GROUP PROJECT FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	WEBG301 - Project Web		
Submission date		**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	05/07/2022
Student Name	Bùi Hương Linh	**Student ID**	GBH200662
Class	GCH1002	**Assessor name**	Nguyen Dinh Tran Long

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student's signature	Linh

Grade:

❒ **Summative Feedback:**	❒ **Resubmission Feedback:**	
Grade:	**Assessor Signature:**	**Date:**
Signature & Date:		

Table of Contents

Part 1 – Users' requirements (GROUP) 6
I. User stories template 6
II. Use case diagram 7
Part 2 – System Design (GROUP) 8
I. Site map 8
II. Entity relationship diagram 9
III. Wireframes 10
1. Home page 10
2. Register 11
3. Login 12
4. Product List 13
5. Category List 15
6. Contact List 16
Part 3 – System Implementation (INDIVIDUAL) 17
I. Source code 19
1. Create project 19
2. Data Fixtures 22
3. Controller 23
4. Form 28
II. Web screenshots 29
Part 4 - Conclusion (INDIVIDUAL) 44
I. Advantages of website 44

II. Disadvantages of website 44
III. Lesson learnt 44
IV. Future improvements 44
Appendix: (GROUP) 45

Figure 1: Use case diagram 7
Figure 2:Site map 8
Figure 3: ERD 9
Figure 4: Home Page 10
Figure 5: Register 11
Figure 6: Login 12
Figure 7: Product List – Add new product 13
Figure 8: Product Detail 14
Figure 9: Category List – Add new category 15
Figure 10: Contact List 16
Figure 11: The MVC design pattern 18
Figure 12: Create database 19
Figure 13: Create entity 20
Figure 14: Contact Entity 21
Figure 15: Create fixtures and code 22
Figure 16: Create controller 23
Figure 17: Contact Controller - Index 24
Figure 18: Contact Controller – Delete 25
Figure 19: Contact Controller – Add 26
Figure 20: Contact Controller – Edit 27
Figure 21: Contact Type 28

Figure 22: Register 29
Figure 23: Login 30
Figure 24: Home Page 31
Figure 25: Product List 32
Figure 26: Product Detail 33
Figure 27: Add product 34
Figure 28: Add success 35
Figure 29: Delete product 36
Figure 30: Delete success 37
Figure 31: Edit product 38
Figure 32: Update success 40
Figure 33: Category page 41
Figure 34: Category Detail 42
Figure 35: Contact Detail 43

Part 1 – Users' requirements (GROUP)

I. User stories template

No	As a <type of user/admin	I want to<global/objective	So that<benefit, result>
1	Admin/Customer	Login/Logout	I can login/logout the website.
2	Admin	Add new product	I can add a new product and all product I can add, but I can't upload image, in the future I can improve.
3	Admin	Edit product	I can edit any product.
4	Admin/Customer	Delete product	I can choose product and delete some product, I can.
5	Admin/customer	View product	I can view any product easily.
6	Admin	Add new category	I can add a new category and all category I can add, but I can't upload image, in the future I can improve
7	Admin	Delete category	I can choose category and delete some category, I can.
8	Admin	Edit category	I can edit and product.
9	Admin	View category	I can view any category easily.
10	Admin	Add new contact	I can add my contact, I want.
11	Admin/Customer	View contact	I can view any contact easily.
12	Admin	Delete contact	I can delete any contact, I want.
13	Admin	Edit contact	I can edit any product

II. Use case diagram

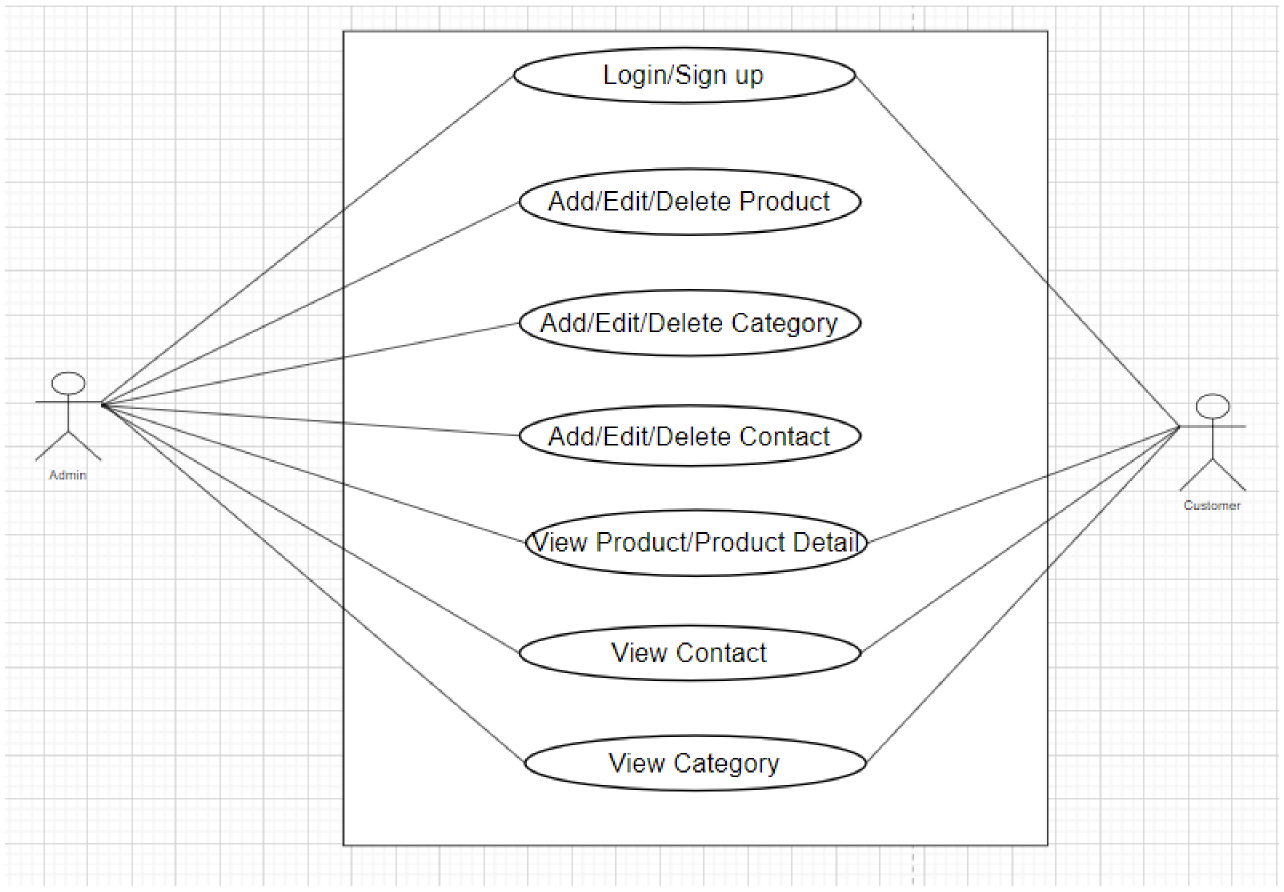


Figure 1: Use case diagram

Customer can only view the product, the product details, and the contact method, whereas admin has access to all features except subscription. Customers should register and provide their details.

Part 2 – System Design (GROUP)

I. Site map

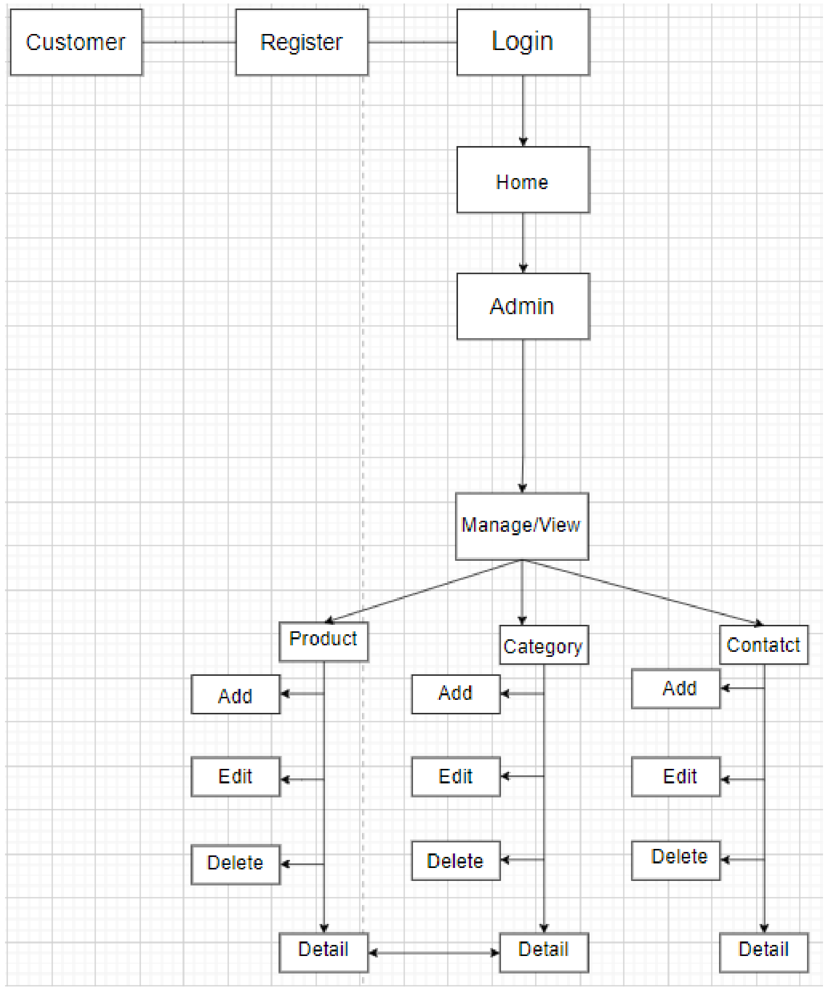


Figure 2:Site map

There are 4 main pages on the website. Customers may view all products on the product page, and there is also a link to a product detail page where they can look more closely and decide whether or not to purchase. Then there is the registration page to register and the login page for the administrator to access the admin site, and the customer's to view the product and enter their information when making a purchase . The admin has a number of options on the admin site for editing a product, category, or contact.

II. Entity relationship diagram

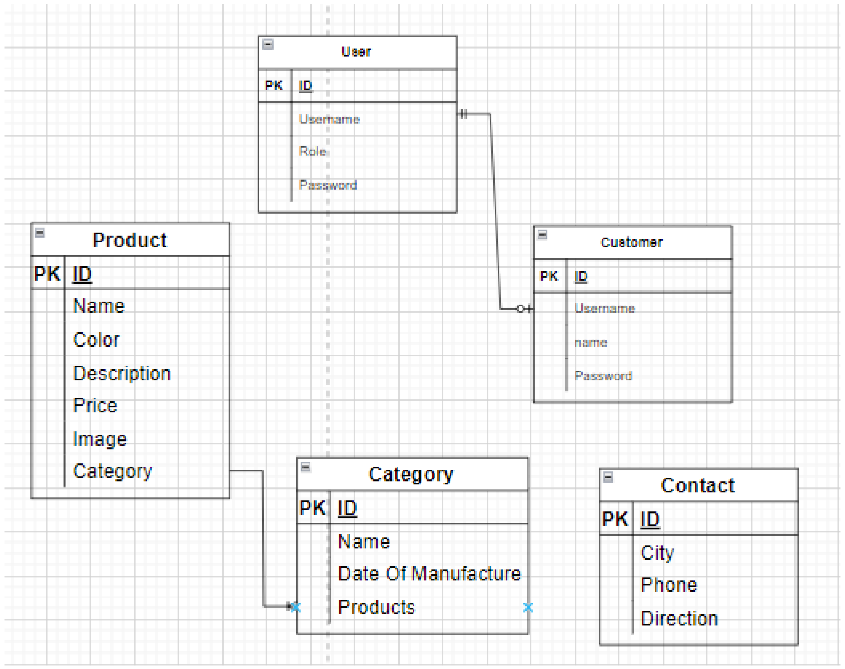


Figure 3: ERD

The project consists of 4 entities. The product and category are connected in a one-to-many relationship. It implies that 1 category will contain many products.The user entity contains the username and login information for all users, including admin and customers. Customers can register for a new account and log in immediately afterwards. The contact is used to display some store information.

III. Wireframes

1. Home page

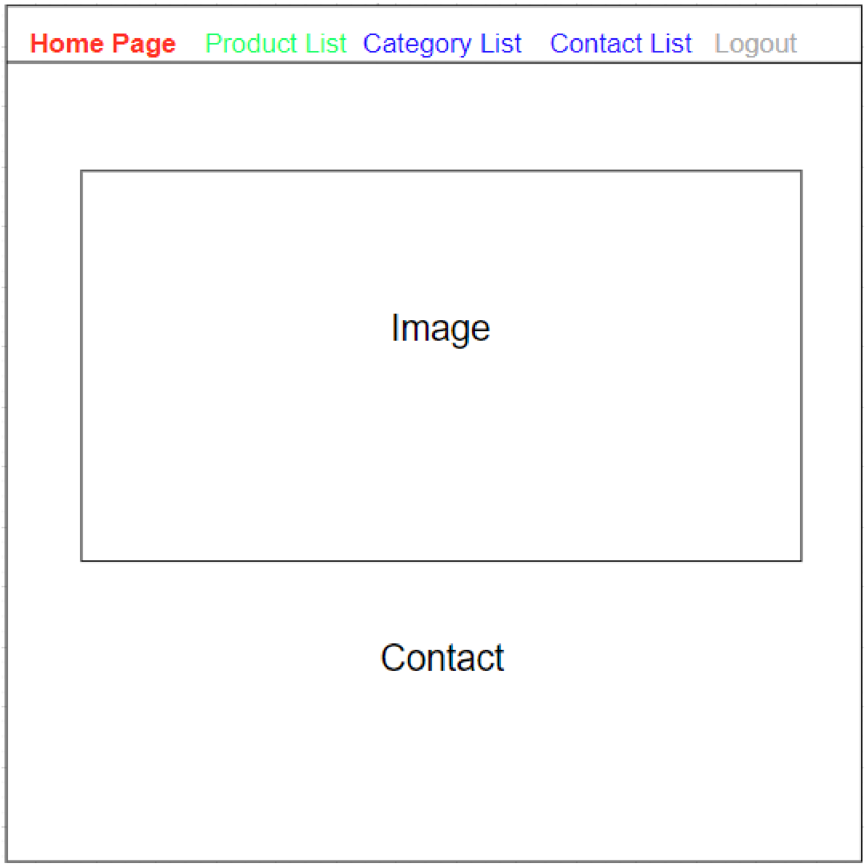


Figure 4: Home Page

This is the wireframe for the homepage. The homepage consists of a navigation bar, in the middle containing an image for promotion and finally contact information.

2. Register

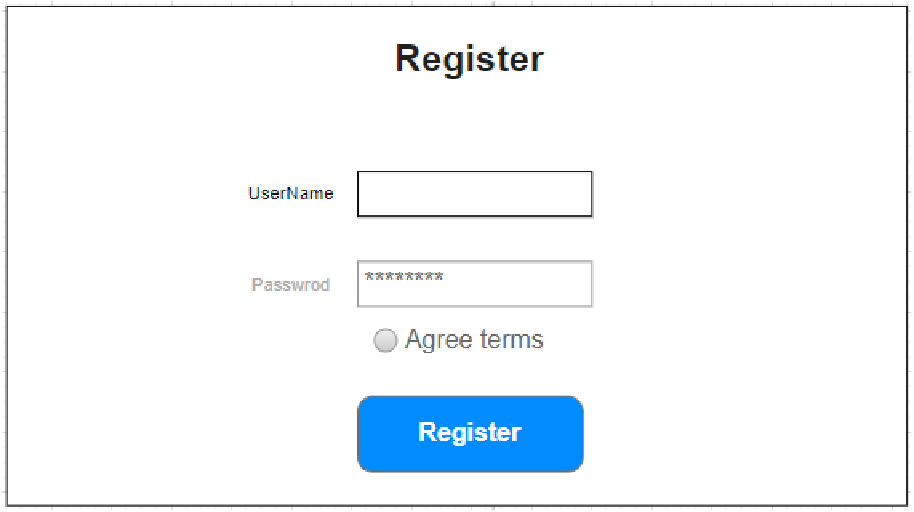


Figure 5: Register

This wireframe is used for customer registering that included username and password for the customer to enter the homepage.

3. Login

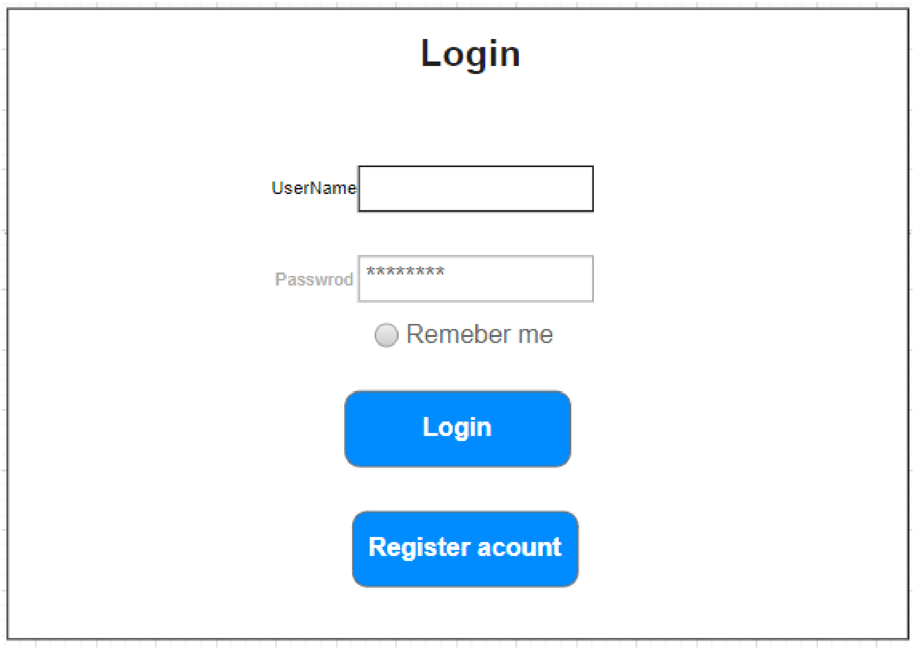


Figure 6: Login

This login wireframe is used by the administrator to access the administration page and by the customer to login. Because the admin and the customer cannot see and perform the same actions, each user's role is defined using login. Both a username and a password are required.

4. Product List

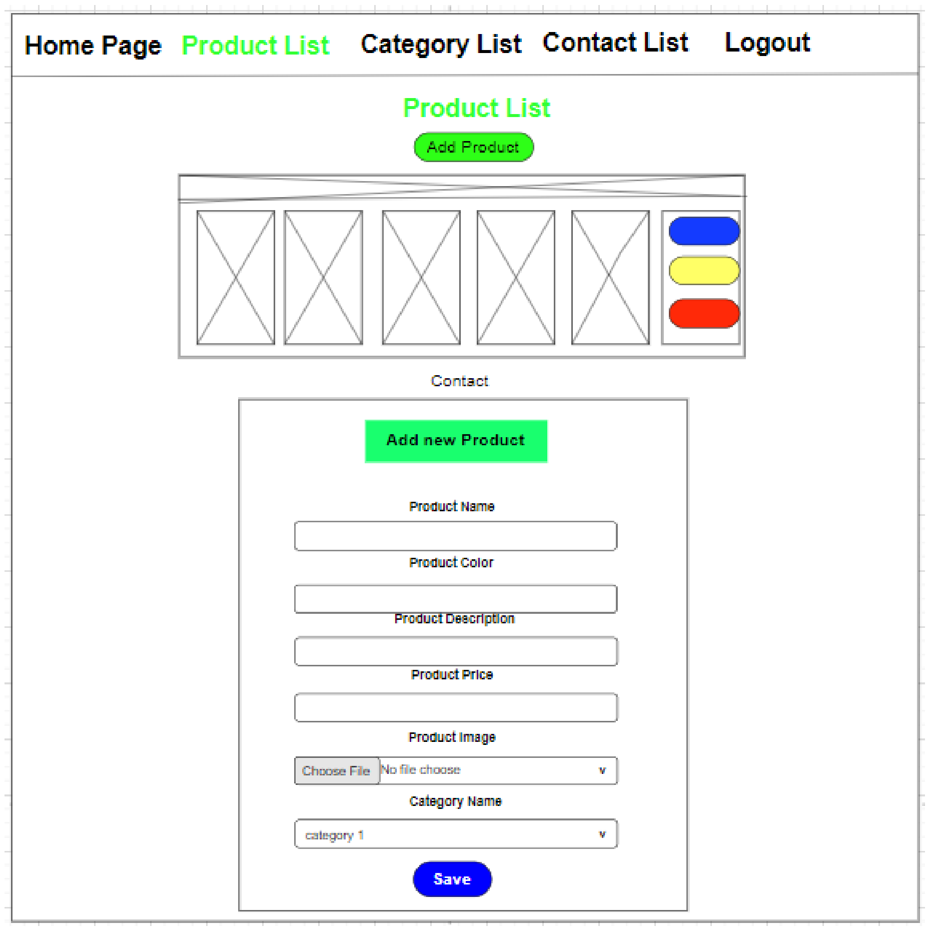
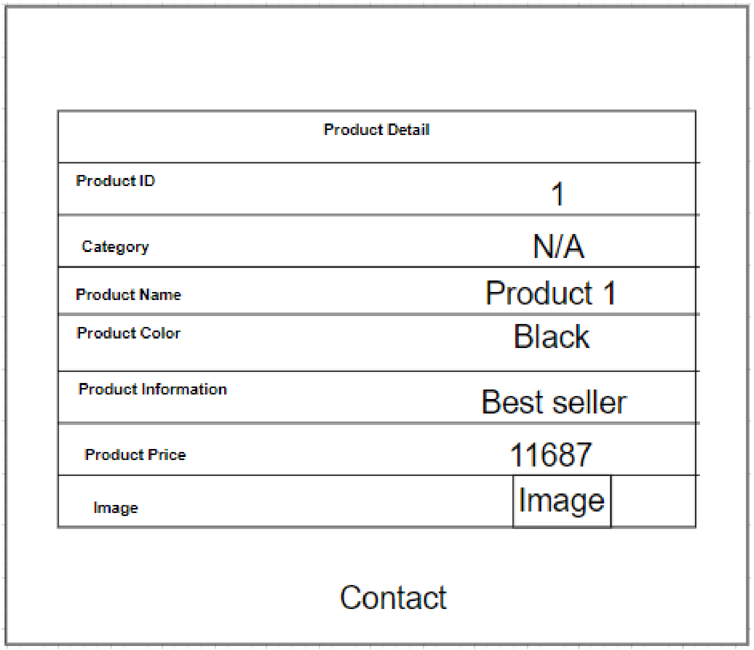


Figure 7: Product List – Add new product

On this page will display all the products, categories, contacts and operations for the administrator to handle all the information on the site. Admins can add products and admins can also edit, update or delete them.



Product Detail	
Product ID	1
Category	N/A
Product Name	Product 1
Product Color	Black
Product Information	Best seller
Product Price	11687
Image	Image

Contact

Figure 8: Product Detail

In this page, the main content will be products with full details that customers need, larger images, more detailed product descriptions.

5. Category List

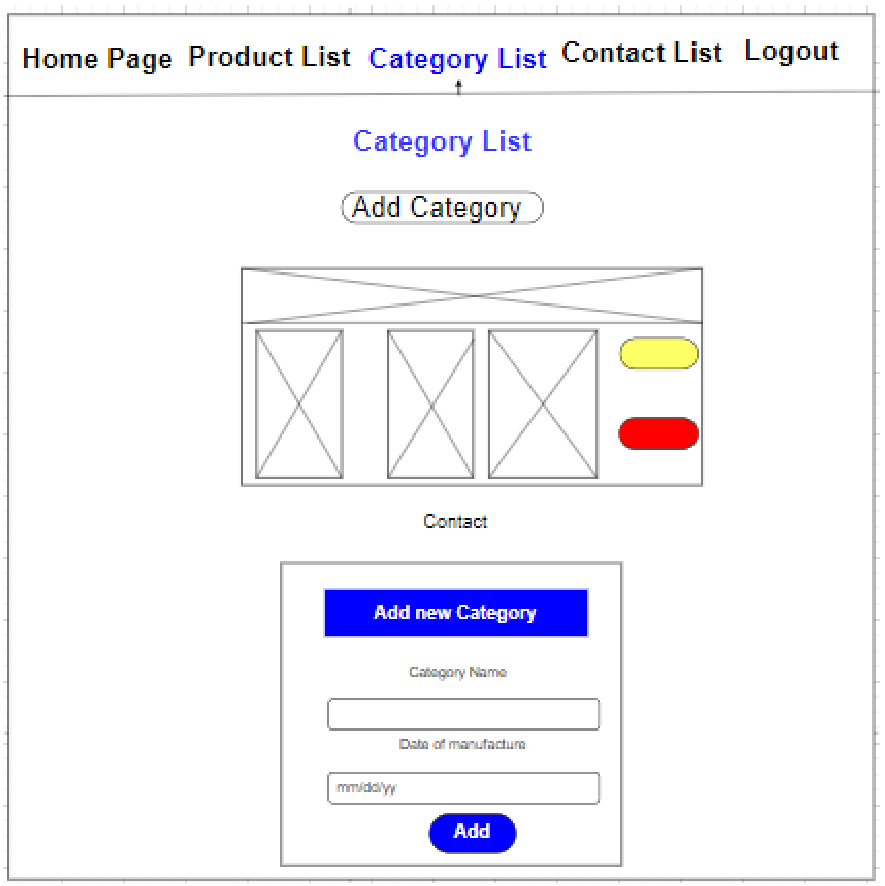


Figure 9: Category List – Add new category

On this page shows all product categories, contacts and operations for the administrator to handle all information on the site. Admin can add categories and admin can also edit and delete them.

6. Contact List

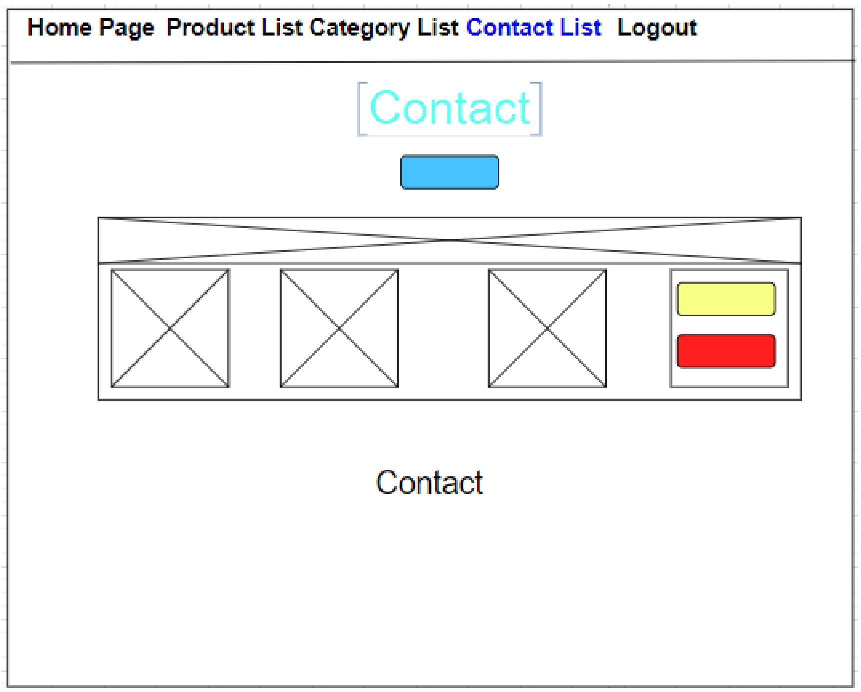


Figure 10: Contact List

On this page we can see the shop contact information. Here admin can add contacts and admin can also edit and delete them.

Part 3 – System Implementation (INDIVIDUAL)

Symfony is built on the MVC architecture, a popular web design pattern with three levels:

• The Model represents the information on which the application operates - its business logic.

• The View converts the model into a web page that the user can interact with.

• The Controller responds to user actions by modifying the model or view as needed.

The MVC architecture separates the business logic (model) from the presentation (view), making it easier to maintain. For example, if an application should be able to run on both standard web browsers and handheld devices, all that is required is a new view; the original controller and model can be kept. The controller helps to hide the protocol information used for the request from the model and view. Furthermore, the model abstracts the logic of the data, making the view and action independent of the application's database type, for example.

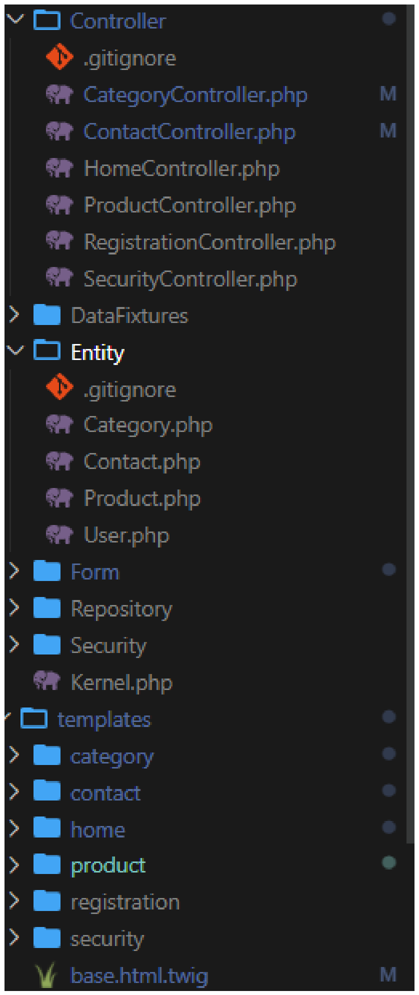


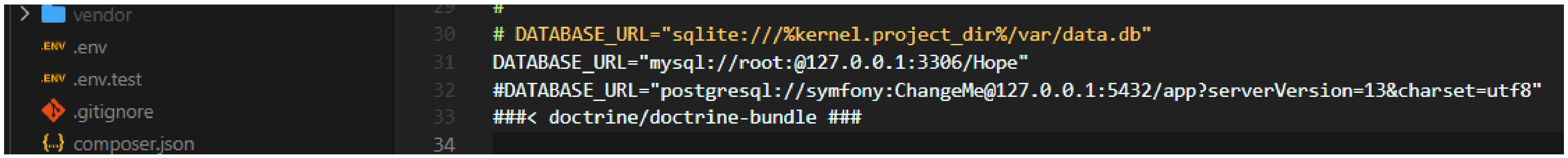
Figure 11: The MVC design pattern

The project contains entities as models, templates as views, and controllers. There is also a forms directory to support templates and a repository for processing the database.. Data constants are used to enter data quickly into the database and are often used. use for loops to add to the database faster.

I. Source code

1. Create project

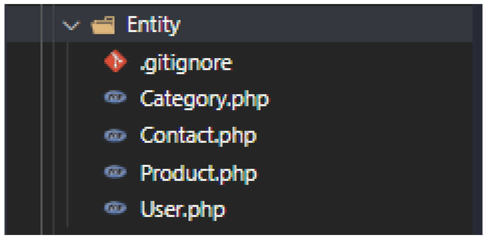
To begin developing a symphony project, we first need run the following command in the terminal: symfony new 'name'--ful--version = 5.3. To ensure stability, we use version 5.3. Then, edit the env file to gain access to the database, and finally, create the database. We need to create prepared entities with relationships between them once we have a database.



```
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root:@127.0.0.1:3306/Hope"
#DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Figure 12: Create database

I started building databases and entities for each table after I had run it. I started the xampp program and configured the file (.env) comment line 32 and set line 31. (DATABASE_URL="mysql://root:@127.0.0.1:3306) /Hope") and start I run the sentence to php bin/console doctrine:database:create to create a database named "Hope".



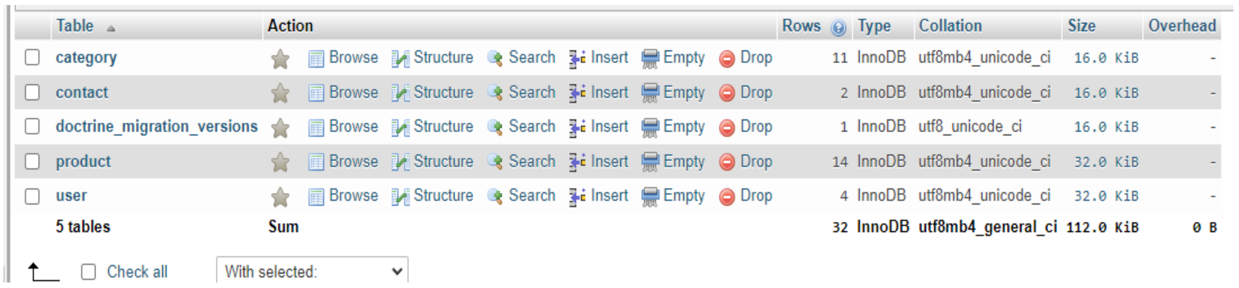
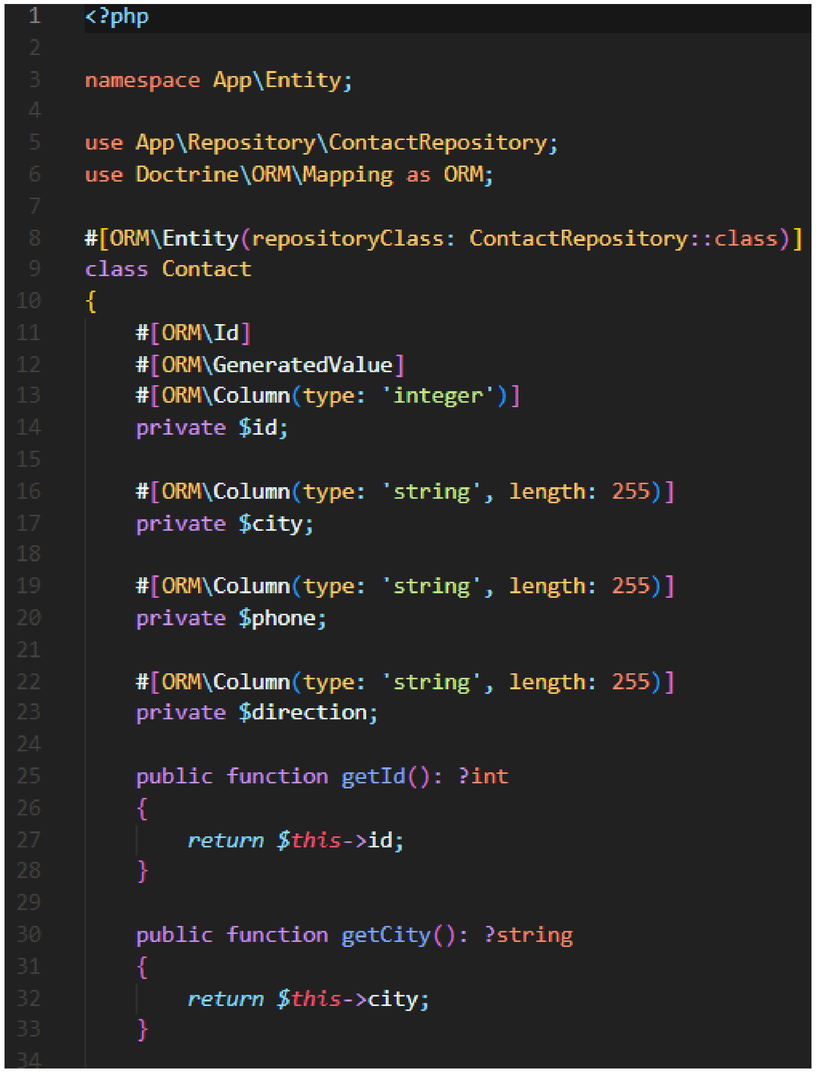


Table	Action	Rows	Type	Collation	Size	Overhead
category	Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
contact	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
doctrine_migration_versions	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8_unicode_ci	16.0 KiB	-
product	Browse Structure Search Insert Empty Drop	14	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
user	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
5 tables	Sum	32	InnoDB	utf8mb4_general_ci	112.0 KiB	0 B

Figure 13: Create entity

After creating it, I start the command (php bin/console make:entity) to create an entity for each table in turn product, category, contact, user. But user, I have to initialize with another command (composer require symfony/security-bundle) first only then can make:entity for the user.



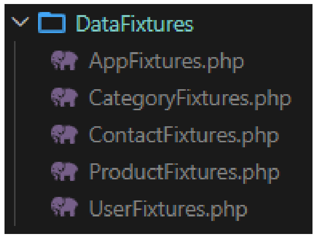
```php
<?php

namespace App\Entity;

use App\Repository\ContactRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: ContactRepository::class)]
class Contact
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $city;

    #[ORM\Column(type: 'string', length: 255)]
    private $phone;

    #[ORM\Column(type: 'string', length: 255)]
    private $direction;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getCity(): ?string
    {
        return $this->city;
    }

```

Figure 14: Contact Entity

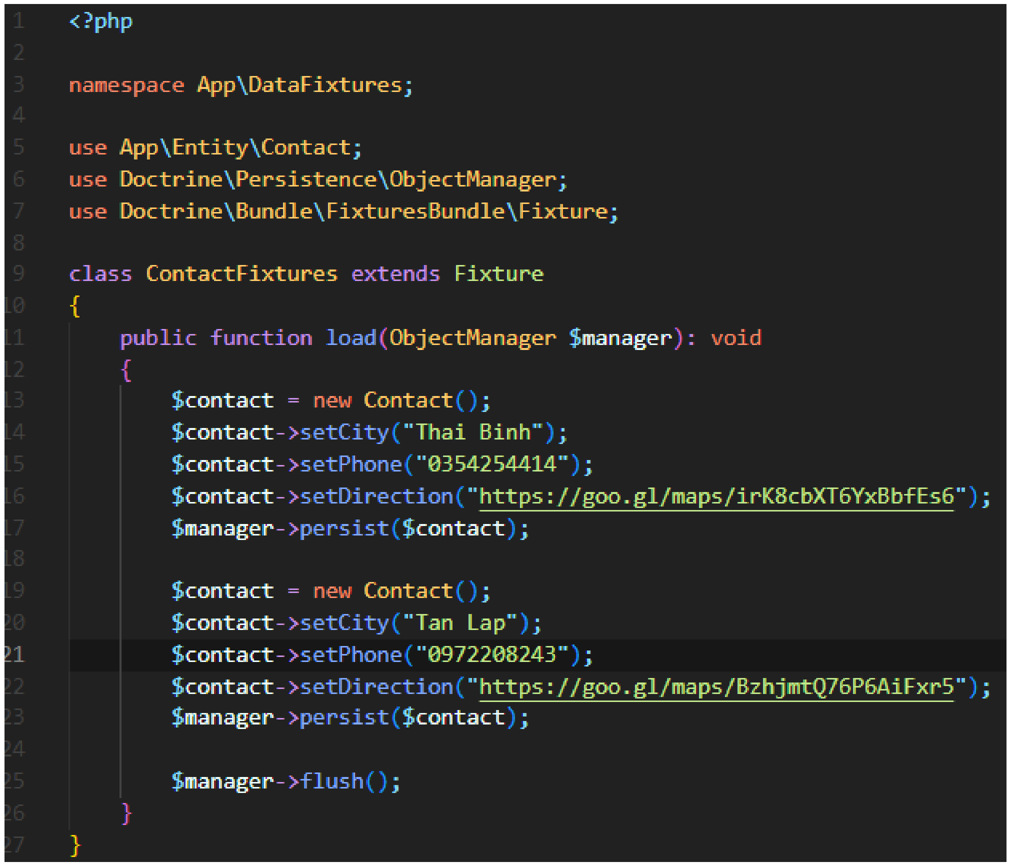
After creating the entity, the code will appear with getter and setter methods for all of the attributes.

Run the 'migration' so that the Symfony framework can generate the sql script, and then apply the following 'php bin/console doctrine:migrations:migrate' to execute the sql that will create the tables and columns.

2. Data Fixtures



```
DataFixtures
    AppFixtures.php
    CategoryFixtures.php
    ContactFixtures.php
    ProductFixtures.php
    UserFixtures.php
```

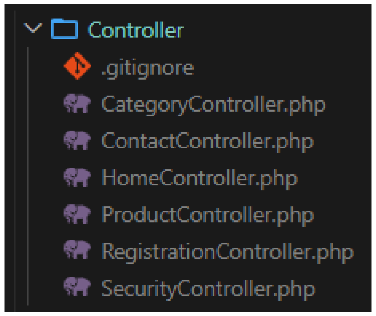


```php
<?php

namespace App\DataFixtures;

use App\Entity\Contact;
use Doctrine\Persistence\ObjectManager;
use Doctrine\Bundle\FixturesBundle\Fixture;

class ContactFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $contact = new Contact();
        $contact->setCity("Thai Binh");
        $contact->setPhone("0354254414");
        $contact->setDirection("https://goo.gl/maps/irK8cbXT6YxBbfEs6");
        $manager->persist($contact);

        $contact = new Contact();
        $contact->setCity("Tan Lap");
        $contact->setPhone("0972208243");
        $contact->setDirection("https://goo.gl/maps/BzhjmtQ76P6AiFxr5");
        $manager->persist($contact);

        $manager->flush();
    }
}
```

Figure 15: Create fixtures and code

Then I run the sentence up (php bin/console make:fixture) to create each fixture named 'contact'.... after coding the fixtures I run (php bin/console doctrine:fixtures:load) to load the fixtures into the DB.

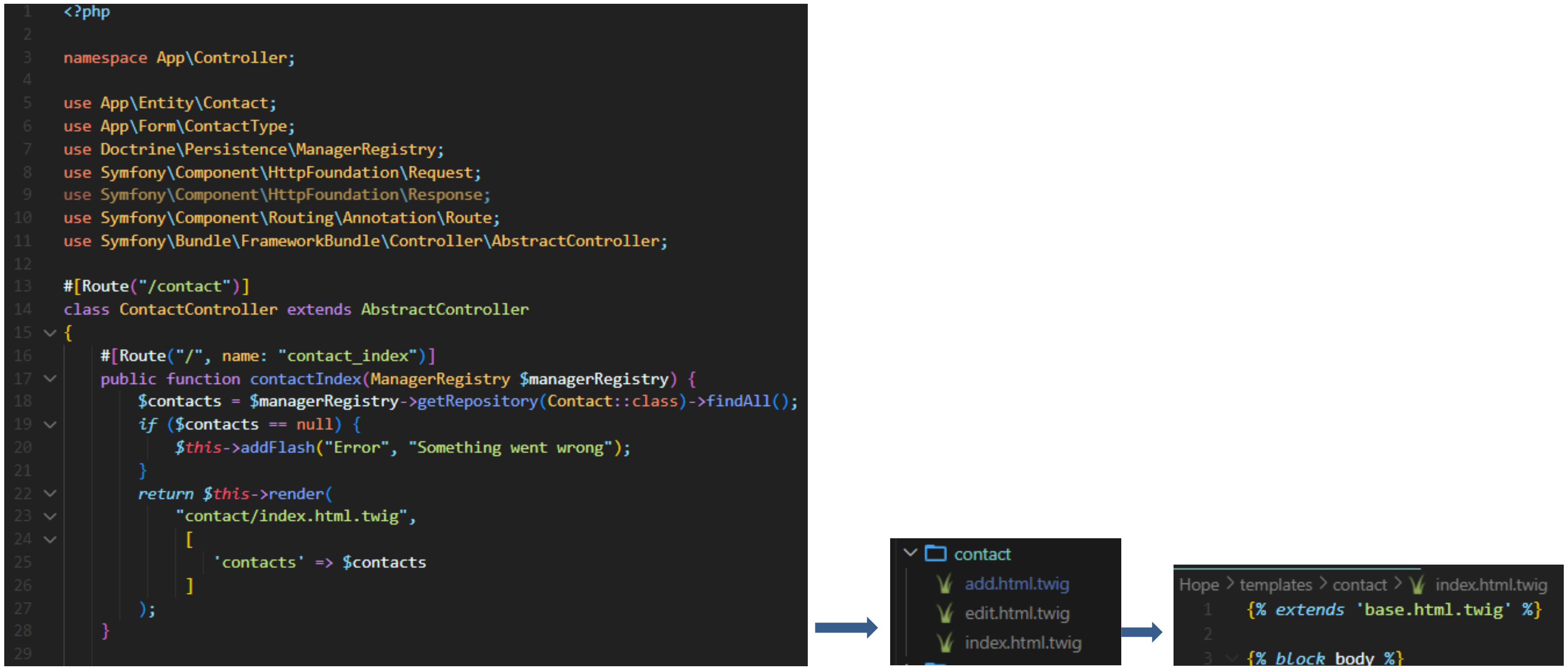
3. Controller



```
Controller
    .gitignore
    CategoryController.php
    ContactController.php
    HomeController.php
    ProductController.php
    RegistrationController.php
    SecurityController.php
```

Figure 16: Create controller

After that, I began naming each controller as I created them using the command (php bin/console make:controller).



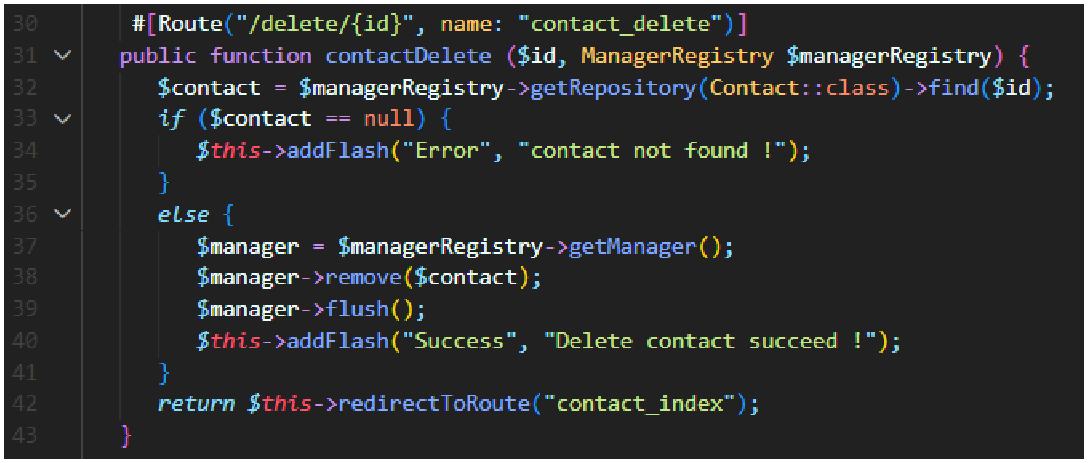
```php
<?php

namespace App\Controller;

use App\Entity\Contact;
use App\Form\ContactType;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

#[Route("/contact")]
class ContactController extends AbstractController
{
    #[Route("/", name: "contact_index")]
    public function contactIndex(ManagerRegistry $managerRegistry) {
        $contacts = $managerRegistry->getRepository(Contact::class)->findAll();
        if ($contacts == null) {
            $this->addFlash("Error", "Something went wrong");
        }
        return $this->render(
            "contact/index.html.twig",
            [
                'contacts' => $contacts
            ]
        );
    }

```

```
contact
    add.html.twig
    edit.html.twig
    index.html.twig
```

```
Hope > templates > contact > index.html.twig
{% extends 'base.html.twig' %}

{% block body %}
```

Figure 17: Contact Controller - Index

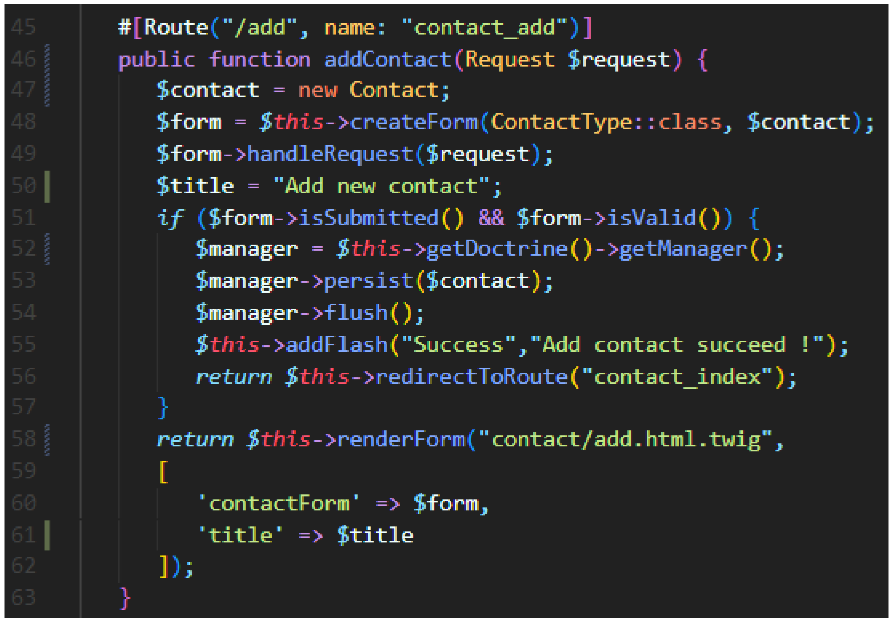
Import all of the libraries that will be used in the controller into the controller. The first is #[Route("/contact")] so that when I click on it, it shows /conatct. All contact data will be found in the index function by using the find all function from the object repository.



```
 #[Route("/delete/{id}", name: "contact_delete")]
public function contactDelete ($id, ManagerRegistry $managerRegistry) {
   $contact = $managerRegistry->getRepository(Contact::class)->find($id);
   if ($contact == null) {
      $this->addFlash("Error", "contact not found !");
   }
   else {
      $manager = $managerRegistry->getManager();
      $manager->remove($contact);
      $manager->flush();
      $this->addFlash("Success", "Delete contact succeed !");
   }
   return $this->redirectToRoute("contact_index");
}
```

Figure 18: Contact Controller – Delete

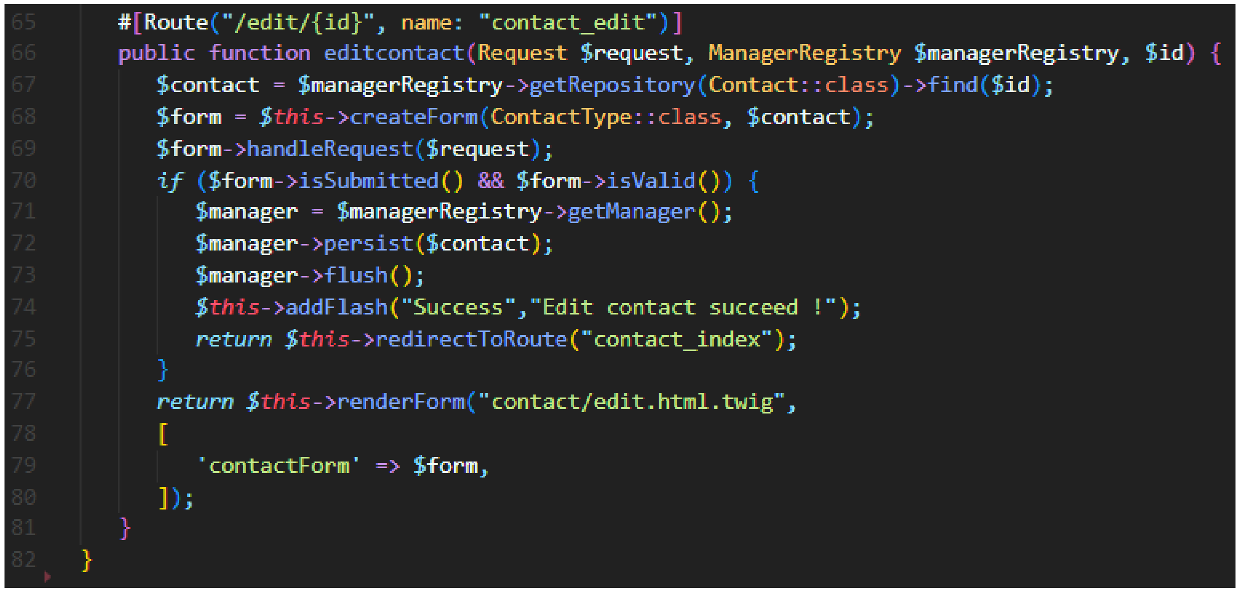
The first delete function is also #[Route("/delete/{id}", name: “contact_delete”) delete will need a parameter to determine the correct and appropriate, and it will take the id of contact you choose and want to delete the statement (/{id}) to specify you want to delete, and if nothing is found, a message will appear, or when the contact item is deleted and will be saved to the database.



```php
    #[Route("/add", name: "contact_add")]
    public function addContact(Request $request) {
        $contact = new Contact;
        $form = $this->createForm(ContactType::class, $contact);
        $form->handleRequest($request);
        $title = "Add new contact";
        if ($form->isSubmitted() && $form->isValid()) {
            $manager = $this->getDoctrine()->getManager();
            $manager->persist($contact);
            $manager->flush();
            $this->addFlash("Success","Add contact succeed !");
            return $this->redirectToRoute("contact_index");
        }
        return $this->renderForm("contact/add.html.twig",
        [
            'contactForm' => $form,
            'title' => $title
        ]);
    }
```

Figure 19: Contact Controller – Add

And the first addition #[Route("/add", name: "contact_add")] create my own form and call the controller, then check the contact to add to the database.

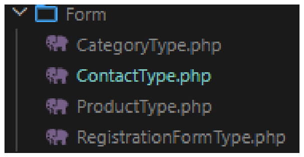


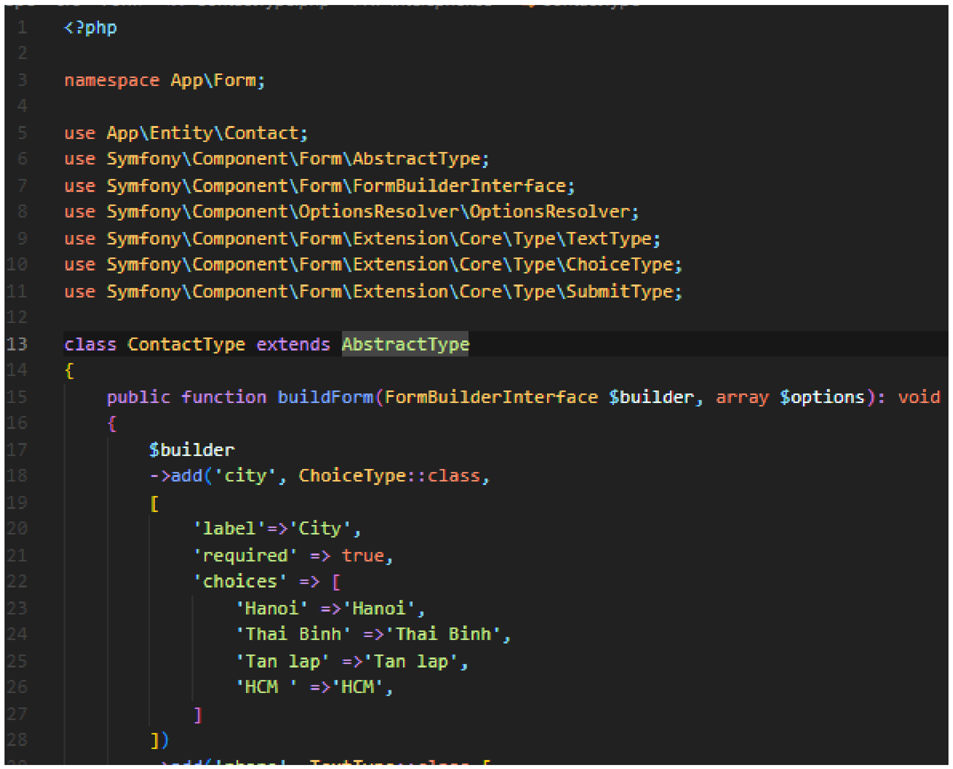
```php
    #[Route("/edit/{id}", name: "contact_edit")]
    public function editcontact(Request $request, ManagerRegistry $managerRegistry, $id) {
        $contact = $managerRegistry->getRepository(Contact::class)->find($id);
        $form = $this->createForm(ContactType::class, $contact);
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $manager = $managerRegistry->getManager();
            $manager->persist($contact);
            $manager->flush();
            $this->addFlash("Success","Edit contact succeed !");
            return $this->redirectToRoute("contact_index");
        }
        return $this->renderForm("contact/edit.html.twig",
        [
            'contactForm' => $form,
        ]);
    }
}
```

Figure 20: Contact Controller – Edit

To edit a contact, you'll need an ID and a form to fill out before saving the data to the database.

4. Form





```php
<?php

namespace App\Form;

use App\Entity\Contact;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class ContactType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
        ->add('city', ChoiceType::class,
        [
            'label'=>'City',
            'required' => true,
            'choices' => [
                'Hanoi' =>'Hanoi',
                'Thai Binh' =>'Thai Binh',
                'Tan lap' =>'Tan lap',
                'HCM ' =>'HCM',
            ]
        ])
```

Figure 21: Contact Type

In the form page i also added all the libraries I use in the contact type file on the left side "City" is the data from dabase assign "label"=> for it's called "city" , 'required' => true, 'choices' => [' địa chỉ'] như 'Tan Lap', 'Thai Binh',....

II. Web screenshots

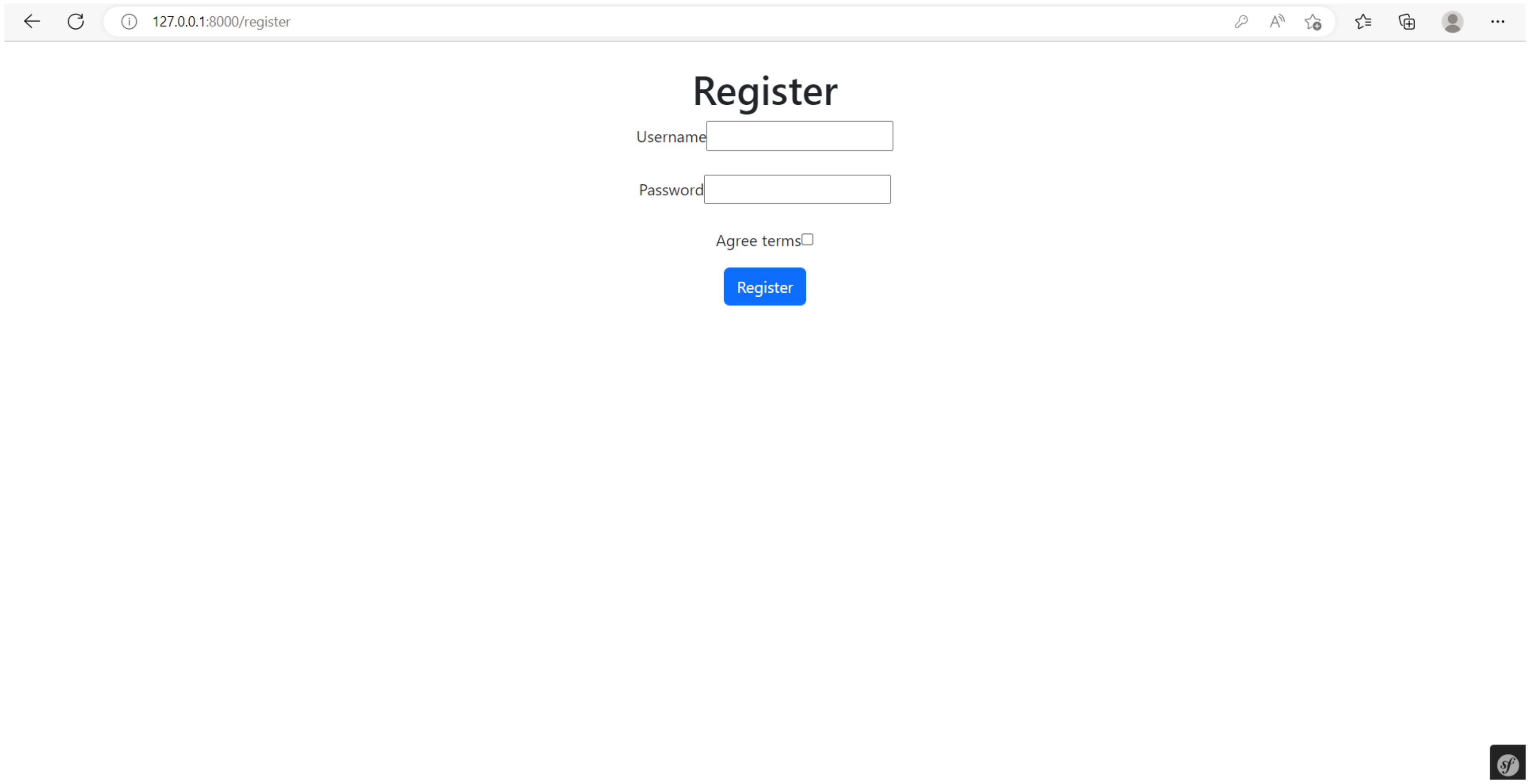


Figure 22: Register

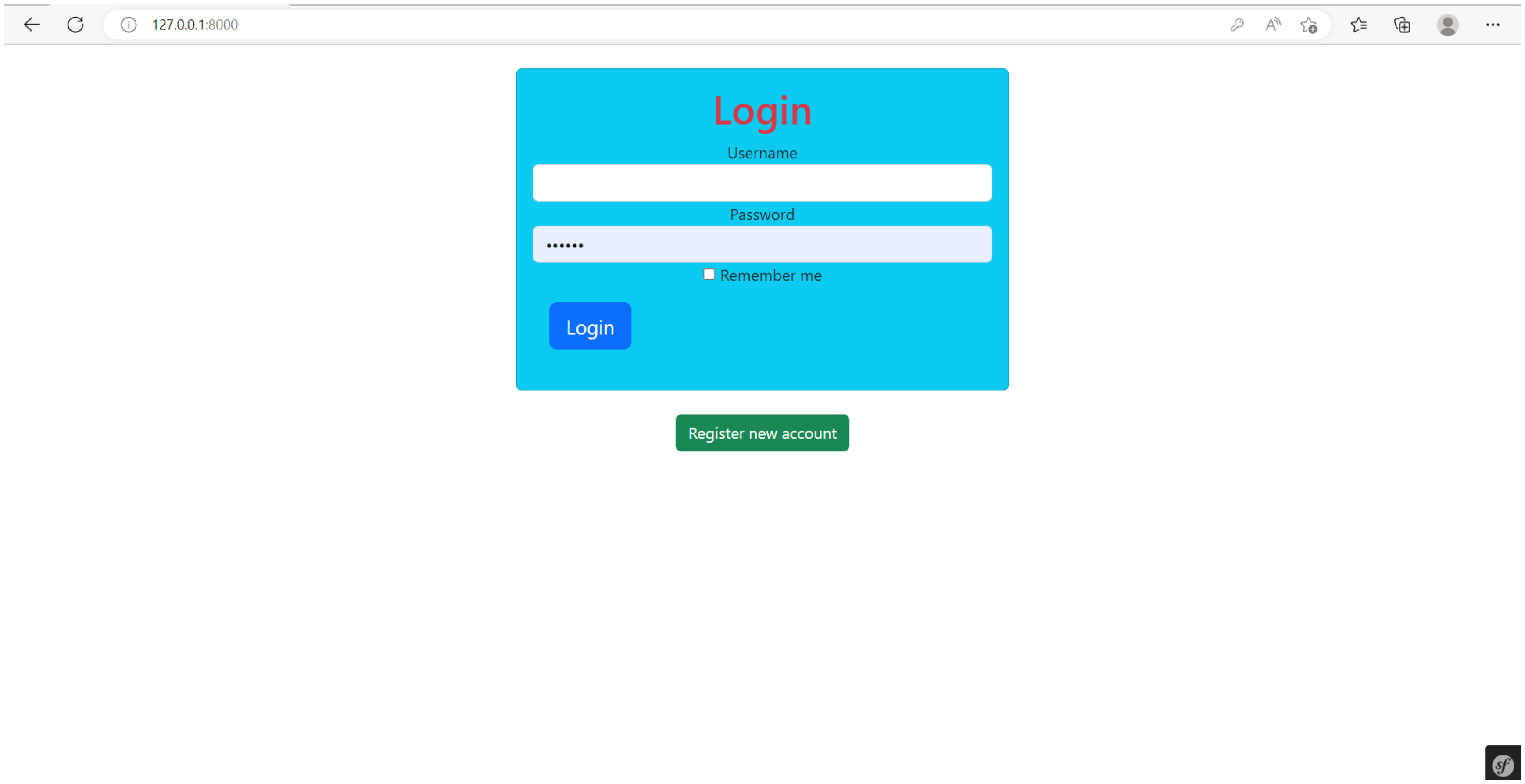


Figure 23: Login

Before logging in to any system when we do not have an account before, we must also register a new account and then log in.

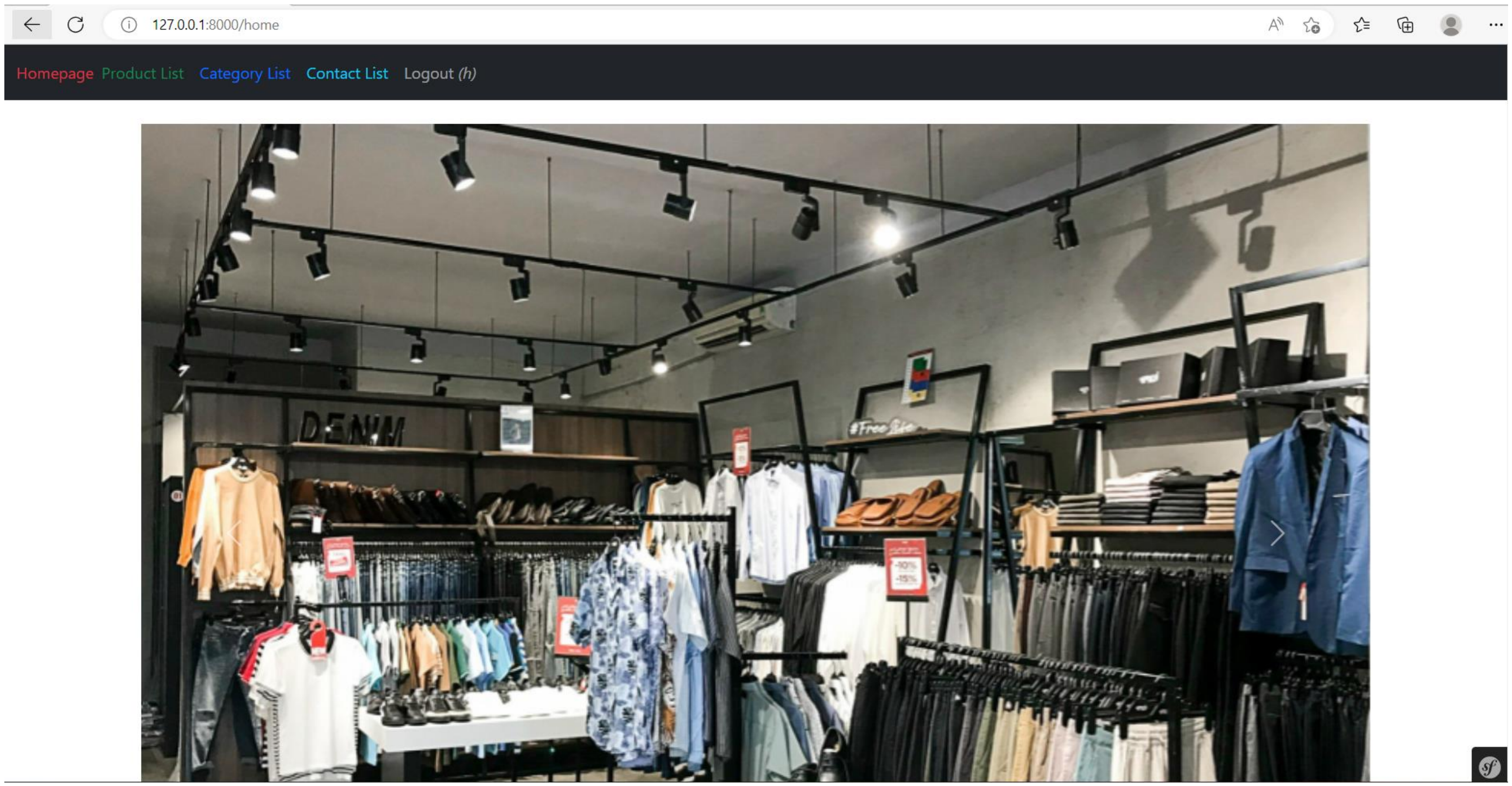
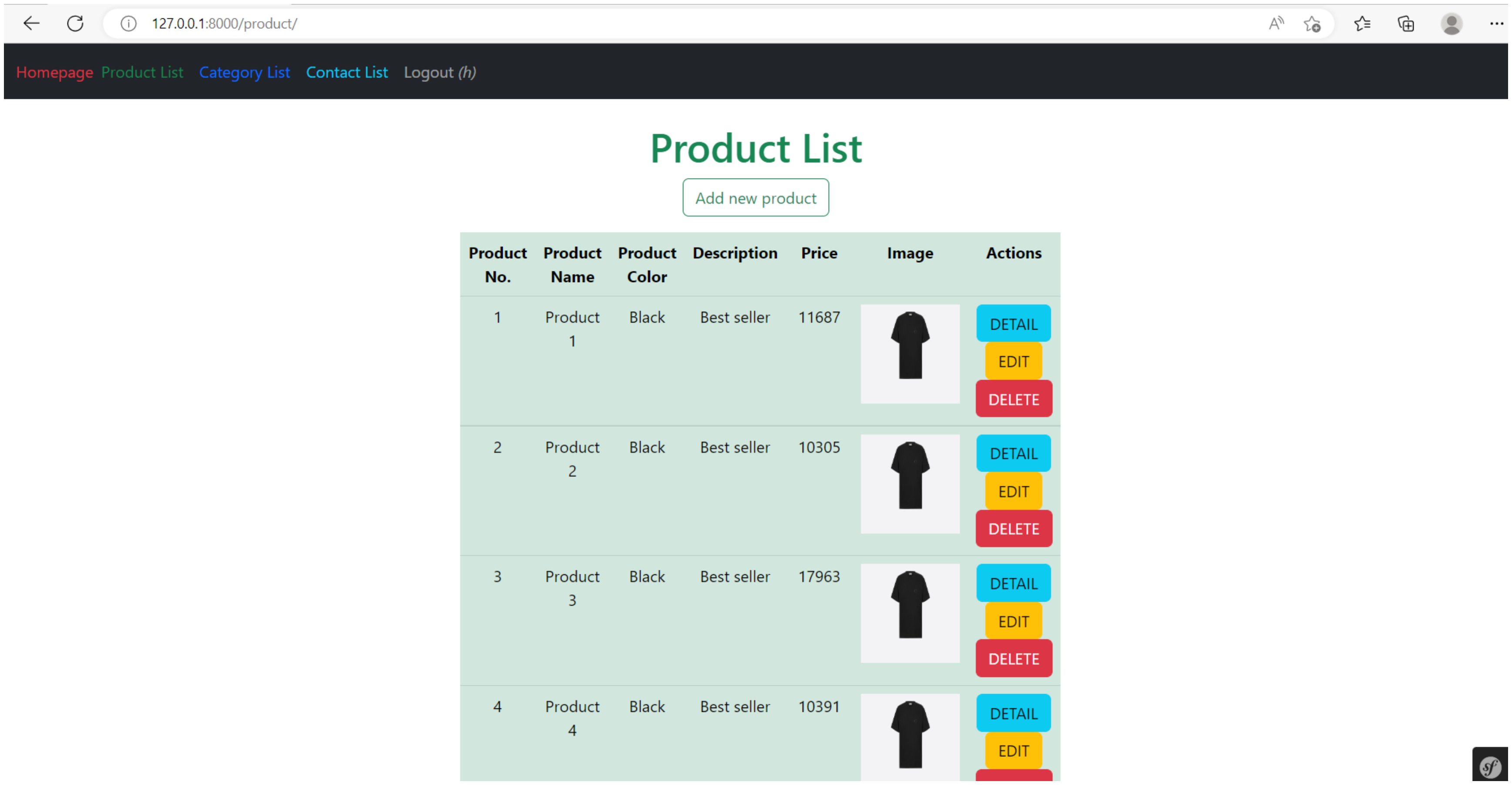


Figure 24: Home Page

This is the home page with images representing all the products and how to contact . If the user wants to see the product, then switch to the Product List to see more details.

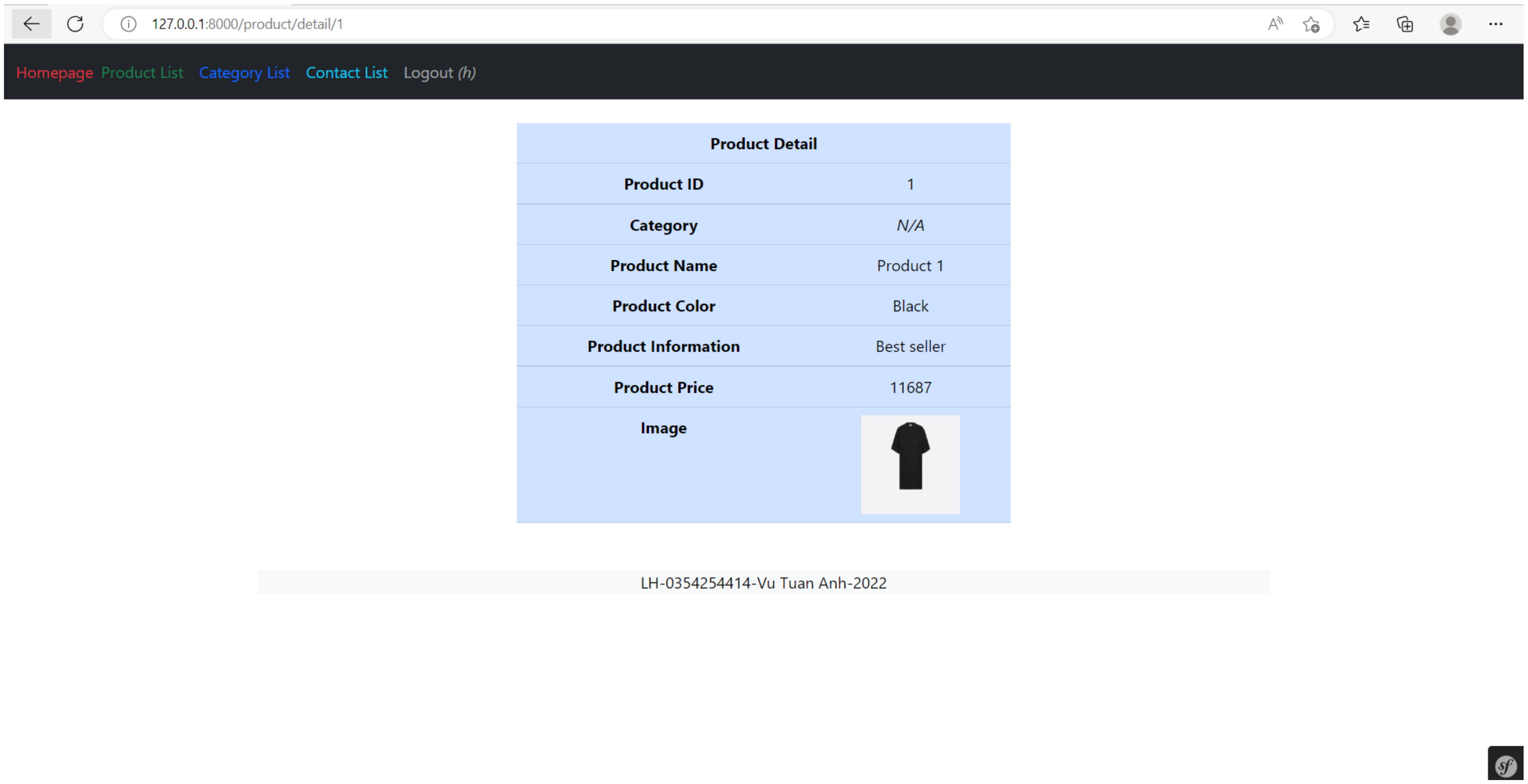


Product List

Product No.	Product Name	Product Color	Description	Price	Image	Actions
1	Product 1	Black	Best seller	11687		DETAIL EDIT DELETE
2	Product 2	Black	Best seller	10305		DETAIL EDIT DELETE
3	Product 3	Black	Best seller	17963		DETAIL EDIT DELETE
4	Product 4	Black	Best seller	10391		DETAIL EDIT

Figure 25: Product List

This page summarizes all product models.

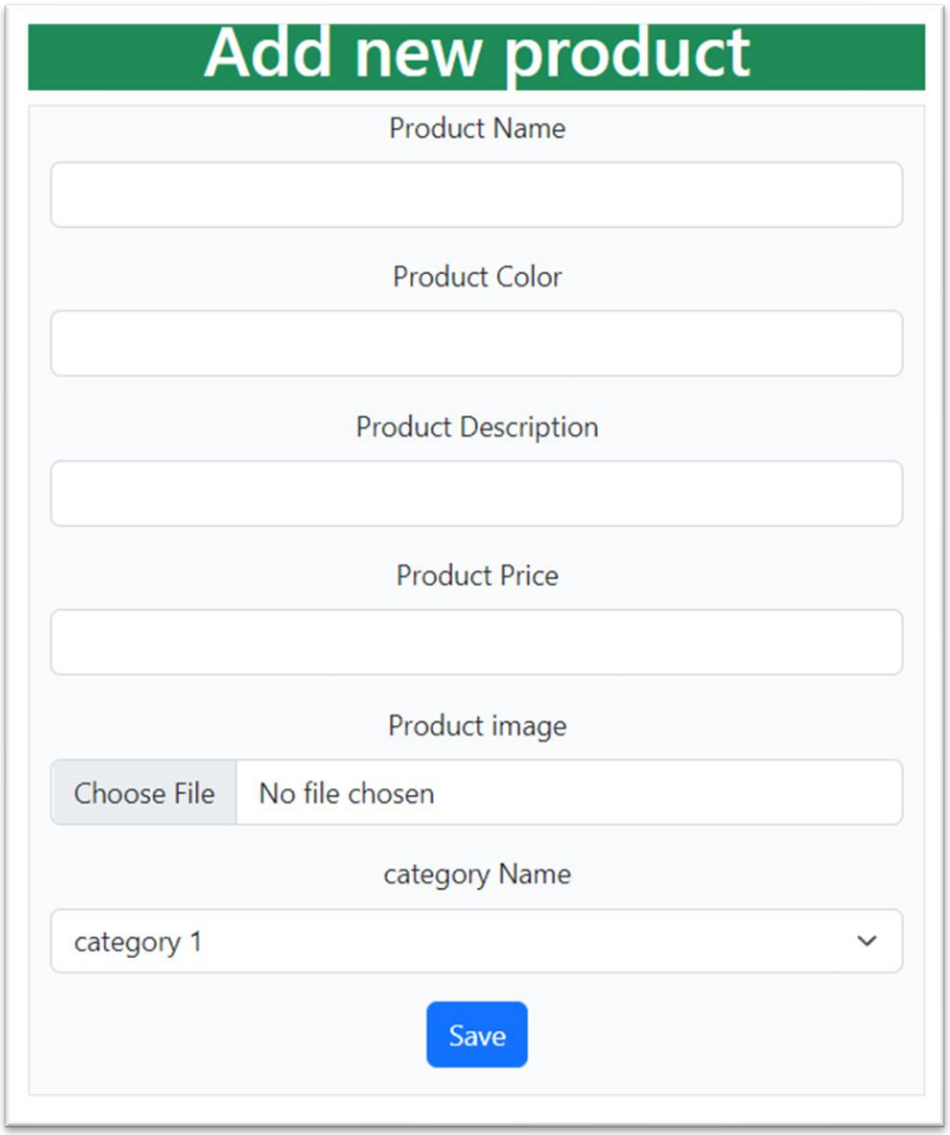


Product Detail	
Product ID	1
Category	*N/A*
Product Name	Product 1
Product Color	Black
Product Information	Best seller
Product Price	11687
Image	

LH-0354254414-Vu Tuan Anh-2022

Figure 26: Product Detail

At the product detail page will appear full information about the product such as color, price, image, ... so that customers have a best choice.


Add new product

Product Name

Product Color

Product Description

Product Price

Product image

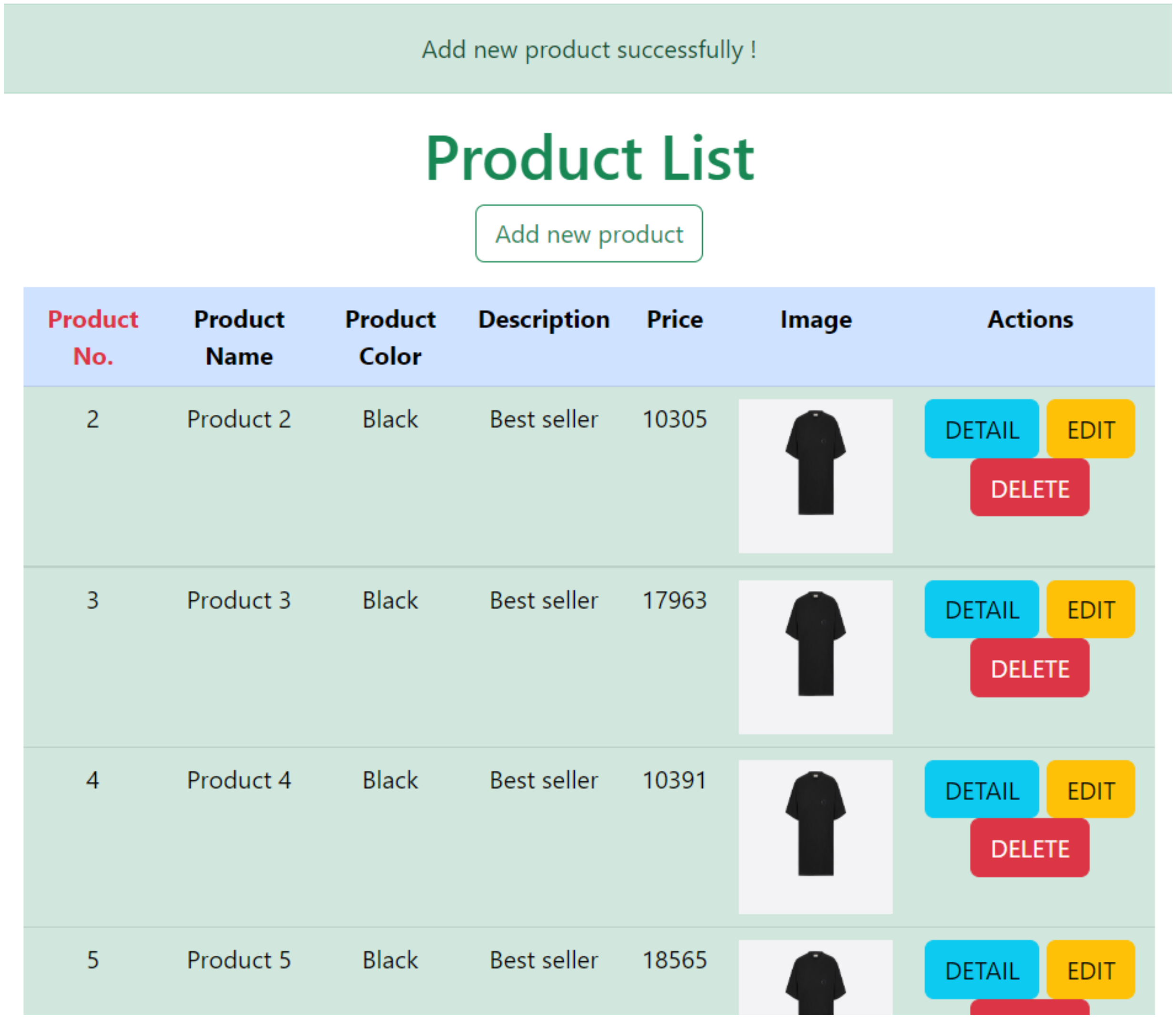
Choose File No file chosen

category Name

category 1

Figure 27: Add product

When the add button is clicked, it will show a copy like the picture above for users and admins to add new products.



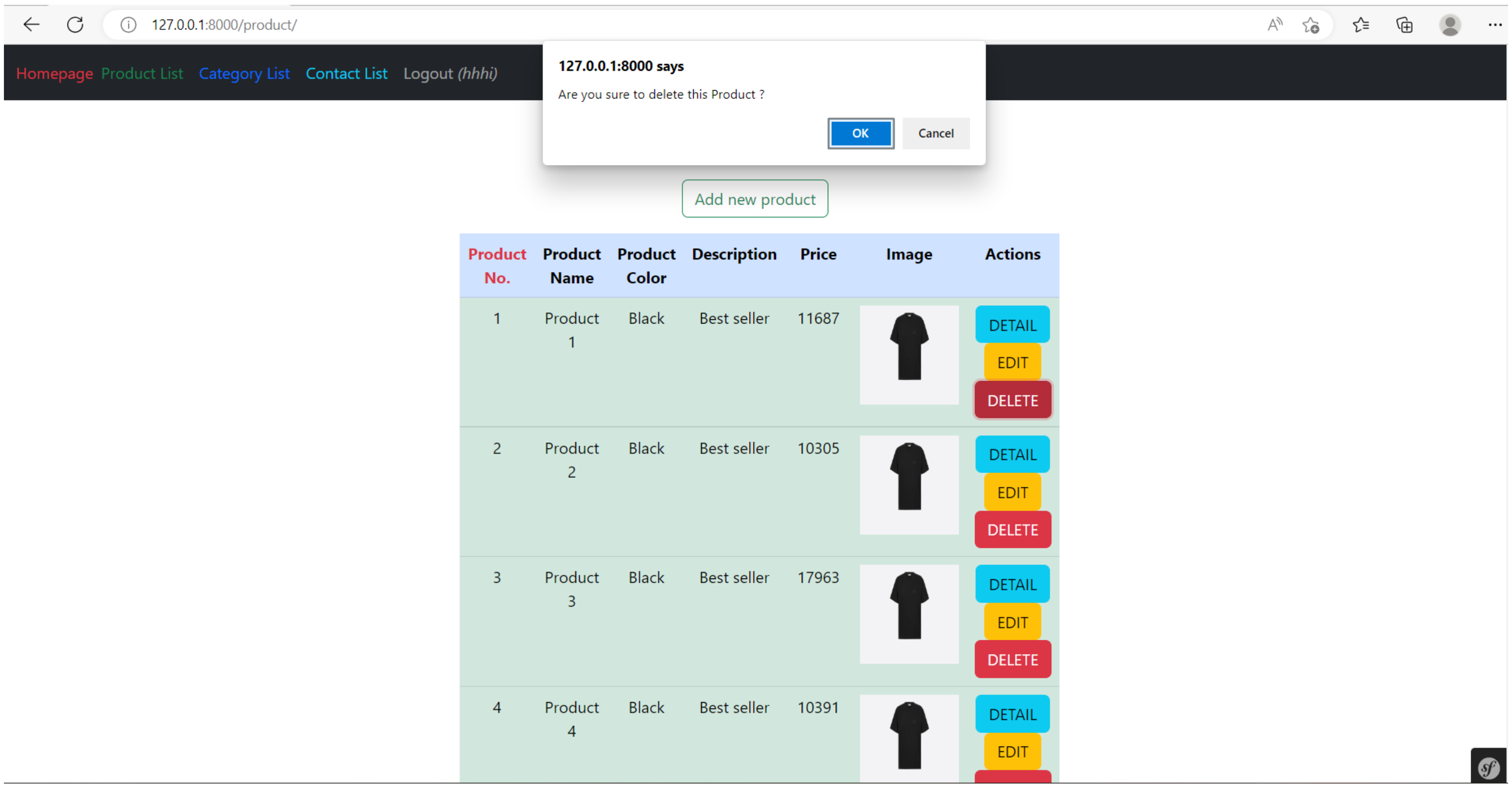
Add new product successfully !

Product List

Product No.	Product Name	Product Color	Description	Price	Image	Actions
2	Product 2	Black	Best seller	10305		DETAIL EDIT DELETE
3	Product 3	Black	Best seller	17963		DETAIL EDIT DELETE
4	Product 4	Black	Best seller	10391		DETAIL EDIT DELETE
5	Product 5	Black	Best seller	18565		DETAIL EDIT

Figure 28: Add success

A new product will be added to the table and a message indicating a successful add will be shown.



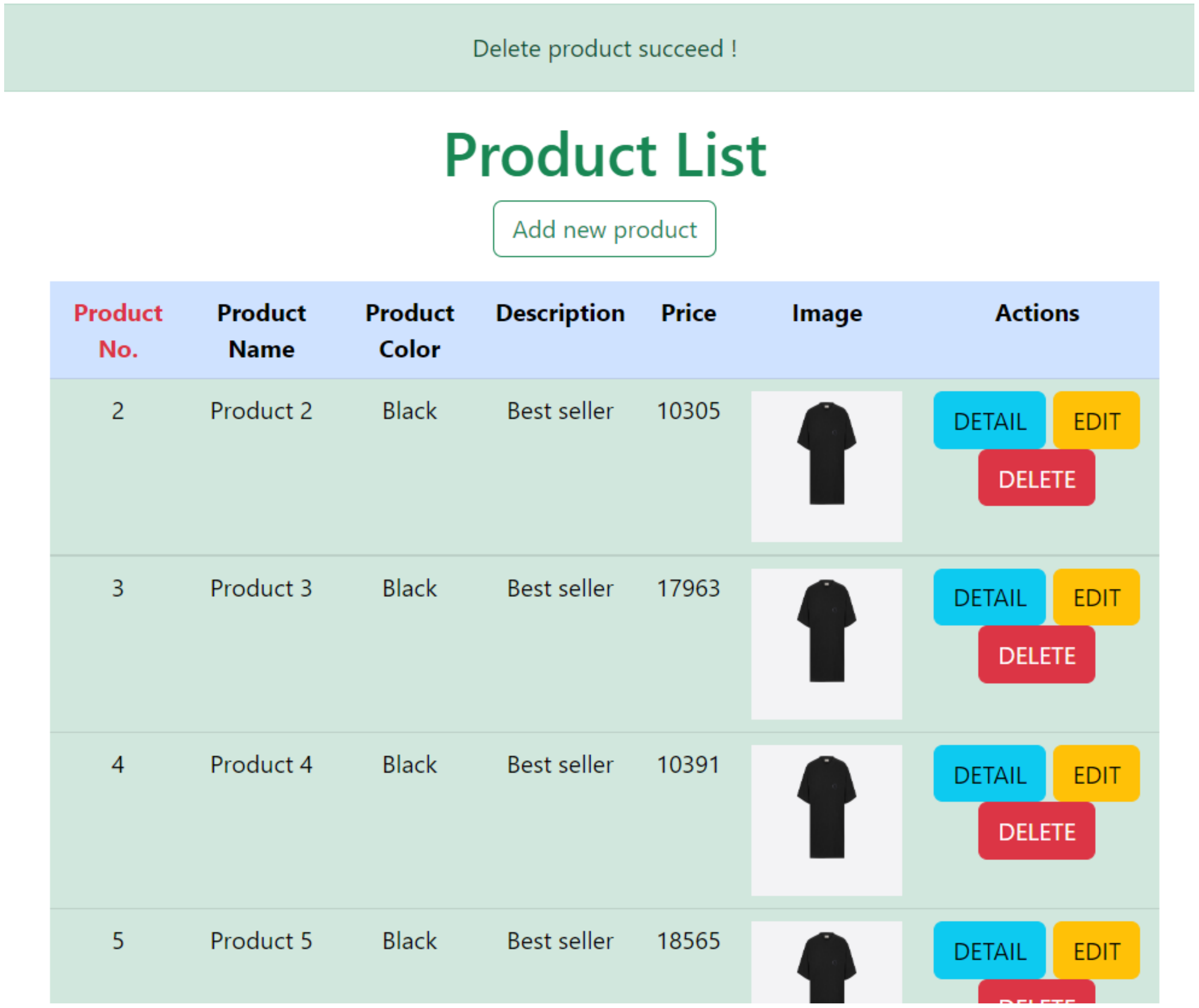
127.0.0.1:8000 says

Are you sure to delete this Product ?

Product No.	Product Name	Product Color	Description	Price	Image	Actions
1	Product 1	Black	Best seller	11687		DETAIL EDIT DELETE
2	Product 2	Black	Best seller	10305		DETAIL EDIT DELETE
3	Product 3	Black	Best seller	17963		DETAIL EDIT DELETE
4	Product 4	Black	Best seller	10391		DETAIL EDIT

Figure 29: Delete product

When user or admin want to delete press delete button and it will ask you if you want to delete or not.

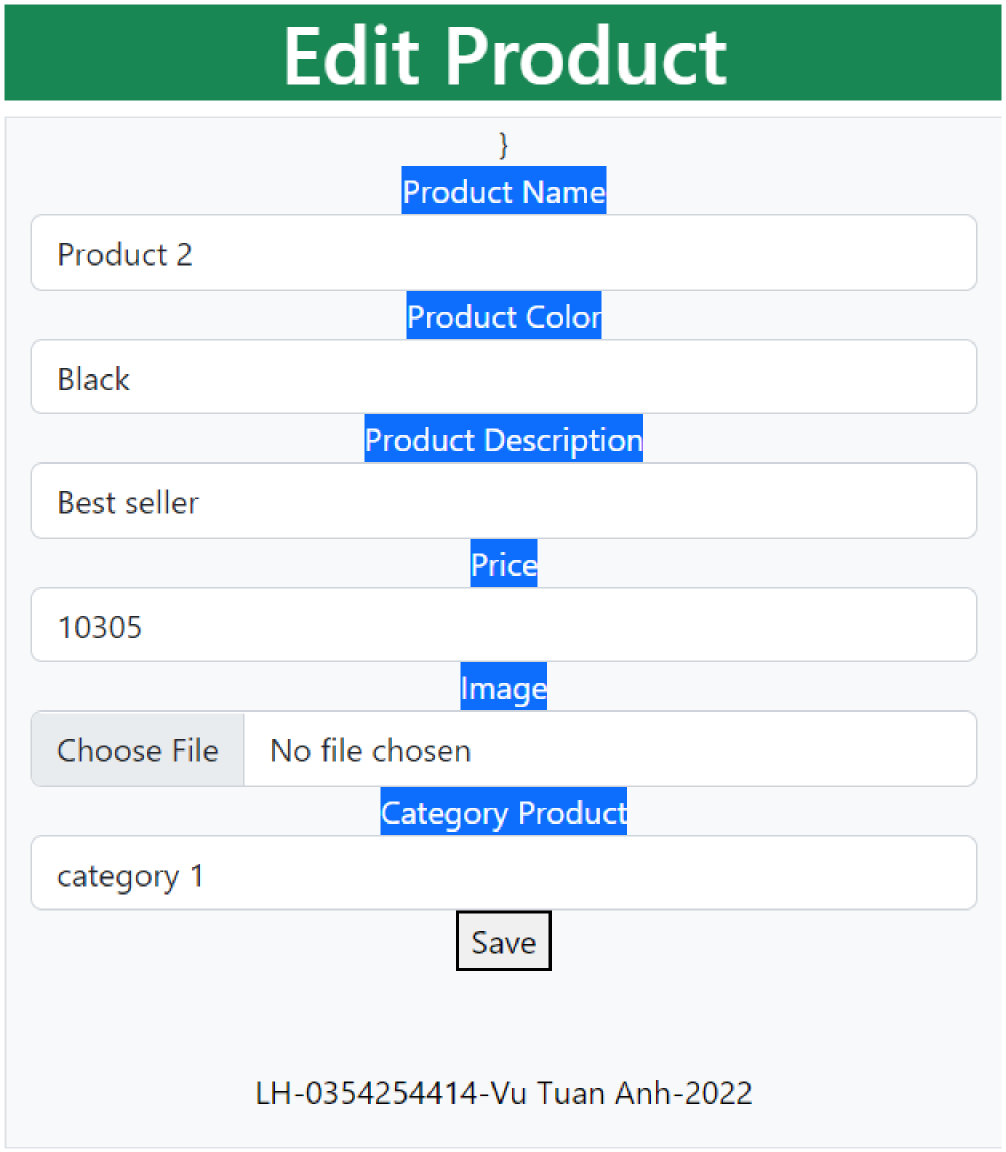

Delete product succeed !

Product List

Product No.	Product Name	Product Color	Description	Price	Image	Actions
2	Product 2	Black	Best seller	10305		DETAIL EDIT DELETE
3	Product 3	Black	Best seller	17963		DETAIL EDIT DELETE
4	Product 4	Black	Best seller	10391		DETAIL EDIT DELETE
5	Product 5	Black	Best seller	18565		DETAIL EDIT

Figure 30: Delete success

After clicking yes, a successful deletion message will appear.



Edit Product

}

Product Name

Product 2

Product Color

Black

Product Description

Best seller

Price

10305

Image

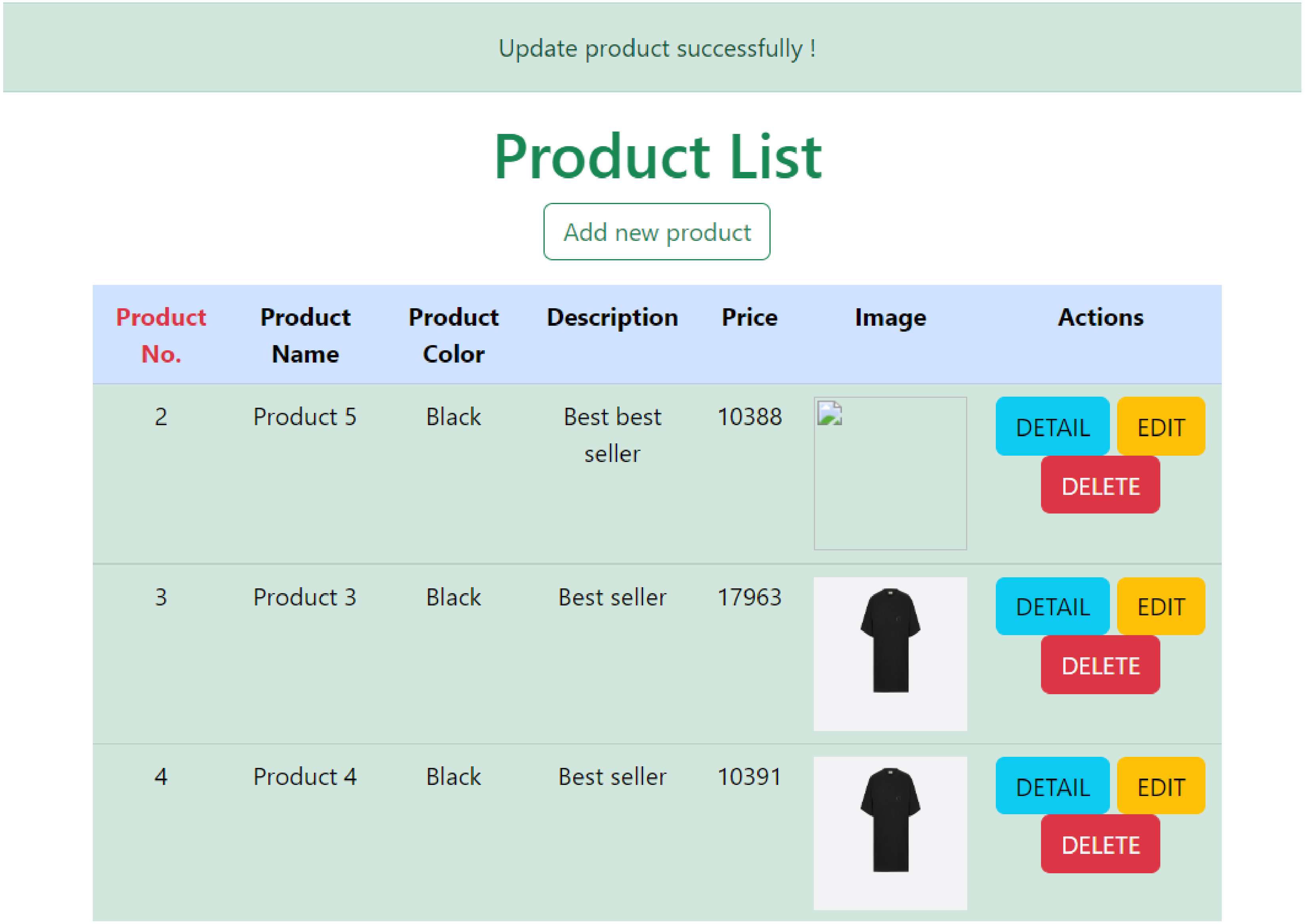
Category Product

category 1

LH-0354254414-Vu Tuan Anh-2022

Figure 31: Edit product

When a user or administrator wants to edit a product in the table, they should click the edit button, which displays the option to edit production.



Update product successfully !

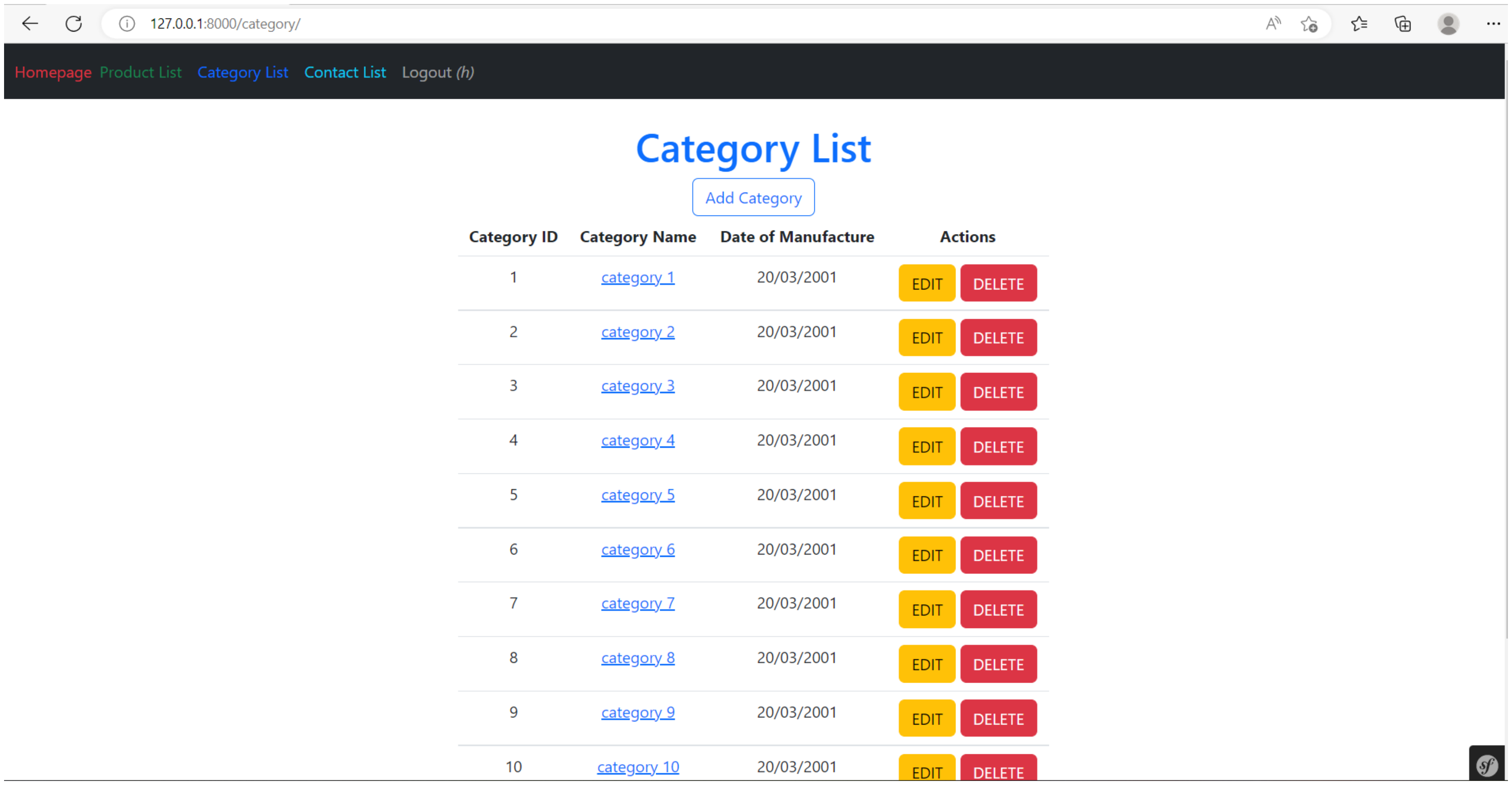
Product List

Add new product

Product No.	Product Name	Product Color	Description	Price	Image	Actions
2	Product 5	Black	Best best seller	10388		DETAIL EDIT DELETE
3	Product 3	Black	Best seller	17963		DETAIL EDIT DELETE
4	Product 4	Black	Best seller	10391		DETAIL EDIT DELETE

Figure 32: Update success

Same Product



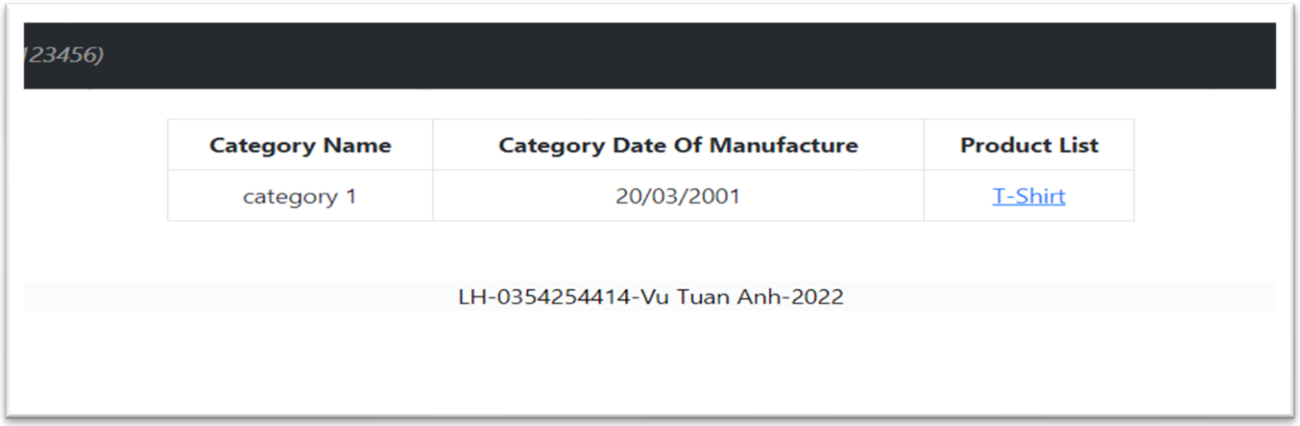
127.0.0.1:8000/category/

Category List

Category ID	Category Name	Date of Manufacture	Actions
1	category 1	20/03/2001	EDIT DELETE
2	category 2	20/03/2001	EDIT DELETE
3	category 3	20/03/2001	EDIT DELETE
4	category 4	20/03/2001	EDIT DELETE
5	category 5	20/03/2001	EDIT DELETE
6	category 6	20/03/2001	EDIT DELETE
7	category 7	20/03/2001	EDIT DELETE
8	category 8	20/03/2001	EDIT DELETE
9	category 9	20/03/2001	EDIT DELETE
10	category 10	20/03/2001	EDIT DELETE

Figure 33: Category page

In a category, admin can view, edit or delete products.

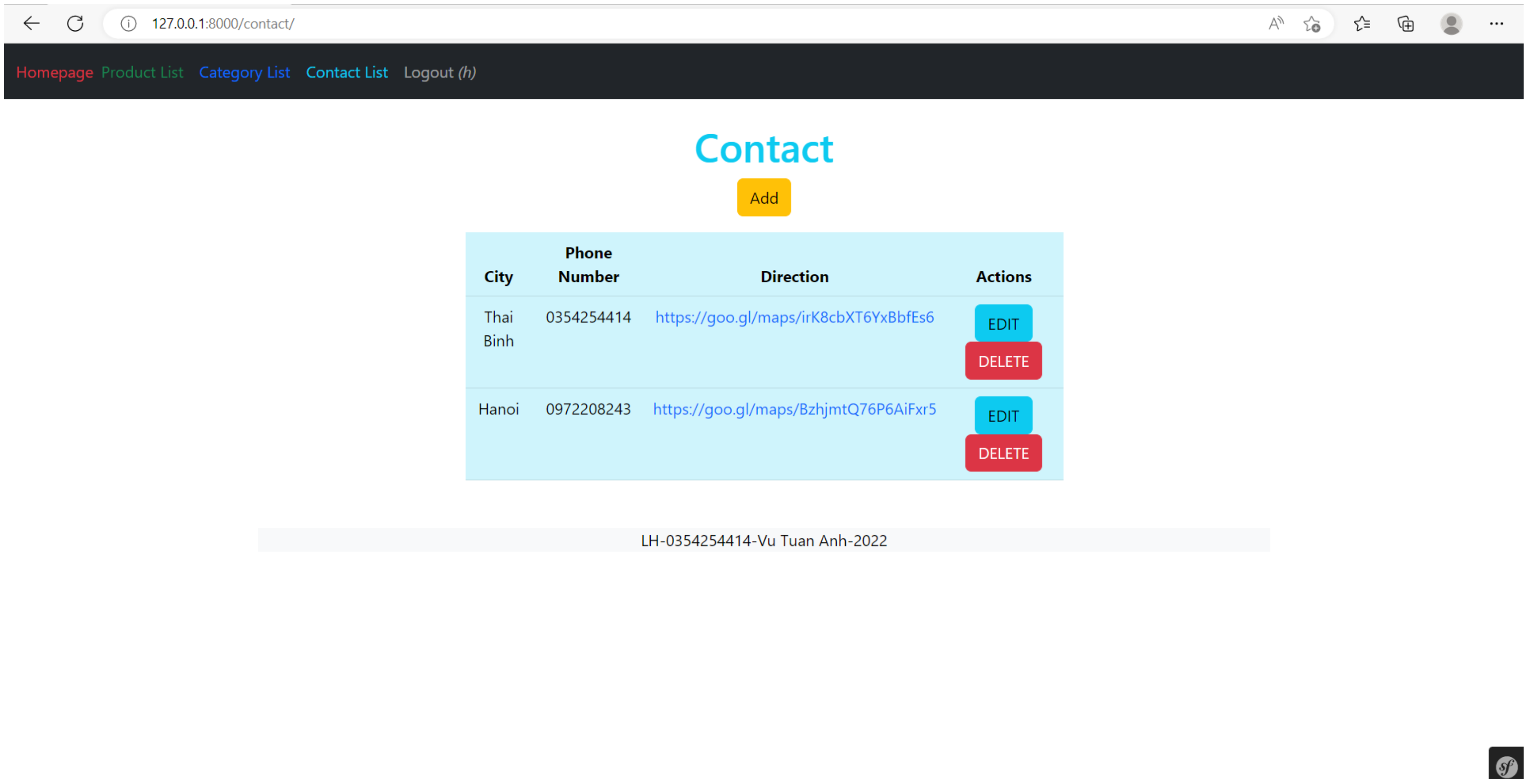


123456)

Category Name	Category Date Of Manufacture	Product List
category 1	20/03/2001	T-Shirt

LH-0354254414-Vu Tuan Anh-2022

Figure 34: Category Detail



127.0.0.1:8000/contact/

Homepage Product List Category List Contact List Logout *(h)*

Contact

Add

City	Phone Number	Direction	Actions
Thai Binh	0354254414	https://goo.gl/maps/irK8cbXT6YxBbfEs6	EDIT DELETE
Hanoi	0972208243	https://goo.gl/maps/BzhjmtQ76P6AiFxr5	EDIT DELETE

LH-0354254414-Vu Tuan Anh-2022

Figure 35: Contact Detail

In the contact section, the admin can view, edit or delete the product, and the customer can view the contact information.

Part 4 - Conclusion (INDIVIDUAL)

I. Advantages of website

With clear design and development, the website contains the majority of the necessary functions. Basic functions such as create, read, update, and delete are used throughout the project.

II. Disadvantages of website

The site still lacks the functionality required to be considered a complete website. There is no shopping cart, search, or many other functions,...
The site's user interface isn't particularly impressive. More implementation is required for the website, particularly in the user interface. Customers will not stay on a website with poor content and design for long. The project does not have an API to test the functionality.

III. Lesson learnt

The goal of this exercise is to learn about Symfony and the MVC pattern. How to organize the website into functions, modules, views, and controllers in order to make it easier to manage. I finished the task I also learned a lot about MVC and API. To me, completing a task is a wonderful thing, but in order to complete a task better, it must be based on my own awareness, not on the awareness of others.

IV. Future improvements

In the future, I hope to use the front-end proficiently to create a better interface to make the site more user-friendly. And I will learn how to use API to test functions. I will explore and learn to secure the website in the best way.

Appendix: (GROUP)

Name	**Role**
Vũ Tuấn Anh	-Design entities relationship. -Create Product. -Frontend Backend Product, Category. -Create controller Product, category.
Bùi Hương Linh	-User entity. -Login/register. -Home page. -Create Controller Contact.

Github link:

https://github.com/anhvt196044/Asm-Nohope