RMIT UNIVERSITY

ASSESSMENT 2

FULL STACK WEB APPLICATION

Course: COSC2430- Web Programming

Group Name	CSS:GO	
Practical Tutor/Time/Day/Loc	Mr.Tom Huynh	
Video Link URL	https://youtu.be/HI9ww7b5c0c	
Prototype Link	https://group9ecommerceweb-dolelongan.b4a.run/	
Pages	11 pages (4 - 15)	
Group Members	**Name**	**Student Number**
1	Do Le Long An	s3963207
2	Truong Hong Van	s3957034
3	Bui Tuan Anh	s3970375
4	Lao Vinh Khang	s3891925
5	Pham Le Quynh Anh	s3927427

Table of content

I. Introduction

Online platforms have taken the place of traditional retailers thanks to the e-commerce sector's rapid development. Online shopping offers convenience with a friendly user interface, and easy access to product information and reviews make it a preferred choice for customers. Lazada is one of the most popular online shopping platforms in Vietnam (VNExpress International January 2023). It offers a wide range of products and services to cater to the Vietnamese consumer.
The main objective of this project is to attract a wide variety of customers, especially those who might not be familiar with online shopping, by offering a clear navigation, design, and process. We aim to improve customer satisfaction, engagement, and eventually platform sales by improving the user experience.
Based on that objective, our intention is to create an e-commerce website that draws inspiration from Lazada but simplifies the user interface and functionality. Users will be able to access all the products and track their deliveries. Essential features like user registration, product classification, and search options will be given priority. Our website will provide all functions for vendors as well as shippers. Additionally, we want to implement a responsive design that ensures an optimal user experience across various devices.

II. Project Description

A. UI requirements

At the preparation stage, we decided to design the UI via Figma, from which we can plan the ideas and finalise the design before moving on to the coding work. To view the prototype, please visit this Fig30ma link
In general, most of the page would have the navigation bar and the footer including the helping links for the users. (Figure UI header) (Figure UI footer). In terms of the responsiveness of the website, we have adapted our website to desktop, mobile and tablet (Figure UI desktop) (Figure UI phone) (Figure UI tablet).

B. Login/Sign Up

The Sign Up page allows the user to register as any role, such as customer, vendor, or shipper. The user can switch between different registration forms with three buttons. It will direct the user to different forms for signing up. There will be some constraints for the username and password validation on the client side and server-side..

After registration, the user can login using their registered username and password. The server will check back with the database to validate the role of the user and direct the user to the correct page.

C. Customer

As customers, they will be able to start shopping at Lazada with all the functionalities, such as seeing product details, adding products to carts, proceeding to checkout, and viewing order history. The Home Page **(Figure 1)** with all the functionalities will be described as below:

Function	Description
View Product Detail	Allows the user to see the product details, such as price, description, image, and reviews of the product
Search and filter products	The users can search for a particular product name on the search bar, and filter the products based on the price range.
Add to cart	Customer can add to cart any product they want to buy and can see the cart before process to checkout
See the order history	After finishing the order, the customer checks the order history and the order status.
Change account details	Customer allows to change their name, address and profile picture if wanted

D. Vendor Dashboard

As vendors, they are offered a wide range of functionalities related to products. In Particular, they can perform CRUD operations on each product in the Vendor Dashboard, and on their user's profile as well.

Function	Description
View vendor dashboard and product	Vendors can views the two main parts 'View all products' and 'Add new products'
Add new product	Allow vendors can add new products accompanied with informations such as name, description, price, category and image
Edit product detail	Vendor can update any fields such as name, description, price, category if necessary
Change account details	Users are at liberty to make any necessary adjustments to the personal information that they entered during the registration process.

E. Shipper Dashboard

In terms of the Shipper role, a shipper, in the registration stage, could choose his preferred Distribution hub to work on. By default, four distribution hub options are provided, namely Ahamove, GHTK, Giao Hàng Nhanh, and Viettel. After logging in, the shipper will be directed to the Shipper Dashboard.

Functionality	Description
View active orders on the Distribution Hub	From the orders stored in the MongoDB database, the system will query the Order collection, based on the active status, and the

	associated Distribution hub name which matches the assigned hub of the current shipper. All the possible orders will be then displayed on the Dashboard.
View details of a specific order	The shipper can click on the Order ID to view the Order details, including the customer name, the customer address, each item in the cart, along with its price and quantity, and the total price.
Update an order's status	The shipper can either change an order's status to "delivered" or "cancelled".
View the user's profile and change details	Similar to the customer and the vendor roles, a shipper could also view and modify his user's profile, with all the information stored in the registration phase.

III. Implementation Details

A. Non-functional

A.1: NPM packages

NPM package	**Description**
express-ejs-layout	By this package, we are able to define layout templates for the pages in the website. In other words, we can avoid the repetition of the same components and partials over multiple .ejs files. The layout templates (homeLayout, loginLayout) can be viewed in the source code.
body-parser	Data from HTTP request bodies. A web browser or mobile app can submit an HTTP request to a server with a data body. Formats include JSON, XML, form data, and plain text. The body-parser is used to extract data from users when they submitted form, the form data should convert to JSON file then its could be save in database, or use for authentication
dotenv	Dotenv is a useful tool for managing and accessing environment variables because it loads them automatically from the.env file. This facilitates better configuration management in various environments and makes it much easier to keep sensitive data isolated from your code.
express-async-handler	Aid the error handling process for the website in asynchronous route handlers. Particularly, the 'asyncHandler', when wrapped by the route handler function, will catch any possible errors, then pass them to the error handling middleware. Hence, it enhances the code maintainability and error management's efficiency.
morgan	Aid in monitoring the HTTP requests made by your Node.js

	applications. Request details such as method, URL, response status, response time, and more can be easily recorded and retrieved later. Morgan is often used in combination with web frameworks like Express to boost logging capabilities.
multer	Node.js middleware for handling multipart/form-data, which is primarily used for uploading files.
multiparty	A module for Node.js that is typically employed for the purpose of parsing and handling multipart/form-data requests. When uploading files or submitting forms that need file uploads, this kind of request is commonly utilised. The procedure of obtaining data and files from these requests is made more straightforward by the package.
node-localstorage	A module for Node.js that offers an implementation of the localStorage API in a manner that is analogous to that which is offered by web browsers. Using a user interface that is both straightforward and intuitive, it enables you to save data locally on the server in the form of key-value pairs.
bcrypt	Generate hash password

A.2: MVC Model

We've settled on the MVC model because it provides a solid foundation upon which to build an efficient and effective code structure for our project.In definition, MVC is a software design pattern that is frequently utilized for the development of user interfaces. It splits the relevant program logic into three pieces that are coupled with each other. This is done to ensure that the internal representations of information are kept completely distinct from the methods in which information is displayed to and accepted from the user.

A.3: Bootstrap

Bootstrap 5 framework is imported and utilized in the building of the website, with a view to alleviating the styling process, as it is designed to be responsive in a wide range of screen widths. We mainly used Bootstrap 5's grid system for responsiveness, and default packages for other styling. Moreover, it is convenient that we can customize the built-in BS5 properties to meet our needs.

A.4: MongoDB (mongoose)

The Mongoose library for Node.js is an Object-Document Mapping (ODM) library that was developed specifically for use with MongoDB. It makes it easy to interface with databases stored in MongoDB using JavaScript by providing this service. By providing a higher-level application programming interface (API) and abstracting the underlying MongoDB driver, Mongoose makes it simpler to establish schemas, construct models, carry out database operations, and manage the relationships between data.

A.5: Validation Service

In terms of website quality, CSS Validation Service (W3C) is chosen to validate the quality of the CSS files. The result is that most of our source code for styling is acceptable and the web pages are

interoperable. The result of the validation is shown below. **(Figure 2)**. With regard to HTML Validation, we were not able to assess individual pages but the whole website address at once, as we prioritised using EJS documents over HTML files. We believe that this is a satisfactory decision, as initially, we are able to embed Javascript logic within HTML markup to generate dynamic content. With the help of EJS layout, we could combine the common webpage parts to create templates, as can be seen in the *'partials'* and *'layout'* folders. For example, the navigation bar, or the header that every page has, is built only one time in the layout, then, other pages can extend and override specific layout sections in individual EJS files. When the request is sent to the server, we can add the server-side logic for the matching page content to be rendered in the *<% body %>* section of the layout. Therefore, we believe that this approach enhances the code reusability, maintainability, and time efficiency. The result of HTML validation of the entire website is shown in **(Figure 2)**.
Overall, although there have been minor errors, our website is examined as relatively user-friendly and interoperable.

A.6: Back4App

Container-as-a-service, a cloud service that helps developers deploy, run, scale, control, and stop containers by applying virtualization on container level.

B. Functional

Regarding Error Handling, the response status is set to 500, indicating Internal Server Error, and a message will be displayed for the user. Otherwise, the response status is set to 404, Not Found.

B.1 Register:

B.1.1: Common registration aspects:

For registration, there are several primary fields that are applicable to all three roles **(Figure 3).** We have some important information, like the username, the password, and the image. Users are able to register themselves by entering the necessary information into these sections. Particularly, we have included some password restrictions in order to demonstrate to users whether or not their passwords are appropriate.
Once registered, the server-side will validate the information that the user has provided with the database and return errors, such as Username already exists, if needed. User's **username** and **password** will be checked throughout all roles to see if they are matched with the conditional regex (*'usernameRegex'* and *'passwordRegex'*).
Last but not least, to make the password more secure, we used the npm bcrypt to convert the password into hash and store the hashed password as the data .We allowed users to upload profile picture as png or jpg file, the image was stored as binary data using npm multer. Last but not least, if the condition was false, return the error to user (*'originURL')*; otherwise, the username and password are correct and saved to MongoDB.

B.1.2: Vendor

In order to register for a vendor, we have the button to switch to the *'register-vendor'* page. In this UI vendor page, there are two extra fields: the business name is set to a unique constraint in all user

Vendor and business address. On the server-side, we used the function **createVendor** to register the account with ‘role’ is ‘vendor’.

B.1.3: Customer

On the customer's form, there are some changes in information input fields. Instead of viewing ‘business name’ and ‘business address’ like a vendor, customers are shown ‘name’ and ‘address’. On the server-side, we used the function **createCustomer** to register the account with ‘role’ is ‘customer’.

B.1.4: Shipper

Last but not least, in the shipper registration page we changed the fields to name which are unique and distribution hubs. The distribution hubs were displayed in a drop box. On the server-side, we used the function **createShipper** to register the account with ‘role’ is ‘shipper’ and set the hub that they choose.

B.2: Login

- Whenever a user logs in, the system checks the username and password in ‘***loginCtrl’***. It will check if the username exists or not, and if it does, the system will check the password. For the password, the database will store the password as a hash password. We call the function *isPasswordMatched* in **userModel** to compare the hash password in the database with the user’s input password.
- After validating the username and password, our ‘***loginCtrl’*** will generate a token and attach to that user. The token will expire after 3 days.
- If the account does not have a specific role and tries to get the route of other user roles like (/home or /vendor), our website will show a custom Error Page. In other words, ***‘authMiddleware’*** will prevent the users from accessing unauthorised pages.
- As for the limitations of the login UI, the “forget password” and "remember password" buttons are not functional; they are a static component on the website, which we will implement in the future.

B.3: Customer

- After successfully logging in as a customer, the server-side (‘***authMiddleware’, ‘userController’, and ‘authController’)*** will route the user based on their token and role. Customers can start to view products from all vendors. To view any product, the user can simply click on that product, and it will direct the customer to the product detail page (*/home/product/<%= product._id%>)* that links to the product ID based on the product that the user clicks on. Another way to view a specific product is to use the search engine on the homepage, the Customers can further filter the min and max price.
- On the server-side, clicking on *“addToCart”* will take the product._id, product.price, and product_name and send them back to the “*cartPage”*. The cart will be stored in the local storage HTML 5 with all the products that the customer wants to buy and the total price, which is all handled by *‘cart.js’*. After reviewing the cart, the customer can choose the Distribution Hub delivery. Once clicked on “Submit Order”, all the data (product name, price,and id, customer name, customer’s address) from the cart local storage is then fetched to *“/order/submit”* with the GET method, which sends the data to the shipper dashboards. The customer can also clear the cart with the “Clear cart” button or ‘Remove’ button to individually remove the product from the cart local storage. For the order history, the customer can view the order that they have made, such as total price and status of the order.

B.4: Vendor

B.4.1: Vendor dashboard

After logging in successfully as a vendor role, users will be directed to the *'Vendor-dashboard'* **(Figure 4)**. The side-nav-bar allowed vendors to navigate between pages, including viewing all products they have added so far, viewing their profile information they added in registration phase and *'log out'* which is the exit button back to the login page.

B.4.2: View all product

- Vendors can view all the product details when located in *'view all product page'*. All information such as name, price, description, category, productNotes, and image, were displayed inside. On the server-side, we sent a request to the database for any and all items that matched the criteria specified by using the v_id that was retrieved from the req.user._id property. As a result of this, we were able to offer each product its own v_id that is completely unique **(Figure 5)**.
- Additionally, there were two **extra** functionalities in the 'view-all-product' page which are update and delete. The vendors can click on the button according to the actions that they want to perform, which is then handled on the server-side. In terms of the functionality of update ('updateProduct'), this function applied four same fields with 'submit-product', which are 'name', 'price', 'description' and 'category'. Then we queried productID to find the existence of products inside the database, if not we return errors to clients, if it exists we update products together with four fields we set above. After updating successfully, redirect latest version products to view all product page **(Figure 6)**. The delete version was familiar with the update one. The only difference is that in this case we used *'findbyIDAndDelete'* in order to delete productID that existed in the database, if we could not find the ID, errors were sent to clients **(Figure 7)**.

B.4.3: Add a product

When clicking on the *'add more product'* button in the vendor dashboard, the page is switched to a 'create product' page. Here, there will be a form that contains fields such as product name, price, category, productNotes, image **(Figure 8)**, where we create the data of a product such as the file's name, category, price, and product details. The extracted data is utilised to build a new product object, and the database is updated to include the name of the uploaded file. When a product is added, the vendor ID of the person who created it is automatically added to the database so that it is known who owns it. After they finish creating the product, the user will see a success message. If the user cannot create a product, an error message will appear to let the user know what the problem is right on the create product page, which increases the UX for the vendors.

B.5: Shipper

On server-side, the routes of the shipper-related pages are handled by the *'shipperRouter'* file, which receives requests and renders a page with the matching route for the display.

Similar to the vendor dashboard, Shipper Dashboard contains a side navigation menu, which provides functionalities such as view the user's profile, return to the Dashboard, and log out. The Shipper's functionalities are all handled by the shipper controller file.

In the Shipper Dashboard **(Figure 9)**, the active orders of the assigned Distribution hub are displayed as cards, so that the user can easily view and choose a specific order.

B.5.1: View active orders

In terms of functionality, the shipper can view all the active orders of the assigned Distribution hub, which has status as 'active', which is handled by the *'checkShipperHub'* function. This function consists of three parameters, *'req'*, *'rep'* and *'next'*, which is a callback function used to pass the control to the next middleware. Initially, the *'hubName'* is extracted from the *'req.user'* object, which is the current shipper. Next, a database query is performed to filter the active orders associated with the extracted *'hubName'* in the Order collection. The result of the query, and the *'hubName'* are then assigned to *'req.shipperOrders'* and *'req.hubName'* respectively, which enables the data to be accessed by subsequent route handlers. Finally, The *'next'* is used to pass control to the next middleware in the chain.

The *'shipperDashboard'* function is responsible for rendering the Shipper Dashboard page, whose route is *'shipper-page/shipper-dashboard'*. To be specific, the *'orderId'* is extracted from the *'req.query'* (assume that the 'orderId' is passed as a query parameter in the request URL) and looked up in the database. If the order is found, the page, with a specified layout path, is rendered in that particular way and is passed as data for the view. If no 'orderId' is provided, the query *'Order.find()'* filters and retrieves the orders based on the *'req.hubName'*. As a result, all the active orders will be displayed in the Dashboard.

B.5.2: View details of a specific order

When clicking on an Order ID, the shipper can view the details of that order **(Figure 10)** . This is achieved by the system retrieving the data based on the Order ID. Particularly, the 'orderId' parameter from the request's URL is extracted using *'req.params.orderId'* and looked up in the database by *'Order.findById()'*. If the order is found, the order detail page (*'shipper-page/orderDetailShipper'*) is rendered, with the order and cart's information passed as data for the view. Otherwise, the response status is set to 404 Not Found and the error handler throws a message *'Order not found'*.

B.5.3: Update order status

The shipper can either accept or cancel an order, whose feature is handled by *'updateOrderStatus'* and *'updateOrderStatusCancel'* functions. The sole difference between the two functions is the first function saves the status to 'delivered', whereas the second one handles the 'cancelled' status.

In terms of the workflow, the *'orderId'* from the request's parameters is first retrieved and then found in the database. The resulting order's *'orderStatus'* property is then changed to *'delivered'* or *'cancelled'*, and saved using *'order.save()'* command, which ensures that the order's status is updated successfully in the database. After that, the updated order will no longer be displayed on the Shipper Dashboard, and the shipper will be redirected to the Shipper Dashboard page. In the event of errors, the response status is set to 500 (Internal Server Error).

B.6: List of extra features:

General:

- Click on 'my profile' have the logout button **(Figure 11)**
- Back button go back to previous page **(Figure 12)**

Page	Extra features	Description
Custom Error Handling Page	Use Unsplash API to generate random image and the custom page itself to handle all errors with '**Go Back'** button	When the users accidentally encounter an error or access a route that does not exist on the website, they will be directed to this page **(Figure 13)**
Login and Register	Display password validation	We have the check password display in the register page which helps users to determine whether password is correct or not **(Figure 14)**
	Toggle eye to make passwords visible	This allow users to see their password when press in this button **(Figure 15)**
	Username validation	When users type username,its do not meet the requirement error appeared, they need re-submit form **(Figure 16)**
	Check validate for passwords	When users type passwords, its do not meet the requirement error appeared, they need re-submit form **(Figure 17)**
	Name validation	When users type name, its do not meet the requirement error appeared, they need re-submit form **(Figure 18)**
	Error query message	Once checked user's input in the server-side, if the information is incorrect, the error message will display right on the form and on the link **(Figure 19)**
Vendor	Perform CRUD operations on products	In 'view all product', beside view products, vendors can update and delete as well **(Figure 20), (Figure 21)**
	View image	View image has added **(Figure 22)**
Customer	Footer 'Bubble' effect	Using JavaScript to locate the 'location' of the cursor on the page through clientX and clientY, the 'bubbling' effect will display at the end of the page **(Figure 23)**
	Navigation bar: display current number of items in cart	Show the number of products customers order in cart using local storage html 5 **(Figure 24)**
	Order history	Show order status, the status changes according to shipper action in shipper-dashboard
	Review Tab	Next to the Product Description Tab and in the Product detail page, once clicked, it will display a Review form
All	Alert box	In addition to other error handlers, the multiple alert

		boxes on different pages are responsible for alerting the action of the user is valid or not. **(Figure 25)**

IV. Conclusion

Building an online store requires extensive preparation, meticulous attention to detail, and a focus on the customer experience. When creating something new, it's important to keep many things in mind, such as using the right technology stack and implementing the best practices and Project Development methodologies.

In addition, there are some drawbacks to operating an ecommerce website. The biggest drawback is scalability; it is important that the online store can accommodate more customers and more transactions as the company expands. The second one is maintenance and updating. E-commerce websites have special maintenance needs, such as frequent updates, bug repairs, and the installation of security patches. Maintaining a website's uptime and security requires dedicating resources to its regular upkeep and monitoring. Therefore, optimizing the code, implementing caching methods, and making use of cloud infrastructure are all examples of scalability measures that should be considered early on.

Last but not least, future work needs to be done to leverage the AI chat box to enhance the user experience when shopping online. By supporting the answer to a query, assist in guiding the purchasing method. Additionally, it can support automated inventory management and individual product suggestions.

V. Peer Evaluation

Roles and Responsibilities			
Trello Board: https://trello.com/invite/b/YPDCAFvR/ATTI3c6171182322837ae156161a1d76826bB17C326E/full stack-web-project **Note:** Everyone involved in manually testing website features and functionalities.			
Team Member's Name	**Role**	**Tasks responsibility**	**Project contribution (%)**
Do Le Long An (s3963207)	Project Manager, Frontend and Backend Developer	- Manage and keep track of Product Backlog on Trello - Planning Sprint Meeting and Tasks Allocation - Execute tasks **(Customer, Shipper and Vendor)** in both client – side and server – side of the website. - Supervise GitHub repository and branches - Deploy Project onto live website	20

Bui Tuan Anh (s3970375)	Technical Leader, Frontend and Backend Developer	- Execute tasks **(Customer, Shipper and Vendor)** in both client – side and server – side of the website. - Technical decision including tools and resources - Keep track of Sprint Backlog and Website Functionalities	20
Lao Vinh Khang (s3891925)	Technical Report Leader, Backend Developer	- Execute tasks for **Vendor** in server – side - Manage and keep track of paperwork progress - Writing report	20
Truong Hong Van (s3957034)	Technical Writer, Designer and Frontend Developer	- Figma UI Designer - Execute tasks in client – side of the website **(buttons, dashboards, carts, order history, login and signup forms, navigation bars, homepage)** - Website Content Writer - Writing report	20
Pham Le Quynh Anh (s3927427)	Technical Writer, Frontend Developer	- Supervise Meeting Minute - Execute tasks in client – side of the website **(login and signup forms)** - Making presentation slides and deciding video flows - Manage and keep track of the presentation video upload.	20

VI. Reference

VNExpress International (12 January 2023) "Innovation helps increase e-commerce firms' competitiveness: Lazada exec', *VNExpress International*, accessed 19 May 2023. https://e.vnexpress.net/news/business/innovation-helps-increase-e-commerce-firms-competitiveness-lazada-exec-4559164.html

OpenAI's ChatGPT (2023), personal communication, accessed 18 May 2023. https://chat.openai.com/

Npm(2023), Npm dotenv, npm website (2023), accessed 12 May 2023. https://www.npmjs.com/package/dotenv

Npm(2023), Npm ejs-layout, npm website (2023), accessed 13 May 2023. https://www.npmjs.com/package/ejs-layout

Npm(2023), Npm body-parser, npm website (2023), accessed 14 May 2023. https://www.npmjs.com/package/body-parser

Npm(2023), Npm express-async-handler, npm website (2023), accessed 14 May 2023. https://www.npmjs.com/package/express-async-handler

Npm(2023), Npm morgan, npm website (2023), accessed 15 May 2023. https://www.npmjs.com/package/morgan

Npm(2023), Npm multer, npm website (2023), accessed 15 May 2023. https://www.npmjs.com/package/multer

Npm(2023), Npm multiparty, npm website (2023), accessed 16 May 2023. https://www.npmjs.com/package/multiparty

Npm(2023), Npm node-localstorage, npm website (2023), accessed 16 May 2023. https://www.npmjs.com/package/node-localstorage

Npm(2023), Npm node.bcrypt.js, npm website (2023), accessed 16 May 2023. https://www.npmjs.com/package/bcrypt

Mdn web docs (2023), Web technology for developers, Mdn web docs website, accessed 17 May 2023. https://developer.mozilla.org/en-US/docs/Web

VII. Appendix

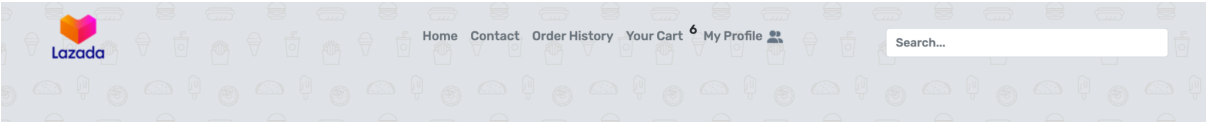


Figure UI: Header requirement

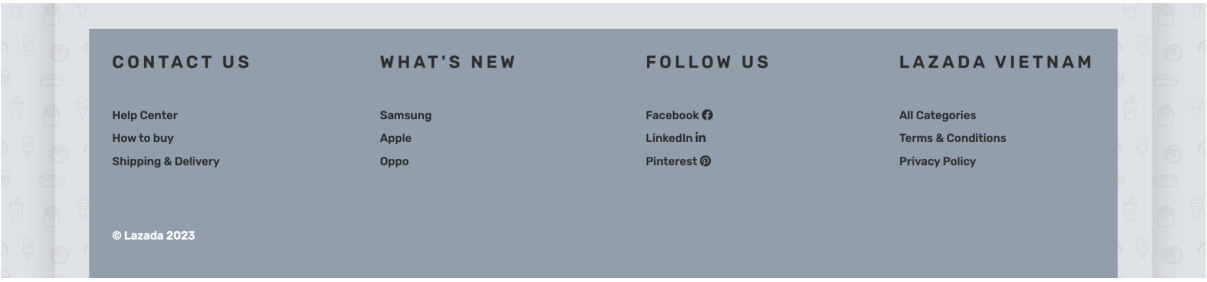


Figure UI: Footer requirement

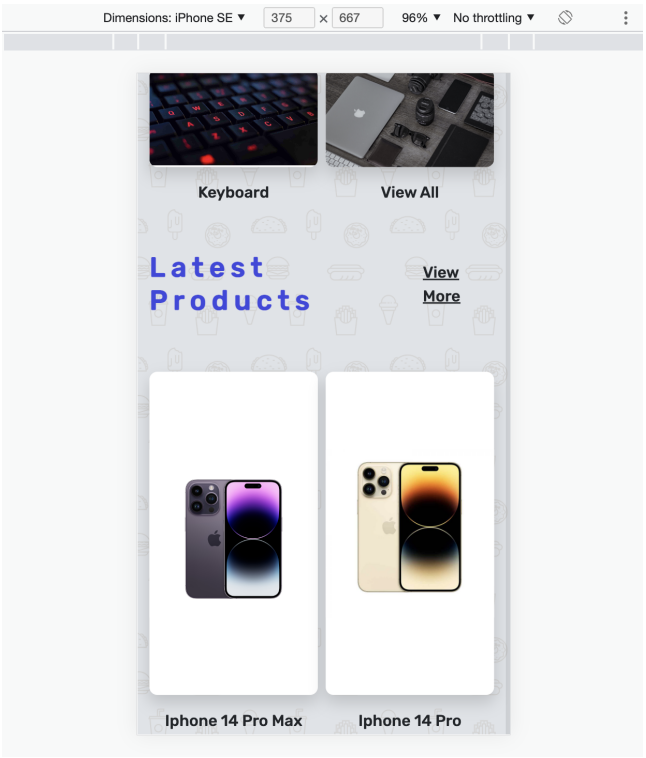


Figure UI: Phone responsive

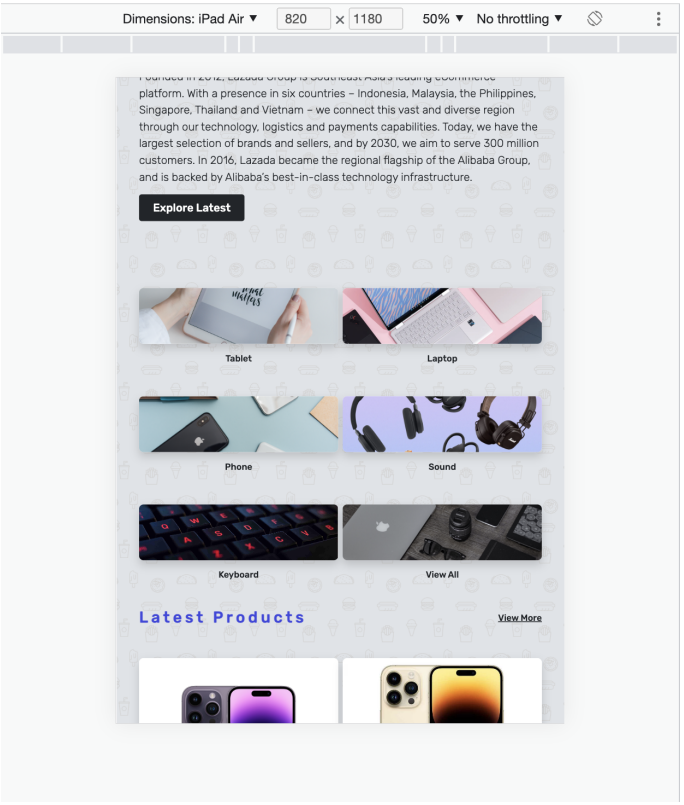


Figure UI: Tablet responsive

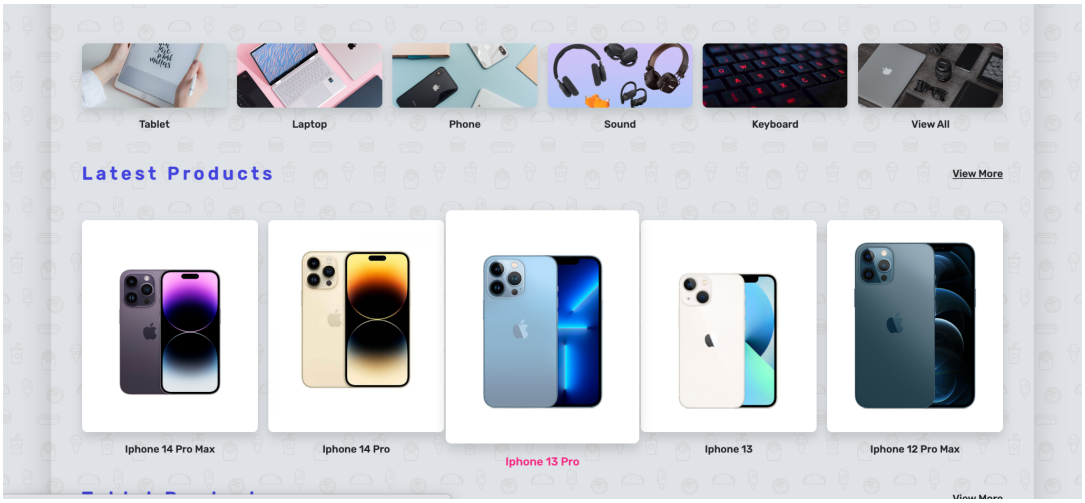


Figure UI: Desktop responsive

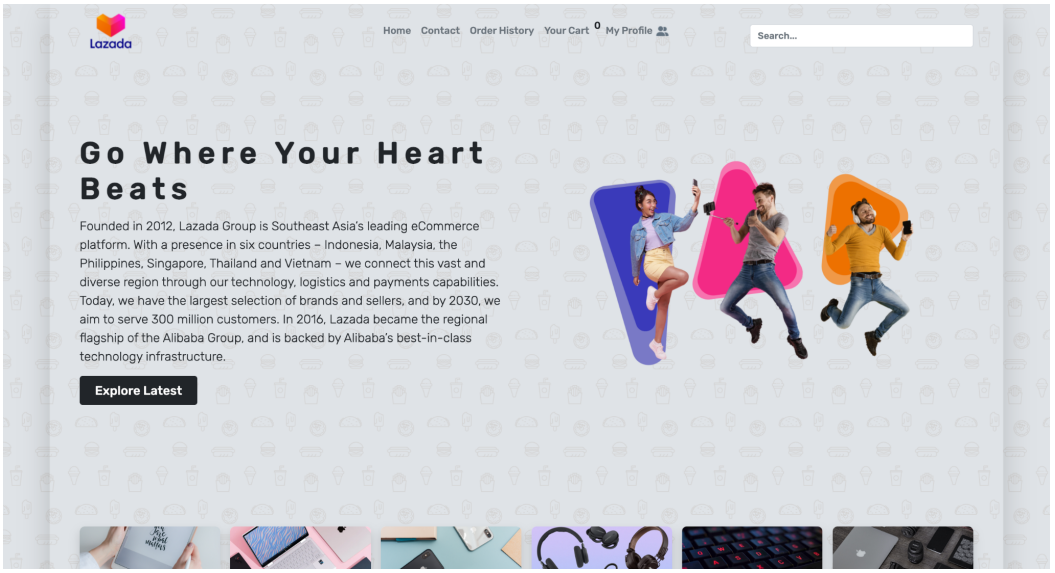


Figure 1: Customer's homepage

Figure 2: Validation

Figure 2a: Validation for product-details.css

Figure 2b: Validation for 404.css

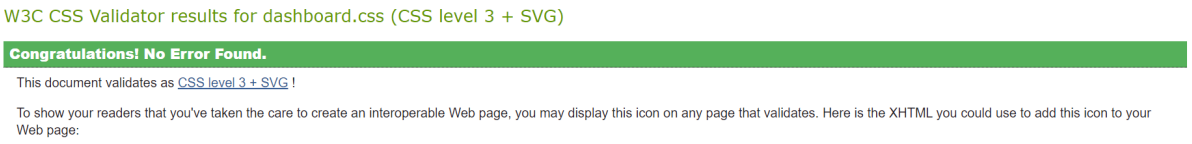


Figure 2c: Validation for dashboard.css

Figure 2d: Validation for profile.css

Sorry! We found the following errors (8)

URI : https://group9ecommerceweb-dolelongan.b4a.run/css/styles.css

12	:root	Parse Error `--nav-bar-bottom-border: 5px solid #637381;`
13	:root	Parse Error `--montserrat: 'Montserrat', sans-serif;`
14	:root	Parse Error `--poppins: 'Poppins', sans-serif;`
15	:root	Parse Error `--open-sans: 'Open Sans', sans-serif;`
16	:root	Parse Error `--bs-font-sans-serif: 'Rubik', sans-serif;`
17	:root	Parse Error `}`
28	*, html	`0` is not a `scrollbar-width` value : `0`

URI : https://group9ecommerceweb-dolelongan.b4a.run/css/login-signup.css

180	button.submit	Value Error : width Too many values or values are not recognized : `32px auto`

Figure 2e: Validation for styles.css and login-signup.css

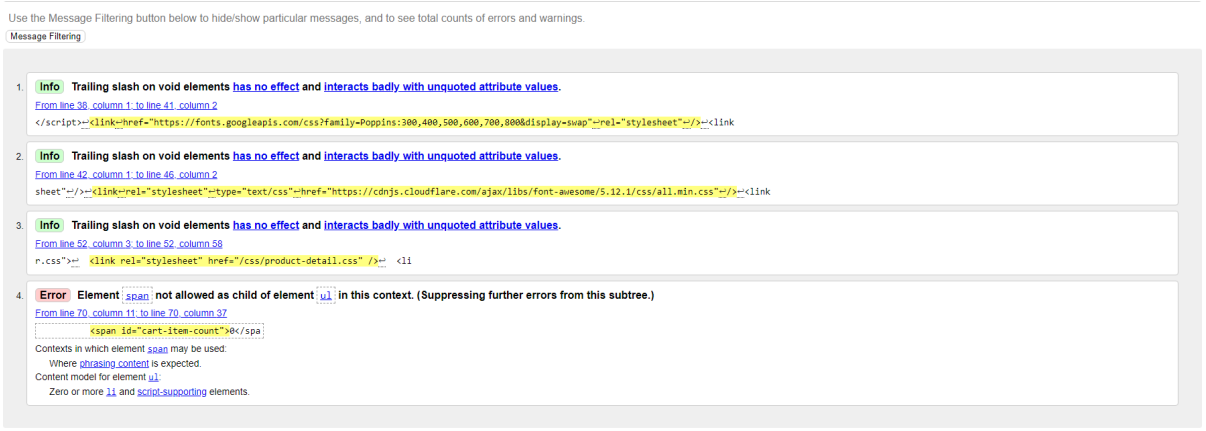


Figure 2g: HTML validation for Order History

Figure 2g: HTML Validation for Vendor dashboard

Figure 2h: HTML validation for Shipper Dashboard

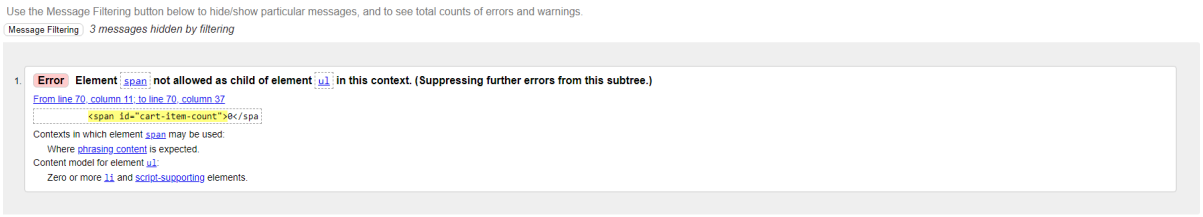


Figure 2i: HTML validation for Homepage

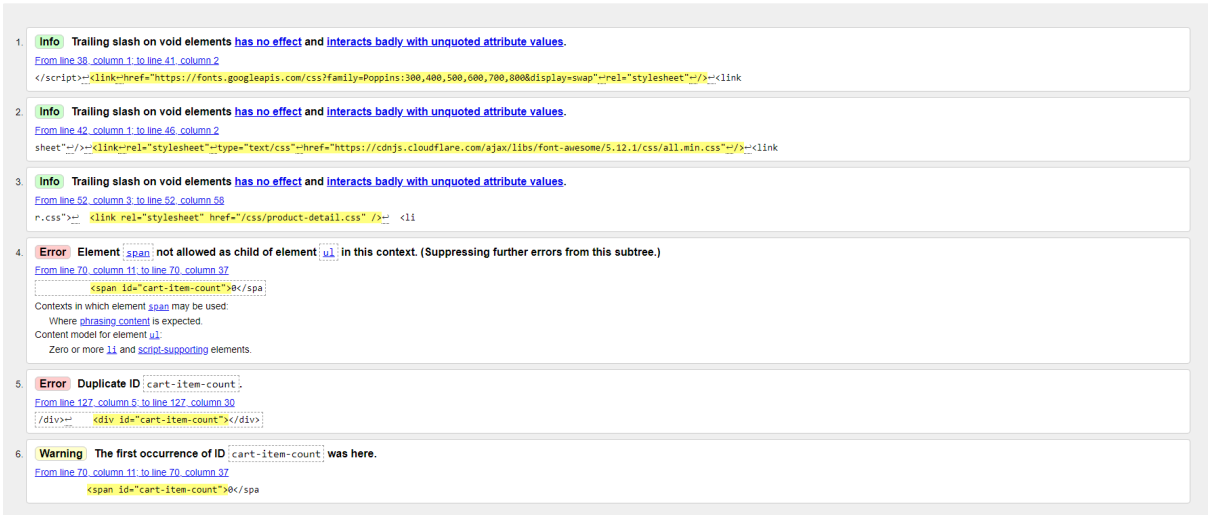


Figure 2j: HTML validation for Your cart page

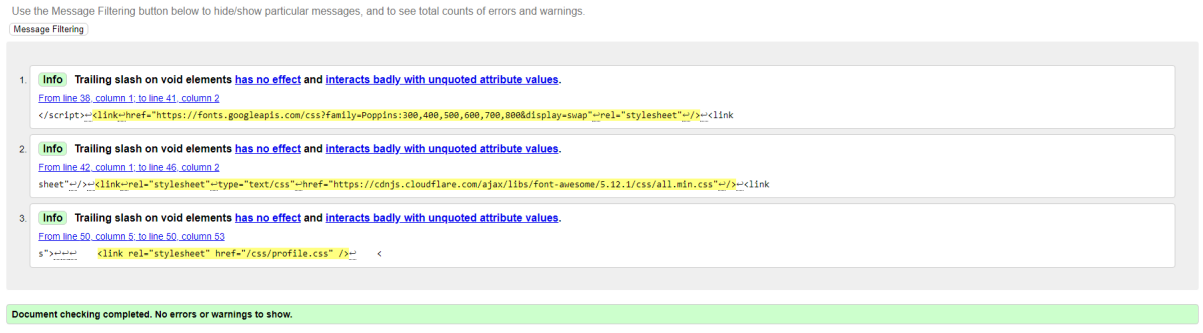


Figure 2k: HTML validation for My Account page

Figure 2l: HTML validation for Login page

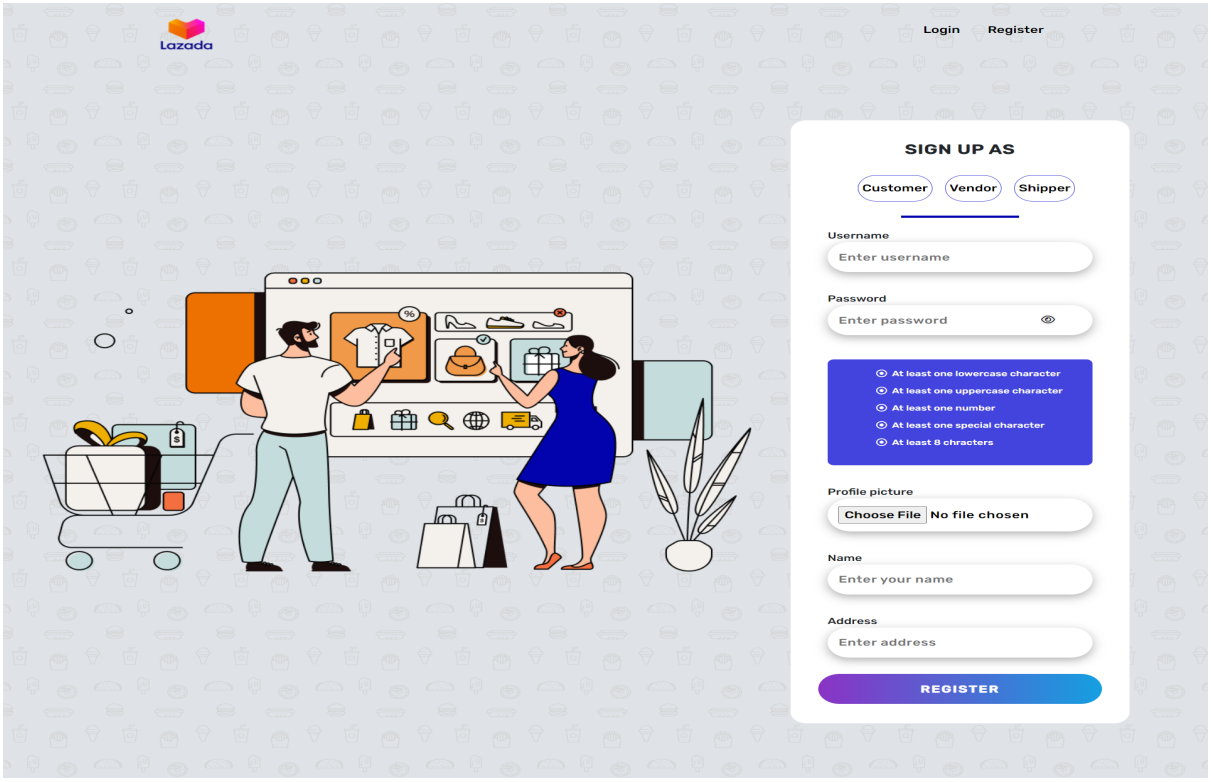


Figure 3: Register page for 3 roles

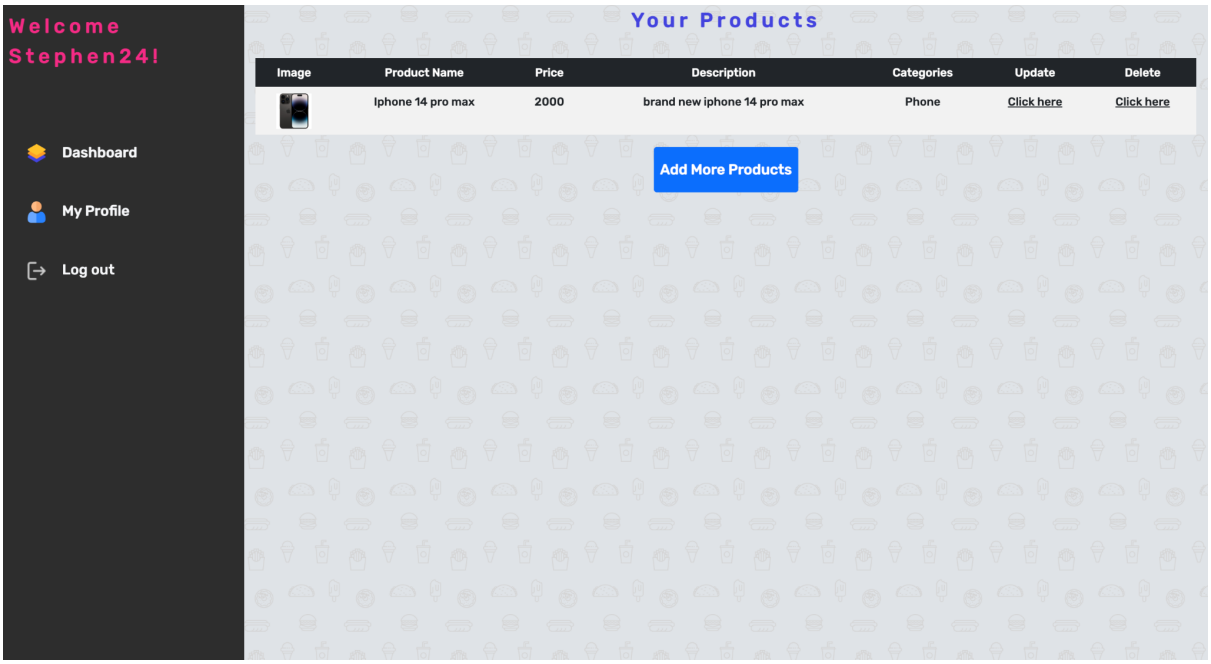


Figure 4: Vendor dashboard

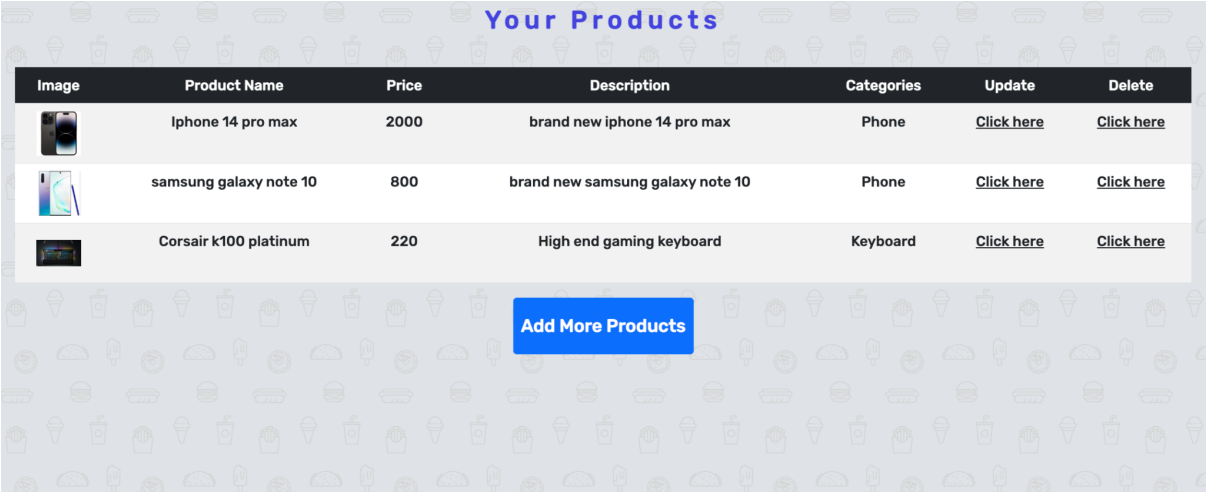


Figure 5: View all product

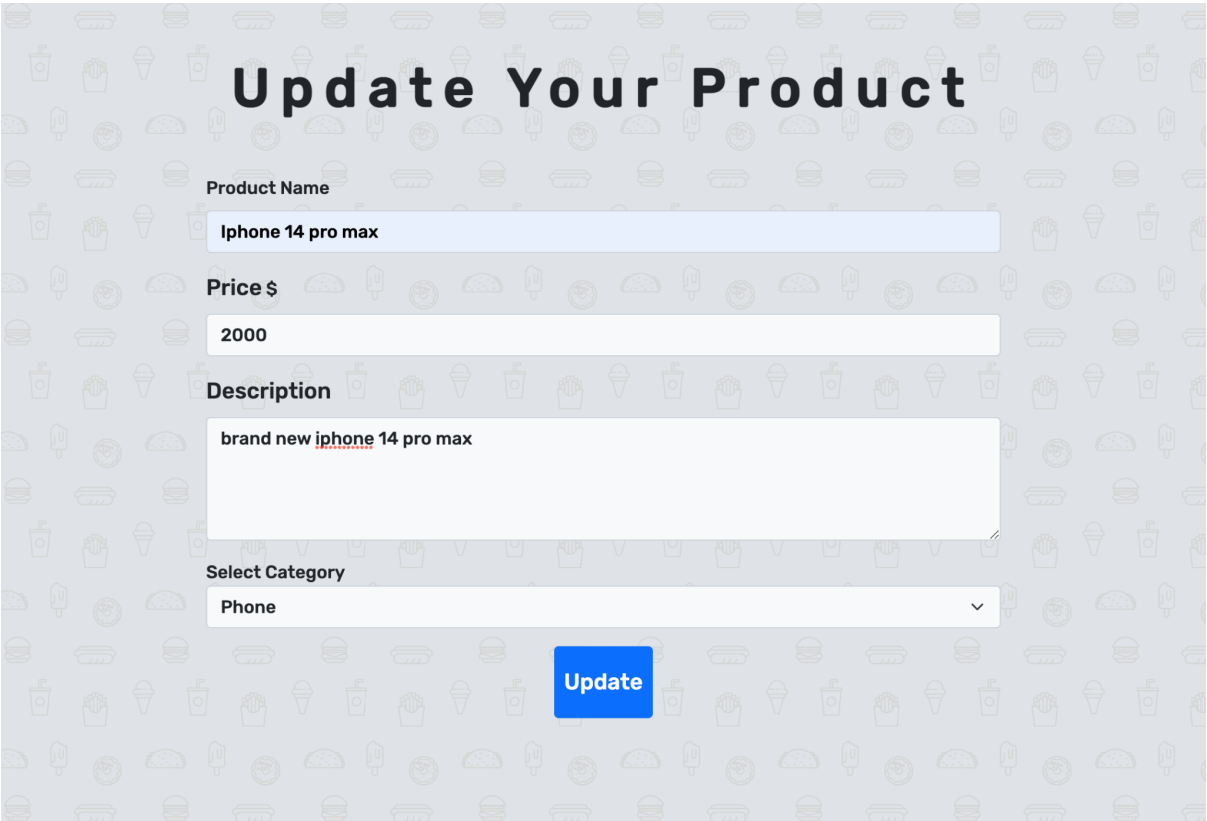


Figure 6: Update products

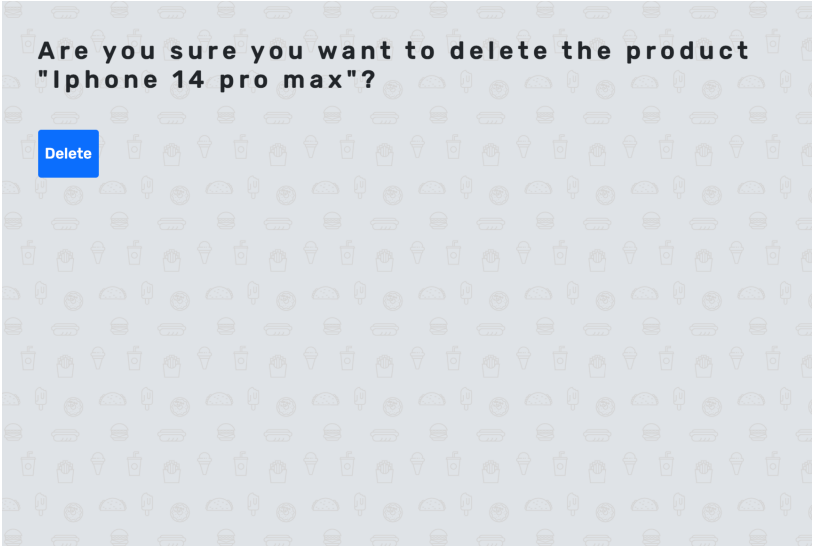


Figure 7: Delete product

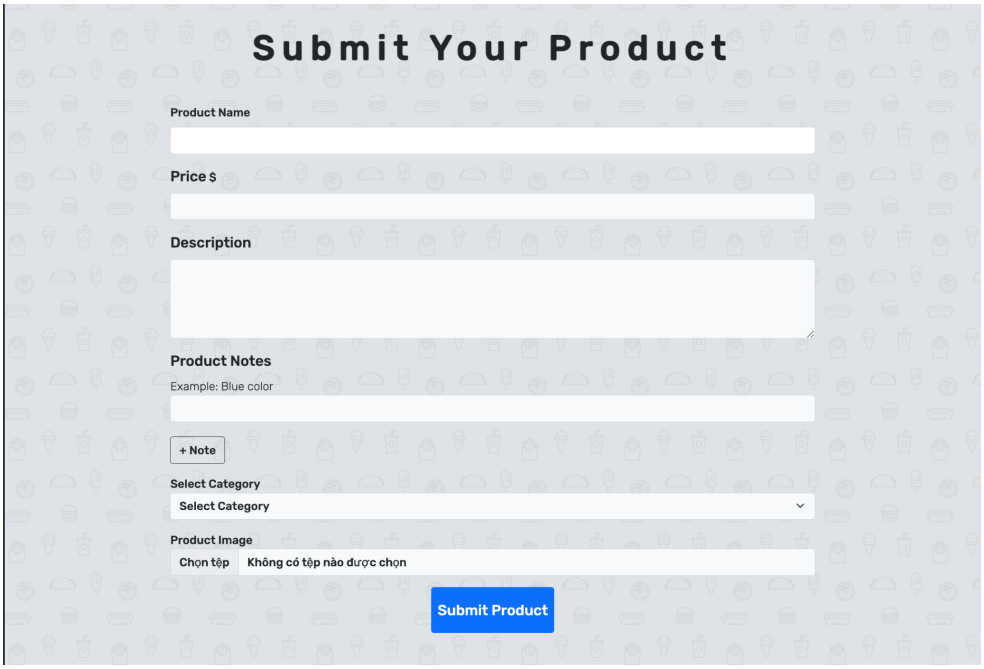


Figure 8: Create new products

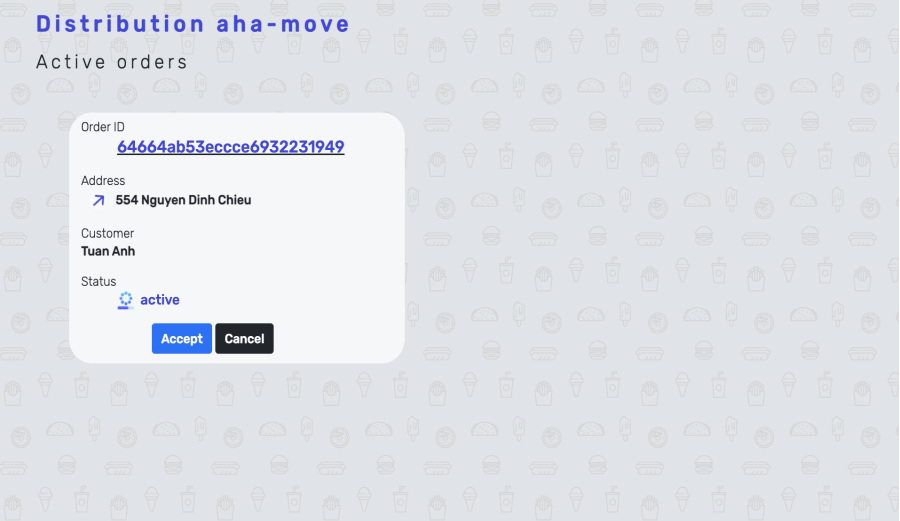


Figure 9 + 10: Shipper dashboard and OrderID

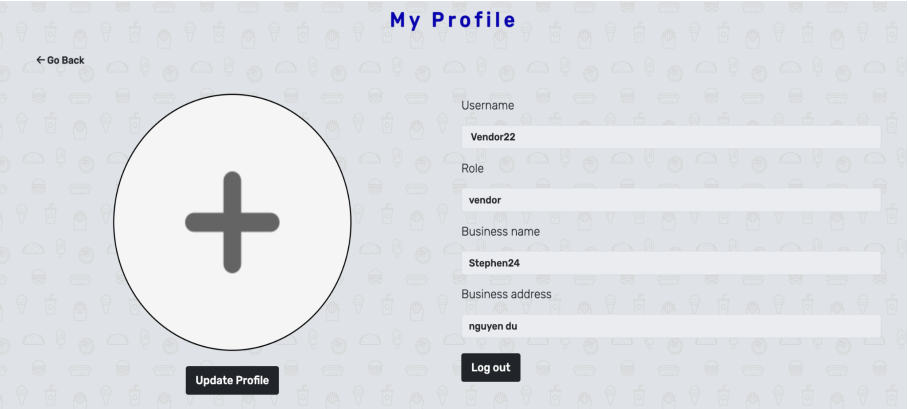


Figure 11 + 12 : Go back button and logout

Figure 13: 404 error page

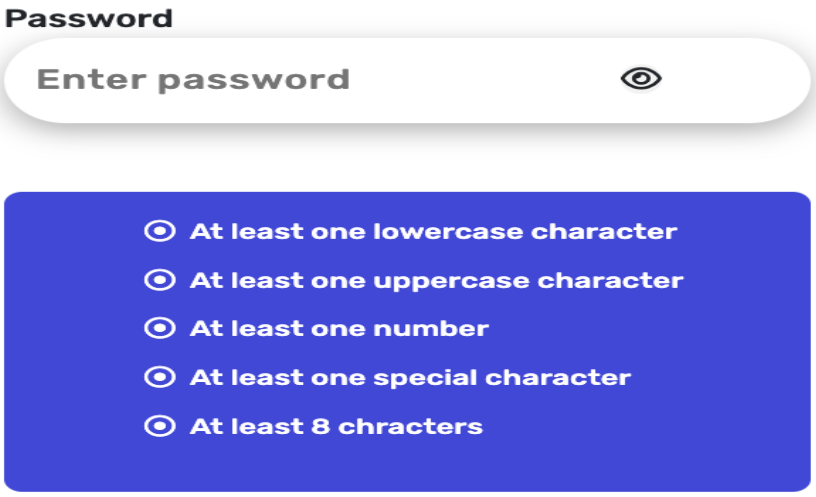


Figure14: Display check password

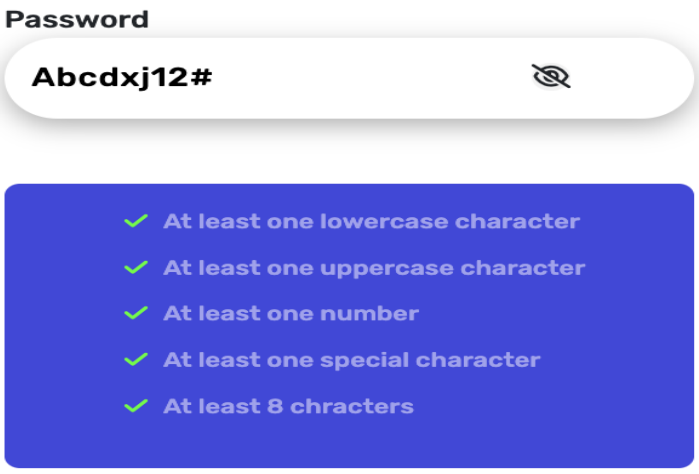


Figure 15: Toggle eyes showing password

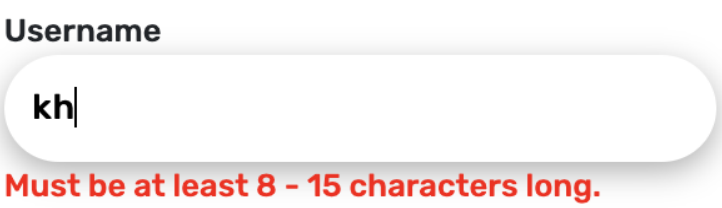


Figure 16: Username validation in register page

Figure 17: Password validation in register page

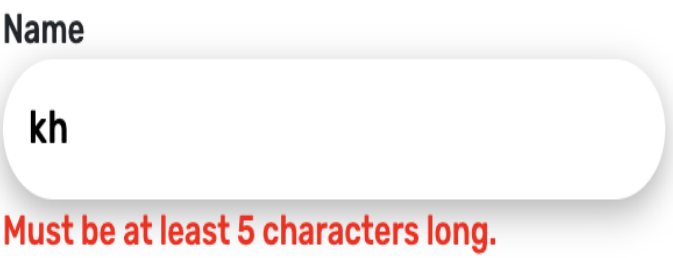
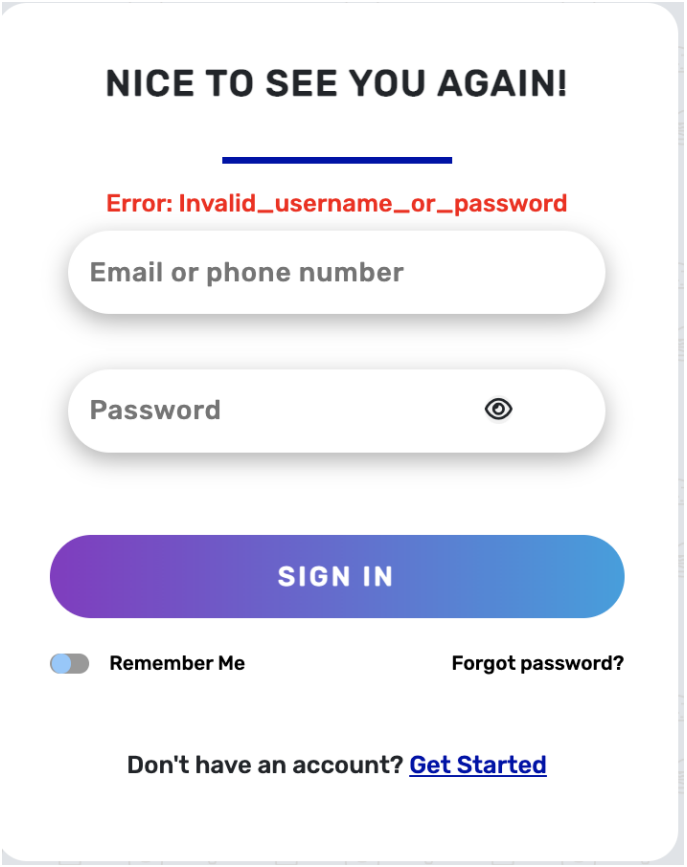


Figure 18: name validation in register page



group9ecommerceweb-dolelongan.b4a.run/?error=Invalid_username_or_password

Figure 19: Error query message

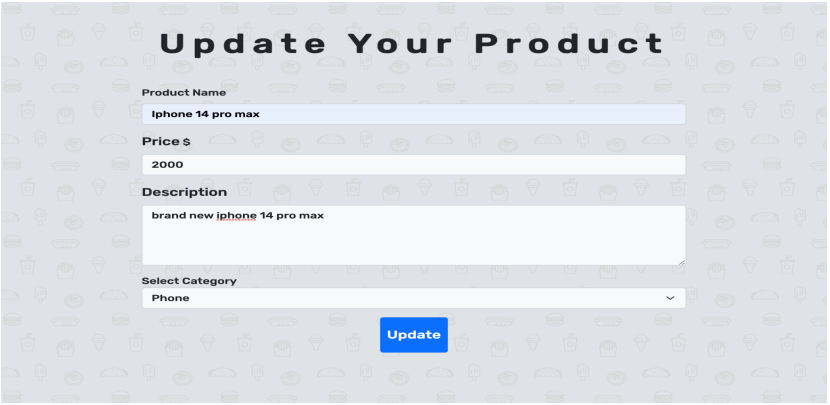


Figure 20: Update function for vendor

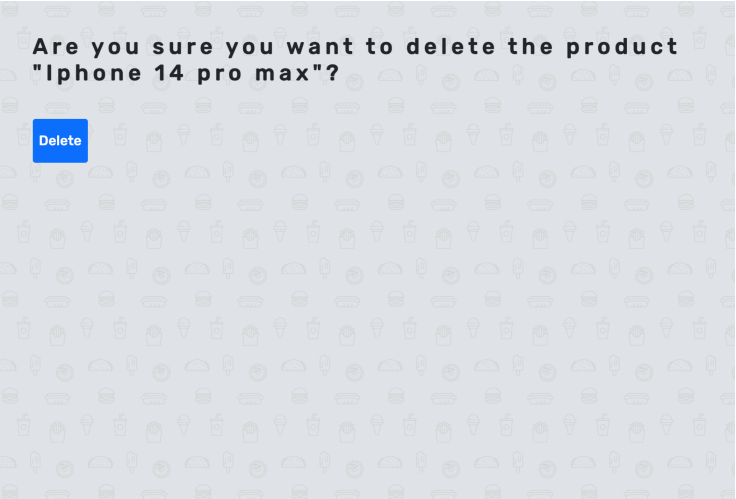


Figure 21: Delete function for vendor

Image	Product Name
	Iphone 14 pro max
	samsung galaxy note 10
	Corsair k100 platinum

Figure 22: View image

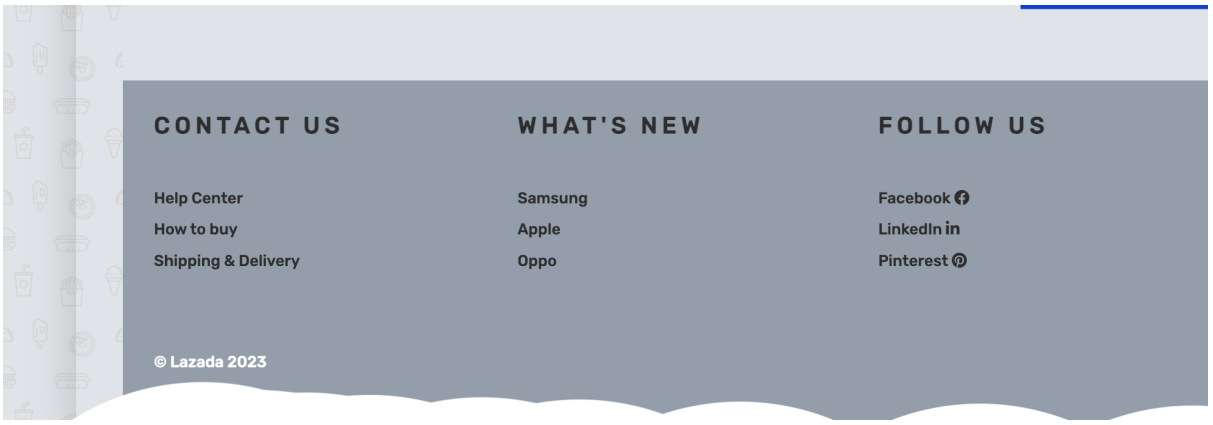


Figure 23: Footer bubble animation

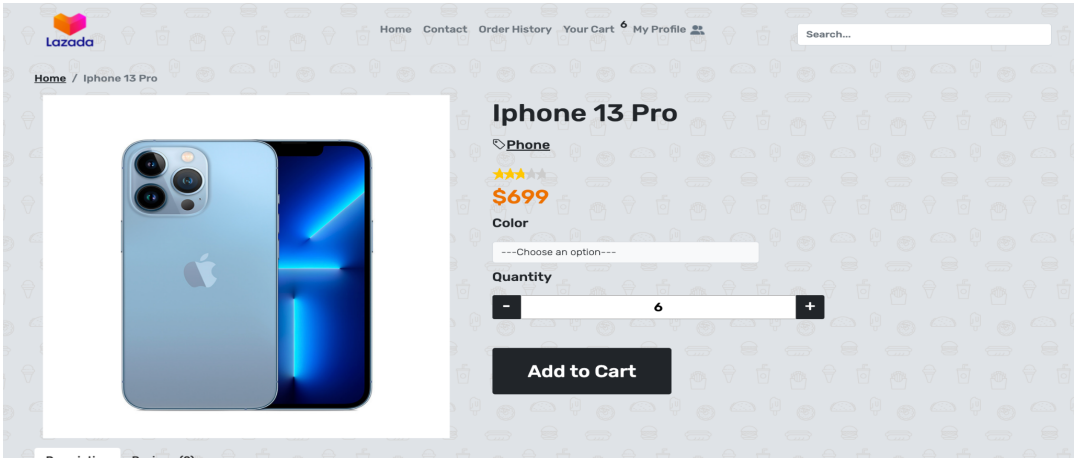


Figure 24: View number of cart

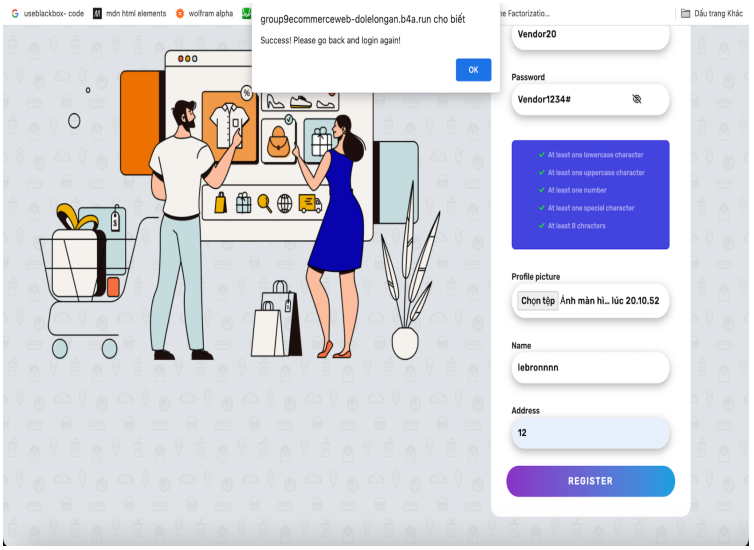


Figure 25: Alert box when register successfully