GROUP PROJECT FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	WEBG301 - Project Web		
Submission date	31/10/2023	**Date Received 1st submission**	31/10/2023
Re-submission Date		**Date Received 2nd submission**	
Student Name	Bui Duy Hung	**Student ID**	GCH200680
Class	GCH1108.1	**Assessor name**	Pham Duc Tho
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	Hung

Grade

❒ **Summative Feedback:**	❒ **Resubmission Feedback:**

Grade:	**Assessor Signature:**	**Date:**

Signature & Date:

Table of Contents

1. User' requirements (group)

1.1. User Stories template

No	As a <type of user/personal>	I want to <global objective>	So that <benefit/result/reason>
1	Admin	Login/logout	I can login/logout my account in the blog post web.
2	Admin	Register	I can register an account in the blog post web.
3	Admin	Create a new post	I can create a new post with the information inside as title and content.
4	Admin	View all my posts and other user's posts	I can view all my posts and other user's post with my permission.
5	Admin	Delete my post and other user's post	I can delete my post and other user's post with my permission.
6	Admin	Update my post and other user's post	I can update my post and other user's post with the information inside as title and content.
7	Admin	Change my avatar	I can change my avatar in the blog post web.
8	Admin	Follow/unfollow other user account	I can follow/unfollow other user account.
9	User	Login/logout	I can login/logout my account in the blog post web.
10	User	Register	I can register an account in the blog post web.

11	User	Create a new post	I can create a new post with the information inside as title and content.
12	User	View all my posts	I can view all my posts created from the database.
13	User	Delete my post	I can delete my post created out the database.
14	User	Update my post	I can update my post created with the information inside as title and content.
15	User	Change my avatar	I can change my avatar in the blog post web.
16	User	Follow/unfollow other user account	I can follow/unfollow other user account.

Table 1: User stories

1.2. Use case diagram (optional)

This blog post website system includes both Admin and User. Admin can create/view/update/delete my posts and can view/update/delete posts of other users. User also has the same features except the secondary functions of Admin. In addition, both Admin and User can follow and unfollow other users' accounts.

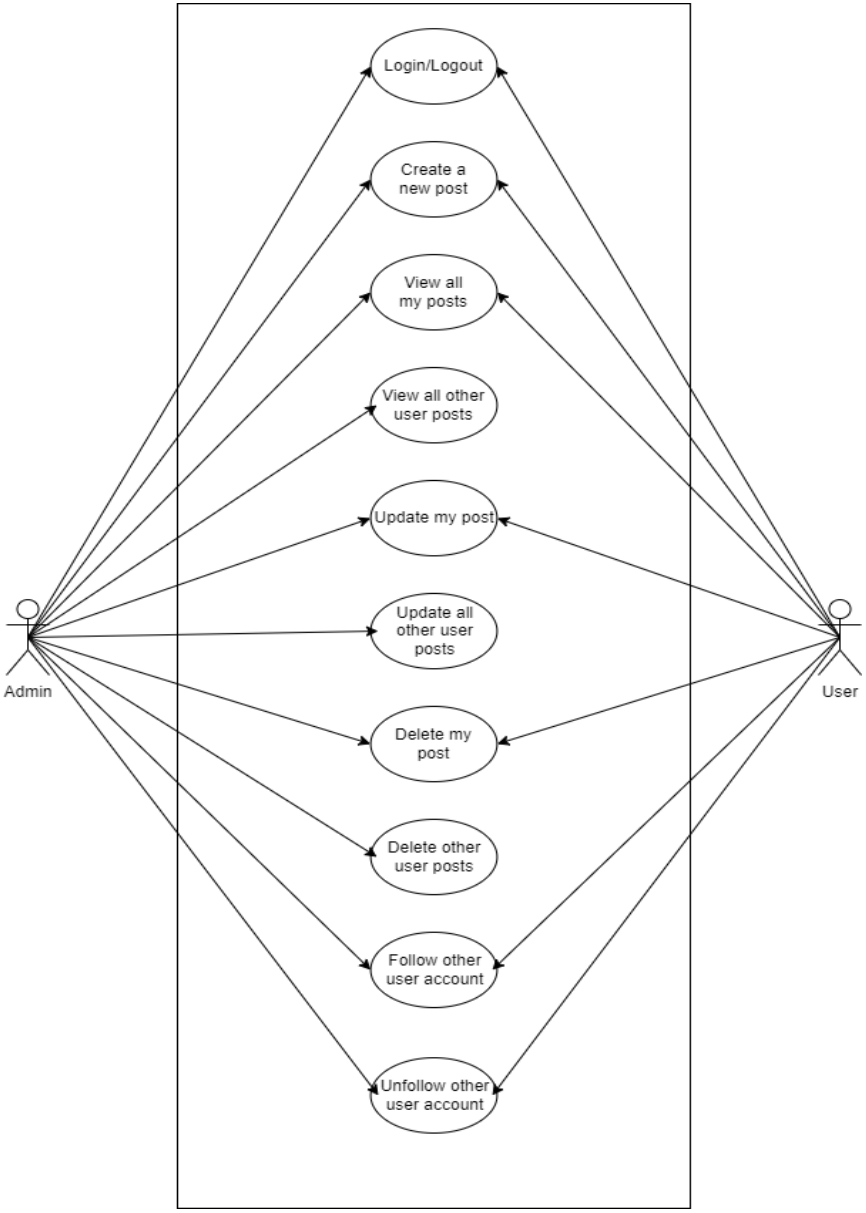
Login/Logout
Create a new post
View all my posts
View all other user posts
Update my post
Update all other user posts
Delete my post
Delete other user posts
Follow other user account
Unfollow other user account
Admin
User

Figure 1: User case diagram

2. System Design (Group)

2.1. Site map

This is our team sitemap including for Admin and User. User belongs to the regular user group and Admin belongs to the highest user group. First, both groups of users can log in, register, and log out of the system. User accounts are logged in and registered normally, except for the Admin account, which will be upgraded based on changes in database rights. A regular user account will be able to create posts, update posts, delete posts and view all posts that have been created. For external Admin accounts, the previous functions are similar to the user's account and will be able to update and delete other users' posts. This is the authority that the Admin account is allowed to use over other user accounts. In addition, both types of accounts can follow other user accounts and there will be two pages showing information about which users are following each other. From user accounts that follow each other, there will be a home page feed that displays posts from user accounts whose accounts we have followed. Finally, there will be the avatar change feature. If the user does not change it then it will have a temporary avatar fixed there.

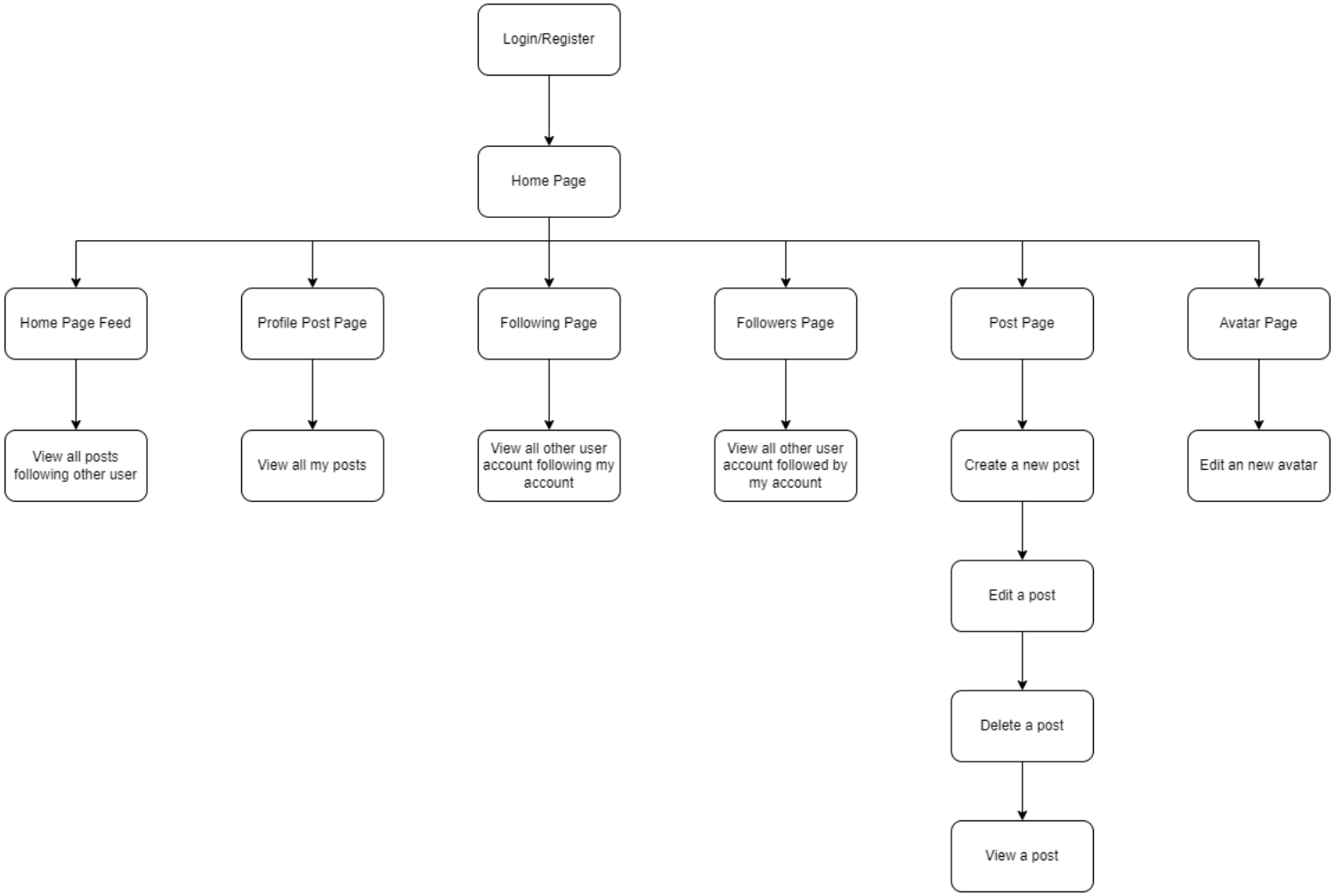

Figure 2: Site map

2.2. Entity Relationship Diagram

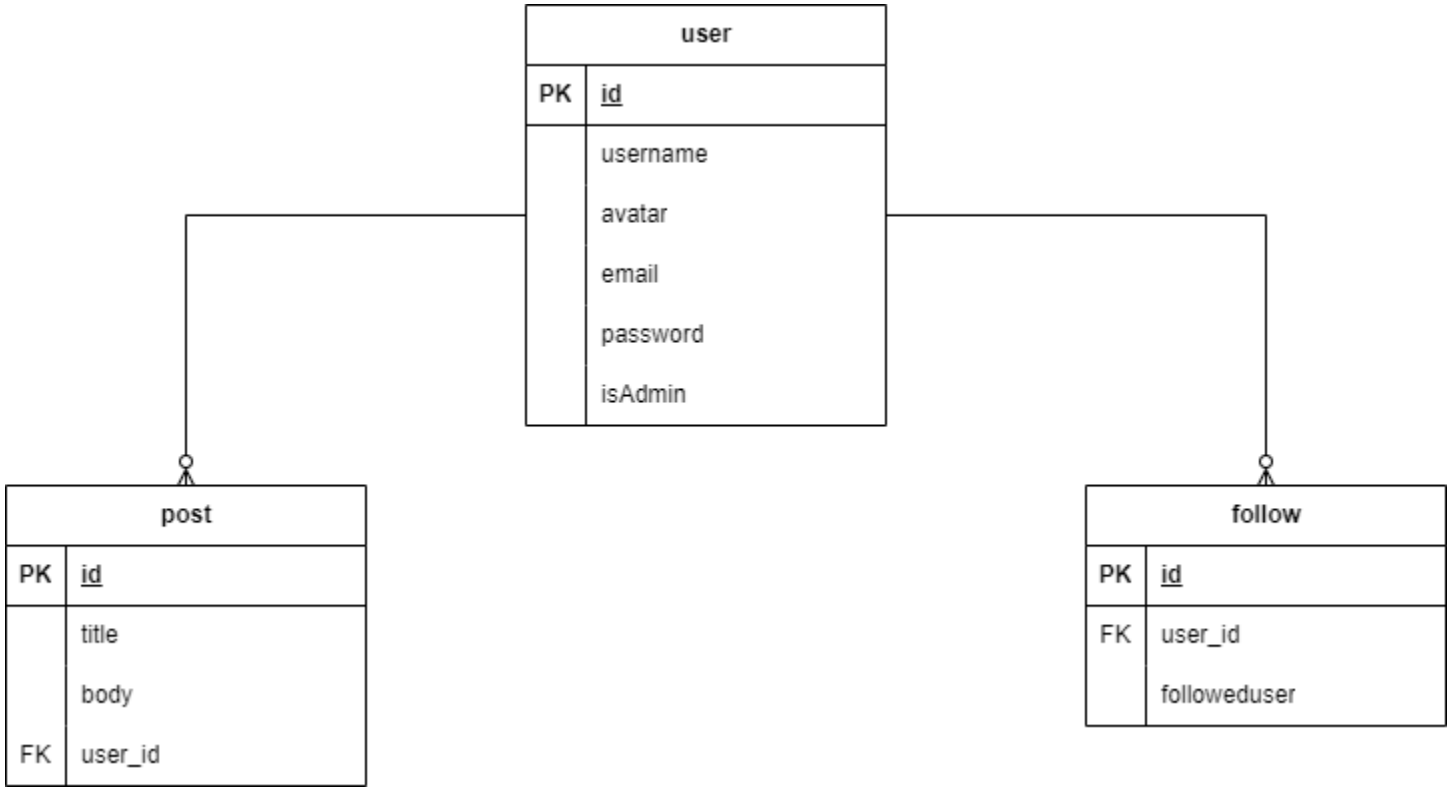

Figure 3: Entity Relationship Diagram

2.3. Wireframes

- Login/Register:

- Login: There are two fields that need to be filled in: username and password to log in
- Register: There are four fields that need to be filled in: username, email, password and password verification to be able to register a new account.

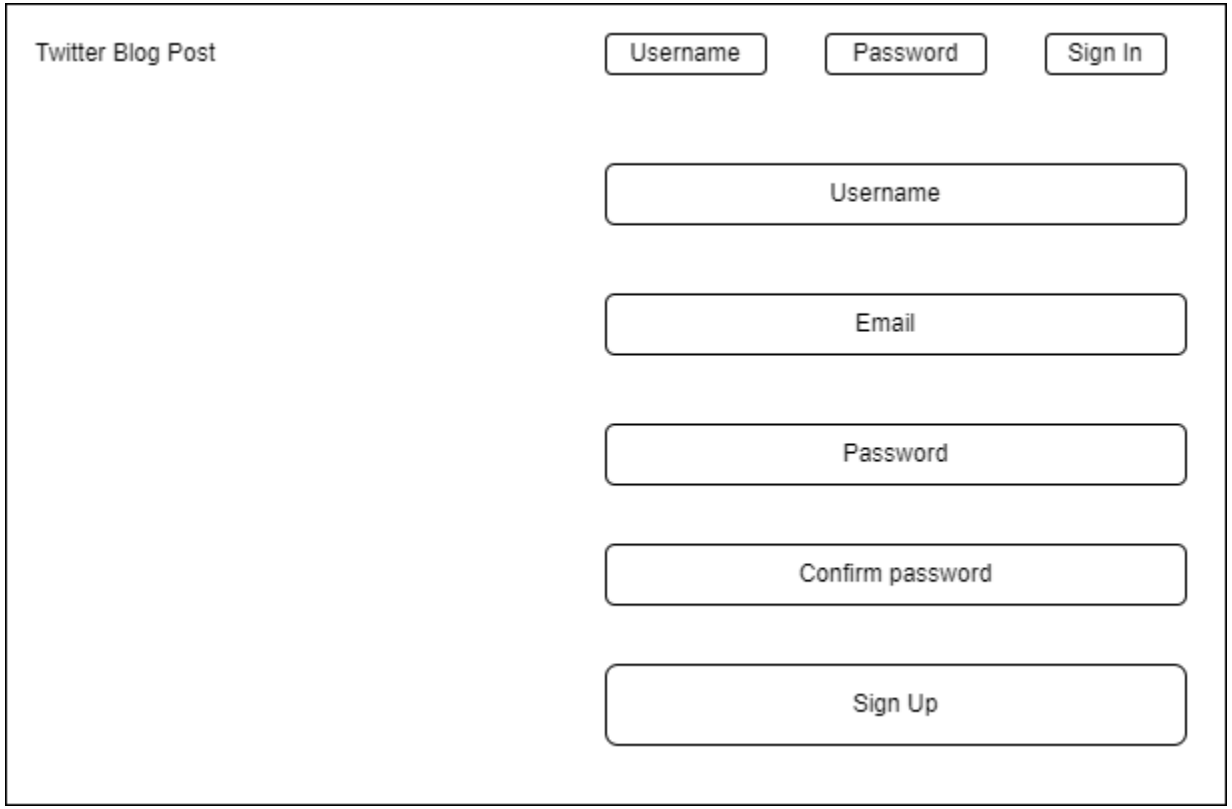

Figure 4: Login/Register Page

- Homepage-feed: Shows all posts where we have followed other user accounts.

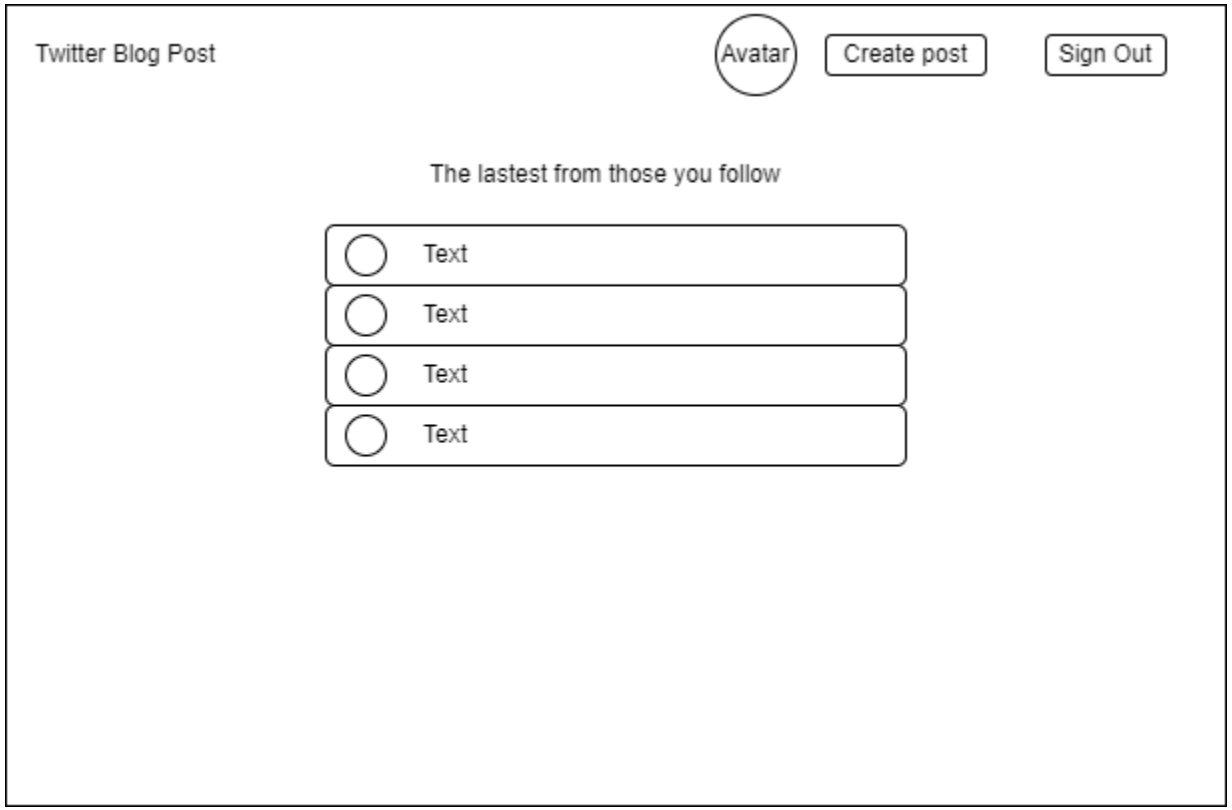

Figure 5: Homepage-feed

- Profile-post page: Displays all the posts we created previously.

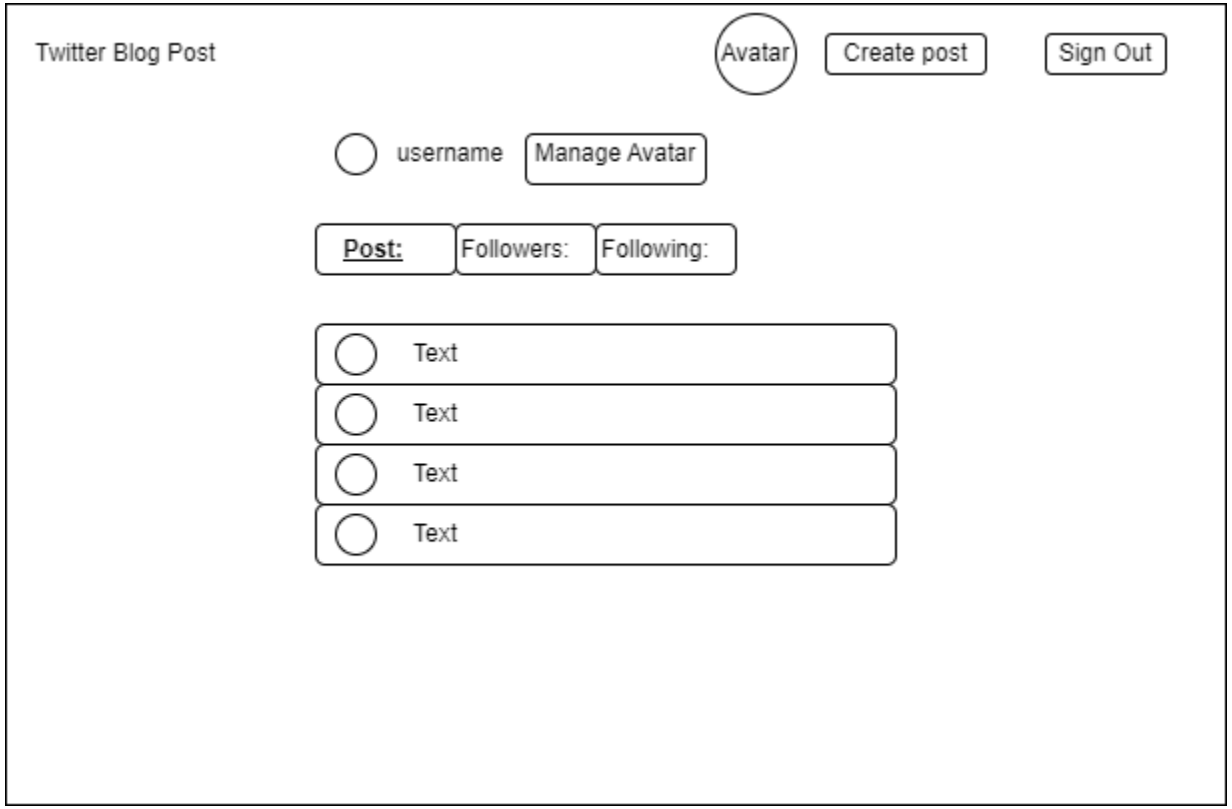

Figure 6: Profile-post page

- Follower page: Showing user accounts that are following my account.

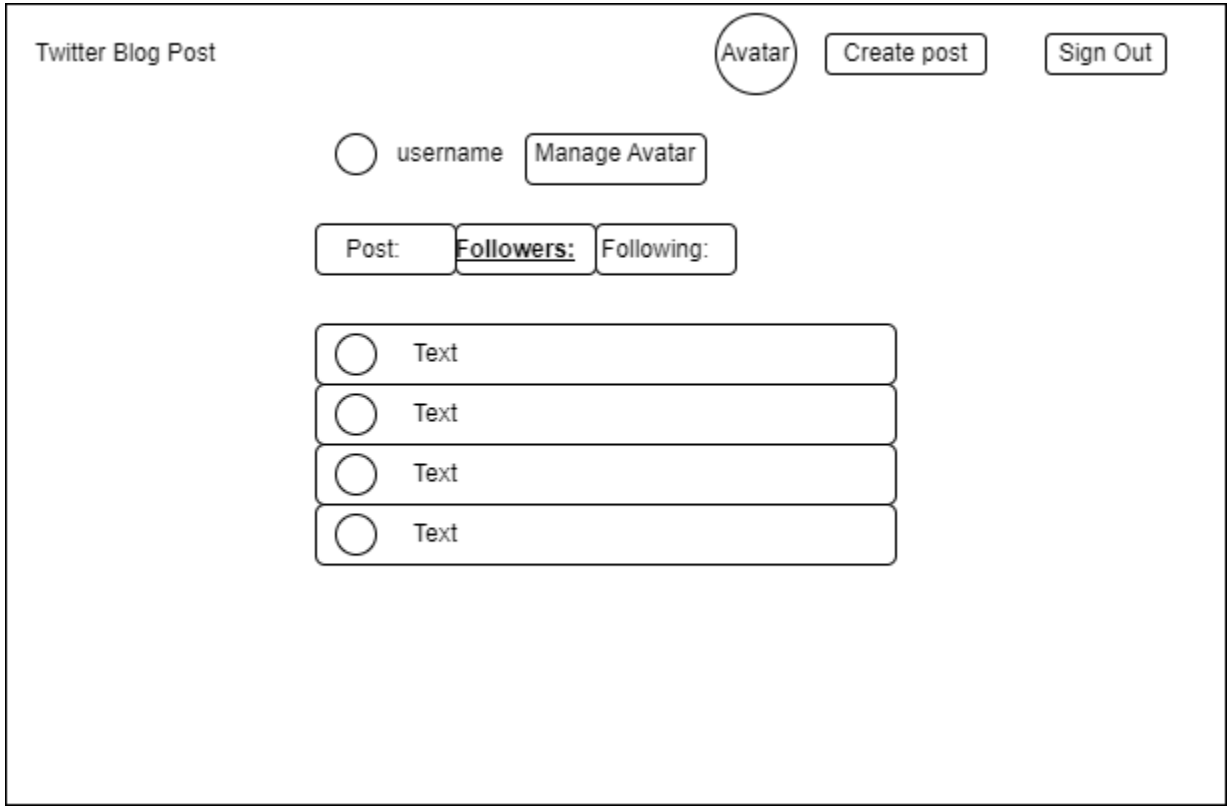

Figure 7: Follower page

- Following page: Shows user accounts that I have followed.

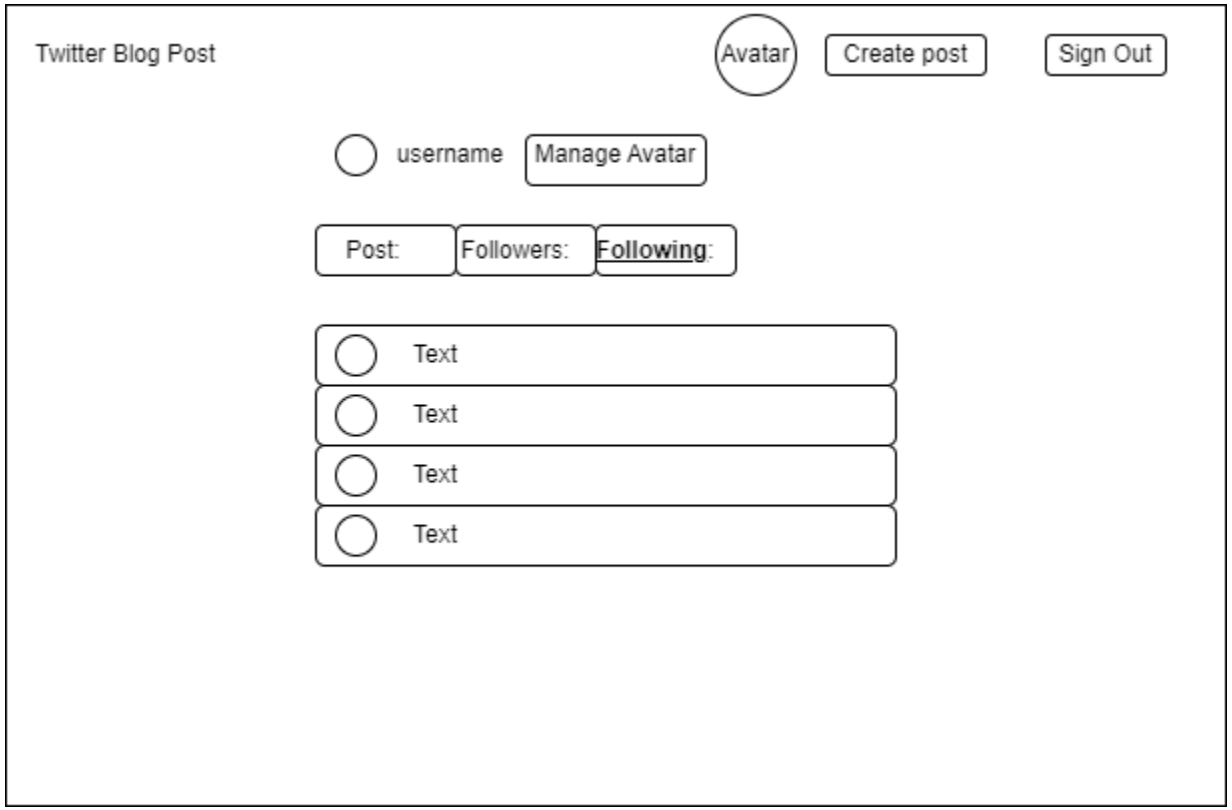

Figure 8: Following page

- Create post page: There are two fields that need to be filled in: title and content to create a complete post.

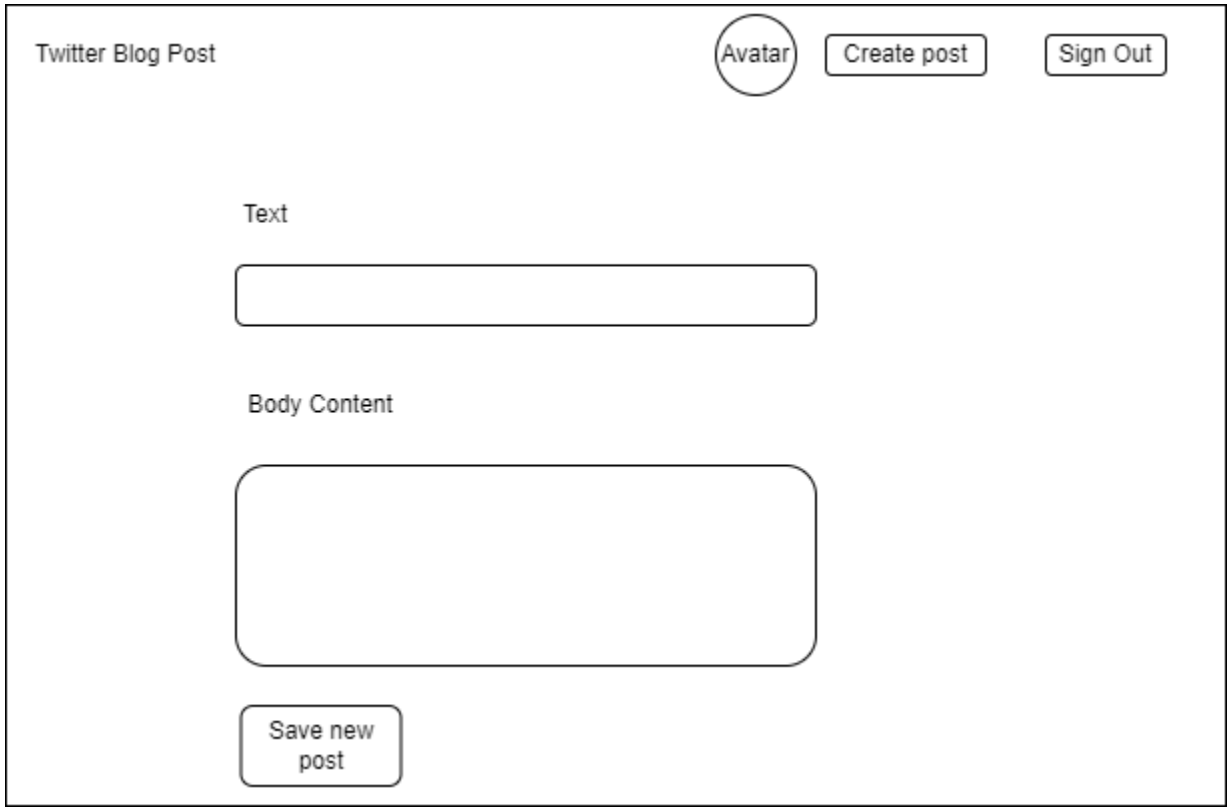

Figure 9: Create post page

- View post page: A previously created post appears.

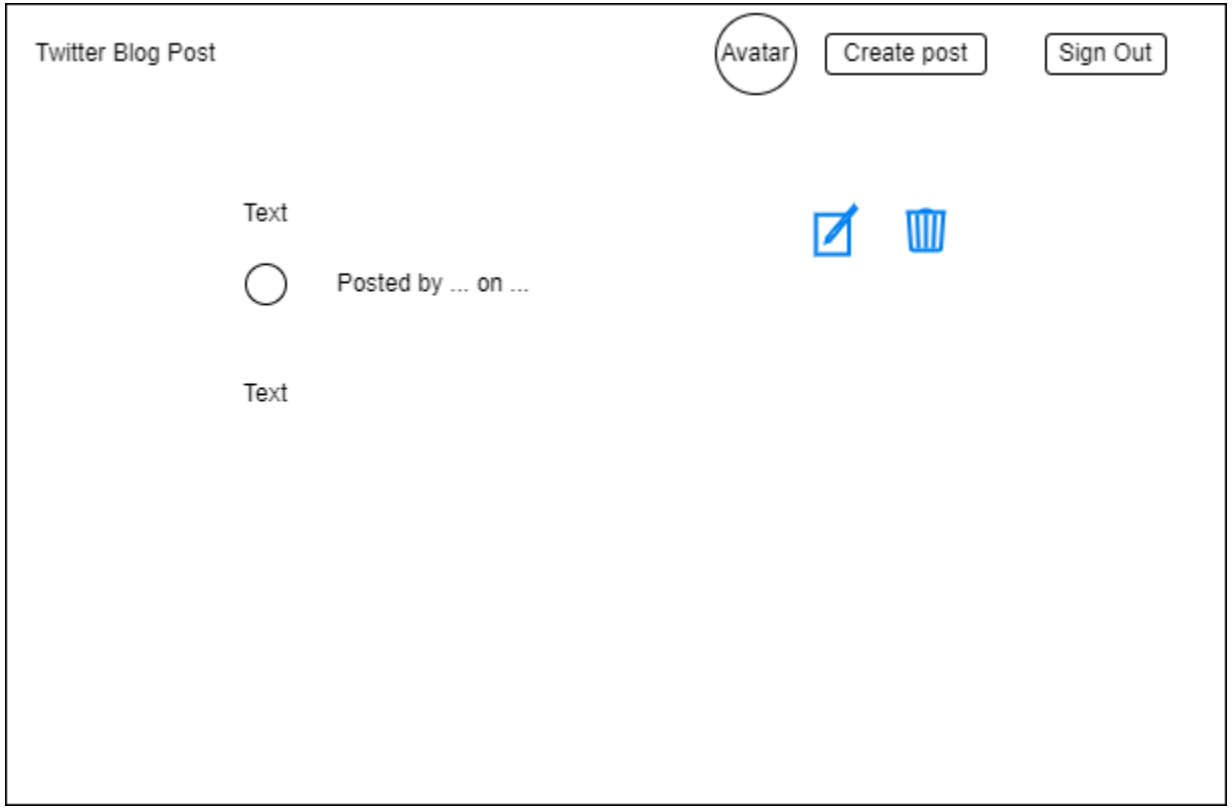

Figure 10: View post page

- Edit post page: A post page will appear with content data of the post we created previously so we can update the content.

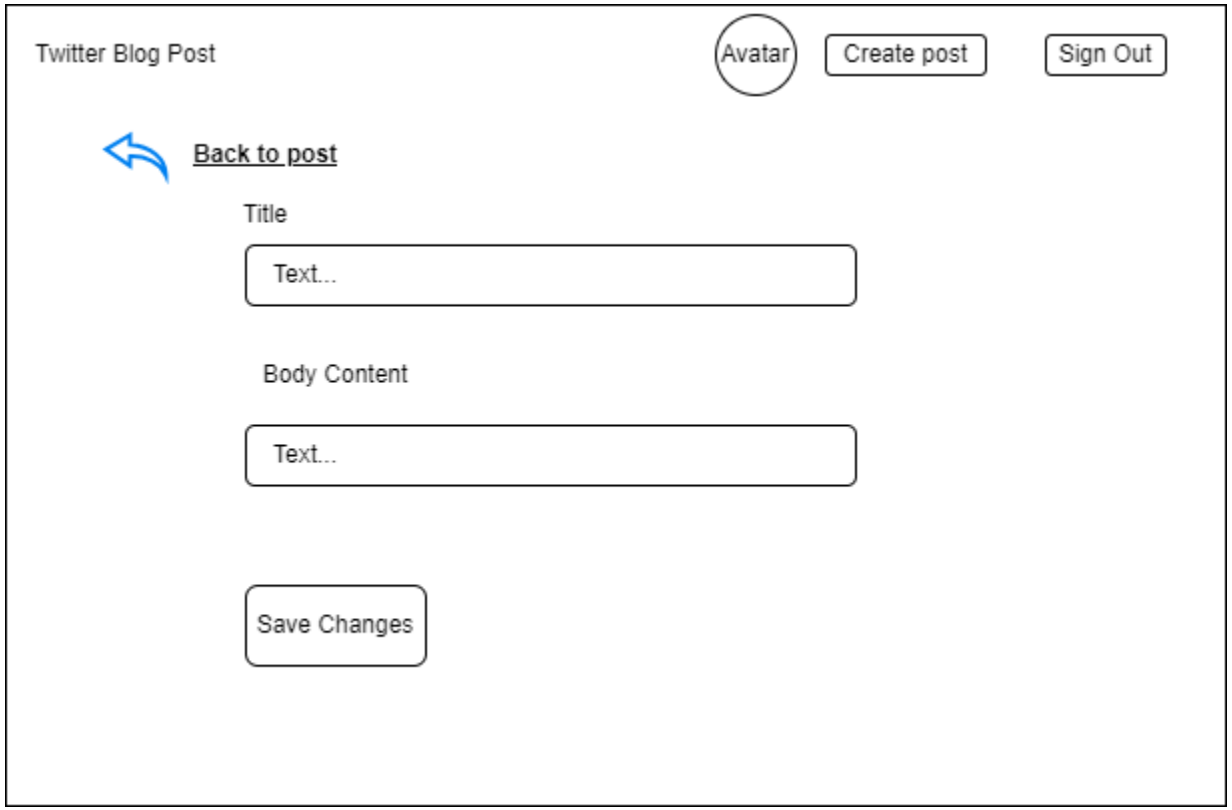

Figure 11: Edit post page

- Avatar page: Change your profile picture to enrich your website.

Figure 12: Avatar page

3. System Implementation (Individual)

3.1. Source code

a. Introduction MVC design pattern in Laravel

MVC is a web application architectural design model of the Laravel framework, in which the application is divided into three main components: Model, View and Controller. First Model is responsible for processing data and storing data in the database. Next, the View is responsible for displaying data to the user. Finally, the Controller is responsible for controlling the application flow and handling requests from users. This MVC design pattern is widely used in many web projects around the world. The advantages of this model include ease of maintenance, expansion, reuse of components, ease of change and upgrade. On the contrary, it can be more complicated than other models because it has many internal components.

b. Attach project structure

- Include migrations file and MVC

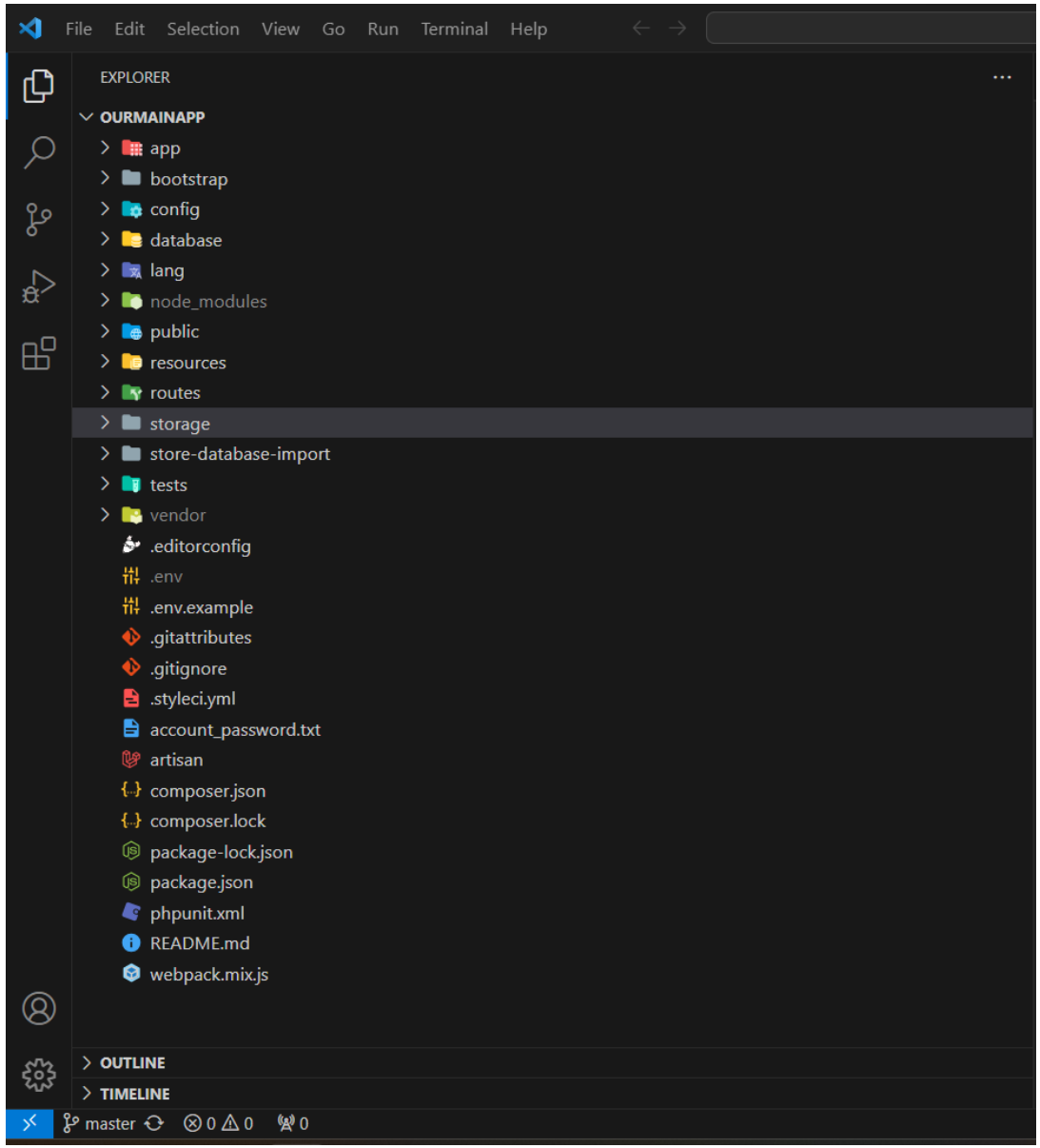
File
Edit
Selection
View
Go
Run
Terminal
Help
EXPLORER
OURMAINAPP
app
bootstrap
config
database
lang
node_modules
public
resources
routes
storage
store-database-import
tests
vendor
.editorconfig
.env
.env.example
.gitattributes
.gitignore
.styleci.yml
account_password.txt
artisan
composer.json
composer.lock
package-lock.json
package.json
phpunit.xml
README.md
webpack.mix.js
OUTLINE
TIMELINE
master
0
0
0

Figure 13: MVC

- User code

- Model: The User model includes fields such as 'username', 'email', 'password'. This is important data information to be able to create a user account and especially in it there is an avatar() function that makes the path for your photos to go to the correct folder or the temporary photo is the photo. saved outside the 'public' folder. There are also 'hasManyThrough' relationship functions like feedPosts() to be able to get the posts of people whose accounts we have followed. The followers() function has a 'hasMany' relationship that will retrieve information about the accounts that are following our account. The followingTheseUsers() function has a 'hasMany' relationship to get information about the accounts we follow. Finally, the posts() function has a 'hasMany' relationship to get all the posts of the currently logged in user account.

User.php × profile-following.blade.php

app > Models > User.php

```php
<?php

namespace App\Models;

use Laravel\Sanctum\HasApiTokens;
use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Casts\Attribute;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'username',
        'email',
        'password',
    ];

    protected function avatar(): Attribute {
        return Attribute::make(get: function($value) {
            return $value ? '/storage/avatars/' . $value : '/fallback-avatar.jpg';
        });
    }

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
```

```php
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function feedPosts() {
        return $this->hasManyThrough(Post::class, Follow::class, 'user_id', 'user_id', 'id', 'followeduser');
    }

    public function followers() {
        return $this->hasMany(Follow::class, 'followeduser');
    }

    public function followingTheseUsers() {
        return $this->hasMany(Follow::class, 'user_id');
    }

    public function posts() {
        return $this->hasMany(Post::class, 'user_id'); // one - many
    }
}
```

Figure 14: Model User

- Controller: In UserController, the user will first have to log in and register an account to access the web system. The fields in the function are tightly linked and have validation to provide rules for appropriate account naming. The logout() function allows us to directly log out of the account we are using and go to the main page with login and registration. There are also functions such as showing posts of logged in user accounts profile(), showing user accounts following our account profileFollowers(), and showing user accounts we follow profileFollowing( ). Finally, the storeAvatar() function helps us change the avatar as desired.

UserController.php

app > Http > Controllers > UserController.php

```php
<?php

namespace App\Http\Controllers;

use App\Models\User;
use App\Models\Follow;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;
use Illuminate\Support\Facades\View;
use Intervention\Image\Facades\Image;
use Illuminate\Support\Facades\Storage;

class UserController extends Controller
{
    // storeAvatar
    public function storeAvatar(Request $request) {
        $request->validate([
            'avatar' => 'required|image|max:3000' // max:3000 KB
        ]);

        $user = auth()->user();

        $filename = $user->id . '-' . uniqid() . '.jpg';
        // uniqid(): generated string of text


        // $request->file('avatar')->store('public/avatars');

        // The third 'Intervention Image Package': edit image
        $imgData = Image::make($request->file('avatar'))->fit(120)->encode('jpg');
        Storage::put('public/avatars/' . $filename, $imgData);

        $oldAvatar = $user->avatar;

        $user->avatar = $filename;
        $user->save(); // save image in database

```

```php
        if ($oldAvatar != "/fallback-avatar.jpg") {
            Storage::delete(str_replace("/storage/", "public/", $oldAvatar));
        } // delete previous avatar

        return back()->with('success', "Congrats on the new avatar.");

    }

    // showAvatarForm
    public function showAvatarForm() {
        return view('avatar-form');
    }

// ----------------------------------------------------------------------------------------

    // getSharedData
    private function getSharedData($user) {
        $currentlyFollowing = 0;

        if (auth()->check()) {
            $currentlyFollowing = Follow::where([['user_id', '=', auth()->user()->id], ['followeduser', '=', $user->id]])->count();
            // If the user is following $user, then it will count the number of rows in the database table using the count() function.
            // $currentlyFollowing = numbers of rows
        }

        // If View:share called, it will display data with all laravel views.
        // Separating the data will help the user to get or not get that data to avoid focusing on a duplicate view when separating that vie
        View::share('sharedData', ['currentlyFollowing' => $currentlyFollowing, 'avatar' => $user->avatar,
        'username' => $user->username, 'posts' => $user->posts()->latest()->get(), 'postCount' => $user->posts()->count(),
        'followerCount' => $user->followers()->count(), 'followingCount' => $user->followingTheseUsers()->count()]);
    }

    // profile
    public function profile(User $user) {
        $this->getSharedData($user);
        return view('profile-posts', ['posts' => $user->posts()->latest()->get()]);
    }
```

```php

    // profileFollowers
    public function profileFollowers(User $user) {
        $this->getSharedData($user);
        return view('profile-followers', ['followers' => $user->followers()->latest()->get()]);
    }

    // profileFollowing
    public function profileFollowing(User $user) {
        $this->getSharedData($user);
        return view('profile-following', ['following' => $user->followingTheseUsers()->latest()->get()]);
    }

// ----------------------------------------------------------------------------------------
    // showCorrectHomepage
    public function showCorrectHomepage()
    {
        if (auth()->check()) { // Authenticate.php
            // check true condition in login function
            return view('homepage-feed', ['posts' => auth()->user()->feedPosts()->latest()->paginate(5)]);
            // paginate(5) limiting the results to 5 per page
        }
        else {
            return view('homepage');
        }
    }

    // logout
    public function logout()
    {
        auth()->logout(); // verify correct user account want to logout
        // https://stackoverflow.com/questions/43585416/how-to-logout-and-redirect-to-login-page-using-laravel-5-4
        return redirect('/')->with('success', 'You are now logged out!');
    }

    // login
    public function login(Request $request)
```

```php
    {
        $incomingFields = $request->validate([
            'loginusername' => 'required',
            'loginpassword' => 'required'
        ]);

        if (auth()->attempt(['username' => $incomingFields['loginusername'], 'password' => $incomingFields['loginpassword']])) {
            $request->session()->regenerate(); // @session directive is a Blade directive used within your Blade views to work with session
            return redirect('/')->with('success', 'You have successfully logged in!');
        }
        else {
            return redirect('/')->with('failure', 'Invalid login!');
        }

    }

    // register
    public function register(Request $request)
    {
        $incomingFields = $request->validate([
            'username' => ['required', 'min:3', 'max:20', Rule::unique('users', 'username')],
            'email' => ['required', 'email', Rule::unique('users', 'email')],
            'password' => ['required', 'min:8', 'confirmed']
        ]);

        $incomingFields['password'] = bcrypt($incomingFields['password']); // encrypt password by 'bcrypt'

        $user = User::create($incomingFields);
        auth()->login($user);
        return redirect('/')->with('success', 'Thank you for creating an account!');
    }
}
```

Figure 15: UserController

- Post code

- Model: In the Post model we will have two fields 'title' and 'body' to be able to create a complete post. There is also a function user() with the 'belongsTo' relationship to determine the dependency relationship with the User table through 'user_id'.

Post.php

app > Models > Post.php

```php
<?php

namespace App\Models;

use Laravel\Scout\Searchable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Post extends Model
{
    use Searchable; // Laravel Scout
    use HasFactory;

    protected $fillable = ['title', 'body', 'user_id']; // Put all entries containing data into each column in the database table afte

    public function toSearchableArray() {
        return [
            'title' => $this->title,
            'body' => $this->body
        ];
    }

    public function user()
    {
        return $this->belongsTo(User::class, 'user_id'); // Reference the '$post' field containing the id account user (single-post.bl
        // 'user_id = auth()->id() in storeNewPost function (PostController.php)
    }

}
```

Figure 16: Post model

- Controller: In PostController, there will be functions that can save the data information the user has entered to push it to the database and from there retrieve it and bring it into the user's website. In addition, there will be CRUD functions for each user's post.

PostController.php × PostPolicy.php

app > Http > Controllers > PostController.php

```php
<?php

namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Support\Str;
use Illuminate\Http\Request;

class PostController extends Controller
{

    // actuallyUpdate: Update post
    public function actuallyUpdate(Post $post, Request $request) {
        $incomingFields = $request->validate([
            'title' => 'required',
            'body' => 'required'
        ]);

        $incomingFields['title'] = strip_tags($incomingFields['title']); // strip_tags will return a string delimited from HTML or PHP
        $incomingFields['body'] = strip_tags($incomingFields['body']);

        $post->update($incomingFields);

        return back()->with('success', 'Post successfully update.');
    }

    // showEditForm: final update post
    public function showEditForm(Post $post) {
        return view('edit-post', ['post' => $post]);
    }

    // delete: delete post
    public function delete(Post $post) {
        $post->delete();
        return redirect('/profile/' . auth()->user()->username)->with('success', 'Post successfully deleted.');
    }

```

```php
    // viewSinglePost
    public function viewSinglePost(Post $post) // $post contains the id value for each post created and must match the incoming variab
    {
        return view('single-post', ['post' => $post]);
    }

    // storeNewPost: create and save post
    public function storeNewPost(Request $request) // create a post
    {
        $incomingFields = $request->validate([
            'title' => 'required',
            'body' => 'required'
        ]);

        $incomingFields['title'] = strip_tags($incomingFields['title']);
        $incomingFields['body'] = strip_tags($incomingFields['body']);
        $incomingFields['user_id'] = auth()->id(); // id of the current user account and assign this id value to $incomingFields['user

        // $incomingFields variable is holding data in array
        $newPost = Post::create($incomingFields); // Save entry to database. Post.php(Model) -> $fillable

        return redirect("/post/{$newPost->id}")->with('success', 'New post successfully created.');
    }

    // showCreateForm
    public function showCreateForm()
    {
        return view('create-post');
    }
}
```

Figure 17: PostController

- Follow code

- Model: In the Follow model, we will have two functions with the relationship "belongsTo" to show the dependence on the User table through 'user_id' and 'followeduser' to be able to determine who is following us or who we are following.

PostController.php | Follow.php

app > Models > Follow.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Follow extends Model
{
    use HasFactory;

    public function userDoingTheFollowing() {
        return $this->belongsTo(User::class, 'user_id');
    } // 'belonging' relationship

    public function userBeingFollowed() {
        return $this->belongsTo(User::class, 'followeduser');
    }

}

```

Figure 18: Follow model

- Controller: In FollowController, there will be two functions representing two functions: create and remove follow other user accounts.

PostController.php | FollowController.php

app > Http > Controllers > FollowController.php

```php

use App\Models\User;
use App\Models\Follow;
use Illuminate\Http\Request;

class FollowController extends Controller
{
    // createFollow
    public function createFollow(User $user) {
        // you cannot follow yourself
        if ($user->id == auth()->user()->id) {
            return back()->with('failure', 'You cannot follow yourself.');
        }

        // you cannot follow someone you're already following
        $existCheck = Follow::where([['user_id', '=', auth()->user()->id], ['followeduser', '=', $user->id]])->count(); // count retur

        if ($existCheck) {
            return back()->with('failure', 'You are already following that user.');
        }

        // new follow if u don't already follow that person.
        $newFollow = new Follow;
        $newFollow->user_id = auth()->user()->id;
        $newFollow->followeduser = $user->id;
        $newFollow->save(); // model

        return back()->with('success', 'User successfully followed.');
    }

    // removeFollow
    public function removeFollow(User $user) {
        Follow::where([['user_id', '=', auth()->user()->id], ['followeduser', '=', $user->id]])->delete();
        return back()->with('success', 'User successfully unfollowed.');
    }
}

```

Figure 19: FollowController

- View: In the view, it will represent all that the user can see and the information we have entered.

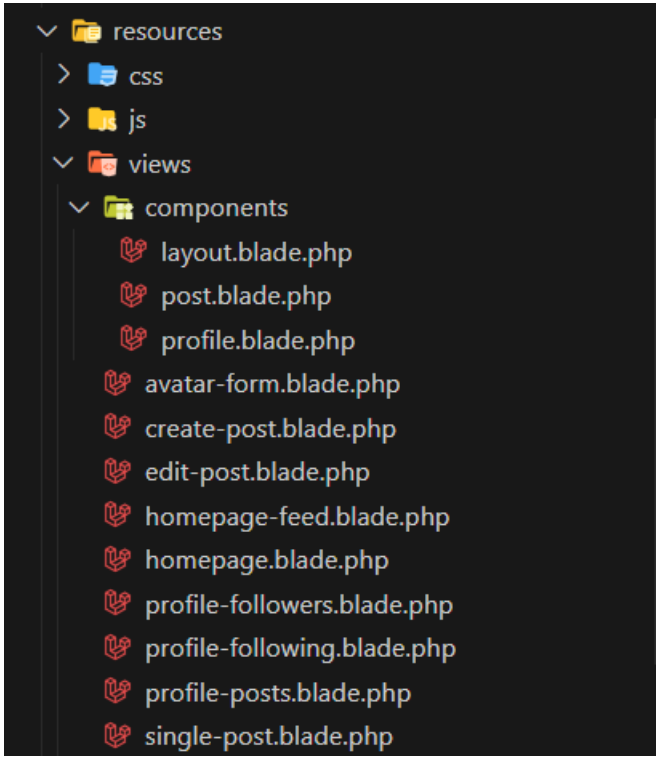

Figure 20: View

c. How can develop a Laravel project

To be able to develop a Laravel project, it will include the following steps:

- Install Composer
- Install Composer environment in windows operating system
- Install the MySQL application
- Install MySQL environment in windows operating system
- Install php version and php environment in windows operating system

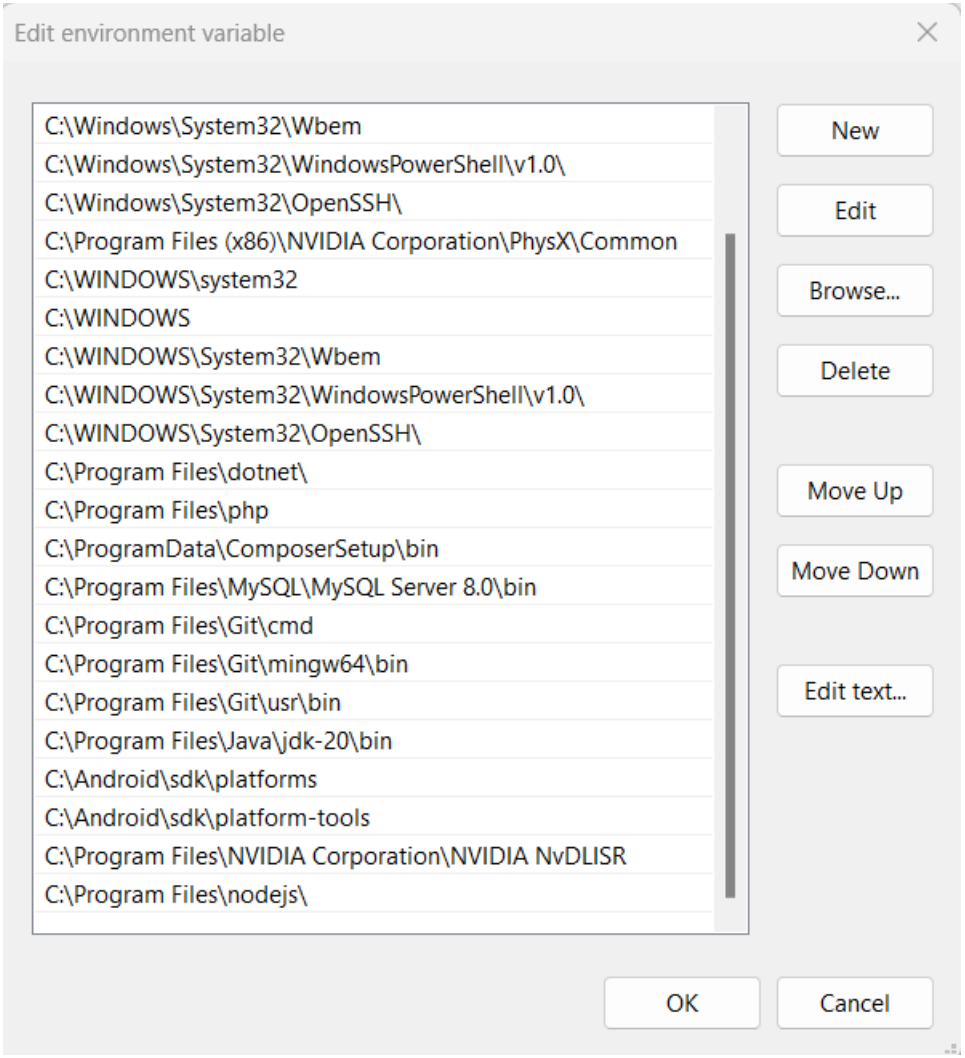

Figure 21: Composer, php and MySQL environment

- After installing the above step, use Command Prompt to create a Laravel project: composer create-project laravel/laravel my-project

- Configure the database through the .env file include lines number 14, 15, 16

PostController.php | .env

.env

```
1  APP_NAME=Laravel
2  APP_ENV=local
3  APP_KEY=base64:LTnVCXaQeGRFehrpU1wWL9G7QdltfKujf97dgsd5ymE=
4  APP_DEBUG=true
5  APP_URL=http://localhost
6
7  LOG_CHANNEL=stack
8  LOG_DEPRECATIONS_CHANNEL=null
9  LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=ourlaravelapp
15 DB_USERNAME=root
16 DB_PASSWORD=root
17
18 BROADCAST_DRIVER=log
19 CACHE_DRIVER=file
20 FILESYSTEM_DISK=local
21 QUEUE_CONNECTION=sync
22 SESSION_DRIVER=file
23 SESSION_LIFETIME=120
24
25 MEMCACHED_HOST=127.0.0.1
26
27 REDIS_HOST=127.0.0.1
28 REDIS_PASSWORD=null
29 REDIS_PORT=6379
30
31 MAIL_MAILER=smtp
32 MAIL_HOST=mailhog
33 MAIL_PORT=1025
34 MAIL_USERNAME=null
35 MAIL_PASSWORD=null
36 MAIL_ENCRYPTION=null
37 MAIL_FROM_ADDRESS=null
```

Figure 22: .env file

- Use cmd to migrate all changes into the database: php artisan migrate

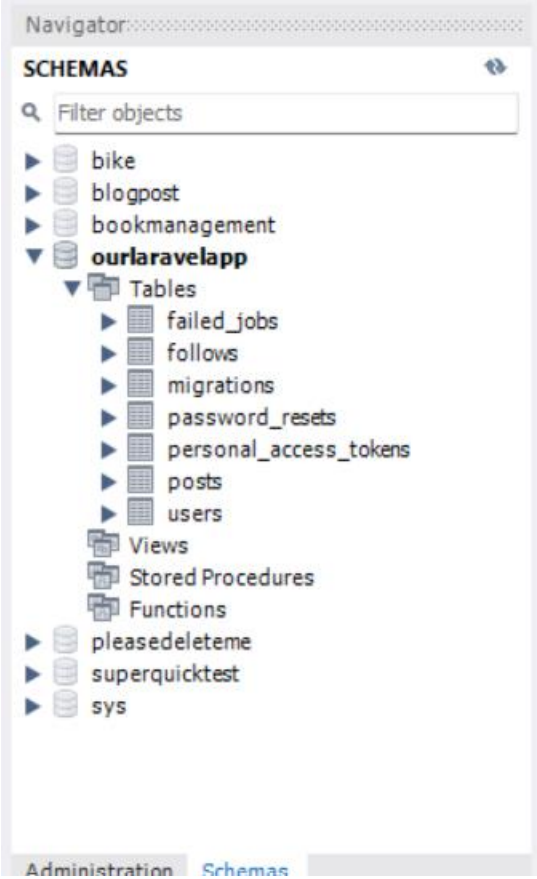

Figure 23: Database MySQL

- Run web: php artisan serve

3.2. Web screenshots

- Login/Register

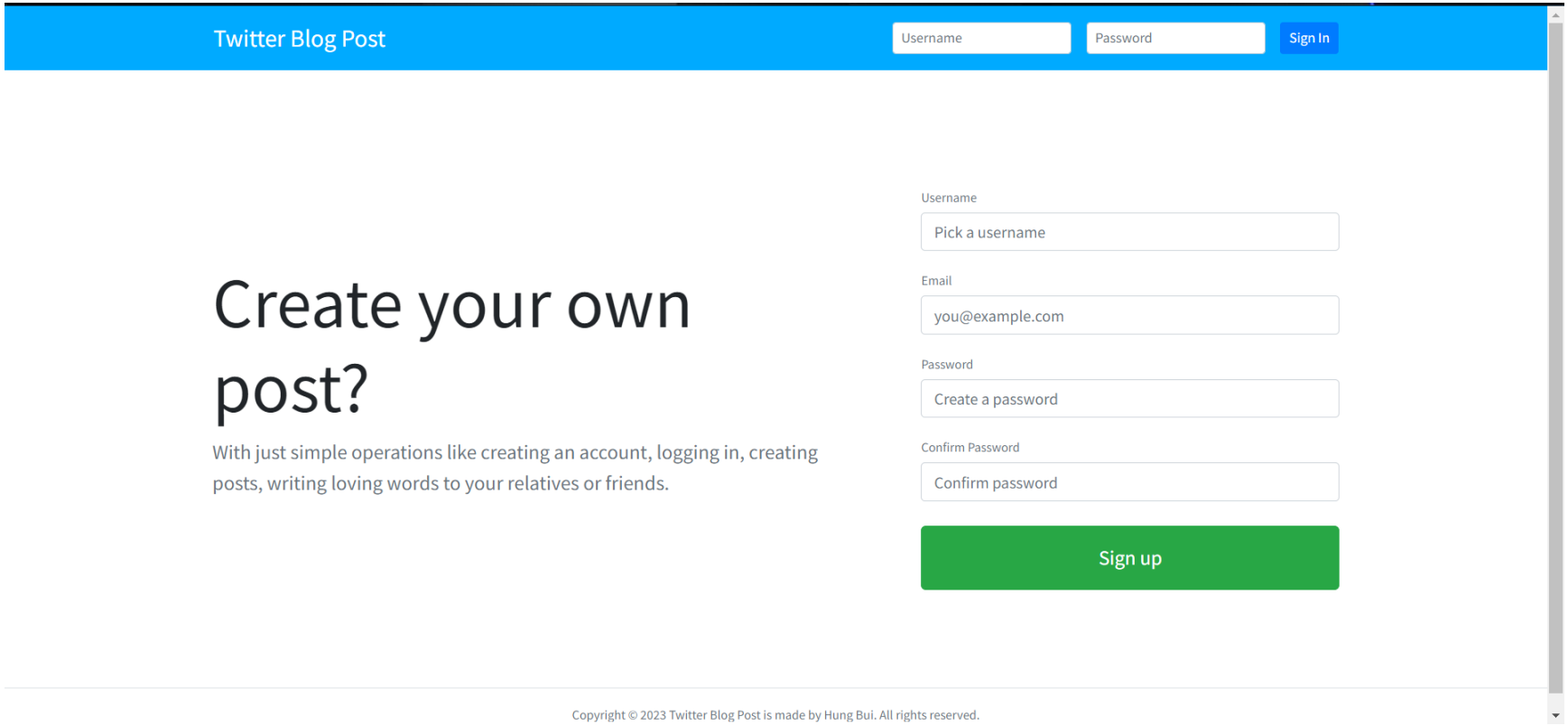

Figure 24: Homepage

- Homepage-feed

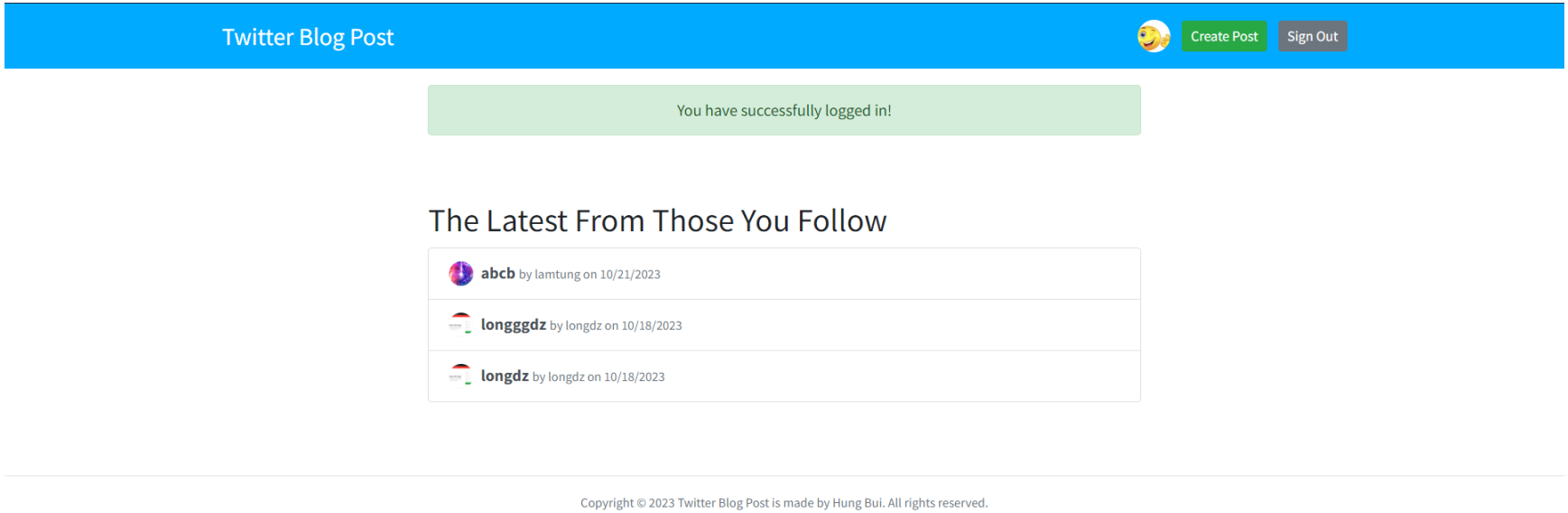

Figure 25: Homepage-feed

- Create-post page with function create a new post

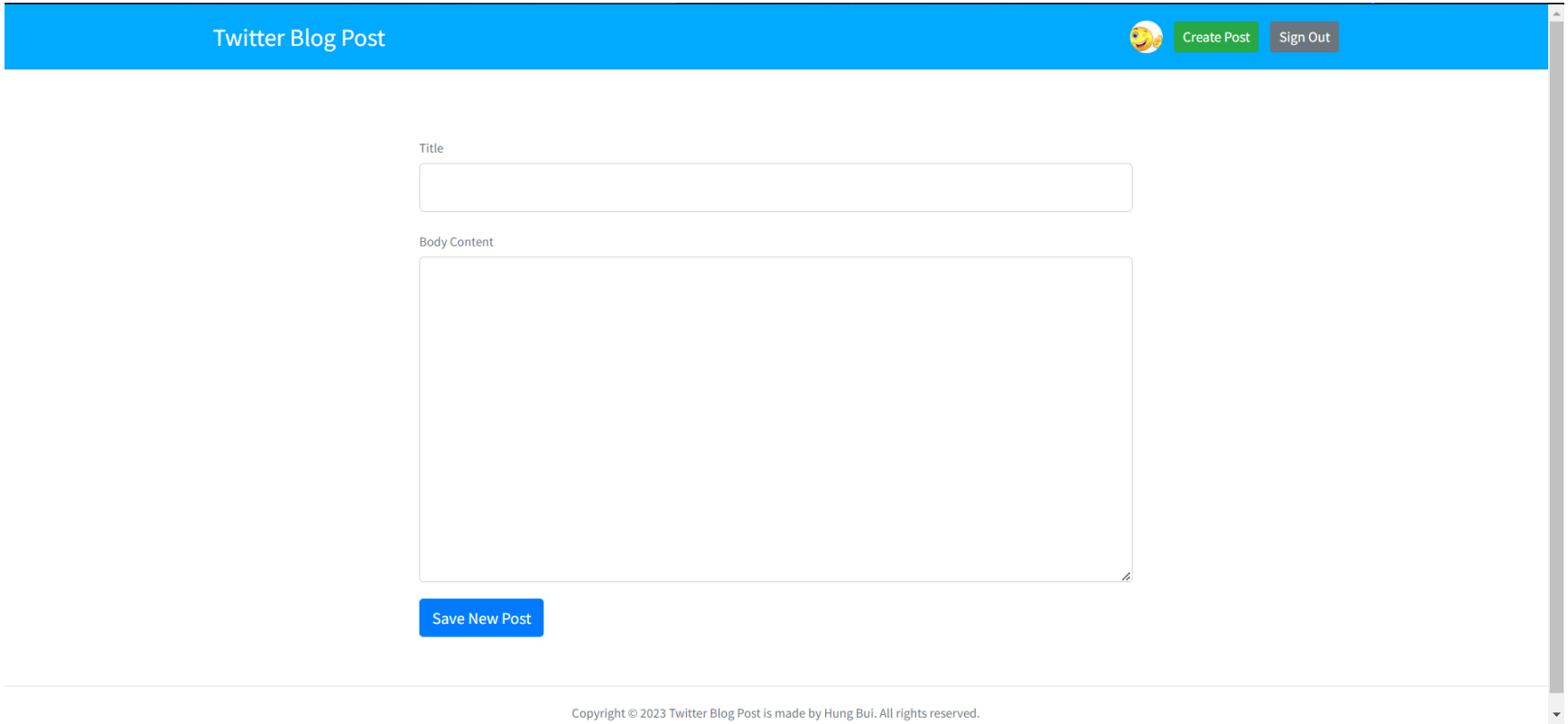

Figure 26: Create-post page

- View-post page with function update, delete a post

Figure 27: View-post page

- Update-post page:

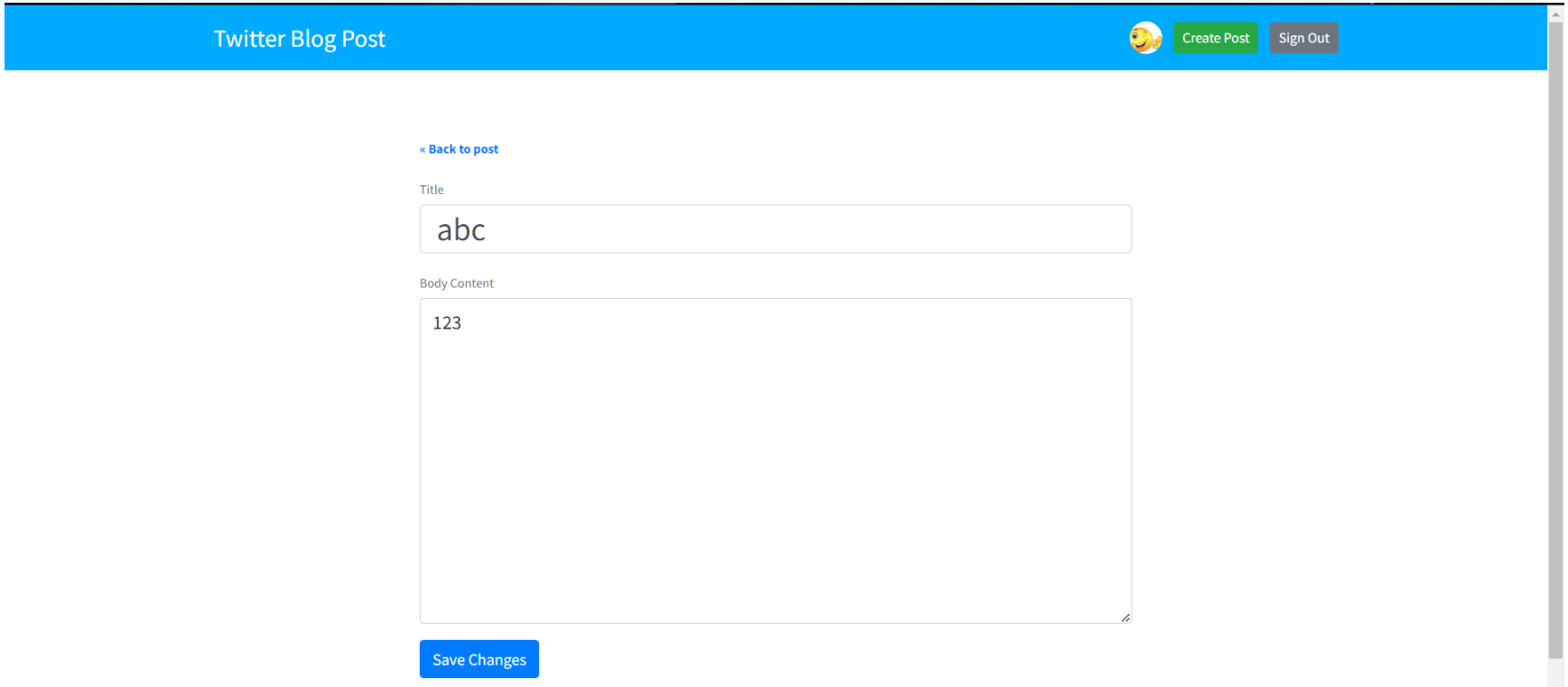

Figure 28: Update-post page

- Profile-post page

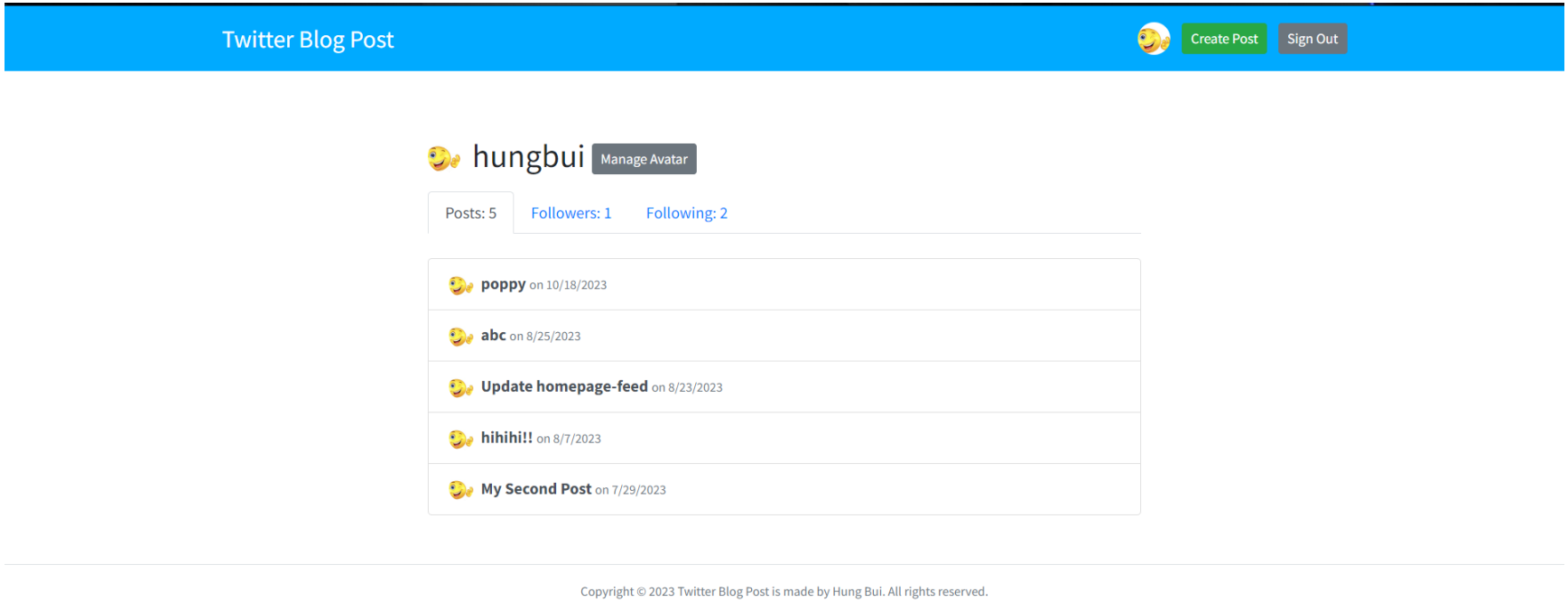

Figure 29: Profile-post page

- Follower page

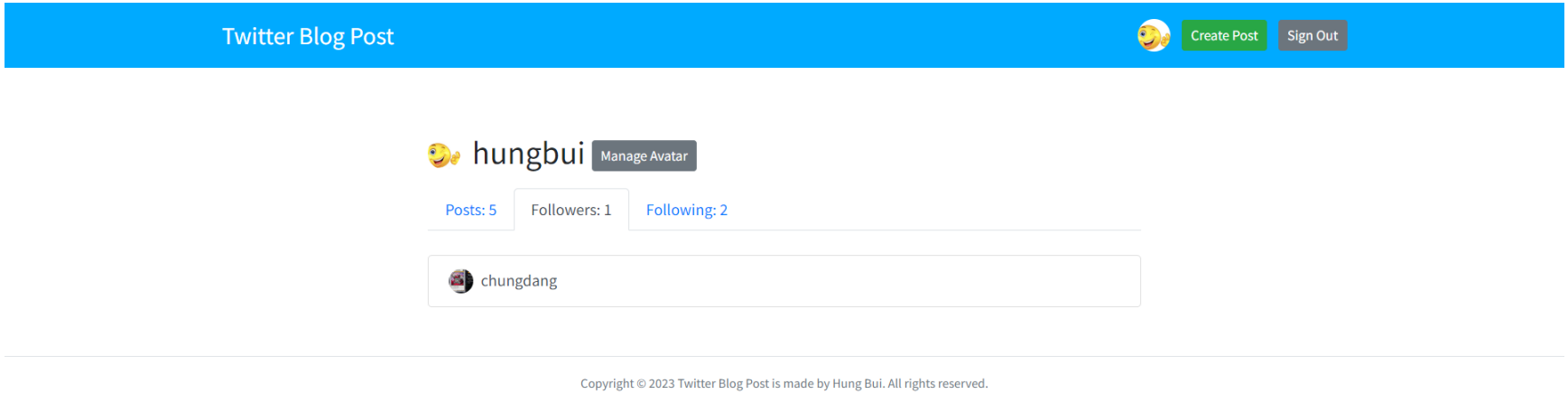

Figure 30: Following page

- Following page

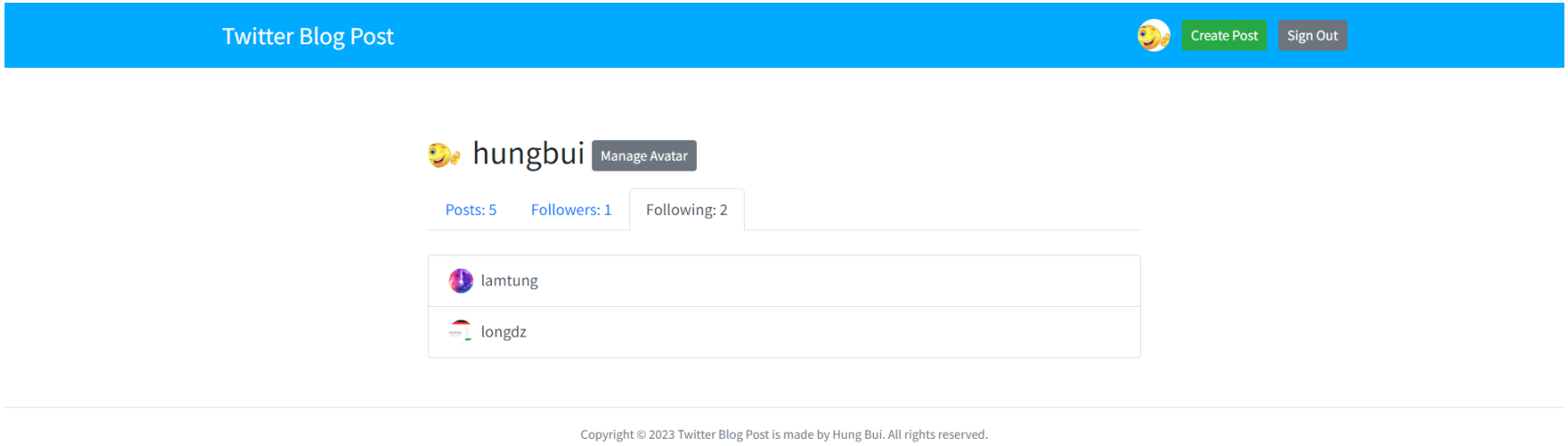

Figure 31: Following page

- Avatar page

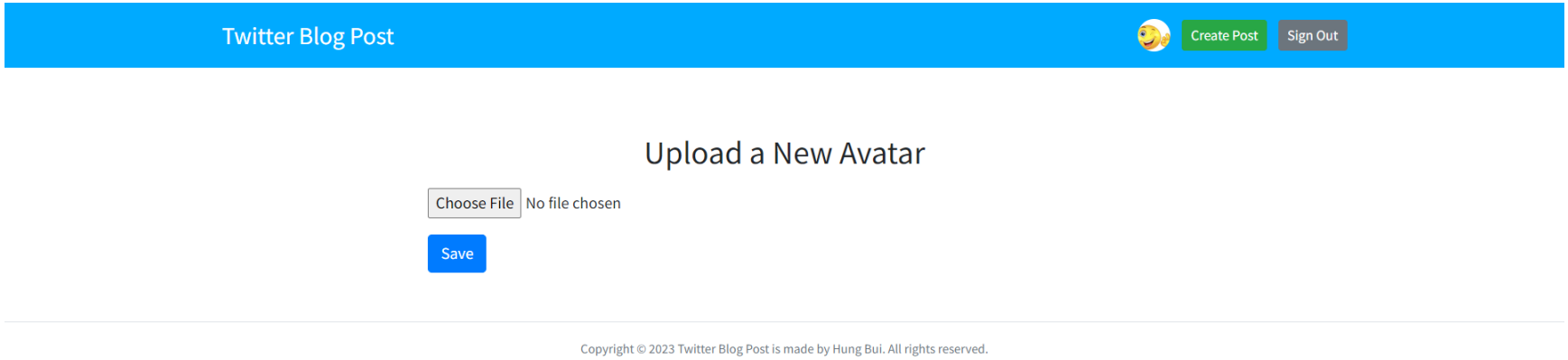

Figure 32: Avatar page

4. Conclusion (Individual)

4.1. Advantage of website

- The minimalist user interface is not too colorful, bringing a pleasant feeling to the user.

- Functions operate normally without errors.

- The website's query speed is stable.

4.2. Disadvantages of website

- The structure of the functions on the website has not been optimized, so the user experience is not truly perfect.

- The functions are not really groundbreaking for users.

- The user interface is minimalist but does not really make users feel that the website is simple.

4.3. Lesson learnt

- Knowing how to use Laravel framework and MCV model in web design makes my web work easier, more convenient and scientific.

- Understand Laravel framework functions and commands.

- Learn how to connect and create database in Laravel easily with command

- Learn how to use GitHub for convenient teamwork and source code management

- Integrate security features such as authentication, encryption, etc. to protect applications from attacks.

- Regularly test and maintain the application to ensure it operates stably and safely.

4.4. Future Improvements

- Unique user interface while still maintaining user convenience.

- Upgrading existing functions makes the user experience feel better.

- Develop more unique features to make users more interested in the website.

- Limit potential errors that occur during web development.

5. Appendix (Group)

5.1. A1 group member list and role (show as table)

No	Name	Role
1	Bui Duy Hung	Front-end, Back-end, Entity: User page, Follow page
2	Nguyen Thanh Tung	Front-end, Back-end, Entity: Post page

Table 2: Group member

5.2. A2 link to code on Github (set public access)

- Link github Project Web: Click here