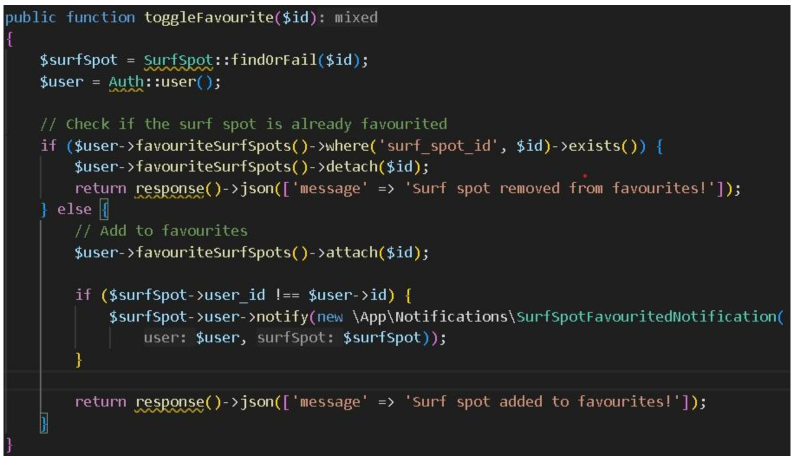
# Owain Peters – Web Application

## Implemented Code

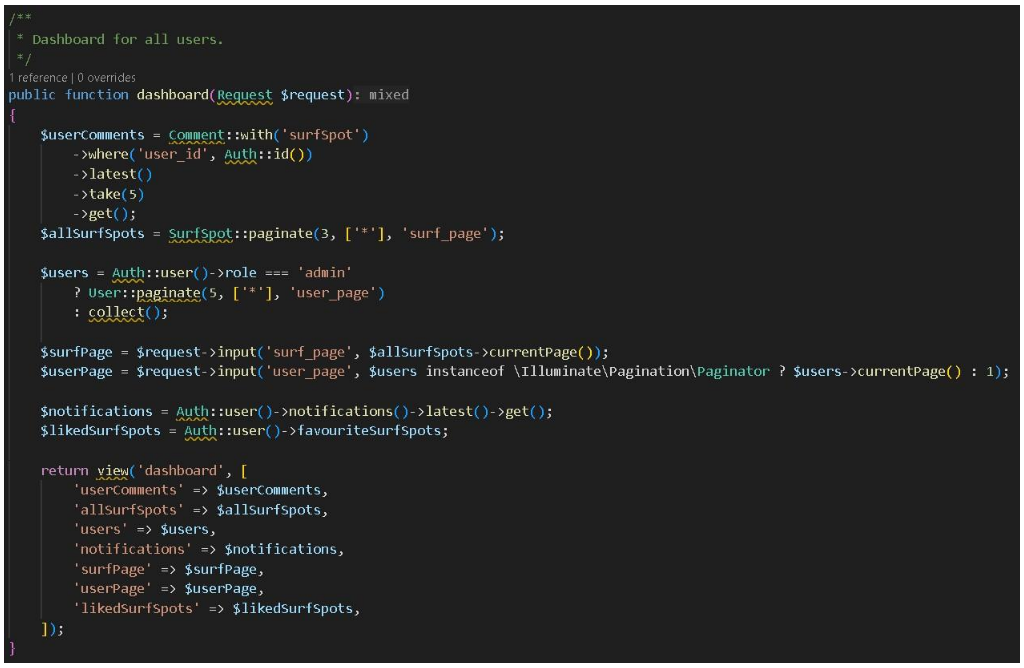
toggleFavourite function in SurfSpotController:
This handles the favorite feature where users can favorite or unfavorite a surf spot, making it easier to see their favorites without having to scroll through each one, increasing the efficiency of the web page.  This is the function code below:



```
public function toggleFavourite($id): mixed
{
    $surfSpot = SurfSpot::findOrFail($id);
    $user = Auth::user();

    // Check if the surf spot is already favourited
    if ($user->favouriteSurfSpots()->where('surf_spot_id', $id)->exists()) {
        $user->favouriteSurfSpots()->detach($id);
        return response()->json(['message' => 'Surf spot removed from favourites!']);
    } else {
        // Add to favourites
        $user->favouriteSurfSpots()->attach($id);

        if ($surfSpot->user_id !== $user->id) {
            $surfSpot->user->notify(new \App\Notifications\SurfSpotFavouritedNotification(
                user: $user, surfSpot: $surfSpot));
        }

        return response()->json(['message' => 'Surf spot added to favourites!']);
    }
}
```

The dashboard functions in SurfSpotController:

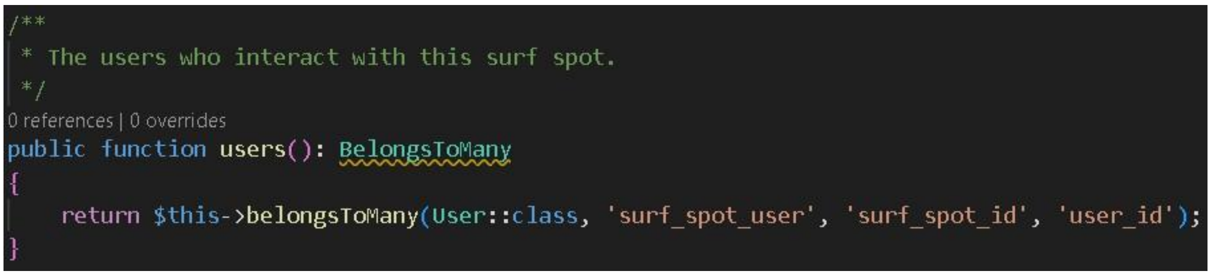
This function handles the information displayed on the dashboard page.  It fetches data for the authenticated user, including their comments, notifications, and favorite surf spots.  In addition to this, it checks the user's role, handles the pagination of surf spots and users for the entire dashboard page, and provides each notification that must be displayed on the dashboard page:



```php
/**
 * Dashboard for all users.
 */
1 reference | 0 overrides
public function dashboard(Request $request): mixed
{
    $userComments = Comment::with('surfSpot')
        ->where('user_id', Auth::id())
        ->latest()
        ->take(5)
        ->get();
    $allSurfSpots = SurfSpot::paginate(3, ['*'], 'surf_page');

    $users = Auth::user()->role === 'admin'
        ? User::paginate(5, ['*'], 'user_page')
        : collect();

    $surfPage = $request->input('surf_page', $allSurfSpots->currentPage());
    $userPage = $request->input('user_page', $users instanceof \Illuminate\Pagination\Paginator ? $users->currentPage() : 1);

    $notifications = Auth::user()->notifications()->latest()->get();
    $likedSurfSpots = Auth::user()->favouriteSurfSpots;

    return view('dashboard', [
        'userComments' => $userComments,
        'allSurfSpots' => $allSurfSpots,
        'users' => $users,
        'notifications' => $notifications,
        'surfPage' => $surfPage,
        'userPage' => $userPage,
        'likedSurfSpots' => $likedSurfSpots,
    ]);
}
```

Users() function **–** SurfSpot Model **:**

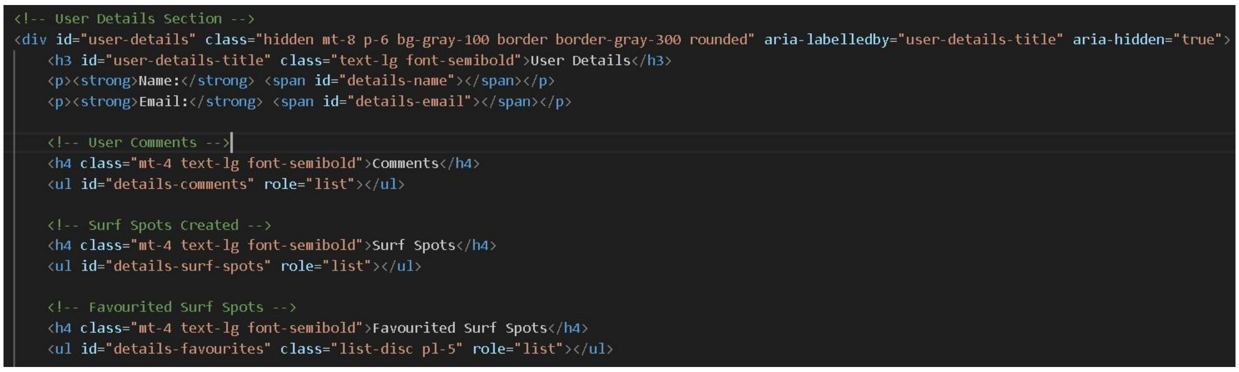
This shows a BelongsToMany relationship between the User, Surf Spot, and the Surf Spot User which is the table that presents the surf spots that have been favored by users:



```php
/**
 * The users who interact with this surf spot.
 */
0 references | 0 overrides
public function users(): BelongsToMany
{
    return $this->belongsToMany(User::class, 'surf_spot_user', 'surf_spot_id', 'user_id');
}
```

User Details Section – dashboard.blade:

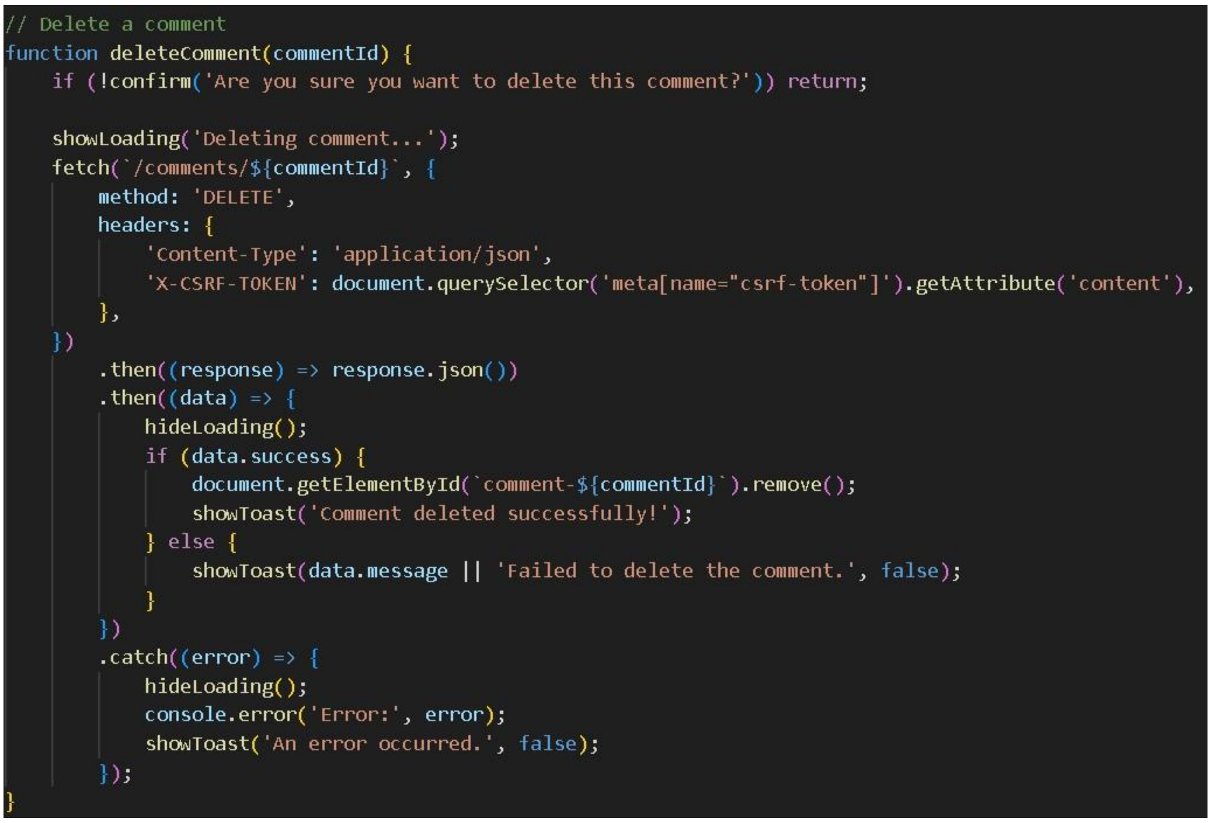
This section within the dashboard.blade file displays the user details, such as their name, email address, the comments and surf spots they've posted, in addition to their favorite surf spots. This is shown in a dropdown section when the user's name is clicked from the user table which is displayed for the admin users. This is to ensure a clean page so it doesn't feel too cluttered with information, and the user can drop down the information if needed:



```html
<!-- User Details Section -->
<div id="user-details" class="hidden mt-8 p-6 bg-gray-100 border border-gray-300 rounded" aria-labelledby="user-details-title" aria-hidden="true">
    <h3 id="user-details-title" class="text-lg font-semibold">User Details</h3>
    <p><strong>Name:</strong> <span id="details-name"></span></p>
    <p><strong>Email:</strong> <span id="details-email"></span></p>

    <!-- User Comments -->
    <h4 class="mt-4 text-lg font-semibold">Comments</h4>
    <ul id="details-comments" role="list"></ul>

    <!-- Surf Spots Created -->
    <h4 class="mt-4 text-lg font-semibold">Surf Spots</h4>
    <ul id="details-surf-spots" role="list"></ul>

    <!-- Favourited Surf Spots -->
    <h4 class="mt-4 text-lg font-semibold">Favourited Surf Spots</h4>
    <ul id="details-favourites" class="list-disc pl-5" role="list"></ul>
```

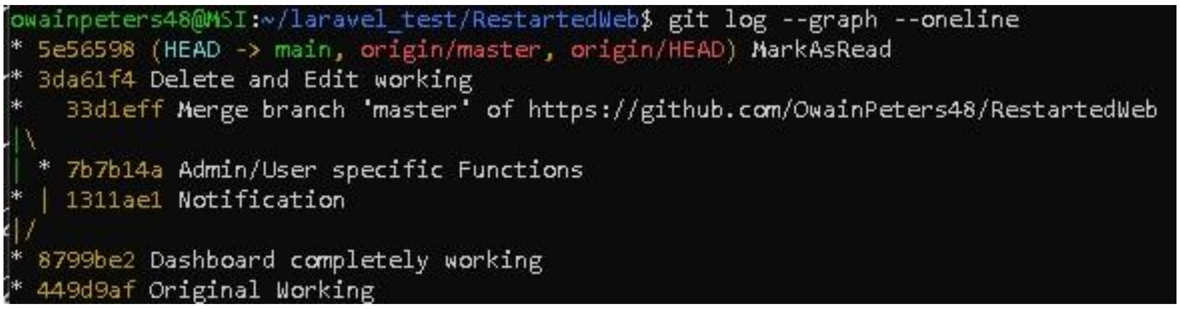
Delete a Comment – dashboard.blade:

This section is JavaScript within the dashboard.blade file. It is the code which is called whenever an admin tries to delete a comment. It used good error handling to carry out the delete function, ensuring easy use for the user:



```
// Delete a comment
function deleteComment(commentId) {
    if (!confirm('Are you sure you want to delete this comment?')) return;

    showLoading('Deleting comment...');
    fetch(`/comments/${commentId}`, {
        method: 'DELETE',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-token"]').getAttribute('content'),
        },
    })
        .then((response) => response.json())
        .then((data) => {
            hideLoading();
            if (data.success) {
                document.getElementById(`comment-${commentId}`).remove();
                showToast('Comment deleted successfully!');
            } else {
                showToast(data.message || 'Failed to delete the comment.', false);
            }
        })
        .catch((error) => {
            hideLoading();
            console.error('Error:', error);
            showToast('An error occurred.', false);
        });
}
```

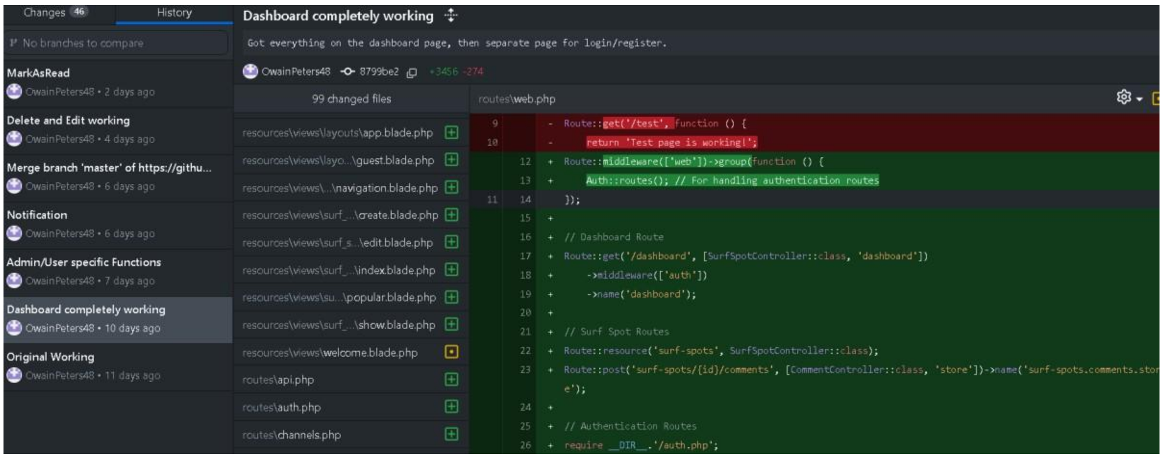
## GitHub



```
owainpeters48@MSI:~/laravel_test/RestartedWeb$ git log --graph --oneline
* 5e56598 (HEAD -> main, origin/master, origin/HEAD) MarkAsRead
* 3da61f4 Delete and Edit working
*   33d1eff Merge branch 'master' of https://github.com/OwainPeters48/RestartedWeb
|\
| * 7b7b14a Admin/User specific Functions
* | 1311ae1 Notification
|/
* 8799be2 Dashboard completely working
* 449d9af Original Working
```

Original Working – This is the original code that I submitted for coursework 1, with the models, controllers, migration tables and seeder/factory files, which gave a simple webpage that involved basic functions such as displaying surf spot information and creating a surf spot.

Dashboard completely working – This is the next version I committed, which includes a working dashboard page, and has an implement Single-Page Application, meaning that all the functions, aside from the login page are displayed on the same dashboard page, without having to access another page to carry out any functions. Additionally, I implemented validation and verification for CRUD functions to ensure error handling and correct user inputs.



Changes 46 | History

No branches to compare

MarkAsRead
OwainPeters48 • 2 days ago

Delete and Edit working
OwainPeters48 • 4 days ago

Merge branch 'master' of https://githu...
OwainPeters48 • 6 days ago

Notification
OwainPeters48 • 6 days ago

Admin/User specific Functions
OwainPeters48 • 7 days ago

Dashboard completely working
OwainPeters48 • 10 days ago

Original Working
OwainPeters48 • 11 days ago

**Dashboard completely working**

```
Got everything on the dashboard page, then separate page for login/register.
```

OwainPeters48 8799be2 +3456 -274

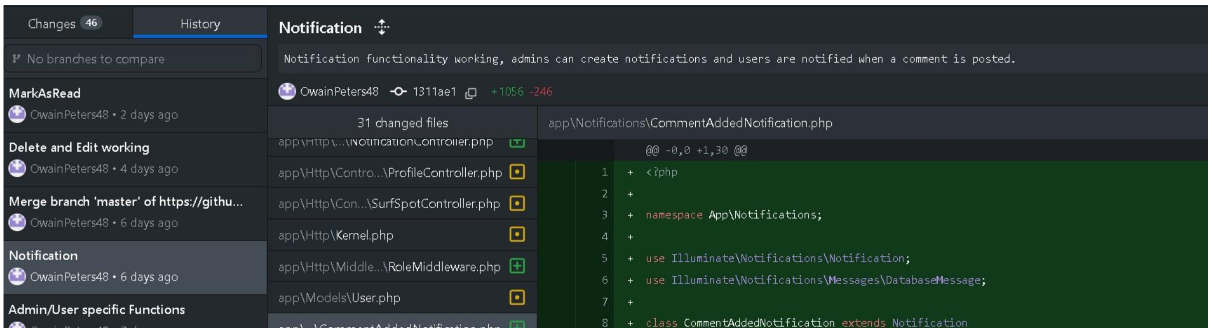
99 changed files

resources\views\layouts\app.blade.php
resources\views\layo...\guest.blade.php
resources\views\...\navigation.blade.php
resources\views\surf_...\create.blade.php
resources\views\surf_s...\edit.blade.php
resources\views\surf_...\index.blade.php
resources\views\su...\popular.blade.php
resources\views\surf_...\show.blade.php
resources\views\welcome.blade.php
routes\api.php
routes\auth.php
routes\channels.php

routes\web.php

```
-  Route::get('/test', function () {
-      return 'Test page is working!';
+  Route::middleware(['web'])->group(function () {
+      Auth::routes(); // For handling authentication routes
   });
+
+  // Dashboard Route
+  Route::get('/dashboard', [SurfSpotController::class, 'dashboard'])
+      ->middleware(['auth'])
+      ->name('dashboard');
+
+  // Surf Spot Routes
+  Route::resource('surf-spots', SurfSpotController::class);
+  Route::post('surf-spots/{id}/comments', [CommentController::class, 'store'])->name('surf-spots.comments.stor
   e');
+
+  // Authentication Routes
+  require __DIR__.'/auth.php';
```

Notification – This version now includes the notification feature, in which I implemented a panel that displays notifications, as well as users being able to post a notification themselves which every other user can see. Users can now favorite or unfavorite a surf spot by clicking the button present under the surf spot information, which appears at the top of the page showing all their favorite surf spots.



Changes 46 | History

No branches to compare

MarkAsRead
OwainPeters48 • 2 days ago

Delete and Edit working
OwainPeters48 • 4 days ago

Merge branch 'master' of https://githu...
OwainPeters48 • 6 days ago

Notification
OwainPeters48 • 6 days ago

Admin/User specific Functions

**Notification**

```
Notification functionality working, admins can create notifications and users are notified when a comment is posted.
```

OwainPeters48 1311ae1 +1056 -246

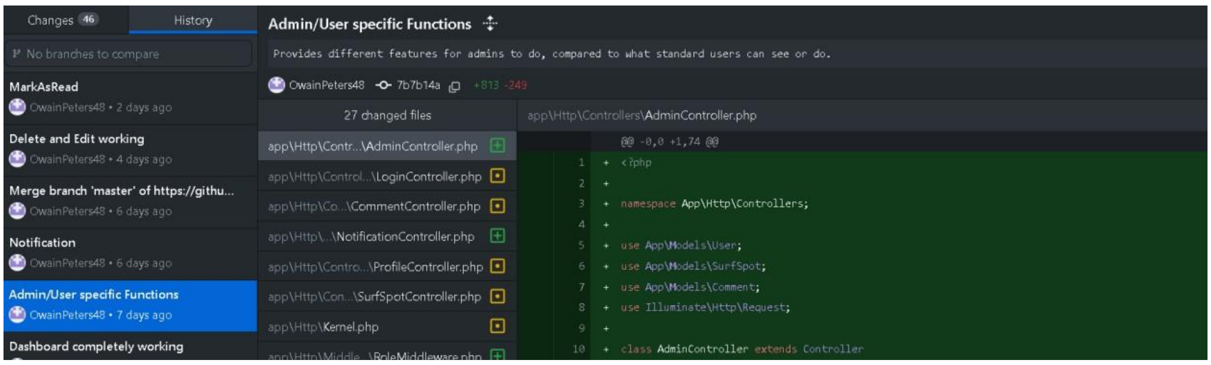
31 changed files

app\Http\...\NotificationController.php
app\Http\Contro...\ProfileController.php
app\Http\Con...\SurfSpotController.php
app\Http\Kernel.php
app\Http\Middle...\RoleMiddleware.php
app\Models\User.php

app\Notifications\CommentAddedNotification.php

```
   @@ -0,0 +1,30 @@
+  <?php
+
+  namespace App\Notifications;
+
+  use Illuminate\Notifications\Notification;
+  use Illuminate\Notifications\Messages\DatabaseMessage;
+
+  class CommentAddedNotification extends Notification
```

Admin/User specific Functions – This commit implements different roles within users and allows them to have different access levels/functions visible to them, so the admin users can carry out more functions than the typical user. I implemented the ability for admins to create surf spots and notifications, in addition to being able to see each user and their profile including their role, the comments they've posted, and the surf spots they have favorited.  The users do not have access to create a surf spot or see the other users or their profile information.



MarkAsRead
OwainPeters48 • 2 days ago
Delete and Edit working
OwainPeters48 • 4 days ago
Merge branch 'master' of https://githu...
OwainPeters48 • 6 days ago
Notification
OwainPeters48 • 6 days ago
Admin/User specific Functions
OwainPeters48 • 7 days ago
Dashboard completely working

**Admin/User specific Functions**

Provides different features for admins to do, compared to what standard users can see or do.

OwainPeters48 7b7b14a +813 -249

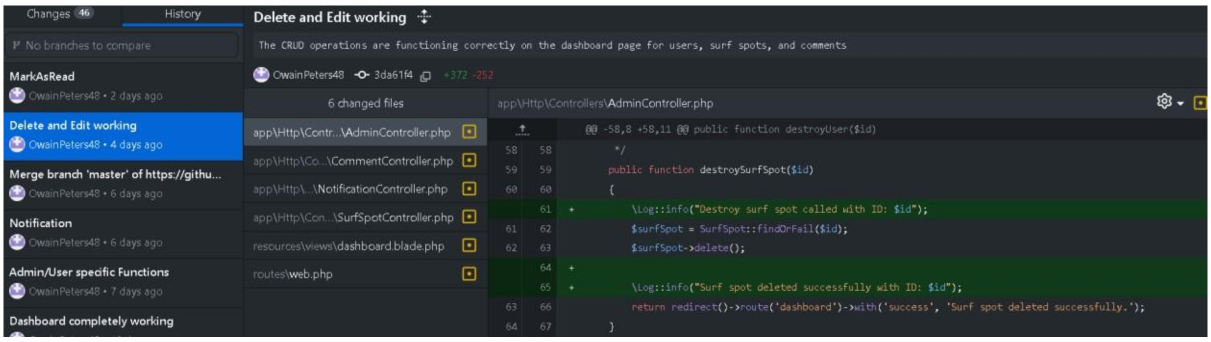
27 changed files

app\Http\Contr...\AdminController.php
app\Http\Control...\LoginController.php
app\Http\Co...\CommentController.php
app\Http\...\NotificationController.php
app\Http\Contro...\ProfileController.php
app\Http\Con...\SurfSpotController.php
app\Http\Kernel.php
app\Http\Middle...\RoleMiddleware.php

app\Http\Controllers\AdminController.php

```
@@ -0,0 +1,74 @@
+ <?php
+
+ namespace App\Http\Controllers;
+
+ use App\Models\User;
+ use App\Models\SurfSpot;
+ use App\Models\Comment;
+ use Illuminate\Http\Request;
+
+ class AdminController extends Controller
```

Delete and Edit Working – In this version, features are added so the admin user can edit and delete both comments and surf spots, in case of needing adjustments to follow the web page's guidelines.



MarkAsRead
OwainPeters48 • 2 days ago
Delete and Edit working
OwainPeters48 • 4 days ago
Merge branch 'master' of https://githu...
OwainPeters48 • 6 days ago
Notification
OwainPeters48 • 6 days ago
Admin/User specific Functions
OwainPeters48 • 7 days ago
Dashboard completely working

**Delete and Edit working**

The CRUD operations are functioning correctly on the dashboard page for users, surf spots, and comments

OwainPeters48 3da61f4 +372 -252

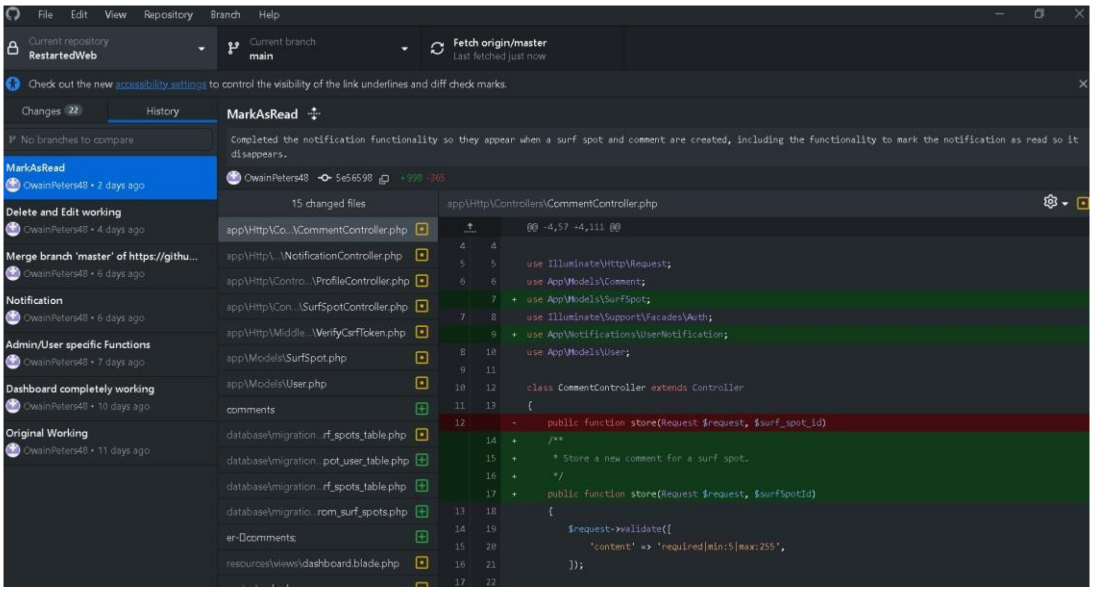
6 changed files

app\Http\Contr...\AdminController.php
app\Http\Co...\CommentController.php
app\Http\...\NotificationController.php
app\Http\Con...\SurfSpotController.php
resources\views\dashboard.blade.php
routes\web.php

app\Http\Controllers\AdminController.php

```
@@ -58,8 +58,11 @@ public function destroyUser($id)
         */
        public function destroySurfSpot($id)
        {
+           \Log::info("Destroy surf spot called with ID: $id");
            $surfSpot = SurfSpot::findOrFail($id);
            $surfSpot->delete();
+
+           \Log::info("Surf spot deleted successfully with ID: $id");
            return redirect()->route('dashboard')->with('success', 'Surf spot deleted successfully.');
        }
```

MarkAsRead – The mark as read feature is implemented in the notification section, where users can click the Mark As Read button and remove the notification from the dashboard, to free up the page in the case of there being too many notifications.



Check out the new accessibility settings to control the visibility of the link underlines and diff check marks.

MarkAsRead
OwainPeters48 • 2 days ago

Delete and Edit working
OwainPeters48 • 4 days ago

Merge branch 'master' of https://githu...
OwainPeters48 • 6 days ago

Notification
OwainPeters48 • 6 days ago

Admin/User specific Functions
OwainPeters48 • 7 days ago

Dashboard completely working
OwainPeters48 • 10 days ago

Original Working
OwainPeters48 • 11 days ago

**MarkAsRead**

```
Completed the notification functionality so they appear when a surf spot and comment are created, including the functionality to mark the notification as read so it disappears.
```

OwainPeters48 5e56598 +998 -365

15 changed files

- app\Http\Co...\CommentController.php
- app\Http\...\NotificationController.php
- app\Http\Contro...\ProfileController.php
- app\Http\Con...\SurfSpotController.php
- app\Http\Middle...\VerifyCsrfToken.php
- app\Models\SurfSpot.php
- app\Models\User.php
- comments
- database\migration...rf_spots_table.php
- database\migration...pot_user_table.php
- database\migration...rf_spots_table.php
- database\migratio...rom_surf_spots.php
- er-Dcomments;
- resources\views\dashboard.blade.php

app\Http\Controllers\CommentController.php

```
@@ -4,57 +4,111 @@

  use Illuminate\Http\Request;
  use App\Models\Comment;
+ use App\Models\SurfSpot;
  use Illuminate\Support\Facades\Auth;
+ use App\Notifications\UserNotification;
  use App\Models\User;

  class CommentController extends Controller
  {
-     public function store(Request $request, $surf_spot_id)
+     /**
+      * Store a new comment for a surf spot.
+      */
+     public function store(Request $request, $surfSpotId)
      {
          $request->validate([
              'content' => 'required|min:5|max:255',
          ]);

```

MVC Architecture

This project aligns with the Model-View-Controller (MVC) architectural pattern, which includes three separate components that provide the apps logic and are designed to organize the code.  These are Model, View, and Controller.

1. Model

   The Models are responsible for interacting with the database and showing the real-world components of the application.

- SurfSpot, Comment, and User Models:

  These models I created represent the database tables 'surf_spots' , 'comments', and 'users', and handle each of their relationships to one another.

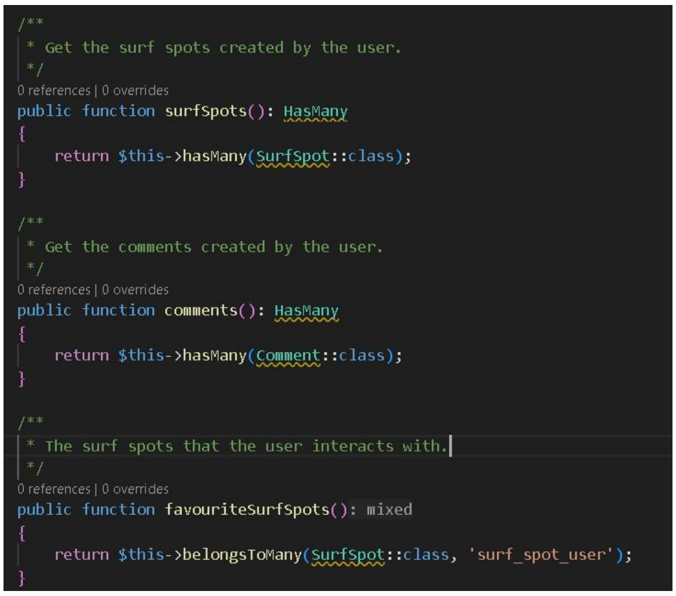
  Key Relationships:

  A SurfSpot has many Comments and belongs to many Users through the favorites relationship.

  A User has many comments, SurfSpots and can have multiple SurfSpots as favourites.

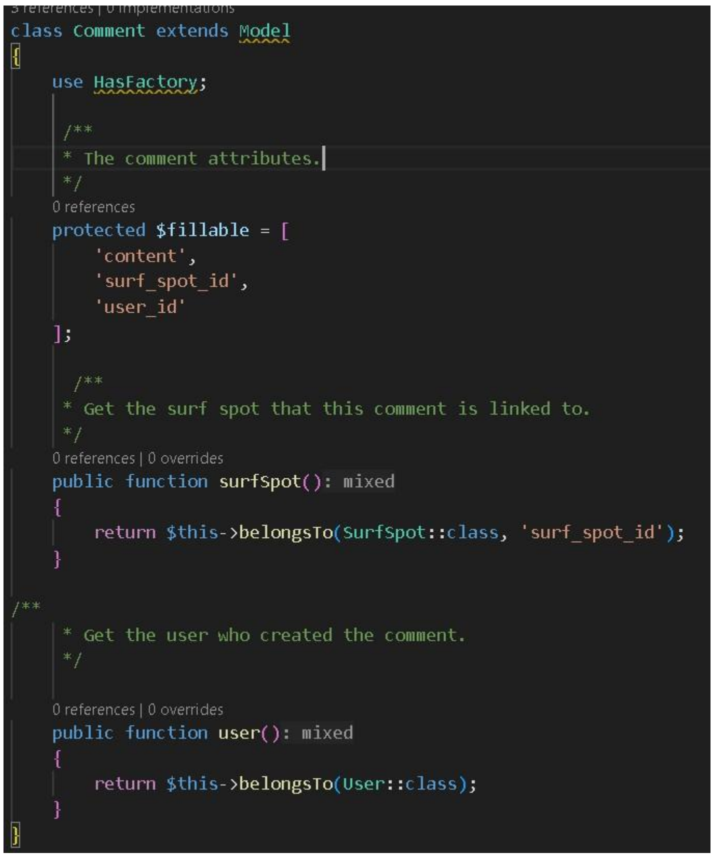
  A Comment belongs to both a User and a SurfSpot.

Example Code of the User Model:



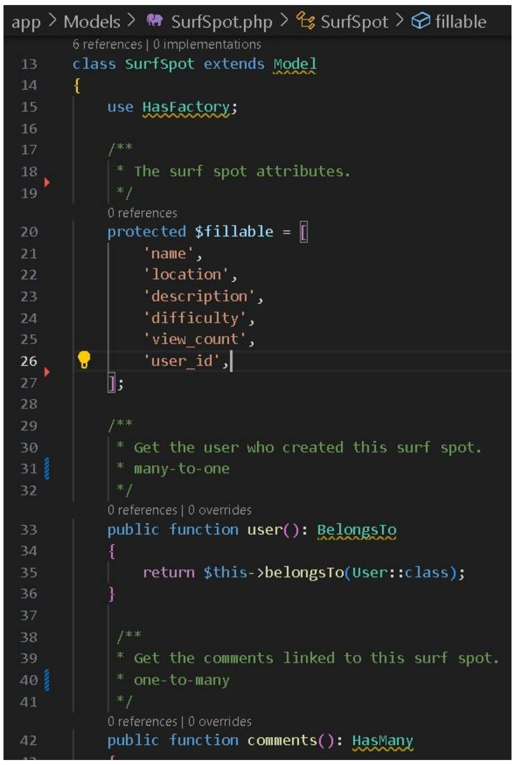
```php
/**
 * Get the surf spots created by the user.
 */
0 references | 0 overrides
public function surfSpots(): HasMany
{
    return $this->hasMany(SurfSpot::class);
}

/**
 * Get the comments created by the user.
 */
0 references | 0 overrides
public function comments(): HasMany
{
    return $this->hasMany(Comment::class);
}

/**
 * The surf spots that the user interacts with.
 */
0 references | 0 overrides
public function favouriteSurfSpots(): mixed
{
    return $this->belongsToMany(SurfSpot::class, 'surf_spot_user');
}
```

Example Code of Comment Model:



```php
3 references | 0 implementations
class Comment extends Model
{
    use HasFactory;

     /**
     * The comment attributes.
     */
    0 references
    protected $fillable = [
        'content',
        'surf_spot_id',
        'user_id'
    ];

      /**
     * Get the surf spot that this comment is linked to.
     */
    0 references | 0 overrides
    public function surfSpot(): mixed
    {
        return $this->belongsTo(SurfSpot::class, 'surf_spot_id');
    }

/**
     * Get the user who created the comment.
     */

    0 references | 0 overrides
    public function user(): mixed
    {
        return $this->belongsTo(User::class);
    }
}
```

Example Code of SurfSpot Model:



```php
6 references | 0 implementations
class SurfSpot extends Model
{
    use HasFactory;

    /**
     * The surf spot attributes.
     */
    0 references
    protected $fillable = [
        'name',
        'location',
        'description',
        'difficulty',
        'view_count',
        'user_id',
    ];

    /**
     * Get the user who created this surf spot.
     * many-to-one
     */
    0 references | 0 overrides
    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }

     /**
     * Get the comments linked to this surf spot.
     * one-to-many
     */
    0 references | 0 overrides
    public function comments(): HasMany
```

2. View

Blade Templates:

This Laravel project uses Blade templating to create web pages.

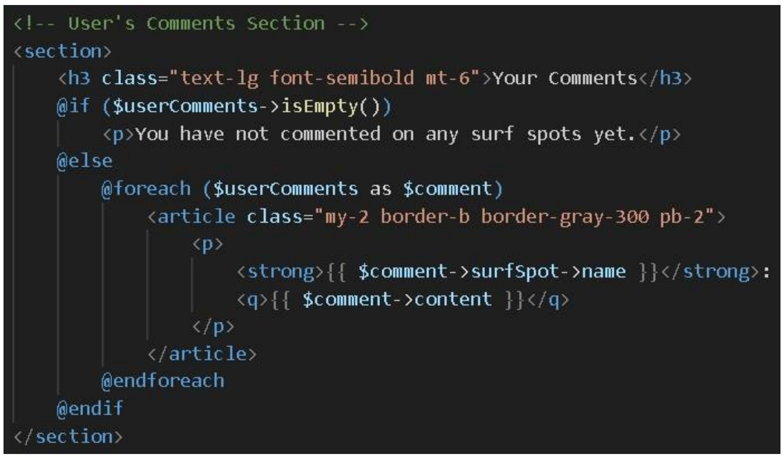
'Dashboard.blade':

This blade file displays the main page with every feature, including the surf spots, notifications, and user-specific functions such as creating or editing a surf spot.

'Login.blade':

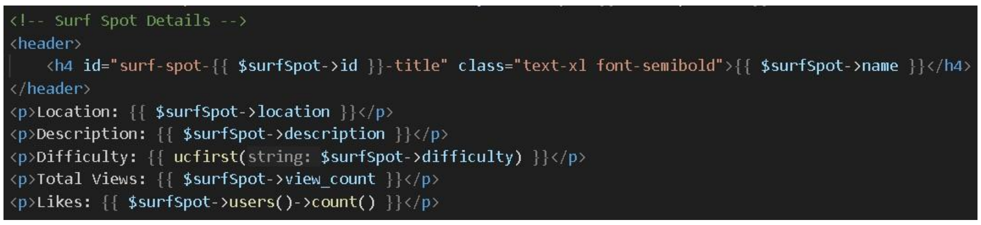
This is the other blade file used for the webpage, which is the home page for users initially entering the webpage, which asks them to login to the page to access the dashboard page.

This section of code below from the dashboard.blade shows all of the comments posted by the current user logged in:



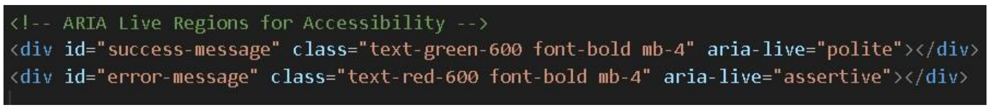
```
<!-- User's Comments Section -->
<section>
    <h3 class="text-lg font-semibold mt-6">Your Comments</h3>
    @if ($userComments->isEmpty())
        <p>You have not commented on any surf spots yet.</p>
    @else
        @foreach ($userComments as $comment)
            <article class="my-2 border-b border-gray-300 pb-2">
                <p>
                    <strong>{{ $comment->surfSpot->name }}</strong>:
                    <q>{{ $comment->content }}</q>
                </p>
            </article>
        @endforeach
    @endif
</section>
```

Surf Spot Details Code:



```
<!-- Surf Spot Details -->
<header>
    <h4 id="surf-spot-{{ $surfSpot->id }}-title" class="text-xl font-semibold">{{ $surfSpot->name }}</h4>
</header>
<p>Location: {{ $surfSpot->location }}</p>
<p>Description: {{ $surfSpot->description }}</p>
<p>Difficulty: {{ ucfirst(string: $surfSpot->difficulty) }}</p>
<p>Total Views: {{ $surfSpot->view_count }}</p>
<p>Likes: {{ $surfSpot->users()->count() }}</p>
```

The below section of code shows the ARIA roles being implemented which comply with the WCAG guidelines:



```
<!-- ARIA Live Regions for Accessibility -->
<div id="success-message" class="text-green-600 font-bold mb-4" aria-live="polite"></div>
<div id="error-message" class="text-red-600 font-bold mb-4" aria-live="assertive"></div>
```

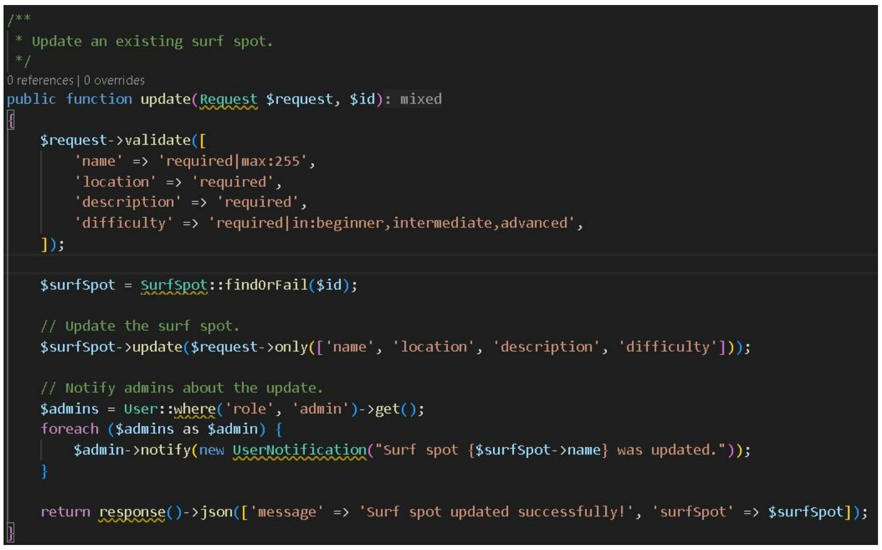
3. Controllers

The Controllers are responsible for connecting the View's to the Model by interacting with the model, while handling the user input.

SurfSpotController:

This handles the logic for displaying, creating, updating and deleting surf spots.

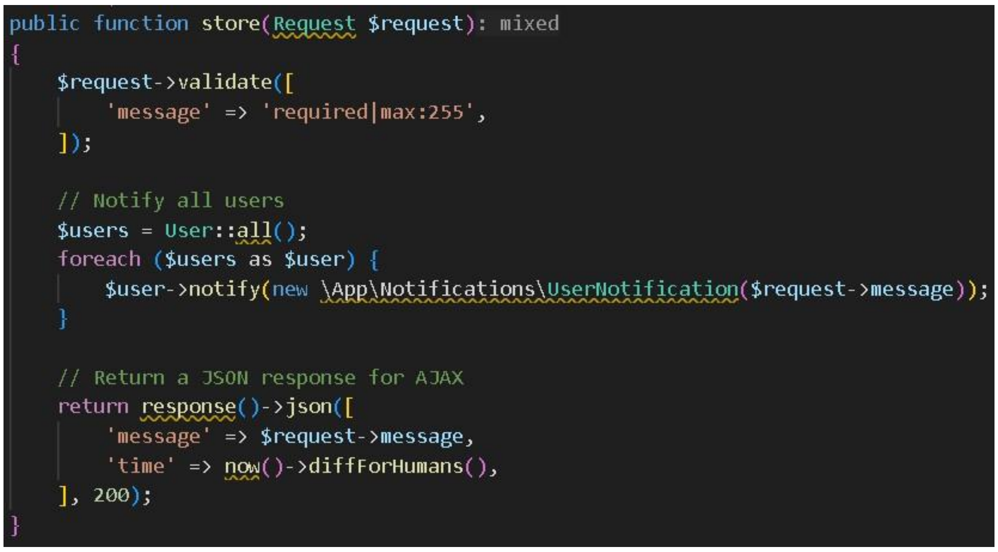
Update Function (Handles updating an existing surf spot):



```php
/**
 * Update an existing surf spot.
 */
0 references | 0 overrides
public function update(Request $request, $id): mixed
{
    $request->validate([
        'name' => 'required|max:255',
        'location' => 'required',
        'description' => 'required',
        'difficulty' => 'required|in:beginner,intermediate,advanced',
    ]);

    $surfSpot = SurfSpot::findOrFail($id);

    // Update the surf spot.
    $surfSpot->update($request->only(['name', 'location', 'description', 'difficulty']));

    // Notify admins about the update.
    $admins = User::where('role', 'admin')->get();
    foreach ($admins as $admin) {
        $admin->notify(new UserNotification("Surf spot {$surfSpot->name} was updated."));
    }

    return response()->json(['message' => 'Surf spot updated successfully!', 'surfSpot' => $surfSpot]);
}
```

NotificationController:

This handles storing notifications, displaying, creating and notifying about notifications, in addition to the marking a notification as read function. Store function (Handles saving the notification in the database, notifying users and displaying the notification using AJAX):

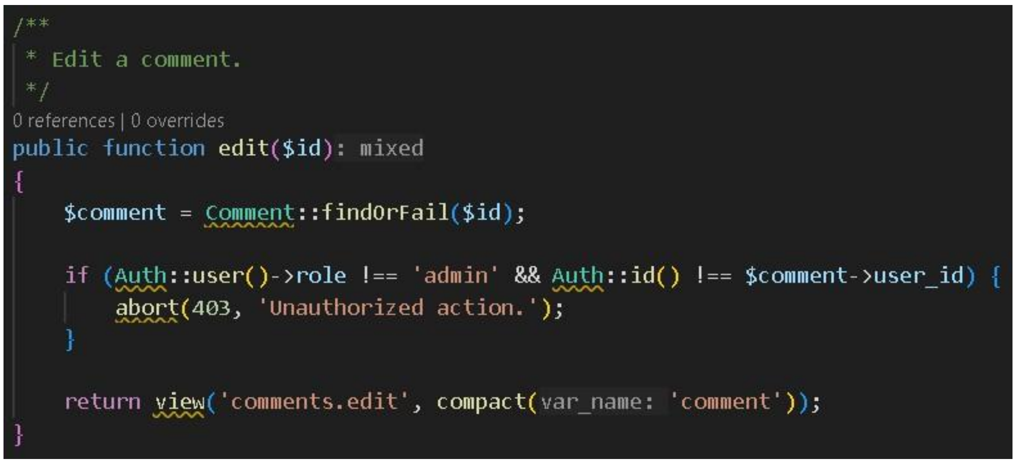


```php
public function store(Request $request): mixed
{
    $request->validate([
        'message' => 'required|max:255',
    ]);

    // Notify all users
    $users = User::all();
    foreach ($users as $user) {
        $user->notify(new \App\Notifications\UserNotification($request->message));
    }

    // Return a JSON response for AJAX
    return response()->json([
        'message' => $request->message,
        'time' => now()->diffForHumans(),
    ], 200);
}
```

CommentController:

This handles the comments and all the operations that can be done for a comment, such as posting, editing, updating and deleting.

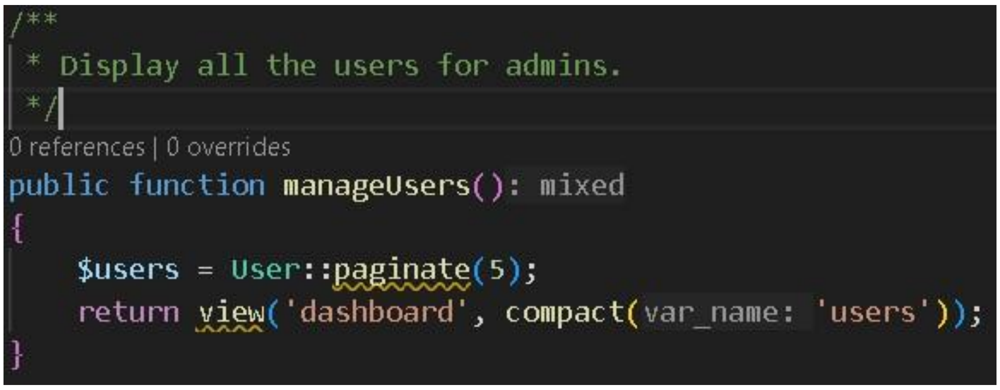
Function to edit a comment:



```
/**
 * Edit a comment.
 */
0 references | 0 overrides
public function edit($id): mixed
{
    $comment = Comment::findOrFail($id);

    if (Auth::user()->role !== 'admin' && Auth::id() !== $comment->user_id) {
        abort(403, 'Unauthorized action.');
    }

    return view('comments.edit', compact(var_name: 'comment'));
}
```

AdminController:

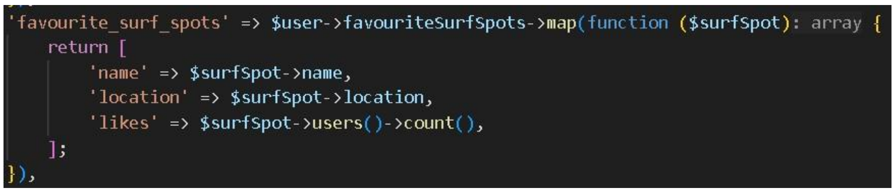
This is responsible for managing the admin specific functionalities within the application.  It uses functions such as updateRole and manageUsers to allow admins to delete models, change a user's roles and display all the users from the User model:



```
/**
 * Display all the users for admins.
 */
0 references | 0 overrides
public function manageUsers(): mixed
{
    $users = User::paginate(5);
    return view('dashboard', compact(var_name: 'users'));
}
```

ProfileController:

This handles the managing and displaying of user profiles, by the show method.  It handles AJAX requests to display the user's information, the comments that they've posted, surf spots they have created, and their favorited surf spots:

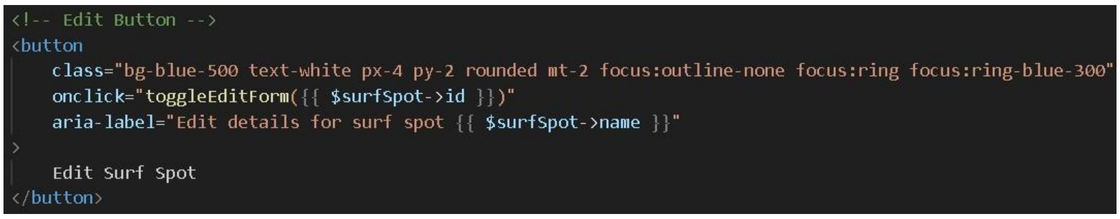


```
'favourite_surf_spots' => $user->favouriteSurfSpots->map(function ($surfSpot): array {
    return [
        'name' => $surfSpot->name,
        'location' => $surfSpot->location,
        'likes' => $surfSpot->users()->count(),
    ];
}),
```

Web Content Accessibility Guidelines (WCAG)

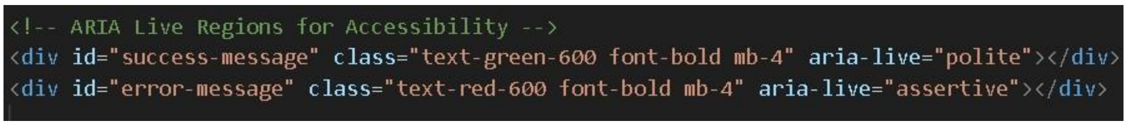
Accessibility has a crucial role in web development, therefore I made sure to implement many WCAG 2.0 principles within my application.

The application uses aria-label attributes to provide accessible descriptions for interactive parts of the page, as shown in the code snipped below:



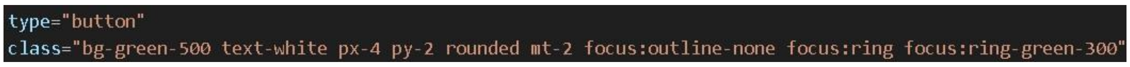
```
<!-- Edit Button -->
<button
    class="bg-blue-500 text-white px-4 py-2 rounded mt-2 focus:outline-none focus:ring focus:ring-blue-300"
    onclick="toggleEditForm({{ $surfSpot->id }})"
    aria-label="Edit details for surf spot {{ $surfSpot->name }}"
>
    Edit Surf Spot
</button>
```

ARIA live regions are used for success and error messages, making sure that the updates are given to all users:



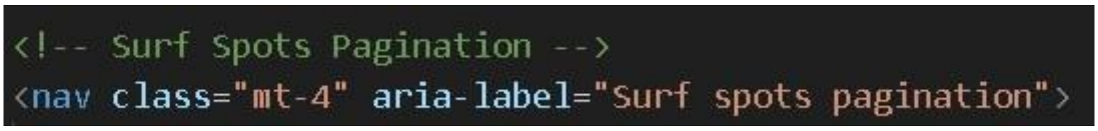
```
<!-- ARIA Live Regions for Accessibility -->
<div id="success-message" class="text-green-600 font-bold mb-4" aria-live="polite"></div>
<div id="error-message" class="text-red-600 font-bold mb-4" aria-live="assertive"></div>
```

I've made sure to include keyboard accessibility, using features such as 'focus:outline-none' and 'focus:ring', which are used to ensure users with keyboards can easily and seamlessly navigate and identify their position on the screen:



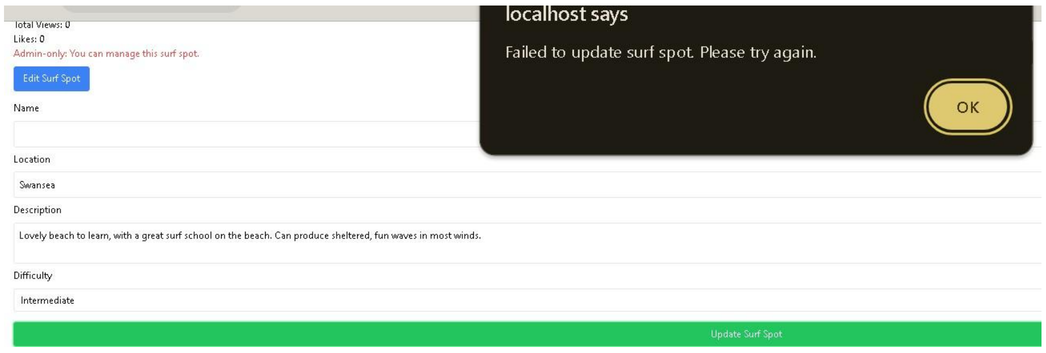
```
type="button"
class="bg-green-500 text-white px-4 py-2 rounded mt-2 focus:outline-none focus:ring focus:ring-green-300"
```

The application additionally uses navigation controls for pagination using the 'aria-label' attribute, which allows for easy and understandable access to each page:



```
<!-- Surf Spots Pagination -->
<nav class="mt-4" aria-label="Surf spots pagination">
```

I have implemented good validation for each user input to allow immediate feedback if a field is entered incorrectly through missing data or invalid data:


Total Views: 0
Likes: 0
Admin-only: You can manage this surf spot.

Name

Location
Swansea

Description
Lovely beach to learn, with a great surf school on the beach. Can produce sheltered, fun waves in most winds.

Difficulty
Intermediate

localhost says

Failed to update surf spot. Please try again.

## Conclusion

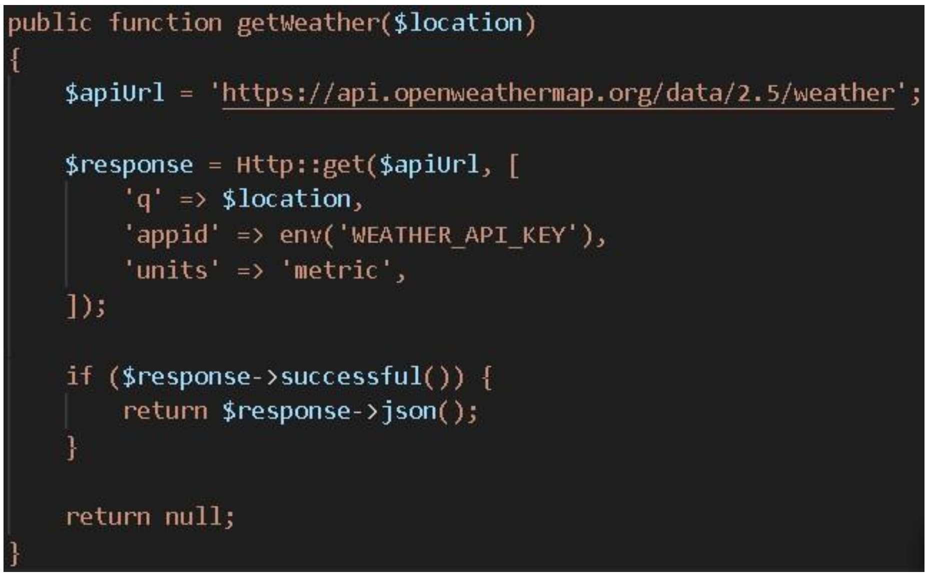
Throughout my project, I have consistently included the use of WCAG 2.0 requirements, such as ARIA attributes, accessible forms and validation to ensure accessibility for all users. However, to improve I could use alt attributes for the use of images if I managed to implement the images feature, which could enhance further the web pages accessibility.

## Service Container and API Communication

Integrating external APIs is a big part of web applications and can provide a good feature for an application.  I have not currently included a working API due to issues arising when attempting to implement a weather API for the surf spot conditions, however I have research sufficiently on how a service container may be used to implement API communication efficiently.

There are many benefits to using a Service Container, such as the ease of testing it, in addition to the scalability by allowing the apps to handle high demand from users, allowing the web app to scale by giving some functionalities to the third-party APIs.

The API could have been implemented using a Weather Service file, and use an API key within the .env file, while using the getWeather function within the WeatherService file to get the weather:



```php
public function getWeather($location)
{
    $apiUrl = 'https://api.openweathermap.org/data/2.5/weather';

    $response = Http::get($apiUrl, [
        'q' => $location,
        'appid' => env('WEATHER_API_KEY'),
        'units' => 'metric',
    ]);

    if ($response->successful()) {
        return $response->json();
    }

    return null;
}
```

I could have used the SurfSpotController to put the weather into each surf spot when it is displayed on the dashboard page, using the location of the surf spot to find the weather on the API.