Submitted by
Elaine Mae A.Bertiz
BSIT 3-C

November 28 ,2024

IT ELEC 1

WEB DEVELOPMENT

Submitted to
Prof.John Paul Azore

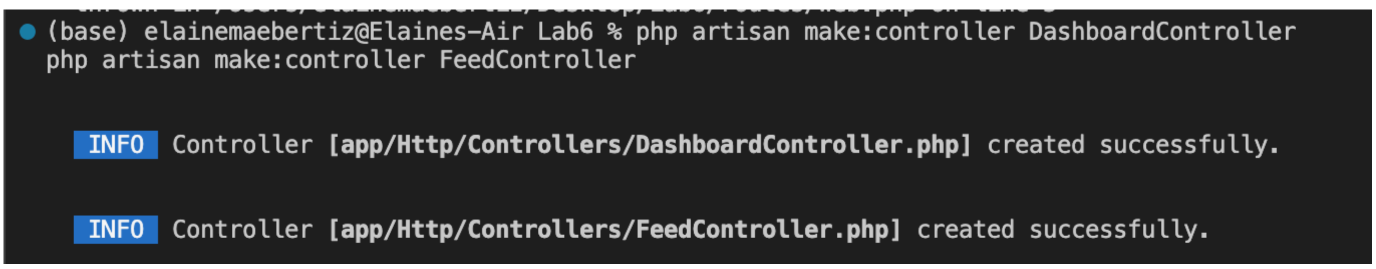
- Part 1: Create Views using Blades templates
- Create controllers that load the dashboard, feed, or equivalent pages.
- Register controllers in routes to link methods to URLs.
- You may create and register more controllers if needed.

- Part 2: Create controllers that load the content for each page (Simulate access to a DB).
- Load information such as posts, ratings, comments, pictures, etc.
- The information must be in the form of an array.

- Part 3: Documentation (Individual)
- Take screenshots of rendered pages and their accompanying code.
- Write brief explanations of controller logic, parameter handling, and route assignments.
- Cite other things you learned while making this.

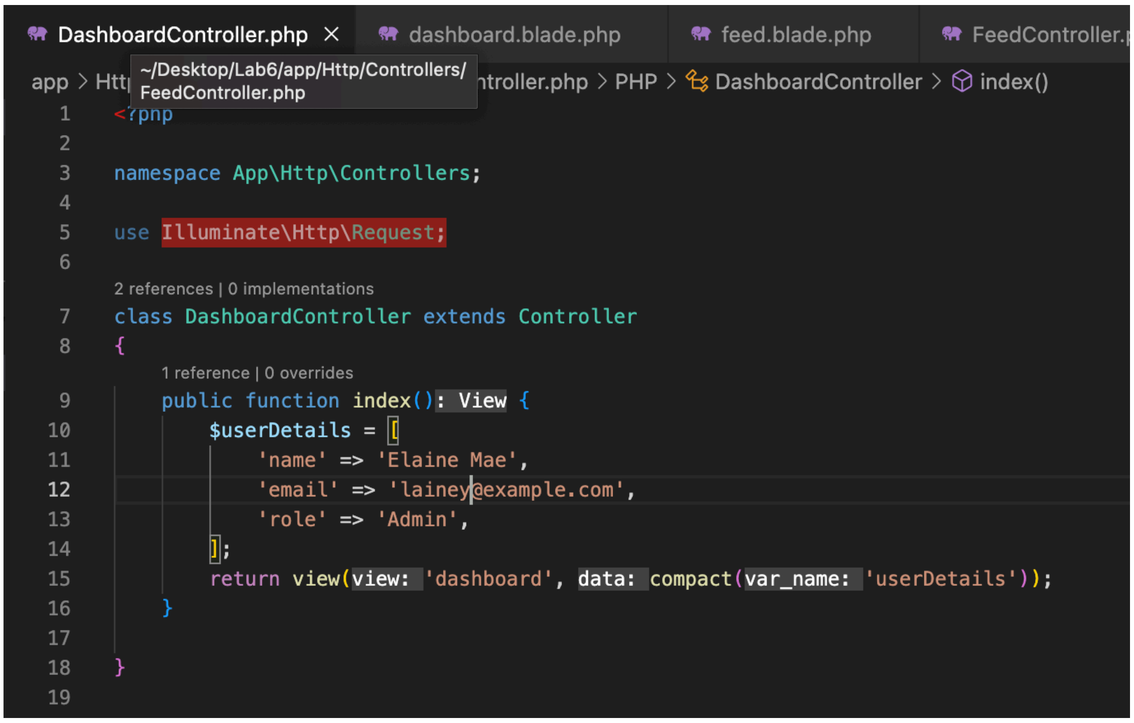
DOCUMENTATION

I have created new middleware named DashboardController & FeedController



```
● (base) elainemaebertiz@Elaines-Air Lab6 % php artisan make:controller DashboardController
  php artisan make:controller FeedController

     INFO  Controller [app/Http/Controllers/DashboardController.php] created successfully.

     INFO  Controller [app/Http/Controllers/FeedController.php] created successfully.
```

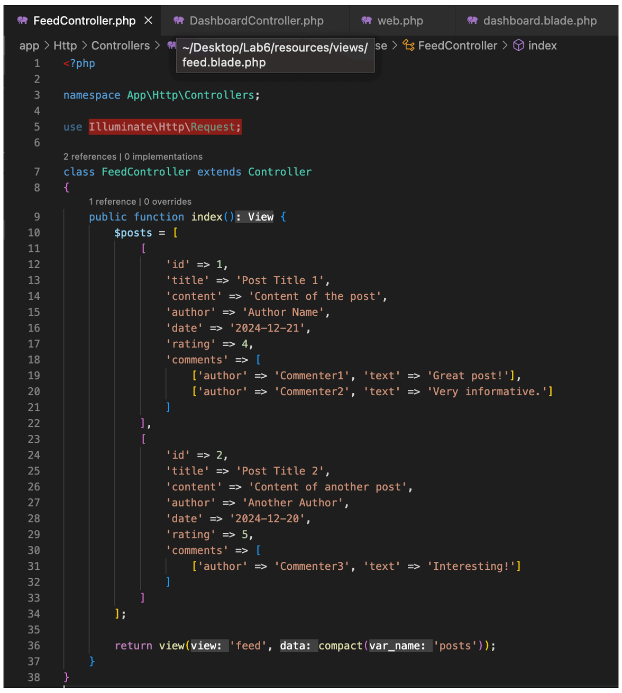
app/Http/Controllers/DashboardController



DashboardController.php
dashboard.blade.php
feed.blade.php
FeedController.p
app > Htt
ntroller.php > PHP > DashboardController > index()
~/Desktop/Lab6/app/Http/Controllers/FeedController.php
2 references | 0 implementations
1 reference | 0 overrides

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class DashboardController extends Controller
{
    public function index(): View {
        $userDetails = [
            'name' => 'Elaine Mae',
            'email' => 'lainey@example.com',
            'role' => 'Admin',
        ];
        return view(view: 'dashboard', data: compact(var_name: 'userDetails'));
    }

}
```

The `DashboardController` fetches user details (name, email, and role) and sends this data to the `dashboard` view. The `index()` method uses the `compact()` function to pass the user details as an array to the view for display.

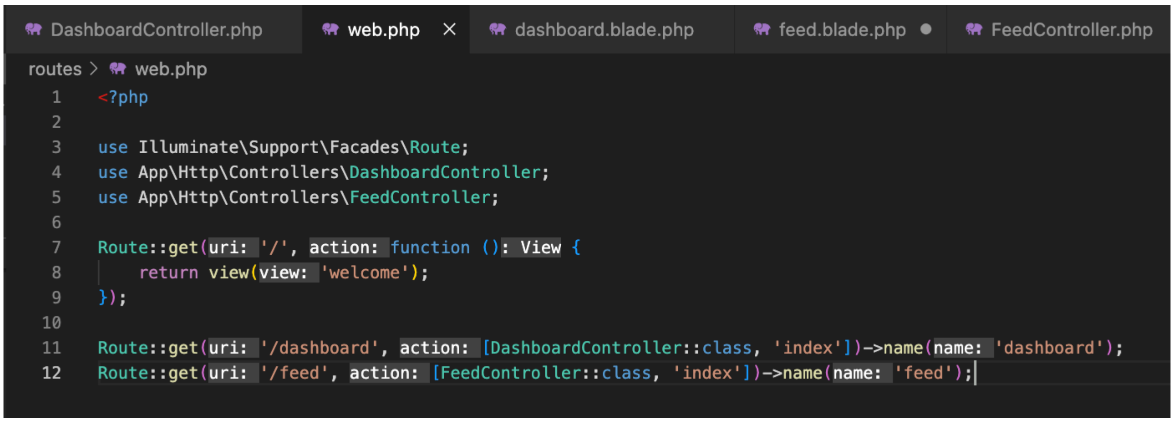

FeedController.php
DashboardController.php
web.php
dashboard.blade.php
app > Http > Controllers > ~/Desktop/Lab6/resources/views/feed.blade.php se > FeedController > index

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

2 references | 0 implementations
class FeedController extends Controller
{
    1 reference | 0 overrides
    public function index(): View {
        $posts = [
            [
                'id' => 1,
                'title' => 'Post Title 1',
                'content' => 'Content of the post',
                'author' => 'Author Name',
                'date' => '2024-12-21',
                'rating' => 4,
                'comments' => [
                    ['author' => 'Commenter1', 'text' => 'Great post!'],
                    ['author' => 'Commenter2', 'text' => 'Very informative.']
                ]
            ],
            [
                'id' => 2,
                'title' => 'Post Title 2',
                'content' => 'Content of another post',
                'author' => 'Another Author',
                'date' => '2024-12-20',
                'rating' => 5,
                'comments' => [
                    ['author' => 'Commenter3', 'text' => 'Interesting!']
                ]
            ]
        ];

        return view(view: 'feed', data: compact(var_name: 'posts'));
    }
}
```

app/Http/Controllers/DashboardController

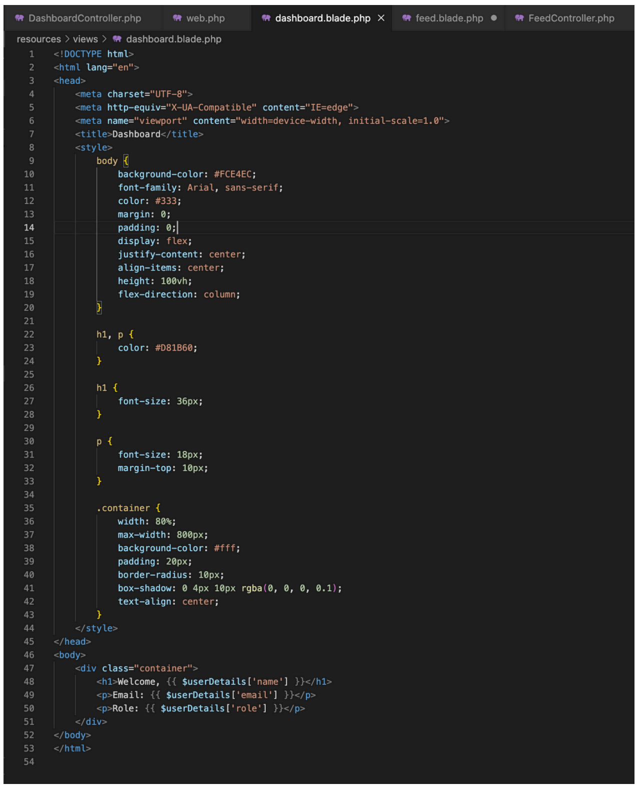
It defines an `index()` method that creates an array of posts, each containing details such as the post's ID, title, content, author, date, rating, and associated comments. The method then passes this array to the `feed` view using the `compact()` function, making the `$posts` data available for display in the view.

Router File



```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\DashboardController;
use App\Http\Controllers\FeedController;

Route::get(uri: '/', action: function (): View {
    return view(view: 'welcome');
});

Route::get(uri: '/dashboard', action: [DashboardController::class, 'index'])->name(name: 'dashboard');
Route::get(uri: '/feed', action: [FeedController::class, 'index'])->name(name: 'feed');
```

The root URL (/) displays the welcome view, /dashboard calls the index method of the DashboardController to display the dashboard, and /feed calls the index method of the FeedController to display the feed.

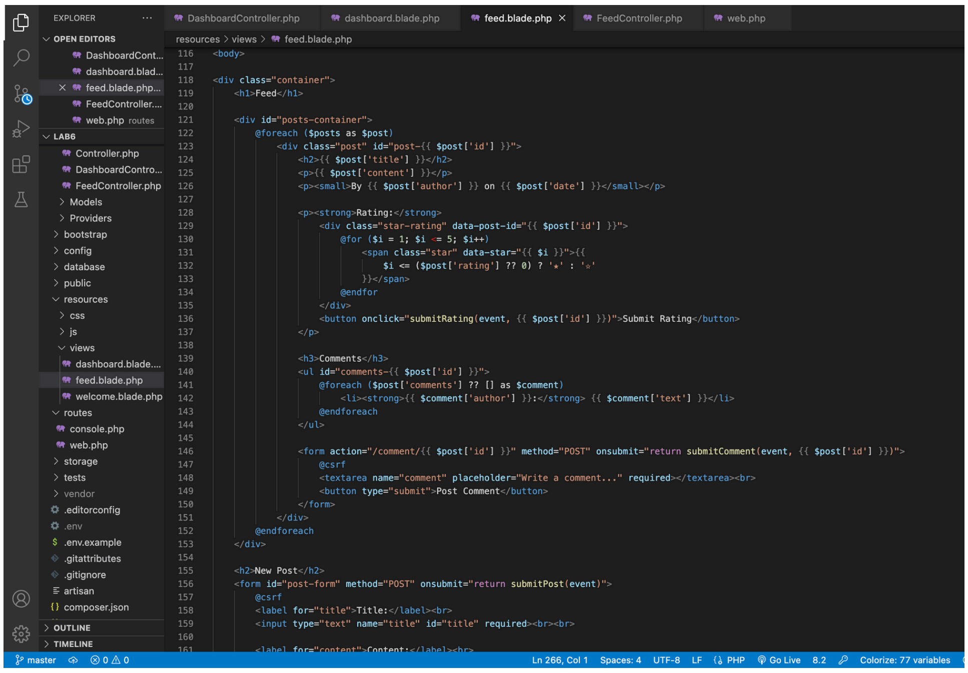


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
    <style>
        body {
            background-color: #FCE4EC;
            font-family: Arial, sans-serif;
            color: #333;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            flex-direction: column;
        }

        h1, p {
            color: #D81B60;
        }

        h1 {
            font-size: 36px;
        }

        p {
            font-size: 18px;
            margin-top: 10px;
        }

        .container {
            width: 80%;
            max-width: 800px;
            background-color: #fff;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome, {{ $userDetails['name'] }}</h1>
        <p>Email: {{ $userDetails['email'] }}</p>
        <p>Role: {{ $userDetails['role'] }}</p>
    </div>
</body>
</html>
```

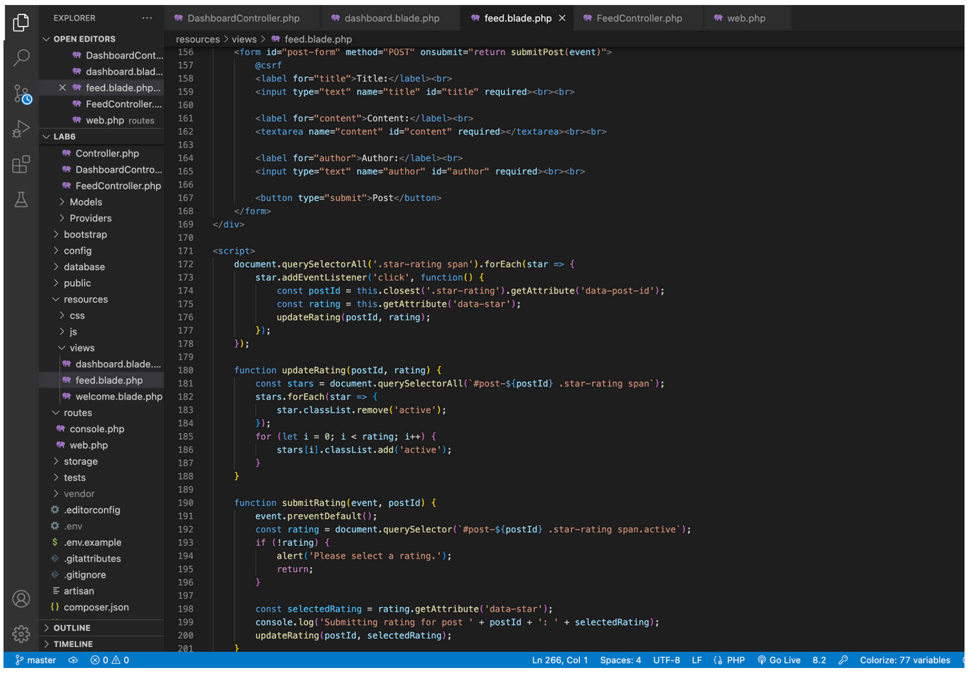
Dashboard:

displays a dashboard page with a welcome message for the user. It shows the user's name, email, and role by dynamically inserting values from the $userDetails array. The layout uses flexbox for centering content and a card-like container with a shadow for styling.

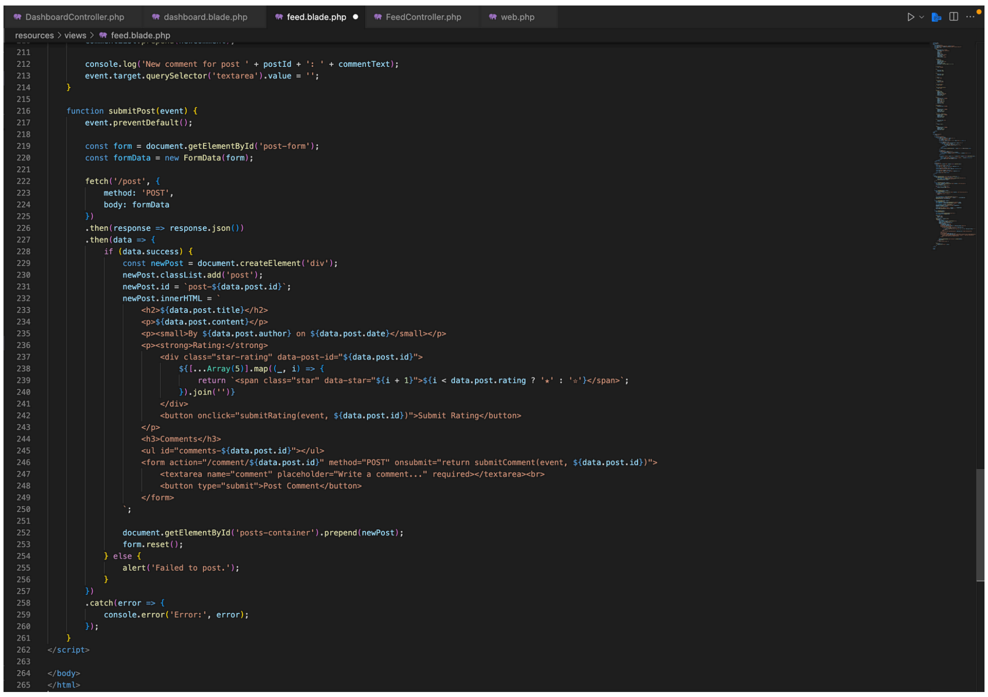
feed.blade.php



```
<body>

<div class="container">
    <h1>Feed</h1>

    <div id="posts-container">
        @foreach ($posts as $post)
            <div class="post" id="post-{{ $post['id'] }}">
                <h2>{{ $post['title'] }}</h2>
                <p>{{ $post['content'] }}</p>
                <p><small>By {{ $post['author'] }} on {{ $post['date'] }}</small></p>

                <p><strong>Rating:</strong>
                    <div class="star-rating" data-post-id="{{ $post['id'] }}">
                        @for ($i = 1; $i <= 5; $i++)
                            <span class="star" data-star="{{ $i }}">{{
                                $i <= ($post['rating'] ?? 0) ? '★' : '☆'
                            }}</span>
                        @endfor
                    </div>
                    <button onclick="submitRating(event, {{ $post['id'] }})">Submit Rating</button>
                </p>

                <h3>Comments</h3>
                <ul id="comments-{{ $post['id'] }}">
                    @foreach ($post['comments'] ?? [] as $comment)
                        <li><strong>{{ $comment['author'] }}:</strong> {{ $comment['text'] }}</li>
                    @endforeach
                </ul>

                <form action="/comment/{{ $post['id'] }}" method="POST" onsubmit="return submitComment(event, {{ $post['id'] }})">
                    @csrf
                    <textarea name="comment" placeholder="Write a comment..." required></textarea><br>
                    <button type="submit">Post Comment</button>
                </form>
            </div>
        @endforeach
    </div>

    <h2>New Post</h2>
    <form id="post-form" method="POST" onsubmit="return submitPost(event)">
        @csrf
        <label for="title">Title:</label><br>
        <input type="text" name="title" id="title" required><br><br>

        <label for="content">Content:</label><br>
```



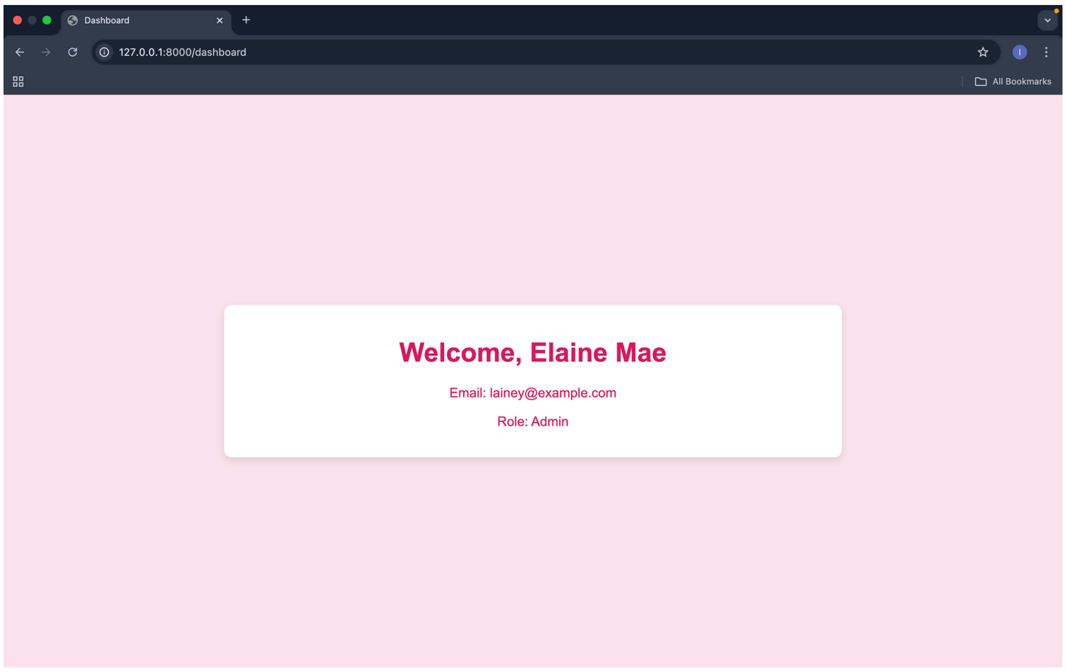
```
    <form id="post-form" method="POST" onsubmit="return submitPost(event)">
        @csrf
        <label for="title">Title:</label><br>
        <input type="text" name="title" id="title" required><br><br>

        <label for="content">Content:</label><br>
        <textarea name="content" id="content" required></textarea><br><br>

        <label for="author">Author:</label><br>
        <input type="text" name="author" id="author" required><br><br>

        <button type="submit">Post</button>
    </form>
</div>

<script>
    document.querySelectorAll('.star-rating span').forEach(star => {
        star.addEventListener('click', function() {
            const postId = this.closest('.star-rating').getAttribute('data-post-id');
            const rating = this.getAttribute('data-star');
            updateRating(postId, rating);
        });
    });

    function updateRating(postId, rating) {
        const stars = document.querySelectorAll(`#post-${postId} .star-rating span`);
        stars.forEach(star => {
            star.classList.remove('active');
        });
        for (let i = 0; i < rating; i++) {
            stars[i].classList.add('active');
        }
    }

    function submitRating(event, postId) {
        event.preventDefault();
        const rating = document.querySelector(`#post-${postId} .star-rating span.active`);
        if (!rating) {
            alert('Please select a rating.');
            return;
        }

        const selectedRating = rating.getAttribute('data-star');
        console.log('Submitting rating for post ' + postId + ': ' + selectedRating);
        updateRating(postId, selectedRating);
    }
```



```

            console.log('New comment for post ' + postId + ': ' + commentText);
            event.target.querySelector('textarea').value = '';
        }

        function submitPost(event) {
            event.preventDefault();

            const form = document.getElementById('post-form');
            const formData = new FormData(form);

            fetch('/post', {
                method: 'POST',
                body: formData
            })
            .then(response => response.json())
            .then(data => {
                if (data.success) {
                    const newPost = document.createElement('div');
                    newPost.classList.add('post');
                    newPost.id = `post-${data.post.id}`;
                    newPost.innerHTML = `
                        <h2>${data.post.title}</h2>
                        <p>${data.post.content}</p>
                        <p><small>By ${data.post.author} on ${data.post.date}</small></p>
                        <p><strong>Rating:</strong>
                            <div class="star-rating" data-post-id="${data.post.id}">
                                ${[...Array(5)].map((_, i) => {
                                    return `<span class="star" data-star="${i + 1}">${i < data.post.rating ? '★' : '☆'}</span>`;
                                }).join('')}
                            </div>
                            <button onclick="submitRating(event, ${data.post.id})">Submit Rating</button>
                        </p>
                        <h3>Comments</h3>
                        <ul id="comments-${data.post.id}"></ul>
                        <form action="/comment/${data.post.id}" method="POST" onsubmit="return submitComment(event, ${data.post.id})">
                            <textarea name="comment" placeholder="Write a comment..." required></textarea><br>
                            <button type="submit">Post Comment</button>
                        </form>
                    `;

                    document.getElementById('posts-container').prepend(newPost);
                    form.reset();
                } else {
                    alert('Failed to post.');
                }
            })
            .catch(error => {
                console.error('Error:', error);
            });
        }
    </script>

</body>
</html>
```

creates a page displaying posts with titles, content, author information, and ratings. Each post has a star rating system where users can click on stars to rate the post, and a comment section where users can write and submit comments. Users can also create new posts by filling out a form with a title, content, and author information. The page dynamically updates the post content, ratings, and comments using JavaScript, which handles the interactions and submissions without reloading the page.

DOCUMENTATION


Dashboard

127.0.0.1:8000/dashboard

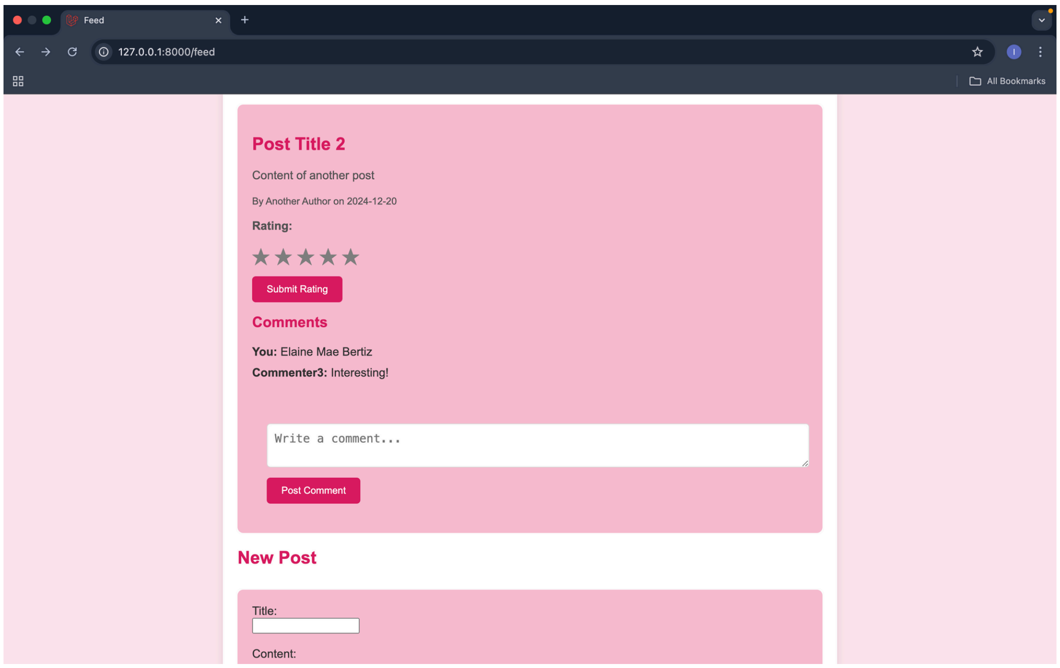
All Bookmarks

Welcome, Elaine Mae

Email: lainey@example.com

Role: Admin

In the dashboard, a welcome message is displayed to the admin, showing their name, email, and role. The data is dynamically passed from the controller to the view for personalized display.


Feed

127.0.0.1:8000/feed

All Bookmarks

Post Title 2

Content of another post

By Another Author on 2024-12-20

Rating:

★★★★★

Submit Rating

Comments

You: Elaine Mae Bertiz

Commenter3: Interesting!

Write a comment...

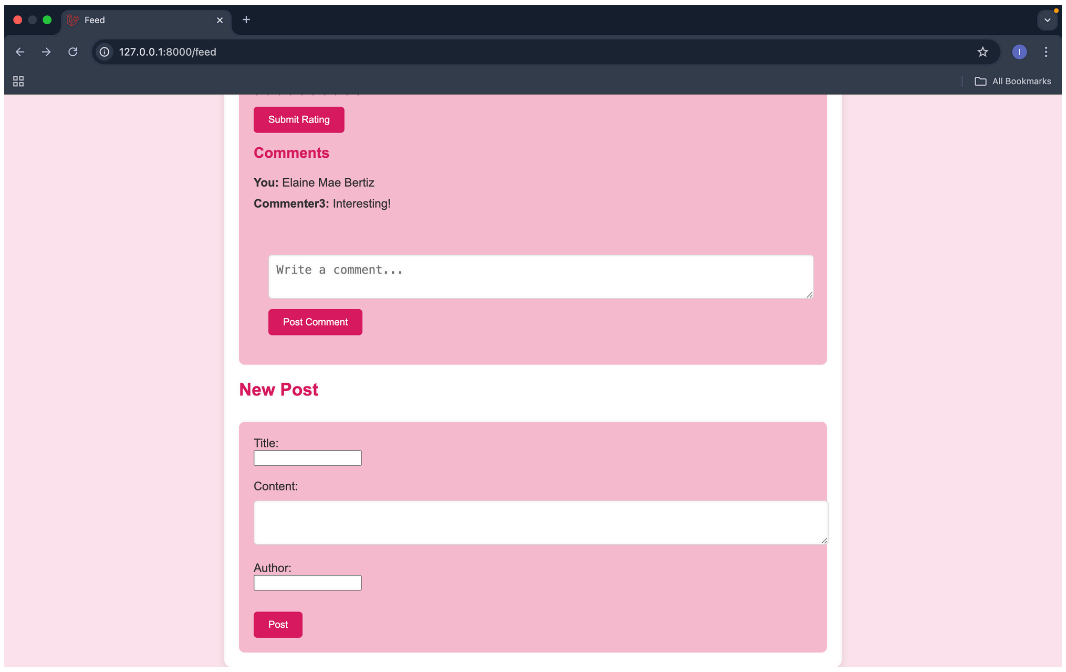
Post Comment

New Post

Title:

Content:

In the feed, users can rate posts using a star rating system and leave comments on each post. The `FeedController` passes post data, including comments and ratings, to the view for display and interaction.


Feed

127.0.0.1:8000/feed

All Bookmarks

Submit Rating

Comments

You: Elaine Mae Bertiz

Commenter3: Interesting!

Write a comment...

Post Comment

New Post

Title:

Content:

Author:

Post

You can also create a new post