ASM 1st Semi Project Duc Anh BH00056

Introduction to C programming language (Trường Cao đẳng Thực hành FPT)

ASSIGNMENT FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Semi Project		
Submission date	8/04/2023	**Date Received 1st submission**	12/04/2023
Re-submission Date		**Date Received 2nd submission**	
Student Name	Mai Duc Anh	**Student ID**	BH00056
Class	IT0501	**Assessor name**	Ngo Thi Mai Loan
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	

Grade

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

TABLE OF CONTENT

I. User Requirements

1. Requirements, both functional and non-functional

1.1 Functional Prerequisites

1.1.1. An Admin role

- Can access the system via the application's initial page.
- New user accounts can be created, edited, or deleted.
- Hotels can be created, updated, and deleted.
- Can add/update/delete Hotels
- Order administration

1.1.2. A Visitor (Not Registered on the Website)

- Can Create an account
- Products are available for viewing.
- Can look for hotels, room

1.1.3. A Member (Registered on the Website)

- Can log in
- Can logout
- Can view hotels, and rooms
- Can search hotels
- Can browse rooms
- Can change password and information
- Check-out and booking
- Orders are allowed

1.2. Non-Functional Requirements

1.2.1 Operational

On platforms like Google Chrome, Firefox, Microsoft Edge, and Coc Coc, it can be used right away.

The platforms can be used on Windows, MacOS, Kali, Android, and iOS as long as the devices are connected and have a browser. The screen will display in big, middle, and tiny versions to fit any device. The system was created using the PHP programming language and the MySQL database. only on the web.

- Google Chrome / Firefox / Microsoft Edge /…
- MacOS / Android / Window / …

1.2.2. Performance

The maximum number of connections and concurrent users that the website can support is 1000. With up to 1 ms of latency, the speed will be improved to give the customer the best experience available.

1.2.3. Security

The system's security will be provided by Cloudflare, and login options include HTTPS/SSH and MFA. On this website platform, all user info will be completely safe and private. The entire system will be upgraded continuously every week from 2:00 to 4:00 to ensure there are no security flaws. During the upgrade period, users can still view goods as usual, but they are unable to place purchases or make payments.

- HTTPS/SSH
- MFA
- Cloudflare

1.2.4 Political Culture

English is the system's main tongue. However, the Vietnamese translation method will be supported by a charity.

- Currency used: Vietnam Dong
- Payment: VISA/Momo/Mastercard (use QR Code)
- Website policies will be updated not based on sexist, racist rules.
- Color: #303030 - #ff5f17 - #f5f5f5
- Licensed trademarks

Use-Case diagram

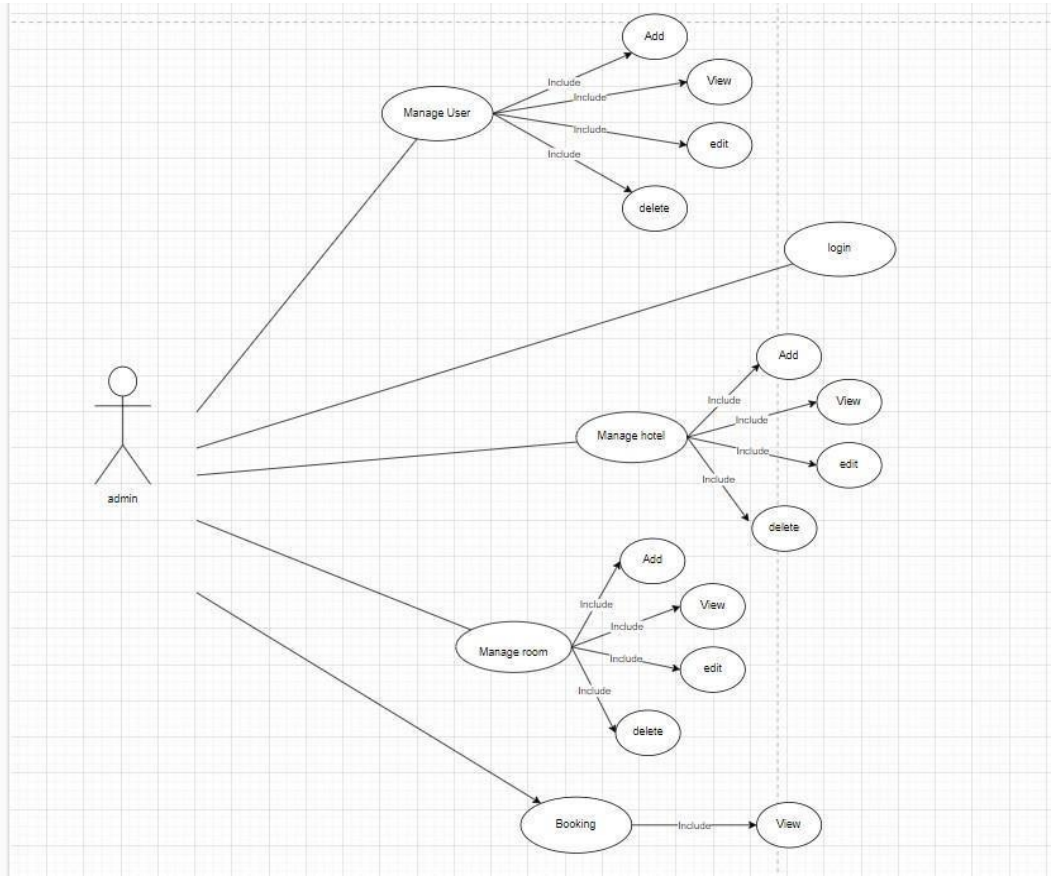

Figure 1: Use case

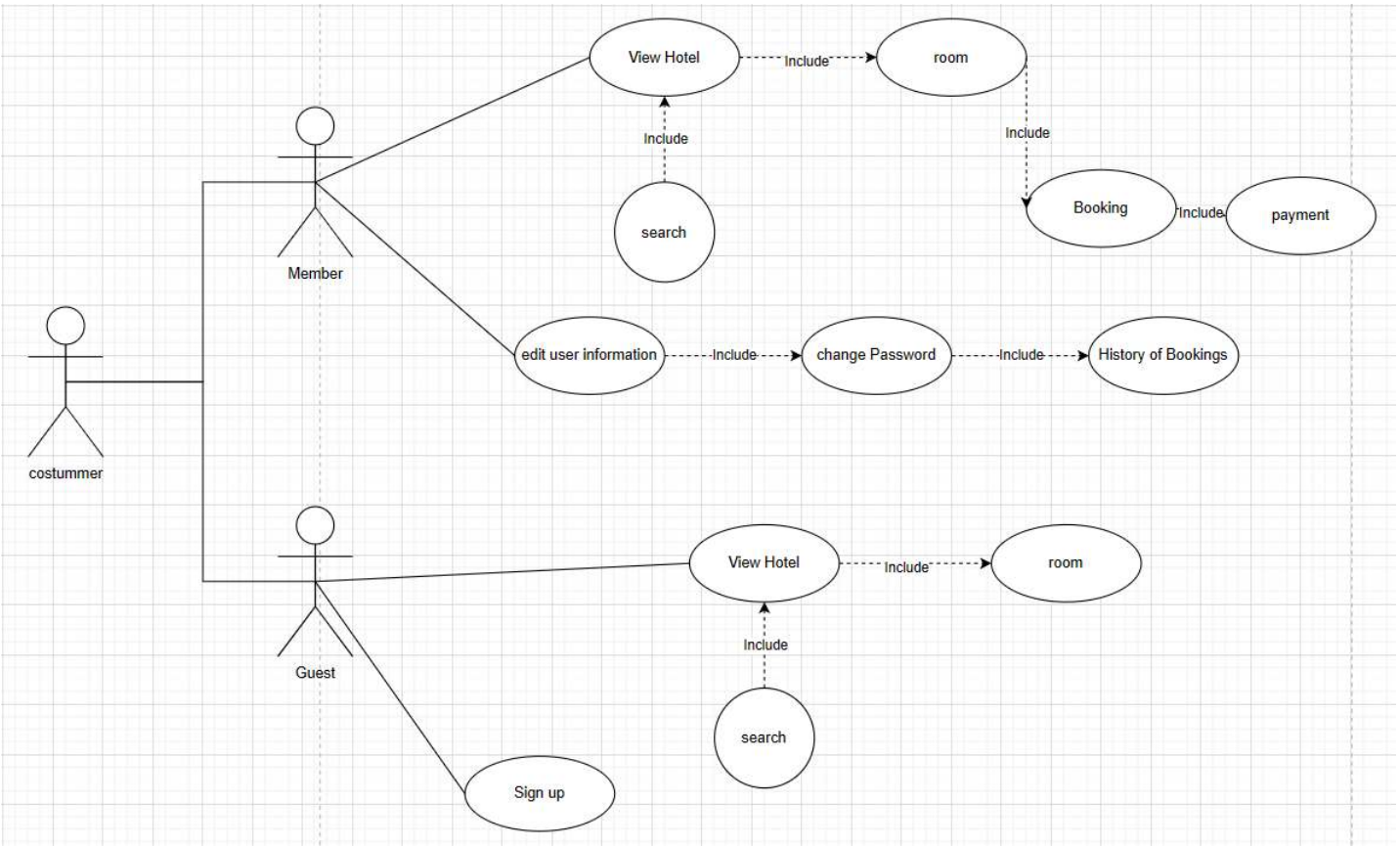

Figure 2: Use case

#UC01: Logout

Name	Log out	**Code**	UC01
Description	Permit the actor to check out of the server with the account.		
Actor	Admin/ Customer	**Trigger**	Actor clicks the logout icon.
Pre-condition	Actor entered the system effectively.		
Postcondition	Go to the login page.		

Activities

Actor	System
Main Flow: Log out successfully	

1	On the profile administration screen, the actor selects the Logout option.		
		2	Go to the login tab after confirming deactivation.

#UC02: Edit Account/Username

Name	Edit Username/Account	**Code**	UC02
Description	Permit the performer to make account changes. (eg: username, ...)		
Actor	Admin/ Customer	**Trigger**	Actor hits the finished trigger
Pre-condition	Actor has successfully edited the account		
Post condition	Go to admin manage user page.		

Activities

Actor		**System**	
Main Flow: Edit User Successfully			
1	On the profile administration screen, the actor selects the Logout option.		
		2	Go to the login tab after confirming deactivation.

#UC03: View hotels

Name	View hotels	**Code**	UC04
Description	Permit character to see product information		
Actor	Admin/ Customer	**Trigger**	Actor clicks on the product
Pre-condition			
Post condition	Go to product detail page		

Activities

Actor		System	
Main Flow: View hotels			
1	select "homepage" in the menu.		
		2	Load the item on the displayed merchandise screen.
3	Click on the product you want to see details		
		4	the product information page should load.

#UC05: View Product Detail

Name	View Product Detail	**Code**	UC05
Description	Permit the performer to look at the system's merchandise details.		
Actor	Admin/ Customer	**Trigger**	Actor presses buy now button
Pre-condition			
Post condition	Go to Product Detail pages		

Activities

Actor		System	
Main Flow: Log out successfully			
1	Actor clicks on the manage product button on the admin page		
		2	Fill in full information
3	Click on the button to create		
		4	Activate the manage merchandise tab.

#UC04: View Product Detail

Name	View Product Detail	**Code**	UC04
Description	Permit the performer to look at the system's merchandise details.		
Actor	Admin/ Customer	**Trigger**	Actor presses buy now button
Pre-condition	Go to Product Detail pages		

Activities

Actor		**System**	
Main Flow: View product detail			
1	Actor selects an item from the store page.		
		2	Keep the merchandise open on the displayed product detail screen.
3	Click on the button to add to cart		
		4	Load into cart page

#UC05: Create A hotels

Name	Create a hotels	**Code**	UC05
Description	Allow the actor to create a new hotels		
Actor	Admin	**Trigger**	Actor presses Create button
Pre-condition	Actor has successfully logged into the system.		
Postcondition	Go to manage product page.		

Activities

Actor	**System**
Main Flow: Log out successfully	

1	Actor clicks on the manage product button on the admin page		
		2	Fill in full information
3	Click on the button to create		
		4	Activate the manage merchandise tab.

II. System Designs

1. Sitemap

Link Sitemap: https://www.gloomaps.com/GeR2qNQxV9

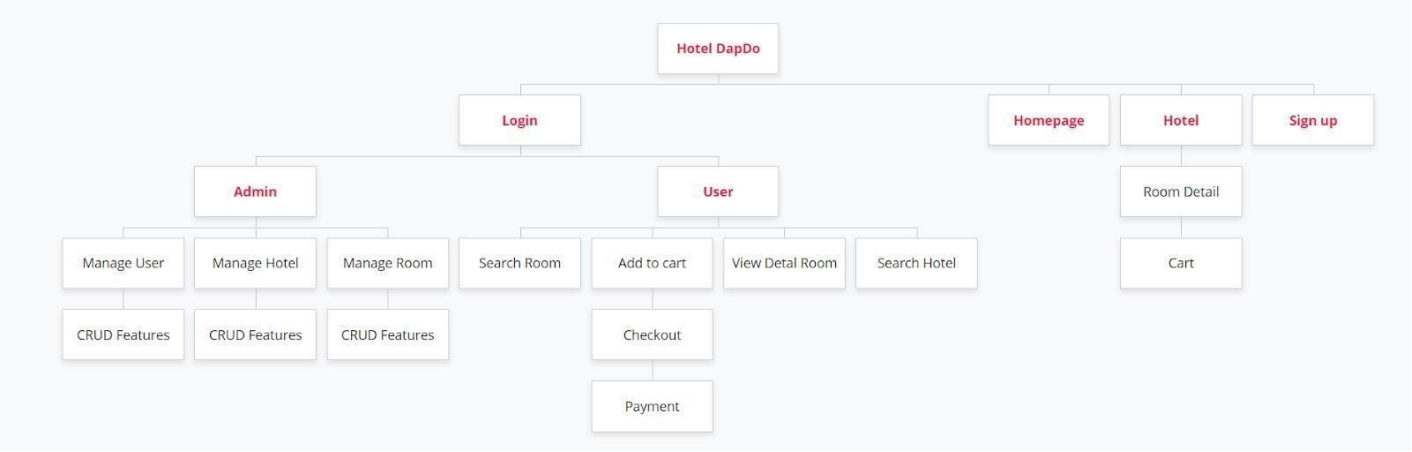

Figure 3: Site Map

2. Entity Relationship Diagram (ERD)

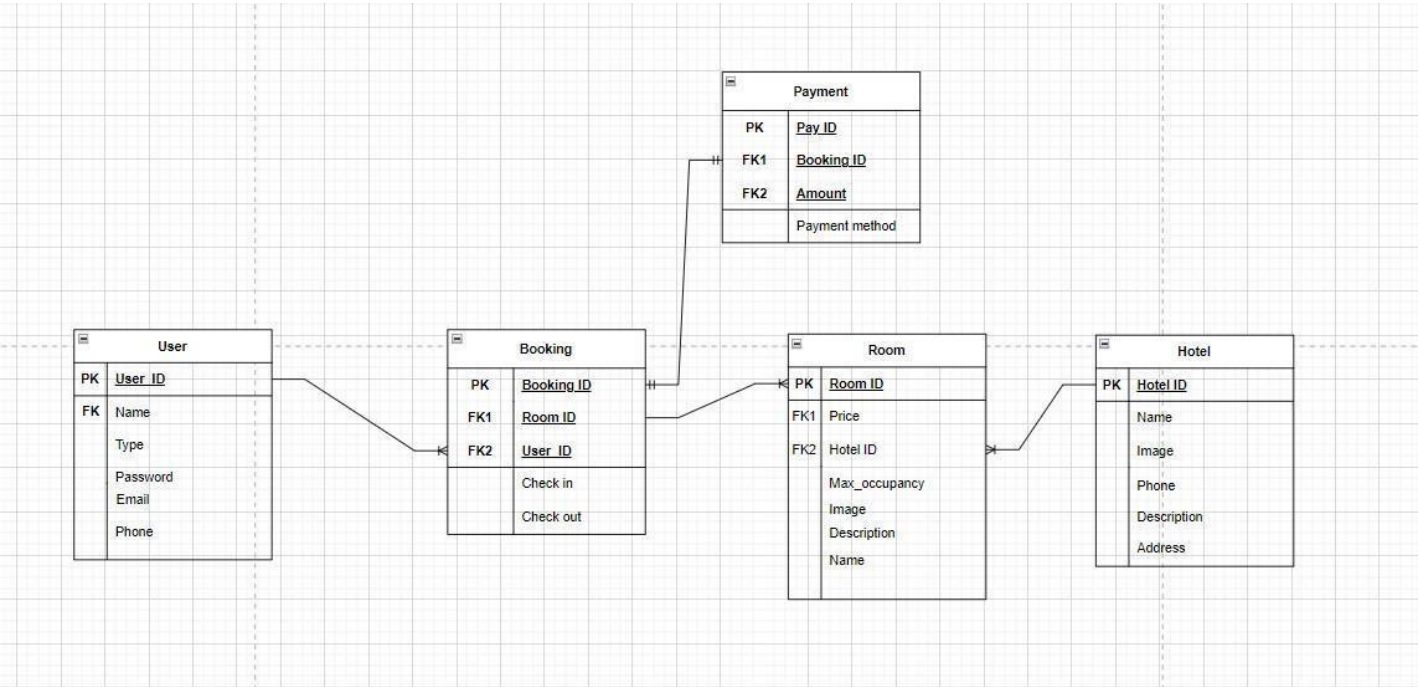

Figure 4: ERD

2.1. ERD Documentation

2.1.1. Users

Field Name	Data Type	Constraint	Explanation
Id	Bigint(20)unsigned	Primary Key	ID of user
Name	Varchar(255)	Not Null	Full name of a user
Email	Varchar(255)	Primary Key	Email of a user
PhoneNumber	Varchar(255)	Not Null	Phone number of a user
Type	Varchar(255)	Not Null	Type of room
Password	Varchar(255)	Not Null	Password of a user account. It is used with a username to log in to the system

2.1.2. Room

Field Name	Data Type	Constraint	Explanation
Id	Bigint(20)unsigned	Primary Key	ID of room
Name	Varchar(255)	Not Null	Full name of a user
Description	Varchar(500)	Not Null	Description for room
Price	decimal(22.2)	Not Null	Price of room
Img	Varchar(255)	Not Null	Image of room
Hotel_id	Bigint(20)unsigned		ID of hotel

2.1.3. *Bookings*

Field Name	Data Type	Constraint	Explanation
Id	Bigint(20)unsigned	Primary Key	ID of booking
User_id	Bigint(20)unsigned	Not Null	ID of user
Room_id	Bigint(20)unsigned	Not Null	ID of room
Checkin_date	date	Not Null	Day of customer booking
Checkout_date	date	Not Null	Day customer check out

2.1.4. *Hotels*

Field Name	Data Type	Constraint	Explanation
Id	Bigint(20)unsigned	Primary Key	ID of hotel
Name	Varchar(255)	Not Null	Full name of a user
Img	Varchar(255)	Not Null	Image of hotel
Phone	Varchar(255)	Not Null	Phone number of a user
Des	Varchar(255)	Not Null	Description of hotel
Address	Varchar(255)	Not Null	Address of hotel

2.1.5. *Payment*

Field Name	Data Type	Constraint	Explanation
Id	Bigint(20)unsigned	Primary Key	ID of payment
Booking_id	Bigint(20)unsigned	Not Null	ID of booking
Amount	Decimal(8,2)	Varchar(255)	Number of people in the room
Payment_method	Varchar(255)	Varchar(255)	Payment methods

3. Wiframes

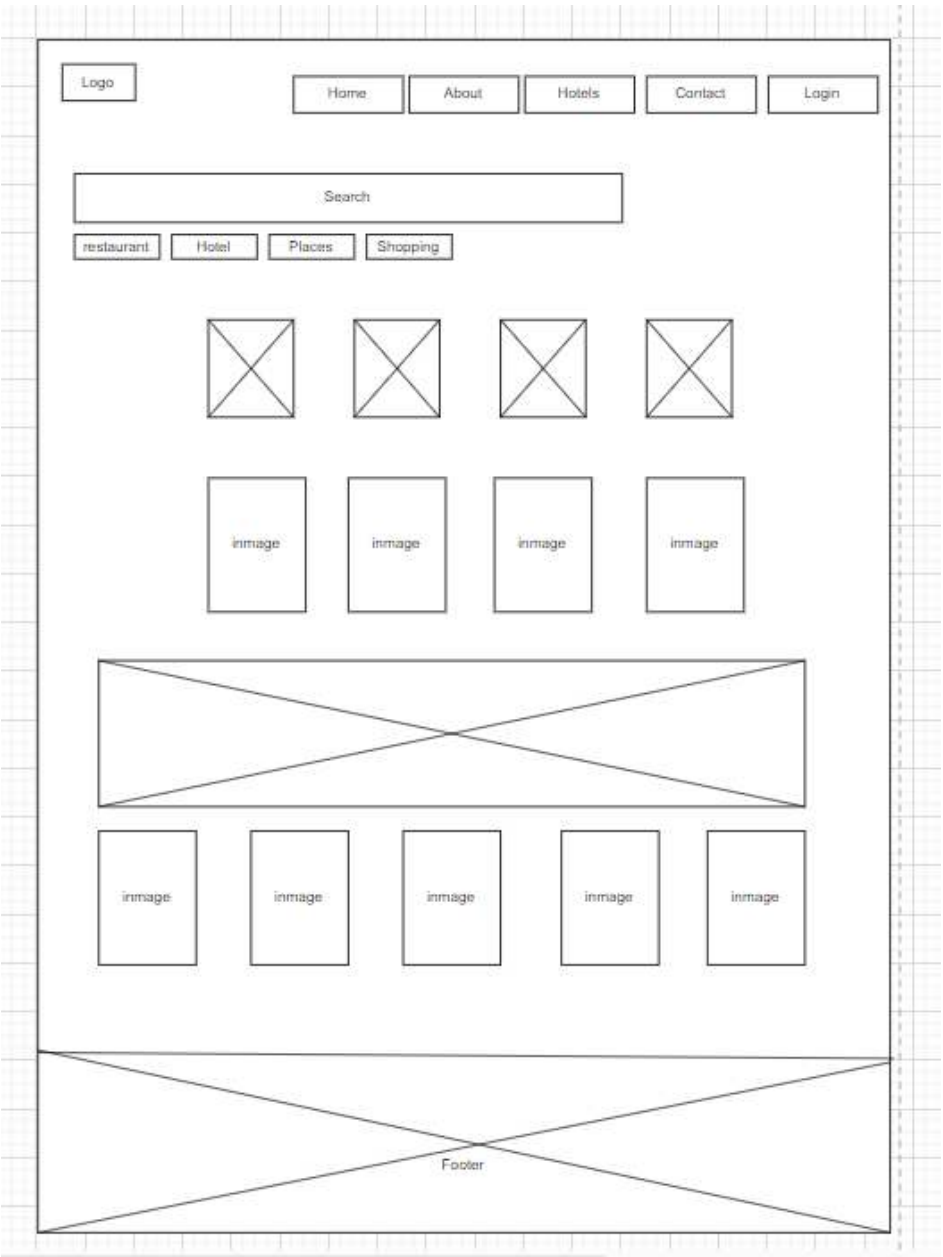

Figure 5: Index Hotel

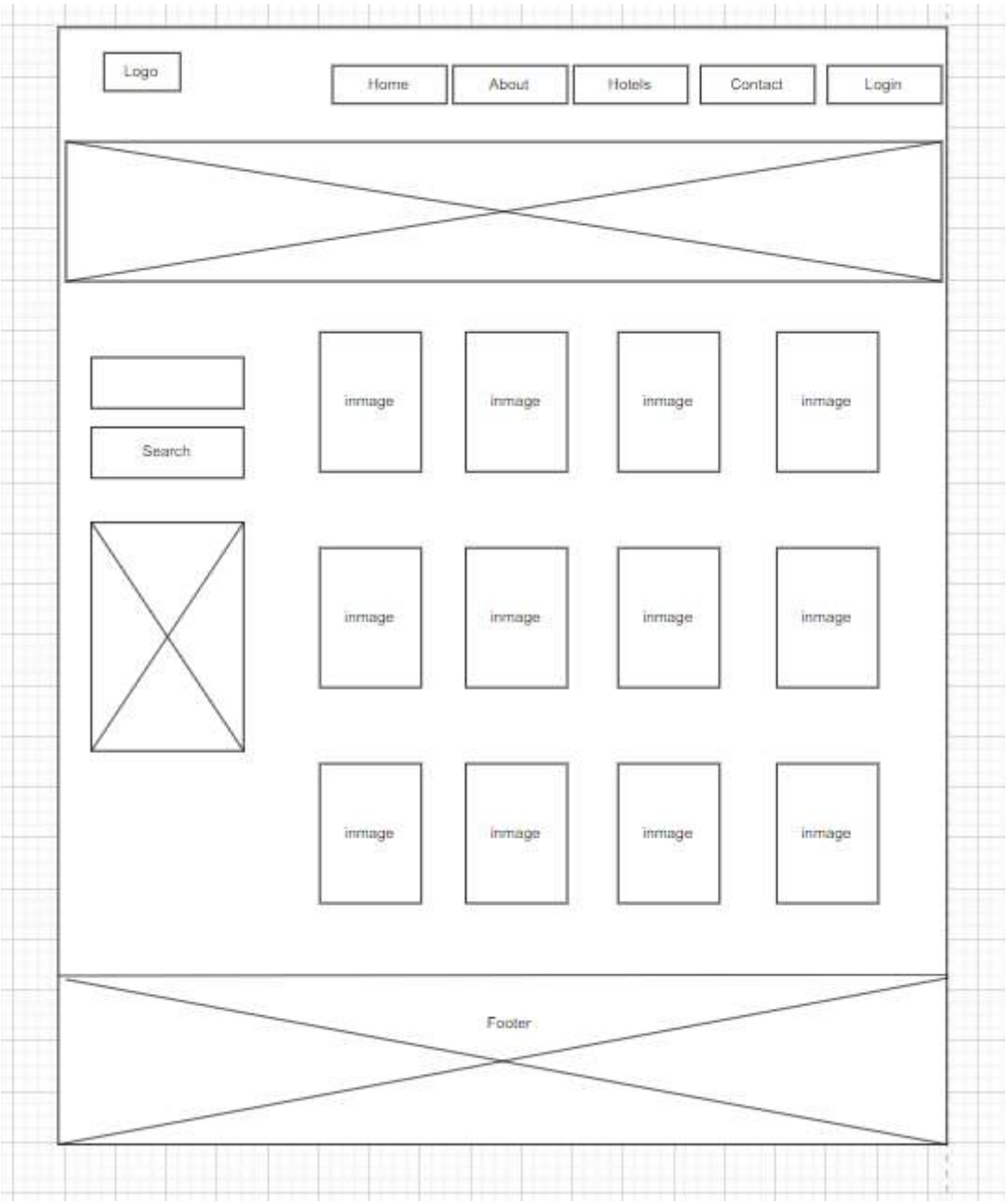

Figure 6: Hotels

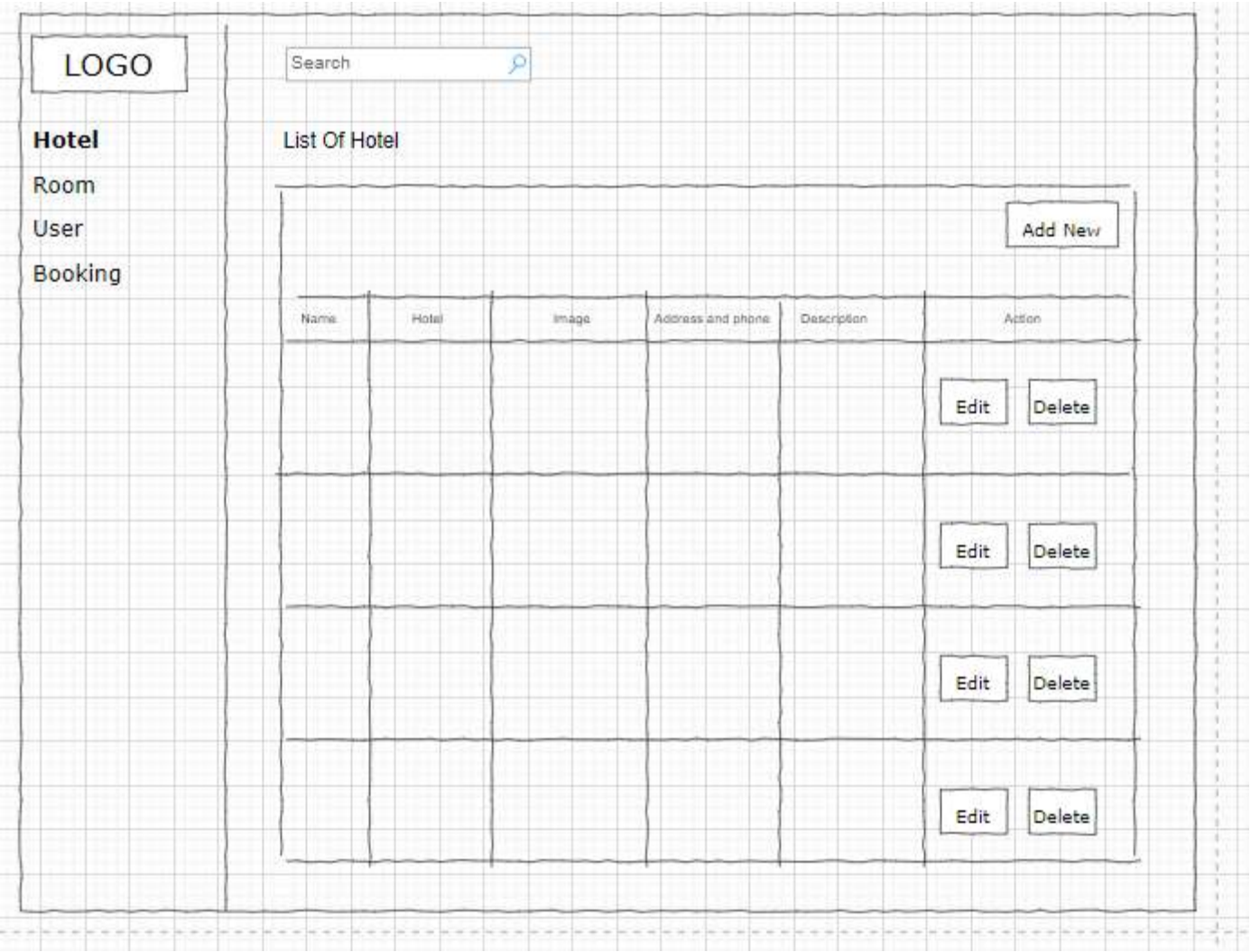

Figure 7: Manage Hotel

III. Implementation

1. Sample Source Code

1.1. Project structure

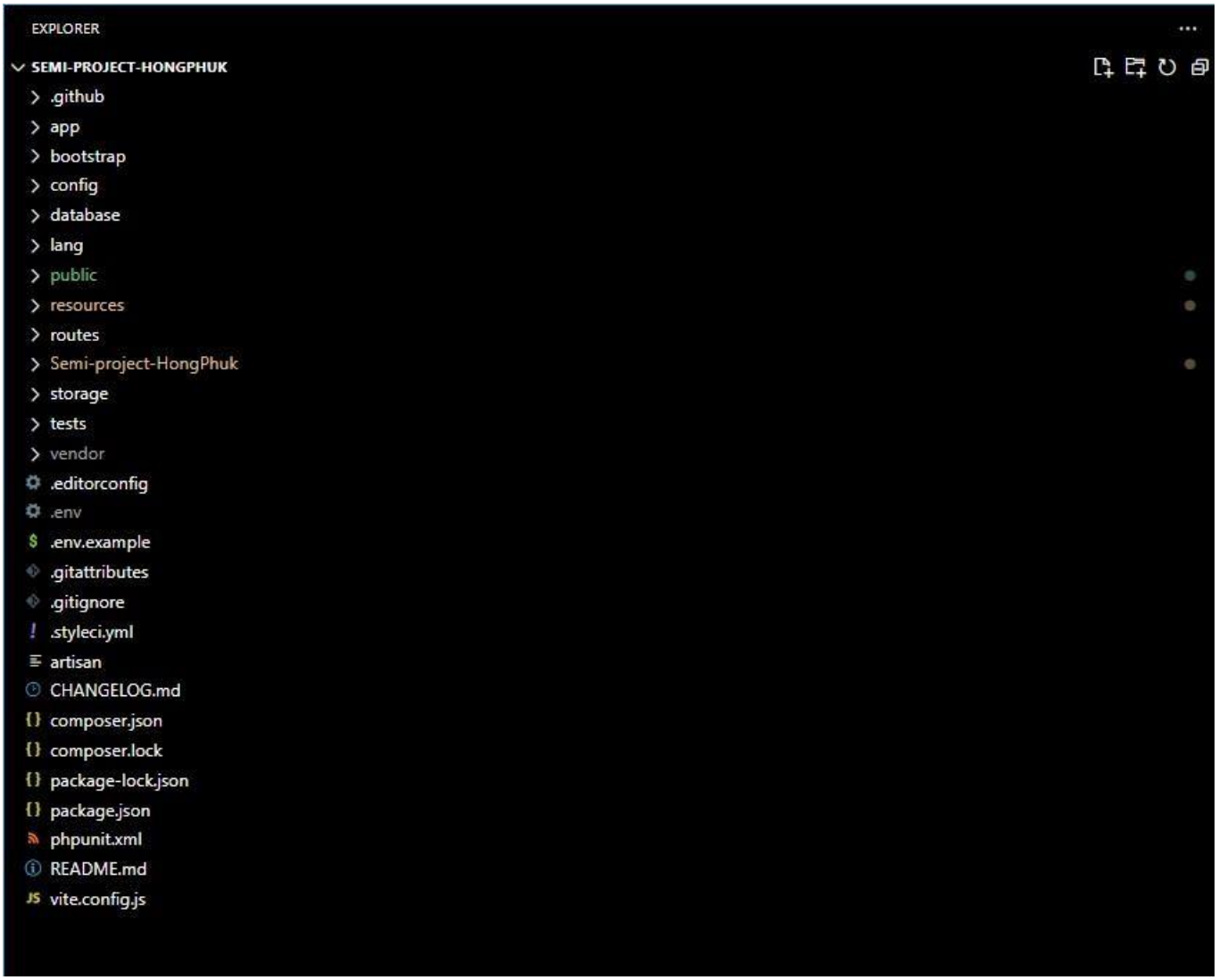

Figure 8: Project Structure: LARAVEL

1.1 Project Elements

In this project, I am in charge of developing partnerships and user systems with the site's goods. I'll be in charge of developing functionalities to add/edit/delete hotels and rooms, manage users, assign people permissions, and configure navigation.

Because Laravel is an MVC framework, all of the functions in the controller follow the MVC pattern's flow. Eg:

```php
Route::group(
    ['prefix' => 'admin'],
    function () {
        // Route::get("/", "TwoFaceAuthsController@index")->name("2fa_setting");
        Route::get('/index', '\App\Http\Controllers\Admin\AdminController@index')->name('admin.index');
        Route::get('/list-booking', '\App\Http\Controllers\Admin\BookingsController1@index')->name('admin.listBookings')->middleware('auth');

        Route::post('/booking/store', '\App\Http\Controllers\Admin\BookingsController1@store')->name('bookings.store');


        Route::get('/hotel', '\App\Http\Controllers\Admin\HotelController@index')->name('admin.hotel');
        Route::post('/hotel/store', '\App\Http\Controllers\Admin\HotelController@store')->name('admin.hotel.store');
        Route::get('/hotel/create', '\App\Http\Controllers\Admin\HotelController@create')->name('admin.hotel.create');
        Route::post('/hotel/create', '\App\Http\Controllers\Admin\HotelController@store')->name('admin.hotel.store');
        Route::get('/hotel/{id}/edit',  '\App\Http\Controllers\Admin\HotelController@edit')->name('admin.hotel.edit');
        Route::post('/hotel/{id}/edit',  '\App\Http\Controllers\Admin\HotelController@update')->name('admin.hotel.update');
        Route::get('/hotel/{id}/delete',  '\App\Http\Controllers\Admin\HotelController@destroy')->name('admin.hotel.delete');
        Route::get('/hotel/{id}',  '\App\Http\Controllers\Admin\HotelController@show')->name('admin.hotel.detail');

        Route::get('/room', '\App\Http\Controllers\Admin\RoomController@index')->name('room');
        Route::post('/room/store', '\App\Http\Controllers\Admin\RoomController@store')->name('room.store');
        Route::get('/room/create', '\App\Http\Controllers\Admin\RoomController@create')->name('room.create');
        Route::post('/room/create', '\App\Http\Controllers\Admin\RoomController@store')->name('room.store');
        Route::get('/room/{id}/edit',  '\App\Http\Controllers\Admin\RoomController@edit')->name('room.edit');
        Route::post('/room/{id}/edit',  '\App\Http\Controllers\Admin\RoomController@update')->name('room.update');
        Route::get('/room/{id}/delete',  '\App\Http\Controllers\Admin\RoomController@destroy')->name('room.delete');
        Route::get('/room/{id}',  '\App\Http\Controllers\Admin\RoomController@show')->name('room.detail');

        Route::get('/users', '\App\Http\Controllers\Admin\UsersController@index')->name('users');
        Route::post('/users/store', '\App\Http\Controllers\Admin\UsersController@store')->name('users.store');
        Route::get('/users/create', '\App\Http\Controllers\Admin\UsersController@create')->name('users.create');
        Route::post('/users/create', '\App\Http\Controllers\Admin\UsersController@store')->name('users.store');
        Route::get('/users/{id}/edit',  '\App\Http\Controllers\Admin\UsersController@edit')->name('users.edit');
        Route::post('/users/{id}/edit',  '\App\Http\Controllers\Admin\UsersController@update')->name('users.update');
        Route::get('/users/{id}/delete',  '\App\Http\Controllers\Admin\UsersController@destroy')->name('users.delete');
        Route::get('/users/{id}',  '\App\Http\Controllers\Admin\UsersController@show')->name('users.detail');
        // Route::get('/{path?}', function($path = null){
        //     return View::make('admin.index');
        // })->where('path', '.*');

    }
);
```

Figure 9: Route

```php
class HotelController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
    // list data
    public function index()
    {
        $hotels = Hotel::all();
        $users = User::all();
        return view('admin.hotel.list', compact('hotels', 'users'));
    }
    // detail data by id
    public function show($id)
    {
        $hotel = Hotel::findOrFail($id);
        return view('admin.hotel.detail', compact('hotel'));
    }
    // view form add data
```

Figure 10: Hotel Controller

- The index method retrieves all hotels and users from the database using the all method on the Hotel and User models. It then returns a view named admin.hotel.list and passes two variables, $hotels and $users, to the view using the compact function.

- The show method takes an $id parameter and uses the findOrFail method on the Hotel model to retrieve a specific hotel from the database with the given ID. If no hotel is found with that ID, it throws a ModelNotFoundException. The method then returns a view named admin.hotel.detail and passes the $hotel variable to the view using the compact function.

```php
public function create()
{
    $hotelAvailable = true;
    $hotels = Hotel::all();
    $users = User::all();
    return view('admin.hotel.create', compact('hotels', 'users', 'hotelAvailable'));
}

// function to save data
public function store(Request $request)
{
    if ($request->isMethod('POST')) {
        $validator = Validator::make($request->all(), [
            'name' => 'required',

            'img' => 'required|image|mimes:jpg,jpeg,png|max:100000',
            'des' => 'required',
            'address' => 'required'
        ]);
        if ($validator->fails()) {
            return redirect()->back()
                ->withErrors($validator)
                ->withInput();
        }
        if ($request->hasFile('img')) {
            $file = $request->file('img');
            $path = public_path('images/product');
            $fileName = time() . '_' . $file->getClientOriginalName();
            $file->move($path, $fileName);
        } else {
            $fileName = 'noname.jpg';
        }
        $newHotel = new Hotel();
        $newHotel->name = $request->name;

        $newHotel->img = $fileName;
        $newHotel->phone = $request->phone;
        $newHotel->des = $request->des;
        $newHotel->address = $request->address;


        $newHotel->save();
        return redirect()->route('admin.hotel')
            ->with('success', 'Product created successfully.');
    }
}
```

Figure 11:Add new and storage hotel

- The create method retrieves all the hotels and users from the database and passes them to a view called admin.room.create. This view is used to create a new room by providing details such as room name, price, maximum occupancy, hotel ID, image, and description.

- The store method is used to save the data submitted by the administrator when creating a new room. It uses validation rules to ensure that all the required fields are filled in correctly. If any of the validation rules fail, it redirects back to the previous page with the error messages and the user's input.

```php
class RoomController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
    // list data
    public function index()
    {
        $rooms = Room::all();
        $hotels = Hotel::all();
        return view('admin.room.list', compact('rooms', 'hotels'));
    }
    // detail data by id
    public function show($id)
    {
        $room = Room::findOrFail($id);
        return view('admin.room.detail', compact('room'));
    }
    // view form add data
```

Figure 12: Room Controller

- These methods are used in an application that allows administrators to manage rooms in hotels. The index method displays a list of all rooms, while the show method displays detailed information about a specific room.

```php
    public function create()
    {
        $hotels = Hotel::all();
        $users = User::all();
        return view('admin.room.create', compact('hotels', 'users'));
    }

    // function to save data
    public function store(Request $request)
    {
        if ($request->isMethod('POST')) {
            $validator = Validator::make($request->all(), [
                'name' => 'required',
                'price' => 'required',
                'img' => 'required|image|mimes:jpg,jpeg,png|max:100000',
                'des' => 'required',
                'hotel_id' => 'required',
                'max_occupancy' => 'required'

            ]);
            if ($validator->fails()) {
                return redirect()->back()
                    ->withErrors($validator)
                    ->withInput();
            }
            if ($request->hasFile('img')) {
                $file = $request->file('img');
                $path = public_path('images/rooms');
                $fileName = time() . '_' . $file->getClientOriginalName();
                $file->move($path, $fileName);
            } else {
                $fileName = 'noname.jpg';
            }
            $input=$request->all();
            $input['img']=$fileName;
            Room::create($input);
            return redirect()->route('room')
                ->with('success', 'Product created successfully.');
        }
    }
```

Figure 13: Add new and storage room

- The create method retrieves all the hotels and users from the database and passes them to a view called admin.room.create. This view is used to create a new room by providing details such as room name, price, maximum occupancy, hotel ID, image, and description.
- The store method is used to save the data submitted by the administrator when creating a new room. It uses validation rules to ensure that all the required fields are filled in correctly. If any of the validation rules fail, it redirects back to the previous page with the error messages and the user's input.

```php
class UsersController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
    // list data
    public function index()
    {
        $users = User::all();
        return view('admin.users.list', compact('users'));
    }
    // detail data by id
    public function show($id)
    {
        $hotel = Hotel::findOrFail($id);
        return view('admin.users.detail', compact('hotel'));
    }
    // view form add data
```

Figure 14: User Controller

- The index() function retrieves all users from the database using the User::all() method, and passes them to the list view using the compact() function. This view will display a list of all the users.
- The show($id) function retrieves a single user from the database using the findOrFail() method, which will throw a 404 error if no user is found with the specified ID. The retrieved user is passed to the detail view using the compact() function. This view will display the details of the user with the specified ID.

```php
public function create()
{
    return view('admin.users.create');
}

// function to save data
public function store(Request $request)
{
    if ($request->isMethod('POST')) {
        $validator = Validator::make($request->all(), [
            'name' => 'required',
            'email' => 'required',
            'phone' => 'required',
            'password' => 'required',
            'type' => 'required'
        ]);
        if ($validator->fails()) {
            return redirect()->back()
                ->withErrors($validator)
                ->withInput();
        }
        $input=$request->all();
        $input['password']=Hash::make($request->password);
        User::create($input);

        return redirect()->route('users')
            ->with('success', 'User created successfully.');
    }
}
```

Figure 15: Add new and storage user

- The create() function is used to return the view for creating new user data.
- The store() function is used to store the data when a POST request is made. First, it validates the input data using Laravel's built-in Validator class. If the validation fails, it redirects back to the previous page with the errors and the user's input data. If the validation passes, it creates a new User instance with the input data and hashed password using Laravel's Hash class, then saves it to the database using the create() method. Finally, it redirects to the user list page with a success message.

```php
class BookingsController1 extends Controller
{
    public function index()
    {
        $roomAvailable = true;
        $bookings = Booking::all();
        if (Auth::user()->type=='user') {
            $bookings = Booking::where('user_id',Auth::id())->get();
        }
        return view('admin.room.list-booking', compact('bookings'));
    }
```

Figure 16: Booking Controller

- This code defines a function called index that retrieves all the bookings and passes them to a view called admin.room.list-booking. If the current authenticated user is of type "user", it only retrieves bookings for that user.

```php
public function store(Request $request)
{
    $input=$request->all();
    // dd(Carbon::createFromFormat('m/d/Y', $input['checkin_date'])->diff(Carbon::createFromFormat('m/d/Y', $input['checkout_date']))->days);
    $validatedData = $request->validate([
        // 'user_id' => 'required|exists:users,id',
        'room_id' => 'required',
        'payment_method' => 'required',
        'checkin_date' => 'required|date',
        'checkout_date' => 'required|date',
    ]);
    if ($request->user_id == '') {
        // dd('ãdf');
        return redirect()->route('login');
    }
    $roomAvailable = $this->isRoomAvailable($validatedData['room_id'], $validatedData['checkin_date'], $validatedData['checkout_date']);

    if (!$roomAvailable) {
        return redirect()->back()->withErrors(['message', 'Sorry, the room is not available for the selected dates.']);
    }
    $input['checkin_date'] = Carbon::createFromFormat('m/d/Y', $input['checkin_date'])->format('Y-m-d 14:00:00');
    $input['checkout_date'] = Carbon::createFromFormat('m/d/Y', $input['checkout_date'])->format('Y-m-d 12:00:00');
    // dd($input);
    $booking = Booking::create($input);
    $room=Room::findOrFail($input['room_id']);
    $payment=[];
    $payment['booking_id']=$booking->id;
    $payment['payment_method']=$booking->id;
    $payment['amount']=(int)Carbon::createFromFormat('Y-m-d 14:00:00', $input['checkin_date'])->diff(Carbon::createFromFormat('Y-m-d 12:00:00', $input['checkout_date']))->days * (int)$room->pri
    Payment::create($payment);
    return redirect()->route('user.detail', ['id' => auth()->user()->id]);
}

private function isRoomAvailable($hotelId, $checkinDate, $checkoutDate)
{
    $bookings = Booking::where('room_id', $hotelId)
        ->where(function ($query) use ($checkinDate, $checkoutDate) {
            $query->whereBetween('checkin_date', [$checkinDate, $checkoutDate])
                ->orWhereBetween('checkout_date', [$checkinDate, $checkoutDate]);
        })
        ->get();

    return $bookings->count() === 0;
}
```

Figure 17: Storage booking

- This is a function for creating a new booking and payment for a room. It first validates the input data and checks if the chosen dates are available by calling the getAvailableDates() method. If the dates are not available, it returns an error message.
- If the dates are available, it creates a new booking and payment record in the database, and then redirects the user to their profile page.
- The getAvailableDates() method generates an array of available dates for the next 30 days. It first selects all the booked dates from the database and then loops through each day between the current date and 30 days in the future. If a day is not already booked, it adds it to the available dates array.

2. Web Screenshot

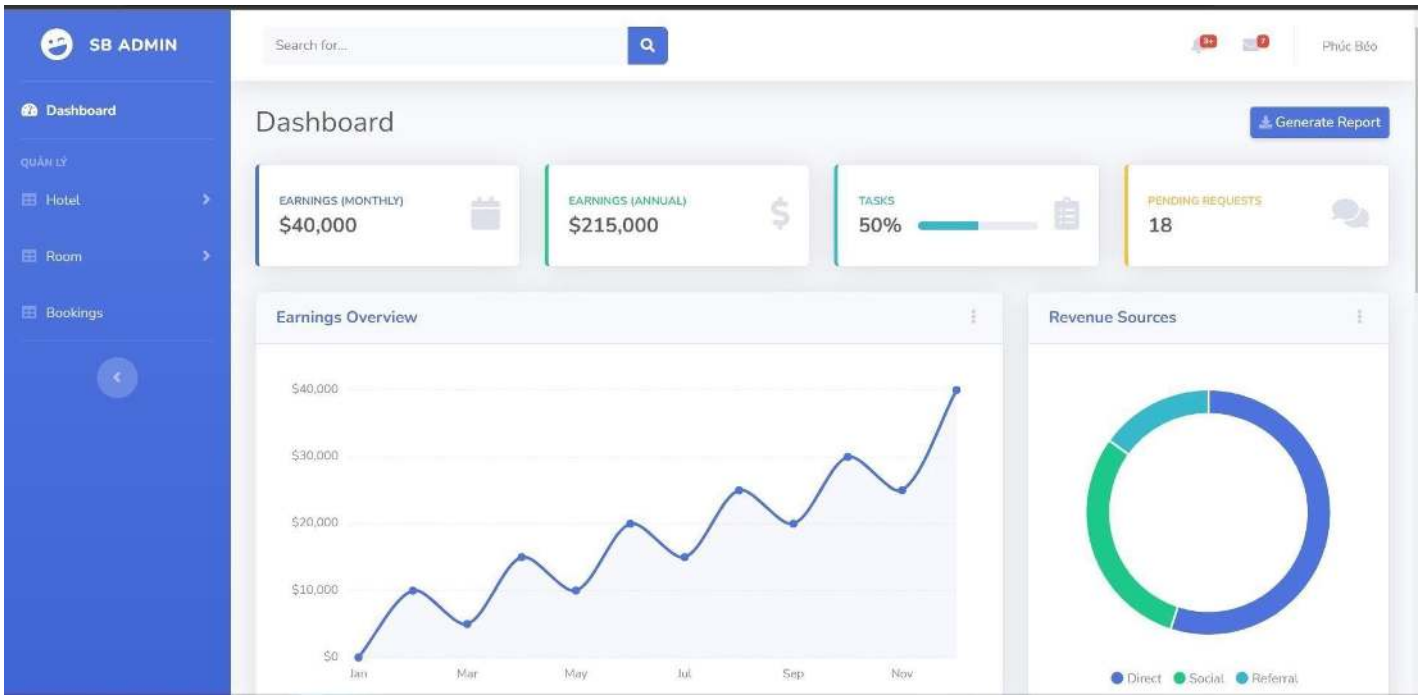

Figure 18: Web-Screenshot: Admin | Dashboard

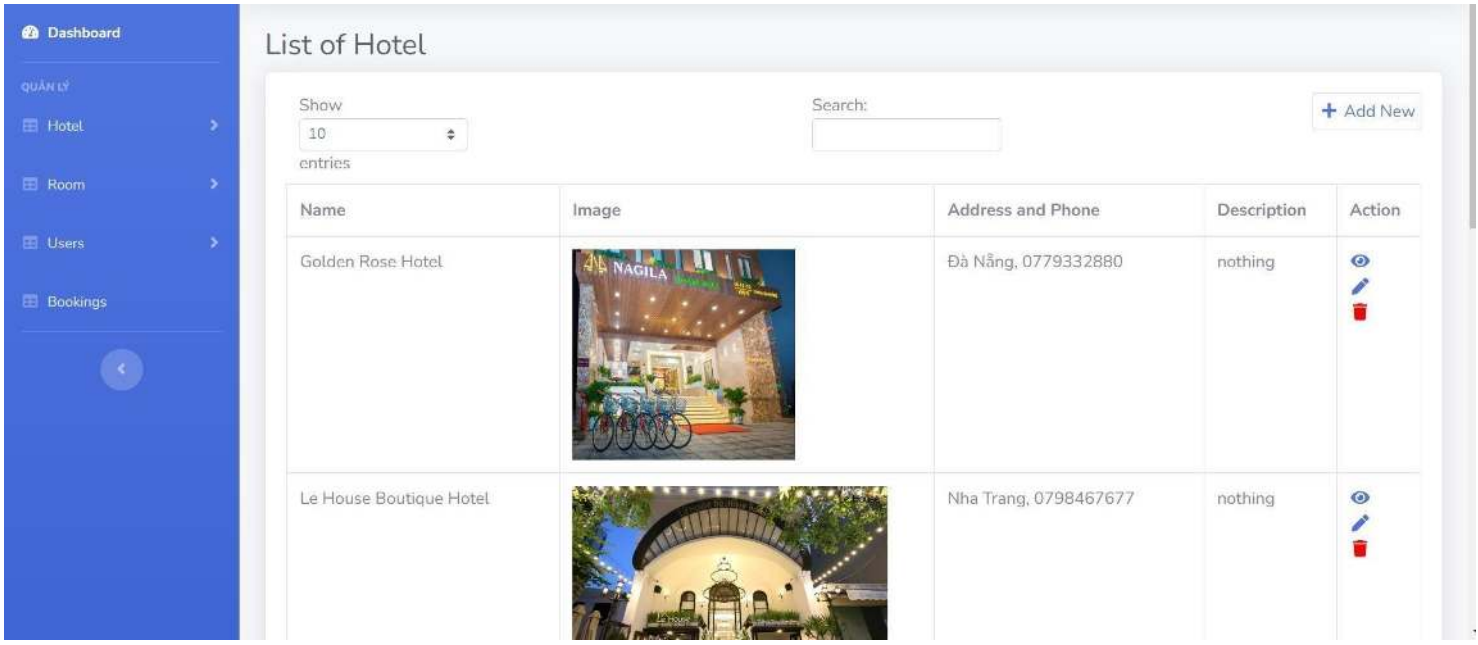

Figure 19: Web-Screenshot: Admin | Hotel

Figure 20: Web-Screenshot: Admin | Add New Hotel

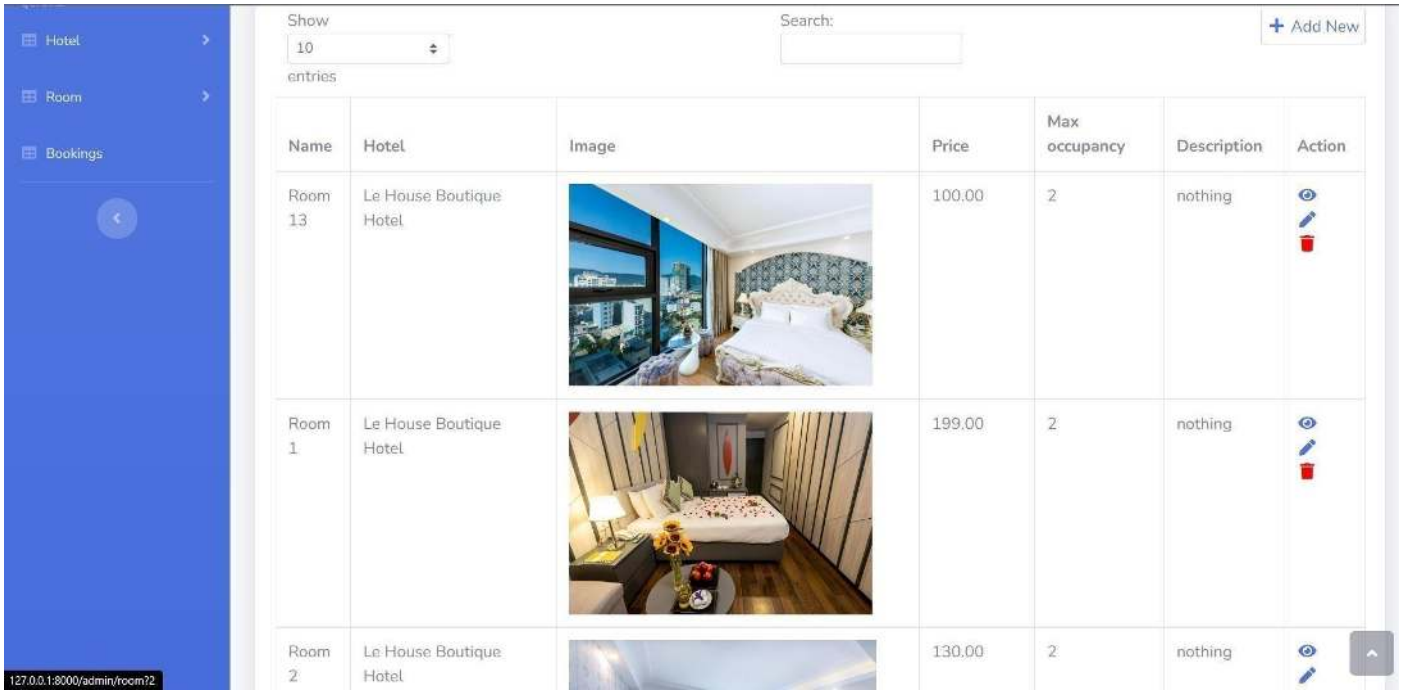

Figure 21: Web-Screenshot: Admin | Room

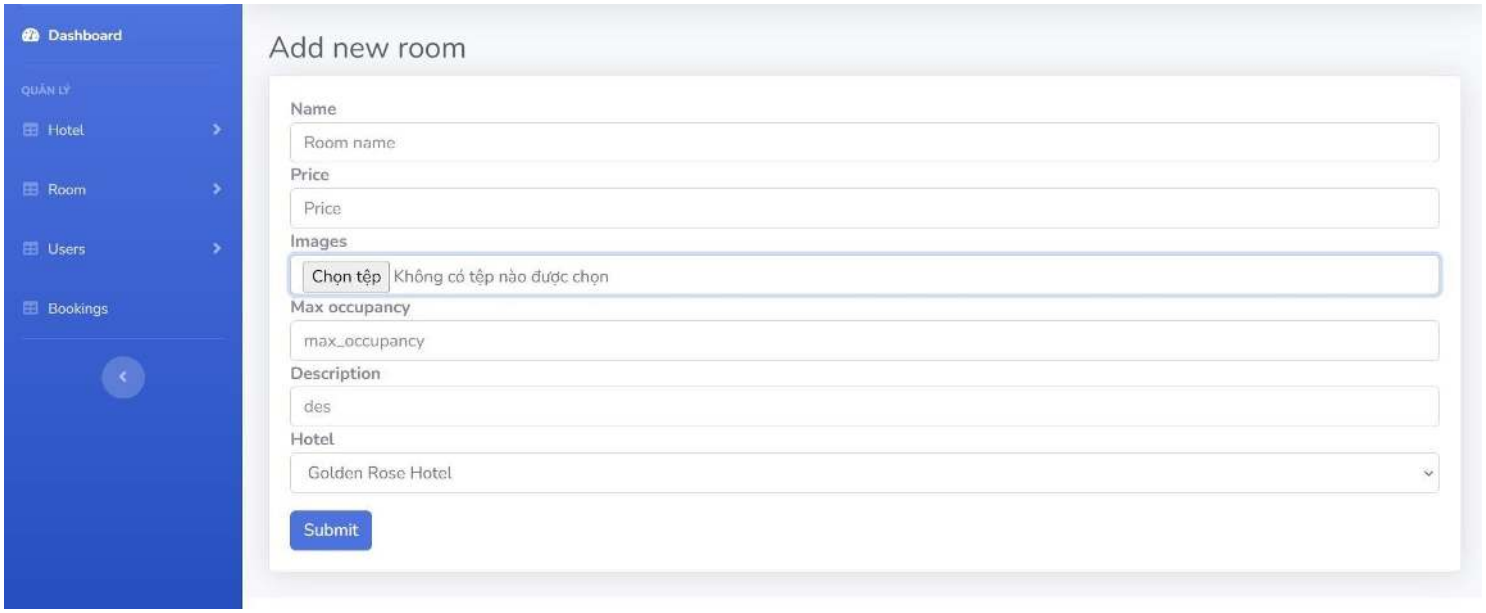

Figure 22: Web-Screenshot: Admin | Add New Room

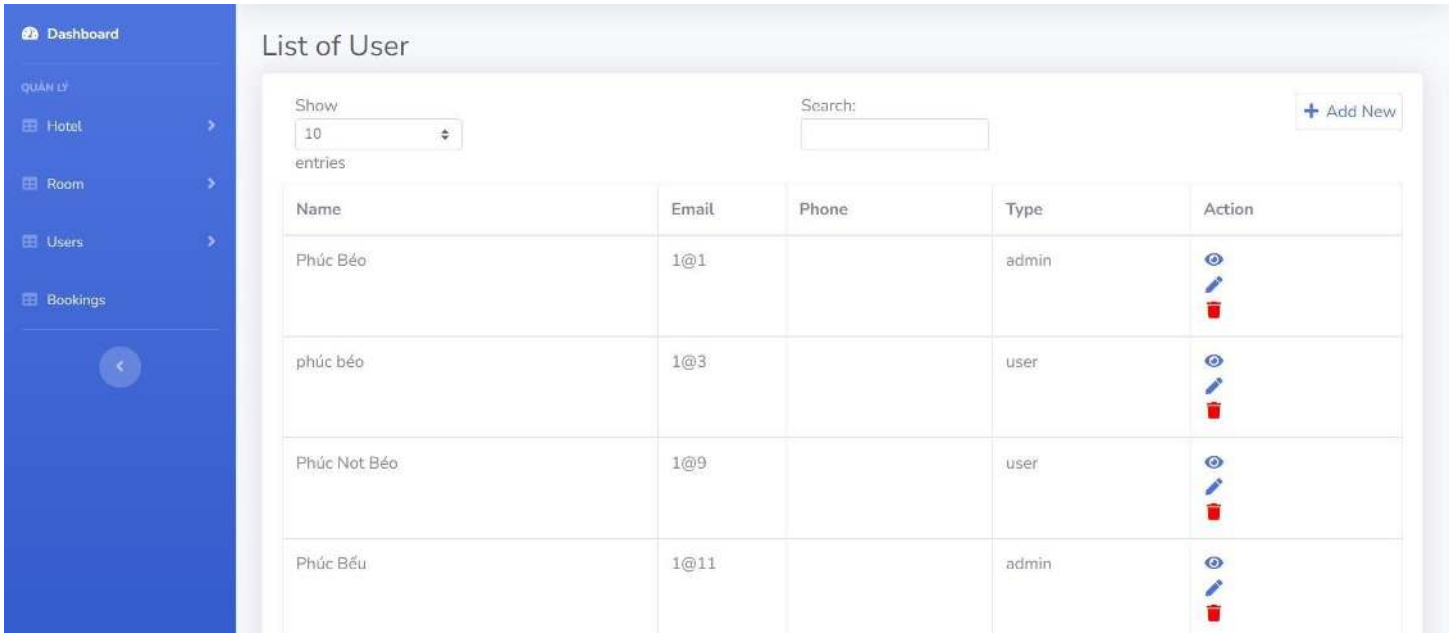

Figure 23: Web-Screenshot: Admin | User

Figure 24: Web-Screenshot: Admin |Add New User

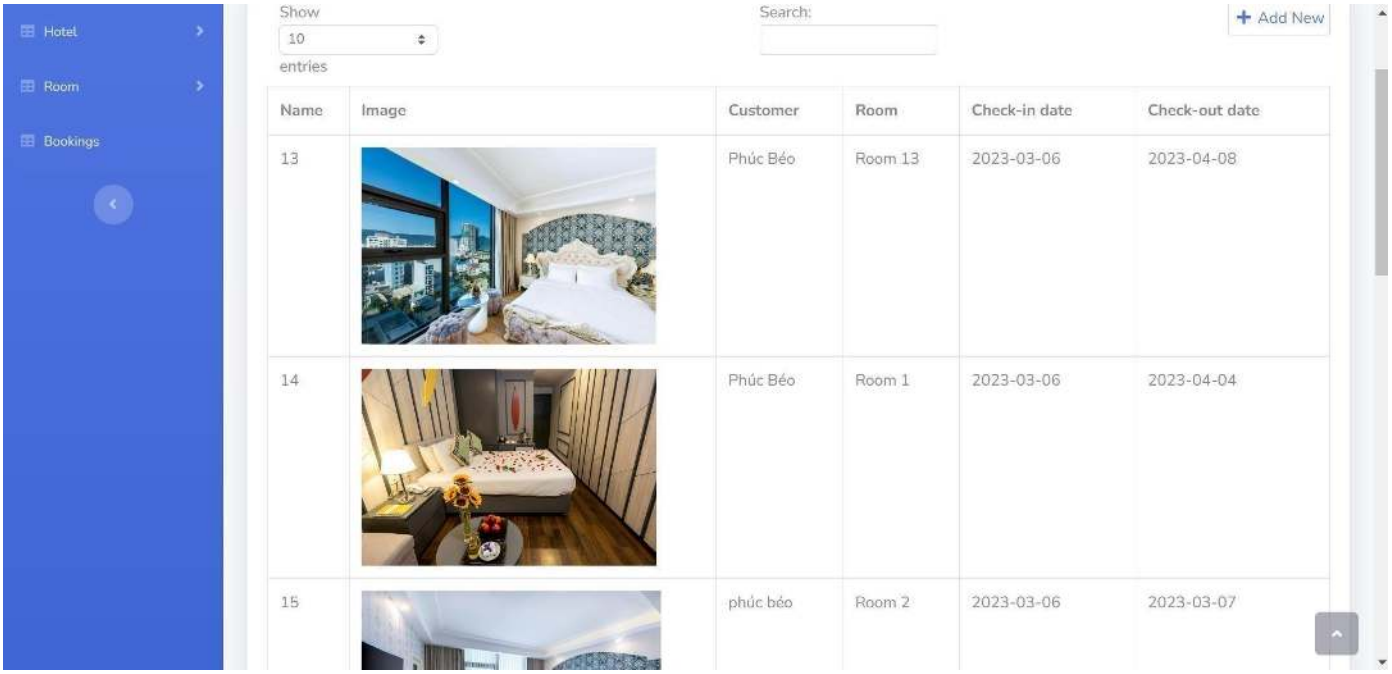

Figure 25: Web-Screenshot: Admin | Booking

IV. Conclusion

We conducted a critical evaluation of the finished product for potential future enhancement after implementing the software. We will create a lot more features in the future that will improve user experience and make it simpler for people to use. I'll list the benefits and drawbacks of the website we developed below.

1. Advantages of the website

The advantages of the web can be mentioned as the interface we use to display it will help users easily identify this is a brand related to the "hotel" product. Below I will list some of the advantages of the site as follows:

- Due to the same unadulterated PHP code as before, we have saved a ton of time by using the Laravel computer language. It also enables the code in my view files to be managed more efficiently when the functions are handled at the Controller and Model.
- The processing sequence is very obvious when using the MVC model, and the separate duties are also managed separately and do not impact other members of the project. It is simple to manage the flow of processing and apps at the same time.

2. Disadvantages of the website

The page's drawback has also had a significant impact. I will address each one in turn below:

▪ Our main issue right now is security. XSS or SQL Injection can quickly target the site's data. Although we made an effort to encrypt the data, the website's security warning feature kept reminding me of the dangers we were exposed to. ▪ Laravel is one of the new languages that our team members must acquire in order to complete this project. For some of our team members, this is a challenging one, making collaboration and teamwork extremely challenging.

3. Lesson Learnt

We learned a lot about the real-world aspects of this endeavor as well:

- Since Laravel is a relatively new and well-liked program, we will gain some additional proficiency with it.
- The MVC design is now better known.

V. Appendix

1. Group member list

No.	Member Name	Member ID	Role
1	Mai Duc Anh	BH00056	Leader
2	Tran Hong Phuc	BH00293	Member
3	Do Khanh Toan	BH00140	Member

2. Task

No.	Task	Assign	Task description
1	Front-End (Admin – Client)	Duc Anh	Complete user interface creation using the Bootstrap framework in conjunction with JavaScript, HTML, and CSS
2	Create a project on Laravel	Duc Anh	Create a Laravel project, crop pages, divide layout, and set the path so that it appears on the user interface.
3	Database creation for transfer with models and manager	Phuc	join using the.env file after starting the database. Configure the data responsibilities and column structure to create links between tables on models.

4	Manage booking	Phuc	Create and link data between the admin side and the customer
8	Manage Hotels	Phuc	Create hotel administration using CRUD processes. then establish a database connection.
9	Manage User	Phuc	With CRUD operations, create account administration features. then establish a database connection.
10	Handling access, autonomy, and preserving individual access rights	Toan	Manages user access rights, data requests, and stores user accounts while working online.
11	Login, Logout, Search	Toan	Login of hotel Search hotel that customer want to book

3. Link to GitHub

Link Github: https://github.com/ducanhaovai/Semi-project

4. GitHub commit evidence

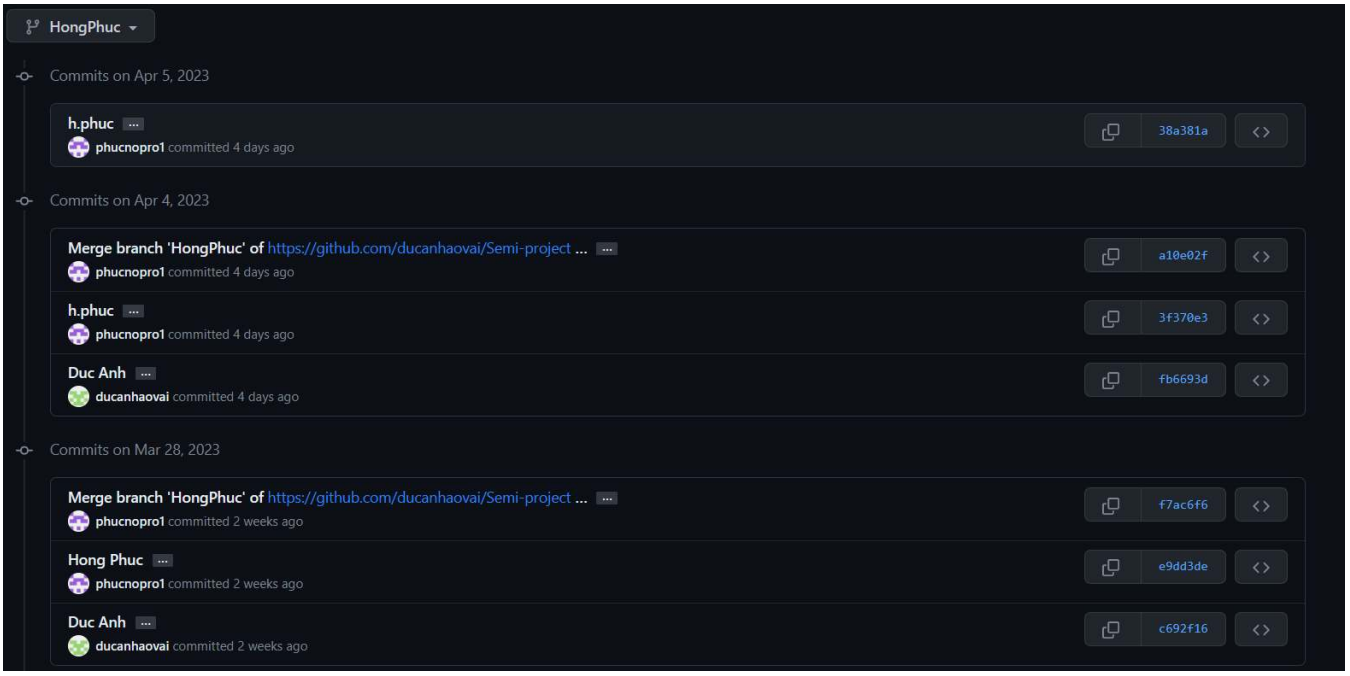

Figure 26: Commit 1

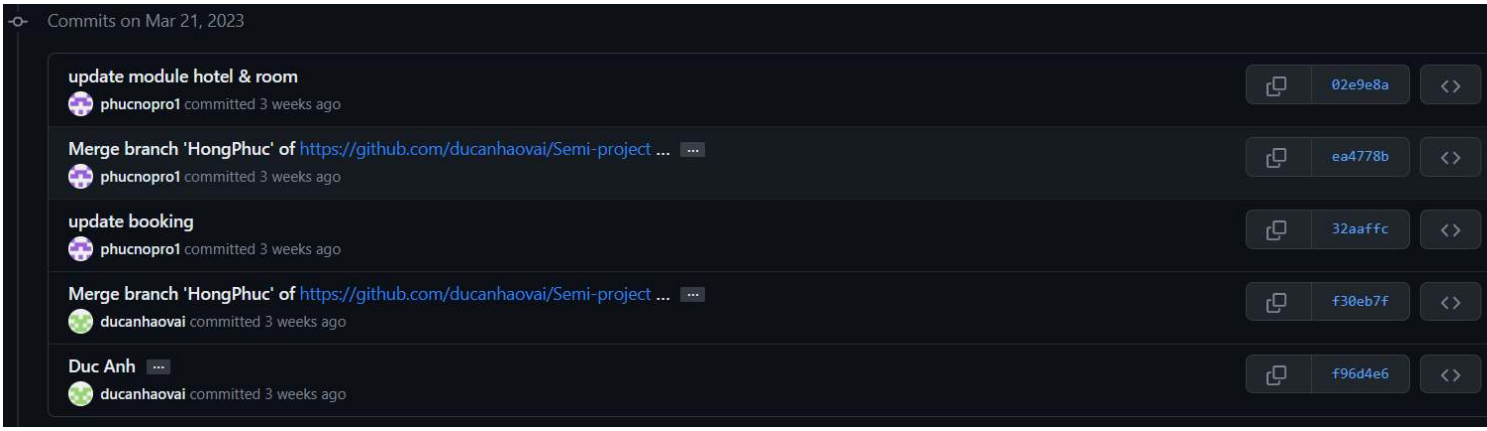

Figure 27: Commit 2

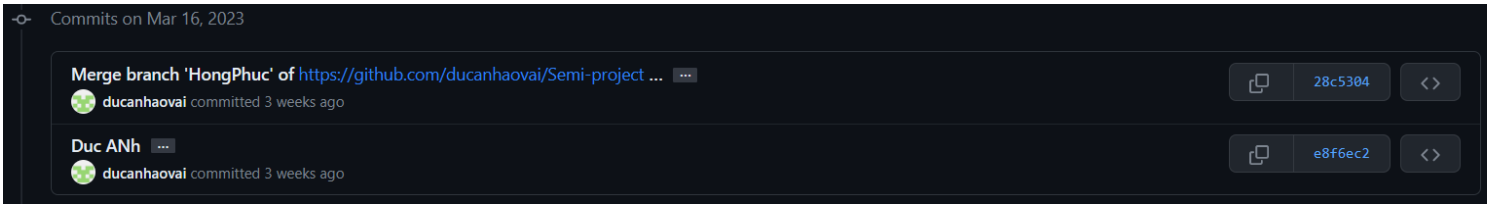

Figure 28:Commit 3

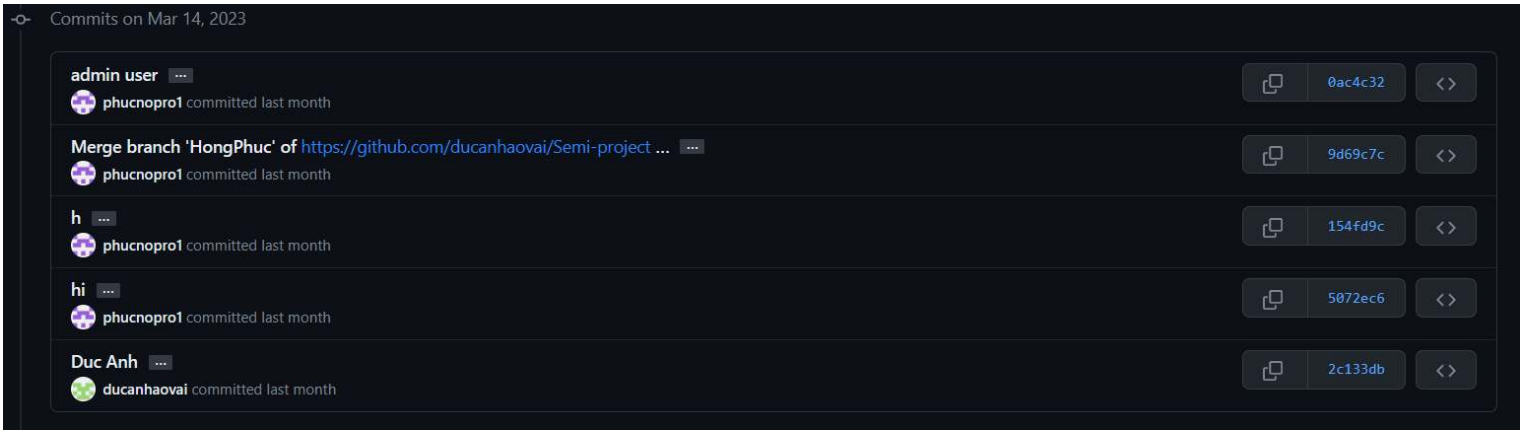

Figure 29: Commit 4

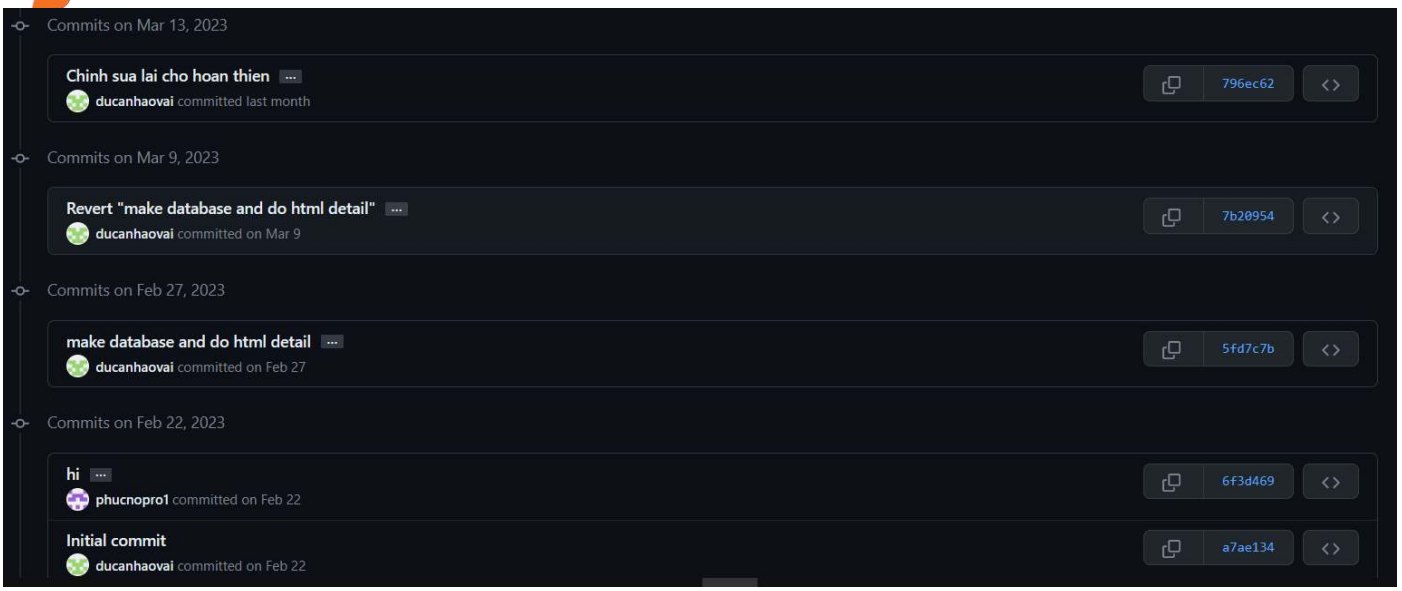

Figure 32: Commits 5