



## Topic 10: E-Commerce website for automotive parts

ID	Name	Phone number	Contribution percentage
ITITIU19228	<a href="#">Trần Nguyễn Thương Trường</a> (leader)	0917085937	100%
ITITIU19173	<a href="#">Trần Hữu Nhật</a>	0342958368	100%
ITITIU19225	<a href="#">Nguyễn Trung Trực</a>	0911680270	100%
ITITIU19181	<a href="#">Lê Quốc Phong</a>	072243023	100%
ITITIU19201	<a href="#">Trần Thanh Sơn</a>	0916407532	100%

### TABLE OF CONTENTS

#### CHAPTER 1: INTRODUCTION

1.1 Introduction about the topic and requirements .....	2
1.2 Planning and duties of members .....	2
1.3 Action workflows .....	2
1.4 Reasons of choose this topic .....	2
1.5 Planning research .....	2, 3

#### CHAPTER 2: ANALYSIS

2.1 Description of project .....	3, 4, 5
2.2 Entity Relationship Diagram (ERD) .....	6
2.3 Relational Database Model .....	
2.4 User requirements .....	

#### CHAPTER 3: IMPLEMENTATION ON POSTGRESQL

3.1 Create the database .....	7
3.2 Input data and testing .....	7, 8

#### CHAPTER 4: CONNECT

4.1 Create interface .....	8
4.2 Connect with API .....	8, 9, 10

#### CHAPTER 5: PRODUCTION

5.1 Overview .....	11-17
5.2 Analysis and unit test .....	17, 18
5.3 Test for use cases .....	18

#### CHAPTER6: CONCLUSION & REFERENCES

## **I. INTRODUCTION**

### **1.1. Requirement of topic**

- Under this project, you can develop a standard e-commerce website that displays products to be sold. Users must create account be able to select the products they want to buy and add them to cart. Then able to make checkout the their orders and payments via a secure payment authenticate gateway.

### **1.2. Requirement of proposal & report**

- Student informations.
- Topic informations.
- Detailed description of the topic.
- Implementation plan.
- Outline of midterm project.
- ERD digram of database system.

### **1.3. Action workflows**

- Generate ideas → plan implementation → build a roadmap → divide specific tasks for each member → development → production.
- Monitor progress, evaluate and check after each phase.
- Building user interface and experience for client (UI/UX).
- Handling logics and interactions action of data in server.
- Initialize the database (CRUD diagram) & connect to server.
- Data is communicated between client and server through API.
- Backend-side is where data can be written and readed from the database.
- Presentations slide & report: All member.
- Frontend - Design team: Trường, Nhật, Phong.
- Backend - Database team: Trường, Trức, Sơn.

### **1.4. Why we choose the topic**

- Nowadays, the e-commerce market is becoming a part of everyday life. With just a few simple steps, through our personal phone or laptop, we can easily search and buy our favorite products.
- The scope and functionality of the website is quite extensive, requiring a combination of programming knowledge to create it ((OOP, Database, API, Algorithm, UI/UX, Responsive web...). Therefore, we can both practice and improve our programming knowledge in the process of website development.
- Famous and popular e-commerce websites such as: "Amazon, Shopee, Tiki, Lazada, Alibaba ...". They are all designed with beautiful interfaces, great experiences to attract consumers.

### **1.5. Planning**

- We think about all the business rules and scenarios that will lead to the project's outcome.
- Almost all businesses now use online services to modernize.
- System scope of use of the website is not limited to age and gender.

- ERD will be in the database. Information fields will be added when the user selects the product.
- Elegant design will please the viewer and regularly updated on the website to increase the number of users.
- But the first and important goal here is to design how the idea, roadmap and workflow go through the system instead of focusing on the front-end and back-end.
- Back to the system, business processes and website operation will be applied as follows:
  - When users register for an account and are entering information on the page, they can shop online by searching for their favorite products and adding to their shopping cart.
  - The products will have illustrations, product names, prices. Products added to the cart will be added to the waiting list. And the user has the right to confirm the purchase or remove it from their list. After purchasing and choosing to buy the product, the user will complete the information and pay at the "checkout-page".
  - A list of discounted products and remaining quantities are also displayed according to the online time frame, along with the top selling products and recommended products are also displayed.
  - Users can easily edit personal information and public delivery address at "account-page".
  - With sellers, they will see the total of all pending and canceled orders, the total of all products in their store. Sellers will add products to their store at the "add-product page". In addition, all information or addresses about the seller is also public at the "shop-profile page".
  - Seller and user can chat directly with each other.
  - Admin has the highest authority in the system, they can see the total number of orders, total number of users and new registrations, total products, seller list, user list, can add-remove-edit users, add-remove-edit seller. Every detail is represented as a visual chart.
  - Admin can chat directly with the seller.
  - In addition, the website system is supplemented with multi-language mode, dark / light mode.

## II. ANALYSIS

### 2.1 Description of topic

- Our platform name is e-shopee. In this project, we refered to [shopee.vn](https://shopee.vn) website as a main focus studying. However, interface and relationship database is minimization and customizing to be appropriate.
- Feather:
  - Multi authenticate account.
  - Verify users through mail code.
  - Online chat messages with seller and admin.
  - Discounting on real time.
  - Dark/Light mode inside.

- Multi languages (English, Vietnamese, Japanese, Spanish, Chinese, Indian, Russia...)
- Languages & tools:
  - Frontend: HTML, CSS, SCSS, JavaScript, TypeScript, Vue...
  - Backend: Laravel, PHP, TypeScript, Blade..
  - Database managements: PostgreSQL, SQL...
  - Support tools: XAMPP, phpMyAdmin, postgreDB, Docker...
  - Docs tools: Postman, Swagger, Draw.io, Github...
- The website system have **3-role** view: Buyer - Seller – Admin in Frontend-side equal to MVC parttern in Backend-side.
- **Buyer role:**
  - ✓ Register/Login-page:
    - Start with new account for buyer.
    - Register with username or email and verification email code.
    - Reset password through email if forgot.
  - ✓ Home-page:
    - Menu of all kinds of products.
    - Search product.
    - Add product to cart.
    - Flash sales in discounts list.
    - List of hint products today.
    - List of discounted products & romotion time remaining.
    - List of top best-selling products.
  - ✓ Product-details-page
    - Show a complete display of product details that the user wants to see
    - Consist (product image, name, sizes, category of, color, quantities, star-votes come-from, descriptions, comments of other users, representation shop preview...).
    - Summary preview about owner shop of this products.
  - ✓ Cart-page:
    - View, Update, Select, Delete all added products in the cart.
    - Display prices, preview image, description, name, quantity of products.
    - Total bill before checkout.
  - ✓ Checkout-page:
    - Setup all information of account profile (name, email, phone number, payment method, credit card details, address, postal code...) before checkout if they're empty.
    - Choose a delivery service and make a note on delivery.
  - ✓ Account-page:
    - Setup, update, delete all information of account profile (name, email, phone number, payment method, credit card details, address, postal code...)
    - Authenticate user.
  - ✓ About-page:
    - Informations & description of project and group.

- ✓ Contact-page:
  - Address details, hotline and email to main company.
  - Send contact messages (active in future).
- ✓ Sitemap-page:
  - Google-map address.
- ✓ Privacy legal & FAQ.
- ✓ Services & contact.
- **Seller role**
  - ✓ Dashboard-page:
    - Visualize view all informations of seller controller.
  - ✓ Orders-page:
    - Display the list of all orders (sort by time).
  - ✓ Product-page:
    - Filling shop information (if empty) before add new categories.
    - Create and update products information (sizes, prices, status, quantities...).
    - List all products on sale and available in the shop.
    - Filter and find products by names, categories, quantities...
  - ✓ Notifications-page:
    - Display all notifications from customer and system.
  - ✓ Shop-preview:
    - Display shop profile (avatar, cover, name, address, total products, review...)
    - Edit shop profile details.
  - ✓ Shop-profile-preview:
    - Display shop profile when public.
  - ✓ Advertising-page:
    - Manage advertising.
  - ✓ Chat-page (active in future):
    - Chat with admin.
    - Chat with customer.
- **Admin role**
  - ✓ Login-page:
    - Login by admin email/password and verify through email.
    - Reset new password through email if forgot.
  - ✓ Dashboard-page:
    - Profile information of admin.
    - Manage customer list and seller list.
    - Total products, total orders, total sales, total new users.
    - Add, Update, Delete any users or sellers.
    - Searching products by categories.
    - View and join advertisements (active in future).
    - Chat with seller (active in future).

## 2.2 Entities relationship diagram (ERD)

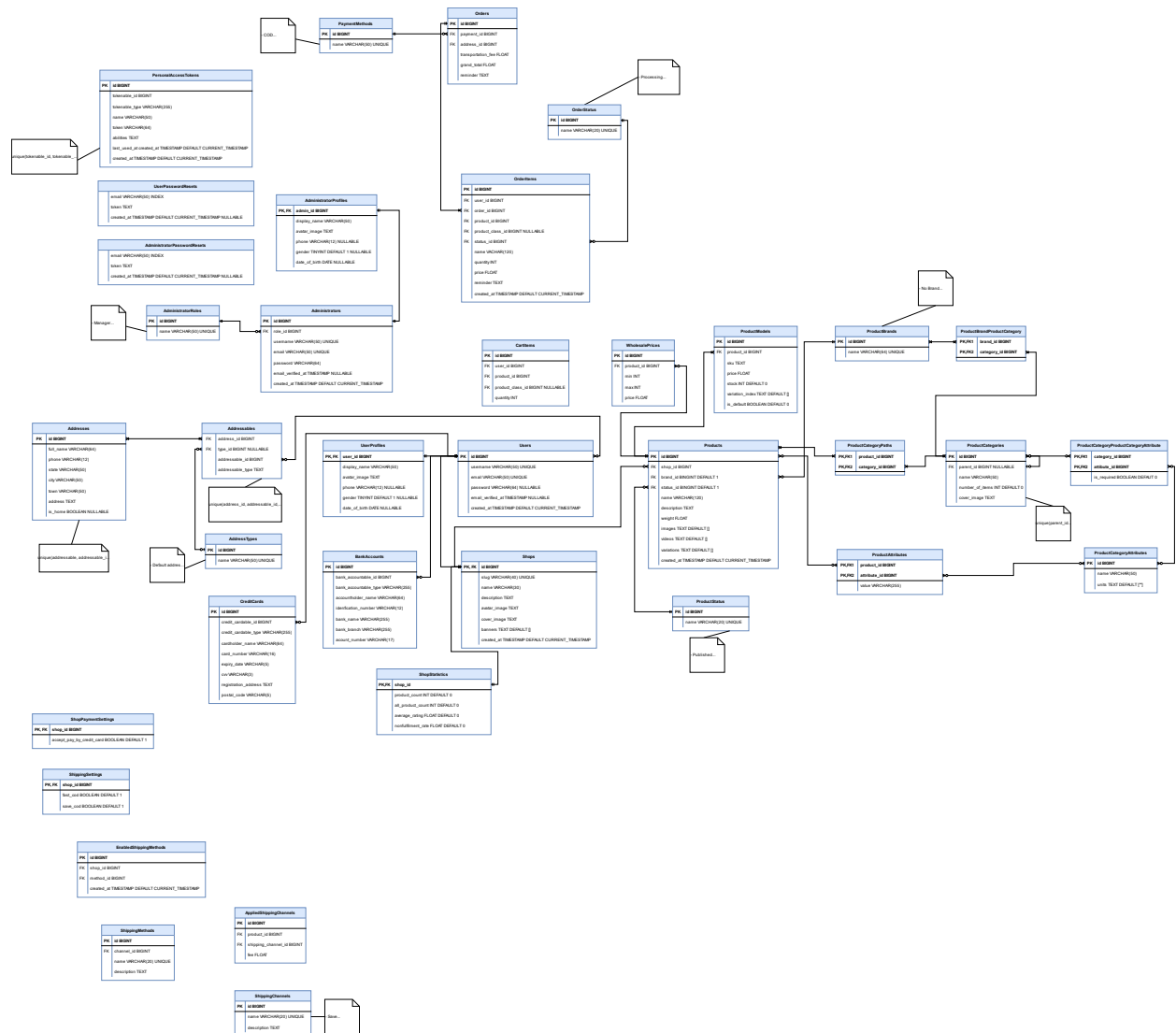


Figure 2.2 Database entities relationship diagram

### III. IMPLEMENTATION ON POSTGRESQL

#### 3.1 Create the database

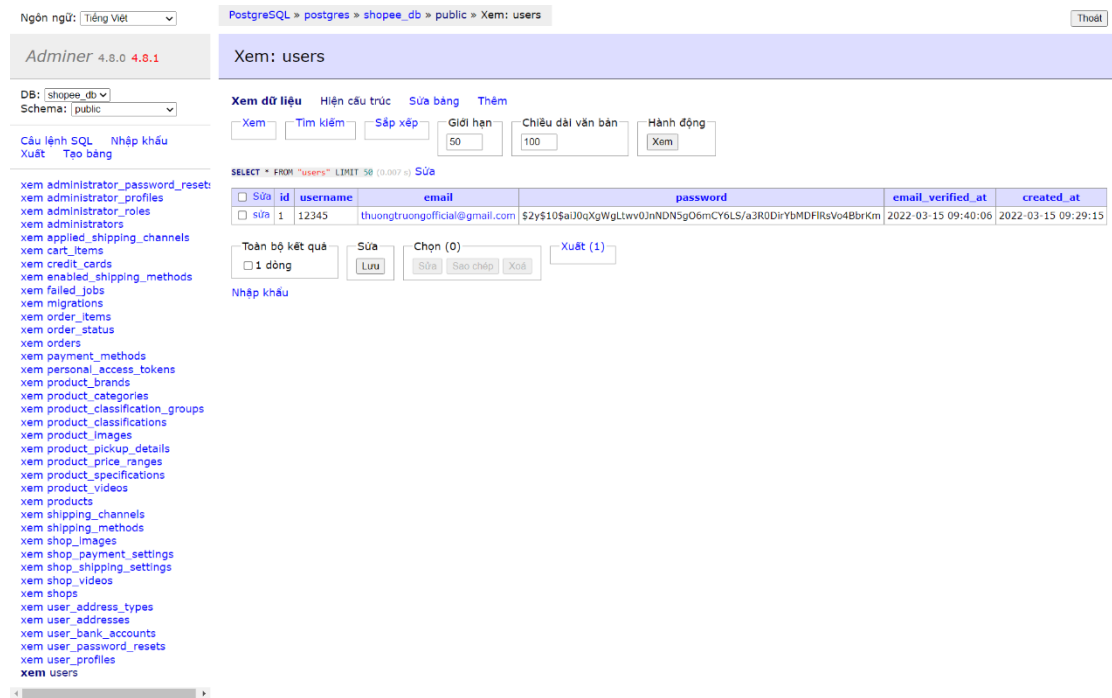


Figure 3.1 Database query in postgresDB playground

#### 3.2 Input data & testing

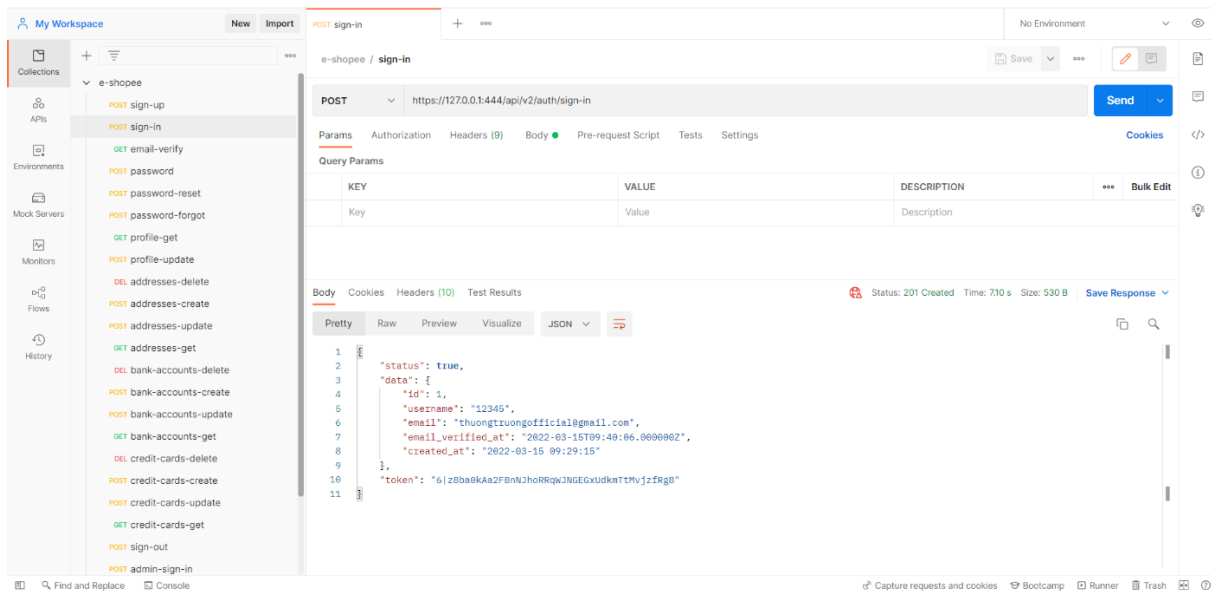


Figure 3.2a Testing API on Postman

**E-Commerce API** OpenAPI 3.0

As this file is API documentation of my simple E-Commerce API, it has not been deployed on the internet, so once this repo and run it before accessing API endpoints

**Server**  
 https://api.swagger.io/v2/swagger.json

Computed URL: https://227.8.8.1:80/api/v2

**Server variables**  
 host: 127.0.0.1  
 port: 80

**resource** Resource management

- POST /resources/images Upload image

**auth** Authentication

- POST /auth/sign-up Sign up an new account

**Parameters**

Name	Description
username	Username of user
email	Email of user
password	Password of user
password_confirmation	Confirm the password

**Responses**

Code	Description	Links
201	Success	No links
400	Invalid input	No links

**password** Password management

- POST /password/reset Reset user's password
- POST /password/forget Request to send reset user password email
- POST /password/reset Request to reset user's password
- POST /password/verify Verify user's password
- POST /password/forget Request to send reset user password email
- POST /password/verify Verify user's password

**account** Account management

- GET /account/profile Get user's profile
- POST /account/profile Update user's profile
- GET /account/addresses Get user's addresses
- POST /account/addresses Create user's address
- DELETE /account/addresses/{id} Delete user's address
- GET /account/bank-accounts Get user's bank account
- POST /account/bank-accounts Create user's bank account
- DELETE /account/bank-accounts/{id} Delete user's bank account
- GET /account/credit-cards Get user's credit cards
- POST /account/credit-cards/{id} Create user's credit card
- DELETE /account/credit-cards/{id} Delete user's credit card

**shop** Shop management

- GET /shops/{id\_or\_slug} Get public information of the shop
- GET /shops/{id\_or\_slug}/products Get product of the shop
- GET /shops Get shop of current user
- POST /shops Create a shop for current user
- DELETE /shops Delete shop of current user
- GET /shops/products Get product of current user

**product** Product management

- GET /products/{id} Get product by ID
- POST /products/{id} Update product
- DELETE /products/{id} Delete product
- POST /products/{id}/inventory Increase/decrease product
- POST /products Create product
- GET /products/categories/{id} Get category by ID
- GET /products/categories/{id}/children Get children of category
- POST /products/categories/{id}/attributes Get all attributes of category
- POST /products/categories/{id}/attributes Create, update, and delete attributes
- GET /products/categories/attributes Get all attributes of categories
- POST /products/categories/attributes Create, update, and delete attributes
- GET /products/categories/attributes/{id}/input Get attributes

Figure 3.2b Testing API on Swagger

## IV. CONNECT

### 4.1 Create interface (this is sample, see more in source code)

```
<route lang="yaml">
  meta:
    layout: buyer/IBRegister
  </route>

<script setup>
  import { useRouter } from "vue-router"
  import { useLoading } from "~/stores/loading"
  import { toast } from "~/stores/toast"
  import { handleError } from "~/helpers/error"
  import AuthRequest from "~/services/auth-request"

  useHead({
    title: "e-shopper | buyer register",
  })

  const { t } = useI18n()

  const router = useRouter()
  const loading = useLoading()
  const toast = toast()

  const payload = reactive({
    username: "",
    email: "",
    password: "",
    password_confirmation: "",
  })

  const handleSubmit = async() => {
    //preventDefault()
    useLoading.isLoading = true
    await AuthRequest.registerUser(payload)
    //then(res) => {
      router.push({ path: "/buyer/login" })
      useLoading.isLoading = false
      useToast.updateToast("success", "Hi, {payload.username}, Check verification email sent in your mailbox", true)
    }
    //catch(error) => {
      return handleError(error)
    }
  }
</script>
```

```
<template>
  <div class="buyer-register-container flex justify-center pt-20">
    <div class="register-form border-1 border-solid border-white rounded-2xl flex w-400px h-400px shadow-4xl bg-[#f0f0f0]>
      <div class="bg-white w-1/2 p-5 rounded-r-2xl flex flex-direction="column">
        <div class="text-center text-white font-weight-bold flex justify-center items-center">
          <h3>Register</h3>
        </div>
        <div class="flex flex-direction="column">
          <div class="form-group">
            <input type="text" value="" placeholder="Username" required/>
          </div>
          <div class="form-group">
            <input type="text" value="" placeholder="Email address" required/>
          </div>
          <div class="form-group">
            <input type="password" value="" placeholder="Password" autocomplete="true" required/>
          </div>
          <div class="form-group">
            <input type="password" value="" placeholder="Password confirmation" autocomplete="true" required/>
          </div>
          <div class="text-center">
            <button type="button" class="btn btn-primary" href="#">Register</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

### 4.2 Connect API by Axios



```
import queryString from 'query-string'
import axios from 'axios'

const AxiosInstance = axios.create({
  baseURL: `${import.meta.env.VITE_BASE_DOMAIN}/api/v2`,
  headers: {
    'Content-Type': 'application/json',
    'X-Requested-With': 'XMLHttpRequest',
  },
  // 'Authorization': localStorage.getItem('token'),
}, {
  paramsSerializer: params => queryString.stringify(params),
  responseEncoding: 'utf8',
})
AxiosInstance.interceptors.request.use(
  async(request) => {
    const token = localStorage.getItem('token')
    request.headers.Authorization = token ? `Bearer ${token}` : ''
    return request
  },
  (error) => {
    return Promise.reject(error)
  },
)
AxiosInstance.interceptors.response.use(
  (response) => {
    if (response && response.data)
      return response.data

    return response
  },
  (error) => {
    // console.error(error.response);
    return Promise.reject(error)
  },
)

export default AxiosInstance
```

Figure 4.2a Connect API by axios config

```
import AxiosInstance from './axios-instance'

class EmailRequest {
  createVerifyEmail() {
    const url = '/email/verify'
    return AxiosInstance.post(url)
  }

  getVerifyEmailById(id, hash, signature) {
    const url = `/email/verify/${id}/${hash}`
    return AxiosInstance.get(url, { id, hash, signature })
  }
}

export default new EmailRequest()
```

Figure 4.2b Email request config

```
import AxiosInstance from './axios-instance'

class AuthRequest {
  registerUser(user) {
    const url = '/auth/sign-up'
    return AxiosInstance.post(url, user)
  }

  loginUser(user) {
    const url = '/auth/sign-in'
    return AxiosInstance.post(url, user)
  }

  logoutUser() {
    const url = '/auth/sign-out'
    return AxiosInstance.post(url)
  }

  loginAdmin(admin) {
    const url = '/auth/admin/sign-in'
    return AxiosInstance.post(url, admin)
  }

  logoutAdmin() {
    const url = '/auth/admin/sign-out'
    return AxiosInstance.post(url)
  }
}

export default new AuthRequest()
```

Figure 4.2d Authenticate request config

```
import AxiosInstance from './axios-instance.js'

class OrderRequest {
  getOrders(params) {
    const url = '/orders/'
    return AxiosInstance.get(url, { params })
  }

  updateOrder(id, status) {
    const url = `/orders/${id}`
    AxiosInstance.put(url, { status })
  }
}

export default new OrderRequest()
```

Figure 4.2e Password request config

```
import AxiosInstance from './axios-instance'

class ResourceRequest {
  createResourceImages(data) {
    const url = '/resource/images'
    return AxiosInstance.post(url, data)
  }
}

export default new ResourceRequest()
```

Figure 4.2c Resource request

```
import AxiosInstance from './axios-instance'

class ShopRequest {
  getShopsById(id_or_slug) {
    const url = `/shops/${id_or_slug}`
    return AxiosInstance.get(url)
  }

  getShopsProductsById(id_or_slug) {
    const url = `/shops/${id_or_slug}/products`
    return AxiosInstance.get(url)
  }

  getShops() {
    const url = '/shops'
    return AxiosInstance.get(url)
  }

  createShops(data) {
    const url = '/shops'
    return AxiosInstance.post(url, data)
  }

  updateShops(data) {
    const url = '/shops'
    return AxiosInstance.put(url, data)
  }

  getShopsProducts() {
    const url = '/shops/products'
    return AxiosInstance.get(url)
  }
}

export default new ShopRequest()
```

Figure 4.2f Shop request config

```
import AxiosInstance from './axios-instance'

class PasswordRequest {
  updatePasswordUser(data) {
    const url = '/password'
    return AxiosInstance.put(url, data)
  }

  forgotPasswordUser(data) {
    const url = '/password/forgot'
    return AxiosInstance.post(url, data)
  }

  resetPasswordUser(data) {
    const url = '/password/reset'
    return AxiosInstance.post(url, data)
  }

  updatePasswordAdmin(data) {
    const url = '/password/admin'
    return AxiosInstance.put(url, data)
  }

  forgotPasswordAdmin(data) {
    const url = '/password/admin/forgot'
    return AxiosInstance.post(url, data)
  }

  resetPasswordAdmin(data) {
    const url = '/password/admin/reset'
    return AxiosInstance.post(url, data)
  }
}

export default new PasswordRequest()
```

Figure 3.1 Password request config

```
import AxiosInstance from './axios-instance.js'

class ProductRequest {
  getProductsById(id) {
    const url = `/products/product/${id}`
    return AxiosInstance.get(url)
  }

  updateProducts(id, data) {
    const url = `/products/${id}`
    return AxiosInstance.put(url, data)
  }

  deleteProductsById(id) {
    const url = `/products/${id}`
    return AxiosInstance.delete(url)
  }

  recoveryProductsById(id) {
    const url = `/products/${id}`
    return AxiosInstance.post(url)
  }

  createProducts(data) {
    const url = '/products'
    return AxiosInstance.post(url, data)
  }

  getCategoriesById(id) {
    const url = `/products/categories/${id}`
    return AxiosInstance.get(url)
  }

  getCategoriesChildrenById(id) {
    const url = `/products/categories/${id}/children`
    return AxiosInstance.get(url)
  }

  createCategoriesAttributesById(id, data) {
    const url = `/products/categories/${id}/attributes`
    return AxiosInstance.post(url, data)
  }

  createCategories(data) {
    const url = '/products/categories'
    return AxiosInstance.post(url, data)
  }

  updateCategories(data) {
    const url = '/products/categories'
    return AxiosInstance.put(url, data)
  }

  deleteCategories(data) {
    const url = '/products/categories'
    return AxiosInstance.delete(url, data)
  }

  getAllCategoriesAttributes(data) {
    const url = '/products/categories/attributes'
    return AxiosInstance.get(url, data)
  }

  createCategoriesAttributes(data) {
    const url = '/products/categories/attributes'
    return AxiosInstance.post(url, data)
  }

  updateCategoriesAttributes(data) {
    const url = '/products/categories/attributes'
    return AxiosInstance.put(url, data)
  }

  deleteCategoriesAttributes(data) {
    const url = '/products/categories/attributes'
    return AxiosInstance.delete(url, data)
  }

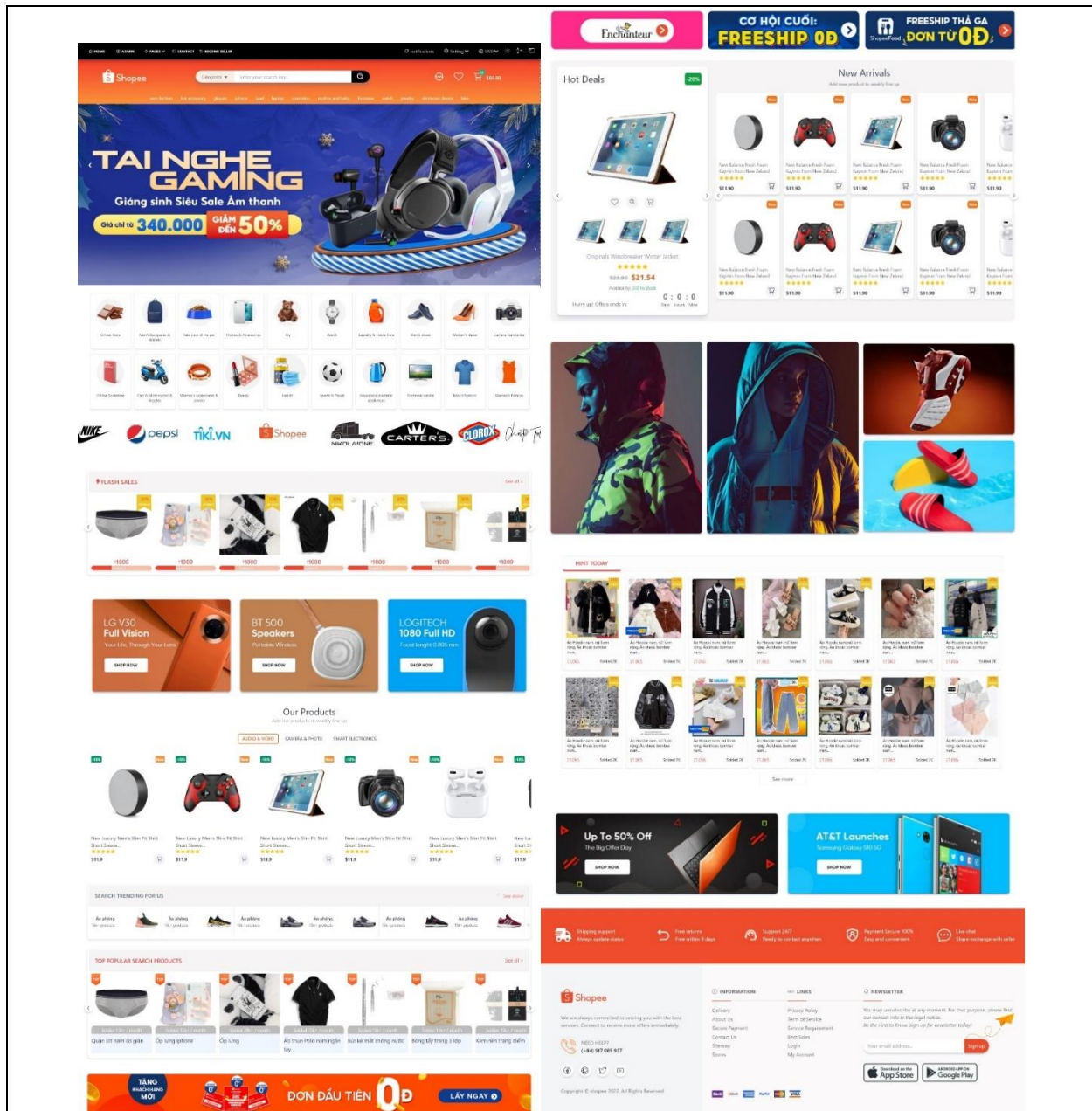
  searchCategoriesAttributes(input) {
    const url = `/products/categories/attributes/${input}`
    return AxiosInstance.get(url)
  }
}

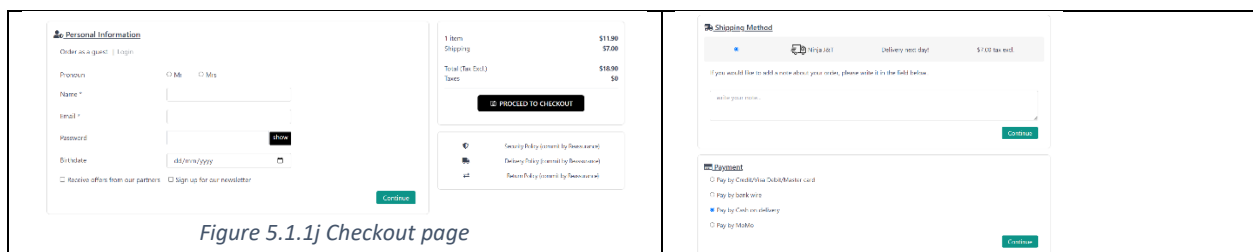
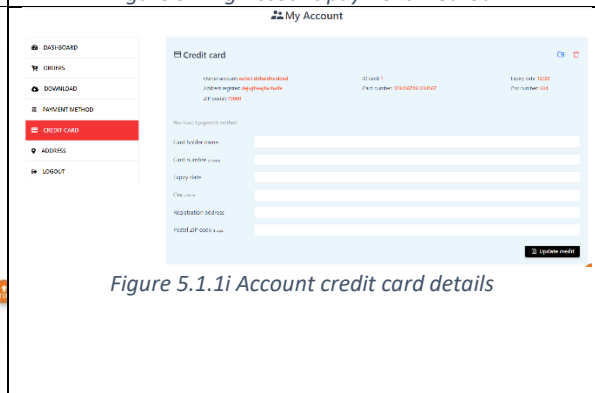
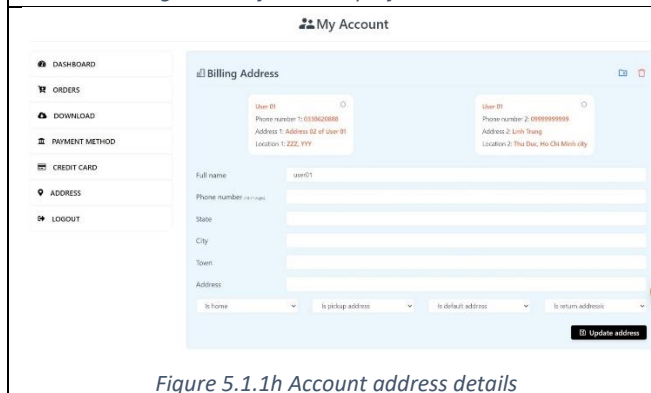
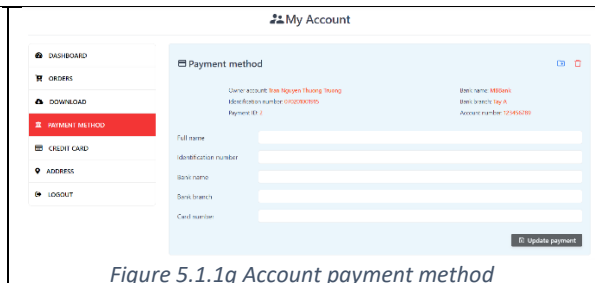
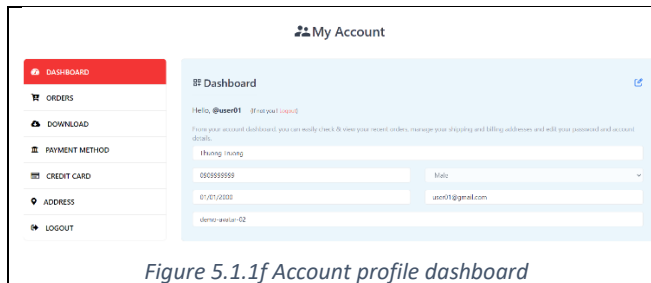
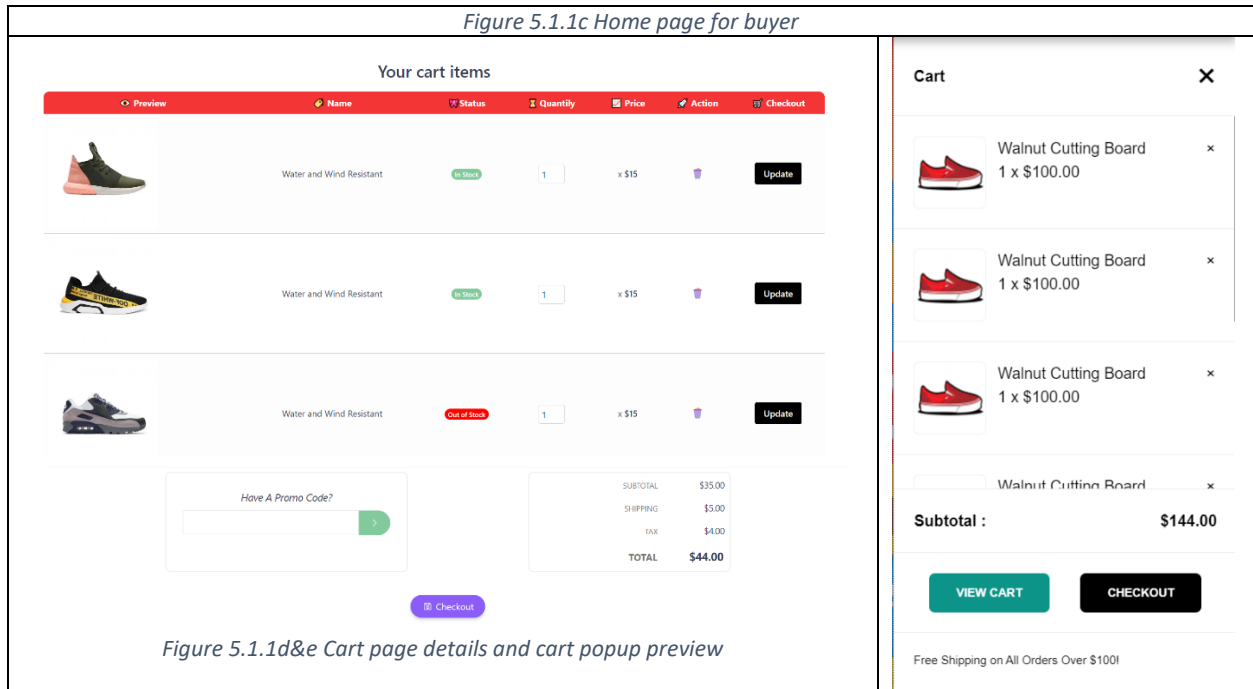
export default new ProductRequest()
```

Figure 3.1 Database query playground 1

## V. PRODUCTION

## 5.1 Overview







## Group 9 – Topic 10 - E-Commerce website for automotive parts

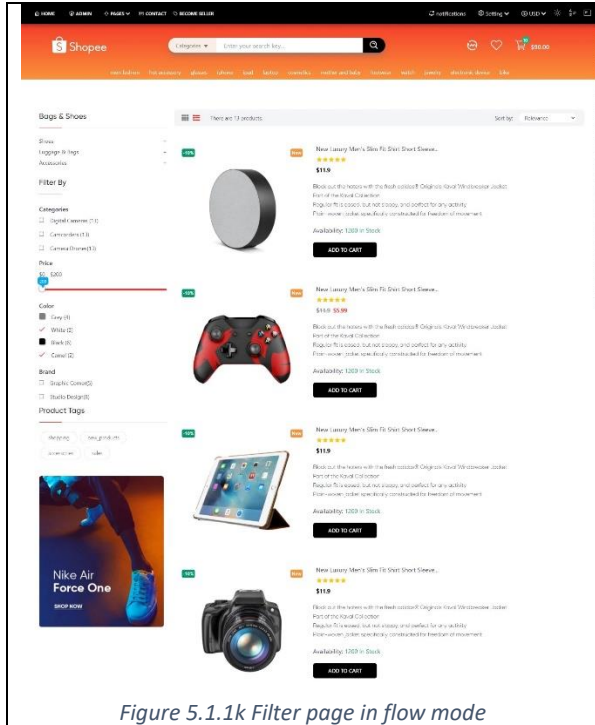


Figure 5.1.k Filter page in flow mode

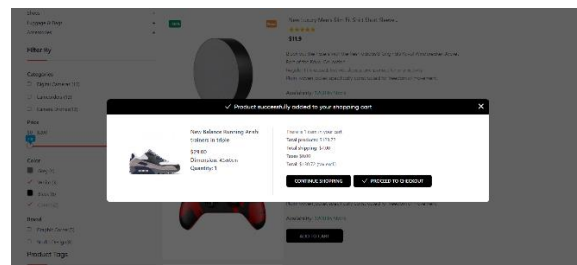


Figure 5.1.m Add to cart popup

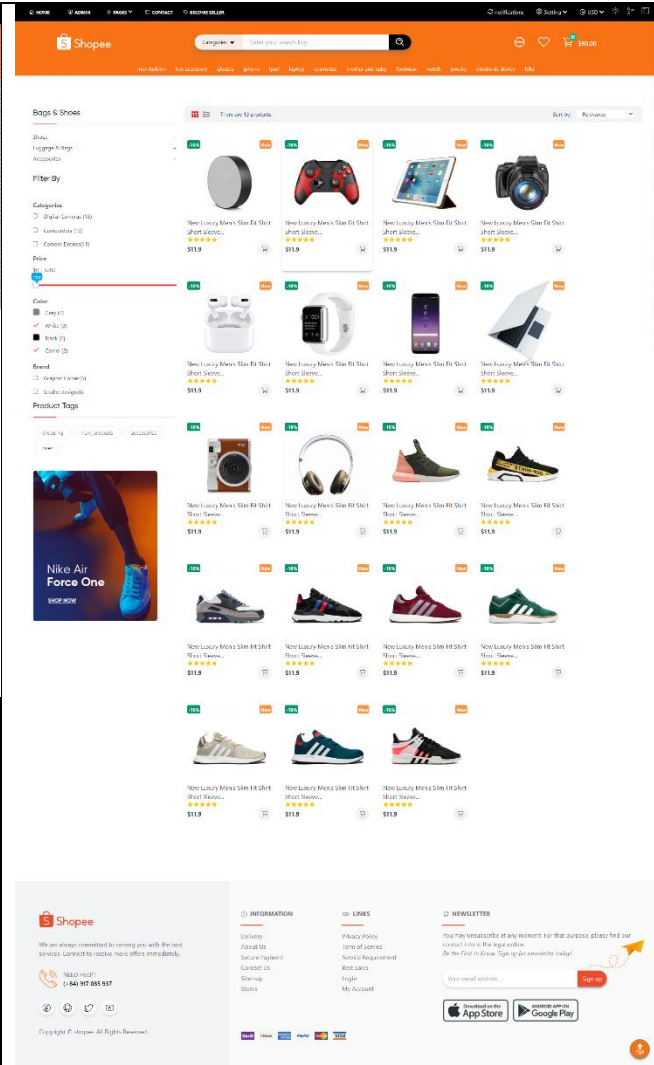


Figure 5.1.l Filter page in grid mode

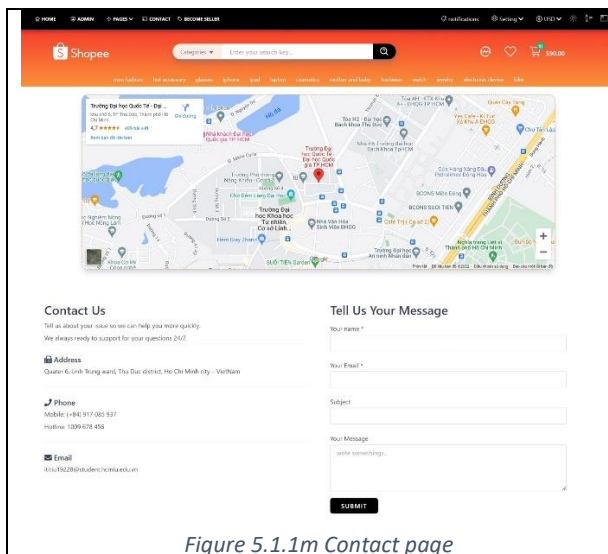


Figure 5.1.m Contact page

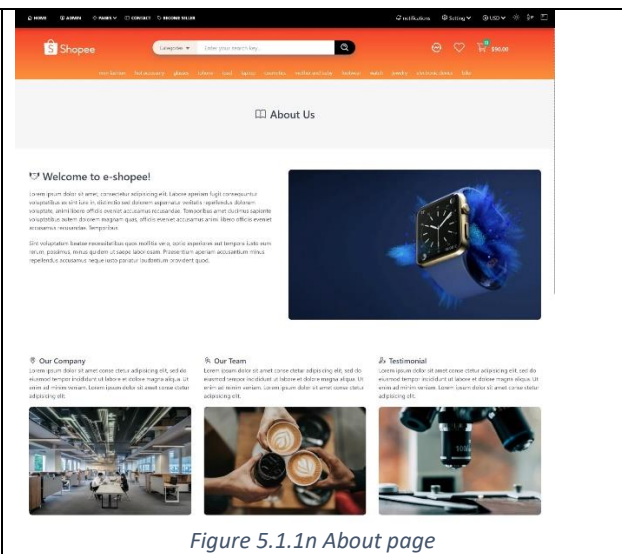


Figure 5.1.n About page



## II. Seller interface

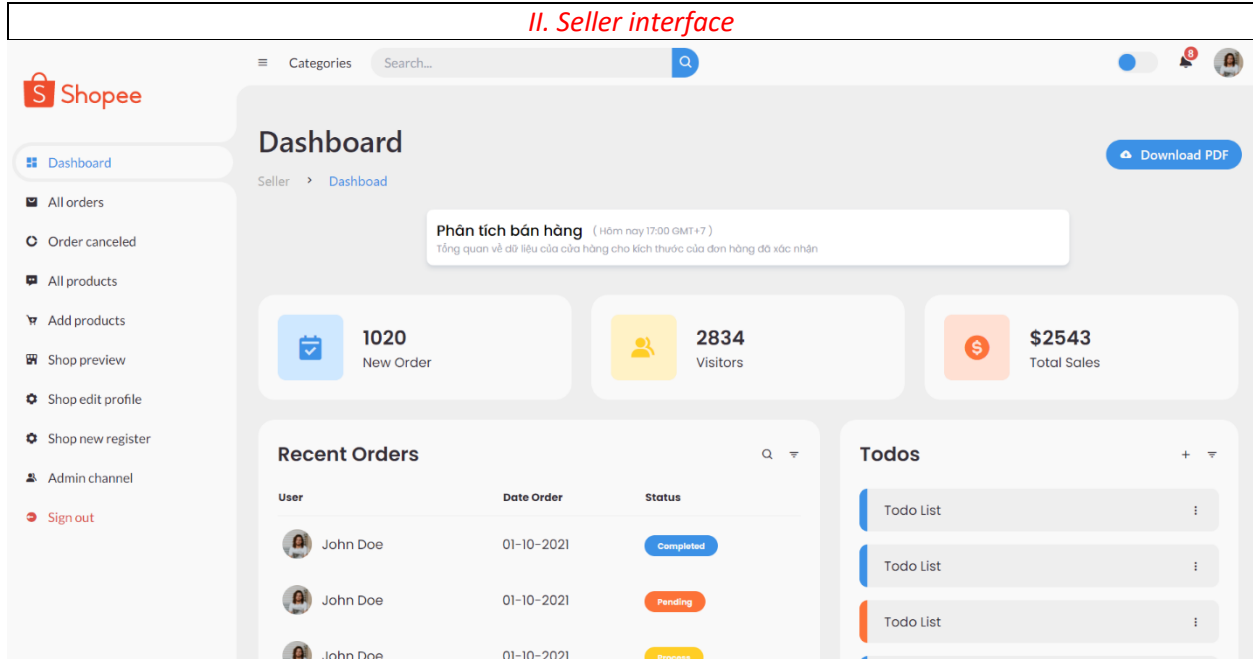


Figure 5.1.2a Seller dashboard page

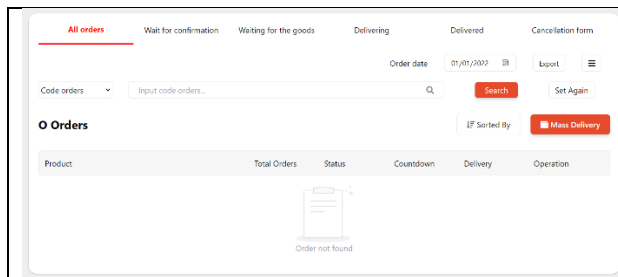


Figure 5.1.2b All orders page

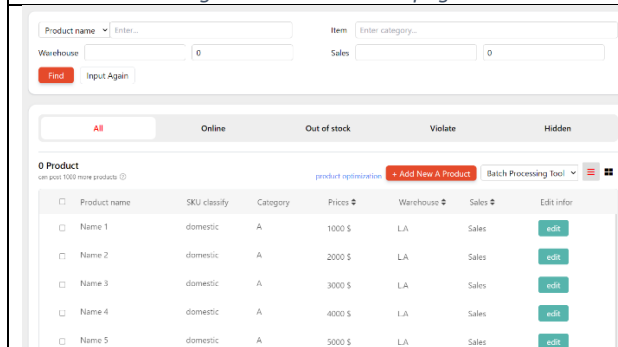


Figure 5.1.2c All products page

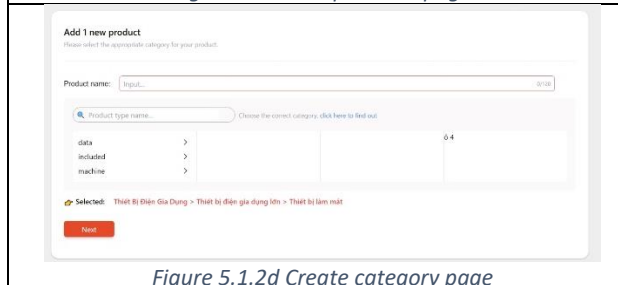


Figure 5.1.2d Create category page

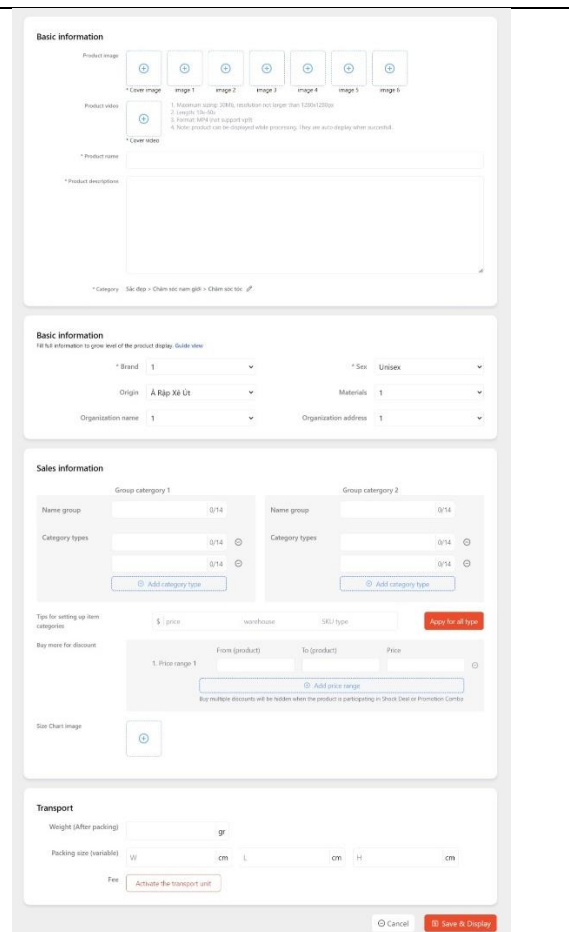
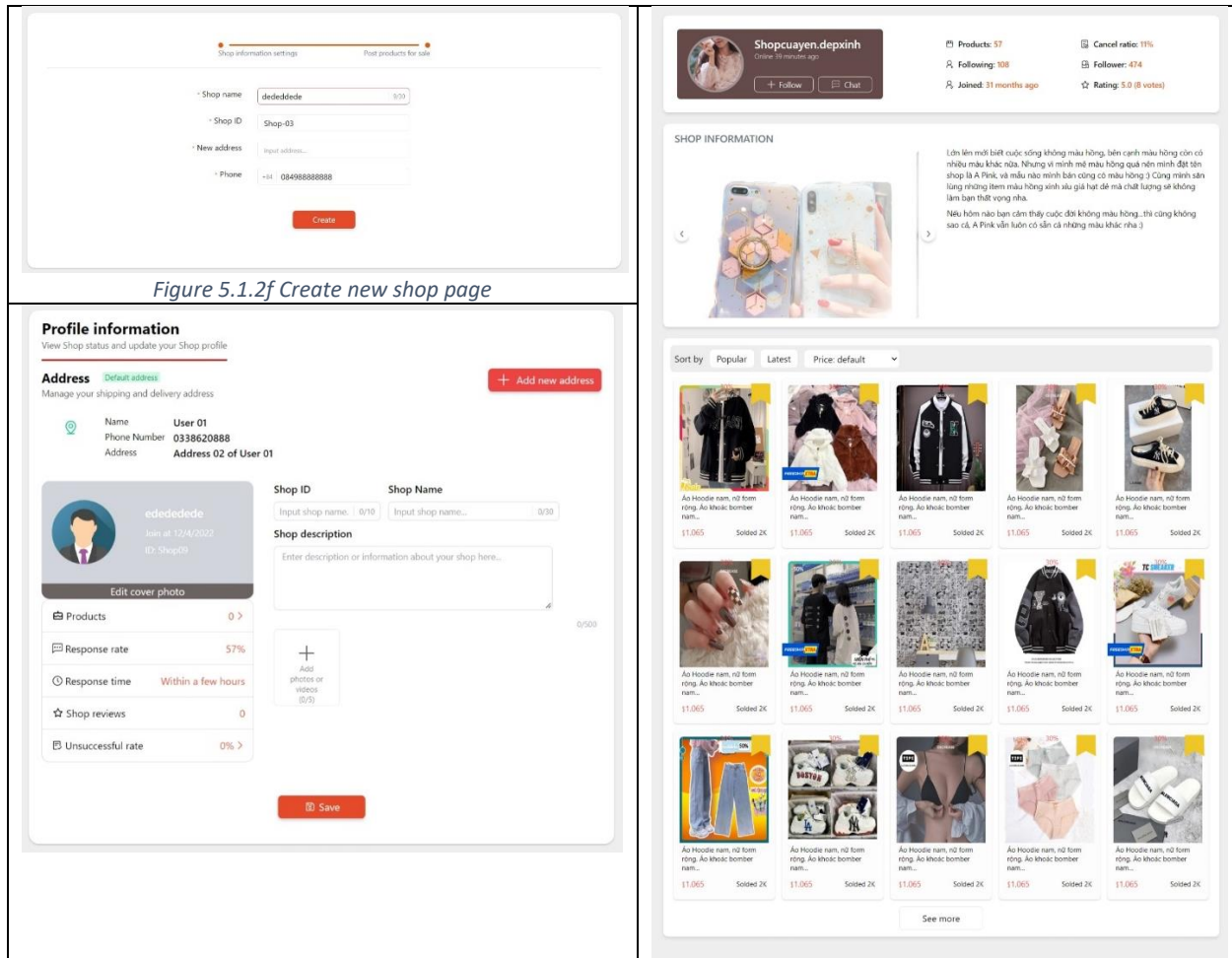


Figure 5.1.2e Add products by category page



### III. Admin interface

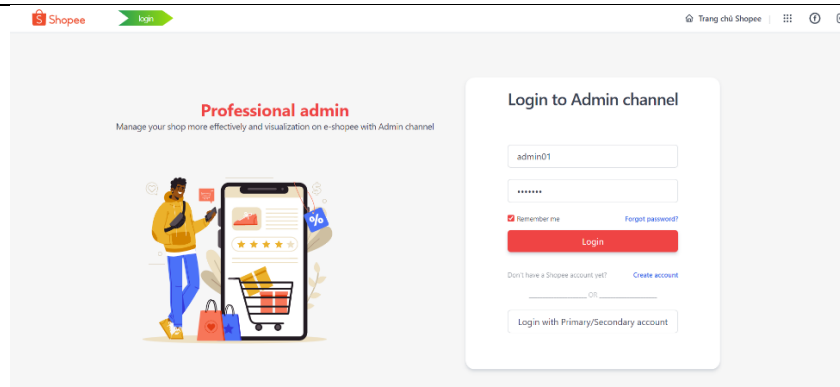






Figure 5.1.3b Admin dashboard page

## 5.2 Analysis and unit test

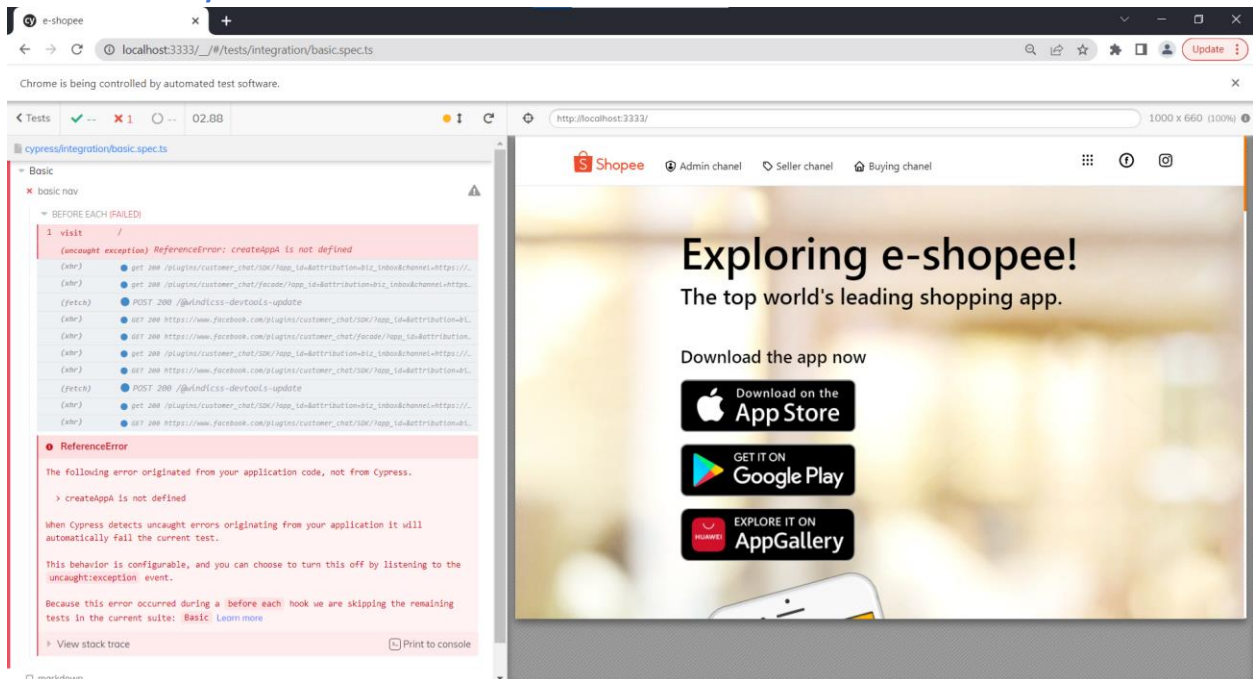
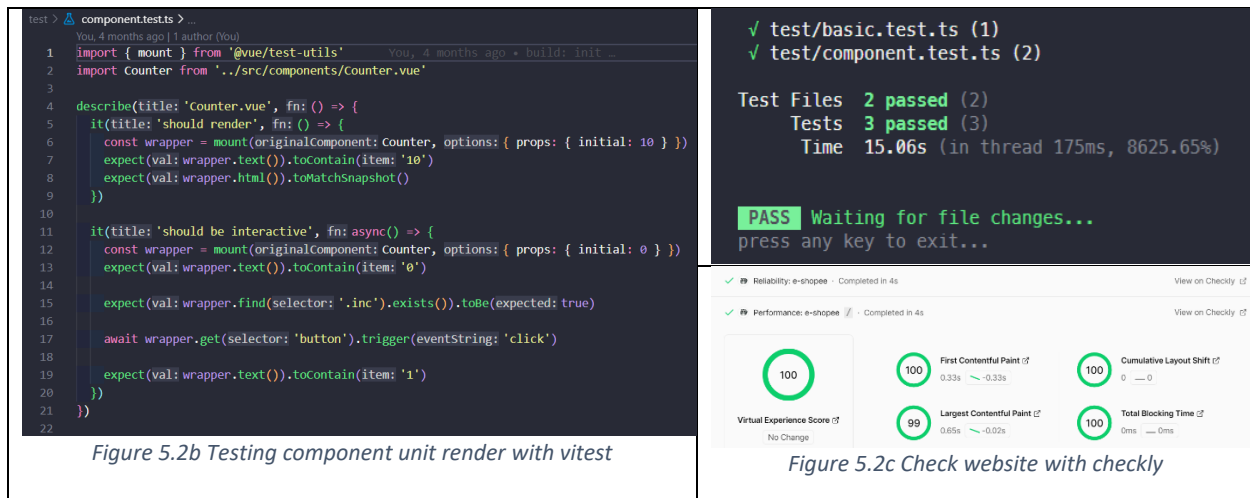


Figure 5.2a Testing router with cypress

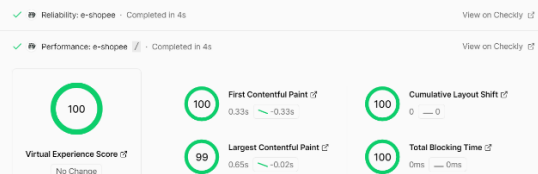


```

✓ test/basic.test.ts (1)
✓ test/component.test.ts (2)

Test Files 2 passed (2)
Tests 3 passed (3)
Time 15.06s (in thread 175ms, 8625.65%)

PASS Waiting for file changes...
press any key to exit...
  
```



### 5.3 Test for use cases

- Before entering a certain url, if you are not logged in, the url will be redirected to the login page.
- In buyer session:
  - If registered with the same name, the server reports an error that the user already exists.
  - Date of birth must be in DD/MM/YYYY format.
  - The registered phone number in the account must be 11 digits.
  - credit card number must be 16 digits, cvv number must be 3 digit, post ZIP code must be 5 digits.
  - When registering and updating an account, the fields cannot be left blank.
- In seller session:
  - When registering to add a product, if the address already exists in the account profile, there is no need to fill it in anymore.
- In admin session:
  - Before login to admin must logout user account.

## VI. CONCLUSIONS

- Through this project, we can learn about create SPA (Single Page App) as dynamic website to communicate with server through API connections.
- Authenticate with token access.
- Learning about MVC model and CRUD actions in API.
- Practicing and enhancing about SQL actions and database knowledge.
- Understanding about principle of database working.
- Improving coding skills and UI/UX.
- Initial forming about microservices project by components system.

