# **Questions:**

1. **Create a user table that has name, email, age, location and password. Now, Display all the data in table with all the filters applied.**
2. **Create a user table that has name, email, age, location and password. Users can sign multiple song. so, create another table for song that has user\_id and the display user table with all the song that he sang with all the filters applied.**
3. **Create a users table, where each user can create multiple customers. Therefore, the customers table should include a user\_id field. Now, perform a reverse relationship query.**
4. **Create a contact us form and send email to admin.**
5. **Create a real time bell icon notification using laravel+livewire and also laravel+ajax.**
6. **List all the users with his status i.e. online or offline.**
7. **Create a product table and apply all the possible filters from frontend for user using AJAX.**
8. **Authentication using laravel ui package and spatie for roles and permission.**
9. **Login with laravel Socialite.**
10. **Write a logic to export database data to pdf,excel and csv file. Also write logic to import excel file to insert into database table.**
11. **laravel all eloquent orm model concept.**
12. **How to host laravel project in c panel.**
13. **Make Laravel website PWA.**
14. **Create a user table that has name, email, age, location and password. Now, Display all the data in table with all the filters applied.**

 public function all($keyword = null, $entries = 10, $minDate = null, $maxDate = null, $sortField = 'created\_at', $sortDirection = 'asc')

    {

        $query = User::query();

        // Apply keyword filter

        if ($keyword) {

            $query->where(function ($q) use ($keyword) {

                $q->where('name', 'like', '%' . $keyword . '%')

                  ->orWhere('email', 'like', '%' . $keyword . '%')

                  ->orWhere('location', 'like', '%' . $keyword . '%');

            });

        }

        // Apply date filters

        if ($minDate) {

            $query->whereDate('created\_at', '>=', $minDate);

        }

        if ($maxDate) {

            $query->whereDate('created\_at', '<=', $maxDate);

        }

        // Apply sorting

        $query->orderBy($sortField, $sortDirection);

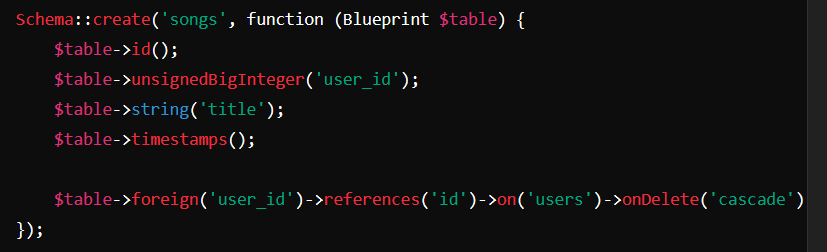
        // Limit results per page

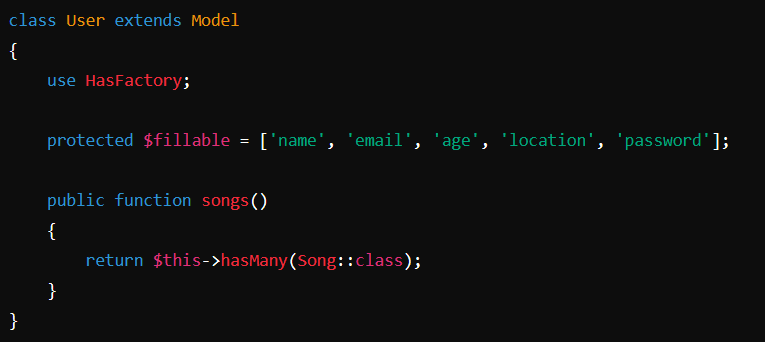
        $users = $query->paginate($entries);

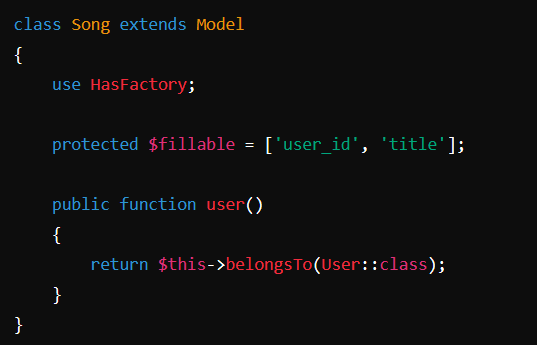
        return view('users.index', compact('users'));

    }

1. **Create a user table that has name, email, age, location and password. Users can sign multiple song. so, create another table for song that has user\_id and the display user table with all the song that he sang with all the filters applied.**







 public function all($keyword = null, $entries = 10, $minDate = null, $maxDate = null, $sortField = 'created\_at', $sortDirection = 'asc')

    {

        $query = User::with('songs');

        // Apply keyword filter for user attributes (name, email, location)

        if ($keyword) {

            $query->where(function ($q) use ($keyword) {

                $q->where('name', 'like', '%' . $keyword . '%')

                  ->orWhere('email', 'like', '%' . $keyword . '%')

                  ->orWhere('location', 'like', '%' . $keyword . '%')

                  ->orWhereHas('songs', function ($q) use ($keyword) {

                      $q->where('title', 'like', '%' . $keyword . '%');

                  });

            });

        }

        // Apply date filters

        if ($minDate) {

            $query->whereDate('created\_at', '>=', $minDate);

        }

        if ($maxDate) {

            $query->whereDate('created\_at', '<=', $maxDate);

        }

        // Apply sorting

        $query->orderBy($sortField, $sortDirection);

        // Limit results per page

        $users = $query->paginate($entries);

        return view('users.index', compact('users'));

    }

1. **Create a users table, where each user can create multiple customers. Therefore, the customers table should include a user\_id field. Now, perform a reverse relationship query.**

 public function all($keyword = null, $entries = 10, $minDate = null, $maxDate = null, $sortField = 'created\_at', $sortDirection = 'asc')

    {

        $query = Customer::with('user'); // Eager load User data with Customers

        // Apply keyword filter to customer and user fields

        if ($keyword) {

            $query->where(function ($q) use ($keyword) {

                $q->where('customer\_name', 'like', '%' . $keyword . '%')

                  ->orWhere('location', 'like', '%' . $keyword . '%')

                  ->orWhereHas('user', function ($q) use ($keyword) {

                      $q->where('name', 'like', '%' . $keyword . '%')

                        ->orWhere('email', 'like', '%' . $keyword . '%');

                  });

            });

        }

        // Apply date filters based on customer created\_at field

        if ($minDate) {

            $query->whereDate('created\_at', '>=', $minDate);

        }

        if ($maxDate) {

            $query->whereDate('created\_at', '<=', $maxDate);

        }

        // Apply sorting, including handling for sorting by user name

        if ($sortField === 'user\_name') {

            // Sorting by related `user.name`

            $query->leftJoin('users', 'customers.user\_id', '=', 'users.id')

                  ->orderBy('users.name', $sortDirection)

                  ->select('customers.\*');

        } else {

            // Default sorting by customer attributes

            $query->orderBy($sortField, $sortDirection);

        }

        // Get paginated results

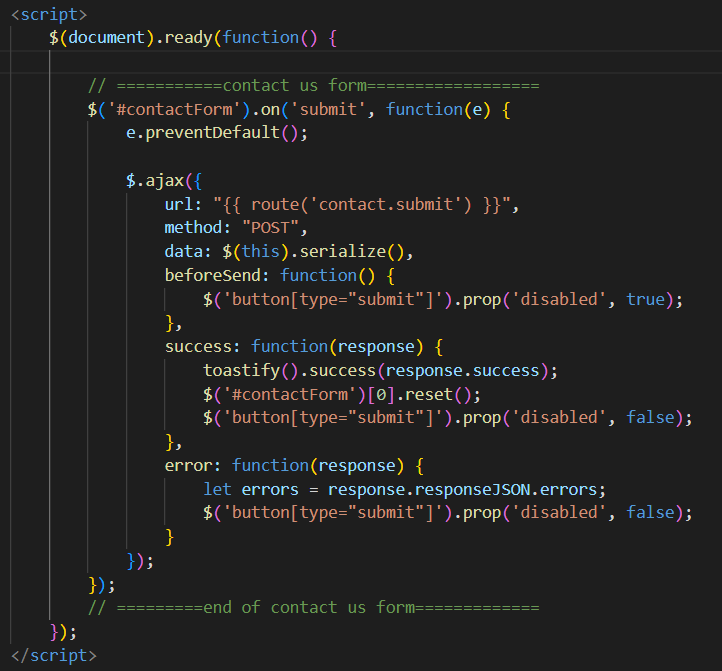
        $customers = $query->paginate($entries);

        return view('customers.index', compact('customers'));

    }

1. **Create a contact us form and send email to admin.**
2. index.blade.php

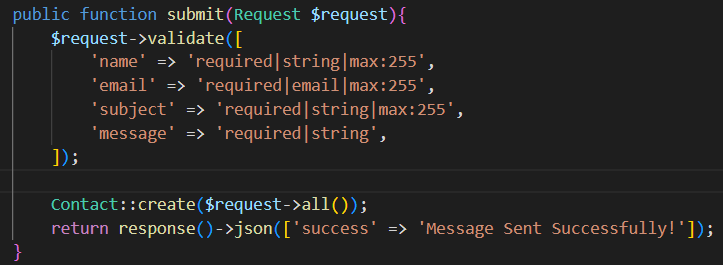




1. In web.php



1. In HomeController.php



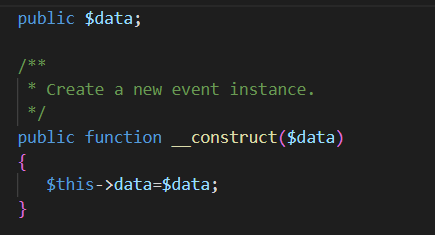
1. This will only save user contact information into db..so now we will create an event & Listener to send email that notify all the admin’s that someone want to contact with us.
2. Make Event:



1. Inside your controller:

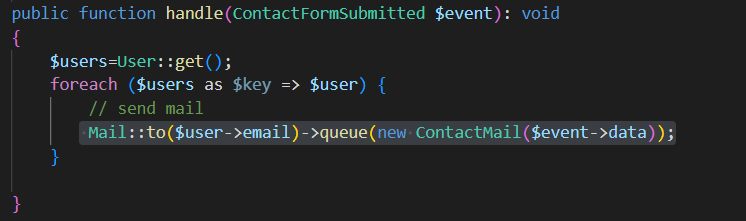


1. Inside your event ContactFormSubmitted.php

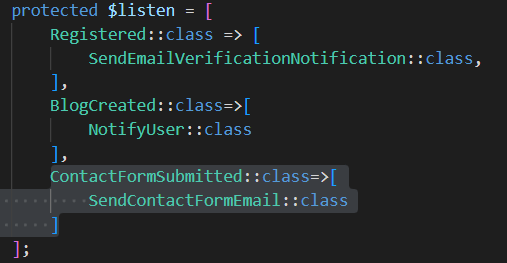


1. Make listener to listen your event(ContactFormSubmitted.php)



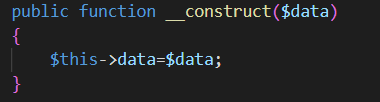


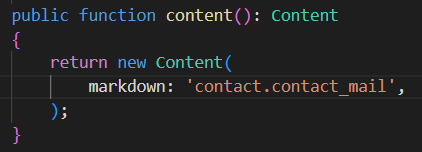
1. Inside EventServiceProvider.php



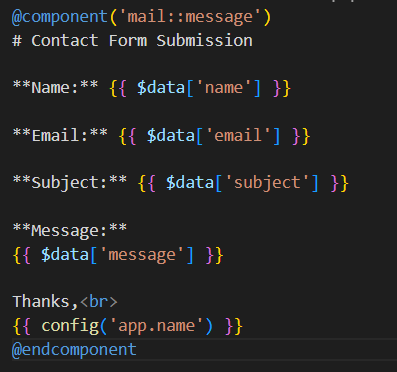
1. Make mail to send mail:



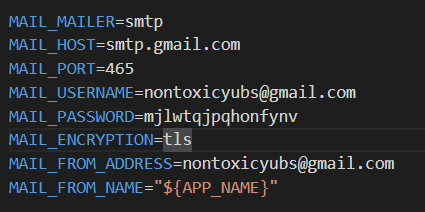




1. Inside views/contact/contact\_mail.blade.php



1. Inside .env



**Get password from security->2-step verification->Next->App Passwords**

1. Queues allow you to defer time-consuming tasks, such as sending emails or

generating reports, to be processed in the background, freeing up your

application to handle other requests.

1. **php artisan queue:table**
2. **in .env**

****

1. **Inside ContactMail.php**

****

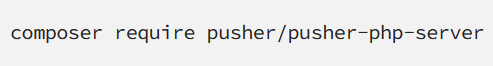
1. **php artisan migrate**
2. **php artisan queue:listen**
3. **Create a real time bell icon notification using laravel+livewire and also laravel+ajax.**

**Pusher For Real Time:**

Official docs:

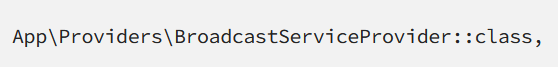
<https://medium.com/@parthpatel0516/laravel-real-time-notifications-with-pusher-3ac72531e15>

1. Install Pusher

****

****

1. Uncomment below code from config/app.php

****

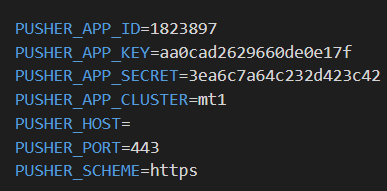
1. In .env

****

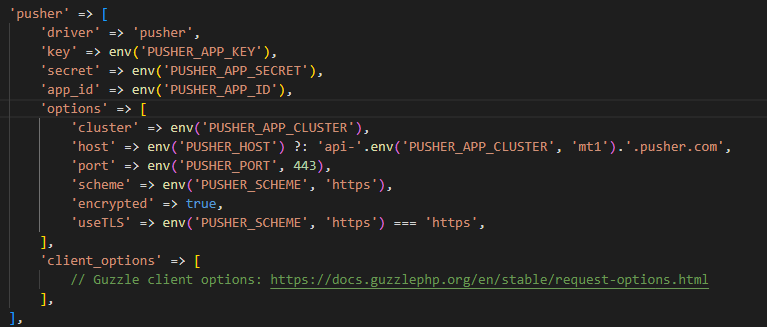
1. Uncomment below code from resources/js/bootstrap.js

****

1. Get all the App Keys from <https://dashboard.pusher.com/>

****

1. **Inside broadcasting.php**

****

1. Create an event

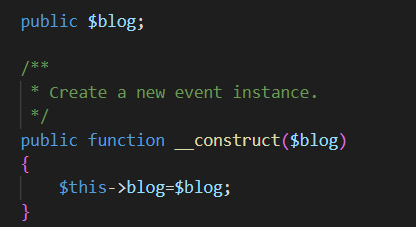
****

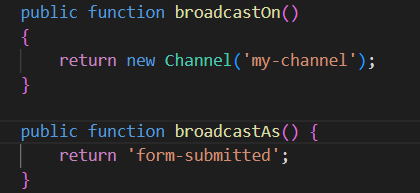
1. Inside call the event from your controller:

****

1. Inside your event:

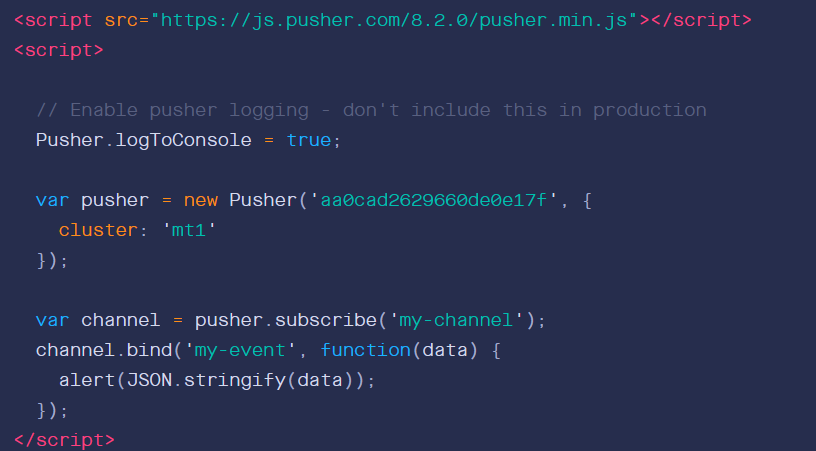
****

****

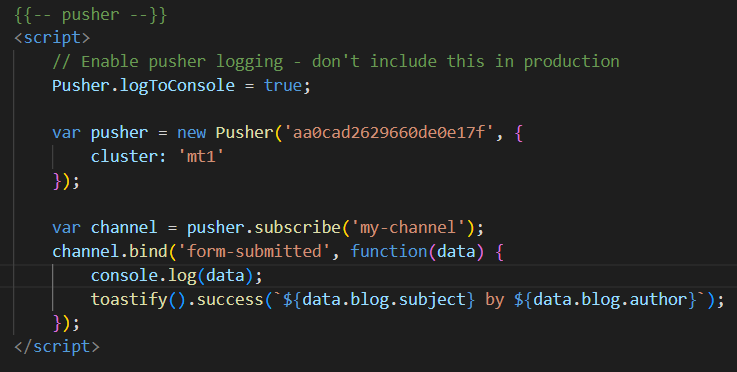
****

1. Inside any blade file where you want notification to be displayed:

->**You can get the below code from pusher dashboard in Getting Started.**

****

**Custom**

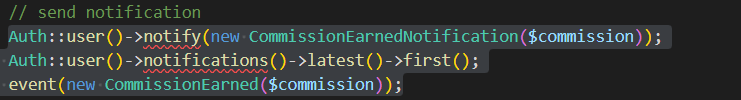
****

1. **php artisan queue:listen**

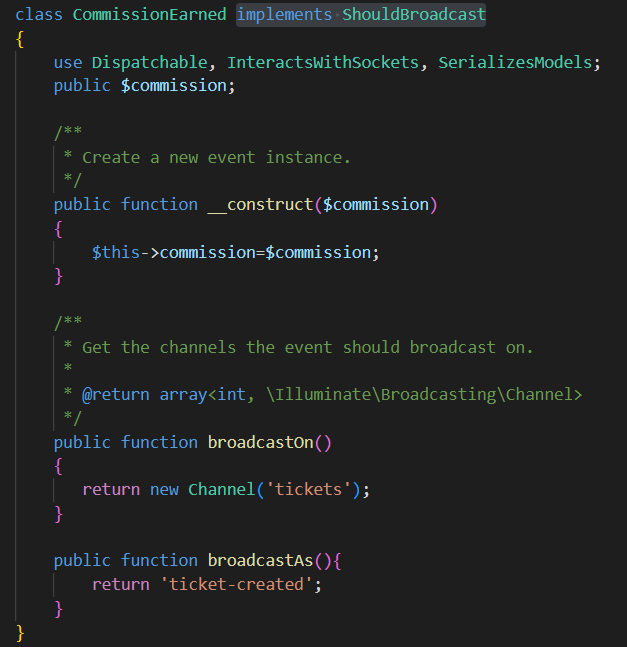
**Real Time Bell Icon Notification Using Pusher & Livewire:**

1. Follow upto step 6 from above setup.
2. Create an event : php artisan make:event:CommissionEarned and notification php artisan make:notification CommissionEarnedNotification.
3. Call event and notification from controller/livewire component.

**Send Notification to only auth user:**

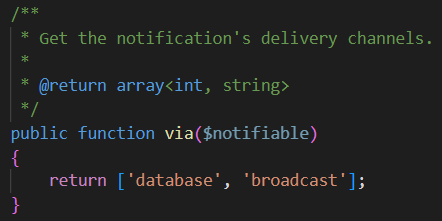


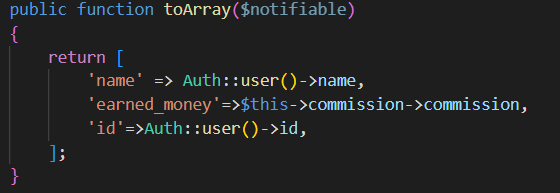
1. Inside CommissionEarned event:



1. Inside CommissionEarnedNotification.php

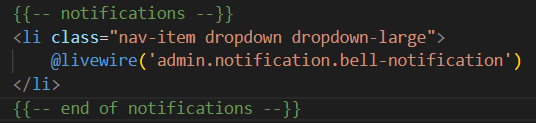




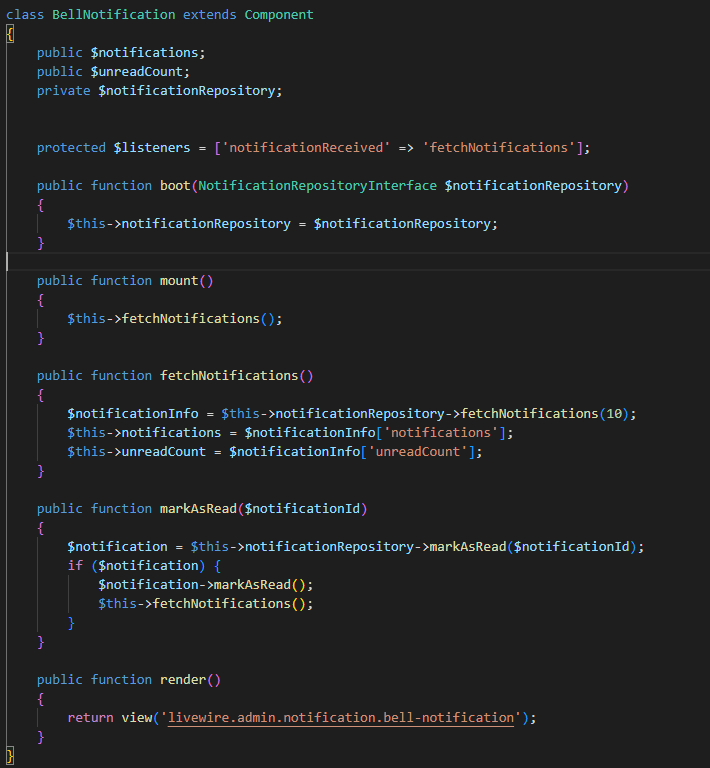




1. Inside master.blade.php



1. Create BellNotification.php livewire component.
2. Inside BellNotification.php



1. Inside NotificationRepository.php:



1. Inside bell-notification.blade.php



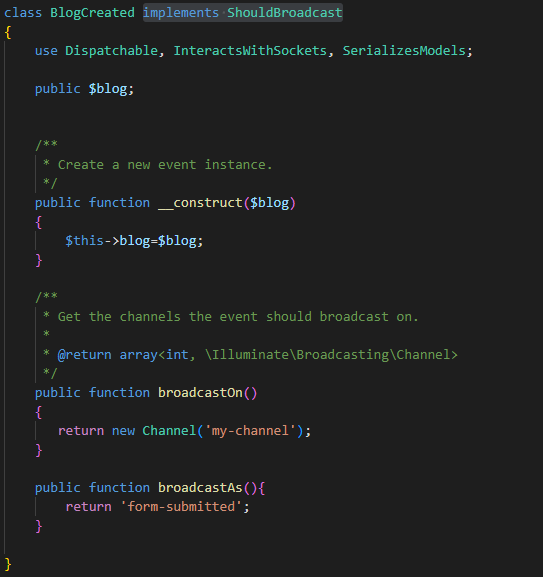


**Bell Icon Notification Using Pusher and AJAX:**

1. Follow upto step 6 from above setup.
2. Create new event: php artisan make:event BlogCreated and notification php artisan make:notification PostCreatedNotification.
3. Inside your method where you want to send notification:



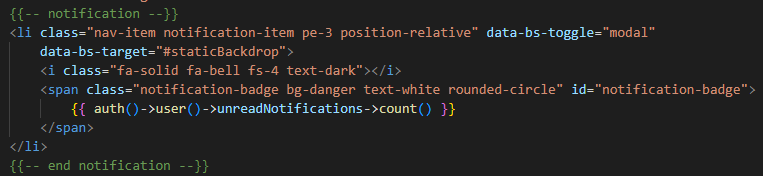
1. Inside BlogCreated event:

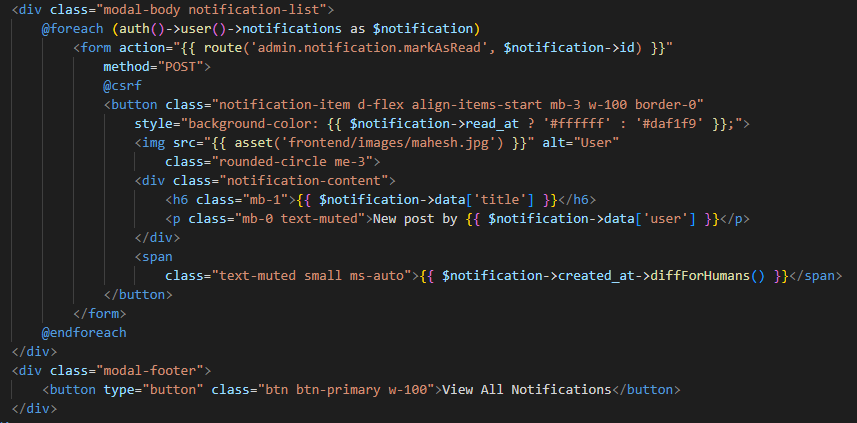


1. Inside PostCreatedNotification.php:

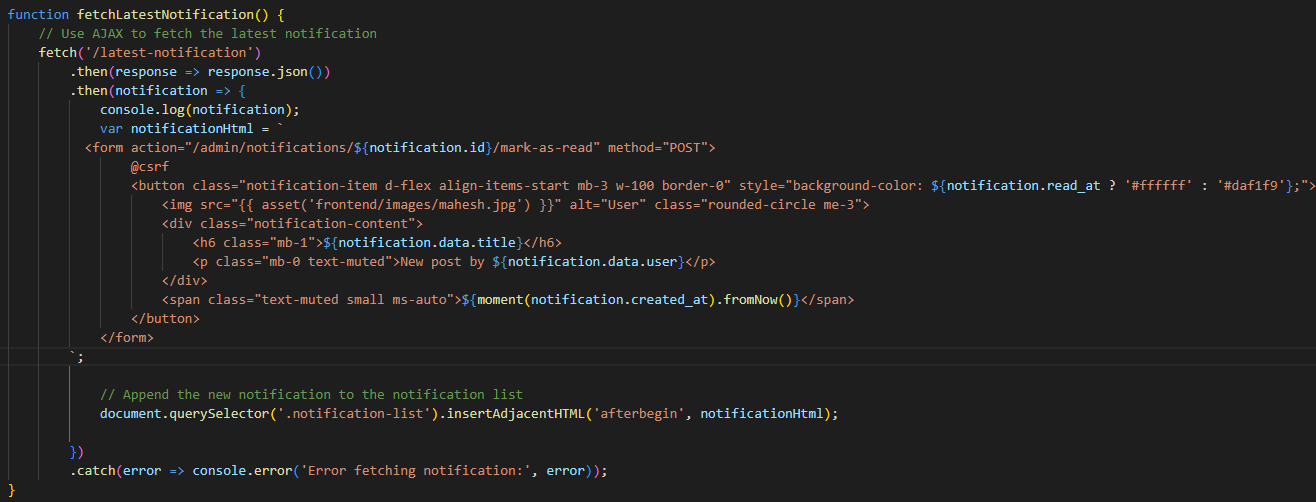


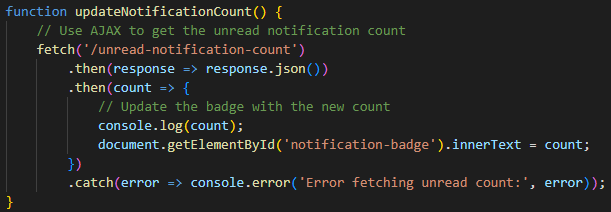
1. Inside master.blade.php



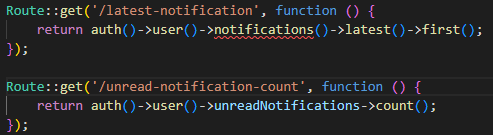




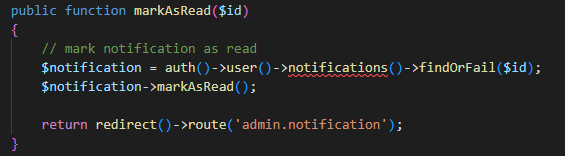




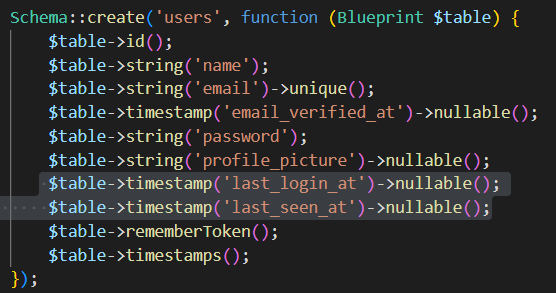
1. Inside web.php



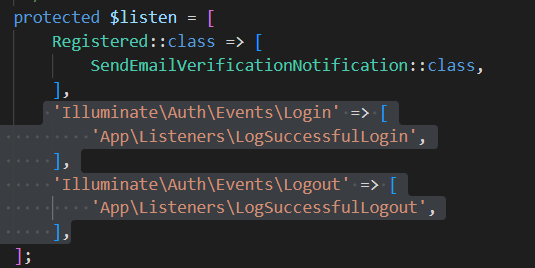
1. Inside NotificationController.php

****

1. **List all the users with his status i.e. online or offline.**
2. In Users table:



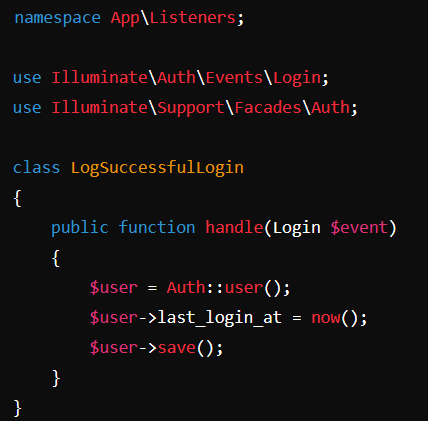
1. Inside EventServiceProvider.php:



1. Create a Listener to check login user:



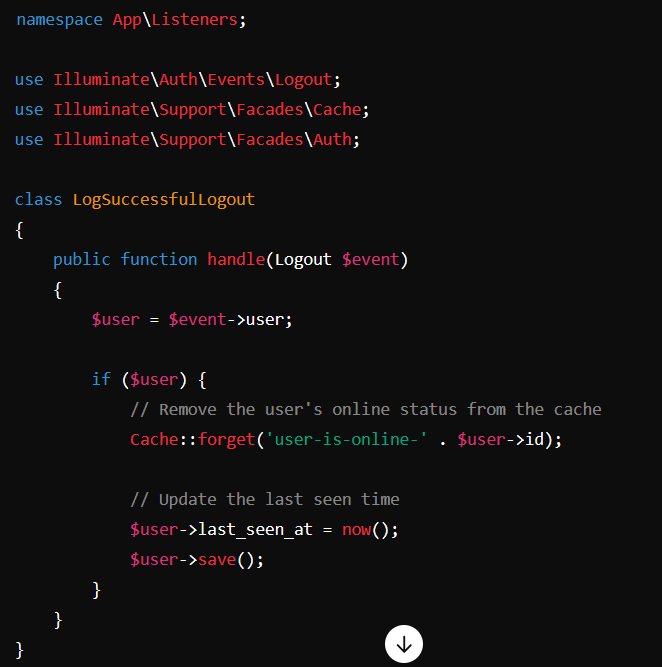
1. Inside app/Listeners/LogSuccessfulLogin.php:



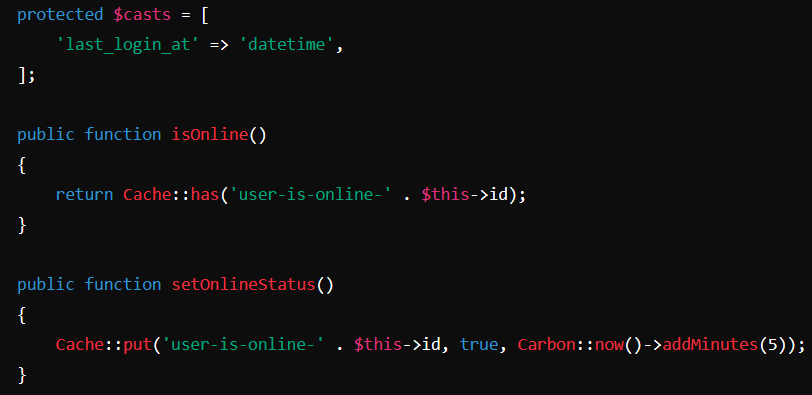
1. Create listener to check logout user:

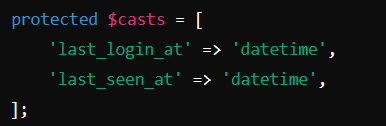


1. Inside app/Listeners/LogSuccessfulLogout.php:



1. Inside app/User.php model:

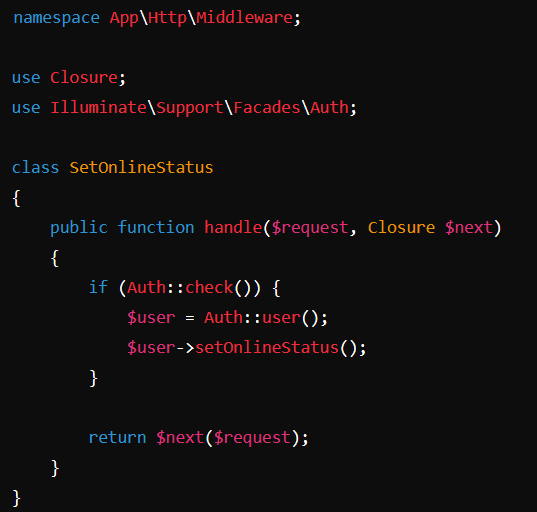




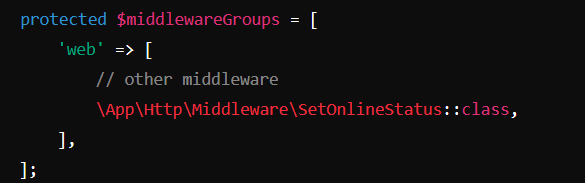
1. Create a middleware:



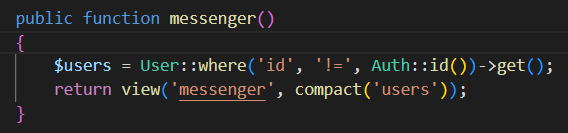
1. Inside app/Http/Middleware/SetOnlineStatus.php:



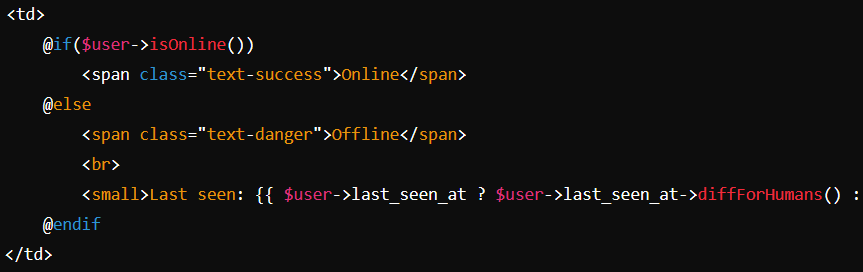
1. Inside app/Http/Kernel.php:



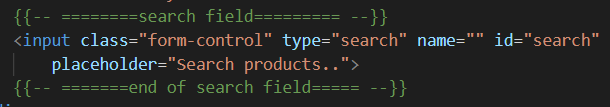
1. Inside your controller:

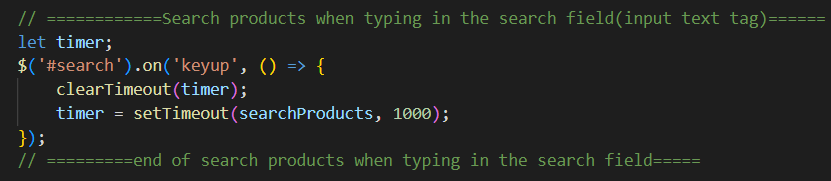


1. Inside your blade file:

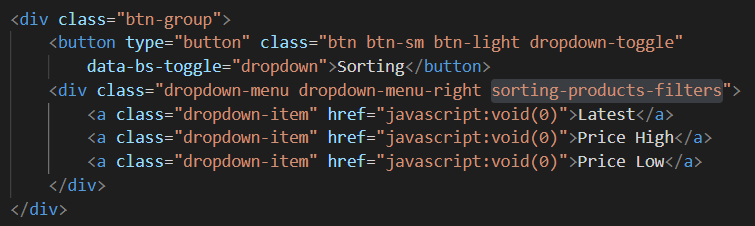


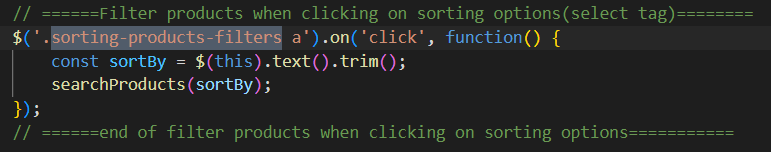
1. **Create a product table and apply all the possible filters from frontend for user using AJAX.**
2. **Filter with Searching Keyword:**



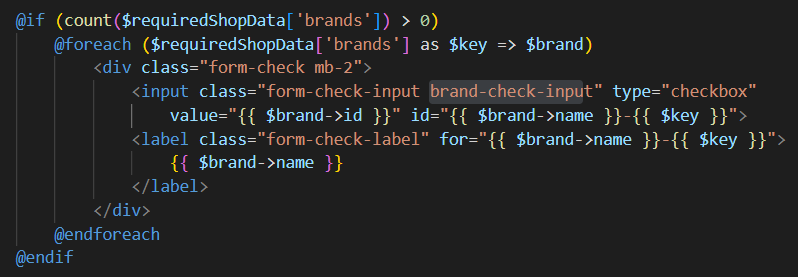
****

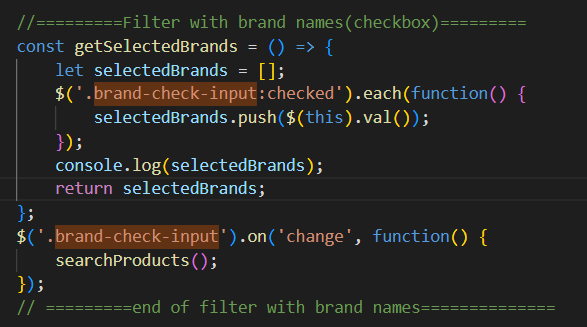
1. **Filter with Select Field:**

****

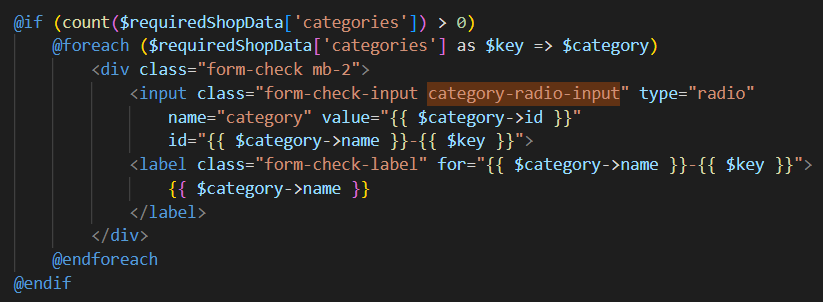
****

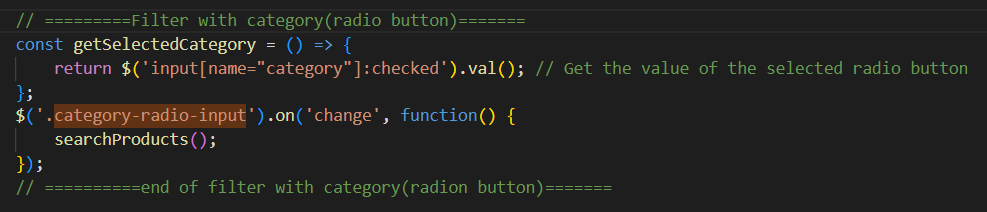
1. **Filter with Checkbox:**

****

****

1. **Filter with radio button:**

****

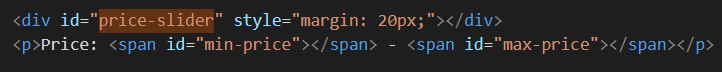
****

1. **Filter with price range:**

{{-- price range slider range --}}

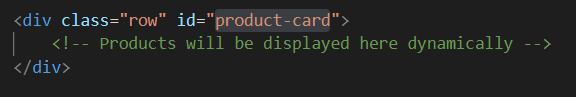
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/noUiSlider/14.6.4/nouislider.min.css" />

    <script src="https://cdnjs.cloudflare.com/ajax/libs/noUiSlider/14.6.4/nouislider.min.js"></script>



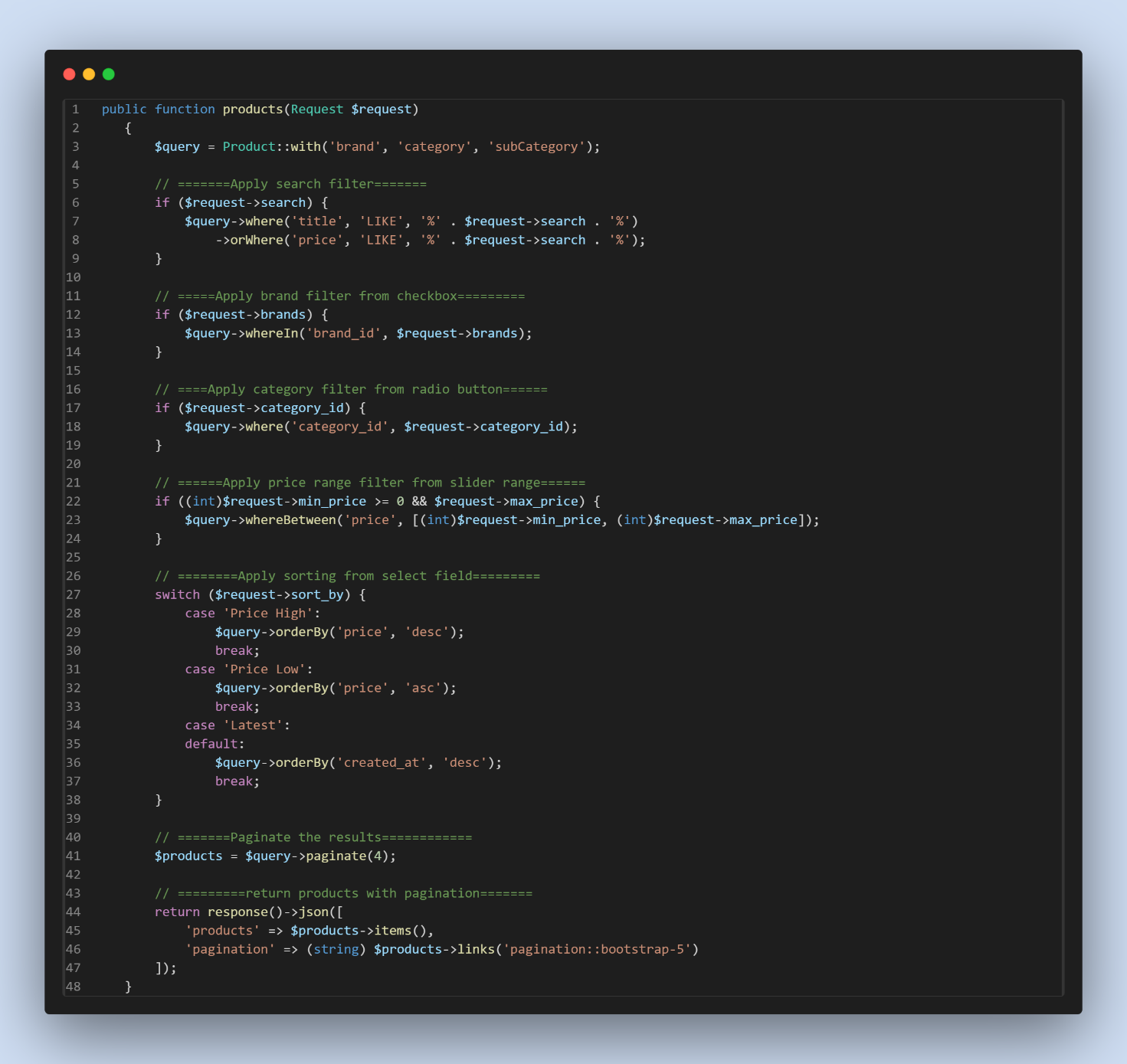


1. Ajax Call for filter and sorting and display products:





1. **Inside Controller:**

****

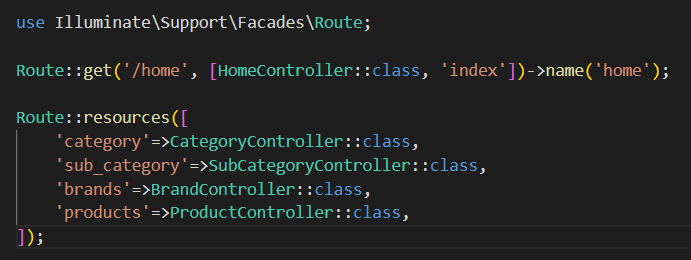
**In web.php**

****

**Inside public.php**

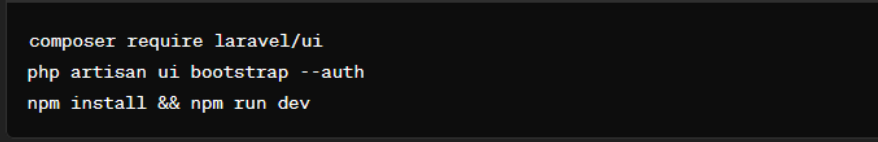
****

**Inside admin.php**

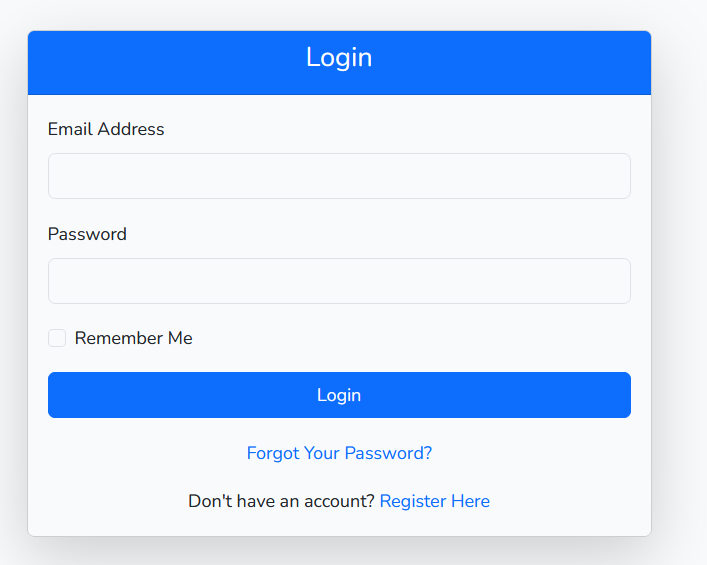
****

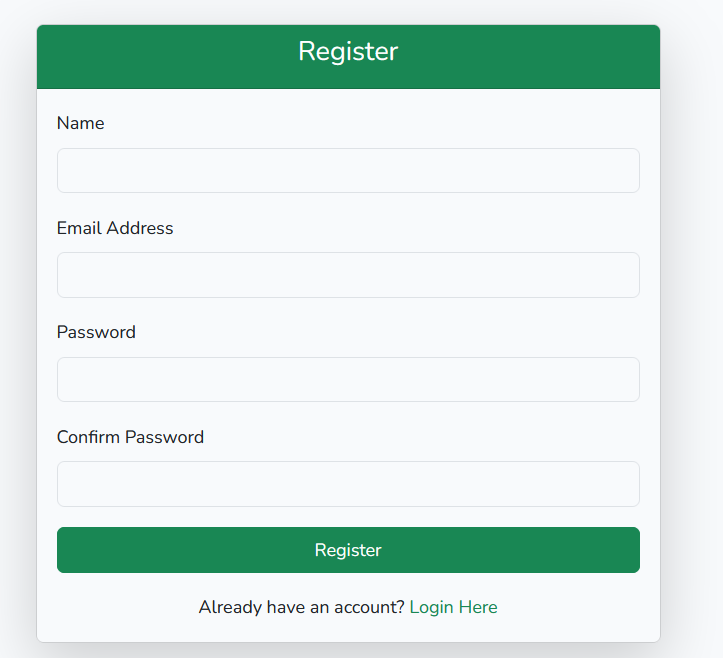
# **Authentication Using Laravel UI Package and spatie for roles & permission:**

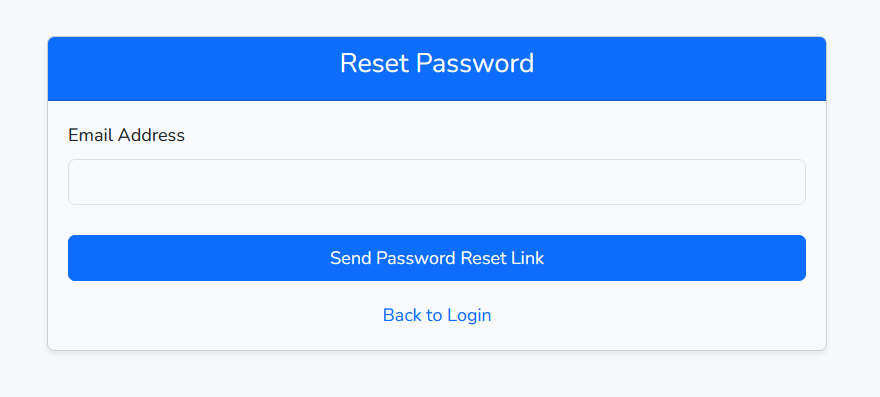
1. **Install laravel ui package**

****

1. **Change UI of login, register and password reset page as your requirements.**

****

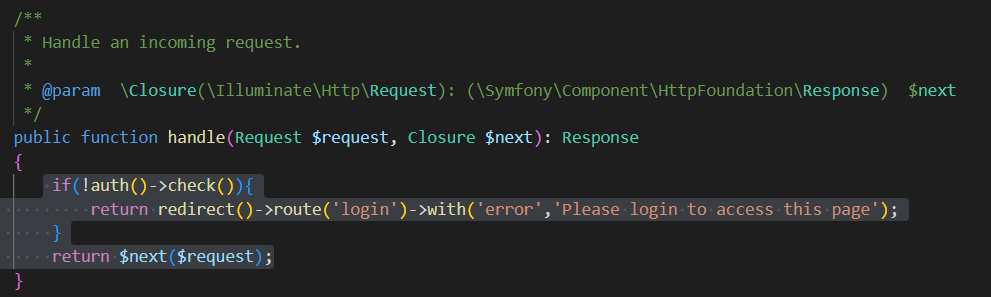
****

****

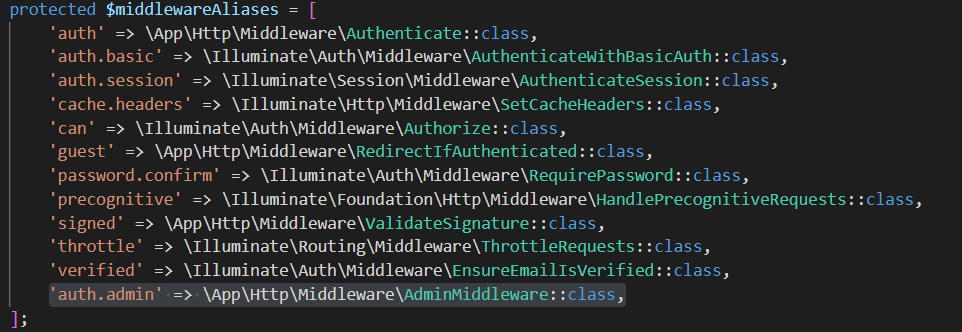
1. **Make middleware for authorization:**

Php artisan make:middleware AdminMiddleware

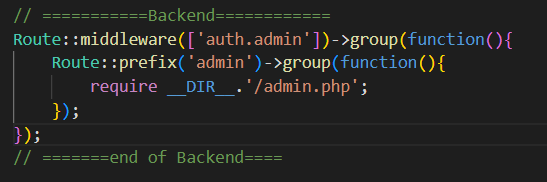
Inside AdminMiddleware

****

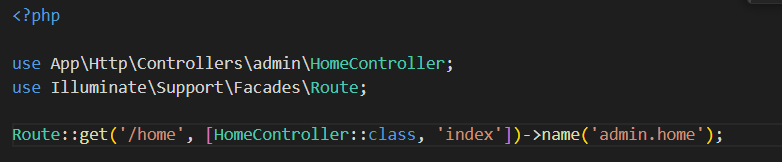
Inside app\Http\Kernel.php

****

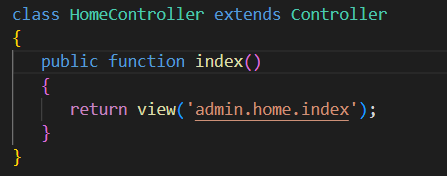
Inside Web.php

****

Inside admin.php

****

Make Admin/HomeController.php

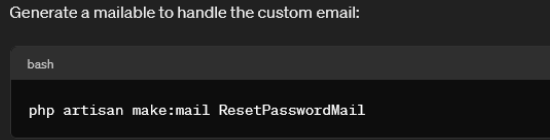


Inside admin/home/index.blade.php make admin dashboard home page

**Search ‘/home’ and change it to ‘/admin/home’**

1. **Reset password through email.**

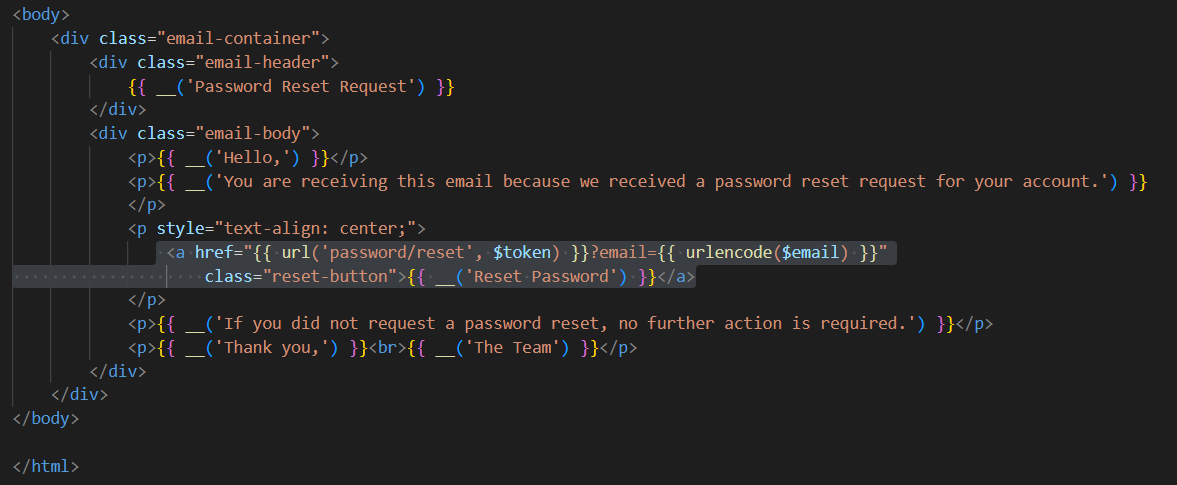
Make mail to send password reset link:

****

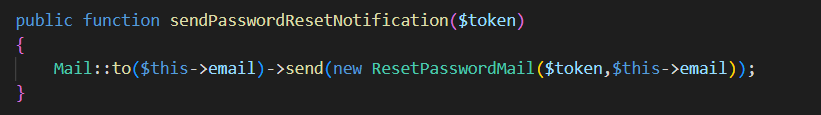
Inside ResetPasswordMail



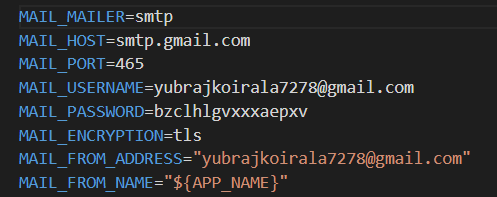
Inside views/emails/reset\_password.blade.php



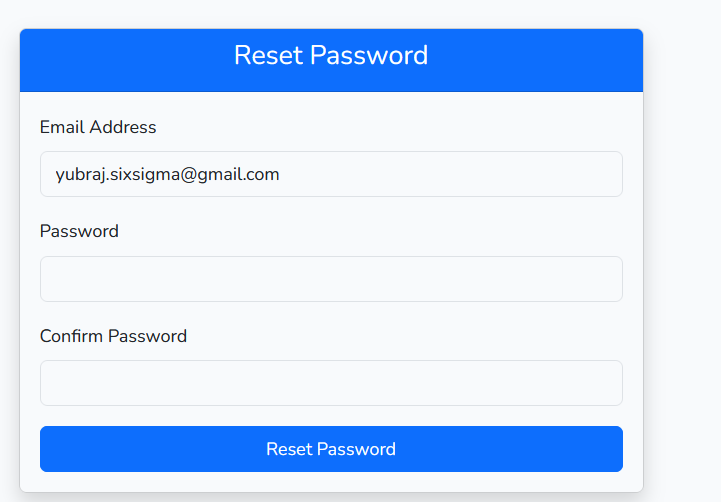
Inside user modal:



Inside .env



Now, The password reset link will come to your email account go and click to it to reset password:



Now when you change password it will be change your password.

**How to handle expire password reset link?? If the password reset link got expire that redirect him to password reset link expired page and give him option to resend password reset link again.**

Inside ResetPasswordController.php

public $email;

    public function showResetForm(Request $request, $token = null)

    {

        $this->email = $request->email;

        // Check if the token is valid or expired

        if (!$this->tokenIsValid($request->email, $token)) {

            return redirect()->route('password.expired');

        }

        // If token is valid, show the password reset form

        return view('auth.passwords.reset')->with(

            ['token' => $token, 'email' => $request->email]

        );

    }

    protected function tokenIsValid($email, $token)

    {

        if (!$token || !$email) {

            return false;

        }

        $broker = $this->broker();

        $user = $broker->getUser(['email' => $email]);

        if (!$user) {

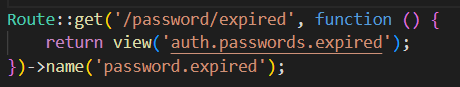
            return false;

        }

        return $broker->tokenExists($user, $token);

    }

Inside Web.php



Inside views/auth/passwords.php

@extends('layouts.app')

@section('content')

<div class="container py-5">

    <div class="row justify-content-center">

        <div class="col-md-6">

            <div class="card shadow-lg">

                <div class="card-header bg-danger text-white text-center">

                    <h5>{{ \_\_('Password Reset Link Expired') }}</h5>

                </div>

                <div class="card-body text-center">

                    <h4 class="text-danger mb-3">{{ \_\_('Oops! The password reset link has expired.') }}</h4>

                    <p class="lead text-muted mb-4">{{ \_\_('For security reasons, your reset link is no longer valid.') }}</p>

                    <p class="mb-4">{{ \_\_('Please request a new password reset link to proceed.') }}</p>

                    <!-- Button to send a new reset link -->

                    <a href="{{ route('login') }}" class="btn btn-lg btn-primary">

                        {{ \_\_('Back to login page') }}

                    </a>

                </div>

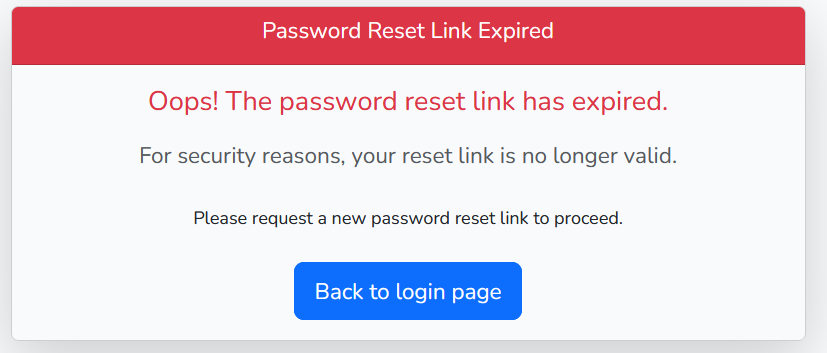
            </div>

        </div>

    </div>

</div>

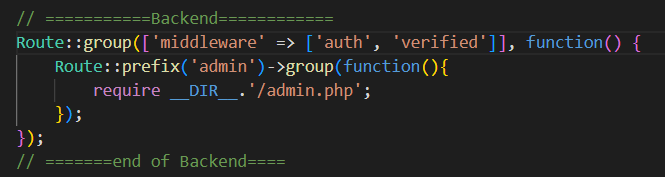
@endsection



1. **Verify Email Before Login through email:**

Inside web.php





Inside VerificationController.php

use Illuminate\Support\Facades\URL;

use Illuminate\Auth\Events\Verified;

public function verify(Request $request)

    {

        // Check if the URL is invalid due to an expired or tampered signature

        if (!URL::hasValidSignature($request)) {

            return redirect()->route('email.link.expire');

        }

        $user = $request->user();

        if ($user->hasVerifiedEmail()) {

            return redirect($this->redirectPath());

        }

        if ($user->markEmailAsVerified()) {

            event(new Verified($user));

        }

        return redirect($this->redirectPath())->with('verified', true);

    }

Make Notification:



<?php

namespace App\Notifications;

use Illuminate\Auth\Notifications\VerifyEmail as BaseVerifyEmail;

use Illuminate\Notifications\Messages\MailMessage;

use Illuminate\Support\Facades\URL;

class VerifyEmailNotification extends BaseVerifyEmail

{

    /\*\*

     \* Build the mail representation of the notification.

     \*

     \* @param  mixed  $notifiable

     \* @return \Illuminate\Notifications\Messages\MailMessage

     \*/

    public function toMail($notifiable)

    {

        $verificationUrl = $this->verificationUrl($notifiable);

        return (new MailMessage)

            ->view('emails.verify', ['user' => $notifiable, 'verificationUrl' => $verificationUrl]);

    }

    /\*\*

     \* Get the verification URL for the given notifiable.

     \*

     \* @param  mixed  $notifiable

     \* @return string

     \*/

    protected function verificationUrl($notifiable)

    {

        return URL::temporarySignedRoute(

            'verification.verify',

            now()->addMinutes(60),

            ['id' => $notifiable->getKey(), 'hash' => sha1($notifiable->getEmailForVerification())]

        );

    }

}

Inside User Modal:





/\*\*

     \* Send the email verification notification.

     \*

     \* @return void

     \*/

    public function sendEmailVerificationNotification()

    {

        $this->notify(new VerifyEmailNotification);

    }

Inside views/emails/verify.blade.php

<!DOCTYPE html>

<html lang="en">

<head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Authentication</title>

</head>

<body>

    <!--wrapper-->

    <div class="wrapper">

        <table width="100%" border="0" cellspacing="0" cellpadding="0">

            <tr>

                <td align="center" style="padding: 20px;">

                    <table class="content" width="600" border="0" cellspacing="0" cellpadding="0"

                        style="border-collapse: collapse; border: 1px solid #cccccc;">

                        <!-- Header -->

                        <tr>

                            <td class="header"

                                style="background-color: #054728; padding: 40px; text-align: center; color: white; font-size: 24px;">

                                {{-- <img src="{{ $message->embed(public\_path('assets/img/logo.png')) }}" alt="Image path"

                                    style="height: 60px; width: auto;"> --}}

                            </td>

                        </tr>

                        <!-- Body -->

                        <tr>

                            <td class="body"

                                style="padding: 30px 40px; text-align: left; font-size: 16px; line-height: 1.6;">

                                Dear <b>[{{ $user->name }}]</b>, <br><br>

                                Thank you for registering.

                            </td>

                        </tr>

                        <!-- Call to action Button -->

                        <tr>

                            <td style="padding: 0px 40px 0px 40px; text-align: center;">

                                <!-- CTA Button -->

                                <table cellspacing="0" cellpadding="0" style="margin: auto;">

                                    <tr>

                                        <td align="center"

                                            style="background-color: #054728; padding: 10px 20px; border-radius: 5px;">

                                            <a href="{{ $verificationUrl }}" target="\_blank"

                                                style="color: #ffffff; text-decoration: none; font-weight: bold;">Click

                                                here to verify</a>

                                        </td>

                                    </tr>

                                </table>

                            </td>

                        </tr>

                        <tr>

                            <td class="body"

                                style="padding: 30px 40px; text-align: left; font-size: 16px; line-height: 1.6;">

                                If the button above does not work, please copy and paste the following link into your

                                web browser: <br><br>

                                <a href="{{$verificationUrl}}" target="\_blank"

                                    style="word-break: break-all;">{{ $verificationUrl }}</a>

                                <br><br>

                                Thank you.

                                <br><br>

                                Thank you,<br>

                                Authentication  System

                            </td>

                        </tr>

                        <!-- Footer -->

                        <tr>

                            <td class="email-footer"

                                style="background-color: #333333; padding: 30px; text-align: center; color: white; font-size: 14px;">

                                <a href="#" target="\_blank" style="word-break: break-all;color:white;text-decoration:none">Support</a>

                                | <a href="#" target="\_blank" style="word-break: break-all;color:white;text-decoration:none">Visit our

                                    website</a> | <a href="#" target="\_blank"

                                    style="word-break: break-all;color:white;text-decoration:none">Log In</a>

                            </td>

                        </tr>

                    </table>

                </td>

            </tr>

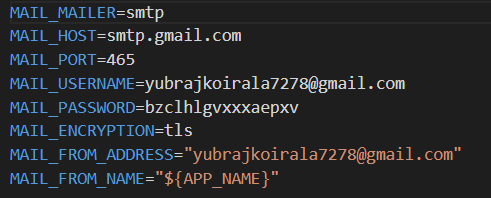
        </table>

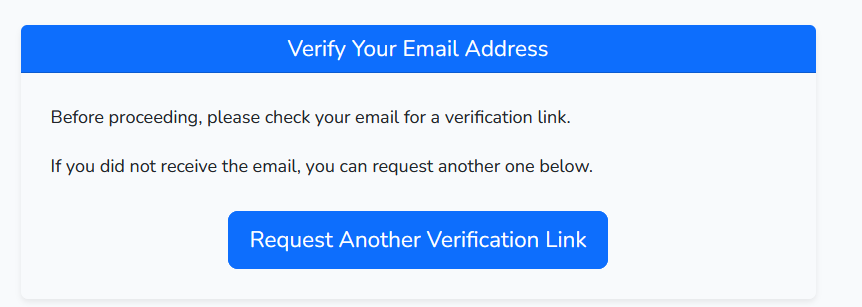
    </div>

</body>

</html>

Inside .env



Now when you register or login then this page will open:

1. **When click on checkbox remember me when login the for 7 days the email and password should be shown while login. Inside login.blade.php**

@push('script')

<script>

    document.addEventListener('DOMContentLoaded', function () {

        const loginInput = document.getElementById('email');

        const passwordInput = document.getElementById('password');

        const rememberCheckbox = document.getElementById('remember');

        // Load saved credentials

        loginInput.value = getCookie('remember\_login') || '';

        passwordInput.value = getCookie('remember\_password') || '';

        rememberCheckbox.checked = !!getCookie('remember\_me');

        // Save credentials to cookies when fields change

        loginInput.addEventListener('input', updateCookies);

        passwordInput.addEventListener('input', updateCookies);

        rememberCheckbox.addEventListener('change', updateCookies);

        function updateCookies() {

            if (rememberCheckbox.checked) {

                setCookie('remember\_login', loginInput.value, 7);

                setCookie('remember\_password', passwordInput.value, 7);

                setCookie('remember\_me', true, 7);

            } else {

                deleteCookie('remember\_login');

                deleteCookie('remember\_password');

                deleteCookie('remember\_me');

            }

        }

        // Cookie handling functions

        function setCookie(name, value, days) {

            const d = new Date();

            d.setTime(d.getTime() + (days \* 24 \* 60 \* 60 \* 1000));

            const expires = "expires=" + d.toUTCString();

            document.cookie = `${name}=${encodeURIComponent(value)};${expires};path=/`;

        }

        function getCookie(name) {

            const nameEQ = name + "=";

            const ca = document.cookie.split(';');

            for (let i = 0; i < ca.length; i++) {

                let c = ca[i].trim();

                if (c.indexOf(nameEQ) === 0) {

                    return decodeURIComponent(c.substring(nameEQ.length, c.length));

                }

            }

            return null;

        }

        function deleteCookie(name) {

            document.cookie = `${name}=; Max-Age=-99999999; path=/`;

        }

    });

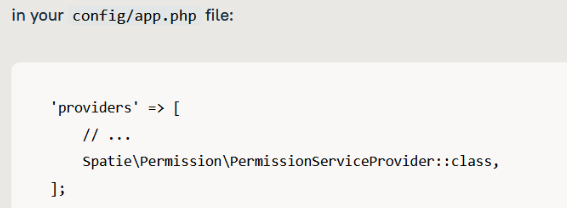
</script>

@endpush

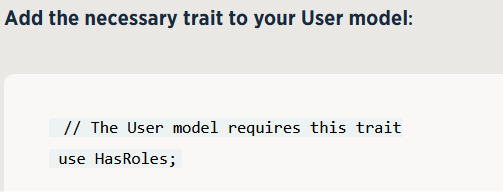
1. **Create three roles admin,articles\_manager and news\_manager. The articles\_manager have only rights to access article related page and dashboard home page. Same logic for news manager. And admin has rights to access all the page.**

<https://spatie.be/docs/laravel-permission/v6/installation-laravel>

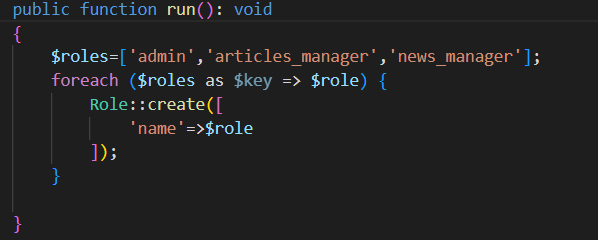
****

****

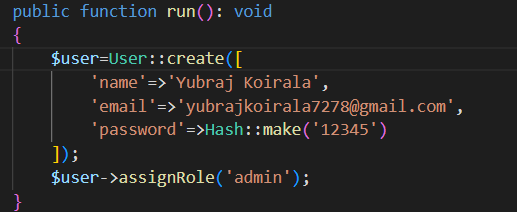
****



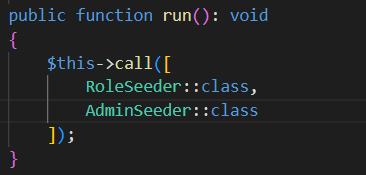
Php artisan make:seeder RoleSeeder



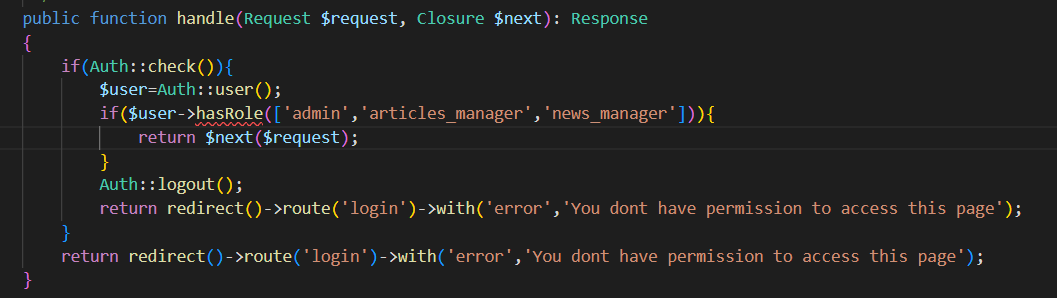
Php artisan make:seeder AdminSeeder



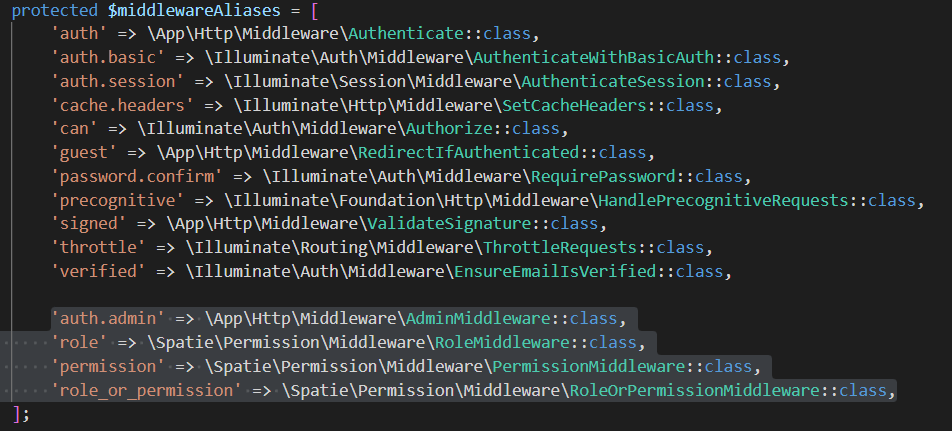
Inside DatabaseSeeder.php



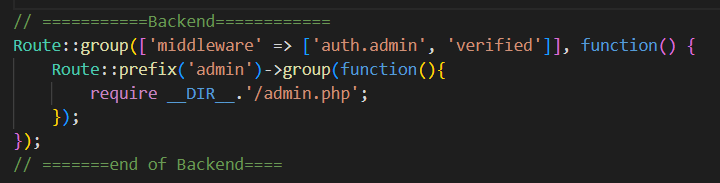
Php artisan make:middleware AdminMiddleware



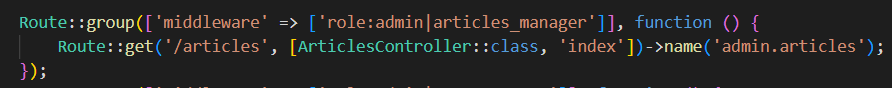
Inside Http/kernel.php



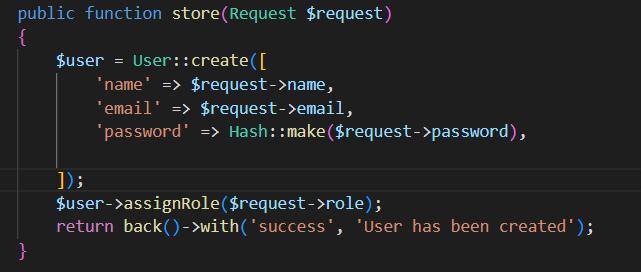
Web.php



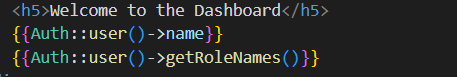
Admin.php



For user create



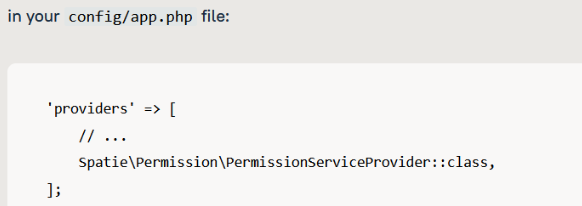
Get role names and auth name in blade



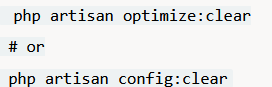
1. **Create a roles with dynamic permissions i.e. the admin should be able to create any roles and he has ability to assign any permissions to any roles.**

<https://spatie.be/docs/laravel-permission/v6/installation-laravel>

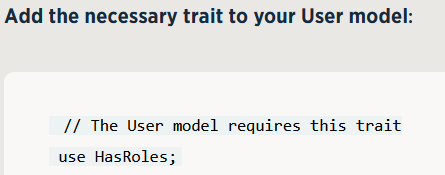




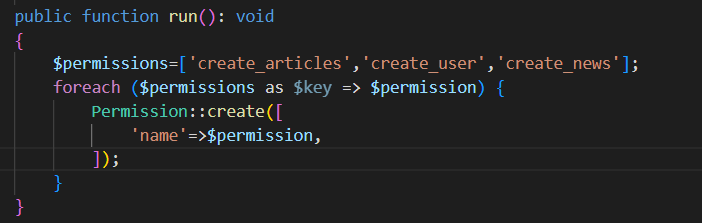




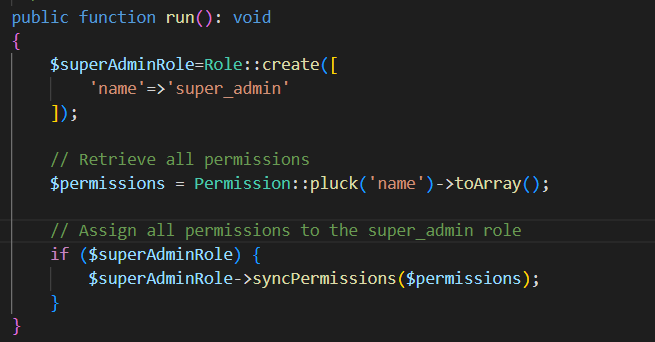




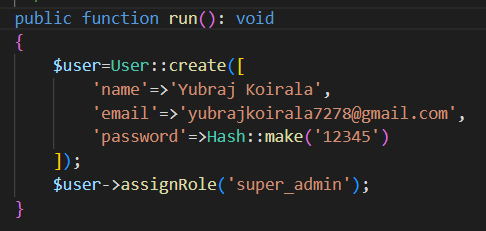
Create a PermissionSeeder:



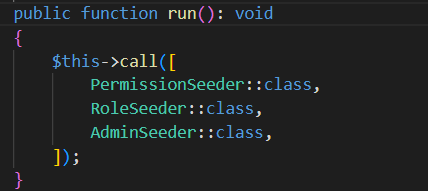
Create RoleSeeder to create super\_admin role and assign all the permissions to super\_admin role:



Create AdminSeeder to create one user and assign him as super\_admin role:

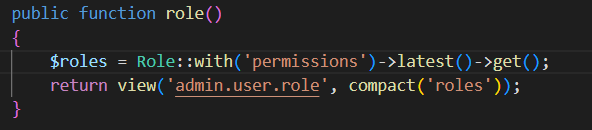


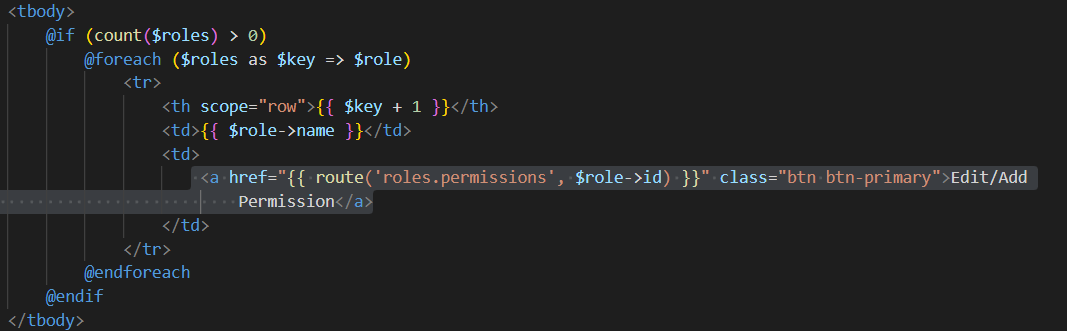
Inside DatabaseSeeder



Create a roles page to display all the role and add button to add/edit permission for that role:

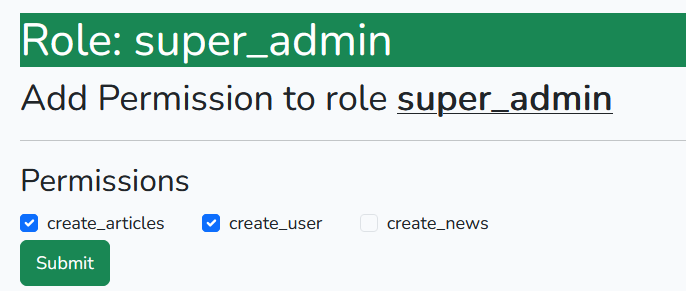


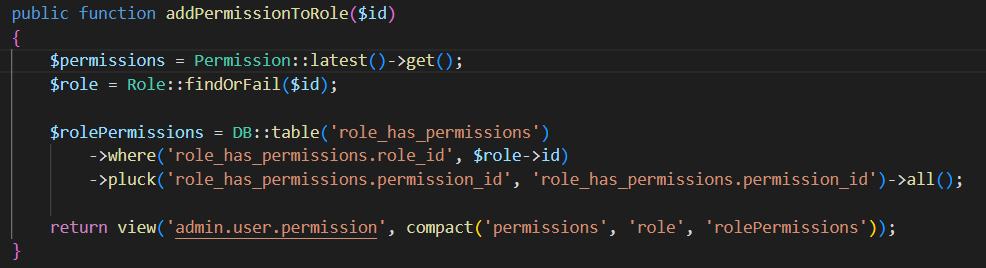


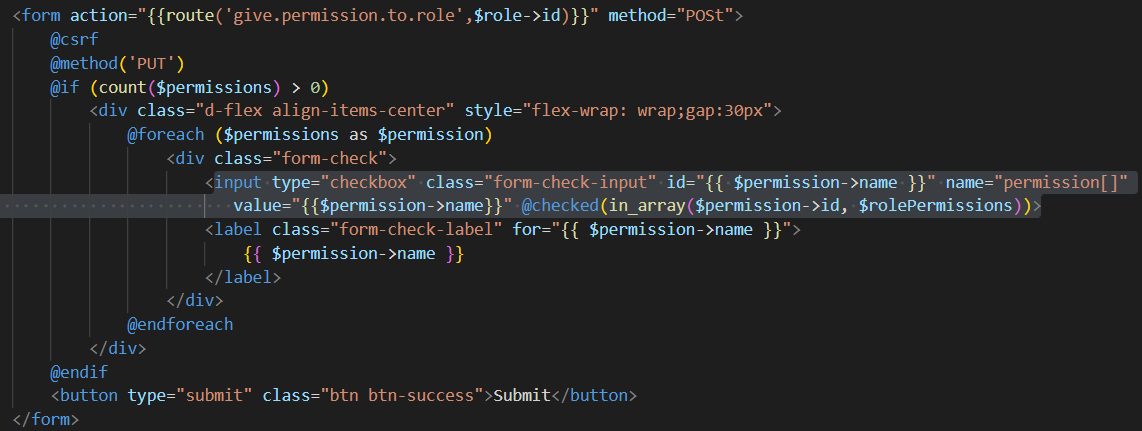




Now, When click on Add/Edit Permission then open another page to add/edit the permissions for that specific role:

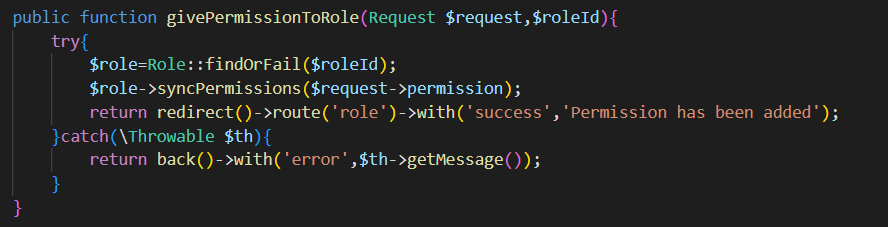






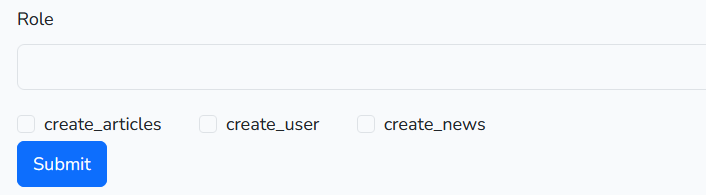
Now when click on submit the permissions selected should be assigned to that role.

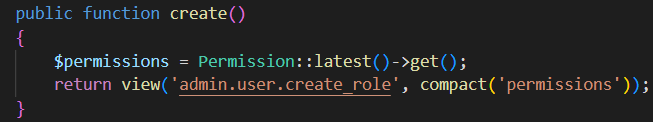




Note: You can create multiple roles and assign permissions as your requirements. And you can also create multiple users and assign multiple roles as your requirements.

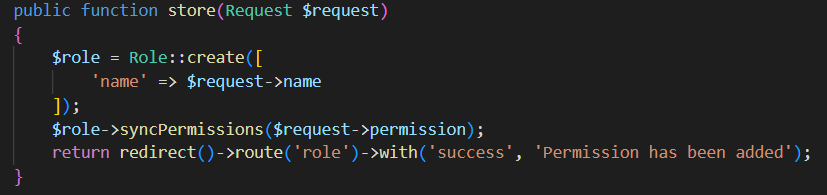
Now, Let’s create another role Manager and give him some permissions. When click on Add New Role btn the below ui page should be opened:







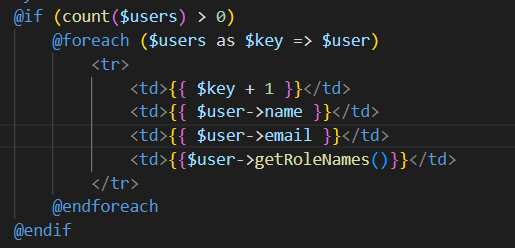
Now When clicked on submit button:



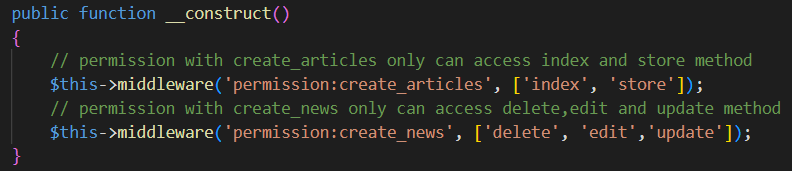
Congratulations a new role is created with selected permissions assigned to him.

Now In users table you can display users with his roles by writing below code:





Now, Inside your controller



Or



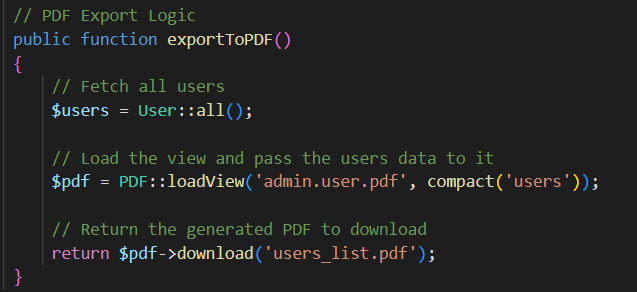
Here, If the role has create\_articles permission then he can access index method of ArticlesController.

1. **Write a logic to export database data to pdf,excel and csv file. Also write logic to import excel file to insert into database table.**
2. **Export to pdf:**

****

Inside your controller:

use Barryvdh\DomPDF\Facade\Pdf;

****

Inside views/admin/user/pdf.blade.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Users List</title>

    <style>

        table {

            width: 100%;

            border-collapse: collapse;

        }

        table, th, td {

            border: 1px solid black;

        }

        th, td {

            padding: 8px;

            text-align: left;

        }

    </style>

</head>

<body>

    <h1>Users List</h1>

    <table>

        <thead>

            <tr>

                <th>Name</th>

                <th>Email</th>

                <th>Created At</th>

            </tr>

        </thead>

        <tbody>

            @foreach($users as $user)

                <tr>

                    <td>{{ $user->name }}</td>

                    <td>{{ $user->email }}</td>

                    <td>{{ $user->created\_at }}</td>

                </tr>

            @endforeach

        </tbody>

    </table>

</body>

</html>

Inside admin.php



In your blade file:



1. **Export to excel:**





Inside your controller:

use App\Exports\UsersExport;

use Maatwebsite\Excel\Facades\Excel;

  public function exportToExcel()

    {

        $users=User::all();

        // Trigger the Excel export and download the file

        return Excel::download(new UsersExport($users), 'users.xlsx');

    }

Inside UsersExports.php

<?php

namespace App\Exports;

use App\Models\User;

use Maatwebsite\Excel\Concerns\FromCollection;

use Maatwebsite\Excel\Concerns\WithHeadings;

use Illuminate\Support\Collection;

class UsersExport implements FromCollection, WithHeadings

{

    protected $users;

    public function \_\_construct(Collection $users)

    {

        $this->users = $users;

    }

    /\*\*

    \* @return \Illuminate\Support\Collection

    \*/

    public function collection()

    {

        return $this->users;

    }

     /\*\*

     \* @return array

     \*/

    public function headings(): array

    {

        return [

            'ID',

            'Name',

            'Email',

            'Created At',

        ];

    }

}

Inside admin.php

****

Inside your blade file:

****

1. **Export to CSV File:**

All process as above to export in excel but change the controller method like below:

 public function exportToCSV()

    {

        $users = User::all();

        // Trigger the CSV export and download the file

        return Excel::download(new UsersExport($users), 'users.csv');

    }

1. **Import excel file and insert it into users table:**

****

**Inside UsersImport.php**

<?php

namespace App\Imports;

use App\Models\User;

use Maatwebsite\Excel\Concerns\ToModel;

use Maatwebsite\Excel\Concerns\WithHeadingRow;

use Illuminate\Support\Facades\Hash;

class UsersImport implements ToModel, WithHeadingRow

{

    /\*\*

     \* @param array $row

     \* @return \Illuminate\Database\Eloquent\Model|null

     \*/

    public function model(array $row)

    {

        // Assuming your Excel file has columns like name, email, and password

        return new User([

            'name' => $row['name'],  // Adjust according to the column name in your Excel file

            'email' => $row['email'],

            'password' => Hash::make($row['password']),  // Hash password before storing it

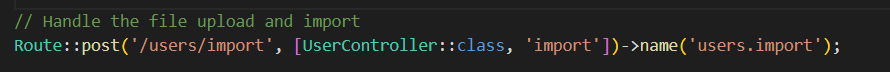
            // Add any other fields you need here

        ]);

    }

}

**Inside admin.php**

****

**Inside your controller:**

// Handle the file upload and import

    public function import(Request $request)

    {

        $request->validate([

            'file' => 'required|mimes:xlsx,csv',  // Validate that the file is an Excel or CSV

        ]);

        // Import the data

        try {

            Excel::import(new UsersImport, $request->file('file'));

            return redirect()->route('admin.user.index')->with('success', 'Users imported successfully!');

        } catch (\Exception $e) {

            return back()->withErrors(['error' => 'There was an issue importing the file.']);

        }

    }

**Inside your blade file:**

<form action="{{ route('users.import') }}" method="POST" enctype="multipart/form-data">

            @csrf

            <!-- File Upload -->

            <div class="mb-3">

                <label for="file" class="form-label">Upload Excel File</label>

                <input type="file" class="form-control" name="file" id="file" required>

            </div>

            <!-- Submit Button -->

            <button type="submit" class="btn btn-primary">Import</button>

        </form>

1. **Laravel authentication with socialite:**
2. Login with google:

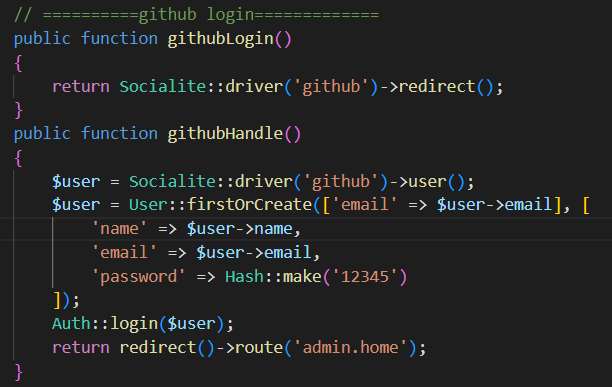
<https://www.youtube.com/watch?v=kleqfI8-0Mo>



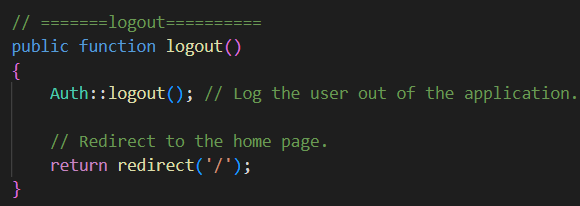
1. Login with github

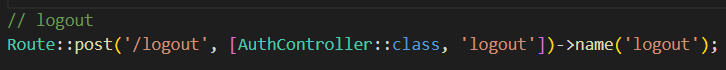
<https://www.youtube.com/watch?v=CbwzjLgcvR4>

<https://www.youtube.com/watch?v=v0IU0zSA6mw>

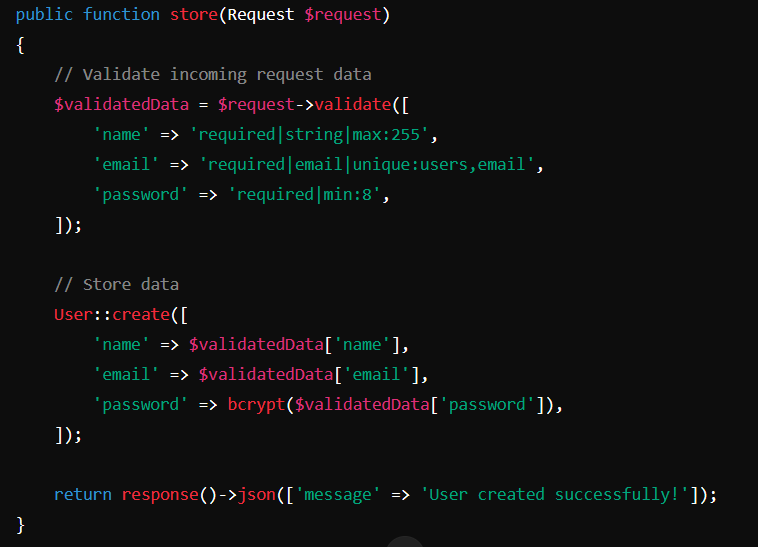


1. Logout



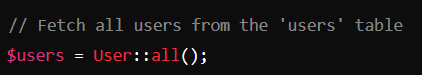


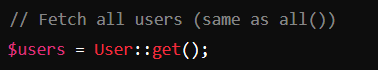
1. **laravel all eloquent orm model concept.**
2. **How to save data into table?**



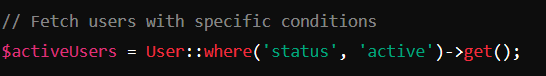
1. **How to retrieve users data from database?**

* Retrieve all records

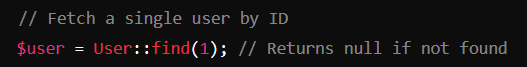




* Apply constraints before fetching.



* Retrieve a single record by its primary key.

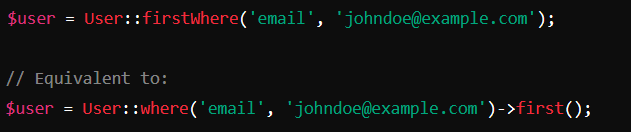


* Retrieve a record by primary key and throw an exception if not found.

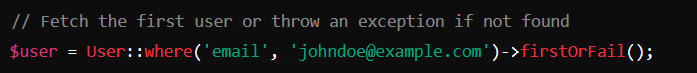


* Retrieve the first matching record.

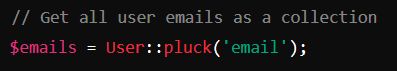


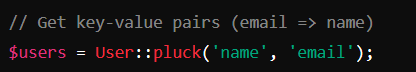


* Retrieve the first matching record or throw an exception.

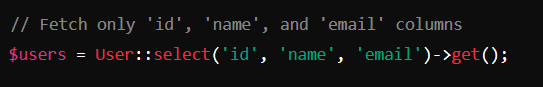


* Retrieve specific column values as a collection.



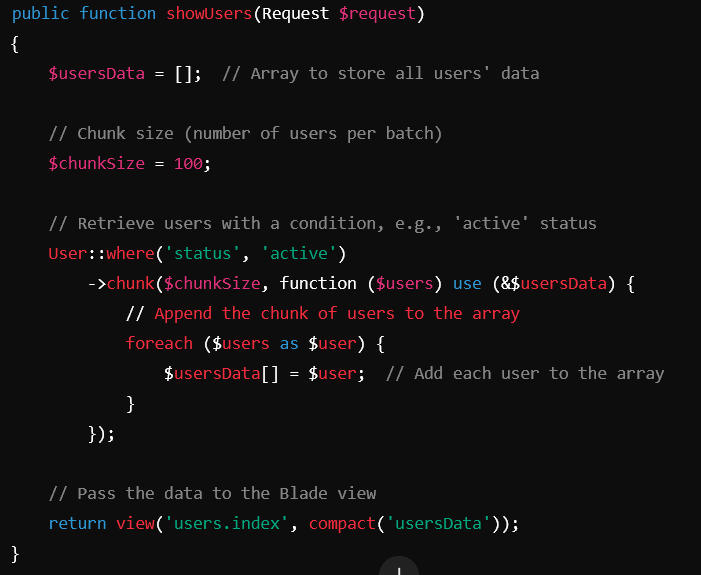


* Select specific columns from the table.

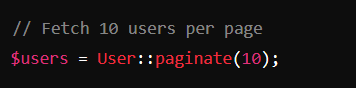


* Process records in chunks to handle large datasets efficiently.

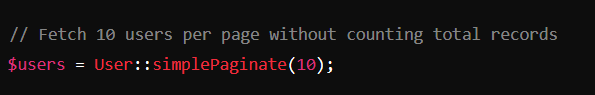




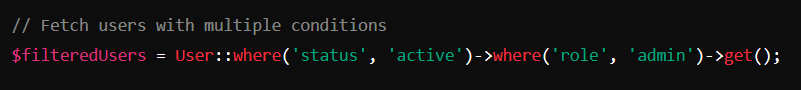
* Retrieve records with pagination.



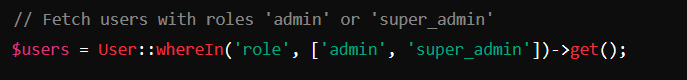
* Retrieve records with simple pagination (no total count).



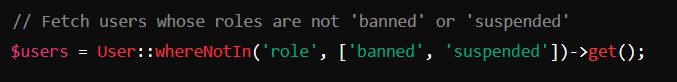
* Apply filtering conditions.



* Filter records where a column matches any value in an array.

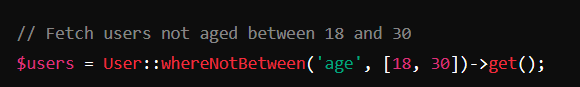


* Filter records where a column does not match any value in an array.

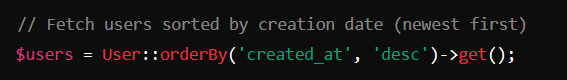


* Filter records based on a range.

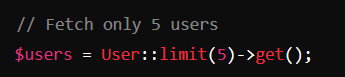




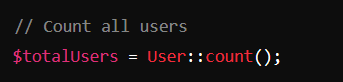
* Sort records by a specific column.

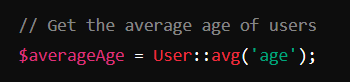


* Limit the number of records retrieved.

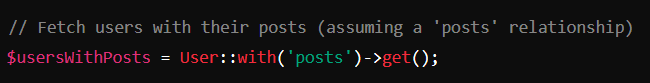


* Retrieve aggregate values like count, sum, average, etc.

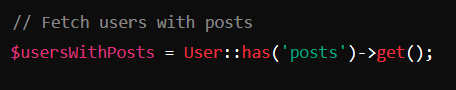




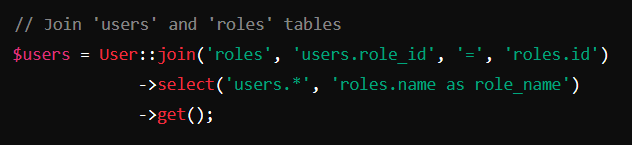
* Retrieve related data via Eloquent relationships.



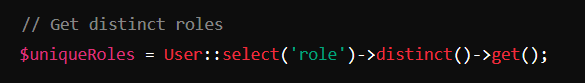
* Filter based on the existence of a relationship.



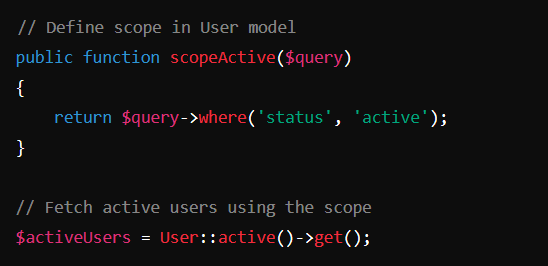
* Join with another table to fetch combined data.



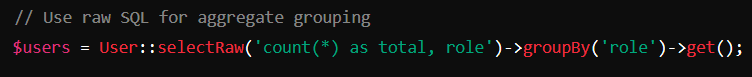
* Retrieve unique records.



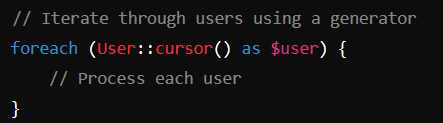
* Define reusable query logic in the model.

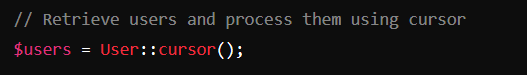


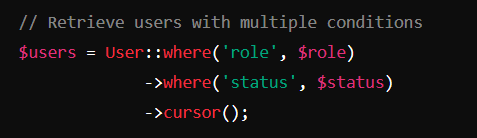
* Execute raw SQL queries for complex use cases.

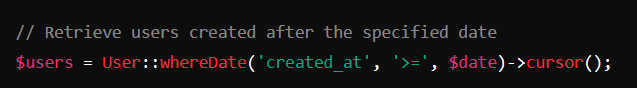


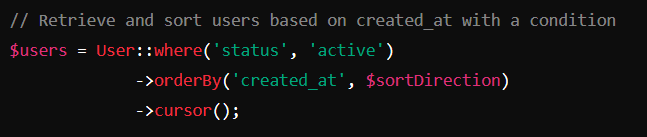
* Iterate through records one at a time with low memory usage.



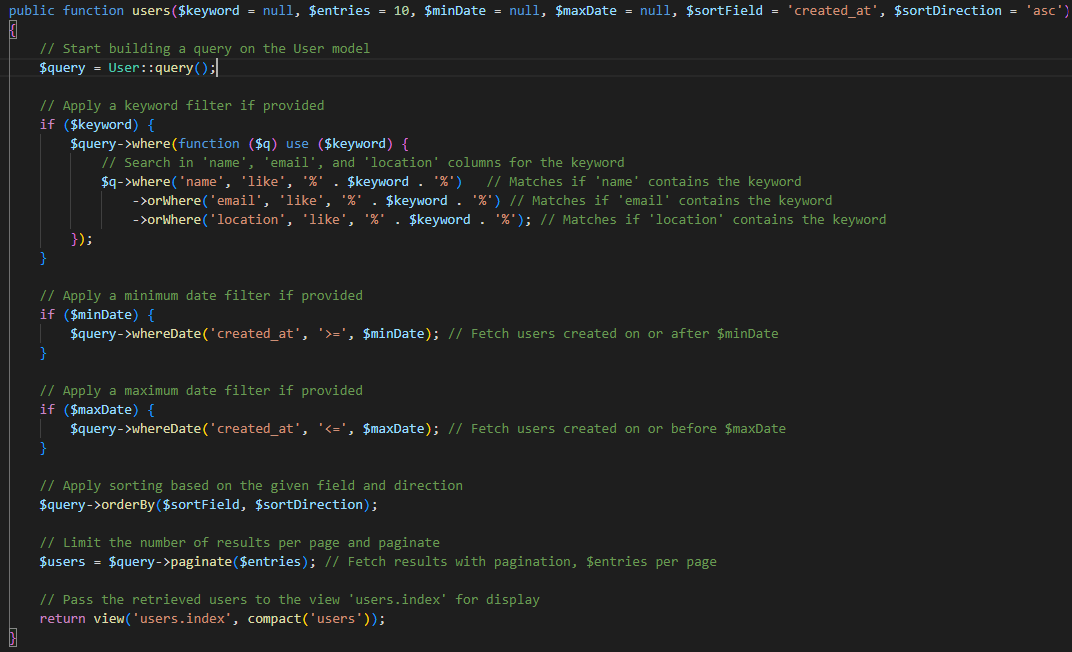




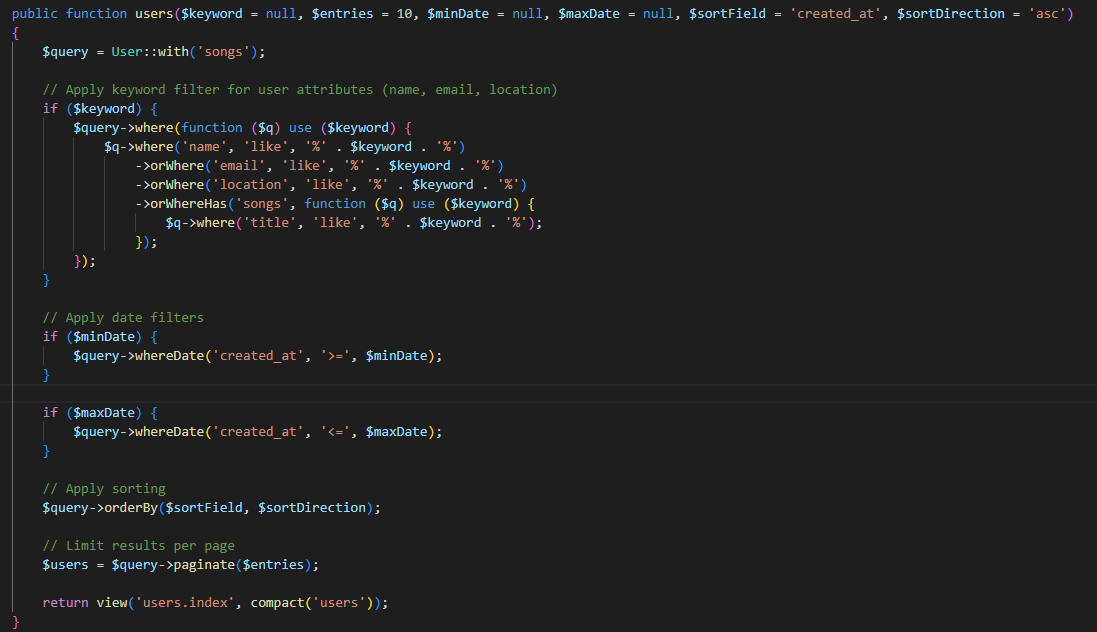




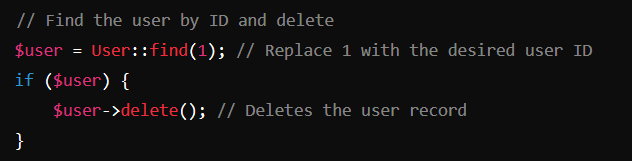
* Retrieve users data with dynamic filters:

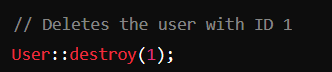


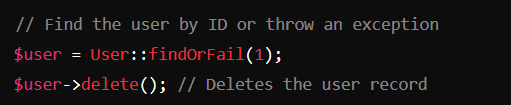
* Retrieve users data with relationship song with dynamic filters.

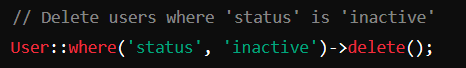


1. **How to delete users data from database?**

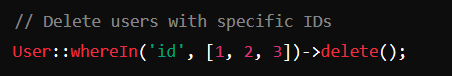
* Delete a Record by ID

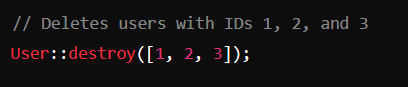


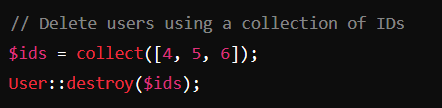
* Delete a Record Using findOrFail
* Delete a Record Using where



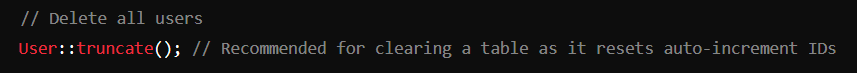
* Delete Multiple Records Using wherein



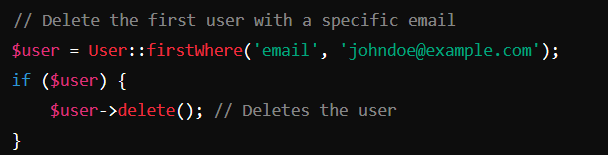


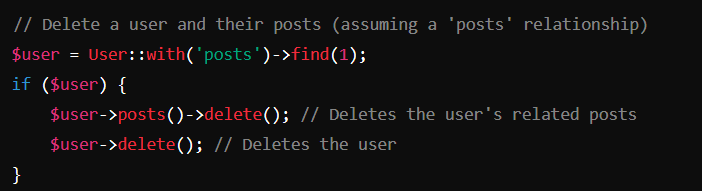


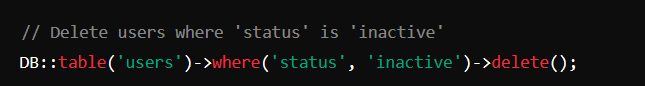
* Delete All Records



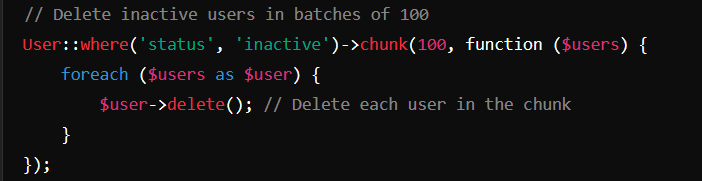
* Find the first matching record and delete it.



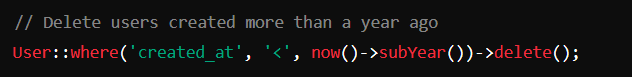
* Delete Using Relationships
* Delete records without loading models by using Query Builder



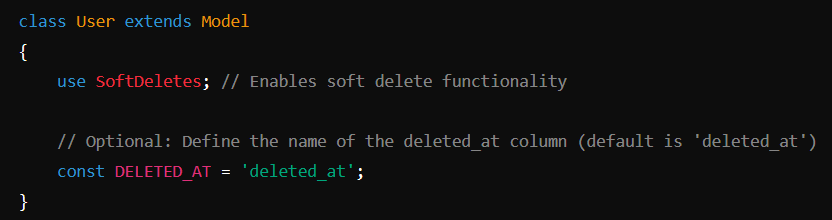
* Delete records in chunks to handle large datasets.

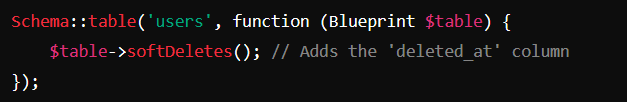


* Use advanced queries for conditional deletion.

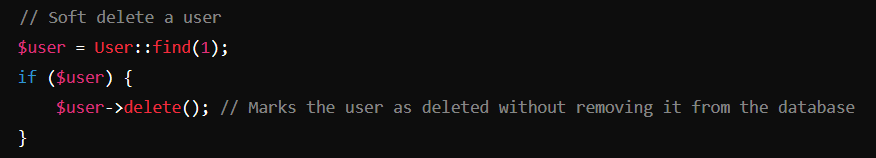


* Soft Delete





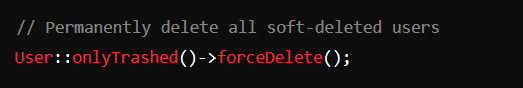
Delete Records with Soft Deletes



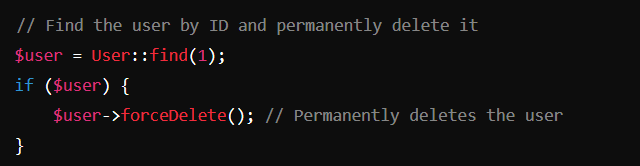
You can restore soft-deleted records using the restore method.



Delete All Soft Deleted Records



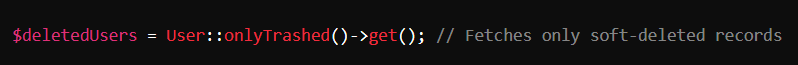
If the model uses soft deletes, use forceDelete to remove the record permanently.



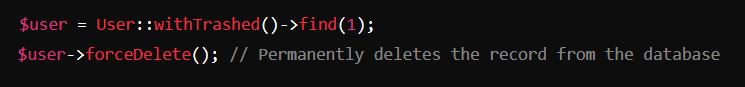
Retrieve all the users with Soft-Deleted Records



Retrieve only Soft-Deleted Records:

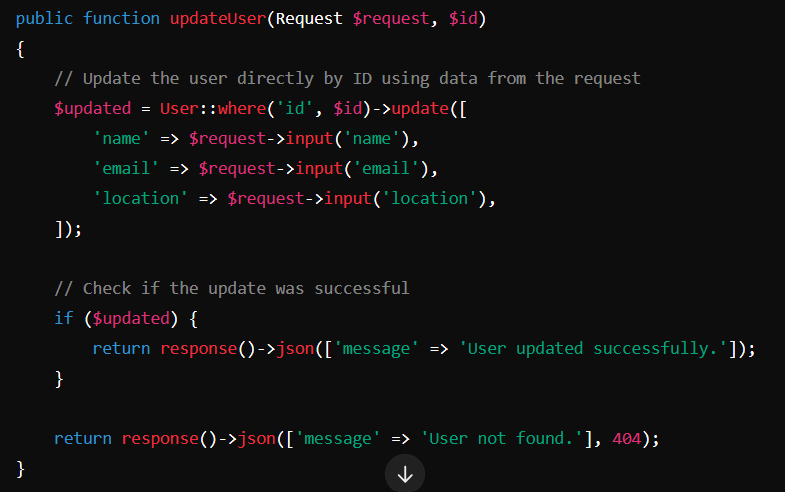


Completely remove a soft-deleted record from the database:

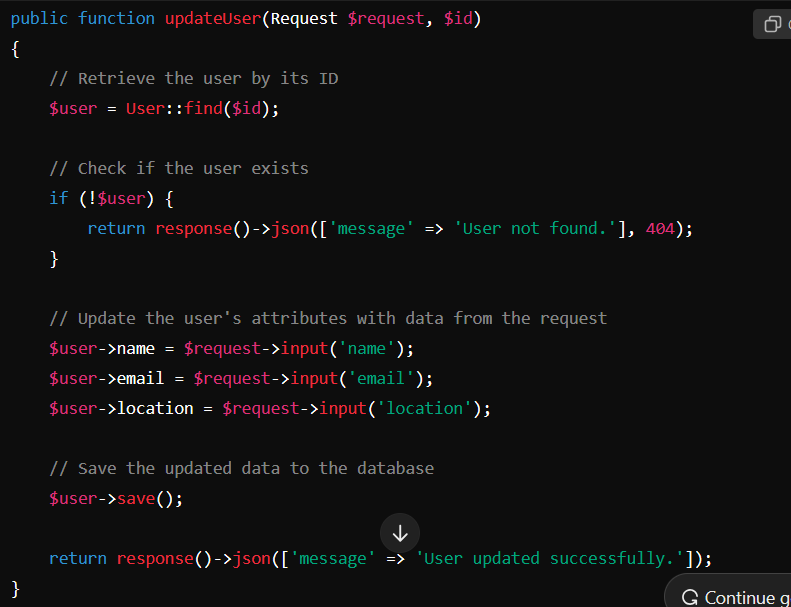


1. **How to update data from database?**

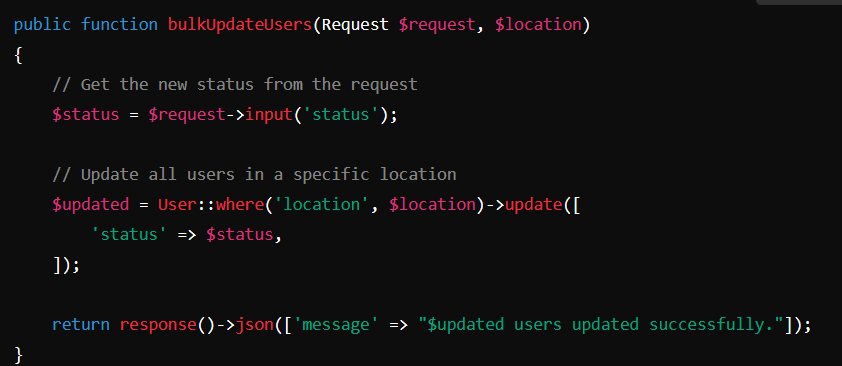
* Update Using the update() Method (with Request)



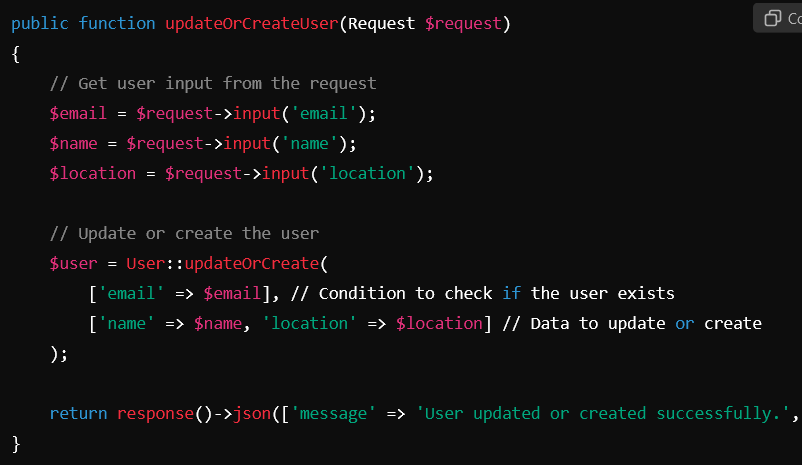
* Update Using Model Instance (with Request)



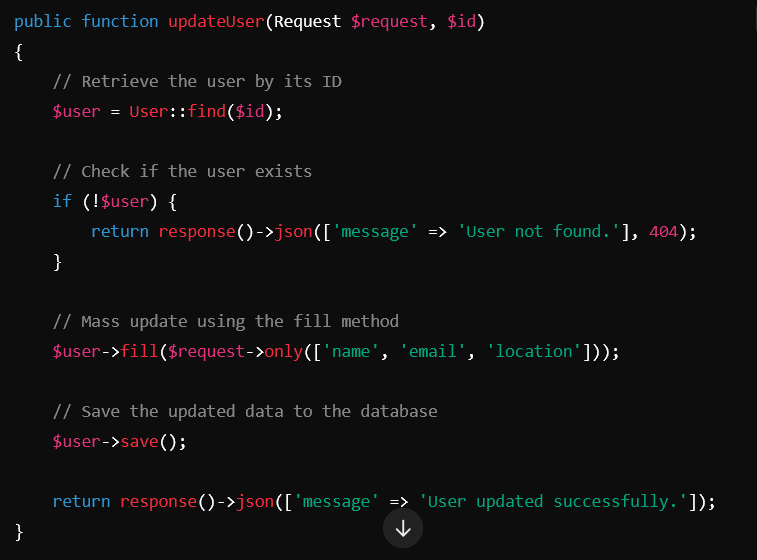
* You can update multiple records at once based on a specific condition, which is efficient for batch processing.



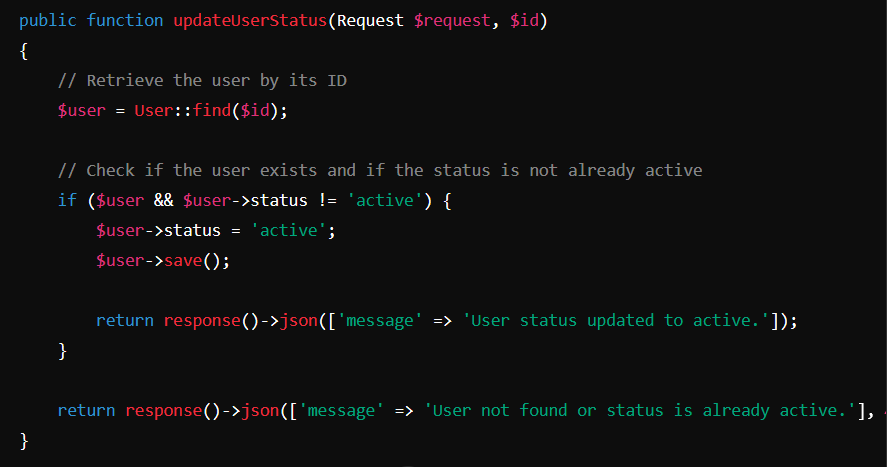
* The **updateOrCreate()** method updates an existing record if it exists, or creates a new one if it doesn’t. It's a very handy method when you want to handle both update and create logic.



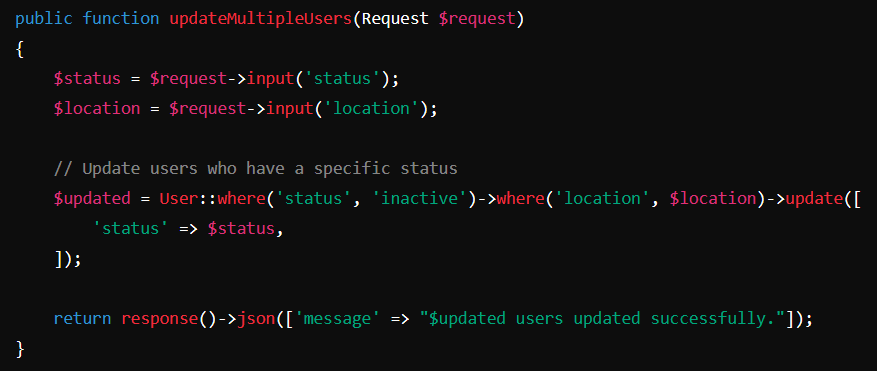
* The **fill()** method is useful for mass-assignment protection. If you have a lot of fields to update, you can pass an array of data to the **fill()** method and then save the model.



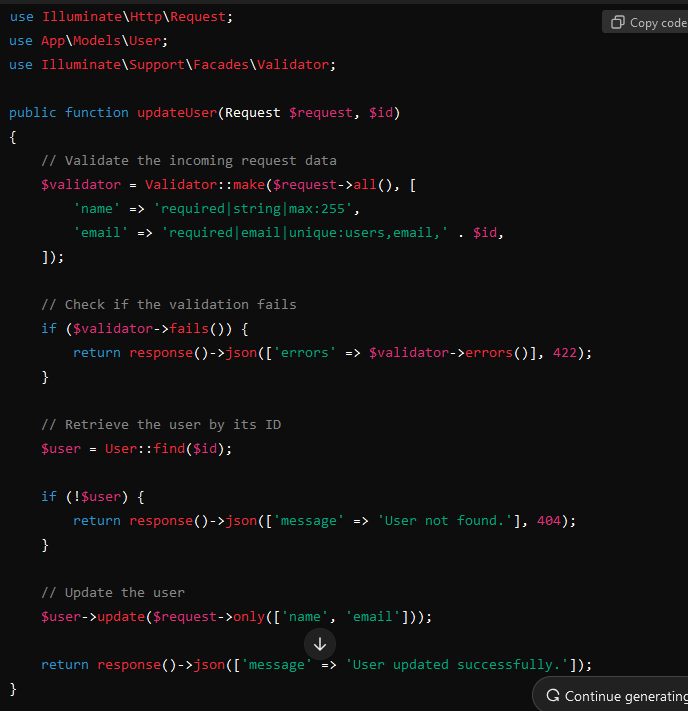
* You can update a record conditionally, such as checking for a specific value or a set of conditions before applying the update.



* You can also update multiple fields across multiple records by using a more advanced query condition.



* In many cases, you will want to validate the incoming request data before performing any update operations.



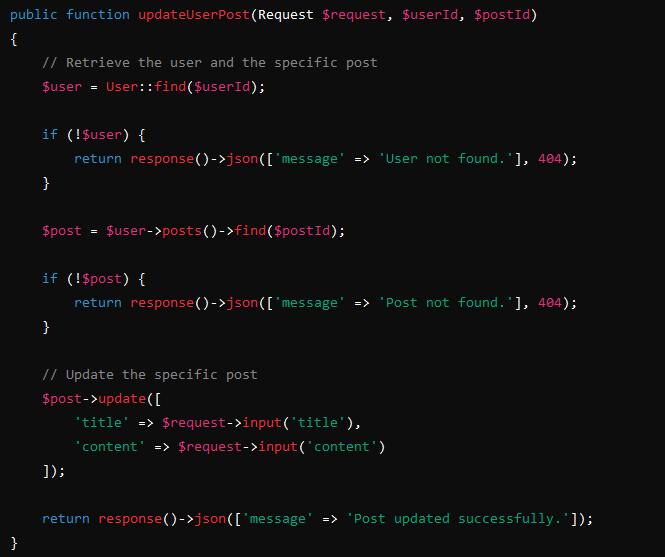
* In a one-to-one relationship, you might have a scenario where a User model has a related Profile model. Here's how to update the related data:



* In a one-to-many relationship, where a User can have many Posts, you can update related records.



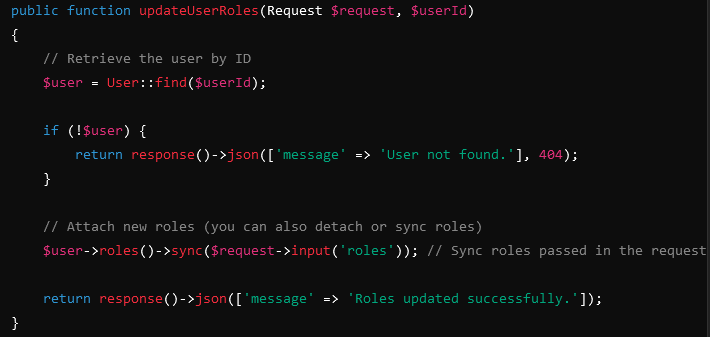
* You can also update a specific related record from a one-to-many relationship.



* In a situation where you have nested relationships, you can update a nested related model using the update method.



* In a many-to-many relationship, where a User can have many Roles, you can update the pivot table or the related records.



1. **How to host laravel project in c panel**

**In .htaccess**

**<IfModule mod\_rewrite.c>**

**<IfModule mod\_negotiation.c>**

**Options -MultiViews -Indexes**

**</IfModule>**

**RewriteEngine On**

**# This rule rewrites the URL to the correct storage path**

**RewriteRule ^Storage/(.\*)$ /storage/$1 [L,NC,R=301]**

**# Ensure the domain uses HTTPS**

**RewriteCond %{HTTPS} off**

**RewriteRule ^ https://%{HTTP\_HOST}%{REQUEST\_URI} [L,R=301]**

**# Handle Authorization Header**

**RewriteCond %{HTTP:Authorization} .**

**RewriteRule .\* - [E=HTTP\_AUTHORIZATION:%{HTTP:Authorization}]**

**# Redirect Trailing Slashes If Not A Folder...**

**RewriteCond %{REQUEST\_FILENAME} !-d**

**RewriteCond %{REQUEST\_URI} (.+)/$**

**RewriteRule ^ %1 [L,R=301]**

**# Send Requests To Front Controller...**

**RewriteCond %{REQUEST\_FILENAME} !-d**

**RewriteCond %{REQUEST\_FILENAME} !-f**

**RewriteRule ^ index.php [L]**

**</IfModule>**

**.env**

**APP\_NAME=Japanese\_Proficiency\_Test**

**APP\_ENV=production**

**APP\_KEY=base64:WMTsd0ahwR1a1nZ8y4pxLJva67gxZxBvFTBBvTeaLRY=**

**APP\_DEBUG=true**

**APP\_URL=https://jptnepal.com.np**

**LOG\_CHANNEL=stack**

**LOG\_DEPRECATIONS\_CHANNEL=null**

**LOG\_LEVEL=debug**

**DB\_CONNECTION=mysql**

**DB\_HOST=127.0.0.1**

**DB\_PORT=3306**

**DB\_DATABASE=jptn\_laravel\_db**

**DB\_USERNAME=jptn\_admin**

**DB\_PASSWORD=c3d\*n4e^mO^Ec4@f**

**BROADCAST\_DRIVER=log**

**CACHE\_DRIVER=file**

**FILESYSTEM\_DISK=local**

**QUEUE\_CONNECTION=database**

**SESSION\_DRIVER=file**

**SESSION\_LIFETIME=120**

**MEMCACHED\_HOST=127.0.0.1**

**REDIS\_HOST=127.0.0.1**

**REDIS\_PASSWORD=null**

**REDIS\_PORT=6379**

**MAIL\_MAILER=smtp**

**MAIL\_HOST=smtp.gmail.com**

**MAIL\_PORT=465**

**MAIL\_USERNAME=freelencerunittesting@gmail.com**

**MAIL\_PASSWORD=aqcdacybfhqsyvsc**

**MAIL\_ENCRYPTION=tls**

**MAIL\_FROM\_ADDRESS="freelencerunittesting@gmail.com"**

**MAIL\_FROM\_NAME="${APP\_NAME}"**

**AWS\_ACCESS\_KEY\_ID=**

**AWS\_SECRET\_ACCESS\_KEY=**

**AWS\_DEFAULT\_REGION=us-east-1**

**AWS\_BUCKET=**

**AWS\_USE\_PATH\_STYLE\_ENDPOINT=false**

**PUSHER\_APP\_ID=**

**PUSHER\_APP\_KEY=**

**PUSHER\_APP\_SECRET=**

**PUSHER\_HOST=**

**PUSHER\_PORT=443**

**PUSHER\_SCHEME=https**

**PUSHER\_APP\_CLUSTER=mt1**

**VITE\_APP\_NAME="${APP\_NAME}"**

**VITE\_PUSHER\_APP\_KEY="${PUSHER\_APP\_KEY}"**

**VITE\_PUSHER\_HOST="${PUSHER\_HOST}"**

**VITE\_PUSHER\_PORT="${PUSHER\_PORT}"**

**VITE\_PUSHER\_SCHEME="${PUSHER\_SCHEME}"**

**VITE\_PUSHER\_APP\_CLUSTER="${PUSHER\_APP\_CLUSTER}"**

**Inside app/providers/AppServiceProvider.php**

public function boot(): void

    {

        try {

            if (env('APP\_ENV') === 'production') {

                URL::forceScheme('https');

            }

        } catch (\Illuminate\Database\QueryException $e) {

        }

    }

**Inside public\_html/public/storage\_link.php**

<?php

// Define paths

$target = \_\_DIR\_\_ . '/../storage/app/public';

$link = \_\_DIR\_\_ . '/storage';

// Check if the symbolic link already exists

if (is\_link($link) || file\_exists($link)) {

// Delete the existing link

unlink($link);

echo "Existing storage link deleted.<br>";

}

// Try to create a new symlink

if (symlink($target, $link)) {

echo "New storage link created successfully!";

} else {

echo "Failed to create storage link. Check permissions.";

}

?>

**To clear caches clear\_cache.php**

<?php

echo "<p><strong>Cache cleared and optimized successfully!</strong></p>";

return;

// Run cache clearing commands

$commands = [

'cache:clear',

'config:clear',

'config:cache',

'route:clear',

'view:clear',

'optimize',

'route:clear'

];

foreach ($commands as $command) {

echo "<p>Running: php artisan $command</p>";

$output = shell\_exec("php $laravelRoot/artisan $command 2>&1");

echo "<pre>$output</pre>";

}

echo "<p><strong>Cache cleared and optimized successfully!</strong></p>";

?>

You can also use below code to clear caches:  
<?php // Manually clear Laravel cache exec('rm -rf ../bootstrap/cache/\*.php'); exec('rm -rf ../storage/framework/views/\*'); exec('rm -rf ../storage/framework/cache/\*'); exec('rm -rf ../storage/framework/sessions/\*'); exec('rm -rf ../storage/framework/routes.php'); echo "Cache cleared!"; ?>