

FINAL PROJECT REPORT

SEMESTER 1, ACADEMIC YEAR: 2024-2025

CT313H: WEB TECHNOLOGIES AND SERVICES

- **Project/Application name:** Furniture Store website
- **GitHub links (for both frontend and backend):** <https://github.com/24-25Sem1-Courses/ct313h03-project-giahungb2111886.git>
- **Demo link:** <https://youtu.be/vgLURx11M7o?si=ztmeJIPR-j3R6iwf>
- **Student ID 1:** B2111886
- **Student Name 1:** Nguyen Le Gia Hung
- **Student ID 2:** B2015007
- **Student Name 2:** Phan Văn Sang
- **Class/Group Number: (e.g, CT313HM03):** M03
- **Task:**
 - + **Nguyen Le Gia Hung:** Frontend, Backend, database, Clip, Word
 - + **Phan Van Sang:** only login register of api backend

Contents

I.	Introduction	3
II.	Details of implemented features.....	3
1.	Feature / Application page 1: Create a new product...	3
2.	Feature / Application page 2: Get product by filter..	7
3.	Feature / Application page 3: Delete all products...	12
4.	Feature / Application page 4: Update product by id_sp 14	
5.	Feature / Application page 5: Delete product by id_sp 19	
III.	Details of implemented features.....	22
1.	Sreach/Filter Products.....	22
2.	Update product.....	23
3.	Add new product.....	25

I. Introduction

- Furniture website

```
• CREATE TABLE `users` (  
    `u_acc` varchar(255) NOT NULL,  
    `acc_pwd` varchar(255) NOT NULL,  
    `u_name` varchar(255) NOT NULL,  
    `u_phone` int(255) NOT NULL,  
    `u_add_street` varchar(255) NOT NULL,  
    `u_add_district` varchar(255) NOT NULL,  
    `u_add_city` varchar(255) NOT NULL,  
    `u_add_province` varchar(255) NOT NULL  
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
• ALTER TABLE `users`  
    ADD PRIMARY KEY (`u_acc`);  
• COMMIT;  
  
• CREATE TABLE `products` (  
    `id_sp` int(10) UNSIGNED NOT NULL,  
    `ten_sp` varchar(100) NOT NULL,  
    `gia_sp` float NOT NULL,  
    `trangthai` tinyint(1) UNSIGNED NOT NULL,  
    `avatar` varchar(50) NOT NULL  
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
• ALTER TABLE `products`  
    ADD PRIMARY KEY (`id_sp`);  
• ALTER TABLE `products`  
    MODIFY `id_sp` int(10) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=23;  
• COMMIT;
```

II. Details of implemented features

1. Feature / Application page 1: Create a new product

- **Description:** Create a new product
- **Screenshots:**

POST

/api/v1/products

Create a new product

^

Create a new product

Parameters

Cancel

Reset

No parameters

Request body

required

multipart/form-data

▼

ten_sp

string

product name

giường ngủ 2m1

☐ Send empty value

gia_sp

string

price

12000000

☐ Send empty value

trangthai

integer

trang thai sp

1

☐ Send empty value

avatarFile

string(\$binary)

Chọn tệp

1d01dbf9620aa954f01b.jpg

☐ Send empty value

Servers

These operation-level options override the global server options.

http://localhost:3000 - Development server

▼

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:3000/api/v1/products' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'ten_sp=giường ngủ 2m1' \
  -F 'gia_sp=12000000' \
  -F 'trangthai=1' \
  -F 'avatarFile=@1d01dbf9620aa954f01b.jpg;type=image/jpeg'
```

Request URL

http://localhost:3000/api/v1/products

Server response

Code	Details
201	<div><p>Response body</p><pre>{ "status": "success", "data": { "product": { "id_sp": 9, "ten_sp": "giường ngủ 2m1", "gia_sp": "12000000", "trangthai": "1", "avatar": "/public/uploads/1728832574884-183535890.jpg" } } }</pre></div> <div><p>Response headers</p><pre>access-control-allow-origin: * connection: keep-alive content-length: 173 content-type: application/json; charset=utf-8 date: Sun,13 Oct 2024 15:16:15 GMT etag: W/"ad-VgG9GhOCwhSrHUtNJ+uq7Wc84I0" keep-alive: timeout=5 location: /api/v1/products/9 x-powered-by: Express</pre></div>

Responses

Code	Description	Links
201	A new product	No links

201 A new product No links

Media type

application/json

Controls Accept header.

Example Value Schema

```

{
  "status": "success",
  "data": {
    "product": {
      "id_sp": 0,
      "ten_sp": "string",
      "gia_sp": "string",
      "trangthai": 0,
      "avatar": "string"
    }
  }
}

```

	id_sp	ten_sp	gia_sp	trangthai	avatar	id_loaisp
<input type="checkbox"/>	Sửa	Chép	Xóa bỏ	9 giường ngủ 2m1	12000000	1 /public/uploads/1728832574884-183535890.jpg

- Implementation details:

- + Does this feature use any libraries other than those introduced in the class (if so, list and state the roles of these new libraries, in both frontend and backend)?
- + Endpoint: "POST /api/v1/products" This is the endpoint that the **createProduct** function in the controller calls to create a new product. This function will receive data from **req.body** and may receive a file from **req.file** if available (for example, a product image).
- + When making a POST request to create a new product, the client sends data in multipart/form-data format. This is commonly used when a file (such as an image) needs to be uploaded along with other data from a form. The expected data structure includes:
 - Parameters in the request body:
 - ten_sp (string): The name of the product.
 - gia_sp (string): The price of the product.
 - trangthai (integer): The status of the product (0 or 1).
 - avatarFile (file, binary): Optional image file for the product
- + When the POST request is successful, the server will respond with a JSON object, usually structured as follows:

```
{
  "status": "success",
  "data": {
    "product": {
      "id_sp": 1,
      "ten_sp": "Tên sản phẩm ví dụ",
      "gia_sp": "100",
      "trangthai": 1,
      "avatar": "/public/uploads/image.png"
    }
  }
}
```

- + 1. Data Storage
 - Table name: products
 - Data Fields: When creating a new product via the POST request, the following fields are stored in the products table: id_sp, ten_sp, gia_sp, trangthai, avatar.
- + 2. Read Data

While the primary function of a POST request is to store data, the system can also read data from the same table to validate or send back a confirmation.

For example, after successfully storing a new product, the details of the created product (including id_sp) can be retrieved from the products table and sent back to the client as part of the response.
- + Client-side States Needed to Implement the POST Feature
 - Input State
 - Validation State
 - Loading State
 - Response State
 - Navigation State

2. Feature / Application page 2: Get product by filter

- **Description:** Get product by filter : Search for product information by product status or product name.
- **Screenshots:**

GET

/api/v1/products

Get products by filter

⌵

Get products by filter

Parameters

Cancel

Name	Description
productStatus boolean (query)	Lọc bằng trạng thái sản phẩm <div>true</div>
productName string (query)	Lọc bằng tên sản phẩm <div>productName</div>
limit integer (query)	Number of records per page <div>5</div>
page integer (query)	Page number of records <div>1</div>

Servers

These operation-level options override the global server options.

http://localhost:3000 - Development server

⌵

Execute

Clear

Responses

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:3000/api/v1/products?productStatus=true&limit=5&page=1' \
  -H 'accept: application/json'
```



Request URL

http://localhost:3000/api/v1/products?productStatus=true&limit=5&page=1

Server response

Code

Details

200

Response body

```
{
  "status": "success",
  "data": {
    "products": [
      {
        "id_sp": 9,
        "ten_sp": "giường ngủ 2m1",
        "gia_sp": 12000000,
        "trangthai": 1,
        "avatar": "/public/uploads/1728832574884-183535890.jpg"
      }
    ],
    "metadata": {
      "totalRecords": 1,
      "firstPage": 1,
      "lastPage": 1,
      "page": 1,
      "limit": 5
    }
  }
}
```



Download

Response headers

```
access-control-allow-origin: *
content-length: 248
content-type: application/json; charset=utf-8
date: Sun, 13 Oct 2024 16:25:17 GMT
etag: W/"f8-yXL8i0fPr2YxMe1bphdNoP3VEfk"
x-powered-by: Express
```

Responses

Responses

Code	Description	Links
200	A list of products	No links

Media type

Controls Accept header.

Example Value Schema

```

{
  "status": "success",
  "data": {
    "products": [
      {
        "id_sp": 0,
        "ten_sp": "string",
        "gia_sp": "string",
        "trangthai": 0,
        "avatar": "string"
      }
    ],
    "metadata": {
      "totalRecords": 0,
      "firstPage": 1,
      "lastPage": 1,
      "page": 1,
      "limit": 5
    }
  }
}

```

– **Implementation details:**

- + The GET feature typically retrieves data from the server without modifying it.
- + Query Parameters:
 - ten_sp, gia_sp, trangthai
 - limit (The number of records to return per page. Default can be set in the backend)
 - page (The specific page number to retrieve, useful for paginated results.)
- + Data Format/Structure Sent
 - When the client makes a request to the API, the query parameters will be sent as part of the URL. For example: GET /api/v1/products?name=ProductA&trangthai=1&limit=10&page=2
- + Expected Response Format:

```

{
  "status": "success",
  "data": {
    "products": [
      {
        "id_sp": 1,
        "ten_sp": "Product A",
        "gia_sp": "20.00",
        "trangthai": 1,
        "avatar": "/public/uploads/product_a.jpg"
      },
      {
        "id_sp": 2,
        "ten_sp": "Product B",
        "gia_sp": "25.00",
        "trangthai": 1,
        "avatar": "/public/uploads/product_b.jpg"
      }
    ],
    "metadata": {
      "totalRecords": 50,
      "firstPage": 1,
      "lastPage": 5,
      "page": 2,
      "limit": 10
    }
  }
}

```

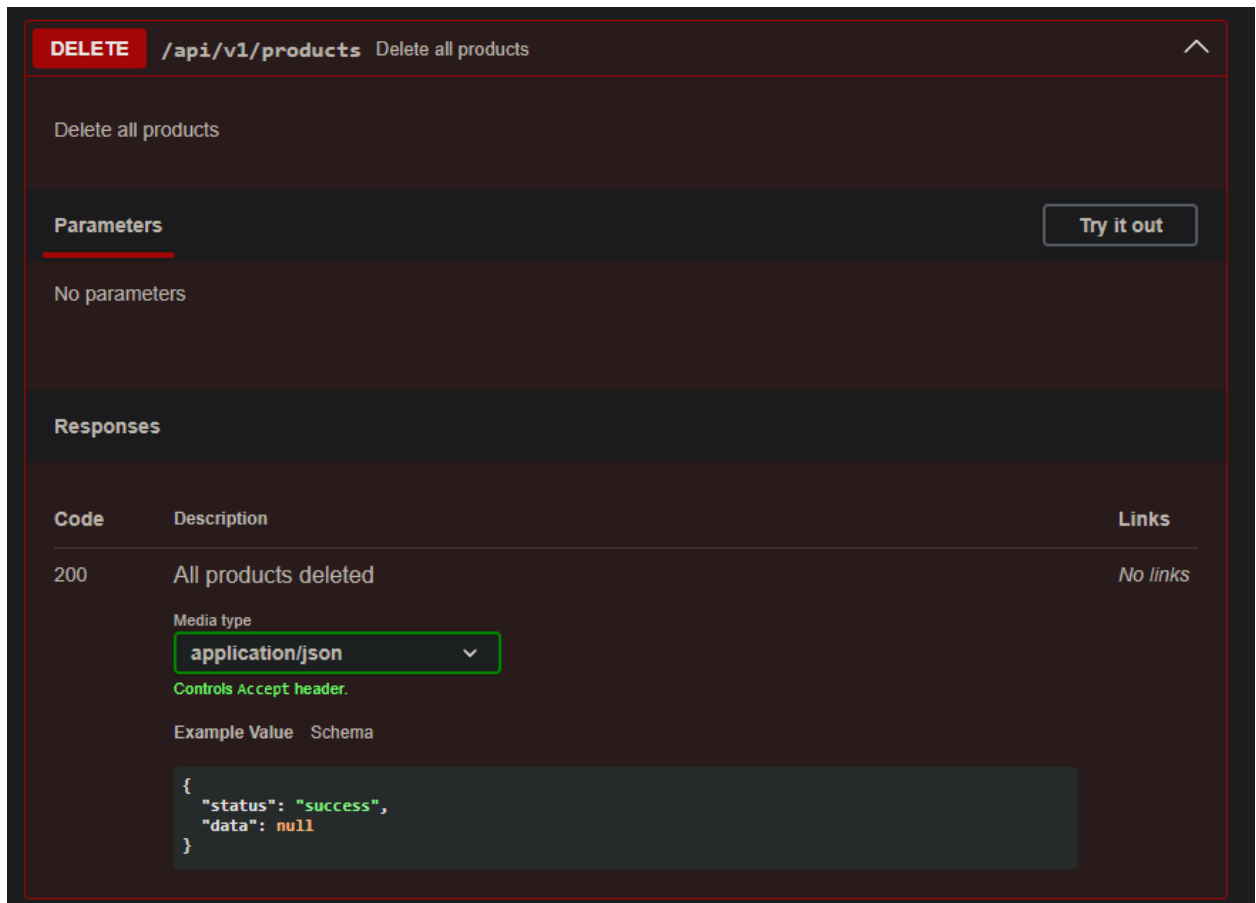
+ Breakdown of the Response Structure

- status: Indicates the response status (e.g., "success").
- data: Contains the actual data returned by the API:
 - products: An array of product objects that match the filter criteria.
 - metadata: Provides additional information about the pagination:
 - totalRecords: Total number of products that match the filter.
 - firstPage: The first page number.
 - lastPage: The last page number based on the total records and limit.
 - page: The current page number requested.

- limit: The number of records per page.
- + Read Data
 - The GET by filter feature is primarily responsible for reading data from the database rather than storing it. This feature allows clients to query and retrieve specific product records based on various filtering criteria, such as ten_sp, gia_sp, and trangthai.
- + Data Source
 - The data is typically read from a products table in the database. This table would include various fields representing the attributes of the products, which may include:
 - id_sp: Unique identifier for each product (Primary Key).
 - ten_sp: Name of the product.
 - gia_sp: Price of the product.
 - trangthai: Status of the product (active, inactive, etc.).
 - avatar: File path for the product's image.
- + Client-Side States Needed to Implement GET by Filter Feature
 - Filter Criteria State
 - Product List State
 - Pagination State
 - Loading State
 - Error State
 - Sorting State

3. Feature / Application page 3: Delete all products

- **Description:** Delete all products : delete all products in database
- **Screenshot:**



- **Implementation details:**
 - + The Delete All Products feature interacts with a specific API endpoint on the server side to remove all products from the database. Here's a detailed description of the relevant API:
 - Endpoint: /api/v1/products
 - Method: DELETE
 - + Data Format/Structure Sent
 - Request Body: Generally, for a DELETE operation that removes all products, you might not need to send a request body. However, some implementations might require an empty body to specify that all products should be deleted. : DELETE /api/v1/products
 - + Data Format/Structure Received
 - Response Body: The server responds with a JSON object indicating the result of the operation. The structure of the response typically includes:
 - status: (string) indicates the status of the response (e.g., "success").
 - data: (object) which may be empty or contain a message confirming that all products have been deleted

- + Read Operation:
- + The feature does not read data before performing the deletion. It simply executes a command to remove all records from the products table.
- + Store Operation:
- + The feature does not store any new data in the database. Its sole purpose is to delete existing data.
- + Database Table Involved
 - Table: The operation directly affects the products table in the database, which contains all the product records. Once the DELETE operation is executed, all entries in this table are removed.
- + Client-side States Needed to Implement the DELETE all products Feature
 - Loading State
 - Confirmation State
 - Success State
 - Error State
 - Product List State

4. Feature / Application page 4: Update product by id_sp

- **Description: PUT by id: Update product by ID**
- **Screenshot:**

PUT

/api/v1/products/{id_sp} Update product by ID

^

Update product by ID

Parameters

Cancel

Reset

Name	Description
id_sp * required integer (path)	Product id <div>9</div>

Request body required

multipart/form-data

▼

ten_sp
string

product name

kính treo tường

☐ Send empty value

gia_sp
string

price

1000000

☐ Send empty value

trangthai
integer

trang thai sp

1

▼

☐ Send empty value

avatarFile
string(\$binary)

Chọn tệp

1d01dbf9620aa954f01b.jpg

☐ Send empty value

Servers

Servers

These operation-level options override the global server options.

http://localhost:3000 - Development server



Execute

Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:3000/api/v1/products/9' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'ten_sp=kính treo tường' \
  -F 'gia_sp=1000000' \
  -F 'trangthai=1' \
  -F 'avatarFile=@1d01dbf9620aa954f01b.jpg;type=image/jpeg'
```



Request URL

http://localhost:3000/api/v1/products/9

Server response

Code Details

200

Response body

```
{
  "status": "success",
  "data": {
    "product": {
      "id_sp": 9,
      "ten_sp": "kính treo tường",
      "gia_sp": "1000000",
      "trangthai": "1",
      "avatar": "/public/uploads/1728840713172-51011298.jpg",
      "id_loaisp": 0
    }
  }
}
```



Download

Response headers

Response headers

```

access-control-allow-origin: *
connection: keep-alive
content-length: 185
content-type: application/json; charset=utf-8
date: Sun, 13 Oct 2024 17:31:53 GMT
etag: W/"b9-rgunWT+gpsQuChTj3Rz1W3rdx/E"
keep-alive: timeout=5
x-powered-by: Express

```

Responses

Code	Description	Links
200	An updated product	No links

Media type

application/json

Controls Accept header.

Example Value Schema

```

{
  "status": "success",
  "data": {
    "product": {
      "id_sp": 0,
      "ten_sp": "string",
      "gia_sp": "string",
      "trangthai": 0,
      "avatar": "string"
    }
  }
}

```

- + The PUT request typically used for updating a product in the server-side API can be described as follows:
 - API Endpoint: /api/v1/products/{id}
 - Method: PUT
 - Description: Updates the details of a specific product identified by its ID.
- + Data Format/Structure Sent
 - The data sent in the PUT request should typically be in the multipart/form-data format, especially if it includes file uploads (like an avatar).
- + Body Structure:
 - id_sp: (integer, required) The ID of the product to update.
 - ten_sp: (string, optional) The name of the product.
 - gia_sp: (string, optional) The price of the product.
 - trangthai: (integer, optional) The status of the product (0 or 1).
 - avatarFile: (file, optional) The new avatar file to upload.
- + Data Format/Structure Received : The response format after a successful update is usually in JSON format. Here's an example of what the response might look like:

```
{
  "status": "success",
  "data": {
    "product": {
      "id_sp": 1,
      "ten_sp": "Updated Product Name",
      "gia_sp": "200.00",
      "trangthai": 1,
      "avatar": "/public/uploads/new-avatar.jpg"
    }
  }
}
```

- + Data Read/Store Operations
- + Reading Data:
 - When a PUT request is made to update a product, the server typically first reads the existing data of the product using its ID (id_sp). This is to ensure that the product exists and to retrieve the current information before making updates.
- + Storing Data:
 - After validating the incoming data, the server updates the existing product record in the database. This involves modifying the fields specified in the PUT request (such as ten_sp, gia_sp, trangthai, and possibly avatarFile).
- + Database Table
 - Table Name:
 - The data for products is usually stored in a database table named products or Products, depending on the naming convention used in the database schema.
- + Client-Side States:
 - Product ID (id_sp): This state holds the unique identifier of the product that is being updated. It is necessary to identify which product in the database should be modified.
 - Product Form Data
 - Validation States: States to track validation errors or success for the input fields, ensuring that the data entered meets the requirements (e.g., non-empty string for ten_sp, valid price format for gia_sp).
 - Loading State

- Error Handling State
- Success State:

5. Feature / Application page 5: Delete product by id_sp

- **Description:** DELETE by id: DELETE product by ID
- **Screenshot:**

Delete product by ID

Parameters

Cancel

Name	Description
id_sp * required integer (path)	Contact ID <input type="text" value="9"/>

Servers

These operation-level options override the global server options.

http://localhost:3000 - Development server

Execute

Clear

Responses

Curl

```
curl -X 'DELETE' \
'http://localhost:3000/api/v1/products/9' \
-H 'accept: application/json'
```

Request URL

http://localhost:3000/api/v1/products/9

Server response

Code

Details

Request URL

`http://localhost:3000/api/v1/products/9`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "status": "success", "data": null }</pre> <p>Response headers</p> <pre>access-control-allow-origin: * connection: keep-alive content-length: 32 content-type: application/json; charset=utf-8 date: Sun, 13 Oct 2024 17:46:44 GMT etag: W/"20-bff5r/a5MyNNWly9hjn8a8pOLDxA" keep-alive: timeout=5 x-powered-by: Express</pre>

Responses

Code	Description	Links
200	Product deleted	No links

Media type:

Controls: Accept header.

Example Value:

```
{
  "status": "success",
  "data": null
}
```

- **Implementation details:**
 - + Server-Side API for PUT Feature
 - Endpoint
 - + URL: `/api/v1/products/:id`
 - + Method: PUT
 - + Description
 - This endpoint is used to update an existing product identified by its unique ID (`id_sp`). It expects to receive the product details in the request body and returns the updated product details in the response.
 - + Data Format/Structure
 - Request Body: The data sent to the server for updating a product is typically formatted as multipart/form-data. Here's a breakdown of the expected structure:

- + Fields:
 - ten_sp (string, required): The name of the product.
 - gia_sp (string, required): The price of the product.
 - trangthai (integer, optional): The status of the product (0 or 1).
 - avatarFile (file, optional): The avatar image file for the product.

- + Response Structure: Upon successful update, the server responds with a JSON object indicating the status of the operation and the updated product details.
 - Response Format:
 - + Status: success
 - + Data:
 - product:
 - id_sp: (integer) The unique identifier of the product.
 - ten_sp: (string) The updated name of the product.
 - gia_sp: (string) The updated price of the product.
 - trangthai: (integer) The updated status of the product.
 - avatar: (string, optional) The file path of the updated avatar image.

- + Data Reading and Storage
 - Reading Data:
 - + When a PUT request is made to update a product, the feature first reads the existing product details from the database using the product ID (id_sp) provided in the request URL. This ensures that the product exists before attempting to update it.
 - Storing Data:
 - + After reading the current data, the feature then updates the product information in the database. The updated data is stored in the products table, which contains the relevant fields for each product

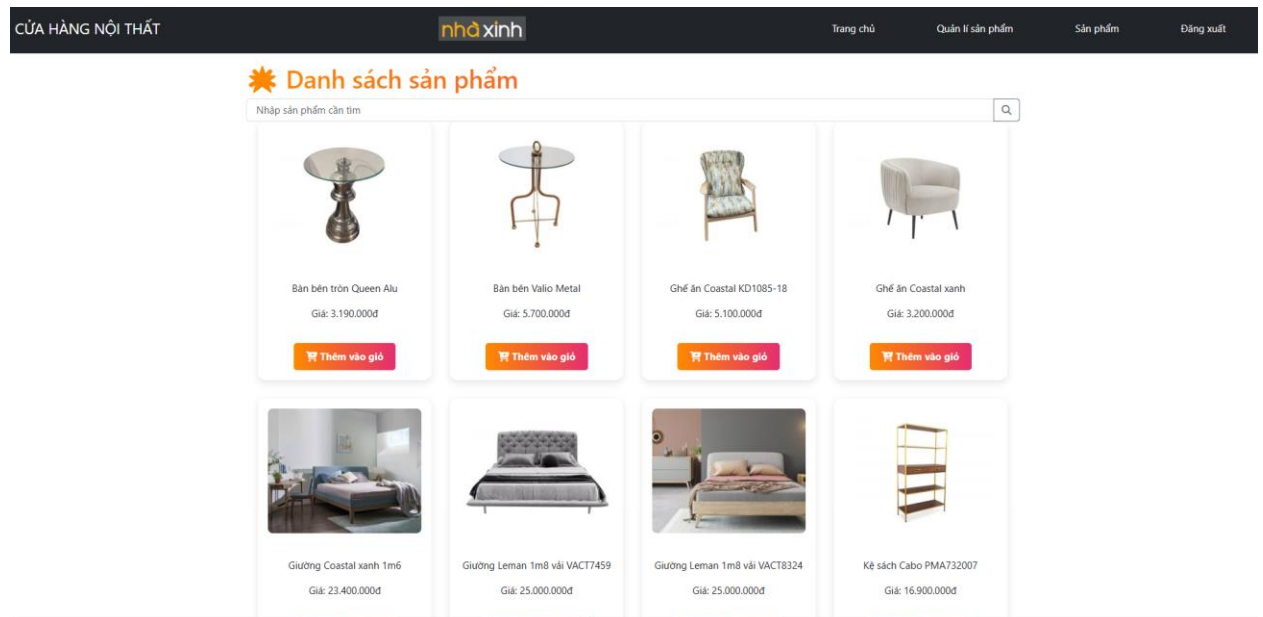
- Client-Side States
 - Product ID State (productId):
 - + This state holds the unique identifier of the product that is being updated. It's used to construct the API endpoint for the PUT request (e.g., /api/v1/products/{id}).

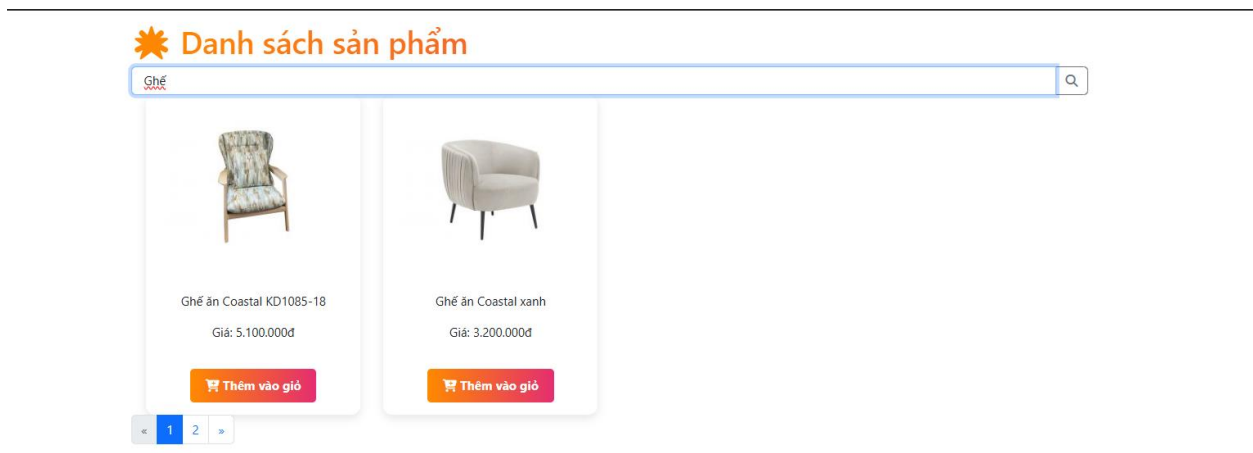
- + Product Form State (productForm):
 - This state contains the data related to the product being updated. It usually includes fields such as:
 - + ten_sp: Product name
 - + gia_sp: Product price
 - + trangthai: Product status
 - + avatar: File upload for the product image (if applicable)
- + Loading State (isLoading)
 - A boolean state to indicate whether the PUT request is in progress. This can be used to show a loading spinner or disable the form while the update is being processed.
- + Error State (error):
- + Success State (success)
- + Avatar Preview State (avatarPreview) (optional)

III. Details of implemented features

1. Sreach/Filter Products

- **Description:** This function enables users to sreach for needed products through sreach bar or filter through a list available products
- **Screenshots:**



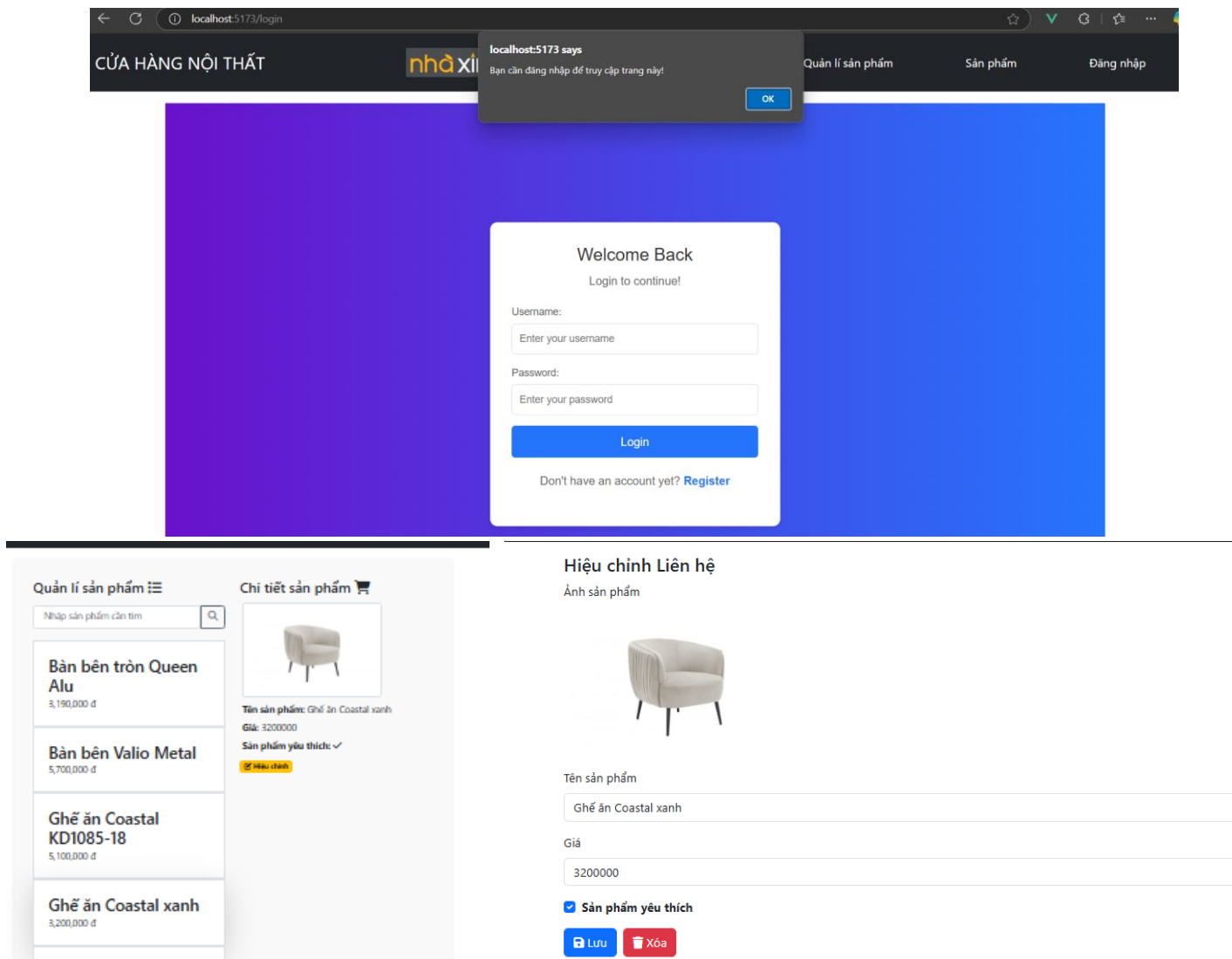


– **Implementation details:**

- + Does this feature use any libraries other than those introduced in the class (if so, list and state the roles of these new libraries, in both frontend and backend)? *This feature does not use any libraries other than those introduced in the class*
- + Which server-side APIs does this feature use? Describe the APIs (endpoint, data format/structure sent and received). *GET /api/v1/products (Detail is explained in Backend section)*
- + Does this feature read/store data? In which table? *This feature read the data from Laptops table, return the appropriate values, and show the result in the laptop list section*
- + Which client-side states are needed to implement this feature? *Require user to log in before running*

2. Update product

- **Description:** The update product functionality allows users to edit information about an existing product. When the user submits the form, the component calls the updateProduct API to update the data on the server. If successful, display the message "Product updated successfully"; if unsuccessful, log an error to the console.
- **Screenshots:**



- + Does this feature use any libraries other than those introduced in the class (if so, list and state the roles of these new libraries, in both frontend and backend)?

Frontend:

@tanstack/vue-query: Manages fetching, caching, and updating server-state data (used for product data fetch and updates).

Backend:

multer: Handles file uploads (e.g., product avatar).

fs: Deletes old avatar files from the file system.

ApiError: Standardizes error handling.

JSend: Structures API responses in a consistent format.

Database library (likely Knex.js or Sequelize): Manages database queries (fetching and updating products).

- + Which server-side APIs does this feature use? Describe the APIs (endpoint, data format/structure sent and received).

GET /products/:id_sp (Retrieves details of a specific product by its ID.)

PUT /products/:id_sp (Updates details of a specific product)

DELETE /products/:id_sp (Deletes a specific product by its ID)

- + Does this feature read/store data? In which table?

This feature read the data from products table, update, modify and delete items from products table

- + Which client-side states are needed to implement this feature?

Require user to log in before running

3. Add new product

- **Description:** This function to add a new product to the products table in the database
- **Screenshots:**

Thêm sản phẩm mới

Ảnh sản phẩm



Tên sản phẩm

Ghế dài sofa

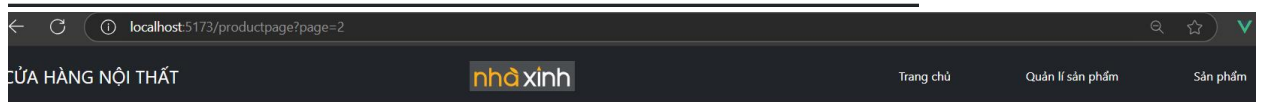
Giá

120000000

☒ Sản phẩm yêu thích

Lưu

Sản phẩm được thêm thành công!



Danh sách sản phẩm

Nhập sản phẩm cần tìm



Bàn cao cấp

Giá: 10.000.000đ

Thêm vào giỏ



Bàn kính

Giá: 3.400.000đ

Thêm vào giỏ



Ghế dài sofa

Giá: 120.000.000đ

Thêm vào giỏ

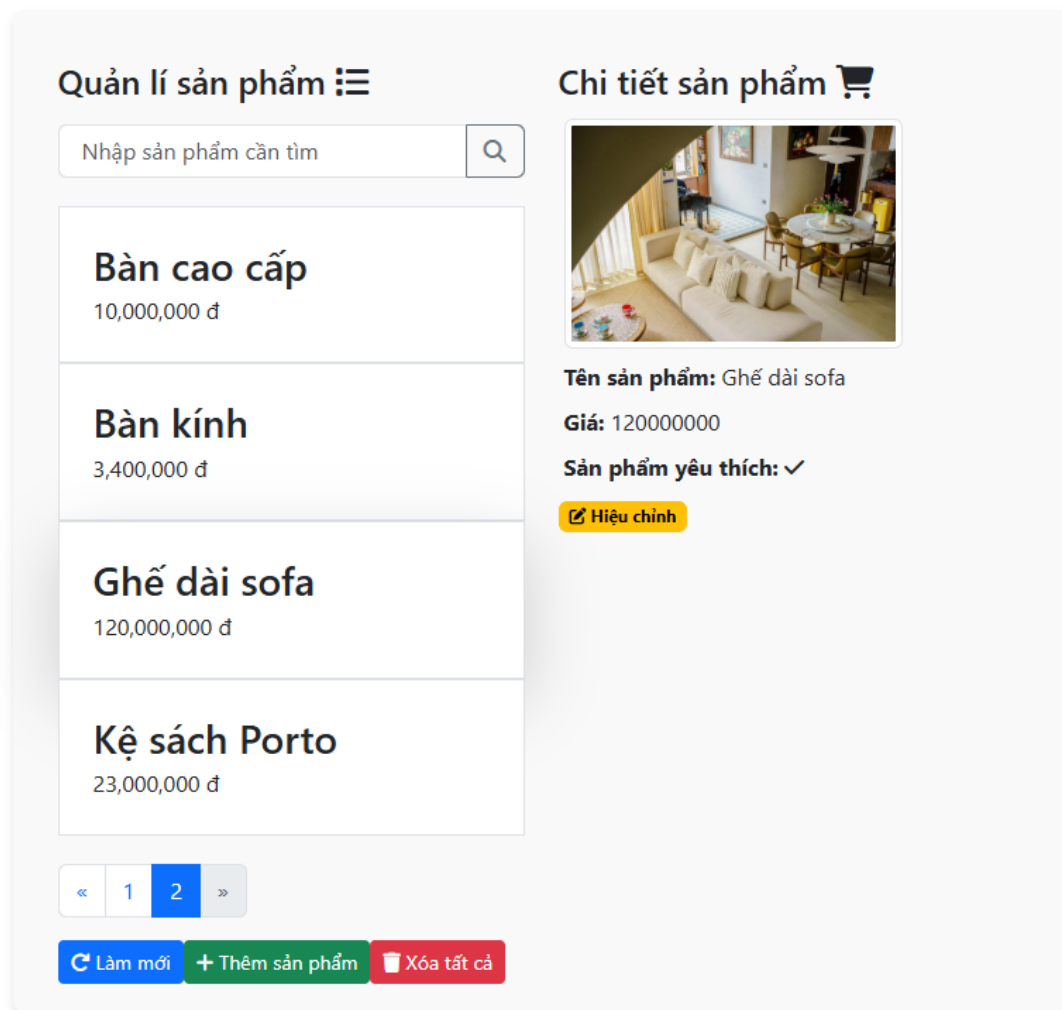


Kệ sách Porto

Giá: 23.000.000đ

Thêm vào giỏ

< 1 2 >



- + Does this feature use any libraries other than those introduced in the class (if so, list and state the roles of these new libraries, in both frontend and backend)? *This feature use “provide” and “inject” API from Vue library to create a global variable to manage the user account state and information (log in state, u_acc, acc_pwd)*
- + Which server-side APIs does this feature use? Describe the APIs (endpoint, data format/structure sent and received).

(Detail is explained in Backend section)

GET /api/v1/products – Get all products information

POST /api/v1/products – Create a new product in products table

DELETE /api/v1/products – Delete all products from the products table

- + Does this feature read/store data? In which table? –

This feature read the data from Laptops table to display them into the table, make it easier to manage.

- *Moreover, “Add” button is used to create a new laptop with the input from the following form while “Delete All” button is used to clear all the items in Laptops table*
 - *The values of the table are automatically updated*
- + Which client-side states are needed to implement this feature? *Require user to log in before running*

HẾT