

Part 1: Setting up Database and Controllers

I. Create a Database and Set Up Connection

1. MySQL Database:

DATABASE eventify1.1

.env file

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=eventify1.1_db
DB_USERNAME=root
DB_PASSWORD=
```

```
SESSION_DRIVER=file
```

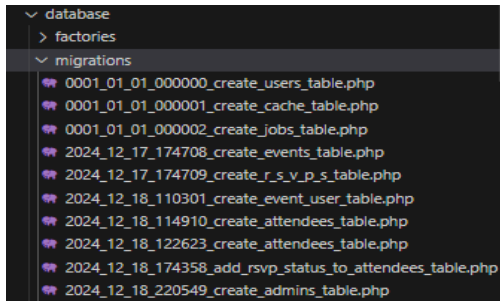
Session Configuration:

config/session.php

```
'driver' => env('SESSION_DRIVER', 'file'),
```

2. Create Tables Using Migrations:

Laravel uses migrations to create tables. To run migrations:



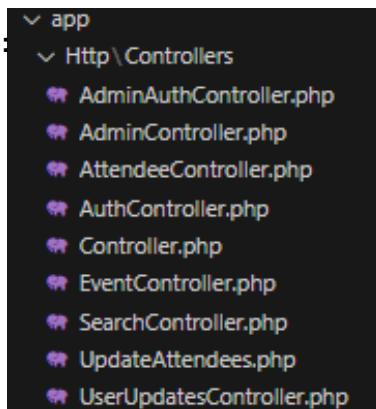
Creating migration files like:

```
php artisan make:migration create_events_table
```

Define the schema in the migration file, then run:

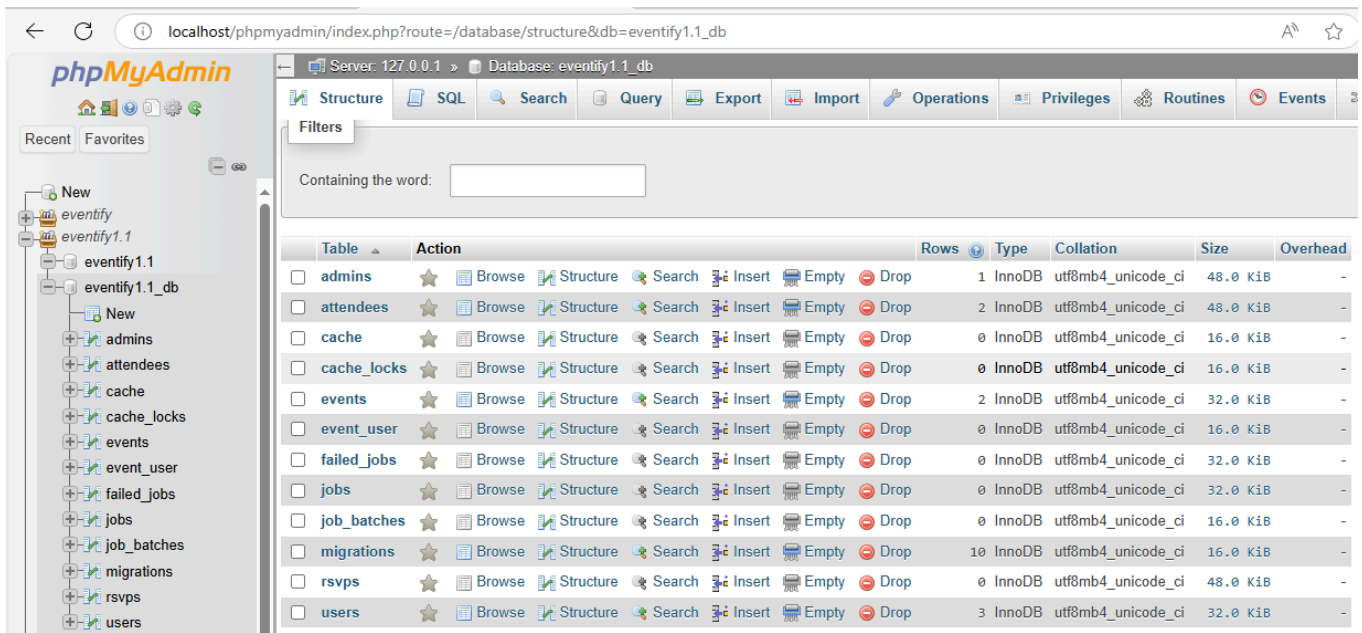
```
php artisan migrate
```

II. Step 2: Create Controllers: Each controller serves different routes and methods to interact with data.



Example: AdminController

```
class AdminController extends Controller
{
    1 reference | 0 overrides
    public function show(): View
    {
        $events = Event::all();
        return view('admin.AdminManageEvent', compact('events'));
    }
    1 reference | 0 overrides
    public function edit(string $id): View{
        $event = Event::findOrFail($id);
        return view('admin.EventEdit', compact('event'));
    }
    0 references | 0 overrides
    public function editUser(Request $request, $id): void{
    }
    1 reference | 0 overrides
    public function update(Request $request, string $id): RedirectResponse{
        $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'required|string',
            'event_date' => 'required|date',
            'location' => 'required|string',
            'category' => 'required|string',
        ]);
        $event = Event::findOrFail($id);
        $event->update($request->all());
        return redirect()->route('admin.ManageEvent')->with('success', 'Event updated successfully');
    }
    1 reference | 0 overrides
    public function destroy(string $id): RedirectResponse{
        $event = Event::findOrFail($id);
        $event->delete();
        return redirect()->route('admin.ManageEvent')->with('success', 'Event deleted successfully');
    }
    1 reference | 0 overrides
    public function dashboard(): View{
        $events = Event::all(); // Fetch all events
        return view('admin.dashboard', compact('events'));
    }
}
```



III.: Register Controllers in `routes/web.php`

In `routes/web.php`, register the routes and associate them with controller methods.

```
Route::get('/events/add-attendees', [EventController::class, 'addAttendeesForm'])->name('attendees');
Route::post('/events/add-attendees', [EventController::class, 'storeAttendee'])->name('events.addAttendees');
Route::get('/events/{event}/edit', [EventController::class, 'edit'])->name('events.edit');

Route::post('checkname', [AttendeeController::class, 'checkName'])->name('checkname');
Route::put('attendees/{id}/rsvp', [AttendeeController::class, 'updateRsvp'])->name('updateRsvp');

Route::put('events/{event}', [EventController::class, 'update'])->name('events.update');
Route::delete('events/{event}', [EventController::class, 'destroy'])->name('events.destroy');

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

// Auth Routes
Route::get('register', [AuthController::class, 'showRegisterForm'])->name('register');
Route::post('register', [AuthController::class, 'register'])->name('register');
Route::get('login', [AuthController::class, 'showLoginForm'])->name('login');
Route::post('login', [AuthController::class, 'login'])->name('login');
Route::post('logout', [AuthController::class, 'logout'])->name('logout');

// Admin Auth Routes
Route::get('admin/register', [AdminAuthController::class, 'showRegisterForm'])->name('admin.register');
Route::post('admin/register', [AdminAuthController::class, 'register'])->name('admin.register');
Route::get('admin/login', [AdminAuthController::class, 'showLoginForm'])->name('admin.login');
Route::post('admin/login', [AdminAuthController::class, 'login'])->name('admin.login');
Route::get('admin/logout', [AdminAuthController::class, 'logout'])->name('admin.logout');

// Event Routes
Route::resource('events', EventController::class);
Route::post('events/{event}/rsvp', [EventController::class, 'rsvp'])->name('events.rsvp');
```

```
// Attendees CRUD
Route::get('attendees/{attendee}/edit', [AttendeeController::class, 'edit'])->name('attendees.edit');
Route::put('attendees/{attendee}', [AttendeeController::class, 'update'])->name('attendees.update');
Route::delete('attendees/{attendee}', [AttendeeController::class, 'destroy'])->name('attendees.destroy');

Route::get('/dashboard', [AdminController::class, 'dashboard'])->name('dashboard');
Route::get('admin/manage', [AdminController::class, 'show'])->name('admin.ManageEvent');
Route::get('manage/{id}/edit', [AdminController::class, 'edit'])->name('admin.editing');
Route::put('admin/{event}', [AdminController::class, 'update'])->name('admin.update');
Route::delete('admin/{event}', [AdminController::class, 'destroy'])->name('admin.destroy');

//admin edit attendees
Route::get('admin/manage-events', [UpdateAttendees::class, 'index'])->name('admin.ManageAttendeesAdmin');
Route::get('admin/attendees/{id}/edit', [UpdateAttendees::class, 'edit'])->name('admin.attendees.edit');
Route::put('admin/attendees/{id}', [UpdateAttendees::class, 'update'])->name('admin.attendees.update');
Route::delete('admin/attendees/{id}', [UpdateAttendees::class, 'destroy'])->name('admin.attendees.destroy');
Route::resource('users', UserUpdatesController::class);

// users
// Show all users
Route::get('/users', [UserUpdatesController::class, 'index'])->name('users.ManageUserslog');

// Show the form for editing a user
Route::get('users/{id}/edit', [UserUpdatesController::class, 'edit'])->name('users.edit');

// Update a user
Route::put('users/{id}', [UserUpdatesController::class, 'update'])->name('users.update');

// Delete a user
Route::delete('users/{id}', [UserUpdatesController::class, 'destroy'])->name('users.destroy');
```

Part 2: Account Registration and Login

I. Implement Registration and Login Pages: `resources/views/auth/`.

Login Form: `views/auth/login.blade.php`

```
<body>
<div class="register-container">
<!-- Image and Intro Section -->
<div class="image-section">

<p>Your journey to unforgettable events starts here. Join us today and explore endless possibilities.</p>
</div>

<!-- Form Section -->
<div class="form-section">
<h1>Log into Your Account</h1>
<form action="{{ url('login') }}" method="POST">
@csrf
<input type="email" name="email" placeholder="Email" class="form-control" required>
<input type="password" name="password" placeholder="Password" class="form-control" required>
<button type="submit">Login</button>
</form>
<div class="footer">
Don't have an account? <a href="{{ url('register') }}">Register here</a>
<br>
<a href="javascript:history.back()" class="back-button">Back</a>
</div>
</div>
</body>
```

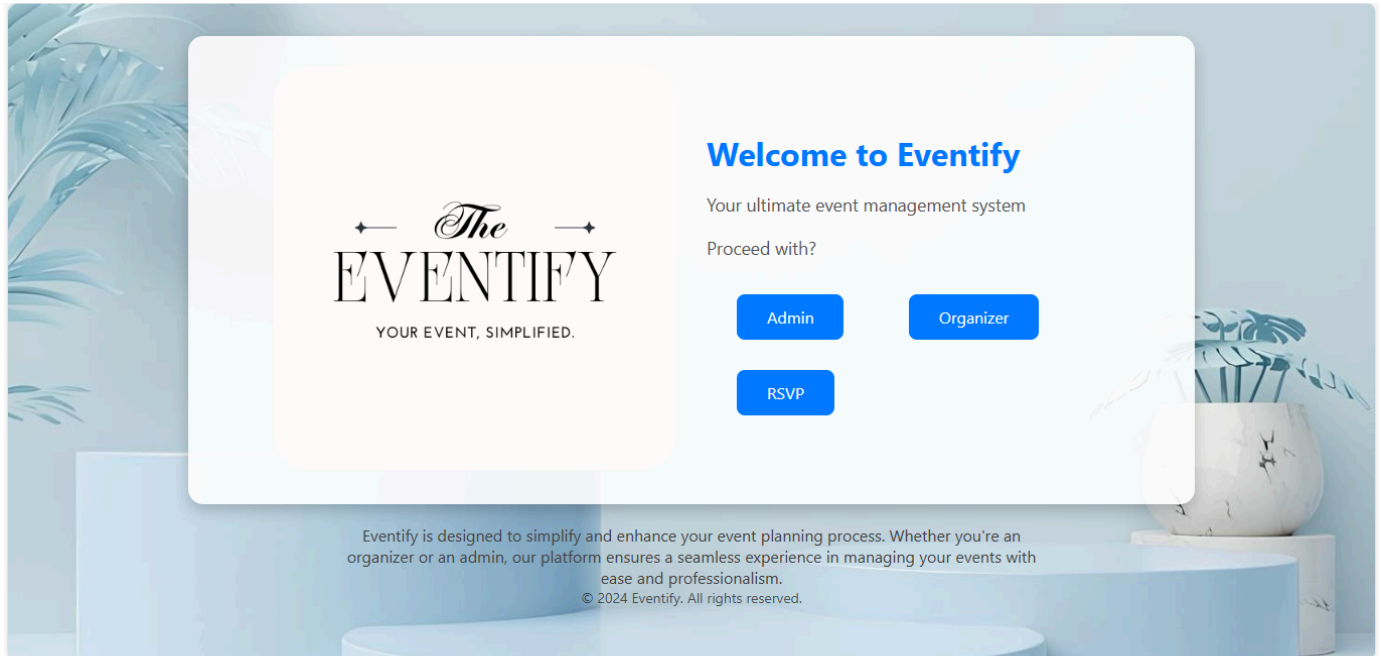
Register Form: `views/auth/register.blade.php`

```
<body>
<div class="register-container">
<!-- Image and Intro Section -->
<div class="image-section">

<p>Your journey to unforgettable events starts here. Join us today and explore endless possibilities.</p>
</div>

<!-- Form Section -->
<div class="form-section">
<h1>Register</h1>
<form action="{{ url('register') }}" method="POST">
@csrf
<input type="text" name="name" placeholder="Name" class="form-control" required>
<input type="email" name="email" placeholder="Email" class="form-control" required>
<input type="password" name="password" placeholder="Password" class="form-control" required>
<input type="password" name="password_confirmation" placeholder="Confirm Password" class="form-control" required>
<button type="submit">Register</button>
</form>
<div class="Footer">
Already have an account? <a href="{{ url('login') }}">Login here</a>
<br>
<a href="javascript:history.back()" class="back-button">Back</a>
</div>
</div>
</body>
```

- Users register and log in via **AuthController**, which uses Laravel's authentication features.
 - The form posts data to the login/register route.
 - Includes CSRF protection (**@csrf**) to prevent cross-site request forgery.
 - Collects user email and password for authentication.
- Implement Role Selection: When registering, we can choose roles like 'Organizer' or 'Admin'. Can add a role column to users table and handle role assignment in the registration process.



For Admin:

 A screenshot of the "Register as Admin" form. The form includes the Eventify logo and tagline. The registration fields are: "Username" (text input), "Email" (text input), and "Password" (text input with a note: "Password must contain atleast 8 characters."). Below the password field is a "Confirm Password" field. A blue "Register" button is at the bottom. At the bottom of the form, there is a link: "Already have an account? Login here Back".

 A screenshot of the "Log in as Admin" form. The form includes the Eventify logo and tagline. The login fields are: "Email" (text input) and "Password" (text input). A blue "Login" button is at the bottom. Below the button is a link: "Don't have an account? Register here Back".

For Organizer:

 A screenshot of the "Register" form for organizers. The form includes the Eventify logo and tagline. The registration fields are: "Name" (text input), "Email" (text input), "Password" (text input), and "Confirm Password" (text input). A blue "Register" button is at the bottom. At the bottom of the form, there is a link: "Already have an account? Login here Back".

 A screenshot of the "Log into Your Account" form for organizers. The form includes the Eventify logo and tagline. The login fields are: "Email" (text input) and "Password" (text input). A blue "Login" button is at the bottom. Below the button is a link: "Don't have an account? Register here Back".

Part 3: Populating Pages Using Data from Database

I. Fetch Data in Controller and Pass to Views

In `AdminController`, fetch data from the `Event` model and pass it to the view.

`$events`

```
public function show(): View
{
    $events = Event::all();
    return view('admin.AdminManageEvent', compact('events'));
}

1 reference | 0 overrides
public function edit(string $id): View{
    $event = Event::findOrFail($id);
    return view('admin.EventEdit', compact('event'));
}

0 references | 0 overrides
public function editUser(Request $request, $id): void{
}

1 reference | 0 overrides
public function update(Request $request, string $id): RedirectResponse{
    $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'required|string',
        'event_date' => 'required|date',
        'location' => 'required|string',
        'category' => 'required|string',
    ]);
    $event = Event::findOrFail($id);
    $event->update($request->all());
    return redirect()->route('admin.ManageEvent')->with('success', 'Event updated successfully');
}

1 reference | 0 overrides
public function destroy(string $id): RedirectResponse{
    $event = Event::findOrFail($id);
    $event->delete();
    return redirect()->route('admin.ManageEvent')->with('success', 'Event deleted successfully');
}

1 reference | 0 overrides
public function dashboard(): View{
    $events = Event::all(); // Fetch all events
    return view('admin.dashboard', compact('events'));
}
```

Model/Event.php

```
namespace App\Models;

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

20 references | 0 implementations
class Event extends Model
{
    use HasFactory;

    0 references
    protected $table = 'events';

    0 references
    protected $fillable = [
        'title',
        'description',
        'event_date',
        'location',
        'category',
        'user_id',
    ];

    0 references | 0 overrides
    public function attendees(): HasMany
    {
        return $this->hasMany(Attendee::class);
    }
}
```

Key Functions:

- **show()**: Fetches all events and passes them to a view.
- **edit()**: Loads a specific event for editing.
- **update()**: Updates an event's details in the database.
- **destroy()**: Deletes an event.

Provides CRUD functionality for event management.

The `$events` variable in the `AdminController` is used to store event data fetched from the database, typically using the `Event` model. It represents a collection of event records that can be passed to a view for display or further processing. Each record in `$events` is accessed as `$event` to display details such as ID, title, and description.

The **AdminController** handles administrative tasks related to events, including creating, editing, updating, and deleting events. It interacts with the `Event` model to perform database operations and passes data to the admin views. The **dashboard()** method loads the admin dashboard, serving as the central hub for managing events. This controller ensures efficient, organized, and secure handling of admin functionalities in the application.

The **Event model** represents the events table in the database and manages event-related data. It uses the `HasFactory` trait for generating test data and specifies the fillable fields (title, description, event_date, location, category, user_id) to allow secure mass assignment. The model defines a one-to-many relationship with the `Attendee` model using the **attendees()** method.

Display Data in Views

[resources/views/admin/AdminManageEvent.blade.php](#):

```
@extends('layouts.adminlayout')
@section('title', 'Manage Events')
@section('header', 'Event List')

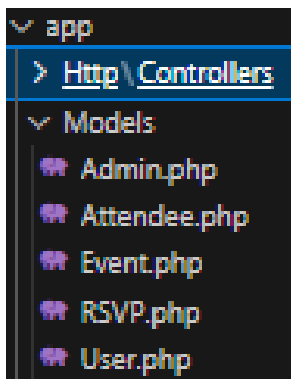
@section('content')


| ID | Title   | Description   | Event Date | Location   | Category   | Action                     |
|----|---------|---------------|------------|------------|------------|----------------------------|
| 1  | Event 1 | Description 1 | 2023-10-26 | Location 1 | Category 1 | <a href="#">Edit Event</a> |


@endsection
```

In the **AdminManageEvent.blade.php** view, the **\$events** data is passed from the controller, where **Event::all()** retrieves all events from the database. The view uses a **@foreach** loop to display each event's details (e.g., id, title, description) in a table. Each row shows event data, and the "Edit Event" link is dynamically created for each event, allowing the admin to edit it. This process ensures that the event data from the database is displayed and managed efficiently in the admin interface.

Models: In [app/Models/Event.php](#), and [app/Models/Attendee.php](#), etc. define relationships and fillable fields:



Event model: Defines a one-to-many relationship with the Attendee model and specifies the fillable fields for event data.

Attendee model: Defines a belongs-to relationship with the Event model and specifies fillable fields for attendee data.

Event Management: Eventify

- User Registration and Authentication
- Event Creation and Management
- RSVP System
- Event Search and Filtering
- Admin Dashboard

For Admin
Dashboard:

Eventify

Welcome, Admin!

ADMIN

Dashboard

Manage Events

Manage Attendees

Manage Users

Upcomming Events

ID	NAME	DESCRIPTION	CREATED AT	UPDATED AT
5	kasal ng aso ko at aso mo	yeho	2024-12-19 08:11:56	2024-12-19 15:18:19
6	kasalq	aylee	2024-12-19 15:19:44	2024-12-19 15:19:44

Eventify

Welcome, Admin!

ADMIN

Dashboard

Manage Events

Manage Attendees

Manage Users

ID	TITLE	DESCRIPTION	EVENT DATE	LOCATION	CATEGORY	ACTION
5	kasal ng aso ko at aso mo	yeho	2029-07-09	sa bahay lang	parang ano	Edit Event

NAME	RSVP STATUS	ACTION

ID	TITLE	DESCRIPTION	EVENT DATE	LOCATION	CATEGORY	ACTION
6	kasalq	aylee	2027-01-02	kung saan mo gusto	beach wedding	Edit Event

NAME	RSVP STATUS	ACTION
andreev	Yes	Edit Name or RSVP?
sarah	Yes	Edit Name or RSVP?

Eventify

Welcome, Admin!

ADMIN

Dashboard

Manage Events

Manage Attendees

Manage Users

Users List

Total Users: 3

NAME	EMAIL	ACTIONS
sarah	hubikim8@gmail.com	Edit
naih	mingkai103019@gmail.com	Edit
Admin Name	janineishe@gmail.com	Edit

Eventify

Welcome, Admin!

ADMIN

Dashboard

Manage Events

Manage Attendees

Manage Users

ID	TITLE	DESCRIPTION	EVENT DATE	LOCATION	CATEGORY	ACTION
5	kasal ng aso ko at aso mo	yeho	2029-07-09	sa bahay lang	parang ano	Edit Event

ID	TITLE	DESCRIPTION	EVENT DATE	LOCATION	CATEGORY	ACTION
6	kasalq	aylee	2027-01-02	kung saan mo gusto	beach wedding	Edit Event

RSVP

←

The

→

EVENTIFY

YOUR EVENT, SIMPLIFIED.

Your journey to unforgettable events starts here. Enter your name and get started!

Check Your Attendance

Enter Your Name

Check

Organizer? [Log in here](#)

[Back](#)

Check Your Attendance

Attendee Name

sarah

RSVP Status

Yes

Update RSVP

Thank You!

Your RSVP has been successfully updated.

Go back to your event

For Organizer
Dashboard:

Eventify

Welcome! naih

Organizer

Dashboard

Create Event

Add Attendees

Manage Events

Upcoming Events

Search by title, Category, date, or location

Search

kasalq

ayiee

Date: 2027-01-02

Location: kung saan mo gusto

Category: beach wedding

VIEW RSVP

Eventify

Welcome! naih

Organizer

Dashboard

Create Event

Add Attendees

Manage Events

Create New Event

Event Title

Event Title

Event Description

Event Description

Event Date

dd/mm/yyyy

Eventify

Welcome! naih

Organizer

Dashboard

Create Event

Add Attendees

Manage Events

Add Attendees to Events

Select Event

-- Choose Event --

Attendee Name

Enter attendee name

Add Attendee

Event Attendees

kasalq

Attendee Name

Eventify

Welcome! naih

Organizer

Dashboard

Create Event

Add Attendees

Manage Events

ID	TITLE	DESCRIPTION	EVENT DATE	LOCATION	CATEGORY	ACTION
6	kasalq	ayiee	2027-01-02	kung saan mo gusto	beach wedding	Edit Event

NAME	RSVP STATUS	ACTION
andreev	Yes	Edit Name or RSVP?
sarah	Yes	Edit Name or RSVP?

Database
For users:

phpMyAdmin

Recent Favorites

New

eventify

eventify1.1

eventify1.1_db

New

admins

attendees

cache

cache_locks

events

event_user

failed_jobs

jobs

job_batches

migrations

rsmps

users

Information schema

Server: 127.0.0.1 » Database: eventify1.1_db » Table: users

Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers

Showing rows 0 - 2 (3 total, Query took 0.0385 seconds.)

SELECT * FROM `users`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	name	email	password	role	created_at	updated_at
<input type="checkbox"/> Edit Copy Delete	1	sarah	hubikim8@gmail.com	\$2y\$12\$MHmZOEHSULyAkZ92CeZ1.7.2VDyR9GWHEN4a66GZNR5...	attendee	2024-12-18 18:13:33	2024-12-18 18:13:33
<input type="checkbox"/> Edit Copy Delete	2	naih	mingkai103019@gmail.com	\$2y\$12\$SdttdtGmj2Y.zN125lmhoekbMFdy1jjiKoR9bnUaSr0E...	attendee	2024-12-19 05:01:25	2024-12-19 05:01:25
<input type="checkbox"/> Edit Copy Delete	4	Admin Name	janineishe@gmail.com	\$2y\$12\$75a.AbjcZMbsZBFVHXG/xO3J4k4ZW1ILV.d4OcZeoWu...	attendee	2024-12-19 12:01:23	2024-12-19 12:01:23

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

For events:

eventify1.1

eventify1.1

eventify1.1_db

New

admins

attendees

cache

cache_locks

events

event_user

failed_jobs

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	title	description	event_date	location	category	user_id	created_at	updated_at
<input type="checkbox"/> Edit Copy Delete	5	kasal ng aso ko at aso mo	yeho	2029-07-09	sa bahay lang	parang ano	1	2024-12-19 08:11:56	2024-12-19 15:18:19
<input type="checkbox"/> Edit Copy Delete	6	kasalq	ayiee	2027-01-02	kung saan mo gusto	beach wedding	2	2024-12-19 15:19:44	2024-12-19 15:19:44

For admins:

admins

attendees

cache

cache_locks

events

Extra options

		id	username	email	password	created_at	updated_at
<input type="checkbox"/>	Edit Copy Delete	1	sarah	sarah@example.com	\$2y\$12\$qMVQF2zV836ePtuvfiXyjebuTFIMVicpQX2CcxUkWvc...	2024-12-19 13:38:46	2024-12-19 13:38:46

For attendees:

attendees

cache

cache_locks

events

		id	event_id	name	rsvp_status	created_at	updated_at
<input type="checkbox"/>	Edit Copy Delete	5	6	andreev	Yes	2024-12-19 15:20:08	2024-12-19 17:25:16
<input type="checkbox"/>	Edit Copy Delete	6	6	sarah	Yes	2024-12-19 15:20:17	2024-12-19 15:20:41