To Subscribe To Channel From Client Side:

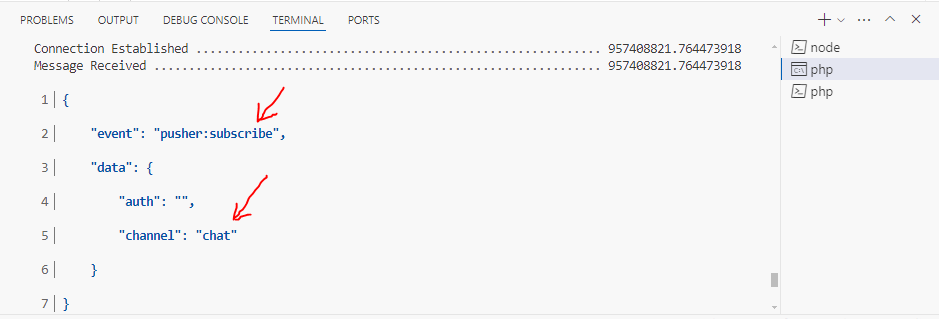
x-init="

console.log('Test Ok !!!');

      Echo.channel('chat');

"

Then Inside The Reverb Console:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x-init="

console.log('Test Ok !!!');

      Echo.channel('chat')

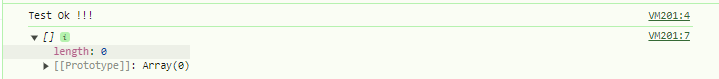
      .listen('Example\\ExampleEvent', (event)=> {

              console.log(event);

           });

">





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Note 1: We Create ExampleEvent Inside The Example Folder, so the Event Name is:

* **Name is**: **Example\\ExampleEvent**

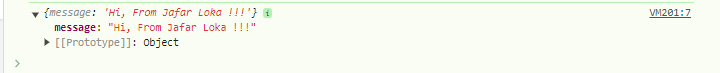
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

After Creating New Variable For Our Event Class:

* Declaration is: public string $message = 'Hi, From Jafar Loka'

public string $message = 'Hi, From Jafar Loka !!!';

Then We Broadcast Event Then The Console Will Be:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

After We Set User As Attribute For ExampleEvent We Do:

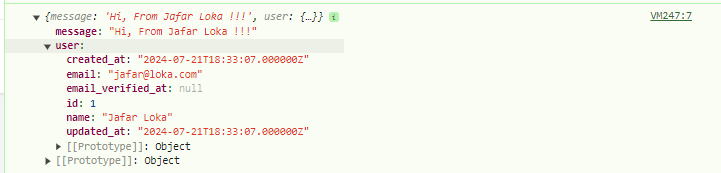
public function \_\_construct(User $user)

{

        $this->user = $user;

}

Broadcast Again, and The Result Will Be:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Note 1**: If We set User property As Protected Then It Will Not Be Broadcasted Any More.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Control What We Broadcast we define broadcastWith-method and return array:

public function broadcastWith(): array {

        return [

        ];

    }

Broadcast Event Again, Even user is public:



After We Set The Valid Parameters:

public function broadcastWith(): array {

        return [

            'email' => $this->user->email,

            'name'  => $this->user->name,

        ];

    }



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

With More Organization:

public function broadcastWith(): array {

        return [

            'user' => [

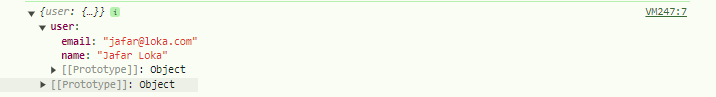
                'email' => $this->user->email,

                'name'  => $this->user->name,

            ],

        ];

    }



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For More Security, we define: protected User $user;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Set Our Event Class Inside Another Folder, for Listening We Set dot (.) Before The Full Path Of Event Name:

x-init="

console.log('Test Ok !!!');

Echo.channel('chat')

      .listen('Example\\ExampleEvent', (event)=> {

            console.log(event);

            }).listen('.App\\Events\\Chat\\MessageEvent', (event) => {

                   console.log(event);

     });

">

Route::get('/broadcast-example-event', function(){

    broadcast(new ExampleEvent(User::find(1), Message::find(1)));

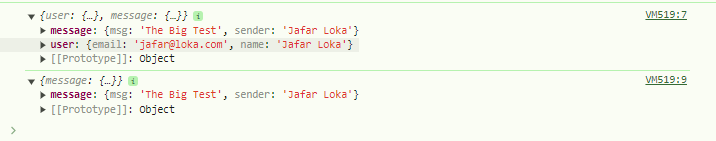
    broadcast(new MessageEvent(Message::find(1)));

    return "Event is Broadcasted Successfully....";

});

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

After We broadcast The Events:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Note**: Inside The Echo Configuration, We Can Define **namespace-property** For All Namespaces In Our Application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Set ShouldBroadcast-interface as The Implements-Interface, Then We Must Handle Queues:

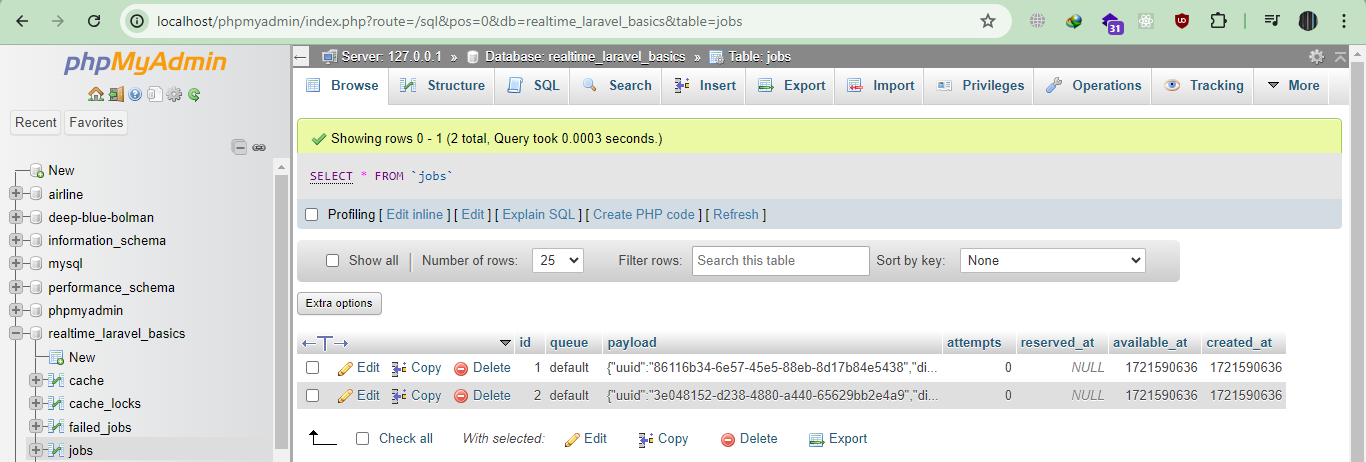
class MessageEvent implements ShouldBroadcast

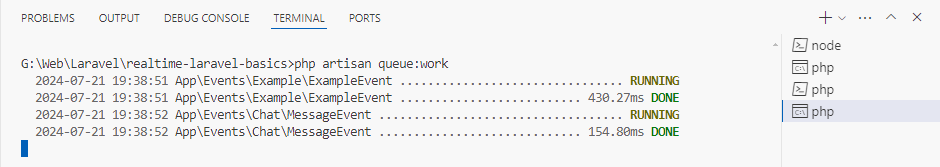
{...}

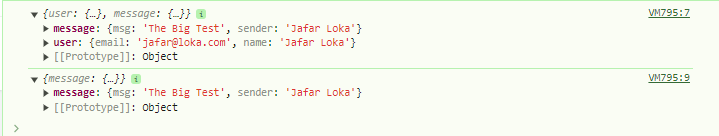
class ExampleEvent implements ShouldBroadcast

{...}

After We broadcast The Events The Jobs-Table Will be (We Don’t Run The Queue Now):







\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class MessageEvent implements ShouldBroadcast

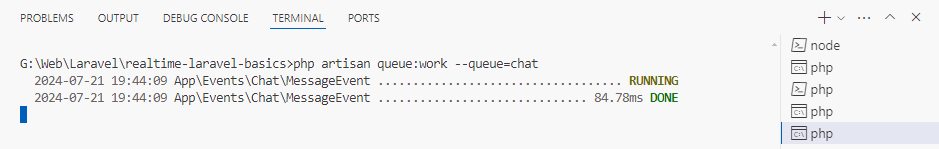
{

    use Dispatchable, InteractsWithSockets, SerializesModels;

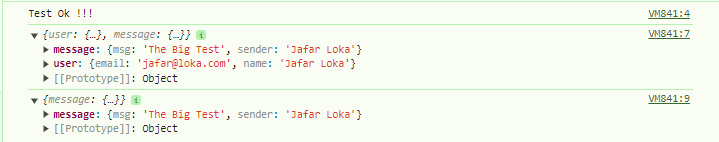
    public string $queue = 'chat';

...

}



After We Broadcast The Event, The Result will be:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

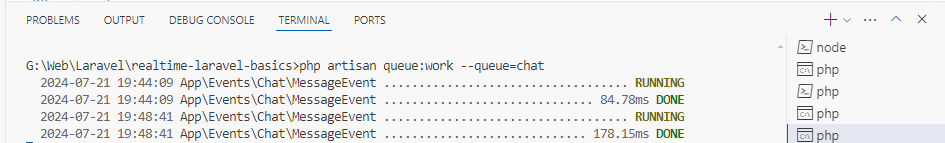
To Define Our Queue Name, In Functional Way We Can use broadcastQueue()-method:

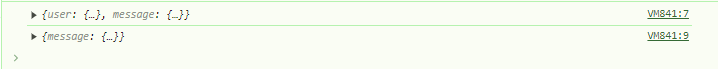
public function broadcastQueue() {

        return 'chat';

    }

Then Broadcast Again, And The Result Will Be:





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create New Private Channel For Our Users, With id-as-parameter:

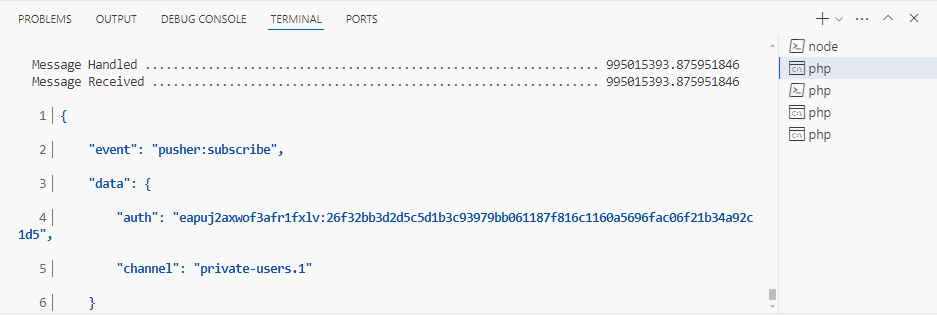
Broadcast::channel('users.{id}', function (User $user, int $id) {

    return (int) $user->id === (int) $id;

});

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

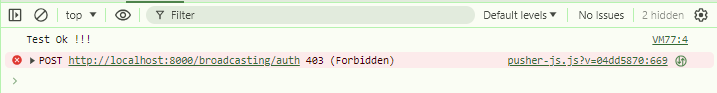
Then On Client Side, We Use: **Echo.private('users.1'),** Where 1 is Our Id:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Note**: If We Change The Id (here is: 1), Then The Result **403 Forbidden**

Echo.private('users.2');



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Note (From Me):** Private Channels Not Worked With ShouldBroadcast And With Queues.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

When We Want To Join The Room, The Channel Of Broadcast-class Must Return The Data That We Want To Display:

Broadcast::channel('room.{roomId}', function(User $user, int $roomId) {

    return $user->only('id', 'name');

});

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Event is: **member\_added**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Get List Of Here Users We Can Use here-function:

<div

class="p-6 text-gray-900"

    x-data="{

                        usersHere: []

    }"

   x-init="

                        Echo.join('room.{{ $room->id }}')

                            .here((users) => {

                                console.log(users);

                                usersHere= users;

                            });

                    ">

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Note: We Can Check If User Exists Before push To The Array, in the joining:

<div

class="p-6 text-gray-900"

      x-data="{

                        usersHere: []

                    }"

      x-init="

      Echo.join('room.{{ $room->id }}').here((users) => {

      console.log(users);

            usersHere= users;

      }).joining(user => { usersHere.push(user); })

.leaving((user) => {

usersHere = usersHere.filter(el => el.id != user.id);

      });

">

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Send Event From User To All User Except The Sender, We Can Use whisper-method:

x-init="

                            const channel = Echo.private('app');

                            setTimeout(()=> {

                                channel.whisper('typing', {

                                    id: 1,

                                });

                            }, 2000);

                        "

                    >

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*