

****Must follow the number 1 documentation to setup the project**

Private Channel

Private channel works with the authenticated users.

From **channels.php**,

```
Broadcast::channel('users.{id}', function (User $user, $id) {  
    return (int) $user->id === (int) $id;  
});
```

Now checking if its working, from **dashboard.php**,

```
<div class="py-12">  
    <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">  
        <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">  
            <div class="p-6 text-gray-900" x-init="privateClass()" x-  
data="{dispatched: false}">  
                {{ __("You're logged in!") }}  
            </div>  
        </div>  
    </div>  
</div>  
  
<script>  
    function privateClass(){  
        Echo.private('users.{{ auth()->id() }}')  
    }  
</script>
```

- Now **x-init** e function add kora hoise jate niche **<script>** block e easily code lekha jay.
- **auth()->id()** diye authenticated specific user ke bojhan hoise
- **x-data="{dispatched: false}"** is a part of alpine.js

Testing if its working, just refresh the web page and view the network tab

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/d...`. The page content includes a logo, a "Dashboard" heading, and a message "You're logged in!". Below the page content, the Chrome DevTools Network tab is open, showing a list of requests. The request `auth` is selected, and its details are displayed in the right-hand pane.

Network Tab Details:

- Request URL:** `http://127.0.0.1:8000/broadcasting/auth`
- Request Method:** `POST`
- Status Code:** `200 OK`
- Remote Address:** `127.0.0.1:8000`
- Referrer Policy:** `strict-origin-when-cross-origin`
- Response Headers:**
 - Cache-Control:** `no-cache, private`
 - Connection:** `close`
 - Content-Type:** `application/json`
 - Date:** `Thu, 31 Oct 2024 04:33:04 GMT`
 - Host:** `127.0.0.1:8000`
 - Set-Cookie:** `laravel_session=eyJpdil6lIZrdUJ0T...`

- Ekhane **auth** er request tar Status Code : 200 OK, cause authenticated user is connected

Step 2: Creating an event:

php artisan make:event OrderDispatched

Full **OrderDispatched.php** code,

```
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Contracts\Broadcasting\ShouldBroadcastNow;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;
use App\Models\User;

class OrderDispatched implements ShouldBroadcastNow
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public function __construct(public User $user)
    {
        //
    }

    public function broadcastOn(): array
    {
        return [
            new PrivateChannel('users.'.$this->user->id),
        ];
    }
}
```

Add routes in **web.php**,

```
Route::get('/broadcast', function () {
    broadcast(new \App\Events\OrderDispatched(User::find(1)));
});
```

Now from **dashboard.php**,

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Dashboard') }}
    </h2>
  </x-slot>

  <div class="py-12">
    <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
      <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
        <div class="p-6 text-gray-900" x-init="privateClass()"
>
          {{ __("You're logged in!") }}
        </div>
      </div>
    </div>
  </div>

  <script>
    function privateClass(){
      Echo.private('users.{{ auth()->id() }}').listen('OrderDispatched',
(event)=>{
        console.log(event)
      })
    }
  </script>
</x-app-layout>
```

- After hitting the route **/broadcast** from route, this output will occur

```
▼ {user: {...}} ⓘ
  ▼ user:
    created_at: "2024-10-31T04:19:22.000000Z"
    email: "tahmid.tf1@gmail.com"
    email_verified_at: null
    id: 1
    name: "Tahmid Ferdous"
    updated_at: "2024-10-31T04:19:22.000000Z"
    ► [[Prototype]]: Object
    ► [[Prototype]]: Object
  >
```

Showing Data based on dispatch

Step 1: Creating an alpine store,

```
{{!-- Alpine js store script --}}  
  
<script>  
  document.addEventListener('alpine:init', () => {  
    Alpine.store('dispatchedStore', {dispatched: false});  
  });  
</script>
```

Step 2: triggering alpine.store variables,

```
Alpine.store('dispatchedStore').dispatched = true;
```

Here is a script code of the function,

```
<script>  
  function privateClass() {  
    Echo.private('users.{{ auth()->id() }}').listen('OrderDispatched', (event) =>  
    {  
      Alpine.store('dispatchedStore').dispatched = true;  
    })  
  }  
</script>
```

Step 3: Using it as a property, **x-if** is also a part of alpine js,

```
<template x-if="$store.dispatchedStore.dispatched">  
  <div>  
    Order has been dispatched  
  </div>  
</template>
```

Now the full updated code is here,

```
<div class="py-12">  
  <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">  
    <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">  
      <div class="p-6 text-gray-900" x-init="privateClass()">  
        <template x-if="$store.dispatchedStore.dispatched">  
          <div>  
            Order has been dispatched  
          </div>  
        </template>  
      </div>  
    </div>  
  </div>  
</div>
```

```
<script>
  function privateClass() {
    Echo.private('users.{{ auth()->id() }}').listen('OrderDispatched', (event) =>
  {
    Alpine.store('dispatchedStore').dispatched = true;
  })
}
</script>

{{-- Alpine js store script --}}

<script>
  document.addEventListener('alpine:init', () => {
    Alpine.store('dispatchedStore', {dispatched: false});
  });
</script>
```