

## 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.

**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.**

```
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'));
}
```

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.

```
Schema::create('songs', function (Blueprint $table) {  
    $table->id();  
    $table->unsignedBigInteger('user_id');  
    $table->string('title');  
    $table->timestamps();  
  
    $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade')  
});
```

```
class User extends Model  
{  
    use HasFactory;  
  
    protected $fillable = ['name', 'email', 'age', 'location', 'password'];  
  
    public function songs()  
    {  
        return $this->hasMany(Song::class);  
    }  
}
```

```
class Song extends Model  
{  
    use HasFactory;  
  
    protected $fillable = ['user_id', 'title'];  
  
    public function user()  
    {  
        return $this->belongsTo(User::class);  
    }  
}
```

```
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'));
}

```

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.

```
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'));
}
```

}

#### 4. Create a contact us form and send email to admin.

a) index.blade.php

```
{{-- =====contact us section===== --}}
<form id="contactForm">
    @csrf
    <div class="form-group">
        <label>Name:</label>
        <input type="text" class="form-control" name="name" required>
    </div>
    <div class="form-group">
        <label>Email:</label>
        <input type="email" class="form-control" name="email" required>
    </div>
    <div class="form-group">
        <label>Subject:</label>
        <input type="text" class="form-control" name="subject" required>
    </div>
    <div class="form-group">
        <label>Message:</label>
        <textarea class="form-control" name="message" required></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
<div id="responseMessage" class="mt-3"></div>
{{-- =====end of contact us section===== --}}
```

```

<script>
    $(document).ready(function() {

        // =====contact us form=====
        $('#contactForm').on('submit', function(e) {
            e.preventDefault();

            $.ajax({
                url: "{{ route('contact.submit') }}",
                method: "POST",
                data: $(this).serialize(),
                beforeSend: function() {
                    $('#button[type="submit"]').prop('disabled', true);
                },
                success: function(response) {
                    toastify().success(response.success);
                    $('#contactForm')[0].reset();
                    $('#button[type="submit"]').prop('disabled', false);
                },
                error: function(response) {
                    let errors = response.responseJSON.errors;
                    $('#button[type="submit"]').prop('disabled', false);
                }
            });
        });
        // =====end of contact us form=====
    });
</script>

```

b) In web.php

```

Route::post('/contact/submit', [HomeController::class, 'submit'])->name('contact.submit');

```

c) In HomeController.php

```

public function submit(Request $request){
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'subject' => 'required|string|max:255',
        'message' => 'required|string',
    ]);

    Contact::create($request->all());
    return response()->json(['success' => 'Message Sent Successfully!']);
}

```



d) 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.

e) Make Event:

```
php artisan make:event ContactFormSubmitted
```

f) Inside your controller:

```
public function submit(Request $request)
{
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'subject' => 'required|string|max:255',
        'message' => 'required|string',
    ]);

    $contactInfo = Contact::create($request->all());

    $data = [
        'name' => $contactInfo['name'],
        'email' => $contactInfo['email'],
        'subject' => $contactInfo['subject'],
        'message' => $contactInfo['message']
    ];

    event(new ContactFormSubmitted($data));

    return response()->json(['success' => 'Message Sent Successfully!']);
}
```

g) Inside your event ContactFormSubmitted.php

```
public $data;

/**
 * Create a new event instance.
 */
public function __construct($data)
{
    $this->data=$data;
}
```

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

```
> php artisan make:listener SendContactFormEmail --event=ContactFormSubmitted
```

```

public function handle(ContactFormSubmitted $event): void
{
    $users=User::get();
    foreach ($users as $key => $user) {
        // send mail
        Mail::to($user->email)->queue(new ContactMail($event->data));
    }
}

```

i) Inside EventServiceProvider.php

```

protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    BlogCreated::class=>[
        NotifyUser::class
    ],
    ContactFormSubmitted::class=>[
        SendContactFormEmail::class
    ]
];

```

j) Make mail to send mail:

```

php artisan make:mail ContactMail

```

```

public function __construct($data)
{
    $this->data=$data;
}

```

```

public function content(): Content
{
    return new Content(
        markdown: 'contact.contact_mail',
    );
}

```

k) Inside views/contact/contact\_mail.blade.php

```

@component('mail::message')
# Contact Form Submission

**Name:** {{ $data['name'] }}

**Email:** {{ $data['email'] }}

**Subject:** {{ $data['subject'] }}

**Message:**
{{ $data['message'] }}

Thanks,<br>
{{ config('app.name') }}
@endcomponent

```

1) Inside .env

```

MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=nontoxicyubs@gmail.com
MAIL_PASSWORD=mjltqjpqhonfynv
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=nontoxicyubs@gmail.com
MAIL_FROM_NAME="${APP_NAME}"

```

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

m) 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.

a) `php artisan queue:table`

b) in .env

```
QUEUE_CONNECTION=database
```

c) Inside ContactMail.php

```
class ContactMail extends Mailable implements ShouldQueue
```

d) `php artisan migrate`

e) `php artisan queue:listen`

## 5. 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

```
composer require pusher/pusher-php-server
```

```
npm install --save laravel-echo pusher-js
```

#### 2) Uncomment below code from config/app.php

```
App\Providers\BroadcastServiceProvider::class,
```

#### 3) In .env

```
BROADCAST_DRIVER=pusher
```

#### 4) Uncomment below code from resources/js/bootstrap.js

```
import Echo from 'laravel-echo';

import Pusher from 'pusher-js';
window.Pusher = Pusher;

window.Echo = new Echo({
  broadcaster: 'pusher',
  key: import.meta.env.VITE_PUSHER_APP_KEY,
  cluster: import.meta.env.VITE_PUSHER_APP_CLUSTER ?? 'mt1',
  wsHost: import.meta.env.VITE_PUSHER_HOST ?? `ws-${import.meta.env.VITE_PUSHER_APP_CLUSTER}.pusher.com`,
  wsPort: import.meta.env.VITE_PUSHER_PORT ?? 80,
  wssPort: import.meta.env.VITE_PUSHER_PORT ?? 443,
  forceTLS: (import.meta.env.VITE_PUSHER_SCHEME ?? 'https') === 'https',
  enabledTransports: ['ws', 'wss'],
});
```

#### 5) Get all the App Keys from <https://dashboard.pusher.com/>

```
PUSHER_APP_ID=1823897
PUSHER_APP_KEY=aa0cad2629660de0e17f
PUSHER_APP_SECRET=3ea6c7a64c232d423c42
PUSHER_APP_CLUSTER=mt1
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
```

6) Inside broadcasting.php

```
'pusher' => [
    'driver' => 'pusher',
    'key' => env('PUSHER_APP_KEY'),
    'secret' => env('PUSHER_APP_SECRET'),
    'app_id' => env('PUSHER_APP_ID'),
    'options' => [
        'cluster' => env('PUSHER_APP_CLUSTER'),
        'host' => env('PUSHER_HOST') ?: 'api-'.env('PUSHER_APP_CLUSTER', 'mt1').'.pusher.com',
        'port' => env('PUSHER_PORT', 443),
        'scheme' => env('PUSHER_SCHEME', 'https'),
        'encrypted' => true,
        'useTLS' => env('PUSHER_SCHEME', 'https') === 'https',
    ],
    'client_options' => [
        // Guzzle client options: https://docs.guzzlephp.org/en/stable/request-options.html
    ],
],
```

7) Create an event

```
php artisan make:event BlogCreated
```

8) Inside call the event from your controller:

```
public function store(BlogRequest $request)
{
    try {
        $dataToStore = $request->only('title', 'slug', 'description', 'image', 'is_published');
        $blog = $this->blogRepository->store($dataToStore);
        $data = ['title' => $blog['title'], 'author' => auth()->user()->name, 'email' => auth()->user()->email];
        event(new BlogCreated($data));
        return redirect()->route('admin.blog')->with('success', 'Blog added successfully');
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}
```

9) Inside your event:

```
class BlogCreated implements ShouldBroadcast
{
    public $blog;

    /**
     * Create a new event instance.
     */
    public function __construct($blog)
    {
        $this->blog=$blog;
    }
}
```

```

public function broadcastOn()
{
    return new Channel('my-channel');
}

public function broadcastAs() {
    return 'form-submitted';
}

```

10) Inside any blade file where you want notification to be displayed:

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

```

<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>

    // Enable pusher logging - don't include this in production
    Pusher.logToConsole = true;

    var pusher = new Pusher('aa0cad2629660de0e17f', {
        cluster: 'mt1'
    });

    var channel = pusher.subscribe('my-channel');
    channel.bind('my-event', function(data) {
        alert(JSON.stringify(data));
    });
</script>

```

Custom

```

{{-- pusher --}}
<script>
    // Enable pusher logging - don't include this in production
    Pusher.logToConsole = true;

    var pusher = new Pusher('aa0cad2629660de0e17f', {
        cluster: 'mt1'
    });

    var channel = pusher.subscribe('my-channel');
    channel.bind('form-submitted', function(data) {
        console.log(data);
        toastify().success(`${data.blog.subject} by ${data.blog.author}`);
    });
</script>

```

11)      php artisan queue:listen

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

- i. Follow upto step 6 from above setup.
- ii. Create an event : php artisan make:event:CommissionEarned and notification php artisan make:notification CommissionEarnedNotification.
- iii. Call event and notification from controller/livewire component.

**Send Notification to only auth user:**

```

// send notification
Auth::user()->notify(new CommissionEarnedNotification($commission));
Auth::user()->notifications()->latest()->first();
event(new CommissionEarned($commission));

```

- iv. Inside CommissionEarned event:

```

class CommissionEarned implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;
    public $commission;

    /**
     * Create a new event instance.
     */
    public function __construct($commission)
    {
        $this->commission=$commission;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn()
    {
        return new Channel('tickets');
    }

    public function broadcastAs(){
        return 'ticket-created';
    }
}

```

v. Inside CommissionEarnedNotification.php

```

public $commission;

/**
 * Create a new notification instance.
 */
public function __construct($commission)
{
    $this->commission=$commission;
}

```



```

/**
 * Get the notification's delivery channels.
 *
 * @return array<int, string>
 */
public function via($notifiable)
{
    return ['database', 'broadcast'];
}

```

```

public function toArray($notifiable)
{
    return [
        'name' => Auth::user()->name,
        'earned_money'=>$this->commission->commission,
        'id'=>Auth::user()->id,
    ];
}

```

```

public function toBroadcast($notifiable)
{
    return new BroadcastMessage([
        'name' => Auth::user()->name,
        'earned_money'=>$this->commission->commission,
        'id'=>Auth::user()->id
    ]);
}

```

vi. Inside master.blade.php

```

{{-- notifications --}}
<li class="nav-item dropdown dropdown-large">
    @livewire('admin.notification.bell-notification')
</li>
{{-- end of notifications --}}

```

vii. Create BellNotification.php livewire component.

viii. Inside BellNotification.php

```

class BellNotification extends Component
{
    public $notifications;
    public $unreadCount;
    private $notificationRepository;

    protected $listeners = ['notificationReceived' => 'fetchNotifications'];

    public function boot(NotificationRepositoryInterface $notificationRepository)
    {
        $this->notificationRepository = $notificationRepository;
    }

    public function mount()
    {
        $this->fetchNotifications();
    }

    public function fetchNotifications()
    {
        $notificationInfo = $this->notificationRepository->fetchNotifications(10);
        $this->notifications = $notificationInfo['notifications'];
        $this->unreadCount = $notificationInfo['unreadCount'];
    }

    public function markAsRead($notificationId)
    {
        $notification = $this->notificationRepository->markAsRead($notificationId);
        if ($notification) {
            $notification->markAsRead();
            $this->fetchNotifications();
        }
    }

    public function render()
    {
        return view('livewire.admin.notification.bell-notification');
    }
}

```

ix. Inside NotificationRepository.php:

```

class NotificationRepository implements NotificationRepositoryInterface
{
    public function fetchNotifications($totalNotification)
    {
        $user = Auth::user();
        if($totalNotification=='all'){
            $notifications = $user->notifications()->latest()->get();
        }else{
            $notifications = $user->notifications()->latest()->take($totalNotification)->get();
        }
        $unreadCount = $user->unreadNotifications->count();
        return [
            'notifications'=>$notifications,
            'unreadCount'=>$unreadCount
        ];
    }

    public function markAsRead($notificationId){
        return Auth::user()->notifications()->find($notificationId);
    }
}

```

x. Inside bell-notification.blade.php

```

<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>
    document.addEventListener('DOMContentLoaded', function() {
        Pusher.logToConsole = true;

        var pusher = new Pusher('aa0cad2629660de0e17f', {
            cluster: 'mt1',
            encrypted: false,
            forceTLS: false
        });

        var channel = pusher.subscribe('tickets');
        channel.bind('ticket-created', function(data) {
            console.log('i am here');
            @this.call('fetchNotifications');
            console.log(data.ticket.id);
        });

        function handleNotificationClick(notificationId, url) {
            @this.call('markAsRead', notificationId).then(() => {
                window.location.href = url;
            });
        }

        function handleNotification(notificationId){
            @this.call('markAsRead', notificationId)
        }
    });
</script>

```

```

<div class="header-notifications-list">
  @foreach ($notifications as $notification)
    @if ($notification->type == 'App\Notifications\TicketCreatedNotification')
      @php
        $created_at = \Illuminate\Support\Carbon::parse($notification->created_at);
        $time_ago = $created_at->diffForHumans();
      @endphp
      <a class="dropdown-item {{ $notification->read_at ? '' : 'unread' }}" href="#"
        onclick="handleNotificationClick('{{ $notification->id }}', '{{ route('admin.support.chat', $notification->data['id']) }}')">
        <div class="d-flex align-items-center">
          <div class="user-online">
            
          </div>
          <div class="flex-grow-1">
            <h6 class="msg-name">
              {{ Auth::user()->hasRole('super_admin') ? 'Ticket number: ' . $notification->data['ticket_number'] . ' (opened)' : 'Ticket number: ' . $notification->data['ticket_number'] }}
              <span class="msg-time float-end">{{ $time_ago }}</span>
            </h6>
            <p class="msg-info">
              {{ Auth::user()->hasRole('super_admin') ? $notification->data['user'] : 'Admin responds to your ticket.' }}
            </p>
          </div>
        </div>
      </a>
    @endif
  @endforeach
</div>

```

## 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:

```
public function store(PostRequest $request)
{
    try {
        $post = $this->postService->addService($request->validated());
        // Notify all users about the new post (except the one creating the post)
        $users = User::where('id', '!=', Auth::user()->id)->get();
        .....foreach ($users as $user) {
        .....$user->notify(new PostCreatedNotification($post));
        .....}
        .....event(new BlogCreated($post));
        return redirect()->route('posts.index')->with('success', 'Post added successfully!');
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}
```

- 4) Inside BlogCreated event:

```
class BlogCreated implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $blog;

    /**
     * Create a new event instance.
     */
    public function __construct($blog)
    {
        $this->blog=$blog;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn()
    {
        return new Channel('my-channel');
    }

    public function broadcastAs(){
        return 'form-submitted';
    }
}
```

- 5) Inside PostCreatedNotification.php:

```

class PostCreatedNotification extends Notification
{
    use Queueable;

    public $post;

    public function __construct(Post $post)
    {
        $this->post = $post;
    }

    public function via($notifiable)
    {
        return ['database', 'broadcast'];
    }

    public function toArray($notifiable)
    {
        return [
            'id' => $this->id,
            'post_id' => $this->post->id,
            'title' => $this->post->title,
            'user' => $this->post->user->name,
        ];
    }
}

```

#### 6) Inside master.blade.php

```

{{-- notification --}}
<li class="nav-item notification-item pe-3 position-relative" data-bs-toggle="modal"
    data-bs-target="#staticBackdrop">
    <i class="fa-solid fa-bell fs-4 text-dark"></i>
    <span class="notification-badge bg-danger text-white rounded-circle" id="notification-badge">
        {{ auth()->user()->unreadNotifications->count() }}
    </span>
</li>
{{-- end notification --}}

```

```

<div class="modal-body notification-list">
  @foreach (auth()->user()->notifications as $notification)
    <form action="{{ route('admin.notification.markAsRead', $notification->id) }}"
      method="POST">
      @csrf
      <button class="notification-item d-flex align-items-start mb-3 w-100 border-0"
        style="background-color: {{ $notification->read_at ? '#ffffff' : '#daf1f9' }};">
        
        <div class="notification-content">
          <h6 class="mb-1">{{ $notification->data['title'] }}</h6>
          <p class="mb-0 text-muted">New post by {{ $notification->data['user'] }}</p>
        </div>
        <span
          class="text-muted small ms-auto">{{ $notification->created_at->diffForHumans() }}</span>
      </button>
    </form>
  @endforeach
</div>
<div class="modal-footer">
  <button type="button" class="btn btn-primary w-100">View All Notifications</button>
</div>

```

```

{{-- =====notification===== --}}
{{-- for human readable time --}}
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.4/moment.min.js"></script>
{{-- pusher link --}}
<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>
  // Enable pusher logging
  Pusher.logToConsole = true;

  var pusher = new Pusher('38806878aa78497c43a3', {
    cluster: 'mt1'
  });

  var creatorId = "{{ auth()->id() }}";
  var channel = pusher.subscribe('my-channel');
  channel.bind('form-submitted', function(data) {
    // Fetch the latest notification and append it
    fetchLatestNotification();
    // Update the notification count
    updateNotificationCount();
  });

```

```

function fetchLatestNotification() {
    // Use AJAX to fetch the latest notification
    fetch('/latest-notification')
        .then(response => response.json())
        .then(notification => {
            console.log(notification);
            var notificationHtml = `
<form action="/admin/notifications/${notification.id}/mark-as-read" method="POST">
    @csrf
    <button class="notification-item d-flex align-items-start mb-3 w-100 border-0" style="background-color: ${notification.read_at ? '#ffffff' : '#d9f9f9'};">
        
        <div class="notification-content">
            <h6 class="mb-1">${notification.data.title}</h6>
            <p class="mb-0 text-muted">New post by ${notification.data.user}</p>
        </div>
        <span class="text-muted small ms-auto">${moment(notification.created_at).fromNow()}</span>
    </button>
</form>
            `;

            // Append the new notification to the notification list
            document.querySelector('.notification-list').insertAdjacentHTML('afterbegin', notificationHtml);
        })
        .catch(error => console.error('Error fetching notification:', error));
}

```

```

function updateNotificationCount() {
    // Use AJAX to get the unread notification count
    fetch('/unread-notification-count')
        .then(response => response.json())
        .then(count => {
            // Update the badge with the new count
            console.log(count);
            document.getElementById('notification-badge').innerText = count;
        })
        .catch(error => console.error('Error fetching unread count:', error));
}

```

## 7) Inside web.php

```

Route::get('/latest-notification', function () {
    return auth()->user()->notifications()->latest()->first();
});

Route::get('/unread-notification-count', function () {
    return auth()->user()->unreadNotifications->count();
});

```

## 8) Inside NotificationController.php

```

public function markAsRead($id)
{
    // mark notification as read
    $notification = auth()->user()->notifications()->findOrFail($id);
    $notification->markAsRead();

    return redirect()->route('admin.notification');
}

```



## 7. List all the users with his status i.e. online or offline.

a) In Users table:

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->string('profile_picture')->nullable();
    $table->timestamp('last_login_at')->nullable();
    $table->timestamp('last_seen_at')->nullable();
    $table->rememberToken();
    $table->timestamps();
});
```

b) Inside EventServiceProvider.php:

```
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    'Illuminate\Auth\Events\Login' => [
        'App\Listeners\LogSuccessfulLogin',
    ],
    'Illuminate\Auth\Events\Logout' => [
        'App\Listeners\LogSuccessfulLogout',
    ],
];
```

c) Create a Listener to check login user:

```
php artisan make:listener LogSuccessfulLogin
```

d) Inside app/Listeners/LogSuccessfulLogin.php:

```
namespace App\Listeners;

use Illuminate\Auth\Events\Login;
use Illuminate\Support\Facades\Auth;

class LogSuccessfulLogin
{
    public function handle(Login $event)
    {
        $user = Auth::user();
        $user->last_login_at = now();
        $user->save();
    }
}
```

e) Create listener to check logout user:

```
php artisan make:listener LogSuccessfulLogout
```

f) Inside app/Listeners/LogSuccessfulLogout.php:


```
namespace App\Listeners;

use Illuminate\Auth\Events\Logout;
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\Auth;

class LogSuccessfulLogout
{
    public function handle(Logout $event)
    {
        $user = $event->user;

        if ($user) {
            // Remove the user's online status from the cache
            Cache::forget('user-is-online-' . $user->id);

            // Update the last seen time
            $user->last_seen_at = now();
            $user->save();
        }
    }
}
```



g) Inside app/User.php model:

```
protected $casts = [
    'last_login_at' => 'datetime',
];

public function isOnline()
{
    return Cache::has('user-is-online-' . $this->id);
}

public function setOnlineStatus()
{
    Cache::put('user-is-online-' . $this->id, true, Carbon::now()->addMinutes(5));
}
```

```
protected $casts = [
    'last_login_at' => 'datetime',
    'last_seen_at' => 'datetime',
];
```

h) Create a middleware:

```
php artisan make:middleware SetOnlineStatus
```

i) Inside app/Http/Middleware/SetOnlineStatus.php:

```

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class SetOnlineStatus
{
    public function handle($request, Closure $next)
    {
        if (Auth::check()) {
            $user = Auth::user();
            $user->setOnlineStatus();
        }

        return $next($request);
    }
}

```

j) Inside app/Http/Kernel.php:

```

protected $middlewareGroups = [
    'web' => [
        // other middleware
        \App\Http\Middleware\SetOnlineStatus::class,
    ],
];

```

k) Inside your controller:

```

public function messenger()
{
    $users = User::where('id', '!=', Auth::id())->get();
    return view('messenger', compact('users'));
}

```

l) Inside your blade file:

```
<td>
    @if($user->isOnline())
        <span class="text-success">Online</span>
    @else
        <span class="text-danger">Offline</span>
        <br>
        <small>Last seen: {{ $user->last_seen_at ? $user->last_seen_at->diffForHumans() :
    @endif
</td>
```

## 8. Create a product table and apply all the possible filters from frontend for user using AJAX.

### a) Filter with Searching Keyword:

```
{{-- =====search field===== --}}  
<input class="form-control" type="search" name="" id="search"  
|   placeholder="Search products..">  
{{-- =====end of search field===== --}}
```

```
// =====Search products when typing in the search field(input text tag)=====  
let timer;  
$('#search').on('keyup', () => {  
|   clearTimeout(timer);  
|   timer = setTimeout(searchProducts, 1000);  
});  
// =====end of search products when typing in the search field=====
```

### b) Filter with Select Field:

```
<div class="btn-group">  
|   <button type="button" class="btn btn-sm btn-light dropdown-toggle"  
|       data-bs-toggle="dropdown">Sorting</button>  
|   <div class="dropdown-menu dropdown-menu-right sorting-products-filters">  
|       <a class="dropdown-item" href="javascript:void(0)">Latest</a>  
|       <a class="dropdown-item" href="javascript:void(0)">Price High</a>  
|       <a class="dropdown-item" href="javascript:void(0)">Price Low</a>  
|   </div>  
</div>
```

```
// =====Filter products when clicking on sorting options(select tag)=====  
$('.sorting-products-filters a').on('click', function() {  
|   const sortBy = $(this).text().trim();  
|   searchProducts(sortBy);  
});  
// =====end of filter products when clicking on sorting options=====
```

### c) Filter with Checkbox:

```

@if (count($requiredShopData['brands']) > 0)
    @foreach ($requiredShopData['brands'] as $key => $brand)
        <div class="form-check mb-2">
            <input class="form-check-input brand-check-input" type="checkbox"
                value="{{ $brand->id }}" id="{{ $brand->name }}-{{ $key }}">
            <label class="form-check-label" for="{{ $brand->name }}-{{ $key }}">
                {{ $brand->name }}
            </label>
        </div>
    @endforeach
@endif

```

```

//=====Filter with brand names(checkbox)=====
const getSelectedBrands = () => {
    let selectedBrands = [];
    $('brand-check-input:checked').each(function() {
        selectedBrands.push($(this).val());
    });
    console.log(selectedBrands);
    return selectedBrands;
};
$('brand-check-input').on('change', function() {
    searchProducts();
});
// =====end of filter with brand names=====

```

#### d) Filter with radio button:

```

@if (count($requiredShopData['categories']) > 0)
    @foreach ($requiredShopData['categories'] as $key => $category)
        <div class="form-check mb-2">
            <input class="form-check-input category-radio-input" type="radio"
                name="category" value="{{ $category->id }}"
                id="{{ $category->name }}-{{ $key }}">
            <label class="form-check-label" for="{{ $category->name }}-{{ $key }}">
                {{ $category->name }}
            </label>
        </div>
    @endforeach
@endif

```



```
// =====Filter with category(radio button)=====
const getSelectedCategory = () => {
    return $('input[name="category"]:checked').val(); // Get the value of the selected radio button
};
$('.category-radio-input').on('change', function() {
    searchProducts();
});
// =====end of filter with category(radian button)=====
```

### e) 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>
```

```
<div id="price-slider" style="margin: 20px;"></div>
<p>Price: <span id="min-price"></span> - <span id="max-price"></span></p>
```

```
// =====Filter with price slider(range tag)=====
const priceSlider = document.getElementById('price-slider');
noUiSlider.create(priceSlider, {
  start: [0, 1000],
  connect: true,
  range: {
    'min': 0,
    'max': 1000
  },
  step: 10,
  tooltips: [true, true],
  format: {
    to: function(value) {
      return parseInt(value);
    },
    from: function(value) {
      return parseInt(value);
    }
  }
});

priceSlider.noUiSlider.on('update', function(values) {
  $('#min-price').text(values[0]);
  $('#max-price').text(values[1]);
});

priceSlider.noUiSlider.on('change', function() {
  searchProducts();
});
// =====end of filter with price slider range=====
```

f) Ajax Call for filter and sorting and display products:

```
<div class="row" id="product-card">
  <!-- Products will be displayed here dynamically -->
</div>
```

```

1 //===== AJAX call to filter and sort products=====
2 const searchProducts = (sortBy = 'latest', pageUrl = null) => {
3   const keyword = $('#search').val(); // Get searching keyword
4   const brands = getSelectedBrands(); // Get the selected brands
5   const category = getSelectedCategory(); // Get the selected category
6   const minPrice = priceSlider.noUiSlider.get()[0]; // Get min price from slider
7   const maxPrice = priceSlider.noUiSlider.get()[1]; // Get max price from slider
8
9   // Determine the URL for pagination or for sorting/filtering
10  const url = pageUrl ? pageUrl : "{{ route('frontend.products') }}";
11
12  // Make AJAX GET request with sorting, brand filters, price range, and category
13  $.get(url, {
14    'search': keyword,
15    'sort_by': sortBy,
16    'brands': brands,
17    'min_price': minPrice,
18    'max_price': maxPrice,
19    'category_id': category
20  })
21  .done(response => {
22    const products = response.products;
23    $('#product-card').empty();
24
25    if (products.length > 0) {
26      products.forEach(product => {
27        let comparePrice = '';
28        if (product.compare_price) {
29          comparePrice =
30            '<span class="h6 text-underline"><del>${product.compare_price}</del></span>';
31        }
32
33        const productCard = `
34          <div class="col-4">
35            <div class="card product-card">
36              <div class="product-image position-relative">
37                <a href="" class="product-img">
38                  
40                </a>
41                <a class="whishlist" href="222"><i class="far fa-heart"></i></a>
42
43                <div class="product-action">
44                  <a class="btn btn-dark" href="#">
45                    <i class="fa fa-shopping-cart"></i> Add To Cart
46                  </a>
47                </div>
48              </div>
49              <div class="card-body text-center mt-3">
50                <a class="h6 link" href="product.php">${product.title}</a>
51                <div class="price mt-2">
52                  <span class="h5"><strong>${product.price}</strong></span>
53                  <span class="h5"><strong>${product.compare_price}</strong></span>
54                </div>
55              </div>
56            </div>
57          </div>
58          `;
59
60          $('#product-card').append(productCard);
61        });
62      } else {
63        $('#product-card').append('<p>No products found</p>');
64      }
65      // Update pagination links
66      $('#pagination-links').html(response.pagination);
67
68      // Bind click event for pagination links
69      $('#pagination-links a').on('click', function(e) {
70        e.preventDefault();
71        const pageUrl = $(this).attr('href');
72        searchProducts(sortBy, pageUrl);
73      });
74    });
75  };
76  // =====end of AJAX call to filter and sort products=====
77
78  //===== Initial product load=====
79  searchProducts();
80  // =====end of Initial product load=====

```

## g) Inside Controller:

```
1 public function products(Request $request)
2 {
3     $query = Product::with('brand', 'category', 'subCategory');
4
5     // =====Apply search filter=====
6     if ($request->search) {
7         $query->where('title', 'LIKE', '%' . $request->search . '%')
8         ->orWhere('price', 'LIKE', '%' . $request->search . '%');
9     }
10
11     // =====Apply brand filter from checkbox=====
12     if ($request->brands) {
13         $query->whereIn('brand_id', $request->brands);
14     }
15
16     // =====Apply category filter from radio button=====
17     if ($request->category_id) {
18         $query->where('category_id', $request->category_id);
19     }
20
21     // =====Apply price range filter from slider range=====
22     if ((int)$request->min_price >= 0 && $request->max_price) {
23         $query->whereBetween('price', [(int)$request->min_price, (int)$request->max_price]);
24     }
25
26     // =====Apply sorting from select field=====
27     switch ($request->sort_by) {
28         case 'Price High':
29             $query->orderBy('price', 'desc');
30             break;
31         case 'Price Low':
32             $query->orderBy('price', 'asc');
33             break;
34         case 'latest':
35         default:
36             $query->orderBy('created_at', 'desc');
37             break;
38     }
39
40     // =====Paginate the results=====
41     $products = $query->paginate(4);
42
43     // =====return products with pagination=====
44     return response()->json([
45         'products' => $products->items(),
46         'pagination' => (string) $products->links('pagination::bootstrap-5')
47     ]);
48 }
```

In web.php

```
// ====Auth=====
Auth::routes([
    'register'=>false,
]);
// ====End of Auth=====

// =====Frontend=====
require __DIR__.'/public.php';
// =====End of Frontend=====

// =====Backend=====
Route::middleware(['auth.admin'])->group(function(){
    Route::prefix('admin')->group(function(){
        require __DIR__.'/admin.php';
    });
});
// =====end of Backend=====

// =====handle wrong url=====
Route::redirect('/{any}', '/', 301);
//=====end of handling wrong url=====
```

### Inside public.php

```
Route::get('/products',[ShopController::class,'products'])->name('frontend.products');
```

### Inside admin.php

```
use Illuminate\Support\Facades\Route;

Route::get('/home', [HomeController::class, 'index'])->name('home');

Route::resources([
    'category'=>CategoryController::class,
    'sub_category'=>SubCategoryController::class,
    'brands'=>BrandController::class,
    'products'=>ProductController::class,
]);
```

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

### a) Install laravel ui package

```
composer require laravel/ui
php artisan ui bootstrap --auth
npm install && npm run dev
```

### b) Change UI of login, register and password reset page as your requirements.

The image shows a login form with a blue header containing the word "Login". Below the header, there are two input fields: "Email Address" and "Password". Below the "Password" field is a checkbox labeled "Remember Me". Below the checkbox is a blue button labeled "Login". Below the button are two links: "Forgot Your Password?" and "Don't have an account? [Register Here](#)".

## Register

Name

Email Address

Password

Confirm Password

Register

Already have an account? [Login Here](#)

## Reset Password

Email Address

Send Password Reset Link

[Back to Login](#)

**c) Make middleware for authorization:**

Php artisan make:middleware AdminMiddleware

## Inside AdminMiddleware

```
/**
 * Handle an incoming request.
 *
 * @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next
 */
public function handle(Request $request, Closure $next): Response
{
    if(!auth()->check()){
        return redirect()->route('login')->with('error','Please login to access this page');
    }
    return $next($request);
}
```

## Inside app\Http\Kernel.php

```
protected $middlewareAliases = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
];
```

## Inside Web.php

```
// =====Backend=====
Route::middleware(['auth.admin'])->group(function(){
    Route::prefix('admin')->group(function(){
        require __DIR__.'/admin.php';
    });
});
// =====end of Backend=====
```

## Inside admin.php

```
<?php

use App\Http\Controllers\admin\HomeController;
use Illuminate\Support\Facades\Route;

Route::get('/home', [HomeController::class, 'index'])->name('admin.home');
```

## Make Admin/HomeController.php



```
class HomeController extends Controller
{
    public function index()
    {
        return view('admin.home.index');
    }
}
```

Inside admin/home/index.blade.php make admin dashboard home page  
**Search '/home' and change it to '/admin/home'**

**d) Reset password through email.**

Make mail to send password reset link:

Generate a mailable to handle the custom email:

```
bash
```

```
php artisan make:mail ResetPasswordMail
```

Inside ResetPasswordMail

```

1  <?php
2
3  namespace App\Mail;
4
5  use Illuminate\Bus\Queueable;
6  use Illuminate\Mail\Mailable;
7  use Illuminate\Queue\SerializesModels;
8
9  class ResetPasswordMail extends Mailable
10 {
11     use Queueable, SerializesModels;
12
13     public $token;
14     public $email;
15
16     /**
17      * Create a new message instance.
18      *
19      * @return void
20      */
21     public function __construct($token, $email)
22     {
23         $this->token = $token;
24         $this->email = $email;
25     }
26
27     /**
28      * Build the message.
29      *
30      * @return $this
31      */
32     public function build()
33     {
34         return $this->view('emails.reset_password')
35             ->with([
36                 'token' => $this->token,
37                 'email' => $this->email,
38             ]);
39     }
40 }
41

```

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

```

<body>
  <div class="email-container">
    <div class="email-header">
      {{ __('Password Reset Request') }}
    </div>
    <div class="email-body">
      <p>{{ __('Hello,') }}</p>
      <p>{{ __('You are receiving this email because we received a password reset request for your account.') }}</p>
      <p style="text-align: center;">
        <a href="{{ url('password/reset', $token) }}"?email={{ urlencode($email) }}"
          ..... class="reset-button">{{ __('Reset Password') }}</a>
      </p>
      <p>{{ __('If you did not request a password reset, no further action is required.') }}</p>
      <p>{{ __('Thank you,') }}<br>{{ __('The Team') }}</p>
    </div>
  </div>
</body>
</html>

```

Inside user modal:

```

public function sendPasswordResetNotification($token)
{
    Mail::to($this->email)->send(new ResetPasswordMail($token,$this->email));
}

```

Inside .env

```

MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=yubrajkoirala7278@gmail.com
MAIL_PASSWORD=bzclhlgvxxxaepxv
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="yubrajkoirala7278@gmail.com"
MAIL_FROM_NAME="{{ APP_NAME }}"

```

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

## Reset Password

Email Address

yubraj.sixsigma@gmail.com

Password

Confirm Password

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

```
Route::get('/password/expired', function () {
    return view('auth.passwords.expired');
})->name('password.expired');
```

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>

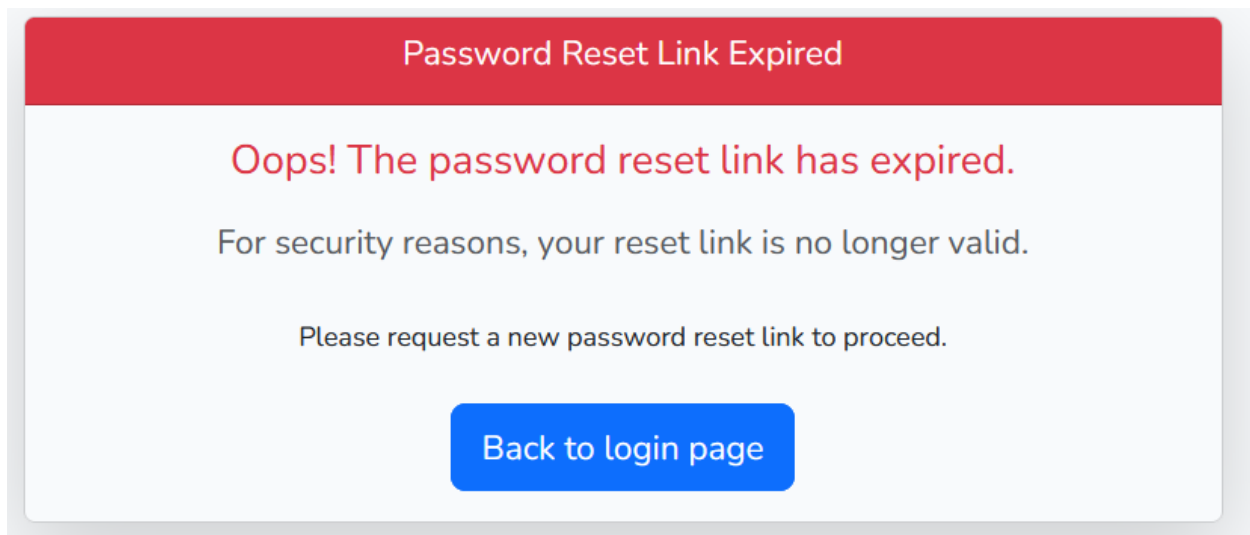
                <div class="card-body text-center">
                    <h4 class="text-danger mb-3">{{ __('Oops! The password reset
link has expired.') }}</h4>
                    <p class="lead text-muted mb-4">{{ __('For security reasons,
your reset link is no longer valid.') }}</p>
                    <p class="mb-4">{{ __('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>
@endsection

```



#### e) Verify Email Before Login through email:

Inside web.php

```
Auth::routes(['verify'=>true]);
```

```

// =====Backend=====
Route::group(['middleware' => ['auth', 'verified']], function() {
    Route::prefix('admin')->group(function(){
        require __DIR__.'/admin.php';
    });
});
// =====end of Backend=====

```

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 artisan make:notification VerifyEmailNotification
```

```

<?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:

```

class User extends Authenticatable implements MustVerifyEmail
{

```

```

    use HasApiTokens, HasFactory, Notifiable;

```

```

/**
 * 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;">
                {{--  --}}
              </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"

```

```

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

```

MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=yubrajkoirala7278@gmail.com
MAIL_PASSWORD=bzclhlgvxxaepxv
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="yubrajkoirala7278@gmail.com"
MAIL_FROM_NAME="${APP_NAME}"

```

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

Verify Your Email Address

Before proceeding, please check your email for a verification link.

If you did not receive the email, you can request another one below.

Request Another Verification Link

**f) 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

- g) 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>

```
composer require spatie/laravel-permission
```

in your config/app.php file:

```
'providers' => [  
    // ...  
    Spatie\Permission\PermissionServiceProvider::class,  
];
```

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

```
php artisan optimize:clear
```

```
# or
```

```
php artisan config:clear
```

```
php artisan migrate
```

### Add the necessary trait to your User model:

```
// The User model requires this trait  
use HasRoles;
```

Php artisan make:seeder RoleSeeder

```

public function run(): void
{
    $roles=['admin','articles_manager','news_manager'];
    foreach ($roles as $key => $role) {
        Role::create([
            'name'=>$role
        ]);
    }
}

```

Php artisan make:seeder AdminSeeder

```

public function run(): void
{
    $user=User::create([
        'name'=>'Yubraj Koirala',
        'email'=>'yubrajkoirala7278@gmail.com',
        'password'=>Hash::make('12345')
    ]);
    $user->assignRole('admin');
}

```

Inside DatabaseSeeder.php

```

public function run(): void
{
    $this->call([
        RoleSeeder::class,
        AdminSeeder::class
    ]);
}

```

Php artisan make:middleware AdminMiddleware

```

public function handle(Request $request, Closure $next): Response
{
    if(Auth::check()){
        $user=Auth::user();
        if($user->hasRole(['admin','articles_manager','news_manager'])){
            return $next($request);
        }
        Auth::logout();
        return redirect()->route('login')->with('error','You dont have permission to access this page');
    }
    return redirect()->route('login')->with('error','You dont have permission to access this page');
}

```

Inside Http/kernel.php

```
protected $middlewareAliases = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

    'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
    'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middleware\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middleware\RoleOrPermissionMiddleware::class,
];
```

Web.php

```
// =====Backend=====
Route::group(['middleware' => ['auth.admin', 'verified']], function() {
    Route::prefix('admin')->group(function(){
        require __DIR__.'/admin.php';
    });
});
// =====end of Backend=====
```

Admin.php

```
Route::group(['middleware' => ['role:admin|articles_manager']], function () {
    Route::get('/articles', [ArticlesController::class, 'index'])->name('admin.articles');
});
```

For user create

```
public function store(Request $request)
{
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
    ]);
    $user->assignRole($request->role);
    return back()->with('success', 'User has been created');
}
```

Get role names and auth name in blade

```
<h5>Welcome to the Dashboard</h5>
{{Auth::user()->name}}
{{Auth::user()->getRoleNames()}}
```

h) 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>

```
composer require spatie/laravel-permission
```

in your config/app.php file:

```
'providers' => [  
    // ...  
    Spatie\Permission\PermissionServiceProvider::class,  
];
```

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

```
php artisan optimize:clear
```

# or

```
php artisan config:clear
```

```
php artisan migrate
```

**Add the necessary trait to your User model:**

```
// The User model requires this trait  
use HasRoles;
```

Create a PermissionSeeder:

```
public function run(): void  
{  
    $permissions=['create_articles','create_user','create_news'];  
    foreach ($permissions as $key => $permission) {  
        Permission::create([  
            'name'=>$permission,  
        ]);  
    }  
}
```



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

```
public function run(): void
{
    $superAdminRole=Role::create([
        'name'=>'super_admin'
    ]);

    // Retrieve all permissions
    $permissions = Permission::pluck('name')->toArray();

    // Assign all permissions to the super_admin role
    if ($superAdminRole) {
        $superAdminRole->syncPermissions($permissions);
    }
}
```

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

```
public function run(): void
{
    $user=User::create([
        'name'=>'Yubraj Koirala',
        'email'=>'yubrajkoirala7278@gmail.com',
        'password'=>Hash::make('12345')
    ]);
    $user->assignRole('super_admin');
}
```

Inside DatabaseSeeder

```
public function run(): void
{
    $this->call([
        PermissionSeeder::class,
        RoleSeeder::class,
        AdminSeeder::class,
    ]);
}
```

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

## Roles table

[Add New Role](#)

#	Role	Permissions	Action
1	Head Manager	<a href="#">create_user</a>	<a href="#">Edit/Add Permission</a>
2	Manager	<a href="#">create_articles</a> <a href="#">create_news</a>	<a href="#">Edit/Add Permission</a>
3	super_admin	<a href="#">create_articles</a> <a href="#">create_user</a> <a href="#">create_news</a>	<a href="#">Edit/Add Permission</a>

```
public function role()
{
    $roles = Role::with('permissions')->latest()->get();
    return view('admin.user.role', compact('roles'));
}
```

```
<tbody>
    @if (count($roles) > 0)
        @foreach ($roles as $key => $role)
            <tr>
                <th scope="row">{{ $key + 1 }}</th>
                <td>{{ $role->name }}</td>
                <td>
                    <a href="{{ route('roles.permissions', $role->id) }}" class="btn btn-primary">Edit/Add
                    ..... Permission</a>
                </td>
            </tr>
        @endforeach
    @endif
</tbody>
```

```
Route::get('/roles/{id}/permissions', [RoleController::class, 'addPermissionToRole']->name('roles.permissions'));
```

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

# Role: super\_admin

## Add Permission to role super\_admin

### Permissions

☒ create\_articles    ☒ create\_user    ☐ create\_news

[Submit](#)

```

public function addPermissionToRole($id)
{
    $permissions = Permission::latest()->get();
    $role = Role::findOrFail($id);

    $rolePermissions = DB::table('role_has_permissions')
        ->where('role_has_permissions.role_id', $role->id)
        ->pluck('role_has_permissions.permission_id', 'role_has_permissions.permission_id')->all();

    return view('admin.user.permission', compact('permissions', 'role', 'rolePermissions'));
}

```

```

<form action="{{route('give.permission.to.role',$role->id)}}" method="POST">
    @csrf
    @method('PUT')
    @if (count($permissions) > 0)
        <div class="d-flex align-items-center" style="flex-wrap: wrap;gap:30px">
            @foreach ($permissions as $permission)
                <div class="form-check">
                    <input type="checkbox" class="form-check-input" id="{{ $permission->name }}" name="permission[]"
                        value="{{ $permission->name }}" @checked(in_array($permission->id, $rolePermissions))>
                    <label class="form-check-label" for="{{ $permission->name }}">
                        {{ $permission->name }}
                    </label>
                </div>
            @endforeach
        </div>
    @endif
    <button type="submit" class="btn btn-success">Submit</button>
</form>

```

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

```

Route::put('/roles/{id}/permission',[RoleController::class,'givePermissionToRole']->name('give.permission.to.role'));

```

```

public function givePermissionToRole(Request $request,$roleId){
    try{
        $role=Role::findOrFail($roleId);
        $role->syncPermissions($request->permission);
        return redirect()->route('role')->with('success','Permission has been added');
    }catch(\Throwable $th){
        return back()->with('error',$th->getMessage());
    }
}

```

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:

Role

☐ create\_articles    ☐ create\_user    ☐ create\_news

Submit

```
public function create()
{
    $permissions = Permission::latest()->get();
    return view('admin.user.create_role', compact('permissions'));
}
```

```
<form action="{{route('role.store')}}" method="POST">
    @csrf
    <div class="mb-3">
        <label for="name" class="form-label">Role</label>
        <input type="text" class="form-control" id="name" name="name">
    </div>
    @if (count($permissions) > 0)
    <div class="d-flex align-items-center" style="flex-wrap: wrap; gap: 30px">
        @foreach ($permissions as $permission)
            <div class="form-check">
                <input type="checkbox" class="form-check-input" id="{{ $permission->name }}" name="permission[]"
                value="{{ $permission->name }}">
                <label class="form-check-label" for="{{ $permission->name }}">
                    {{ $permission->name }}
                </label>
            </div>
        @endforeach
    </div>
    @endif
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Now When clicked on submit button:

```
public function store(Request $request)
{
    $role = Role::create([
        'name' => $request->name
    ]);
    $role->syncPermissions($request->permission);
    return redirect()->route('role')->with('success', 'Permission has been added');
}
```

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:

#	Name	Email	Roles
1	Yubraj Koirala	yubrajkoirala7278@gmail.com	["super_admin"]
2	Ralph Richmond	tesymidaqy@mailinator.com	["super_admin"]

```

@if (count($users) > 0)
    @foreach ($users as $key => $user)
        <tr>
            <td>{{ $key + 1 }}</td>
            <td>{{ $user->name }}</td>
            <td>{{ $user->email }}</td>
            <td>{{ $user->getRoleNames() }}</td>
        </tr>
    @endforeach
@endif

```

Now, Inside your controller

```

public function __construct()
{
    // permission with create_articles only can access index and store method
    $this->middleware('permission:create_articles', ['index', 'store']);
    // permission with create_news only can access delete,edit and update method
    $this->middleware('permission:create_news', ['delete', 'edit','update']);
}

```

Or

```

Route::get('/articles', [ArticlesController::class, 'index'])->name('admin.articles')->middleware('permission:create_articles');

```

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

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.

a) Export to pdf:

```
composer require barryvdh/laravel-dompdf
```

Inside your controller:

```
use Barryvdh\DomPDF\Facade\Pdf;
```

```
// PDF Export Logic
public function exportToPDF()
{
    // Fetch all users
    $users = User::all();

    // Load the view and pass the users data to it
    $pdf = PDF::loadView('admin.user.pdf', compact('users'));

    // Return the generated PDF to download
    return $pdf->download('users_list.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

```
Route::get('/users/export-pdf', [UserController::class, 'exportToPDF'])->name('users.export.pdf');
```

In your blade file:

```
<a href="{{ route('users.export.pdf') }}" class="btn btn-success">Export to PDF</a>
```

b) Export to excel:

```
composer require maatwebsite/excel
```

```
php artisan make:export UsersExport --model=User
```

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

```
Route::get('/users/export', [UserController::class, 'exportToExcel']->name('users.export'));
```

Inside your blade file:

```
<a href="{{ route('users.export') }}" class="btn btn-success">Excel</a>
```



### c) 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');
}
```

### d) Import excel file and insert it into users table:

```
php artisan make:import UsersImport --model=User
```

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

```
// Handle the file upload and import
Route::post('/users/import', [UserController::class, 'import'])->name('users.import');
```

### 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>
```

### i) Laravel authentication with socialite:

a) Login with google:

<https://www.youtube.com/watch?v=klegf18-0Mo>

```
// =====google login=====
public function googleLogin()
{
    return Socialite::driver('google')->redirect();
}

public function googleHandle()
{
    try {
        $user = Socialite::driver('google')->user();
        $user = User::firstOrCreate(['email' => $user->email], [
            'name' => $user->name,
            'email' => $user->email,
            'password' => Hash::make('12345')
        ]);
        Auth::login($user);
        return redirect()->route('admin.home');
    } catch (\Throwable $th) {
        dd($th->getMessage());
    }
}
```

b) Login with github

<https://www.youtube.com/watch?v=CbwzjLgcVR4>

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

```
// =====github login=====
public function githubLogin()
{
    return Socialite::driver('github')->redirect();
}

public function githubHandle()
{
    $user = Socialite::driver('github')->user();
    $user = User::firstOrCreate(['email' => $user->email], [
        'name' => $user->name,
        'email' => $user->email,
        'password' => Hash::make('12345')
    ]);
    Auth::login($user);
    return redirect()->route('admin.home');
}
```

c) Logout

```
// =====logout=====
public function logout()
{
    Auth::logout(); // Log the user out of the application.

    // Redirect to the home page.
    return redirect('/');
}
```

```
// logout
Route::post('/logout', [AuthController::class, 'logout']->name('logout'));
```

## 11) laravel all eloquent orm model concept.

### a) How to save data into table?

```
public function store(Request $request)
{
    // Validate incoming request data
    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|min:8',
    ]);

    // Store data
    User::create([
        'name' => $validatedData['name'],
        'email' => $validatedData['email'],
        'password' => bcrypt($validatedData['password']),
    ]);

    return response()->json(['message' => 'User created successfully!']);
}
```

### b) How to retrieve users data from database?

- Retrieve all records

```
// Fetch all users from the 'users' table
$users = User::all();
```

```
// Fetch all users (same as all())
$users = User::get();
```

- Apply constraints before fetching.

```
// Fetch users with specific conditions
$activeUsers = User::where('status', 'active')->get();
```

- Retrieve a single record by its primary key.

```
// Fetch a single user by ID
$user = User::find(1); // Returns null if not found
```

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

```
// Fetch user by ID or throw ModelNotFoundException if not found
$user = User::findOrFail(1);
```

- Retrieve the first matching record.

```
// Fetch the first user with a specific email
$user = User::where('email', 'johndoe@example.com')->first();
```

```
$user = User::firstWhere('email', 'johndoe@example.com');
```

```
// Equivalent to:
```

```
$user = User::where('email', 'johndoe@example.com')->first();
```

- Retrieve the first matching record or throw an exception.

```
// Fetch the first user or throw an exception if not found
$user = User::where('email', 'johndoe@example.com')->firstOrFail();
```

- Retrieve specific column values as a collection.

```
// Get all user emails as a collection
$emails = User::pluck('email');
```

```
// Get key-value pairs (email => name)
$users = User::pluck('name', 'email');
```

- Select specific columns from the table.

```
// Fetch only 'id', 'name', and 'email' columns
$users = User::select('id', 'name', 'email')->get();
```

- Process records in chunks to handle large datasets efficiently.

```
// Process 100 users at a time
User::chunk(100, function ($users) {
    foreach ($users as $user) {
        // Process each user
    }
});
```

```

public function showUsers(Request $request)
{
    $usersData = []; // Array to store all users' data

    // Chunk size (number of users per batch)
    $chunkSize = 100;

    // Retrieve users with a condition, e.g., 'active' status
    User::where('status', 'active')
        ->chunk($chunkSize, function ($users) use (&$usersData) {
            // Append the chunk of users to the array
            foreach ($users as $user) {
                $usersData[] = $user; // Add each user to the array
            }
        });

    // Pass the data to the Blade view
    return view('users.index', compact('usersData'));
}

```

- Retrieve records with pagination.

```

// Fetch 10 users per page
$users = User::paginate(10);

```

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

```

// Fetch 10 users per page without counting total records
$users = User::simplePaginate(10);

```

- Apply filtering conditions.

```

// Fetch users with multiple conditions
$filteredUsers = User::where('status', 'active')->where('role', 'admin')->get();

```

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

```

// Fetch users with roles 'admin' or 'super_admin'
$users = User::whereIn('role', ['admin', 'super_admin'])->get();

```

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

```
// Fetch users whose roles are not 'banned' or 'suspended'
$users = User::whereIn('role', ['banned', 'suspended'])->get();
```

- Filter records based on a range.

```
// Fetch users aged between 18 and 30
$users = User::whereBetween('age', [18, 30])->get();
```

```
// Fetch users not aged between 18 and 30
$users = User::whereNotBetween('age', [18, 30])->get();
```

- Sort records by a specific column.

```
// Fetch users sorted by creation date (newest first)
$users = User::orderBy('created_at', 'desc')->get();
```

- Limit the number of records retrieved.

```
// Fetch only 5 users
$users = User::limit(5)->get();
```

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

```
// Count all users
$totalUsers = User::count();
```

```
// Get the average age of users
$averageAge = User::avg('age');
```

- Retrieve related data via Eloquent relationships.

```
// Fetch users with their posts (assuming a 'posts' relationship)
$usersWithPosts = User::with('posts')->get();
```

- Filter based on the existence of a relationship.

```
// Fetch users with posts
$usersWithPosts = User::has('posts')->get();
```

- Join with another table to fetch combined data.



```
// Join 'users' and 'roles' tables
$users = User::join('roles', 'users.role_id', '=', 'roles.id')
    ->select('users.*', 'roles.name as role_name')
    ->get();
```

- Retrieve unique records.

```
// Get distinct roles
$uniqueRoles = User::select('role')->distinct()->get();
```

- Define reusable query logic in the model.

```
// Define scope in User model
public function scopeActive($query)
{
    return $query->where('status', 'active');
}

// Fetch active users using the scope
$activeUsers = User::active()->get();
```

- Execute raw SQL queries for complex use cases.

```
// Use raw SQL for aggregate grouping
$users = User::selectRaw('count(*) as total, role')->groupBy('role')->get();
```

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

```
// Iterate through users using a generator
foreach (User::cursor() as $user) {
    // Process each user
}
```

```
// Retrieve users and process them using cursor
$users = User::cursor();
```

```
// Retrieve users with multiple conditions
$users = User::where('role', $role)
    ->where('status', $status)
    ->cursor();
```

```
// Retrieve users created after the specified date
$users = User::whereDate('created_at', '>=', $date)->cursor();

// Retrieve and sort users based on created_at with a condition
$users = User::where('status', 'active')
    ->orderBy('created_at', $sortDirection)
    ->cursor();
```

- Retrieve users data with dynamic filters:

```
public function users($keyword = null, $entries = 10, $minDate = null, $maxDate = null, $sortField = 'created_at', $sortDirection = 'asc')
{
    // Start building a query on the User model
    $query = User::query();

    // Apply a keyword filter if provided
    if ($keyword) {
        $query->where(function ($q) use ($keyword) {
            // Search in 'name', 'email', and 'location' columns for the keyword
            $q->where('name', 'like', '%' . $keyword . '%') // Matches if 'name' contains the keyword
                ->orWhere('email', 'like', '%' . $keyword . '%') // Matches if 'email' contains the keyword
                ->orWhere('location', 'like', '%' . $keyword . '%'); // Matches if 'location' contains the keyword
        });
    }

    // Apply a minimum date filter if provided
    if ($minDate) {
        $query->whereDate('created_at', '>=', $minDate); // Fetch users created on or after $minDate
    }

    // Apply a maximum date filter if provided
    if ($maxDate) {
        $query->whereDate('created_at', '<=', $maxDate); // Fetch users created on or before $maxDate
    }

    // Apply sorting based on the given field and direction
    $query->orderBy($sortField, $sortDirection);

    // Limit the number of results per page and paginate
    $users = $query->paginate($entries); // Fetch results with pagination, $entries per page

    // Pass the retrieved users to the view 'users.index' for display
    return view('users.index', compact('users'));
}
```

- Retrieve users data with relationship song with dynamic filters.

```

public function users($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'));
}

```

### c) How to delete users data from database?

- Delete a Record by ID

```

// Find the user by ID and delete
$user = User::find(1); // Replace 1 with the desired user ID
if ($user) {
    $user->delete(); // Deletes the user record
}

```

```

// Deletes the user with ID 1
User::destroy(1);

```

- Delete a Record Using findOrFail

```

// Find the user by ID or throw an exception
$user = User::findOrFail(1);
$user->delete(); // Deletes the user record

```

- Delete a Record Using where

```

// Delete users where 'status' is 'inactive'
User::where('status', 'inactive')->delete();

```

- Delete Multiple Records Using wherein

```

// Delete users with specific IDs
User::whereIn('id', [1, 2, 3])->delete();

```

```
// Deletes users with IDs 1, 2, and 3
User::destroy([1, 2, 3]);
```

```
// Delete users using a collection of IDs
$ids = collect([4, 5, 6]);
User::destroy($ids);
```

- Delete All Records

```
// Delete all users
User::truncate(); // Recommended for clearing a table as it resets auto-increment IDs
```

- Find the first matching record and delete it.

```
// Delete the first user with a specific email
$user = User::firstWhere('email', 'johndoe@example.com');
if ($user) {
    $user->delete(); // Deletes the user
}
```

- Delete Using Relationships

```
// Delete a user and their posts (assuming a 'posts' relationship)
$user = User::with('posts')->find(1);
if ($user) {
    $user->posts()->delete(); // Deletes the user's related posts
    $user->delete(); // Deletes the user
}
```

- Delete records without loading models by using Query Builder

```
// Delete users where 'status' is 'inactive'
DB::table('users')->where('status', 'inactive')->delete();
```

- Delete records in chunks to handle large datasets.

```
// Delete inactive users in batches of 100
User::where('status', 'inactive')->chunk(100, function ($users) {
    foreach ($users as $user) {
        $user->delete(); // Delete each user in the chunk
    }
});
```

- Use advanced queries for conditional deletion.

```
// Delete users created more than a year ago
User::where('created_at', '<', now()->subYear())->delete();
```

- Soft Delete

```
class User extends Model
{
    use SoftDeletes; // Enables soft delete functionality

    // Optional: Define the name of the deleted_at column (default is 'deleted_at')
    const DELETED_AT = 'deleted_at';
}
```

```
Schema::table('users', function (Blueprint $table) {
    $table->softDeletes(); // Adds the 'deleted_at' column
});
```

#### Delete Records with Soft Deletes

```
// Soft delete a user
$user = User::find(1);
if ($user) {
    $user->delete(); // Marks the user as deleted without removing it from the database
}
```

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

```
// Restore a soft-deleted user
$user = User::withTrashed()->find(1);
if ($user) {
    $user->restore(); // Restores the soft-deleted user
}
```

#### Delete All Soft Deleted Records

```
// Permanently delete all soft-deleted users
User::onlyTrashed()->forceDelete();
```

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

```
// Find the user by ID and permanently delete it
$user = User::find(1);
if ($user) {
    $user->forceDelete(); // Permanently deletes the user
}
```

Retrieve all the users with Soft-Deleted Records

```
$users = User::withTrashed()->get(); // Includes soft-deleted records in the result
```

Retrieve only Soft-Deleted Records:

```
$deletedUsers = User::onlyTrashed()->get(); // Fetches only soft-deleted records
```

Completely remove a soft-deleted record from the database:

```
$user = User::withTrashed()->find(1);
$user->forceDelete(); // Permanently deletes the record from the database
```

#### d) How to update data from database?

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

```
public function updateUser(Request $request, $id)
{
    // Update the user directly by ID using data from the request
    $updated = User::where('id', $id)->update([
        'name' => $request->input('name'),
        'email' => $request->input('email'),
        'location' => $request->input('location'),
    ]);

    // Check if the update was successful
    if ($updated) {
        return response()->json(['message' => 'User updated successfully.']);
    }

    return response()->json(['message' => 'User not found.'], 404);
}
```

- Update Using Model Instance (with Request)

```

public function updateUser(Request $request, $id)
{
    // Retrieve the user by its ID
    $user = User::find($id);

    // Check if the user exists
    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Update the user's attributes with data from the request
    $user->name = $request->input('name');
    $user->email = $request->input('email');
    $user->location = $request->input('location');

    // Save the updated data to the database
    $user->save();

    return response()->json(['message' => 'User updated successfully.']);
}

```

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

```

public function bulkUpdateUsers(Request $request, $location)
{
    // Get the new status from the request
    $status = $request->input('status');

    // Update all users in a specific location
    $updated = User::where('location', $location)->update([
        'status' => $status,
    ]);

    return response()->json(['message' => "$updated users updated successfully."]);
}

```

- 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.

```

public function updateOrCreateUser(Request $request)
{
    // Get user input from the request
    $email = $request->input('email');
    $name = $request->input('name');
    $location = $request->input('location');

    // Update or create the user
    $user = User::updateOrCreate(
        ['email' => $email], // Condition to check if the user exists
        ['name' => $name, 'location' => $location] // Data to update or create
    );

    return response()->json(['message' => 'User updated or created successfully.'],
}

```

- 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.

```

public function updateUser(Request $request, $id)
{
    // Retrieve the user by its ID
    $user = User::find($id);

    // Check if the user exists
    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Mass update using the fill method
    $user->fill($request->only(['name', 'email', 'location']));

    // Save the updated data to the database
    $user->save();

    return response()->json(['message' => 'User updated successfully.']);
}

```



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

```
public function updateUserStatus(Request $request, $id)
{
    // Retrieve the user by its ID
    $user = User::find($id);

    // Check if the user exists and if the status is not already active
    if ($user && $user->status != 'active') {
        $user->status = 'active';
        $user->save();

        return response()->json(['message' => 'User status updated to active.']);
    }

    return response()->json(['message' => 'User not found or status is already active.'], 404);
}
```

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

```
public function updateMultipleUsers(Request $request)
{
    $status = $request->input('status');
    $location = $request->input('location');

    // Update users who have a specific status
    $updated = User::where('status', 'inactive')->where('location', $location)->update([
        'status' => $status,
    ]);

    return response()->json(['message' => "$updated users updated successfully."]);
}
```

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

```
use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Validator;

public function updateUser(Request $request, $id)
{
    // Validate the incoming request data
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email,' . $id,
    ]);

    // Check if the validation fails
    if ($validator->fails()) {
        return response()->json(['errors' => $validator->errors()], 422);
    }

    // Retrieve the user by its ID
    $user = User::find($id);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Update the user
    $user->update($request->only(['name', 'email']));

    return response()->json(['message' => 'User updated successfully.']);
}
```

Copy code

Continue generating

- 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:

```

public function updateUserProfile(Request $request, $userId)
{
    // Retrieve the user by its ID
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Update the user's profile relationship
    $user->profile->update([
        'bio' => $request->input('bio'),
        'location' => $request->input('location')
    ]);

    return response()->json(['message' => 'Profile updated successfully.']);
}

```

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

```

public function updateUserPosts(Request $request, $userId)
{
    // Retrieve the user by its ID
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Update all posts for this user
    $user->posts()->update([
        'status' => $request->input('status')
    ]);

    return response()->json(['message' => 'Posts updated successfully.']);
}

```

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

```

public function updateUserPost(Request $request, $userId, $postId)
{
    // Retrieve the user and the specific post
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    $post = $user->posts()->find($postId);

    if (!$post) {
        return response()->json(['message' => 'Post not found.'], 404);
    }

    // Update the specific post
    $post->update([
        'title' => $request->input('title'),
        'content' => $request->input('content')
    ]);

    return response()->json(['message' => 'Post updated successfully.']);
}

```

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

```

public function updateUserWithProfileAndPosts(Request $request, $userId)
{
    // Retrieve the user by its ID
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Update the user's profile
    $user->profile()->update([
        'bio' => $request->input('bio')
    ]);

    // Update the user's posts
    $user->posts()->update([
        'status' => $request->input('status')
    ]);

    return response()->json(['message' => 'User, profile, and posts updated successfully.'], 200);
}

```

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

```

public function updateUserRoles(Request $request, $userId)
{
    // Retrieve the user by ID
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['message' => 'User not found.'], 404);
    }

    // Attach new roles (you can also detach or sync roles)
    $user->roles()->sync($request->input('roles')); // Sync roles passed in the request

    return response()->json(['message' => 'Roles updated successfully.'], 200);
}

```

## 12) 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_DEPRECATED_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!";  
?>
```

