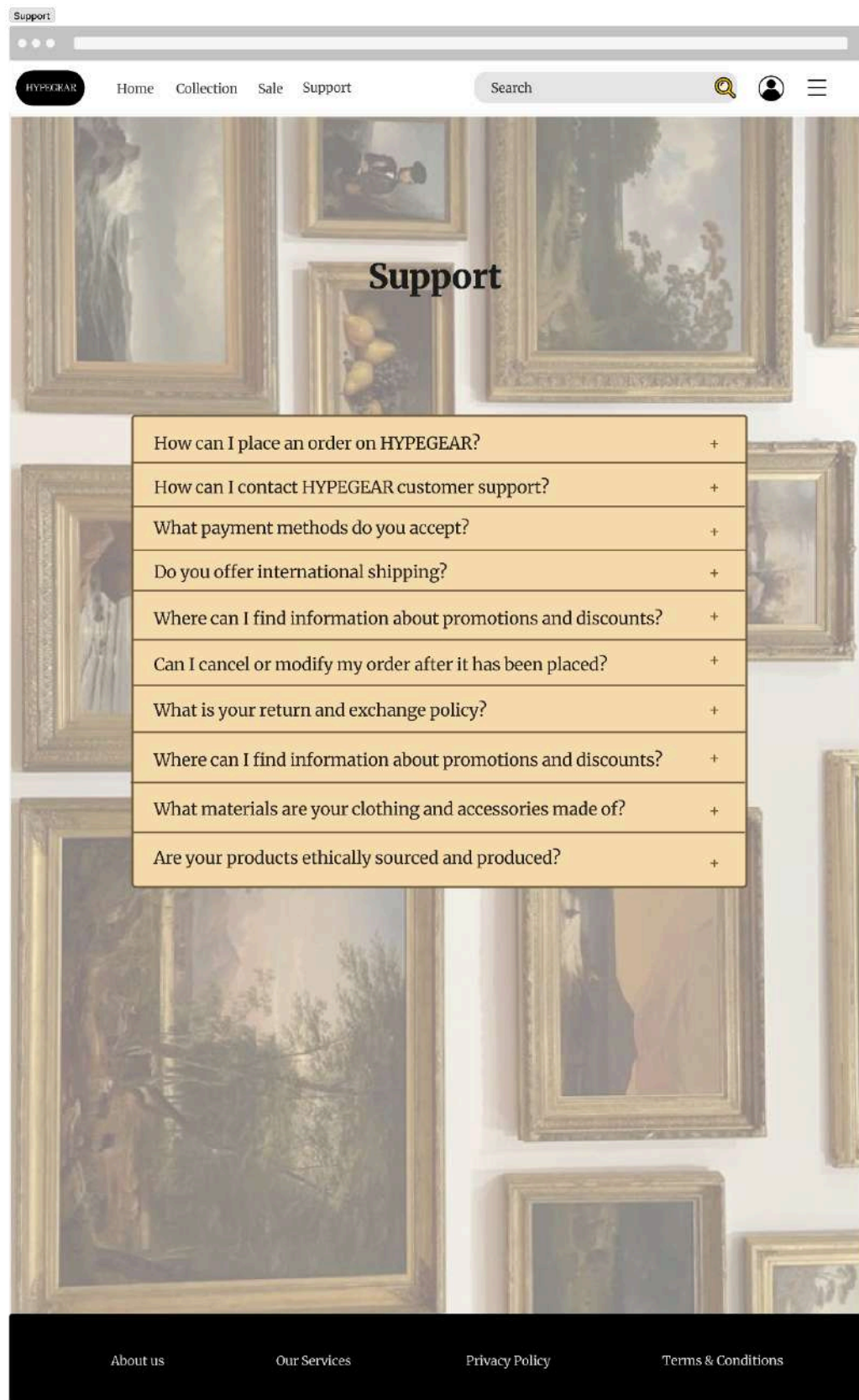
Sale : If users click on sale, you'll be taken to its own page. There, users can explore more about the sale products of that time. For example, in the picture below we have the winter sale and black Friday.



Detail of sale page: It will show information as same as normal detail page but the price will be a discounted price.

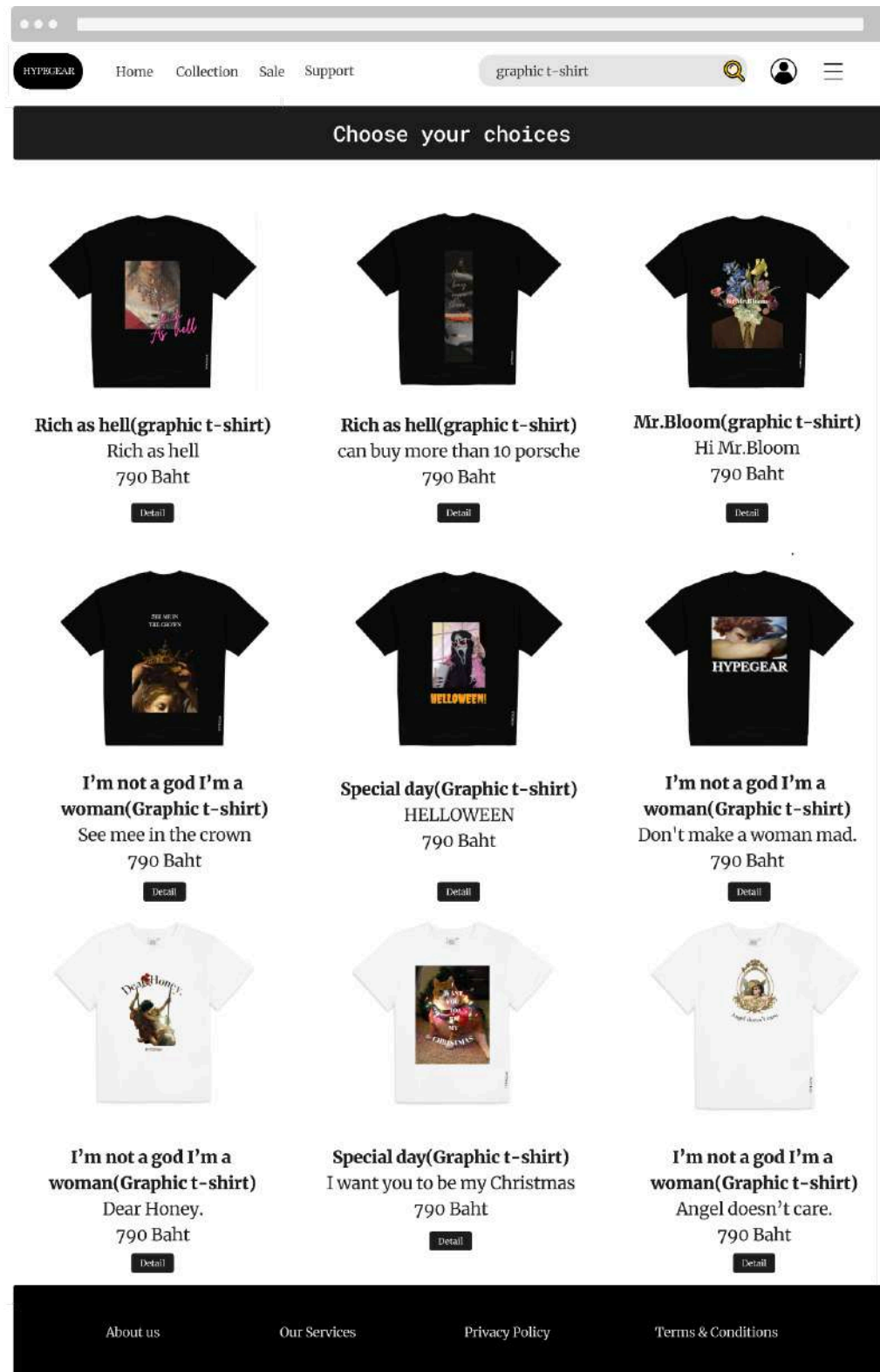


Support : Users can access support by clicking on the navigation bar, where they will find answers to frequently asked questions. If their question isn't addressed, they can easily reach out to our customer support team.



3. Search Page

Users can search for products by entering keywords, and relevant results will be displayed. For instance, if you search for "graphic t-shirt," all matching graphic t-shirts will appear. You can then navigate to the detailed page of a specific item by clicking on detail.



Advance Search : Our website features an advanced search function for finding specific products. By clicking on the search logo, users can access comprehensive product information. They have the flexibility to add only the necessary details; not all input boxes need to be filled. This allows users to search with partial information such as name, price, or material. On the other hand, if a customer enters an ID of the product, the search will display only one product because each ID is unique to each product.

After selecting a product, users can proceed to the Detail Page by clicking on the "Buy now" button. Here, they can access additional information about the product and make a purchase if desired. To return to the homepage, users simply need to click on the "Home" button. If they wish to search for another item, they can click on the search box again.

The screenshot shows a website interface for a t-shirt collection. At the top, there is a navigation bar with links: "HYPERZAR", "Home", "Collection", "Sale", and "Support". A search bar is located on the right side of the navigation bar. Below the navigation bar, on the left, there is a search form with input fields for "Name:", "ID:", "Price:", and "Material:", followed by a "Search" button. On the right, a large image of a black t-shirt with a graphic design is displayed. Below the t-shirt image, the text "information of the Production" is visible. A "Buy now" button is located below the t-shirt image. Below the main product image, there are three smaller images of t-shirts with different designs. Below these, there are three blue boxes containing text descriptions for different collections: "Im not a god Im a human", "Special day", and "Mr.Bloom". Each box has a "see more" button. At the bottom, there is a row of four t-shirt images, each with a "BUY" button below it. The footer of the website contains links: "About us", "Our Services", "Privacy Policy", and "Terms & Conditions".

4. Login Page

Admin&User login page : This page will appear when you click on the "Profile" logo, which is a login area for both users and admins, and individuals who do not have an account can register by clicking on "Sign Up". The username used for login purposes is an email while for admins, it is the admin ID followed by their email. If you prefer not to log in, users can click on "Home" to return to the homepage and view the products.

HYDEGEAR

Home Collection Sale Support

Search

Log In

Username

Password

Login

[Sign up](#)

About us Our Services Privacy Policy Terms & Conditions

User Register's page : This page is for users who want to register and don't have an account yet. They can click on the "Profile" logo then they'll see the login page, where they should click "Sign Up" to access this registration page. Here, users need to fill in all the required information and then click "Create Account". Currently, to log in, users have to click "Profile", but in the future, logging in might happen automatically after creating an account.

HYDEGEAR

Home Collection Sale Support

Search

Register

Firstname

Lastname

Email

Password

DateOfBirth

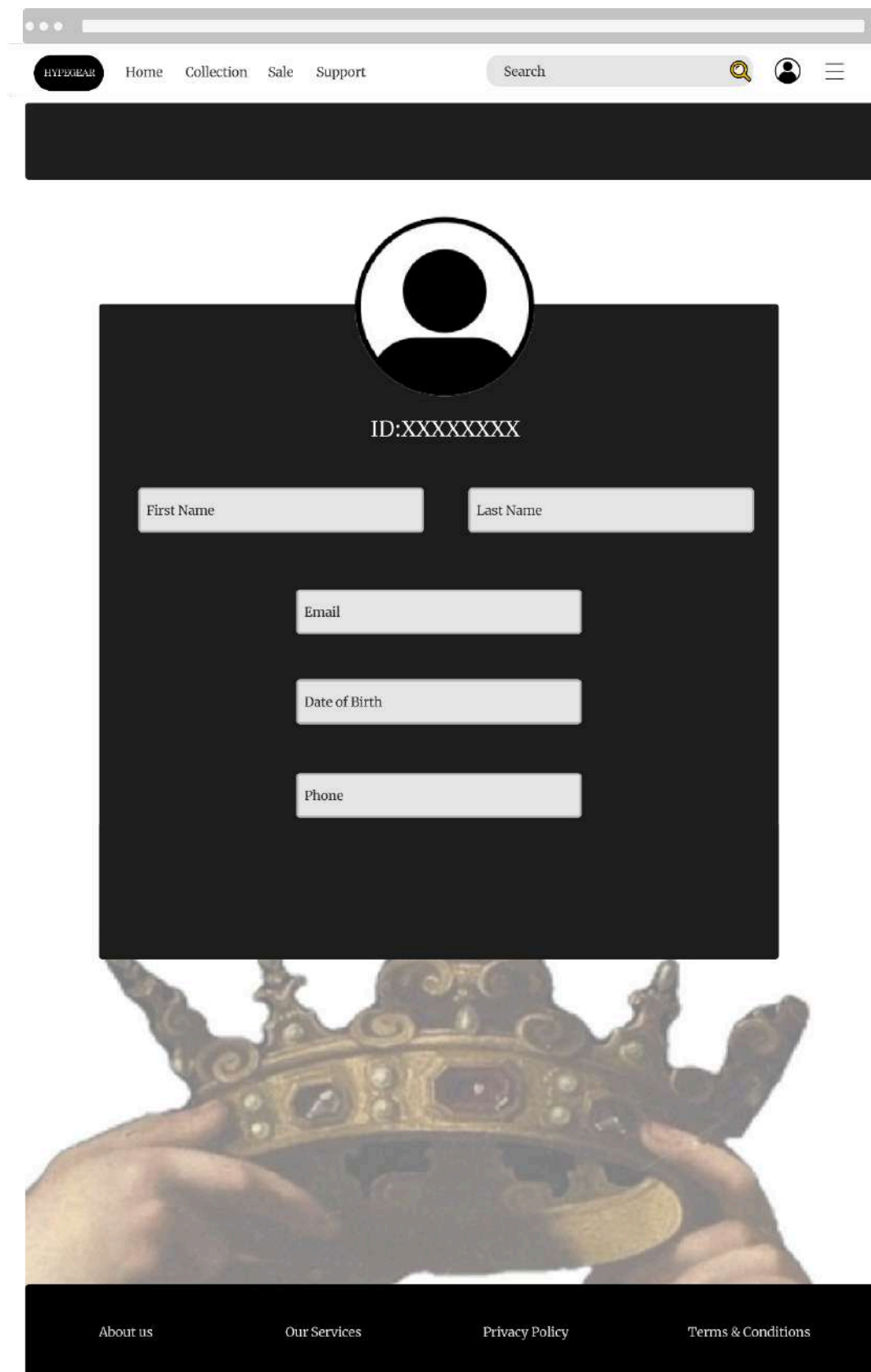
Phone

Create Account

[Already have account?](#)

About us Our Services Privacy Policy Terms & Conditions

Admin or User log in already : This page appears after users or admins have logged in and click again on the "Profile" logo. It displays the account information.



The screenshot shows a web browser window displaying the HYPEGEAR website. The navigation bar includes links for Home, Collection, Sale, and Support, along with a search bar and user profile icons. The main content area features a large circular profile picture placeholder, the text "ID:XXXXXXXX", and several input fields for user information: First Name, Last Name, Email, Date of Birth, and Phone. The background of the profile section is a dark, textured image. The footer contains links for About us, Our Services, Privacy Policy, and Terms & Conditions.

HYPEGEAR Home Collection Sale Support Search

ID:XXXXXXXX

First Name Last Name

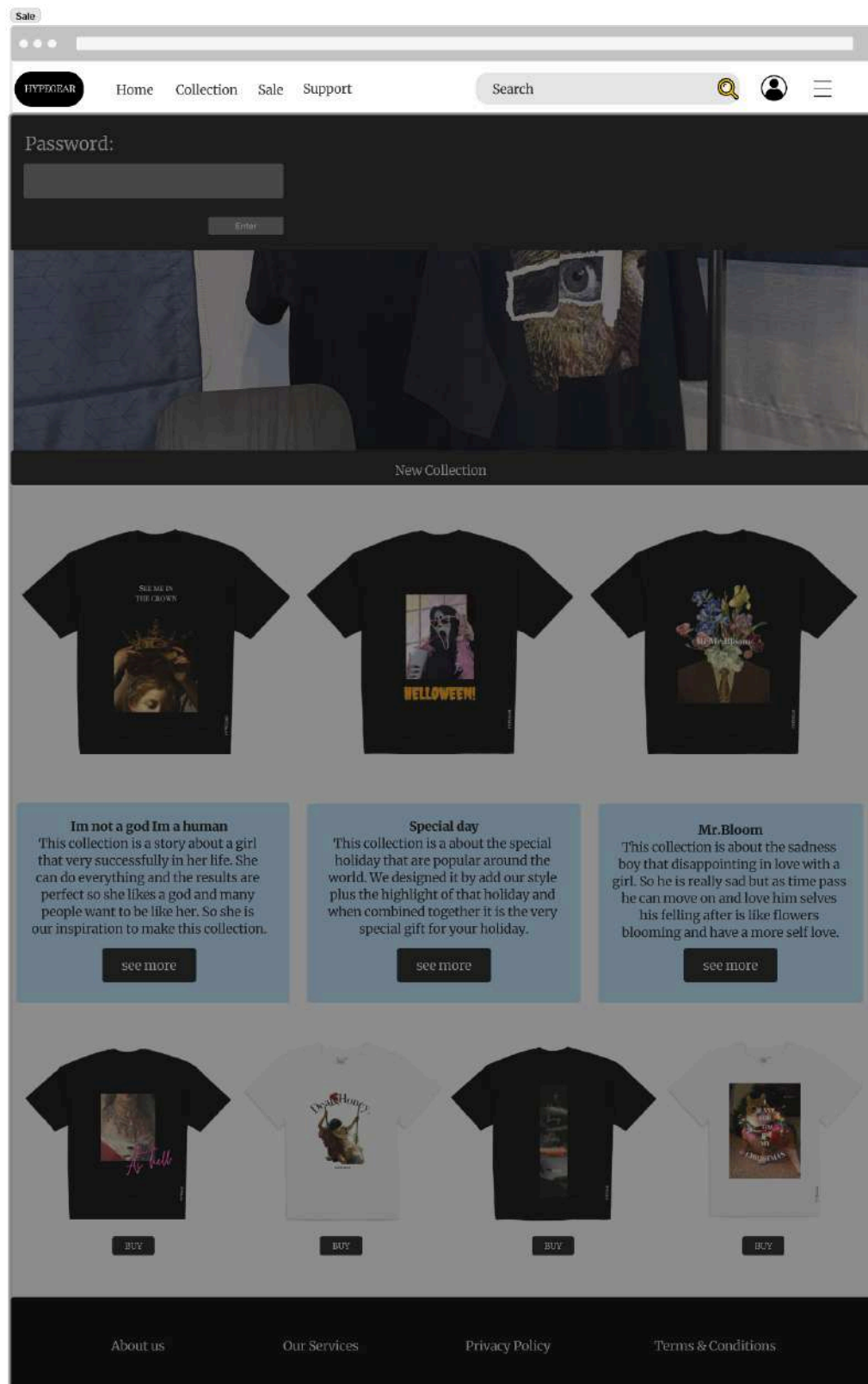
Email

Date of Birth

Phone

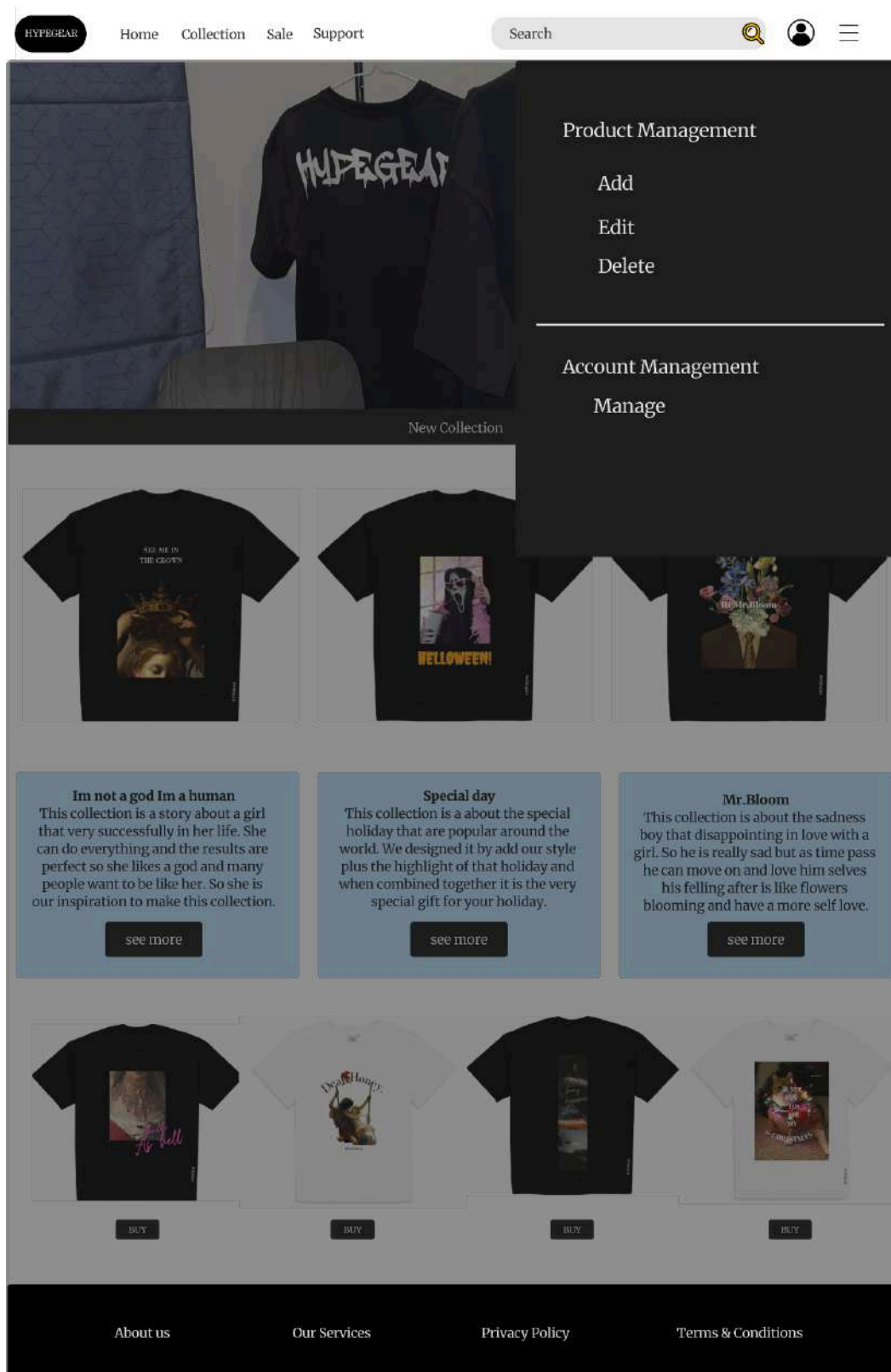
About us Our Services Privacy Policy Terms & Conditions

Admin page : This page opens when clicking on the "Option Bar". It asks for a password to log in as an admin for editing products or user accounts. To return to the homepage, simply click on "Home".



5. Product Management

Next page provides 2 sections, the first section is the Product Management section allowing admin to add, edit, and delete the product. And another section is Account Management section, admin can click to manage the user accounts.




Add page : The Add page allows the administrator to add the new product by adding pictures of the product with the button 'Add image', putting the name and ID of the product, price, collection, description, material of the product, and the amount to add for each size of product. After the administrator finishes this section, press 'Add product' to save this product in the system.

HYPERGEAR

HomeCollectionSaleSupport

Search

Add Product



Add image

Name: HELLOWEEN(Graphic t-shirt)ID: cs4ghr

Price: 790 Baht

Collection: Special day

Description: Graphic black tee

Material: Cotton 100%

Amount

+

-

S

M

L

XL

(Num)

(Num)

(Num)

(Num)

Add product

About us

Our Services

Privacy Policy

Terms & Conditions

Search for Edit page : This page allows the administrator to search for the ID of the product to edit. After searching the ID, it will display the information about the product and click 'Edit' to edit products, and 'Delete' to delete products.

Search ID to find the User

HYPERGEAR Home Collection Sale Support Search

Search ID

ID:XXXXXXXX

First Name Last Name

Email

Date of Birth

Phone

Update Delete


About us Our Services Privacy Policy Terms & Conditions

Edit page : After clicking 'Edit,' administrators are directed to the edit page where they can modify product details. This includes changing images, name, ID, price, collection, description, and materials. They can also manage the inventory for each size. Upon completion, administrators simply click 'Update product' to save the changes in the system.

Website Wireframe

HYPERGRAR Home Collection Sale Support Search

Edit Product



Name: ID:

Price:

Collection:

Description:

Material:

Amount:

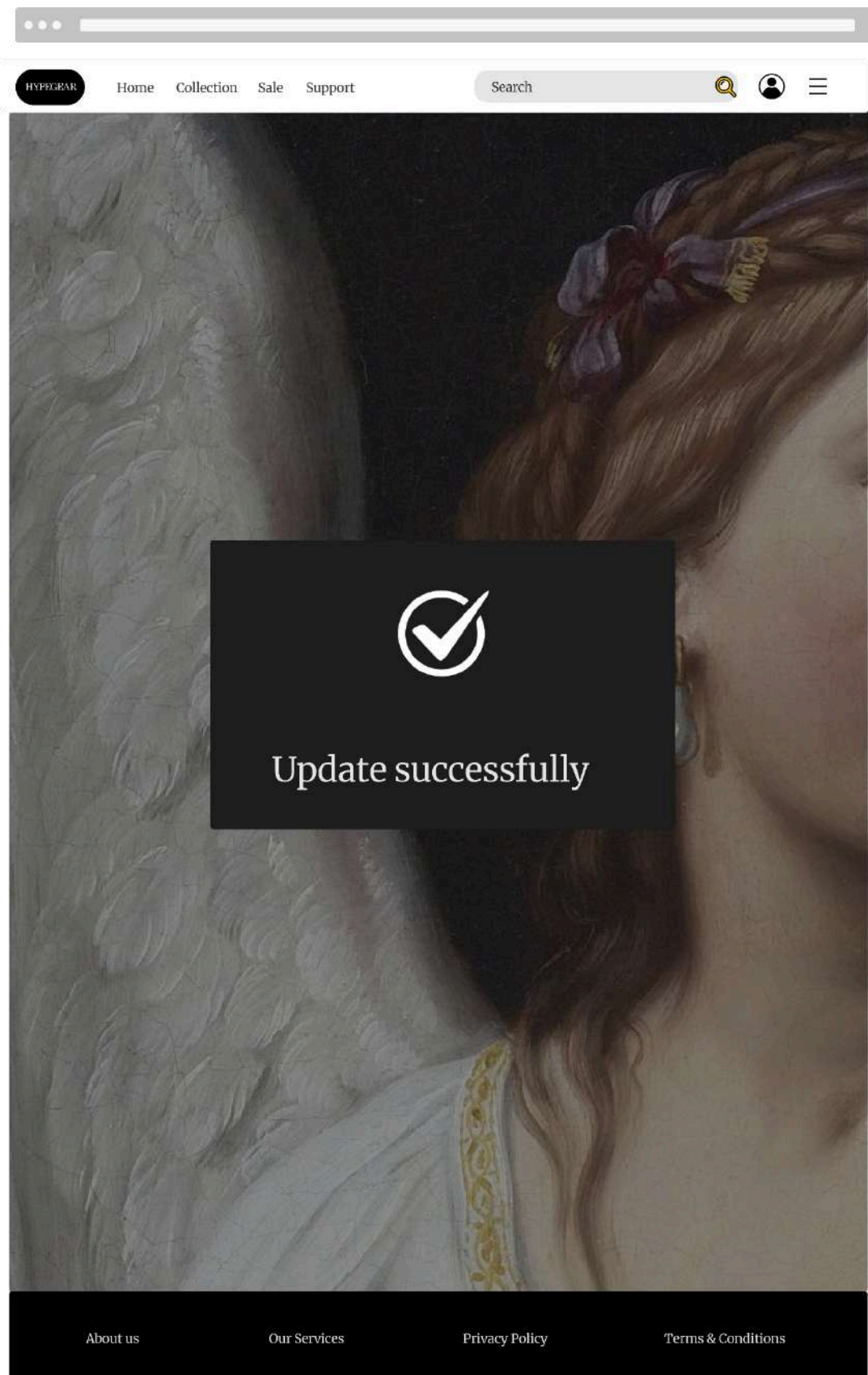
+
-

S	M	L	XL
↓	↓	↓	↓
(Num)	(Num)	(Num)	(Num)

Update product

About us Our Services Privacy Policy Terms & Conditions

Edit success page : When administrators successfully update the products, a confirmation message 'Update successfully' is displayed to ensure the changes have been saved.




Delete page : After clicking the delete button, administrators are shown all information referring to the product, including images, name, ID, price, collection, description, and materials. If the administrator confirms their intention to delete the product, they can proceed by clicking on 'Delete product' to remove it from the system.

Website Wireframe

HYPERGEAR Home Collection Sale Support Search

Delete Product



Add image

Name: ID:

Price:

Collection:

Description:

Material:

Amount :

+
-

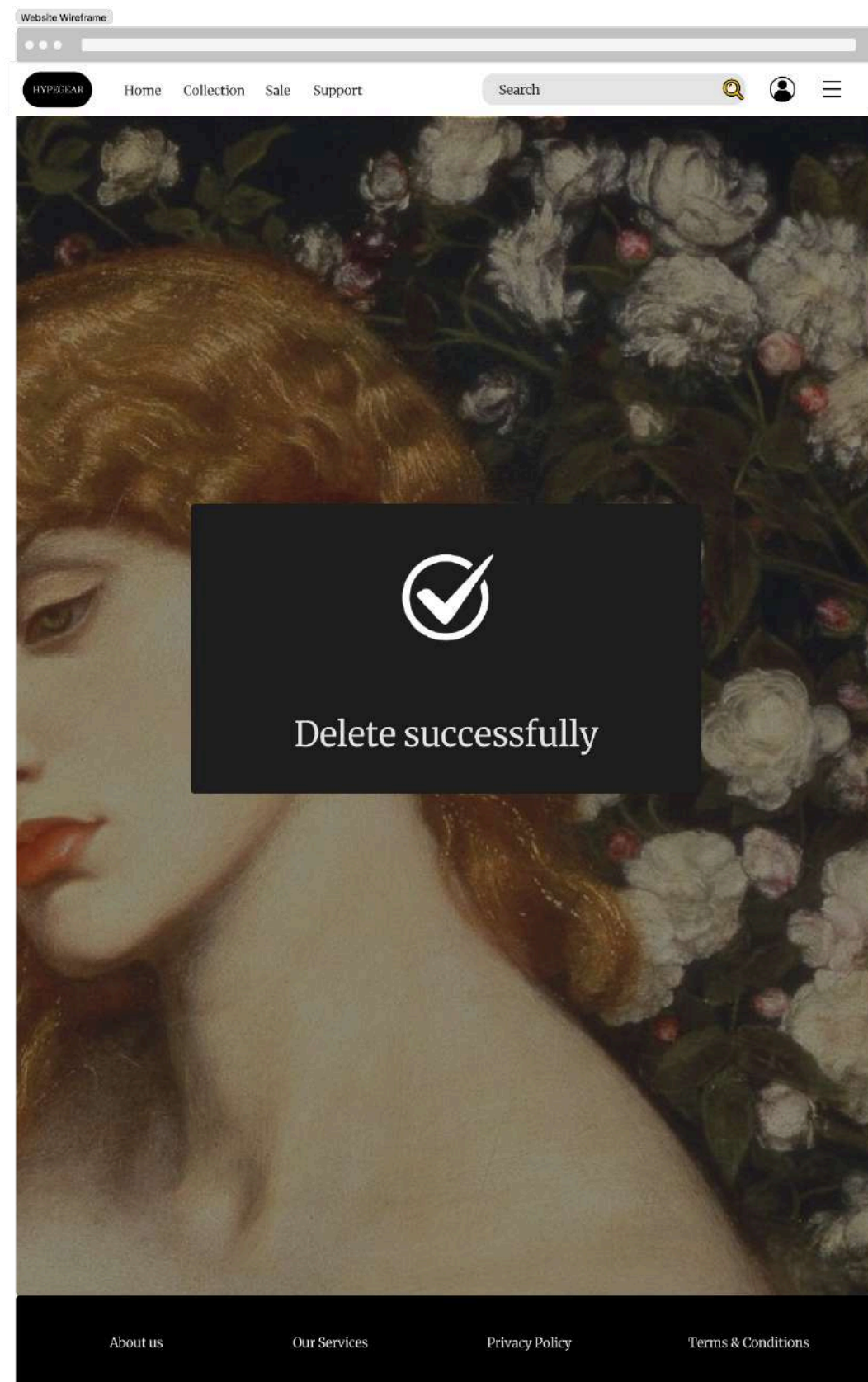
S M L XL

(Num) (Num) (Num) (Num)

Delete product

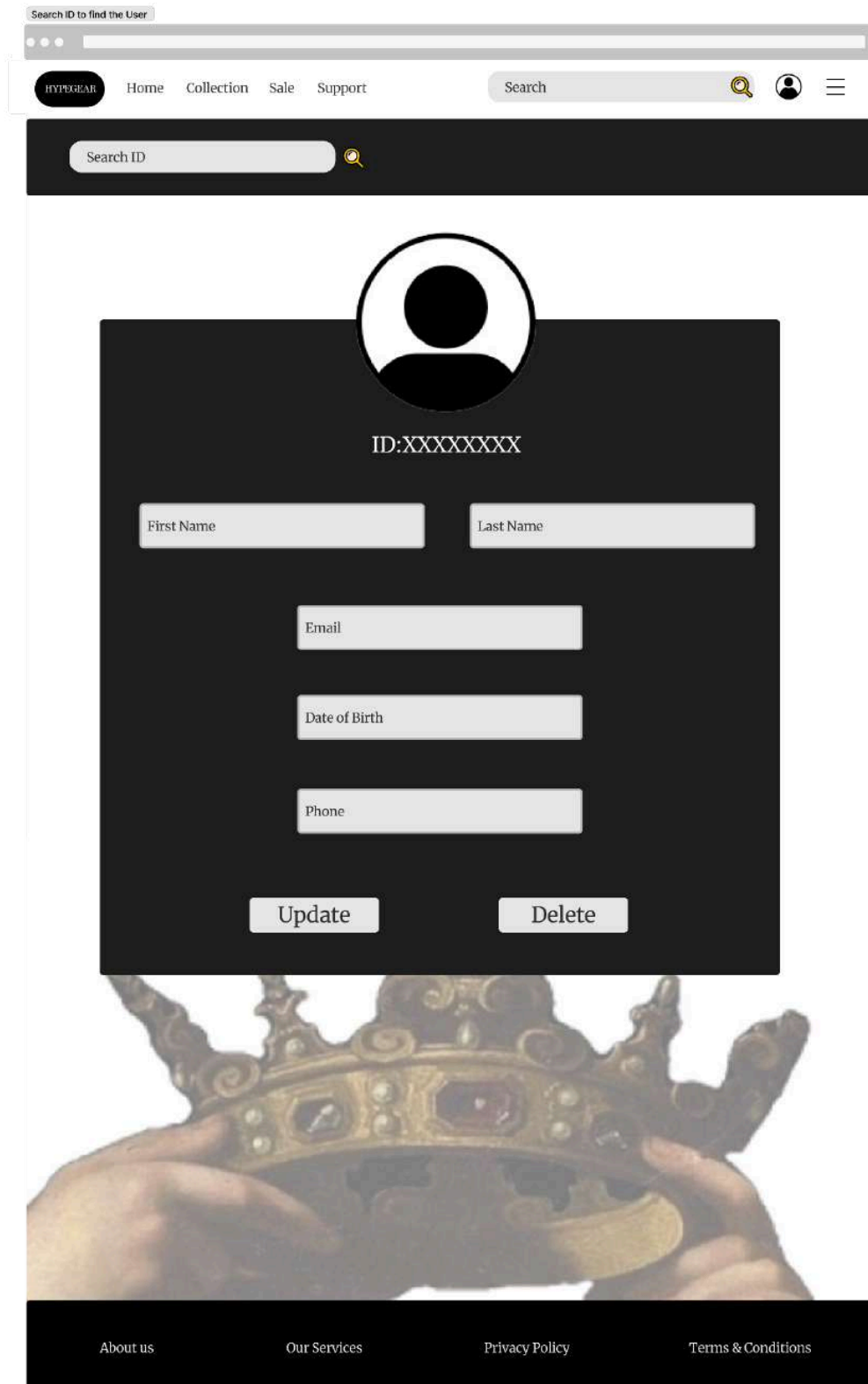
About us Our Services Privacy Policy Terms & Conditions

Delete success page : When administrators successfully update the products, a confirmation message 'Delete successfully' is displayed to ensure the changes have been saved.



6. User Account Management

To manage the user's account page, administrators utilize a search function to locate users by their ID. Once the correct ID is found, administrators can access and review the user's information, including their first name, last name, email address, date of birth, and phone number.



The screenshot displays a web application interface for user account management. At the top, a navigation bar includes the 'HYPERGEAR' logo and links for 'Home', 'Collection', 'Sale', and 'Support'. A search bar is positioned on the right side of the navigation bar. Below the navigation bar, a dark header section contains a 'Search ID' input field with a magnifying glass icon. The main content area features a large circular profile picture placeholder, followed by the text 'ID:XXXXXXXX'. Below this, there are five input fields for user information: 'First Name', 'Last Name', 'Email', 'Date of Birth', and 'Phone'. At the bottom of the form, there are two buttons: 'Update' and 'Delete'. The background of the page shows a faint image of hands holding a crown. The footer contains links for 'About us', 'Our Services', 'Privacy Policy', and 'Terms & Conditions'.

Search ID to find the User

HYPERGEAR Home Collection Sale Support Search

Search ID

ID:XXXXXXXX

First Name Last Name

Email

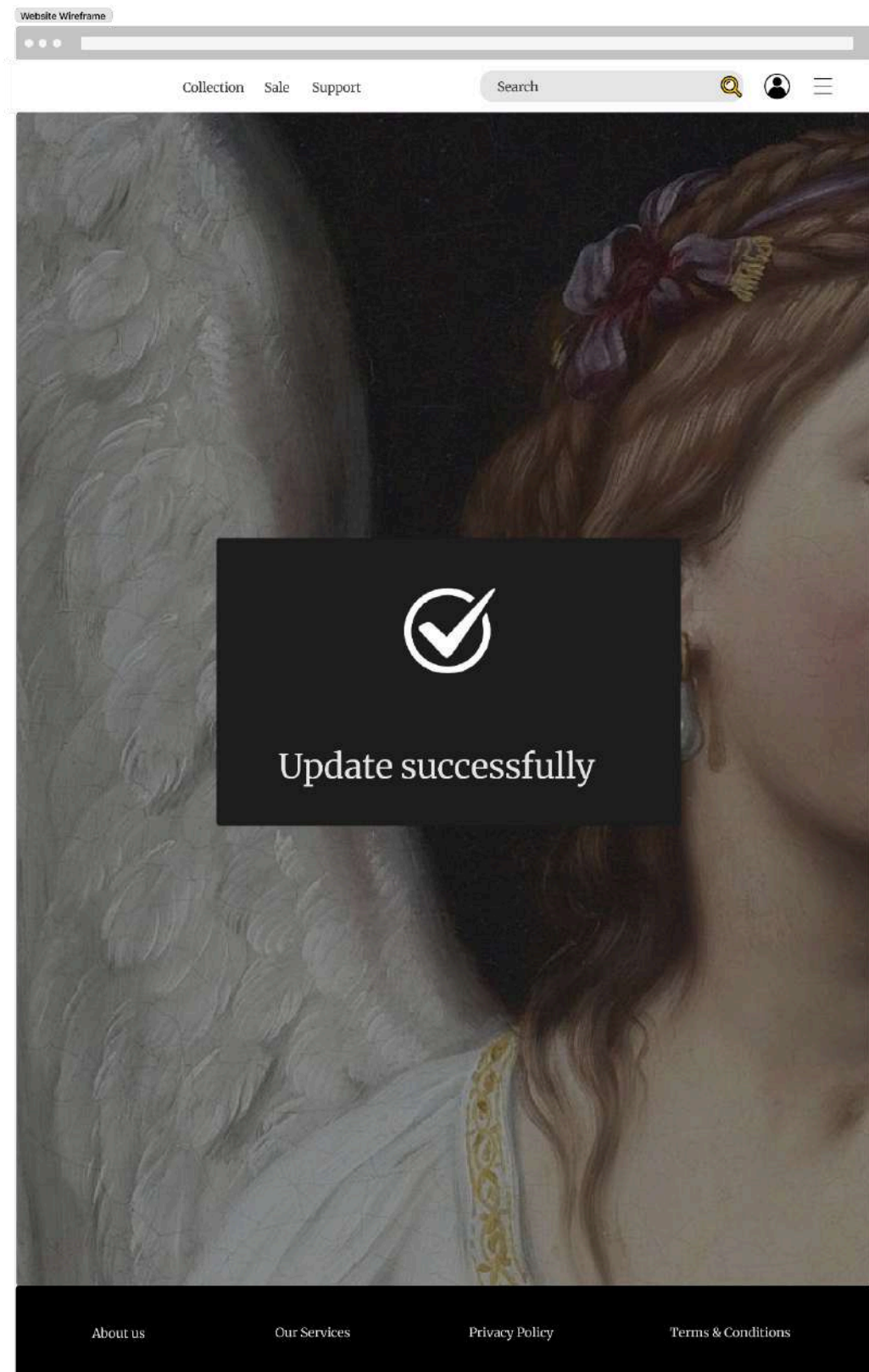
Date of Birth

Phone

Update Delete

About us Our Services Privacy Policy Terms & Conditions

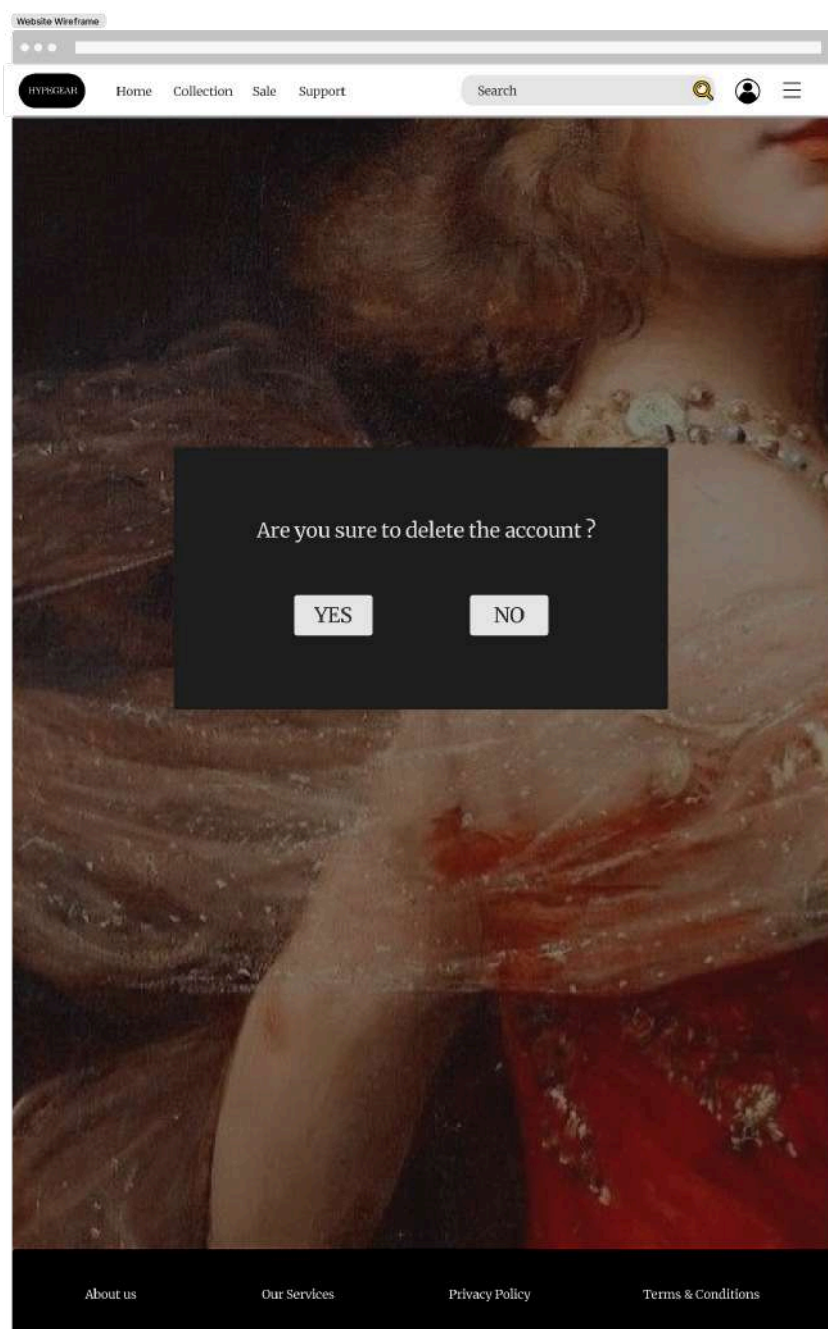
Update : Administrators can update user profiles by modifying first name, last name, email, date of birth, or phone number. Upon completing the updates, administrators simply click 'Update' to ensure the user's information is promptly updated in the system.



Delete : This page will prompt the Administrator to confirm the deletion by asking, "Are you sure you want to delete the account?" It will provide options with buttons labeled 'YES' and 'NO' to allow the Administrator to reconsider their decision before proceeding

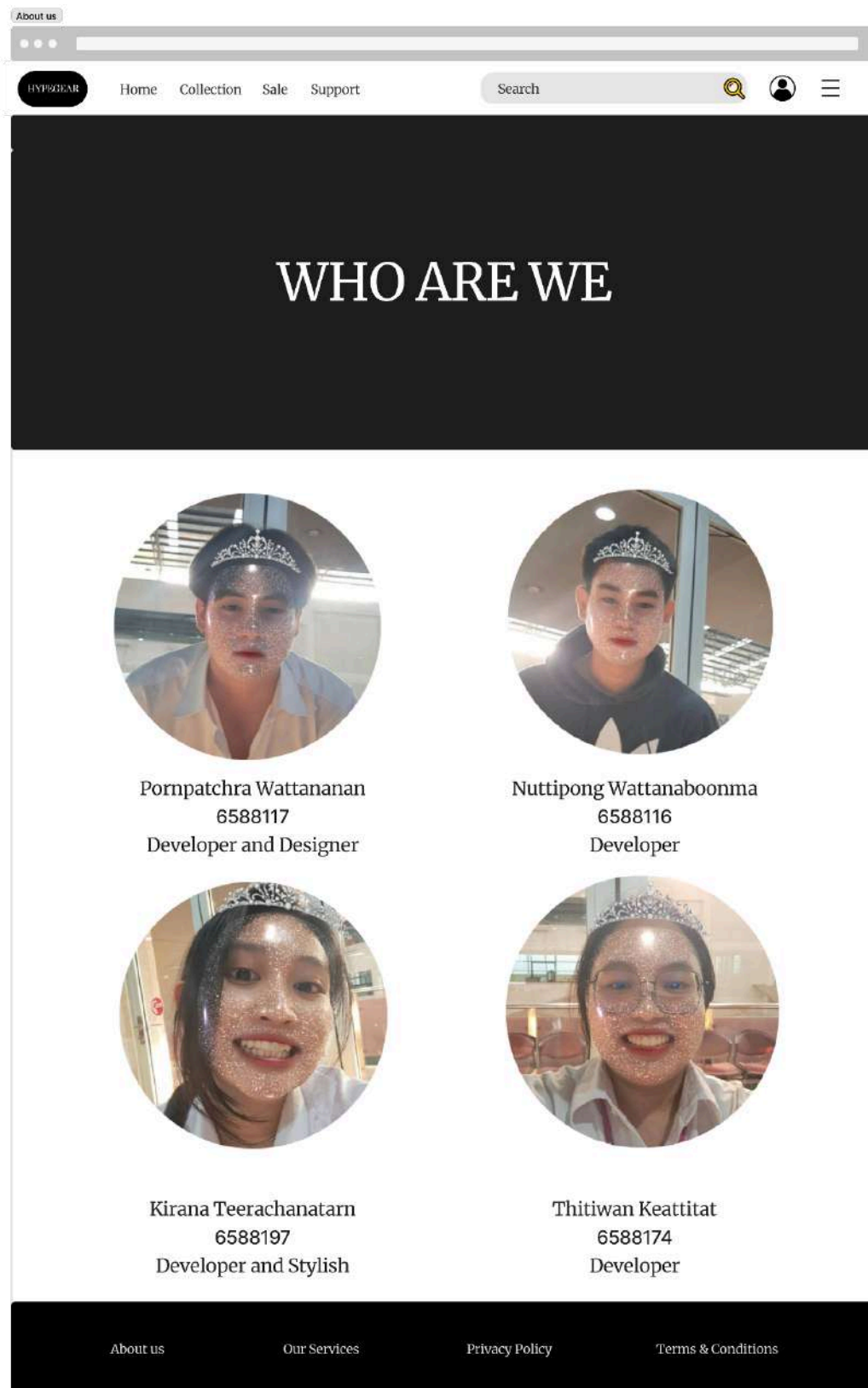
YES button : When administrators successfully delete a user's account, a confirmation message stating 'Delete successful' is displayed, ensuring that the action has been completed and the changes have been saved.

NO button : If the administrator clicks the 'NO' button, they will be redirected back to the search page where they can continue to look up the user's information by ID.



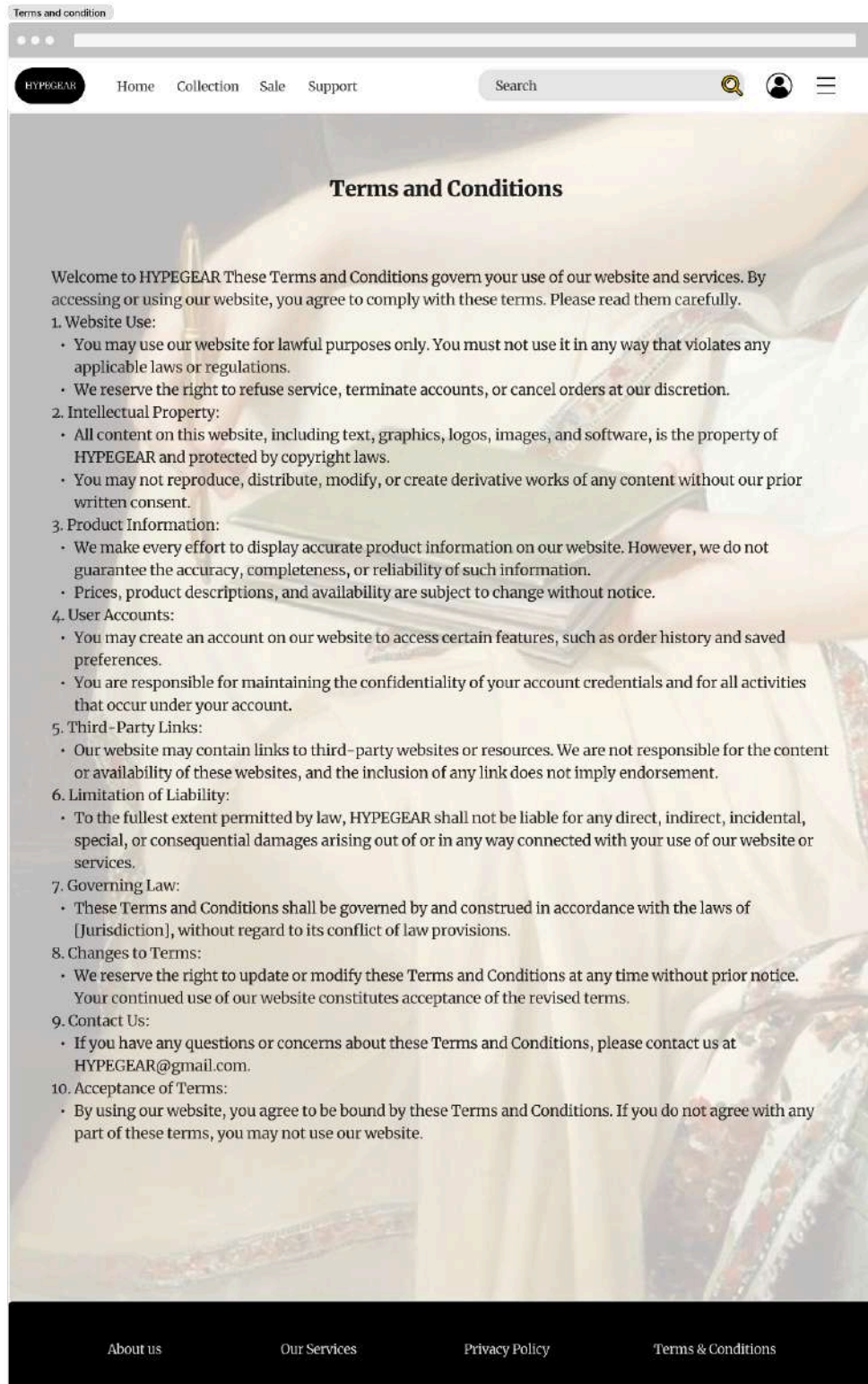
7. Team Page

This page is about who are the members of HYPEGEAR team and the role of each person.



1.8 Additional Page

Our service page : Our Services page is about the offer and the terms governing your use of these services. It is about Online Store, Account Registration, Newsletter Subscription, Customer Support, Product Customization, Social Media Engagement, Events and Promotion, Feedback and Reviews, Modification to services, and Acceptance of service.



Privacy Policy page : Our privacy policy page is about how we manage users data and control your interactions with our brand. Find more about our dedication to safeguarding your privacy and the personal information we gather, making sure that everything is transparent at all times. Learn about the goals of data collecting, such as marketing communication, order processing, and customer assistance. You may be confident that we never sell your information to outside parties without getting your permission. Recognize your rights over your data and learn how to use them. We place a high priority on data security and take strong precautions to keep your information safe from unwanted access. Regular upgrades guarantee adherence to changing regulations.



Terms & Conditions page : Terms & Conditions page is about respecting copyright and using content responsibly. We strive for accurate product info but can't guarantee it. Create an account for access, but keep credentials confidential. We're not liable for damages from website use. Links to third-party sites are provided but not endorsed. Changes to terms may occur without notice.

Terms and condition

HYPEGEAR Home Collection Sale Support Search

Terms and Conditions

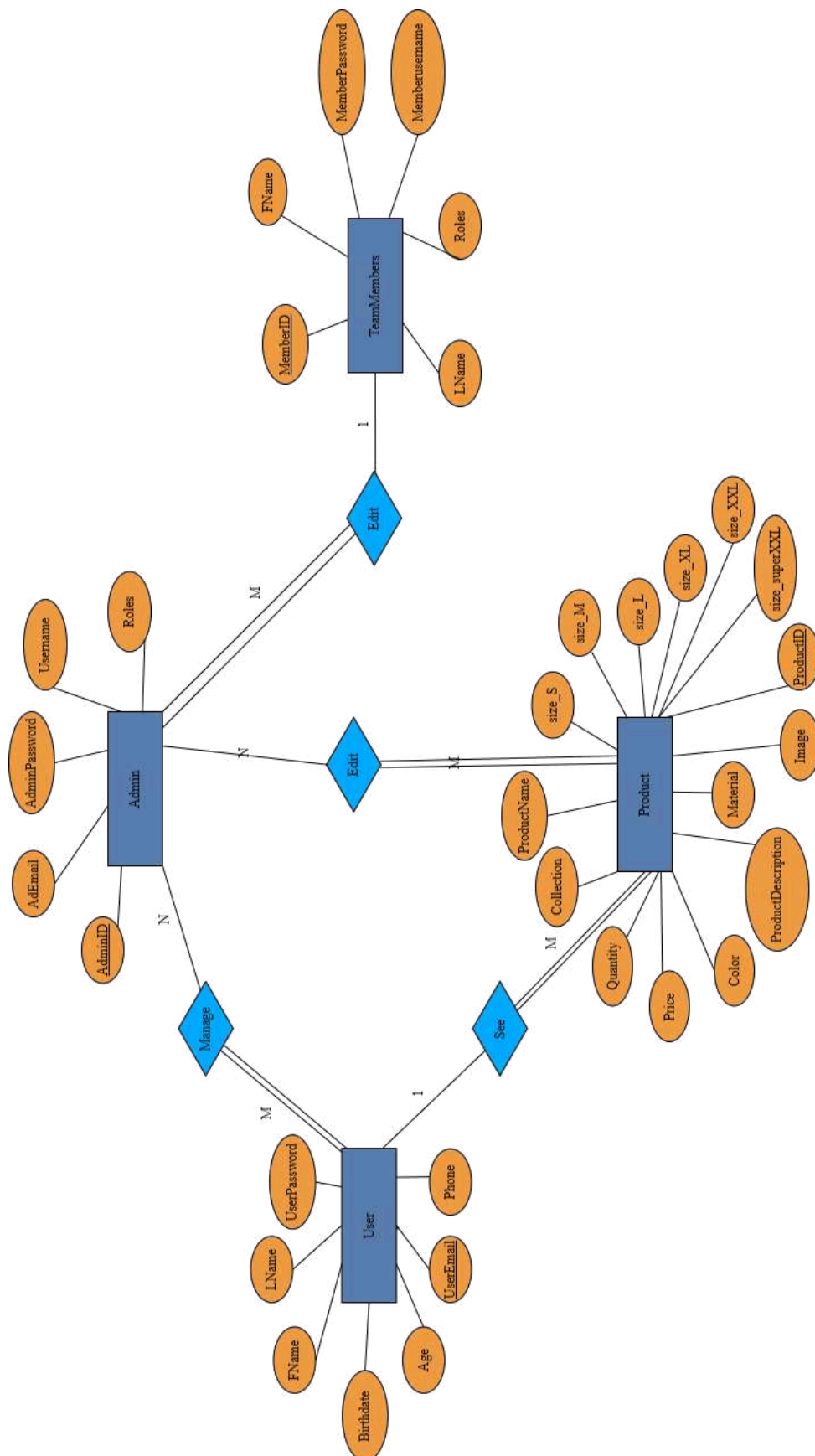
Welcome to HYPEGEAR These Terms and Conditions govern your use of our website and services. By accessing or using our website, you agree to comply with these terms. Please read them carefully.

- 1. Website Use:**
 - You may use our website for lawful purposes only. You must not use it in any way that violates any applicable laws or regulations.
 - We reserve the right to refuse service, terminate accounts, or cancel orders at our discretion.
- 2. Intellectual Property:**
 - All content on this website, including text, graphics, logos, images, and software, is the property of HYPEGEAR and protected by copyright laws.
 - You may not reproduce, distribute, modify, or create derivative works of any content without our prior written consent.
- 3. Product Information:**
 - We make every effort to display accurate product information on our website. However, we do not guarantee the accuracy, completeness, or reliability of such information.
 - Prices, product descriptions, and availability are subject to change without notice.
- 4. User Accounts:**
 - You may create an account on our website to access certain features, such as order history and saved preferences.
 - You are responsible for maintaining the confidentiality of your account credentials and for all activities that occur under your account.
- 5. Third-Party Links:**
 - Our website may contain links to third-party websites or resources. We are not responsible for the content or availability of these websites, and the inclusion of any link does not imply endorsement.
- 6. Limitation of Liability:**
 - To the fullest extent permitted by law, HYPEGEAR shall not be liable for any direct, indirect, incidental, special, or consequential damages arising out of or in any way connected with your use of our website or services.
- 7. Governing Law:**
 - These Terms and Conditions shall be governed by and construed in accordance with the laws of [Jurisdiction], without regard to its conflict of law provisions.
- 8. Changes to Terms:**
 - We reserve the right to update or modify these Terms and Conditions at any time without prior notice. Your continued use of our website constitutes acceptance of the revised terms.
- 9. Contact Us:**
 - If you have any questions or concerns about these Terms and Conditions, please contact us at HYPEGEAR@gmail.com.
- 10. Acceptance of Terms:**
 - By using our website, you agree to be bound by these Terms and Conditions. If you do not agree with any part of these terms, you may not use our website.

About us Our Services Privacy Policy Terms & Conditions

Task 2: Data Model

ERD



Entity

1. Admin

Attribute: AdminID, Username, Roles, AdEmail, AdminPassword

- AdminID: This attribute is a primary key of this entity enabling the creation of unique emails that distinguish administrators.
- Username: As a name of admin
- Roles: As a duty of admin in our website.
- AdEmail: As email of Admin.
- AdminPassword: As a Password for login to account.

Relation:

- Many admin can manage many users' accounts.
- Many admin can edit many products.
- Many admin gets edited by one TeamMembers.

2. User

Attribute: UserEmail, FName, Lname, UserPassword, Birthdate, Phone, Age

- UserEmail: As Name for login Username of user and it is a primary key of this entity.
- FName: This attribute keeps the first name of the user and shows on the view page.
- Lname: This attribute keeps the last name of the user and shows on the view page.
- UserPassword: This attribute keeps the password of the user and keeps it in the database.
- Birthdate: This attribute keeps information about birth of the customer and age
- Phone: The contact of the user phone number.
- Age: This attribute keeps the age of the user and shows on the view page.

Relation:

- One user can see many products.
- Many users get managed by many admin.

3. Product

Attribute: ProductID, ProductName, Collection, Material, price, ProductDescription, size_S, size_M, size_L, size_XL, size_XXL, size_superXL, Quantity, Color, Image

- ProductID : is a primary key of this entity enabling the creation of unique emails that distinguish administrators.
- ProductName: is the attribute that shows the name of the product.
- Collection: is the attribute that collects the same style of clothes together for example Mr.Bloom collection.
- Description: is the attribute that describes the products.
- Material: is the attribute that tells users what that cloth is made of.
- Price: is the attribute that describes the price of a product.
- ProductDescription: is the attribute describing the information of the product.
- size_S: is the attribute to keep the data in number. It describes the size of the product in stock.
- size_M: is the attribute to keep the data in number. It describes the size of the product in stock.
- size_L: is the attribute to keep the data in number. It describes the size of the product in stock.
- size_XL: is the attribute to keep the data in number. It describes the size of the product in stock.
- size_XXL: is the attribute to keep the data in number. It describes the size of the product in stock.
- size_superXL: is the attribute to keep the data in number. It describes the size of the product in stock.
- Quantity: is the attribute that shows total product in stock.
- Color: is the attribute that shows the color of the products.
- Image: is the attribute that keeps the url of the image and shows the image of that product.

Relation:

- Many Products can be edited by many admin .
- Many products can be seen from one user(per account).

4. TeamMembers

Attribute: MemberID, Memberusername, MemberPassword, FName, LName, Roles

- MemberID: is a primary key of this entity enabling the creation of unique emails that distinguish administrators.
- Memberusername: This attribute keeps the username of the members and shows on the view page.
- MemberPassword: This attribute keeps the password of the members and shows on the view page.
- FName: This attribute keeps the first name of the member and shows on the view page.
- LName: This attribute keeps the last name of the member and shows on the view page.
- Roles: is the attribute that tells the roles of members.

Relation: One team member can edit many admins.

Relation

- “Edit” relation between Product and Admin: It includes the Update,Delete,Add function in our website.
- “Edit” relation between TeamMembers and Admin: It includes the Update,Delete,Add function in our website.
- “Manage” relation between User and Admin: It includes the Update,Delete function in our website.
- “See” relation between User and Product: Users can only see details of product in our website.

Phase II

Project Overview

!!!* Before test please set page size to 1400px * 800px.*!!!

Home page

This is the first page that users can see when they are on our website. This page shows some products and collections of products (t-shirts). Users will see the collection of products, and they can click see more to see the story and details of those products in that collection. This page also has a navigation bar that links to many pages and features, including a search and advance search button, an account icon to click to register and log in, and a menu button on the top right for administrators for user account management, product management, and admin management. And also link to the “guess the new collection” page, sale page, support page, and home page. Users can go back to the home page by clicking the home button, or if they click on the logo, they will also go back to the home page.

Register page(sign up)

This page will be shown to users who want to register on the website. Users need to fill in their first name, last name, email, password, birthdate, age, and phone number to register. After registration, the information will be kept in the database, and the administrator will see it on the user list page.

Login page

This page will show the users who have already registered and want to log in to our website. Users need to type their email and password to log in.

Search page

On this page, users can search for products by name, collection, or color, but they can only choose one type. After the search, products will show based on what users searched for. When users want to buy the products, just click on the image of that product or any space in the box around the product, and it will go to the product detail page, where they can select the size and amount they want to buy.

Advance search page

On this page, users can search for specific products by texting their name, collection, and color. After the search, only one product that matches the searched information will show up on the right side. Users can click the buy button, and it will go to the product detail page, where they can select the size and amount they want to buy.

Detail page

This page will show the details of the products including the name of the product, ID of the product, price of the product, collection of the product, product description, users can choose the size and quantity of the product.

Admins list page

This page is for administrators. It will show the list of the admins. It will show the admin name, admin ID, admin role, and admin email. Administrators can edit the information by clicking the edit button on the right side, administrators can update and delete every information and it will show the new information at the admins list page.

Users list page

This page is for administrators. It will show the list of the users that register to our website. It will show user firstname, user lastname, user email, user age, user phone number, and user email. Administrators can edit the information by clicking the edit button on the right side, administrators can update and delete every information and it will show the new information at the users list page.

Product list page

This page is for administrators. It will show all the products that we have, including product ID, product name, collection, material, price, product description, size, quantity, color, and image. Administrators can edit the information by clicking the edit button on the right side. Administrators can update, delete every information, and can add products. Everything that has changed will show the new information on the product list page.

Home page and the logo

When users click on these two it will directly go back to the home page.

Guess the new collection

This page is for users to guess the new style or collaboration in a new collection.

Sale page

This page will show the products that were on sale at that time but now nothing is on sale.

Support page

This page will show many common questions that users commonly ask.

About us page

This page will show information including a picture of team members, name, student ID, and roles of every team member.

Our Services

This page will show the information about our service we offer and the terms governing user use of the services.

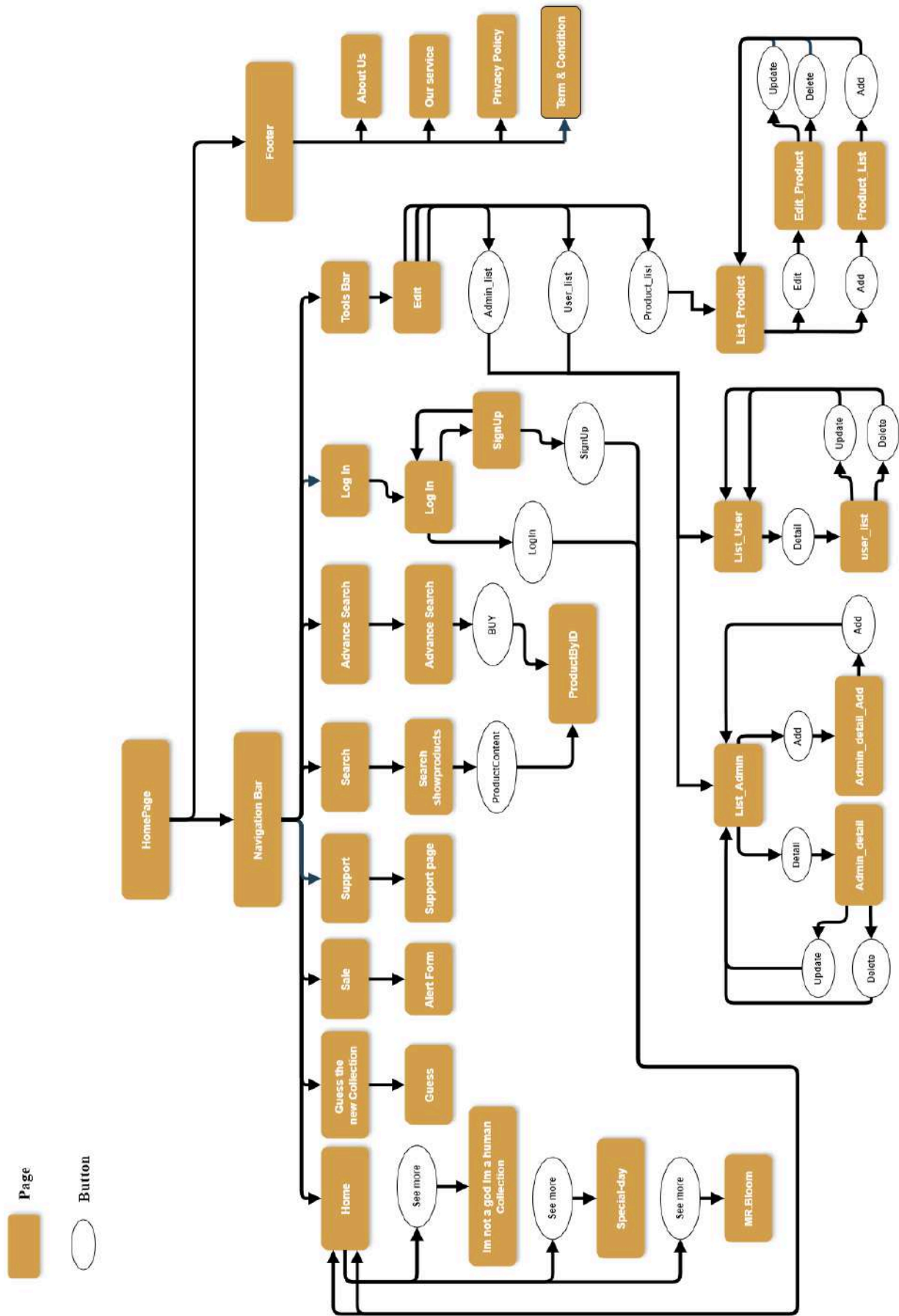
Privacy Policy page

This page will show the privacy policy of our website that shows how we collect, use, and safeguard your data.

Terms & Conditions page

This page will show terms and conditions that govern user use of our website. By accessing or using our website, users agree to comply with these terms.

Navigation Diagram of your web application



Details of your web application and code

Every page has a navigation bar that can link to every page.

Home_page.html

```
<!DOCTYPE html>
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Home</title>
```

```
<link rel="stylesheet" href="/homepage.css">
```

```
<script src="/Nav/nav.js"></script>
```

This meta tag sets the viewport to the width of the device and sets the initial zoom level to 1.0. It's important for responsive design, ensuring that the page displays properly on different devices and screen sizes. The title is profile, this is the title of the web page, which appears in the browser's title bar. This page links an external CSS file named homepage.css to style the content of the page and it links an external JavaScript file named nav.js it's used for the navigation function.

```
<div style="width: 1400px; height: 650px; background-image: url(Home/Home_page1.png); background-size: cover; ">
```

```
<div class = "overlay_text"><b>HYPEGEAR</b><br><div style="font-size: 30px;"> where fashion meets excitement</div> </div>
```

```
</div>
```

This creates a banner section with a background image and overlay text displaying the brand name HYPEGEAR and a text Where Fashion Meets Excitement. The background size is set to cover the entire area.

```
<div class="btn_new_collection black"><p>New Collection</p></div>
```

This creates a label New Collection. It stands at the top of the new collection to tell users that this is a new collection.

```
<div>
```

```
<div class="spare_picture">
```

`<img src = "Home/Crown_black_tee.png" class="top left image
img_home_size_2">`

`<img src = "Home/Halloween.png" class="top middle image
img_home_size_2">`

`<img src = "Home/Bloom.png" class="top right image
img_home_size_2">`

`</div>`

`
`

`<div class="box">`

`<section class="textbox">`

`<p class="white textbox_text">`

`<h2>Im not a god Im a human</h2>`

This collection is a story about a girl that very successfully in her life. She can do everything and the results are perfect so she likes a god and many people want to be like her. So she is our inspiration to make this collection.

`</p>`

`
`

`<button
class="btn_seemore black">See more</button> `

`</section>`

`<section class="textbox">`

`<p class="white ">`

`<h2>Special day</h2>`

This collection is about the special holiday that are popular around the world. We designed it by add our style plus the highlight of that holiday and when combined together it is a very special gift for your holiday.

`</p>`

<button class="btn_seemore black">See
more</button>

</section>

<section class="textbox">

<p class="white textbox_text">

<h2>Mr.Bloom</h2>

This collection is about a sadness boy that disappointing in love with a girl. So he is really sad but as time pass he can move on and love him selves his felling after is like flowers blooming and have a more self love.

</p>

<button class="btn_seemore black">See
more</button>

</section>

</div>

This section contains three images displayed in a line. These are different collections. Each section contains a name of the collection, short details and a See more button which links to more details about the 3 collections.

<div class="spare_picture">

<img src
="https://drive.google.com/thumbnail?id=14AbGzclpw7Yx-3ZczZCXeh59PlJAo
mxR" id="PID010">

<img src =
"https://drive.google.com/thumbnail?id=1M5CKN_bnyam3EzKY8V-g7UIrDObx
pAWA" id ="PID001" >

```
<img src =  
"https://drive.google.com/thumbnail?id=1s5IRF4uGBP8_9xZA8w6YRsD21fsIm  
Uc_" id="PID011">
```

```
<img src =  
"https://drive.google.com/thumbnail?id=1IZNYEtBI1p0iW11PFjUHz1XvMoLijsl  
k" id="PID006">
```

```
</div>
```

This section shows four images displayed in a row. These are to show some products to users to see.

```
<footer-component></footer-component>
```

This is the bottom part of the website. It links to four pages including About us, Our Services, Privacy Policy, Term & Conditions.

Guess.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Guess</title>
```

```
<script src="/Guess.js"></script>
```

```
<script src="/Nav/nav.js"></script>
```

```
</head>
```

```
<body>
```

```
<navbar-component></navbar-component>
```

```
<div id="Foodlists">
```

```

</div>

<footer-component></footer-component>

</body>

<style>

  div img{

    width: 1400px;

    height: 800px;

    display: flex;

  }

</style>

</html>

```

This page includes a navigation bar, a div with the id Foodlists (which for displaying pizza), and a custom footer component. This sets some CSS styles inline. It applies to all images inside div elements, setting their width to 1400 pixels, height to 800 pixels, and display style to flex.

Guess.js

```

document.addEventListener("DOMContentLoaded", function() {

  const Api_key = "84969907b9624629a252dbdd1bc8e0e4";

  var name = "Pizza";

  // console.log(name.value);

  // The Api key for go in the website is apiKey

  let want =
  "https://api.spoonacular.com/recipes/complexSearch?apiKey="+Api_key+"&que
  ry="+name;

  fetch(want)

```

```

.then((result) => result.json())

.then((data) => {

    console.log(data);

    for(var i=0;i<data.number;i++){

        document.querySelector("#Foodlists").innerHTML = "<li><b><p>" +
data.results[i].title + "</p></b>";

        document.querySelector("#Foodlists").innerHTML = "<img src=\"\" +
data.results[i].image + "\"></li>";

    }

})

.catch((err) => console.log(err));

});

```

This is for fetching data from the Spoonacular API and then displaying the image of the pizza on the web page. DOMContentLoaded is for loading the process from a html file. This fetch function makes a GET request to the api url, then converts the response to JSON format. The first .then() block processes the returned JSON data, while the second .then() block displays it on the webpage. Any errors during the fetch operation are caught and logged to the console using .catch(). The loop displays title and image.

Login.html

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Login</title>

```



```
<!-- <link rel="stylesheet" href="homepage.css"> -->
```

```
<link rel="stylesheet" href="/login.css">
```

```
<script src="/Nav/nav.js" defer></script>
```

```
<script src = "/login.js"></script>
```

```
<navbar-component></navbar-component>
```

```
</head>
```

```
<body>
```

```
<nav class="bar"></nav>
```

```
<div class="container">
```

```
<div class="bg">
```

```
<div class="content">
```

```
HYPEGEAR
```

```
</div>
```

```
</div>
```

```
<div class="box_login">
```

```
<div class="head_login"><b>Log in</b></div>
```

```
<form>
```

```
<input type="email" placeholder="Email" id="Name">
```

```

        <input type="password" placeholder="Password" id="PW">

        <div>

            <!-- go to the process login -->

            <button type="submit" onclick="loginForm()"
id="login">Login</button>

            <!-- go to Register -->

            <a href="/Register"><p ><u>Sign-up?</u></p></a>

        </div>

    </form>

</div>

</div>

<footer-component></footer-component>

</body>

</html>

```

This page is a login page with email and password inputs, a login button, and a link to sign up page, along with navigation bar and a footer component.

Login.js

```
function loginForm() {

    const username = document.getElementById("Name").value;

    const password = document.getElementById("PW").value;

    // check both the user and the password is correct

    fetch('http://localhost:2021/login', {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ email: username, password: password})

    })

    .then(response => {

        console.log(response)

        if (response.status===201) {

            // if some of the them are wrong return this from the status that our of the
length

            alert("Incorrect email or password. Please try again.");

            throw new Error("Authentication failed.");

        }

        return response.json();

    })

}
```

```

.then(data => {

    window.location.href="/"

})

.catch(error => {

    console.error('Error Login user:', error);

});

}

```

The loginForm() function returns the values entered in the email and password fields, then sends them to a local server through a POST request. If the response status is not 201, indicating a successful login, But if the status is 201 it will alert the user error. If the login is successful, it redirects the user to the homepage. If there's any error during the process, it's logged to the console.

Register.html

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Register</title>

    <link rel="stylesheet" href="/Register.css">

    <script src = "/Register.js"></script>

    <script src="/Nav/nav.js" defer></script>

    <navbar-component></navbar-component>

```

</head>

<body>

<nav class="bar"></nav>

<div class="container">

<div class="bg">

<div class="content">

HYPEGEAR

</div>

</div>

<div class="box_login">

<div class="head_login">Register</div>

<form class="formsubmit">

<input type="text" placeholder="Firstname" pattern="[A-Za-z]+"
id="Fame">

<input type="text" placeholder="Lastname" pattern="[A-Za-z]+"
id="Lame">

<input type="email" placeholder="Email" id="email">

<input type="password" placeholder="Password" id="PW">

<input type="date" placeholder="yyyy/mm/dd" id="bd">

<input type="number" placeholder="Age" id="age">

<input type="text" placeholder="phone number" pattern="[0-9]{10}"
id="Pnumber">

<div>

```

        <button type="submit" onclick="registerForm()">Sign in</button>

        <a href="/login"><p><u>Login?</u></p></a>

    </div>

</form>

</div>

</div>

<footer-component></footer-component>

</body>

</html>

```

This page is for a registration page with input fields for first name, last name, email, password, birthdate, age, and phone number. It has a button to submit the registration form and a link to the login page.

Register.js

```

function registerForm() {

    const fname = document.getElementById("Fame").value;

    const lname = document.getElementById("Lame").value;

    const email = document.getElementById("email").value;

    const password = document.getElementById("PW").value;

    const birthdate = document.getElementById("bd").value;

    const phone = document.getElementById("Pnumber").value;

    const age = document.getElementById("age").value;

    const userdata = {

        "UserEmail": email,

        "FName": fname,

```



```

    "LName": lname,
    "UserPassword": password,
    "Birthdate": birthdate,
    "Phone": phone,
    "Age": age
  };

  fetch('http://localhost:2021/register', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ userData: userdata })
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('Failed to register user. Status: ' + response.status);
    }
    return response.json();
  })
  .then(data => {
    window.location.href="/"
  })
  .catch(error => {
    console.error('Error registering user:', error);
  });

```

```
});  
  
}
```

The registerForm() function retrieves the values entered in the registration form fields (first name, last name, email, password, birthdate, phone number, and age), constructs a user data object, and sends it to a local server through a POST request. If the registration is successful, the user is redirected to the homepage. If there's any error during the process, users have to do it again.

Edit.html

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
  <meta charset="UTF-8">  
  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  
  <title>Edit</title>  
  
  <link rel="stylesheet" href="/Edit.css">  
  
  <script src="/Nav/nav.js"></script>  
  
  <script src="Edit.js"></script>  
  
</head>  
  
<body>  
  
  <navbar-component></navbar-component>  
  
  <nav class="bar"></nav>  
  
  <div class="box">  
  
    <!-- using to type admin,user,Product -->  
  
    <button id="Ad" onclick="Change(id)">Admins_list</button>
```

```

    <button id="User" onclick="Change(id)">User_list</button>

    <button id="Product" onclick="Change(id)">Product_list</button>

</div>

<footer-component></footer-component>

</body>

</html>

```

This page is for editing lists of admins, users, and products. The page contains a navigation bar and footer.

Edit.js

```

function Change(id){
    if(id=="Ad"){
        window.location.href = "/List_Admin";
    }
    else if(id=="User"){
        window.location.href = "/List_User";
    }
    else{
        window.location.href = "/List_product";
    }
}

```

This function Change(id) redirects to different list pages based on the button clicked. If the button is for admins, it redirects to /List_Admin. If it's for users, it redirects to /List_User; and if it's for products, it redirects to /List_Product.

Product.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Product</title>

  <link rel="stylesheet" href="/product.css">

  <script src="/product.js"></script>

  <script src="/Nav/nav.js" defer></script>

  <navbar-component></navbar-component>

</head>

<body>

  <nav class="bar"></nav>

  <br>

  <div id="page" class="container">

    </div>

    <br>

    <footer-component></footer-component>

  </body>

</html>
```

This page is a product page, where product details will be displayed.

Product.js

```
document.addEventListener("DOMContentLoaded", function() {

    // get the data from the url

    const urlParams = new URLSearchParams(window.location.search);
    //URLSearchParams// a built-in JavaScript class that provides methods and
    properties for working with query parameters in a URL.

    // this method for getting the information from the url

    const id = urlParams.get('ID');

    fetch('http://localhost:2021/product', {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ product_id: id }) // Other hand you can use the
        pamas to depend on your database we recommend using body easier to use

    })

    .then(response => {

        if (!response.ok) {

            throw new Error('Network response was not ok');

        }

        return response.json();

    })

    .then(data => {

        const Detail = data.data;
```

```

const Img = Detail.Image;

const Name = Detail.ProductName;

const Des = Detail.ProductDescription;

const qu = Detail.Quantity;

const price = Detail.Price;

const mt = Detail.Meterial;

const color = Detail.Color;

const collection = Detail.Collection;

```

```

document.getElementById('page').innerHTML=

```

```

<div class="size">

```

```

```

```

</div>

```

```

<div class="Detail">

```

```

    <div style="display: flex; align-items:
flex-end;"><h1>${Name}<sub><small>@${id}</small></sub></h1></div>

```

```

    <div><h2>${price} THB</h2></div>

```

```

    <h2 id="event">Collection: ${collection}</h2>

```

```

    <h2>Description:</h2>

```

```

    <h3>${Des}</h3>

```

```

    <div class="same_line">

```

```

        <h2>SIZE:</h2>

```

```

        <select id="option">

```



```

    <option id="S">S</option>

    <option id="M">M</option>

    <option id="L">L</option>

    <option id="XL">XL</option>

    <option id="XXL">XXL</option>

    <option id="superXL">superXL</option>

</select>

```

```

</div>

```

```

<h2><label>Quantity:</label><input type="number" id='number'></h2>

```

```

<div class="center">

```

```

    <button class = "box_product_2" type="button">Buy</button>

```

```

</div>

```

```

</div>

```

```

`;

```

```

document.getElementById('number').value=qu;

```

```

})

```

```

.catch(error => {

```

```

    console.error('Error fetching data:', error);

```

```

});

```

```

});

```

This script fetches product details based on the localhost. It sends a POST request to the server with the product ID, retrieves the product data, and fills in the HTML with the product information. This code fetches product details based on the ID

parameter from the URL, updates the page with the product information, and sets the quantity input field to the available quantity.

hosting.js

```
const express = require('express');
```

```
// need for the url reading
```

```
const path = require('path');
```

```
const port = 2020;
```

```
const Project_web = express();
```

```
const router = express.Router();
```

```
Project_web.use(router);
```

```
router.use(express.json());
```

```
router.use(express.urlencoded({ extended: true }));
```

```
router.use("/Nav",express.static(path.join(__dirname, 'Nav')));
```

```
router.use("/product",express.static(path.join(__dirname, 'porduct')));
```

```
function call(Name,file,url){
```

```
    router.use(express.static(path.join(__dirname, Name)));
```

```

    router.get(url, (req, res) => {

    res.sendFile(path.join(__dirname,Name, file));

    console.log(`working: Server AT ${file}`)

    });

}

// // Home_page

call('Home_page_website','Home_page.html','/');


// // See_more

call('See_more','Im-not-a-god-Im-a-human-Collection.html','/Im-not-a-god-Im-a-human-Collection');

call('See_more','MR.Bloom.html','/MR.Bloom');

call('See_more','Special day.html','/Special-day');


// Footer

call('Member','About_us.html','/About_us');

call('Ourservice','Ourservice.html','/Ourservices');

call('Privacy','Privacy.html','/Privacy');

call('TermandCondition','terms_and_conditions.html','/TermandCondition');

call('Support','support.html','/Support');

// call('Front-end')

```

// login

call('Login','login.html','/login');

call('Register','Register.html','/Register');

// search

call('Adv_search','Advance Search.html','/Advance-Search');

call('Adv_search','Search_showproducts.html','/Search_showproducts');

// Password_Process

call('Edit','Edit.html','/Edit');

// List

call('List','list_user.html','/List_User')

call('List','List_admin.html','/List_Admin')

call('List','Admin_detail.html','/Admin_detail')

call('List','user_list.html','/user_list')

call('List_Product','List_product.html','/List_product')

call('List','Admin_detail_add.html','/Admin_detail_add')

// Product

call('product','product.html','/Detail')

// Guess it

call('Guess','Guess.html','/Guess')

```
// listen import

Project_web.listen(port, () => {

  console.log(`Server listening on port: ${port}`)

})
```

This code sets up routes to serve static files for different pages and resources. It uses the **express.static** to function static files from the specified directories. The call function simplifies the creation of routes for each page. Finally, the server listens on port 2020.

Advance Search.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0"> x

  <title>Advance Search</title>


  <link rel="stylesheet" href="/Advance Search.css">

  <script src='/Advance Search.js'></script>

  <script src="/Nav/nav.js" defer></script>

</head>

<body>

  <navbar-component></navbar-component>
```

```

<nav class="bar"></nav>

<div class="AdvanceSearch">

  <div class="half">

    <form id="form">

      <label for="Advance_Search"><h1><b>Advance
Search</b><br></h1></label>

      <label for = "Name"><b>Name: </b></label>

      <input type = "text" id = "Name"  required><br>

      <label for = "title"><b>color:</b></label>

      <input type = "text" id = "Color"  required><br>

      <label for = "Price"><b>collection:</b></label>

      <input type = "text" id = "collection"  required><br>

      <button class="button-35" type="submit" id="submit"
>SEARCH</button>

    </form>

  </div>

  <div id="show" class="half">

    <!-- Show the Product that want it -->

  </div>

</div>

</div>

<footer-component></footer-component>

</body>

</html>

```

This page is an advanced search page. It includes a form with inputs for Name, Color, and Collection, and a submit button. This file links with the Advance Search.js.

Advance Search.js

```
document.addEventListener("DOMContentLoaded", function() {

    const form = document.getElementById('form');

    // List for the the user that click on the product

    form.addEventListener('submit', function (event) {

        event.preventDefault();

        const name = document.getElementById('Name').value;

        const color = document.getElementById('Color').value;

        const collection = document.getElementById('collection').value;

        // go to the database

        fetch('http://localhost:2021/adsearch', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            // setting information for the req.body.information

            body: JSON.stringify({ Name: name, Color: color, Collection: collection })

        })

        .then(response => {

            if (response.status=== 201) {

                // from the data that there are not have that product see the 201 for
                make it work and using the alert to tell the user that you information are wrong
```



```

        alert("There are no product that you are searching?")

        throw new Error('Network response was not ok');

    }

    return response.json();
})

.then(data => {

    const Detail=data.data;

    const Name = Detail.ProductName;

    const mt = Detail.Meterial;

    const Price = Detail.Price;

    const Img = Detail.Image;

    const color = Detail.Color;

    const di = Detail.ProductID;

    // this process setting the a html to the id='show'

    document.getElementById('show').innerHTML=`

    <div class="Product">

    <img src = ${Img} class = "imgproducts" >

    <div class="info">

        <div id="collection"><b><br>${Name}<br>${mt}</b></div>

        <div id="color">${color}</div>

        <div id="price">${Price} Baht<br></div>

        <button class="button-35" type="submit"
onclick="gotodetail('${di}')">buy</button>

    </div>

```

```

        </div>

        </div>`;

    })

    .catch(error => {

        console.error('Error fetching data:', error);

    });

});

});

// send the id product to add to the url and use pamas or call other way.

function gotodetail(d) {

    console.log(d)

    const data = {

        "ID": d

    }

    const queryString = new URLSearchParams(data).toString();

    const url = "/product.html?" + queryString;;

    window.location.href = url

}

```

The DOMContentLoaded event listener ensures that the JavaScript code executes only after the entire HTML document has loaded. The form event listener captures form submissions from the advanced search form. The fetch function sends a POST request to the server with the search criteria, and the then method handles the server's response. If the response indicates a successful match, the gotodetail function dynamically generates HTML to display the product details. The gotodetail function redirects the user to the product detail page when they click the 'Buy' button, passing the product ID as a parameter. If no match is found, an alert informs the user.

Search_showproducts.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Searched Products</title>

  <link rel="stylesheet" href="/Search_showproducts.css">

  <script src="/Nav/nav.js" defer></script>

  <script src="/Search_showproducts.js"></script>

</head>

<body>

  <navbar-component></navbar-component>

  <nav class="bar"></nav>

  <div class="choose-bar">Choose your choices</div>

  <div id="store" class="text">

    <!-- input data from the js -->

  </div>

  <br>

  <footer-component></footer-component>

</body>

</html>
```

This page displays searched products from users insert. Users can see the details of all products.

Search_showproducts.js

```
// method that readying the data from dom

document.addEventListener("DOMContentLoaded", function() {

    const Data = localStorage.getItem("Search_Data");

    console.log(Data)

    // getting to the database

    fetch('http://localhost:2021/List_Product', {

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        },

    })

    .then(result => result.json())

    .then(data => {

        const infro = data.data;

        var Check=true;

        console.log(infro);

        // process running to database

        for (var i=0;i<infro.length;i++){

            const List=infro[i]

            const Name = List.ProductName

            const Collection = List.Collection

            const Color= List.Color

            const img=List.Image
```

```

const mt=List.Meterial

const price=List.Price

const id = List.ProductID

const d = String(''+id+'')

localStorage.setItem(`${id}`,id);

// showing only the product that search for

if(Data===Name){

    Check=false;

    document.getElementById("store").innerHTML+=`

<div class="textbox" onclick="gotodetail(${d})">

<img src=${img} class="imgproducts">

<div class="detail">

    <h3><b>${Name}<br>${mt}</b></h3>

    <h3>${Color}</h3>

    <h3>${price} Baht</h3>

</div>

</div>`;

}else if(Data===Collection){

    Check=false;

    document.getElementById("store").innerHTML+=`

<div class="textbox" onclick="gotodetail(${d})">

<img src=${img} class="imgproducts">

<div class="detail">

    <h3><b>${Name}<br>${mt}</b></h3>

```

```

        <h3>${Color}</h3>

        <h3>${price} Baht</h3>

    </div>

</div>`;

}else if(Data===Color){

    Check=false;

    document.getElementById("store").innerHTML+=`

    <div class="textbox" onclick="gotodetail(${d})">

    <img src=${img} class="imgproducts">

    <div class="detail">

        <h3><b>${Name}<br>${mt}</b></h3>

        <h3>${Color}</h3>

        <h3>${price} Baht</h3>

    </div>

    </div>`;

}else if(Data===id){

    Check=false;

    document.getElementById("store").innerHTML+=`

    <div class="textbox" onclick="gotodetail(${d})">

    <img src=${img} class="imgproducts">

    <div class="detail">

        <h3><b>${Name}<br>${mt}</b></h3>

        <h3>${Color}</h3>

        <h3>${price} Baht</h3>

```

```

        </div>

        </div>`;

    }

}

// if not the search you find it show all the product
if(Check){

    console.log("in")

    for (var i=0;i<infro.length;i++){

        const List=infro[i]

        const Name = List.ProductName

        const Collection = List.Collection

        const Color= List.Color

        const img=List.Image

        const mt=List.Meterial

        const price=List.Price

        const id = List.ProductID

        const d = String(''+id+'')

        localStorage.setItem(`${id}`,id);

        document.getElementById("store").innerHTML+=`

        <div class="textbox" onclick="gotodetail(${d})">

        <img src=${img} class="imgproducts">

        <div class="detail">

            <h3><b>${Name}<br>${mt}</b></h3>

            <h3>${Color}</h3>

```



```

        <h3>${price} Baht</h3>

    </div>

</div>`;

    };

}

})

.catch(error => {

    console.error('Error fetching data:', error);

});

});

function gotodetail(d) {

    const data = {

        "ID": d

    }

    // setting the ID to the url for get the data from the Paramas /detail/:id like this

    const queryString = new URLSearchParams(data).toString();

    const url = "/product.html?" + queryString;;

    window.location.href = url

}

```

The DOMContentLoaded event listener waits for the page to load, then retrieves search data from local storage. It sends a request to the server to fetch all products. The retrieved data is iterated through, checking if any product matches the search query. If a match is found based on name, collection, color, or ID, it's displayed on the page. If no match is found, all products are displayed. The gotodetail function is called

when a user clicks on a product. It takes the product ID and go to the product detail page.

nav.js

class NavBar extends HTMLElement { //declares a new class named NavBar which extends HTMLElement

constructor() {

super(); // calls the constructor of the parent class (HTMLElement in this case). This is necessary because NavBar is extending HTMLElement

}

// lifecycle callback that gets invoked when the custom element is inserted into the DOM (Document Object Model).

connectedCallback() {

// information to learn html

this.innerHTML = `

<nav class="Menu">

<div class="catagorize">

<div class="Hypelogo">

HYPEGEAR

</div>

<div class="Hypefunction">

** Home**

** Guess the new Collection **

<div id="ya" class="line space_between white"> Sale </div>

** Support **

</div>

<div class="searchbox">

<from>

<input type="text" id="ss" placeholder="Search">

</from>

Advance Search

</div>

</div>

</nav >

<style>

font-face {

font-family: Playfair;

font-style: normal;

font-weight: 400;

font-stretch: 100%;

src:

url(https://fonts.gstatic.com/s/playfair/v2/0nkQC9D7PO4KhmUJ5_zTZ_4MYQX
znAK-TUcZXKO3UMnW6VNpe4-SiiZ4b8h5G3GutPkUeugeqyIA5g.woff2)
format('woff2');

unicode-range: U+0301, U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;

}

.Menu{

```
Top: 0;

width: 1400px ;

position: fixed;

z-index: 1;

background-color: white;
}

.catagorize{

display: flex;

width: 100%;

position: relative;

justify-content: first-start;

align-items: baseline;

flex-direction: row;
}

body{

margin: 0%;

width: 1400px;
}

.bar{

height: 5rem;
}

.Hypelogo{

display: flex;

position: relative;
```

```

    justify-content: flex-start;

    text-align: center;

    padding: 1rem 2rem;
}

.Hypelogo.Hypefunction, .searchbox, .toolbar{

    flex-grow: 1;
}

.Hypefunction{

    font-size: 1rem;

    position: left;

    display: flex;

    flex-flow : flex-start;

    align-items: baseline;

    justify-content:flex-start;

    flex-wrap: wrap;

    flex-direction: row;
}

.searchbox{

    flex: 1;

    display: flex;

    justify-content: flex-end;

    align-items: center;

    padding: 0 2rem;

    gap: 2rem;

```

```
}  
  
.searchbox input{  
    outline:none;  
    background: white;  
    height: 100%;  
    padding:0 10px;  
    font-size:16px;  
    width: 350px;  
}  
  
a{  
    text-decoration: none;  
    padding: 0%;  
}  
  
.logo_hyper_home_page{  
    padding: 1rem;  
    border: 10cm;  
    border-radius: 49%;  
    font-family: 'Playfair Display'serif;  
    font-size: 52;  
    color: white;  
    background-color: black;  
}  
  
.line{  
    display: inline-block;
```

```

}

.space_between{
    padding-left: 5ch;
}

.lastspace{
    padding-left: 1ch;
}

.shape{
    border-radius: 30%;
}

.black{
    color: white;
}

.white{
    color: black;
}

#ss{
    width: 7cm;
    margin-right: 0.2ch;
}

.Logo_size{
    width: 20px ;
    height: 20px;
    float: right;

```



```
}  
  
.btn{  
    padding: 1px;  
    padding-left: 15px;  
    width: 8rem;  
    height: 20px;  
    border-style: none;  
    font-weight: bold;  
    outline: none;  
    margin: auto;  
    background-color:rgb(239, 153, 41);  
}  
  
.btn:hover{  
    background-color:rgba(230, 162, 95, 0.783);  
}  
  
footer{  
    width: 1400px;  
    color-background: black;  
}  
  
.bottom_footer{  
    padding: 14px 16px;  
    background-color: black;  
    display: flex;  
    justify-content: space-evenly;
```

```
}
```

```
#ya{
```

```
    cursor: pointer;
```

```
}
```

```
</style>
```

```
`;
```

// import than you can't put the html in from the script in the script using the document create the get script next step using the append child to add the script element after innerhtml

```
const script = document.createElement('script');
```

// other import the data the get the element need to get all of it otherwise the append child will not work

```
script.textContent = `
```

```
const inform = document.getElementById("ss");
```

```
console.log("Search is working");
```

```
inform.addEventListener("keyup", function(event) {
```

```
    if (event.key === "Enter") {
```

```
        gotosearch(inform.value);
```

```
    }
```

```
});
```

```
function gotosearch(data) {
```

```
    localStorage.setItem("Search_Data",data);
```

```
    window.location.href = "/Search_showproducts";
```

```
}
```

```

const click_it = document.getElementById("ya");

click_it.addEventListener("click", function(event){

    alert("There are no any Products sale Yet See you in a Winter!!!!♥");

});

`;

this.appendChild(script);

}

}

// built-in JavaScript object that provides methods for registering and working with
custom elements.

customElements.define('navbar-component', NavBar);

class Footer extends HTMLElement {

    constructor() {

        super();

    }

    connectedCallback() {

        this.innerHTML = `

        <footer>

        <div class="bottom_footer black">

            <a href="/About_us" class="black">About us</a>

            <a href="/Ourservices" class="black">Our Services</a>

            <a href="/Privacy" class="black">Privacy Policy</a>

            <a href="/TermandCondition" class="black">Term & conditions</a>

        </div>

```

```

        </footer>`;
    }
}

customElements.define('footer-component', Footer);

```

The NavBar class defines a custom HTML element called navbar-component, which creates a navigation bar for a webpage. It extends the HTMLElement class and includes a connectedCallback method that populates the navigation bar with links, a search input, and other elements. It also dynamically adds JavaScript functionality to handle search input and click events. The Footer class defines a custom HTML element called footer-component, which creates a footer for a webpage. It extends the HTMLElement class and includes a connectedCallback method that populates the footer with links. These custom elements can be used in HTML documents to easily add navigation bars and footers to web pages.

List_user.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>List_User</title>
  <script src="/Nav/nav.js"></script>
  <link rel="stylesheet" href="/List_css.css">
  <script src="list_user.js"></script>
</head>
<body>
  <navbar-component></navbar-component>
  <nav class="bar"></nav>
  <h1>List_User</h1>
  <div class="div_center">
    <div class="container">
      <div class="header">
        <div class="Num">List</div>
        <div class="inform">Information of User</div>

```

```

        <div class="Edit">Edit</div>
    </div>
    <div id="list">
        <!-- detial information -->
    </div>
</div>
</div>
</div>

<footer-component></footer-component>
<script>
    document.addEventListener("DOMContentLoaded", function() {
        // same thing at Ad
        using();
    });

</script>
</body>
</html>

```

This html file connects with the external CSS file, which is /List_css.css". This file shows the list of users in order. And the script is document.addEventListener("DOMContentLoaded" function() is for letting the function work all the time when opening this page.

List_user.js

```

function using(){
    fetch('http://localhost:2021/List_User', {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        },
    })
    .then(result => result.json())
    .then(data => {
        console.log(data);
    })
}

```

```

const infro = data.data;
console.log(infro);
for (var i=0;i<infro.length;i++){
  const List=infro[i]
  const firstname=List.FName
  const lastname=List.LName
  const email=List.UserEmail
  const Call = List.Phone
  const Age= List.Age
  // amzing thing that it have the problem that the function in here can get the
information directly need to be in string * important '"'," is not same
  const ID = String(''+email+'')
  document.getElementById("list").innerHTML+=`
<div class='box'>
<div id="number">${i+1}</div>
<div id="User_Data">
  <p>${firstname} ${lastname} (Contact: <span
class="color_it">${Call}</span>) Age:${Age} email: <a href = "mailto:
${email}" >${email}</a></p>
</div>
<button id="Detail" onclick="updateuser(${ID})">Detail</button>
</div>
`;
}
})
.catch(error => {
  console.error('Error fetching data:', error);
});
}
// go to the detail of the user
function updateuser(infro){
  console.log(infro)
  // sent data to cloud for the user_list take it
  localStorage.setItem("User_id",infro);
  window.location.href="/user_list";
}

```

This file sends the request to the database to get information about all users and show it in the table, which can expand or shrink depending on the number of rows in

the database. And have the Edit button go to the user_list page to edit or delete the information.

User_list.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Detail User</title>
  <script src="/Nav/nav.js"></script>
  <link rel="stylesheet" href="/user_list.css">
  <script src="/user_list.js"></script>
</head>
<body>
  <navbar-component></navbar-component>
  <nav class="bar"></nav>

  <div class="container">
    <div class="box_login">
      <div class="head_login"><b>User Information</b></div>

      <div id="userDetails"></div>
      <form class="add-user-form">
        <label for="newFirstName">First Name:<input type="text"
placeholder="First Name" id="newFirstName"></label>
        <label for="newLastName">Last Name:<input type="text"
placeholder="Last Name" id="newLastName"></label>
        <label for="newEmail">Email:<input type="email"
placeholder="Email" id="newEmail" disabled></label>
        <label for="age">Age:<input type="number" placeholder="Age"
id="age"></label>
        <label for="PhoneNumber">Phone:<input type="number"
placeholder="Phone Number" id="PhoneNumber">
        <div class="center">
          <button type="submit" id="updateBtn"
onclick="updateuser()">Update</button>

```



```

        <button type="submit" id="deleteBtn"
onclick="deleteuser()">Delete</button>
    </div>
</form>
</div>
</div>
<div class="br"></div>

<footer-component></footer-component>
</body>
</html>

```

This file connects with the external CSS file `user_list.css`. And there are text boxes for admin to update their information namely First name, Last name, Email, Age, Phone number and then click to update information or delete the account by Update and Delete button.

User_List.js

```

document.addEventListener("DOMContentLoaded", function() {
    const email = localStorage.getItem("User_id")
    // get information from the cloud
    console.log({ id: email})

    fetch('http://localhost:2021/User', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ id: email})
    })
    .then(result => result.json())
    .then(data => {
        console.log(data);
        const infro = data.data;

        document.getElementById("newFirstName").value=infro.FName
        document.getElementById("newLastName").value=infro.LName
    })

```

```

        document.getElementById("newEmail").value=info.UserEmail
        document.getElementById("age").value=info.Age
        document.getElementById("PhoneNumber").value=info.Phone
    })
    .catch(error => {
        console.error('Error fetching data:', error);
    });
});

function updateUser() {
    // Get user information from input fields
    const email = localStorage.getItem("User_id")
    const firstName = document.getElementById("newFirstName").value;
    const lastName = document.getElementById("newLastName").value;

    const age = document.getElementById("age").value;
    const phone = document.getElementById("PhoneNumber").value;

    const userData = {
        "FName": firstName,
        "LName": lastName,
        "Age": age,
        "Phone": phone
    };

    fetch("http://localhost:2021/updateUser", {
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            email: email,
            userData: userData
        })
    })
    .then(response => {
        if (response.ok) {
            return response.json();
        }
    })
}

```

```

        throw new Error("Failed to update user information");
    })
    .then(data => {
        // something see the to see the list page for see the result
        window.location.href='/List_User';
    })
    .catch(error => {
        console.error("Error updating user information:", error);
    });
}

```

```

// delete user
function deleteuser() {
    // Get user information from input fields
    const email = localStorage.getItem("User_id")

    console.log({ id: email})

    fetch('http://localhost:2021/deleteUser', {
        method: 'DELETE',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ id: email})
    })
    .then(result => result.json())
    .then(data => {
        console.log(data)
        window.location.href='/List_User';
    })
    .catch(error => {
        console.error('Error fetching data:', error);
    });
}

```

This file is a stored document.addEventListner method to call the information for each product on the user_list page. After clicking the Update button, it will go to the updateuser() function to send all data that users edit to the database by using <http://localhost:2021/updateUser>.

List_admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>List Admin</title>
  <script src="/Nav/nav.js"></script>
  <link rel="stylesheet" href="/List_css.css">
  <script src="List_admin.js"></script>
</head>

<body>
  <navbar-component></navbar-component>
  <nav class="bar"></nav>
  <h1>List_Admin</h1>

  <div class="div_center">
    <button id="Add" >Add</button>
    <div class="container">
      <div class="header">
        <div class="Num">List</div>
        <div class="inform">Information of Admin</div>
        <div class="Edit">Edit</div>
      </div>
      <div id="list">
        <!-- Detail from the js come here -->
      </div>
    </div>
  </div>

  <footer-component></footer-component>
<script>
  document.addEventListener("DOMContentLoaded", function() {
    // loading this using function other hand you can use the DOM Content to
    replace and don't need the using function
    using();
  });
```

```

document.getElementById("Add").addEventListener("click", function() {
    // wait for to click on the button detail
    window.location.href = "/Admin_detail_add.html";
});

```

```

</script>
</body>
</html>

```

This file connects with the external CSS, which is /List_css.css, and the javascript file, List_admin.js. This file also shows the information of the admin in the form of a table. And have a script document.

addEventListener("DOMContentLoaded", function() to ensure that this page works all the time when opening the page and document.

getElementById("Add").addEventListener("click", function()) to go to Admin_detail_add to add the new admin information when clicking the button.

List_admin.js

```

function using(){
    // auto running all the list
    fetch('http://localhost:2021/List_Admin', {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        },
    })
    .then(result => result.json())
    .then(data => {
        console.log(data);
        const infro = data.data;
        console.log(infro);
        for (var i=0;i<infro.length;i++){
            const List=infro[i]
            const email=List.AdEmail
            const ID = List.AdminID
            const roles= List.Roles

```

```

const userName= List.UserName
const data = String(''+ID+'');
document.getElementById("list").innerHTML+=`
<div class='box'>
<div id="number">${i+1}</div>
<div id="Admin_data">
  <p>${userName}(${ID}) Role:${roles} email: <a href = "mailto: ${email}"
>${email}</a></p>
</div>
<button id="Detail" onclick="updateadmin(${data})">Detail</button>
</div>
`;
}
})
.catch(error => {
  console.error('Error fetching data:', error);
});
}
// go to the Admin detail for see the information and update or kick admin
function updateadmin(data){
  console.log(data)
  localStorage.setItem("admin_id",data);
  window.location.href="/Admin_detail";
}

```

This file sends the request to the database by method get. Then use a for loop to run each admin's information in the table, and each admin has a detail button to go to the admin_detail page to edit or delete.

Admin_detail.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Detail Admin</title>
  <script src="/Nav/nav.js"></script>

```

```

<link rel="stylesheet" href="/Admin_detail.css">
<script src="/Admin_detail.js"></script>

</head>
<body>
  <navbar-component></navbar-component>
  <nav class="bar"></nav>
  <div class="box">
    <div class="Head"><b>Admin Details</b></div>
    <!-- detail of the Admin -->
    <form class="center">
      <label for="adID">AdminID: <input type="text" id="adID"
disabled></label>
      <label for="username">Username: <input type="text"
id="username"></label>
      <label for="roles">Roles: <input type="text" id="roles"></label>
      <label for="email">AdEmail: <input type="text" id="email"></label>
      <label for="password">AdminPassword: <input type="text"
id="password"></label>
      <div class="middle">
        <button type="submit" onclick="updateadmin()">Update</button>
        <button type="submit" onclick="deleteadmin()">Delete</button>
      </div>
    </form>
  </div>
  <footer-component></footer-component>

</body>
</html>

```

This file connects with /Admin_detail.css, which is an external CSS file, and the javascript file is /Admin_detail.js. The file contains the text boxes to let the admin edit the information, namely, admin ID, username, roles, email, and admin password. Then click the Update or Delete button to update data in the database or delete this account from the database.

Admin_detail.js

```
// function auto running add see the content
document.addEventListener("DOMContentLoaded", function() {
  // get the inform from cloud
  const admin_id = localStorage.getItem("admin_id")
  // console.log({ id: email})

  fetch('http://localhost:2021/admin', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ id: admin_id})
  })
  .then(result => result.json())
  .then(data => {
    console.log(data);
    const infro = data.data;
    // set the value to show detail
    document.getElementById("adID").value=infro.AdminID
    document.getElementById("username").value=infro.UserName
    document.getElementById("roles").value=infro.Roles
    document.getElementById("email").value=infro.AdEmail
    document.getElementById("password").value=infro.AdminPassword
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
});

function updateadmin() {
  // Get user information from input fields
  const admin_id = localStorage.getItem("admin_id")
  const username = document.getElementById("username").value;
  const roles = document.getElementById("roles").value;
  const emails = document.getElementById("email").value;
  const password = document.getElementById("password").value;

  const adminData = {
```



```

    "Username":username,
    "Roles": roles,
    "AdEmail": emails,
    "AdminPassword": password
  };

  console.log(adminData)
  fetch("http://localhost:2021/updateAdmin", {
    method: "PUT",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      id: admin_id ,
      userData: adminData
    })
  })
  .then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error("Failed to update admin information");
  })
  .then(data => {
    // when the process done set to the list page to see the detail that add
    window.location.href="/List_Admin";
  })
  .catch(error => {
    console.error("Error updating admin information:", error);
  });
};

function deleteadmin() {
  // Get user information from input fields
  const admin_id = localStorage.getItem("admin_id")
  fetch('http://localhost:2021/deleteAdmin', {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json'
    },

```

```

        body: JSON.stringify({ id: admin_id })
    })
    .then(result => result.json())
    .then(data => {
        // delete Admin kick he or she out add go to list that their kicked or not
        window.location.href='/List_Admin';
    })
    .catch(error => {
        console.error('Error fetching data:', error);
    });
}

```

There is a `document.addEventListener("DOMContentLoaded", function()` to get the data from <http://localhost:2021/admin> to ensure that when you click on this page, there will be data shown in each box already. Function `updateadmin()` method to update the data and send it to the database by <http://localhost:2021/updateAdmin> when admin clicks update. The `deleteadmin()` method is used when the admin wants to delete this account by clicking the Delete button, and it sends the ID of the admin to the database to delete it.

Admin_detail_add.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add Admin</title>
    <script src="/Nav/nav.js"></script>
    <link rel="stylesheet" href="/Admin_detail.css">
    <script src="/Admin_detail_add.js"></script>
</head>
<body>
    <navbar-component></navbar-component>
    <nav class="bar"></nav>
    <div class="box">
        <div class="Head"><b>Admin Details</b></div>

```

```

    <form class="center">
        <label for="adID">AdminID: <input type="text" id="adID"
required></label>
        <label for="username">Username: <input type="text" id="username"
required></label>
        <label for="roles">Roles: <input type="text" id="roles"
required></label>
        <label for="email">AdEmail: <input type="text" id="email"
required></label>
        <label for="password">AdminPassword: <input type="text"
id="password" required></label>
        <div class="middle">
            <button type="submit" onclick="insertadmin()"
id="add">Add</button>
        </div>
    </form>
</div>
<br>
<div class="br"></div>
<footer-component></footer-component>
<script>
    document.getElementById("add").addEventListener("click", function() {
        // wait for the add be click add go to the list again to show the detail the
admins is added
        window.location.href = "/List_Admin.html";
    });
</script>

</body>
</html>

```

This file connects with the external CSS, which is /Admin_detail.css, and the script /Admin_detail_add.js. The file aims to add the new admin by providing the text boxes for the new admin to fulfill information namely, Admin ID, Username, Roles, Email, Password. Then click the Submit button to submit the data to the system. And have `document.getElementById("add").addEventListener("click", function()` if click this button, it will go to List_Admin page that means add data already.

Admin_detail_add.js

```
function insertadmin() {
  // Get user information from input fields
  const admin_id = document.getElementById("adID").value;
  const username = document.getElementById("username").value;
  const roles = document.getElementById("roles").value;
  const emails = document.getElementById("email").value;
  const password = document.getElementById("password").value;
  // set the data easy in put *import that Name "AdminID" must be same database or if
  // it not work you can run out in console to see what are there format
  const adminData = {
    "AdminID": admin_id,
    "UserName" : username,
    "Roles": roles,
    "AdEmail": emails,
    "AdminPassword": password
  };
  // for checking the format of the data
  // console.log(adminData)
  fetch("http://localhost:2021/insertAdmin", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      admin: adminData // Pass user data for update to the body
    })
  })
  .then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error("Failed to insert admin information");
  })
  .then(data => {
    console.log(data)
    window.location.href="/List_Admin";
  })
}
```

```
.catch(error => {  
  console.error("Error insert admin information:", error);  
});  
};
```

This file has a function `insertadmin()` that sends the information of a new admin to the database at <http://localhost:2021/insertAdmin>. After adding an admin, go to `/List_Admin` to go back and see the list of admins.

Details of web services and code

```
// get List_Admins
router.get('/List_Admin', (req, res) => {
  const sql = 'select * from Admins';
  connection.query( sql, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results});
  });
});
```

This function is to get the data list of all admins from the database. This function handles GET requests to '/List_Admin'. It selects all admins from the Admins table in the database. When it succeeds, it sends a response with the retrieved admin data.

```
// get List_User
router.get('/List_User', (req, res) => {
  const sql = 'select * from Users;';
  connection.query( sql, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results});
  });
});
```

This function is to get the data list of all users from the database. This function handles GET requests to '/List_User'. It selects all users from the Users table in the database. When it succeeds, it sends a response with the retrieved user data.

```
// get all Product
router.get('/List_Product', (req, res) => {
  const sql = 'select * from Product;';
  connection.query( sql, function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results});
  });
});
```

This function is to get the data list of all products from the database. This function handles GET requests to '/List_Product'. It selects all products from the

Product table in the database. When it succeeds, it sends a response with the retrieved product data.

```
//register (passsss)
router.post('/register', (req, res) => {

  const data = req.body.userData;

  if (!data) {
    return res.status(400).json({ error: true, message: 'Please provide user information' });
  }

  const sql = 'INSERT INTO Users SET ?';

  connection.query(sql, data, (error, results) => {
    if (error) {
      console.error('Error signing up user:', error);
      return res.status(500).json({ error: 'An error occurred while signing up' });
    } else {
      console.log('User signed up successfully');
      return res.status(201).json({ message: 'User signed up successfully' });
    }
  });
});
```

This function is used to tell the users that the registration process is pass or not pass. This function handles POST requests to '/register'. It pulls user data from the request body and checks if it exists. If the data is valid, it then sends a response indicating successfully signing up but if the data is invalid, it then sends a response indicating failure. But if there is no data, then “please provide user information”.

```
// login (passsss)
router.post('/login', (req, res) => {
  const email = req.body.email;
  const password = req.body.password;
  if (!email) {
    return res.status(400).send({ error: true, message: 'Please provide email' });
  }
  else if (!password) {
    return res.status(400).send({ error: true, message: 'Please provide the password' });
  }

  connection.query('SELECT * FROM users WHERE userEmail = ? AND UserPassword = ?', [email, password], function (error, results) {
    if (error) {
      console.error('Error fetching user information:', error);
      return res.status(500).json({ error: true, message: 'Internal server error', details: error.message });
    }

    if (results.length === 0) {
      return res.status(201).json({ error: true, message: 'User not found' });
    }

    // Return the first result from the query
    const userData = results[0];
    return res.status(200).json({ error: false, data: userData, message: 'Login successful' });
  });
});
```

This function is used to tell the users that the login process is pass or not pass. This function handles POST requests to '/login'. It confirms the existence of the email and password by pulling them from the request body. If the user is found, it sends the

user data as a response indicating successful login. But if users did not insert an email it will alert “Please provide email”. If users did not insert the password it will alert “Please provide the password”. If user did not type anything it will respond to the user not found.

```
// select :detail product click image then pull information to front end (passsss)
router.post('/product', (req, res) => {
  const productId = req.body.product_id;

  connection.query('SELECT * FROM Product WHERE ProductID = ?', productId, function (error, results) {
    if (error) {
      console.error('Error fetching product:', error.message);
      return res.status(500).json({ error: true, message: 'An error occurred', details: error.message });
    }

    if (results.length === 0) {
      return res.status(404).json({ message: 'Product not found' });
    }

    const product = results[0];
    return res.status(200).json({ error: false, data: product, message: 'Product retrieved successfully' });
  });
});
```

This function is used to select a product from the Product table if id matches with the ProductID. This function handles POST requests to '/product'. It selects the product based on the ID. If the product is found or the ID is matched, it sends the response “Product retrieved successfully but if not it will response “Product not found”.

```
// update product
router.put('/updateProduct', (req, res) => {
  const product_id = req.body.id;
  const product = req.body.productData;

  if (!product_id || !product) {
    return res.status(400).send({ error: admin, message: 'Please provide Product information' });
  }
  connection.query("UPDATE Product set ? WHERE ProductID = ?", [product, product_id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Product has been updated successfully.' });
  });
});
```

This function is used to update products. This function handles PUT requests to '/updateProduct'. It pulls the product ID and updated product data from the request body, then validates their presence. If both are present, it will update the product information and it sends a response “Product has been updated successfully”.


```
// delete product (from stock and product table)
router.delete('/deleteProduct', (req,res) =>{
  const product_id = req.body.id;

  if (!product_id){
    return res.status(400).send({ error: true, message: 'Please provide product_id' });
  }
  connection.query('DELETE FROM Stock WHERE ProductID = ?', [product_id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Product has been deleted from "Stock" successfully.' });
  });
  connection.query('DELETE FROM Product WHERE ProductID = ?', [product_id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Product has been deleted successfully.' });
  });
});
```

This function is used to delete products from stock and product tables. This function handles DELETE requests to '/deleteProduct'. It pulls the product ID from the request body and validates its presence. If the ID is present, it will remove the product and it will send a response “Product has been deleted successfully” or “Product has been deleted from “Stock” successfully” when it matched the ID in stock. But if not it will response “Please provide product_id”

```
// advance search (pass)
router.post('/adsearch', (req, res) => {
  const name = req.body.Name;
  const color = req.body.Color;
  const collection = req.body.Collection;

  const sql = `
    SELECT * FROM Product
    WHERE ProductName LIKE ? AND Color LIKE ? AND Collection LIKE ?
  `;

  connection.query(sql, [name,color,collection], (error, results) => {
    if (error) {
      console.error('Error executing query:', error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }

    if (results.length === 0) {
      return res.status(201).json({ error: true, message: 'No products found' });
    }
    const product = results[0];
    return res.send({ error: false, data: product, message: 'Products search successful' });
  });
});
```

This is a function for advanced search. This function handles POST requests to '/adsearch'. It pulls the search parameters (name, color, collection) from the request body. Then, it constructs an SQL query to perform an advanced search based on these parameters. If the product matched every parameters it will show that product as a result but if there is no information in the search parameters it will respond “No products found”.

```
// insert admin
router.post('/insertAdmin', (req, res) => {
  let admin = req.body.admin;

  if (!admin) {
    return res.status(400).send({ error: true, message: 'Please provide Admin information' });
  }
  connection.query("INSERT INTO Admins SET ? ", admin, function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results.affectedRows, message: 'New admin has been created successfully.'});
  });
});
```

This function is for inserting admin. This function handles POST requests to '/insertAdmin'. It pulls admin data from the request body and checks if it exists. If the data is valid, it will insert the admin data into the Admins table, it will send a response “New admin has been created successfully” But if admin data from the request body does not exist it will respond “Please provide Admin information”.

```
// select admin
router.post('/admin', (req, res) => {
  const adminId = req.body.id;

  connection.query('SELECT * FROM Admins WHERE AdminID = ?', adminId, function (error, results) {
    if (error) {
      console.error('Error fetching product:', error.message);
      return res.status(500).json({ error: true, message: 'An error occurred', details: error.message });
    }

    if (results.length === 0) {
      return res.status(404).json({ message: 'Admin not found' });
    }

    const product = results[0];
    return res.status(200).json({ error: false, data: product, message: 'Admin retrieved successfully' });
  });
});
```

This function is for selecting admin. This function handles POST requests to '/admin'. It pulls the admin ID from the request body and selects the admin based on the ID. If the admin is found, it will respond “Admin retrieved successfully” But if admin is not found it will respond “Admin not found”.

```
// update admin
router.put('/updateAdmin', (req, res) => {
  const admin_id = req.body.id;
  const admin = req.body.userData;

  if (!admin_id || !admin) {
    return res.status(400).send({ error: admin, message: 'Please provide Admin information' });
  }
  connection.query("UPDATE Admins set ? WHERE AdminID = ?", [admin, admin_id], function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results.affectedRows, message: 'Admin has been updated successfully.'});
  });
});
```

This function is for updating admin. This function handles PUT requests to '/updateAdmin'. It pulls the admin ID and updated admin data from the request body and validates their presence. If both are present, it will update the admin information and send a response “Admin has been updated successfully” But if both are not present, it will respond “Please provide admin information”.

```
// delete admin
router.delete('/deleteAdmin', (req,res) =>{
  const admin_id = req.body.id;

  if (!admin_id){
    return res.status(400).send({ error: true, message: 'Please provide admin_id' });
  }
  connection.query('DELETE FROM Admins WHERE AdminID = ?', [admin_id], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Admin has been deleted from successfully.'});
  });
});
```

This function is for delete admin. This function handles DELETE requests to '/deleteAdmin'. It pulls the admin ID from the request body and validates its presence. If the ID is present, it will remove the admin and it will send a response “Admin has been deleted successfully” but if the ID is not present it will respond “Please provide admin_id”.

```
// insert user
router.post('/insertUser', (req, res) => {
  let user = req.body.userData;

  if (!user) {
    return res.status(400).send({ error: true, message: 'Please provide User information' });
  }
  connection.query("INSERT INTO Users SET ? ", user, function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results.affectedRows, message: 'New user has been created successfully.'});
  });
});
```

This function is for inserting users. This function handles POST requests to '/insertUser'. It pulls user data from the request body and checks if it exists. If the data is valid, it will insert the user data into the Users table and it will send a response “New user has been created successfully” but if the data is invalid it will respond “Please provide User information”.

```
// select user
router.post('/User', (req, res) => {
  const userMail = req.body.id;
  connection.query('SELECT * FROM Users WHERE UserEmail = ?', userMail, function (error, results) {
    if (error) {
      console.error('Error fetching product:', error.message);
      return res.status(500).json({ error: true, message: 'An error occurred', details: error.message });
    }

    if (results.length === 0) {
      return res.status(404).json({ message: 'User not found' });
    }

    const product = results[0];
    return res.status(200).json({ error: false, data: product, message: 'User retrieved successfully' });
  });
});
```

This function is for select users. This function handles POST requests to '/User'. It pulls the user email from the request body and selects the user based on the email. If the user is found, it will respond “User retrieved successfully” but if the user is not found it will respond “User not found”.

```
// update user
router.put('/updateUser', (req,res) => {
  const user_email = req.body.email;
  const user = req.body.userData;

  if (!user_email || !user) {
    return res.status(400).send({ error: user, message: 'Please provide User information' });
  }
  connection.query("UPDATE Users set ? WHERE UserEmail = ?",[user,user_email], function (error, results) {
    if (error) throw error;
    return res.send({error: false, data: results.affectedRows, message: 'User has been updated successfully.'})
  });
});
```

This function is for updating users. This function handles PUT requests to '/updateUser'. It pulls the user email and updated user data from the request body and validates their presence. If both are present, it will update the user information and send a response “User has been updated successfully” but if not it will respond “Please provide User information”.

```
// delete user
router.delete('/deleteUser', (req,res) =>{
  const user_email = req.body.id;

  if (!user_email){
    return res.status(400).send({ error: true, message: 'Please provide user_email' });
  }
  connection.query('DELETE FROM Users WHERE userEmail = ?', [user_email], function (error, results) {
    if (error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'User has been deleted from successfully.'});
  });
});
```

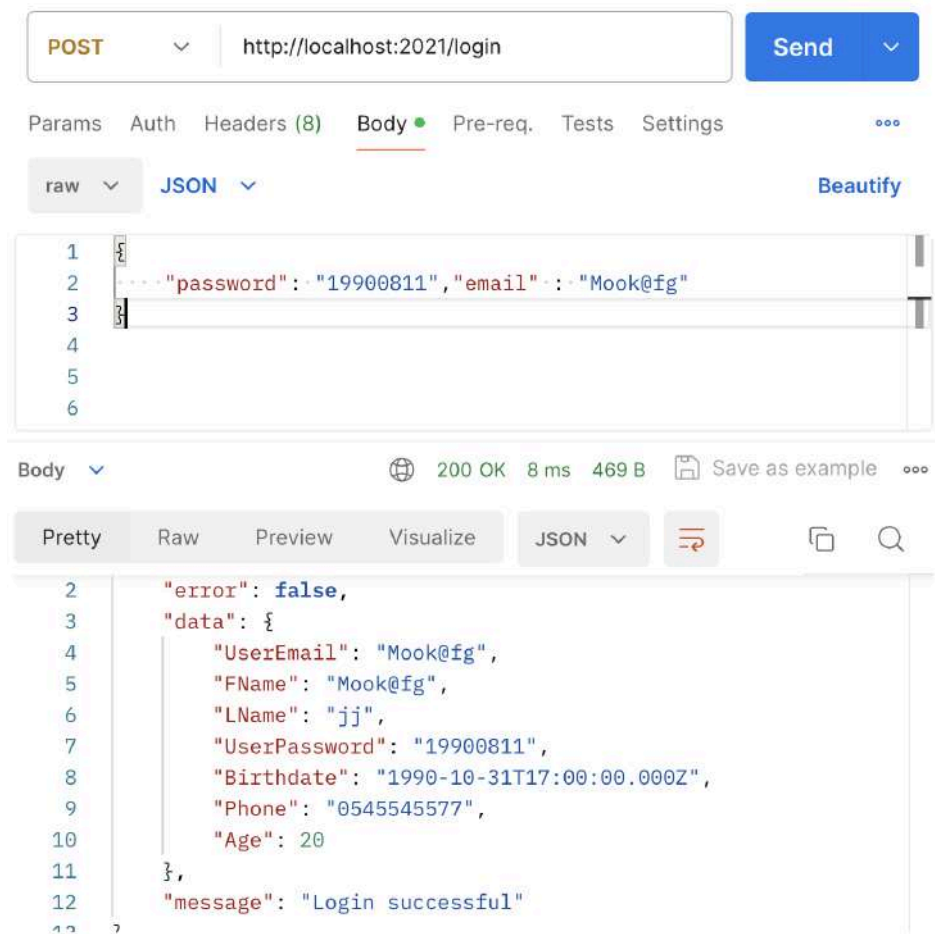
This function is for delete users. This function handles DELETE requests to '/deleteUser'. It pulls the user email from the request body and validates its presence. If the email is present, it will delete the user and send a response “User has been deleted successfully” but if the email is not present it will respond “Please provide user_email”.

```
app.listen(process.env.PORT, function() {
  console.log("Server listening at Port "
+ process.env.PORT);
});
```

This process is very important because if it dont have it the server will not work. This function starts the server and listens for incoming requests on the port defined in the environment variable 'PORT'. When the server starts, it logs a message indicating that the server is listening on the specified port.

The testing results of web services using Postman

- Log-In



- Register

POST http://localhost:2021/register

Params Auth Headers (8) **Body** Pre-req. Tests Settings

raw JSON Beautify

```

3      "UserEmail": "Mook@fg",
4      "FName": "Mook@fg",
5      "LName": "jj",
6      "UserPassword": "19900811",
7      "Birthdate": "1990-11-01",
8      "Phone": "0545545577",

```

Body 201 Created 9 ms 313 B Save as example

Pretty Raw Preview Visualize JSON

```

1  {
2    "message": "User signed up successfully"
3  }

```

• Select All Admins

GET http://localhost:2021/List_Admin

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (8) Test Results Status: 200 OK Time: 13 ms Size: 1.05 KB Save as example

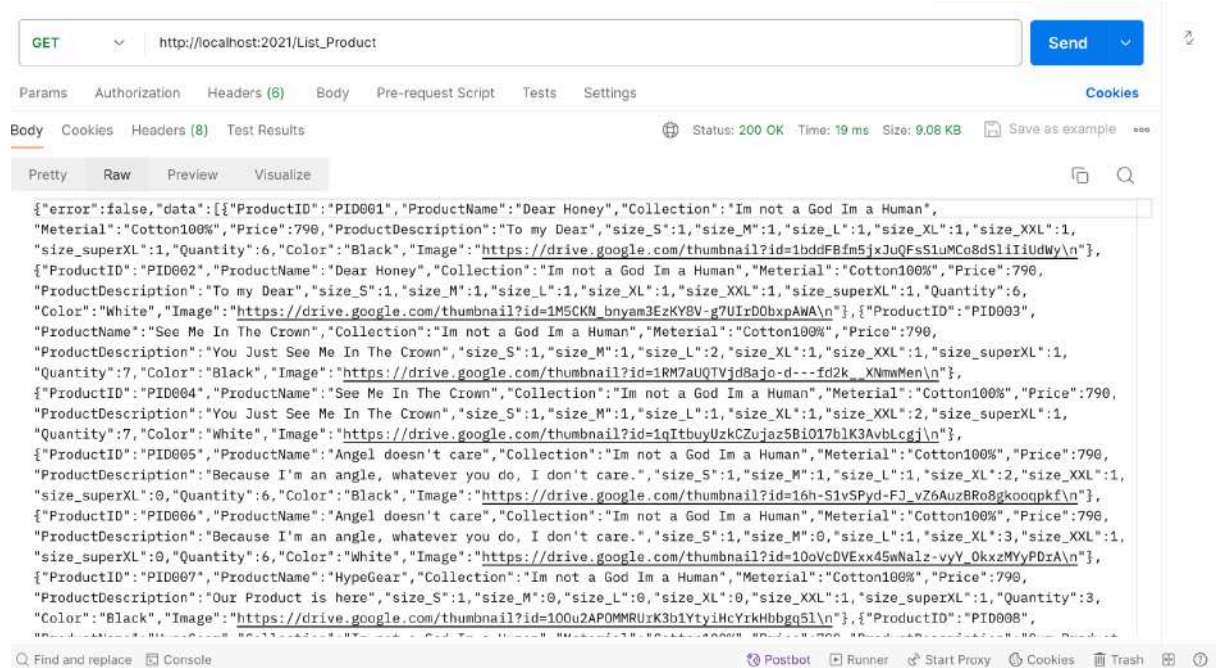
Pretty Raw Preview Visualize

```

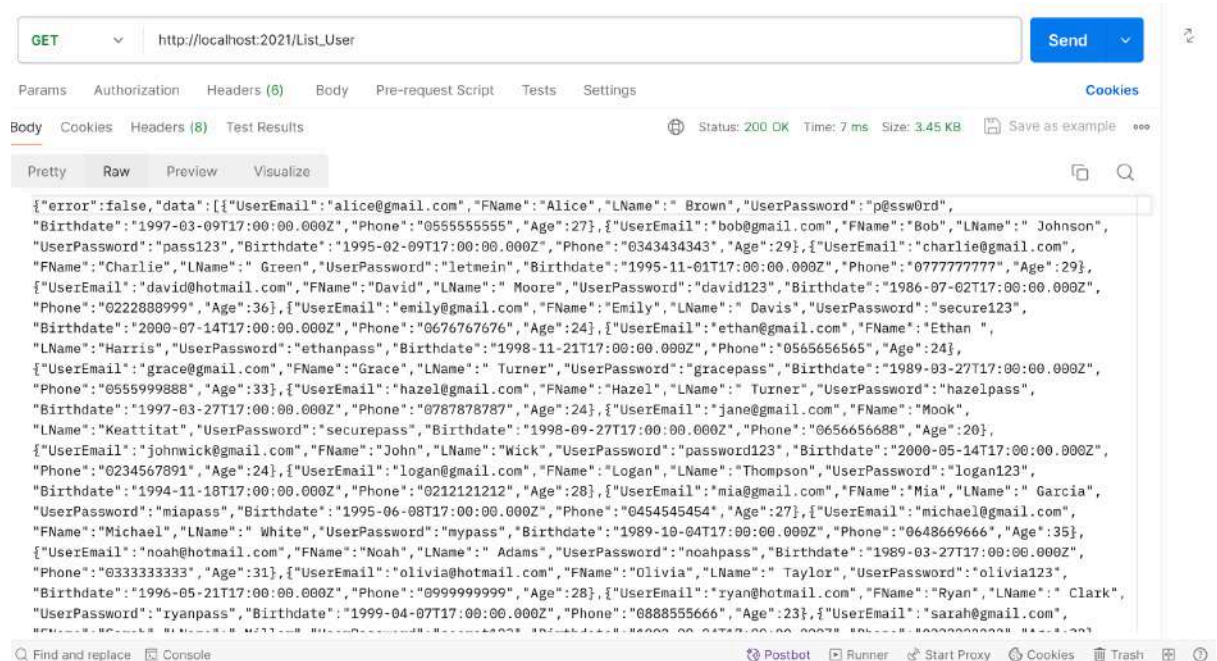
{"error":false,"data":[{"AdminID":"AD002","UserName":"admin_emily","Roles":"ContentManager","AdEmail":"emily.smith@gmail.com",
"AdminPassword":"EmilyInParis00"},{"AdminID":"AD003","UserName":"admin_mike","Roles":"Moderator","AdEmail":"mike.wilson@gmail.com",
"AdminPassword":"MikeIsSupercool!"}, {"AdminID":"AD004","UserName":"admin_susan","Roles":"SiteAdmin","AdEmail":"susantofam@gmail.com",
"AdminPassword":"Sus@ntoFC"}, {"AdminID":"AD005","UserName":"admin_alex","Roles":"ContentManager","AdEmail":"alexsonprx@gmail.com",
"AdminPassword":"AlexDesk!"}, {"AdminID":"AD006","UserName":"admin_lisa","Roles":"Moderator","AdEmail":"lisa111@gmail.com",
"AdminPassword":"LalIsasipretty1"}, {"AdminID":"AD007","UserName":"admin_david","Roles":"SiteAdmin","AdEmail":"davi.lish@gmail.com",
"AdminPassword":"d4v4dprx"}]}

```

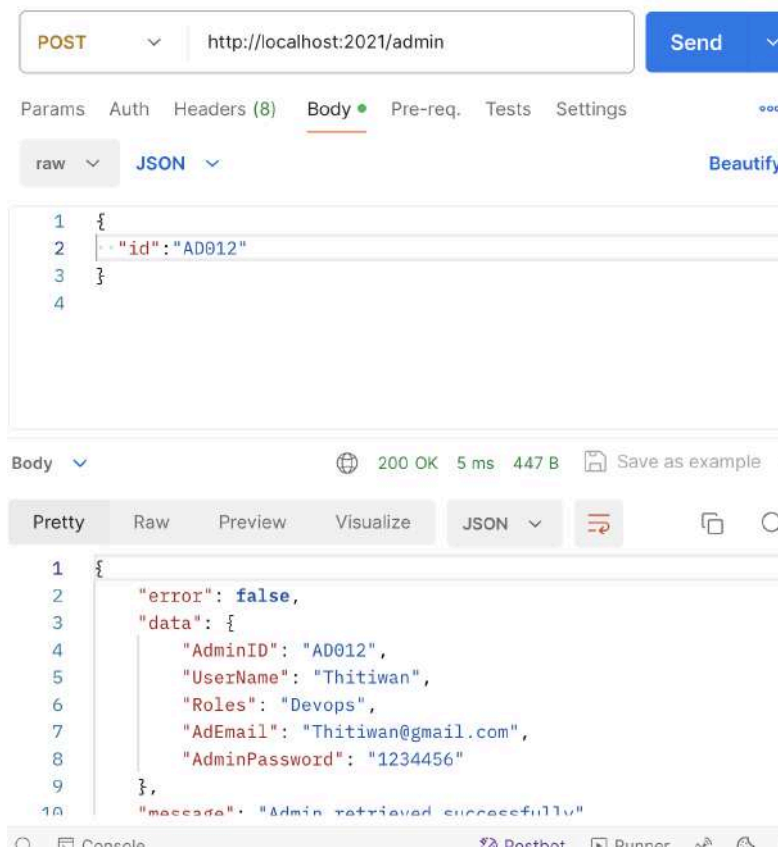
• Select All Products



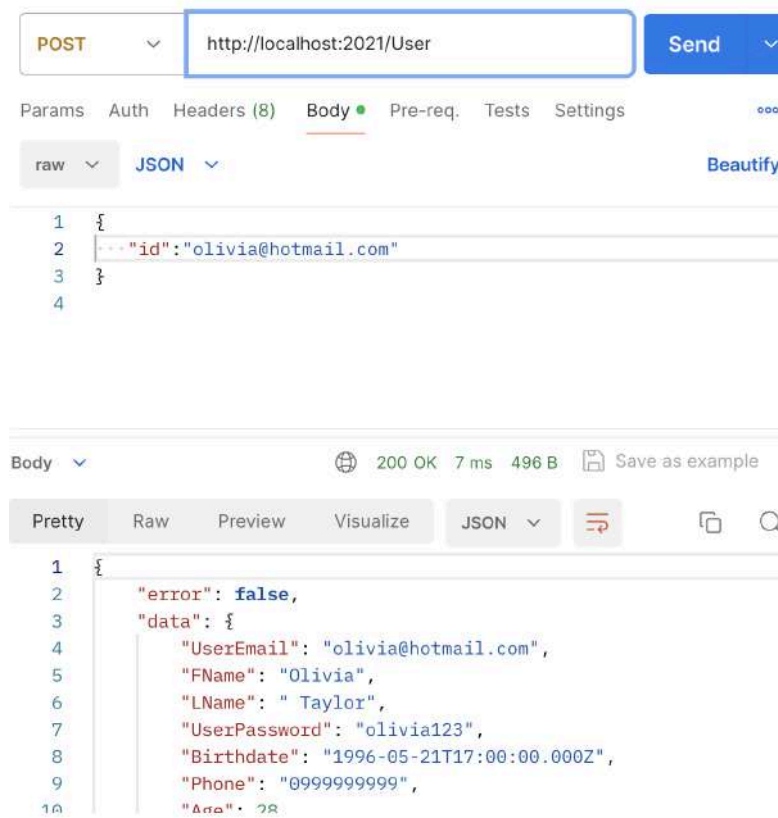
• Select All Users



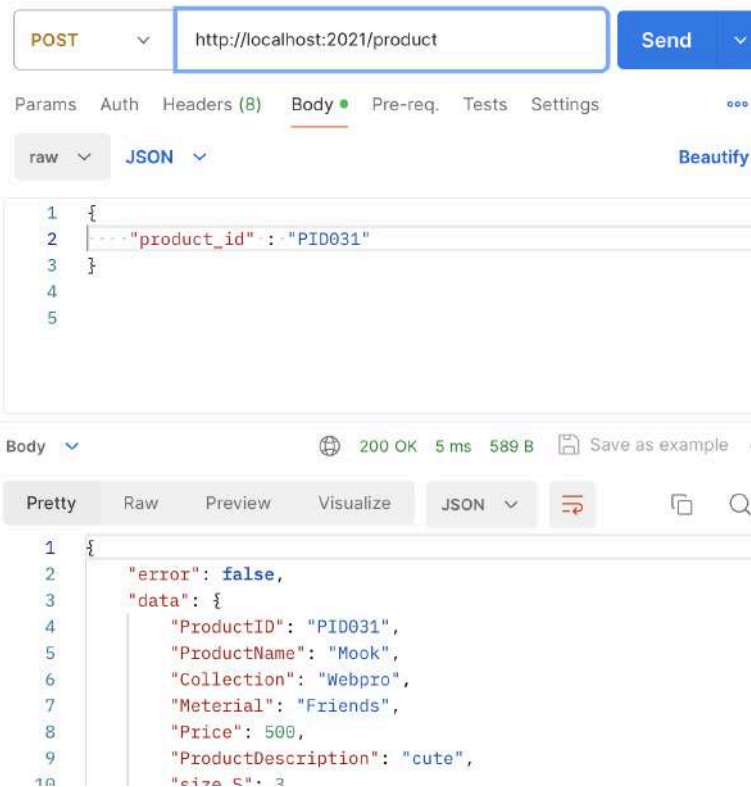
• Select Admin by Admin ID



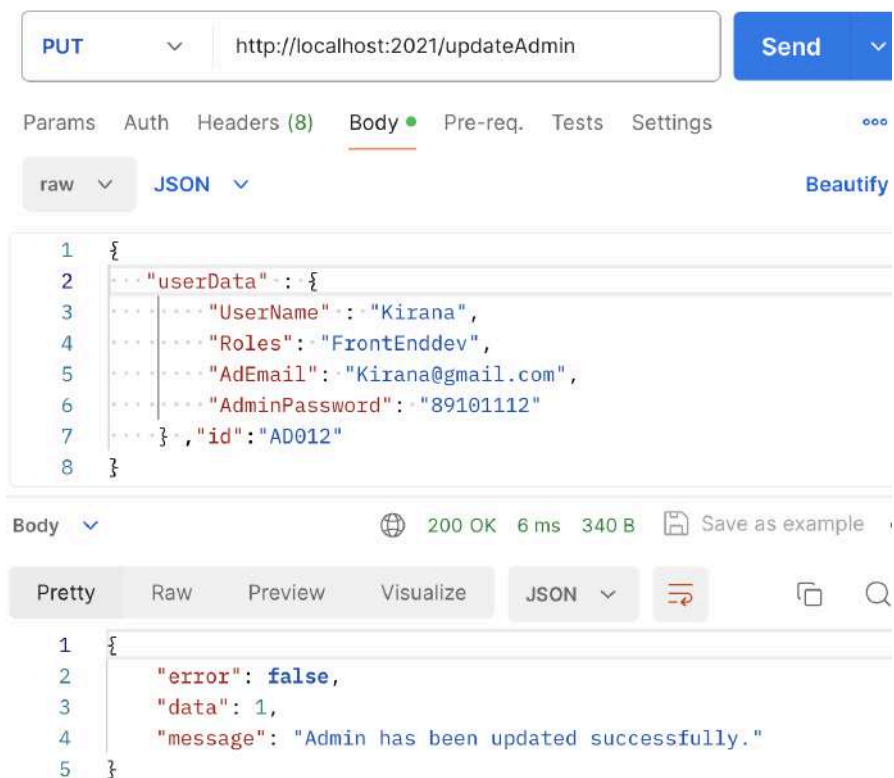
- **Select Users by Email**



- **Select Product by product ID**



- Update Admin



- Update User

PUT Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "userData": {
3     "FName": "Mook",
4     "LName": "Keattitat",
5     "Age": "20",
6     "Phone": "0656656688"
7   }, "email": "jane@gmail.com"
8 }
```

Body 200 OK 35 ms 339 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "User has been updated successfully."
5 }
```

- Update Product

PUT Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "productData": {
3     "ProductName": "Mook",
4     "Collection": "Love",
5     "Color": "pink",
6     "price": "50"
7   }, "id": "PID001"
8 }
9
10
11
```

Body 200 OK 11 ms 342 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "Product has been updated successfully."
5 }
```

- **Delete Admin by Admin ID**

DELETE http://localhost:2021/deleteAdmin

Params Auth Headers (8) **Body** Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "id": "AD012"
3 }
4
```

Body 200 OK 5 ms 345 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "Admin has been deleted from successfully."
5 }
```

- **Delete User by Email**

DELETE http://localhost:2021/deleteUser

Params Auth Headers (8) **Body** Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "id": "Mook@fg"
3 }
4
```

Body 200 OK 11 ms 344 B Save as example

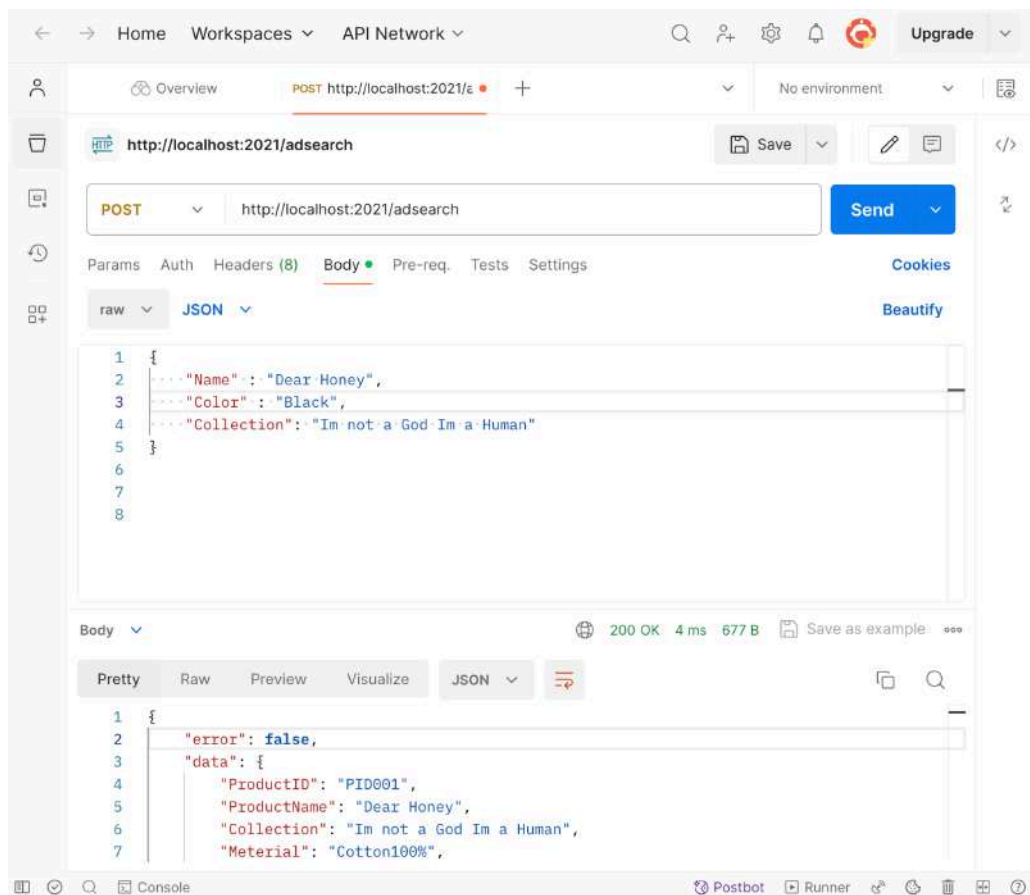
Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "User has been deleted from successfully."
5 }
```

- **Delete Product by Product ID**



• Advance Search



• Insert Admin

POST http://localhost:2021/insertAdmin Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
2 {
3   "admin": {
4     "AdminID": "AD012",
5     "UserName": "Thitiwan",
6     "Roles": "Devops",
7     "AdEmail": "Thitiwan@gmail.com",
8     "AdminPassword": "1234456"
9   }
}
```

Body 200 OK 6 ms 344 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "New admid has been created successfully."
5 }
```

- **Insert Product**

POST http://localhost:2021/insertProduct Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "productData": {
3     "ProductID": "PID031",
4     "ProductName": "Mook",
5     "Collection": "Webpro",
6     "Meterial": "Friends",
7     "Price": 500,
8     "ProductDescription": "cute",
9   }
}
```

Body 200 OK 8 ms 346 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": false,
3   "data": 1,
4   "message": "New product has been created successfully."
5 }
```

Reference

Picture of collection I'm not a god I'm a woman(Graphic t-shirt) Dear honey.[Пин от пользователя Brittany Smith на доске Art / illustrations | Коллажные фотографии, Милые рисунки, Картины \(pinterest.com\)](#)

Picture of collection I'm not a god I'm a woman(Graphic t-shirt) Don't make woman mad.[Pin by bella on dark academia | Renaissance art paintings, Renaissance art, Emotional art \(pinterest.com\)](#)

Picture of collection Rich as hell(Graphic t-shirt) can buy more than 10 porsche[Top Tier Drives: A Closer Look at the Most Opulent Cars Ever Built | Luxury cars, Dream cars, Porsche \(pinterest.com\)](#)

Picture of collection Special day(Graphic t-shirt) I want you to be my christmas[Who doesn't like looking at pictures of cats? ** Check this useful article by going to the link at the image. #cattips | Kitty, Cat memes, Christmas cats \(pinterest.com\)](#)

Picture of collection Rich as hell(Graphic t-shirt) Do ur be(s)t [Pin by francieli on lit. —> born in blood | Aesthetic themes, Dark aesthetic, Mafia \(pinterest.com\)](#)

Picture of collection MR.Bloom(Graphic t-shirt) Hi Mr.Bloom [\(30\) Pinterest, https://www.pinterest.com/pin/537476536795612391/, https://www.pinterest.com/pin/362821313747283765/](#)

Picture of collection I'm not a god I'm a woman(Graphic t-shirt) see me in the crown <https://www.pinterest.com/pin/1009791547687684061/>

Small inspiration [BOYY - Luxury Designer Fashion](#)

Small inspiration https://www.goyard.com/eu_en/