

HƯỚNG DẪN FIX LỖI KHI INSTALL LARAVEL/REVERB

Lỗi hiển thị:

```
PS C:\dez\php-k24wd02\laravel\laravel-app-03> composer require laravel/reverb
./composer.json has been updated
Running composer update laravel/reverb
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

TypeError

Pusher\Pusher::__construct(): Argument #1 ($auth_key) must be of type string, null given, called in C:\dez\php-k24wd02\laravel\laravel-app-03\vendor\laravel\framework\src\Illuminate\Broadcasting\BroadcastManager.php on line 350

at vendor\pusher\pusher-php-server\src\Pusher.php:63
 59      * @param ClientInterface|null $client [optional] - a Guzzle client to use for all HTTP requests
 60      *
 61      * @throws PusherException Throws exception if any required dependencies are missing
 66
 67      $useTLS = true;

1 vendor\laravel\framework\src\Illuminate\Broadcasting\BroadcastManager.php:350
  Pusher\Pusher::__construct()

2 vendor\laravel\framework\src\Illuminate\Broadcasting\BroadcastManager.php:328
  Illuminate\Broadcasting\BroadcastManager::pusher()

Script @php artisan package:discover --ansi handling the post-autoload-dump event returned with error code 1
```

Lỗi này xảy ra vì Laravel khi khởi tạo driver Pusher để broadcast thì không tìm thấy hoặc không load được các biến môi trường cần thiết trong file `.env`.

Tập trung vào dòng lỗi này:

```
Pusher\Pusher::__construct(): Argument #1 ($auth_key) must be of type string, null given
```

Nghĩa là `$auth_key` đang bị *null*, tức là `PUSHER_APP_KEY` (hoặc các biến tương tự) chưa được set đúng.

Nguyên nhân:

Trong file `.env` chưa có các biến cấu hình Pusher :

Thêm Cấu hình Pusher trong file `.env`:

```
65  
66 BROADCAST_CONNECTION=reverb  
67 BROADCAST_DRIVER=reverb  
68  
69 REVERB_APP_ID=app-1  
70 REVERB_APP_KEY=dev-key-123456  
71 REVERB_APP_SECRET=dev-secret-abcdef  
72 REVERB_HOST=127.0.0.1  
73 REVERB_PORT=8080  
74 REVERB_SCHEME=http  
75  
76 VITE_REVERB_APP_KEY=dev-key-123456  
77 VITE_REVERB_HOST=127.0.0.1  
78 VITE_REVERB_PORT=8080  
79 VITE_REVERB_SCHEME=http  
80 VITE_APP_NAME="{{APP_NAME}}"
```

- Nếu các giá trị này không có, Laravel sẽ truyền null vào constructor của Pusher >> gây lỗi.
- Sau khi hoàn tất setup cấu hình theo ảnh trên, nhớ bấm `Ctrl+F` để search coi có bị trùng tên không nếu trùng thì comment nó lại giống vậy:

```
35  
36 #BROADCAST_CONNECTION=log
```

- Vì cấu hình trong `.env` không được trùng

Clear cache config:

```
php artisan config:clear  
php artisan cache:clear
```

Kiểm tra lại code trong file

```
export {{}};  
declare global {  
  interface window {  
    Echo: any;  
    Pusher: any;  
  }  
}
```

Đoạn code trên dùng cho mục đích **TypeScript declaration merging** (khai báo mở rộng global types) để giúp TypeScript hiểu được các biến toàn cục (`window.Echo` và `window.Pusher`) mà bình thường TypeScript không biết.

Phân tích từng phần:

```
export {};
```

- Đây là cách chuyển đổi file thành một **module** trong TypeScript.
- Nếu không có dòng này, file `.d.ts` hoặc `.ts` sẽ được coi là **script toàn cục**, dễ gây trùng cho mỗi namespace.
- Khi đã là module, chúng ta có thể mở rộng các interface có sẵn như `Window` .

```
declare global {  
  interface window {  
    Echo: any;  
    Pusher: any;  
  }  
}
```

1. declare global { ... }

Mục đích Mở rộng (augment) các khai báo type toàn cục (global scope).

2. interface window { ... }

- Ở đây chúng ta đang dùng **augment** interface `Window` của trình duyệt.
- Object `window` vốn đã tồn tại trong DOM lib (`lib.dom.d.ts`), nhưng không có thuộc tính `Echo` và `Pusher` .
- TypeScript mặc định sẽ báo lỗi khi chúng ta viết `window.Echo = ...` hoặc `window.Pusher = ...` vì chắc chắn nó làm gì có type nào bên trong!.

3. Echo: any; Pusher: any;

- Khai báo rằng `window` có thể có 2 thuộc tính mới là `Echo` và `Pusher` .
- Kiểu `any` để tránh lỗi type-checking.
- Sau này mở rộng dự án ra, chúng ta có thể thay `any` bằng type cụ thể (ví dụ: `LaravelEcho` hoặc `PusherStatic`).

Ví dụ thực tiễn

- Giúp TypeScript **không báo lỗi** khi chúng ta sử dụng lại module trên:

Ở đây chúng ta thấy `Echo` & `Pusher` đã được sử dụng và gọi như 1 module.

```
window.Echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
});
```

```
window.Pusher = Pusher;
```

- Nếu không có đoạn `declare global`, TypeScript sẽ báo lỗi:

```
Property 'Echo' does not exist on type 'Window & typeof globalThis'.
```

Chạy lại cài đặt LARAVEL REVERB:

```
composer require laravel/reverb
```

```
PS C:\dez\php-k24wd02\laravel\laravel-app-03> composer require laravel/reverb
./composer.json has been updated
Running composer update laravel/reverb
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

inertiajs/inertia-laravel ..... DONE
laravel/pail ..... DONE
laravel/reverb ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE

90 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

 INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
Using version ^1.5 for laravel/reverb
```

Cài đặt hệ thống Broadcasting của Laravel:

```
npm i laravel-echo pusher-js
```

- `npm i laravel-echo pusher-js` >> Cài **thư viện frontend** để client (React/Vue/Blade) có thể kết nối tới Reverb/Pusher và lắng nghe các event real-time.

Vậy bây giờ chúng ta đã cài đặt thành công:

- **Backend:** Laravel broadcasting (Cấu hình Laravel server-side broadcasting với Reverb có thể tự chạy WebSocket server thay cho Pusher cloud).
- **Frontend:** Echo + PusherJS để subscribe và nhận sự kiện.

Thêm ROUTES cho Broadcast Channel (khai báo quyền truy cập vào các kênh realtime)

routes\channels.php

```
<?php

use App\Models\Conversation;
use App\Models\User;
use Illuminate\Support\Facades\Broadcast;

Broadcast::channel(
    'conversation.{conversationId}',
    function (User $user, int $conversationId) {
        return Conversation::whereKey($conversationId)
            ->whereHas('users', fn($q) => $q->whereKey($user->id))
            ->exists();
    }
);
```

Đoạn code trên định nghĩa **Broadcast Channel** trong Laravel – nơi khai báo **quyền truy cập** vào các kênh realtime.

1. `Broadcast::channel('conversation.{conversationId}', ...)`

- Khai báo một channel có tên động: `conversation.{conversationId}` .
- `{conversationId}` là **wildcard** >> sẽ được thay bằng ID thật khi client subscribe.

Ví dụ: `Echo.private('conversation.12')` thì `$conversationId = 12` .

2. function (User \$user, int \$conversationId)

- Đây là **callback authorization function**.
- Laravel sẽ gọi function này khi một user cố gắng **subscribe** vào channel.
- `$user` là user đang login (tự động lấy từ guard).
- `$conversationId` là tham số lấy từ `{conversationId}` trong tên channel.

3. Logic kiểm tra:

```
return Conversation::whereKey($conversationId)
    ->whereHas('users', fn($q) => $q->whereKey($user->id))
    ->exists();
```

- `Conversation::whereKey($conversationId)`
| | Lấy đúng record conversation theo ID.
- `->whereHas('users', fn($q) => $q->whereKey($user->id))`
| | Kiểm tra trong mỗi quan hệ `users` của conversation đó có tồn tại user với ID bằng `$user->id` hay không.
- `->exists()`
| | Nếu tồn tại >> trả về `true` (cho phép subscribe).
| | Nếu không tồn tại >> `false` (bị từ chối).

Ý nghĩa

- Chỉ những user nào là **thành viên của conversation** mới được phép join vào channel realtime của cuộc trò chuyện đó.
- Nếu không, Laravel sẽ trả về **403 Forbidden** khi client cố subscribe.

Một Ví dụ dễ để hiểu hơn:

Hiện tại trong database có 2 tables là `conversations` và `conversation_user` (bảng trung gian).

- User **ID=5** cố subscribe (tham gia realtime) vào `conversation.10`.
- Laravel chạy function callback:
 - Tìm conversation ID=10.
 - Kiểm tra trong `conversation_user` có dữ liệu của (`conversation_id=10, user_id=5`) không.
- Nếu có >> cho phép join.
- Nếu không >> từ chối.

Đây là **authorization rule** cho **private channel**. Nó đảm bảo **chỉ thành viên trong conversation** mới được nghe/broadcast sự kiện **realtime** của **conversation** đó.

Broadcast một Event qua Reverb

A. `php artisan vendor:publish --tag=verbatim-config`

- Lệnh này publish file cấu hình `verbatim.php` từ package Laravel Reverb ra thư mục `config/`.
- File này chứa:
 - Host, port mà Reverb server sẽ chạy.
 - Các option bảo mật (auth, TLS, v.v).

Sau bước này mở thư mục `config/` ra sẽ thấy file:

```
config/verbatim.php
```

B. `php artisan make:event MessageCreated`

- Tạo một class Event mới ở thư mục:

```
app/Events/MessageCreated.php
```

Chỉnh lại code event để **broadcast** được:

```

namespace App\Events;

use App\Models\Message;

use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class MessageCreated
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    public function __construct(public Message $message){}

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('conversation.' . $this->message->conversation_id)
        ];
    }

    public function broadcastAs(): string { return 'MessageCreated'; }

    public function broadcastWith(): array
    {
        return [
            'message' => [
                'id' => $this->message->id,
                'body' => $this->message->body,
                'created_at' => $this->message->created_at->toISOString(),
                'user' => [
                    'id' => $this->message->user_id,
                    'name' => $this->message->user?->name ?? '',
                ]
            ]
        ];
    }
}

```


1. Import và class definition

```
use App\Models\Message;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class MessageCreated
{
    use Dispatchable, InteractsWithSockets, SerializesModels;
```

- **Dispatchable** : Cho phép event được dispatch bằng `MessageCreated::dispatch($message)` .
- **InteractsWithSockets** : Hỗ trợ khi dùng broadcasting, giúp tránh gửi lại event cho chính user đã tạo ra event đó (ngăn vòng lặp socket).
- **SerializesModels** : Tự động serialize/deserialize model (`Message`) khi event được đưa vào queue.

2. Constructor

```
public function __construct(public Message $message) {}
```

- Nhận một instance `Message` khi event được tạo ra.
- Ở đây chúng ta viết rút gọn: `public Message $message` thay vì khai báo property thủ công.

3. broadcastOn()

```
public function broadcastOn(): array
{
    return [
        new PrivateChannel('conversation.' . $this->message->conversation_id)
    ];
}
```

- Xác định **channel** mà event này sẽ được broadcast đến.
- Ở đây: `conversation.{conversation_id}` (ví dụ: `conversation.10`).
- `PrivateChannel` nghĩa là channel này **cần authorize** (xem ở `routes/channels.php`).
- Trả về array vì một event có thể phát lên nhiều channel.

4. broadcastAs()

```
public function broadcastAs(): string
{
    return 'MessageCreated';
}
```

- Đặt tên event khi broadcast ra ngoài.
- Trên frontend, client sẽ nghe sự kiện này với `listen('.MessageCreated', callback)`.

5. broadcastWith()

```
public function broadcastWith(): array
{
    return [
        'message' => [
            'id'      => $this->message->id,
            'body'     => $this->message->body,
            'created_at' => $this->message->created_at->toISOString(),
            'user'     => [
                'id'    => $this->message->user_id,
                'name'  => $this->message->user?->name ?? '',
            ],
        ],
    ];
}
```

- Xác định **payload** (data) được gửi ra channel.
- Ở đây, chúng ta đang phát một JSON có cấu trúc như ví dụ dưới:

```
{
  "message": {
    "id": 123,
    "body": "Hello world",
    "created_at": "2025-08-29T12:34:56Z",
    "user": {
      "id": 5,
      "name": "John"
    }
  }
}
```

- `user?->name ?? ''` : Dùng **nullsafe operator** để tránh lỗi nếu `$message->user` không tồn tại.

Luồng hoạt động tổng thể

1. User gửi tin nhắn → Controller tạo record `Message` .
2. Gọi `MessageCreated::dispatch($message)` .
3. Laravel broadcast event này qua **Reverb server** trên channel `conversation.{id}` .
4. Frontend (Echo) subscribe vào `conversation.{id}` :

```
Echo.private(`conversation.${conversationId}`).listen('.MessageCreated', (e) => {
    console.log(e.message);
});
```

5. Client nhận object JSON như trên >> hiển thị ngay tin nhắn realtime.

Tóm lại::

Class này định nghĩa rõ:

- **Event phát đi đâu** (`broadcastOn`)
- **Tên gì trên frontend** (`broadcastAs`)
- **Payload kèm theo** (`broadcastWith`)

Nhờ đó, client (Echo) có thể nhận đúng dữ liệu cần để hiển thị tin nhắn mới realtime.

3. Trigger Event

Khởi tạo Controller: `MessageController` sau khi lưu message ta buộc phải gọi:

```
MessageCreated::dispatch($message);
```

Laravel sẽ tự động broadcast qua **Reverb server**.

Chạy command line:

```
php artisan make:controller MessageController
```

`app\Http\Controllers\MessageController.php`

```

namespace App\Http\Controllers;

use App\Events\MessageCreated;
use App\Models\Conversation;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class MessageController extends Controller
{
    public function store(Request $request): RedirectResponse
    {
        $userId = $request->user()->id;
        $data = $request->validate([
            'conversation_id' => ['required', 'exists:conversations,id'],
            'body'            => ['nullable', 'string', 'max:5000'],
        ]);

        $conversation = Conversation::findOrFail($data['conversation_id']);
        abort_unless($conversation->isParticipant($userId), 403);

        $created = $conversation->messages()->create([
            'user_id' => $userId,
            'body'    => $data['body'] ?? '',
        ]);
        $created->load('user:id,name');
        MessageCreated::dispatch($created);
        $conversation->touch();

        return back();
    }
}

```

Phân tích MessageController@store

Mục đích:

- Nhận POST từ form chat
- Xác thực dữ liệu và quyền truy cập vào cuộc trò chuyện
- Ghi bản ghi messages
- Eager-load user để payload broadcast đầy đủ
- Phát MessageCreated (real-time)
- Cập nhật updated_at của conversations để sắp xếp theo hoạt động mới nhất
- Redirect về trang trước

Giải thích:

```
public function store(Request $request): RedirectResponse
{
    $userId = $request->user()->id;
```

- Lấy **ID user đang đăng nhập** từ guard mặc định. Mọi thao tác tiếp theo gắn chặt với user này.

```
$data = $request->validate([
    'conversation_id' => ['required', 'exists:conversations,id'],
    'body'            => ['nullable', 'string', 'max:5000'],
]);
```

- Trước khi lưu hay lấy data đều phải xác thực **Validation !!!**:
 - `conversation_id` bắt buộc và phải tồn tại trong bảng `conversations`.
 - `body` cho phép rỗng (nullable), là chuỗi, tối đa 5000 ký tự.
- Nếu sai rule → Laravel trả 422, dừng luồng.

```
$conversation = Conversation::findOrFail($data['conversation_id']);
```

- Tải **Conversation** theo khóa chính. Nếu không có >> 404.

```
abort_unless($conversation->isParticipant($userId), 403);
```

- **Authorization cấp business**: chỉ cho phép thành viên của conversation gửi tin.
- `isParticipant()` sẽ kiểm tra table trung gian `conversation_user` coi bên trong nó có (conversation_id, user_id) tương ứng không.
- Không phải thành viên >> 403.

```
$created = $conversation->messages()->create([
    'user_id' => $userId,
    'body'    => $data['body'] ?? '',
]);
```

- Tạo bản ghi messages qua **quan hệ** `Conversation::messages()`:
 - Gắn `user_id` người gửi
 - `body` cho phép rỗng.

```
$created->load('user:id,name');
```

- **Eager-load** quan hệ user (chỉ lấy id,name) để:
 - Giảm N+1 khi serialize
 - Đảm bảo payload của event có đủ thông tin người gửi (khớp với hàm broadcastWith() đã viết).

```
MessageCreated::dispatch($created);
```

- **Phát event domain.** Nếu event MessageCreated implements ShouldBroadcast :
 - Laravel sẽ broadcast tới channel (ví dụ PrivateChannel('conversation.{id}'))
 - Tên event bên client là .MessageCreated (do broadcastAs()), payload theo broadcastWith() .

```
$conversation->touch();
```

- **Cập nhật updated_at** của cuộc trò chuyện để danh sách hội thoại sắp xếp theo hoạt động mới nhất (nằm bên thanh sidebar chat bên trái).

```
return to_route('chat.show', $conversation)->setStatusCode(303);
```

303 buộc trình duyệt thực hiện **GET** sau POST.

5. Khởi động Reverb Server cùng với lệnh composer run dev:

Thêm lệnh này:

```
php artisan reverb:start
```

Vào composer.json

```
"dev": [  
    "Composer\\Config::disableProcessTimeout",  
    "npx concurrently -c \"#93c5fd,#c4b5fd,#fdb474\" \"php artisan serve\" \"php arti:  
    ],
```

>> Mặc định mở WebSocket server ở cổng 8080.

Bây giờ client sẽ connect và nhận event.

6. Thêm ROUTES:

```
<?php

use App\Http\Controllers\ConversationController;
use App\Http\Controllers\MessageController;
use Illuminate\Support\Facades\Route;

Route::middleware("auth")->group(function () {
    // tải về danh sách conversations
    Route::get('/chat', [ConversationController::class, 'index'])
        ->name('chat.index');

    // xem chi tiết 1 conversation
    Route::get(
        '/chat/{conversation}',
        [ConversationController::class, 'show']
    )->name('chat.show');

    // tạo mới 1 conversation
    Route::post('/chat', [ConversationController::class, 'store'])
        ->name('chat.store');

    // gửi 1 message
    Route::post('/messages', [MessageController::class, 'store'])
        ->name('messages.store');
});
```

Sau đó chạy server như bình thường, để ý sẽ thấy nó xuất hiện thêm 1 port mới của reverb:

```
PS C:\dez\php-k24wd02\laravel\laravel-app-03> composer run dev
> Composer\Config::disableProcessTimeout
> npx concurrently -c "#93c5fd,#c4b5fd,#fdba74" "php artisan serve" "php artisan queue:listen --tries=1"
  "npm run dev" --names='server,queue,vite' "php artisan reverb:start"
[vite]
[vite] > dev
[vite] > vite
[vite]
[queue]
[queue] INFO Processing jobs from the [default] queue.
[queue]
[server]
[server] INFO Server running on [http://127.0.0.1:8000].
[server]
[server] Press Ctrl+C to stop the server
[server]
[3]
[3] INFO Starting server on 0.0.0.0:8080 (127.0.0.1).
[3]
[vite] 9:43:02 AM [vite] (client) info: Types generated for actions, routes, form variants
[vite] Plugin: @laravel/vite-plugin-wayfinder
[vite]
[vite] VITE v7.1.2 ready in 2955 ms
[vite]
[vite] → Local: http://localhost:5173/
[vite] → Network: use --host to expose
[vite]
[vite] LARAVEL v12.25.0 plugin v2.0.0
[vite]
[vite] → APP_URL: http://127.0.0.1:8000
```


Hướng dẫn fix một số lỗi phổ biến:

Trong quá trình chạy nếu bị lỗi như ảnh:

```
[plugin:vite:react-babel] C:\dez\php-k24wd02\laravel\laravel-app-03\resources\js\components\chat\MessageInput.tsx: Invalid shorthand property initializer. (8:24)
  11 |
    C:/dez/php-k24wd02/laravel/laravel-app-03/resources/js/components/chat/MessageInput.tsx:8:24
   6 |         const inputRef = useRef(null);
   7 |         const { data, setData, post, processing, reset, clearErrors }
   8 |             conversation_Id = conversationId,
      |                               ^
   9 |         body: '',
  10 |     });
```

Thì mở file:

resources\js\components\chat\MessageInput.tsx

Chỉnh lại khoảng dòng code thứ 8 hoặc 9 tùy thông báo lỗi:

- dấu = thành dấu :
- conversation_Id key thành conversation_id

```
8 |         const { data, setData, post, processing,
9 |             conversation_id: conversationId,
10 |         body: '',
```

Nếu gặp lỗi như ảnh:

```
✖ Uncaught ReferenceError: useRef is not defined
  at MessageInput (MessageInput.tsx:6:22)
  Show ignore-listed frames
```

thì vào file MessageInput.tsx và import useRef vào, bấm Ctrl+Spacebar :

```
it default function MessageInput({ conversationId }) {
const inputRef = useRef(null);
const { data, setData, useRef
conversation_Id: co useRefSnippet
useRef
```

```
import { useRef } from 'react';

export default function MessageInput() {
  const inputRef = useRef(null);
}
```

Sau đó qua bên `resources\js\pages\Chat\Show.tsx` chỉnh lại đoạn dependencies như sau:

```
resources > js > pages > Chat > Show.tsx > Show
1 // @ts-nocheck
2 import ChatLayout from '@component
3 import ConversationList from '@cor
4 import MessageInput from '@compon
5 import MessageList from '@compon
6 import AppLayout from '@layouts/a
7 import { useEffect, useState } fro
8
9 export default function Show({ conv
10   const [items, setItems] = useSt
11
12   useEffect(() => {
13     setItems(messages.data);
14   }, [conversation.id]);
15
```

Nếu gặp lỗi như ảnh:

```
✖ Uncaught ReferenceError: queryParams is not defined
    at dashboard.url (index.ts:91:33)
    at dashboard (index.ts:77:20)
    at Welcome (welcome.tsx:19:39)
    Show ignore-listed frames
```

Chạy lại server sẽ hết

Lỗi bấm chọn 1 người không tạo Conversation Direct Message (chat 1 - 1) được:

```
01.Fix_ERROR_REVERB.md U ConversationController.php M X
app > Http > Controllers > ConversationController.php > ConversationController > store
12 class ConversationController extends Controller
57 {
58
59     public function store(Request $request): RedirectResponse
60     {
61         $userId = $request->user()->id;
62
63         $data = $request->validate([
64             'user_ids' => ['required', 'array', 'min:1'],
65             'name' => ['nullable', 'string', 'max:255'],
66             'user_ids.*' => ['integer', 'exists:users,id', 'different:' . $userId],
67         ]);
68
69         $isDirect = count($data['user_ids']) === 1;
70         $conversation = Conversation::create([
71             'is_direct' => $isDirect,
72             'name' => $isDirect ? null : ($data['name'] ?? null),
73             'created_by' => $userId
74         ]);
75         $conversation->users()->sync(array_unique([...$data['user_ids'], $userId]));
76         return redirect()->route('chat.show', $conversation);
77     }
78 }
79
```

app\Http\Controllers\ConversationController.php

sửa validation của method store bên ConversationController từ required thành nullable

Lỗi Direct message nhưng lại hiển thị Group + id (Group #\${c.id}):

resources\js\components\chat\ConversationList.tsx

chỉnh lại câu lệnh điều kiện như bên dưới.

```
<div className="conv-title">{c.name} || (c.is_direct ? 'Direct message' : `Group #${c.id}`)
```

Thêm class name để có css phù hợp:

```

import { Link, usePage } from '@inertiajs/react';

export default function ConversationList({ conversations = [] }) {
  const { url } = usePage();
  if (!conversations.length) return <div>No Conversation Found.</div>;

  return (
    <ul className="conv-list">
      {conversations.map((c) => {
        const active = url === `/chat/${c.id}`;

        return (
          <li key={c.id} className="conv-item">
            <Link href={`/${c.id}`} className={`conv-link ${active ? 'active' : ''}`>
              <div className="conv-title">{c.name} || (c.is_direct ? 'Direct message' : '')</div>
              <div className="conv-sub">
                {c.last_message.user.name}
                <span>: </span>
                {c.last_message.body.slice(0, 40) + '...'}
              </div>
            </Link>
          </li>
        );
      })}
    </ul>
  );
}

```

Hướng dẫn test Chat App:

Bước 1: Tạo Conversation:

Private Chat (isDirect = true/1)

New Conversation

☒

Anjali Spinka

lavinia.lockman@example.net

☐

Carolanne Tillman

roob.cloyd@example.org

☐

Domenick Cartwright

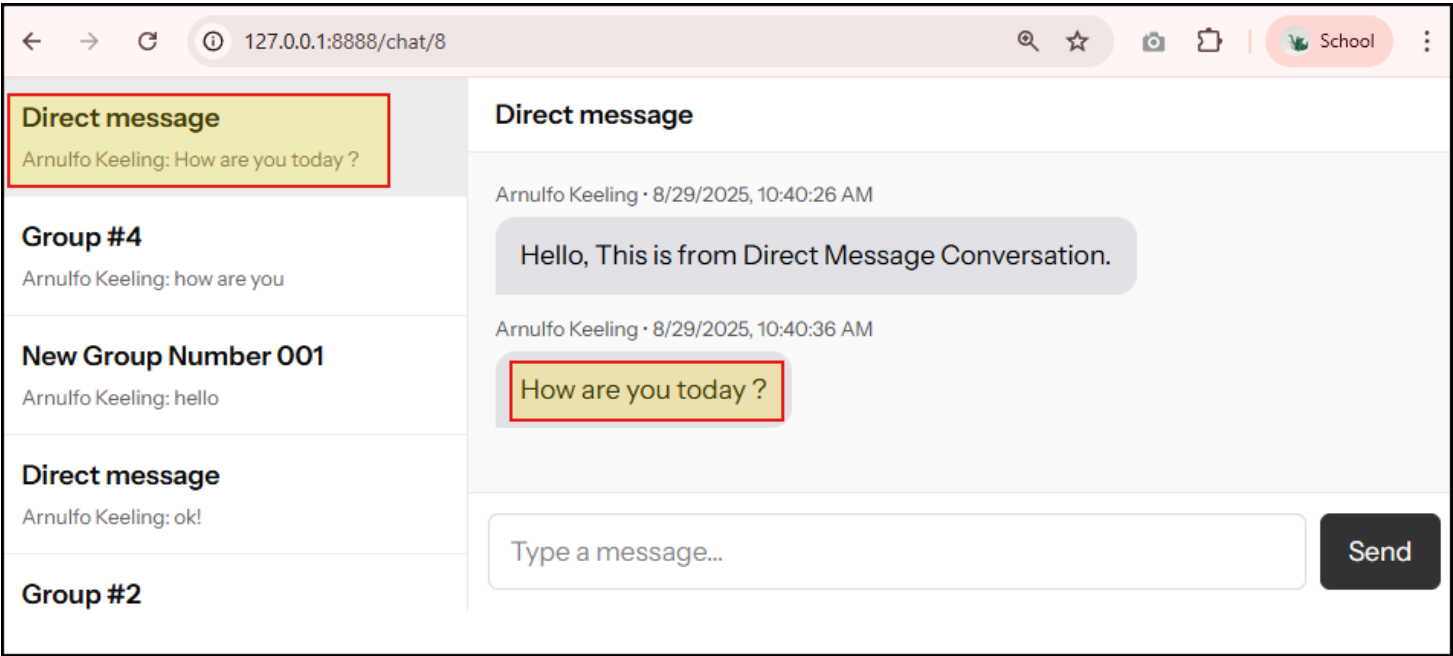
xnitzsche@example.org

☐

Jeffry Strosin

Create

Chỉ chọn 1 person sau đó bấm Create .



Quan sát thấy đoạn trích dẫn 40 ký tự bên dưới khung Conversation bên trái

Group Chat (isDirect = false/0)

New Conversation

My new group chat!

☒

Anjali Spinka

lavinia.lockman@example.net

☒

Carolanne Tillman

roob.cloyd@example.org

☐

Domenick Cartwright

xnitzsche@example.org

☐

Jeffry Strosin

Create

Để tạo group yêu cầu phải chọn ít nhất 2 persons:

←

→

↺

127.0.0.1:8888/chat/9

🔍

☆

📷

📁

School

⋮

My new group chat!

Arnulfo Keeling: Welcome !!!

Direct message

Arnulfo Keeling: How are you today ?

Group #4

Arnulfo Keeling: how are you

New Group Number 001

Arnulfo Keeling: hello

Direct message

Arnulfo Keeling: ok!

Group #2

Arnulfo Keeling: Officia quis occaecati
aperiam odit assu

voluptas numquam quos

Juvenal Jones: Voluptatem doloremque
provident aut reru

My new group chat!

Arnulfo Keeling · 8/29/2025, 10:43:46 AM

Hello, everybody this is our Conversation Group !!!

Arnulfo Keeling · 8/29/2025, 10:43:57 AM

Welcome !!!

Type a message...

Send

Vậy là xong !!!

Trong trường hợp vẫn bị lỗi có thể tham khảo source đi chung với bài hướng dẫn này!