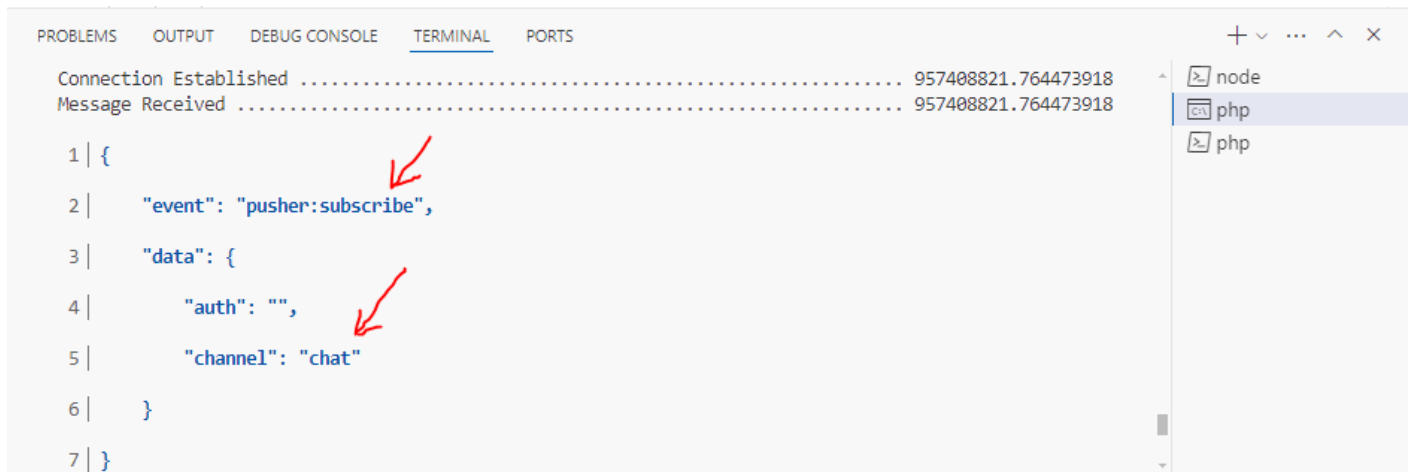


To Subscribe To Channel From Client Side:

```
x-init="
    console.log('Test Ok !!!');
    Echo.channel('chat');
"
```

Then Inside The Reverb Console:



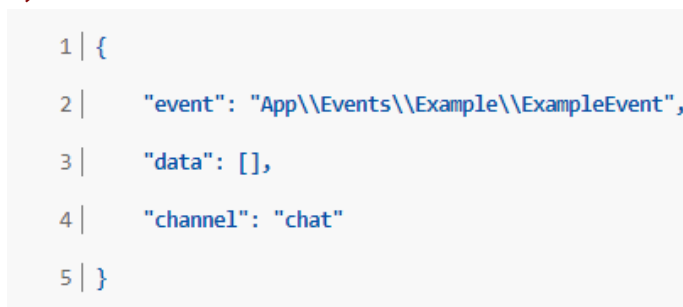
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Connection Established ..... 957408821.764473918
Message Received ..... 957408821.764473918

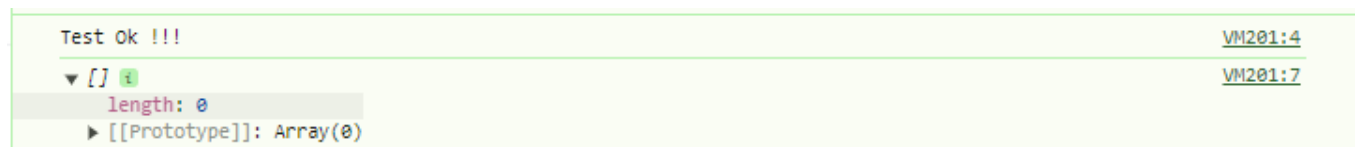
1 | {
2 |   "event": "pusher:subscribe",
3 |   "data": {
4 |     "auth": "",
5 |     "channel": "chat"
6 |   }
7 | }
```

\*\*\*\*\*

```
x-init="
    console.log('Test Ok !!!');
    Echo.channel('chat')
        .listen('Example\\ExampleEvent', (event)=> {
            console.log(event);
        });
">
```



```
1 | {
2 |   "event": "App\\Events\\Example\\ExampleEvent",
3 |   "data": [],
4 |   "channel": "chat"
5 | }
```



```
Test Ok !!!
▼ [] i
  length: 0
  ► [[Prototype]]: Array(0)
```

\*\*\*\*\*

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:

```
▼ {message: 'Hi, From Jafar Loka !!!'} ⓘ VM201:7
  message: "Hi, From Jafar Loka !!!"
  ▶ [[Prototype]]: Object
>
```

\*\*\*\*\*

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:

```
▼ {message: 'Hi, From Jafar Loka !!!', user: {...}} ⓘ VM247:7
  message: "Hi, From Jafar Loka !!!"
  user:
    created_at: "2024-07-21T18:33:07.000000Z"
    email: "jafar@loka.com"
    email_verified_at: null
    id: 1
    name: "Jafar Loka"
    updated_at: "2024-07-21T18:33:07.000000Z"
    ▶ [[Prototype]]: Object
    ▶ [[Prototype]]: Object
```

\*\*\*\*\*

**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:

```
▶ [] VM247:7  
>
```

After We Set The Valid Parameters:

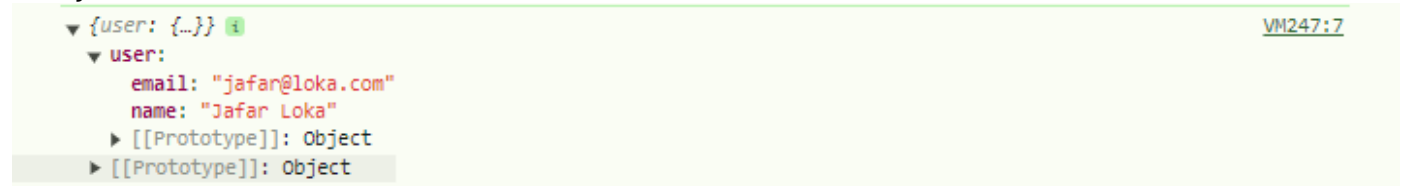
```
public function broadcastWith(): array {  
    return [  
        'email' => $this->user->email,  
        'name'  => $this->user->name,  
    ];  
}
```

```
▼ {email: 'jafar@loka.com', name: 'Jafar Loka'} i VM247:7  
  email: "jafar@loka.com"  
  name: "Jafar Loka"  
  ▶ [[Prototype]]: Object  
>
```

\*\*\*\*\*

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:

```
▼ {user: {...}, message: {...}} ⓘ VM519:7
  ► message: {msg: 'The Big Test', sender: 'Jafar Loka'}
  ► user: {email: 'jafar@loka.com', name: 'Jafar Loka'}
  ► [[Prototype]]: Object
▼ {message: {...}} ⓘ VM519:9
  ► message: {msg: 'The Big Test', sender: 'Jafar Loka'}
  ► [[Prototype]]: Object
>
```

\*\*\*\*\*

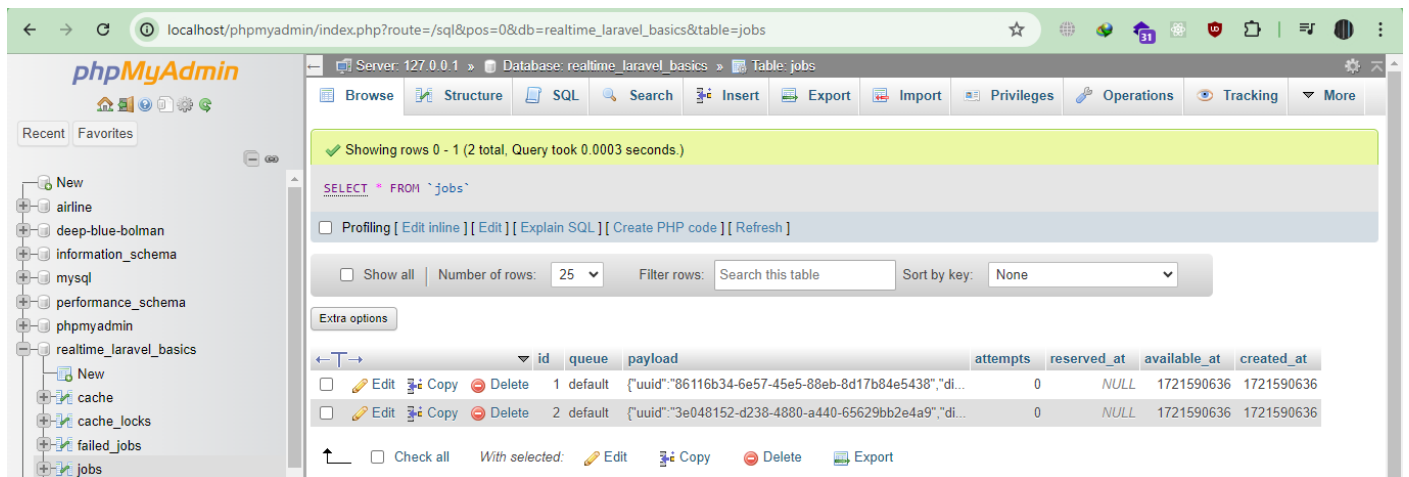
**Note:** Inside The Echo Configuration, We Can Define [namespace-property](#) For All Namespaces In Our Application.

\*\*\*\*\*

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

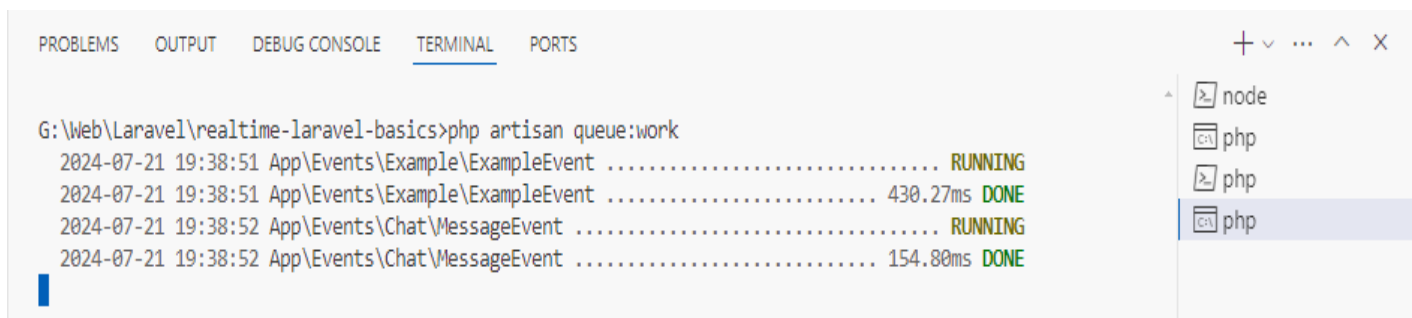
```
class MessageEvent implements ShouldBroadcast
{...}
class ExampleEvent implements ShouldBroadcast
{...}
```

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

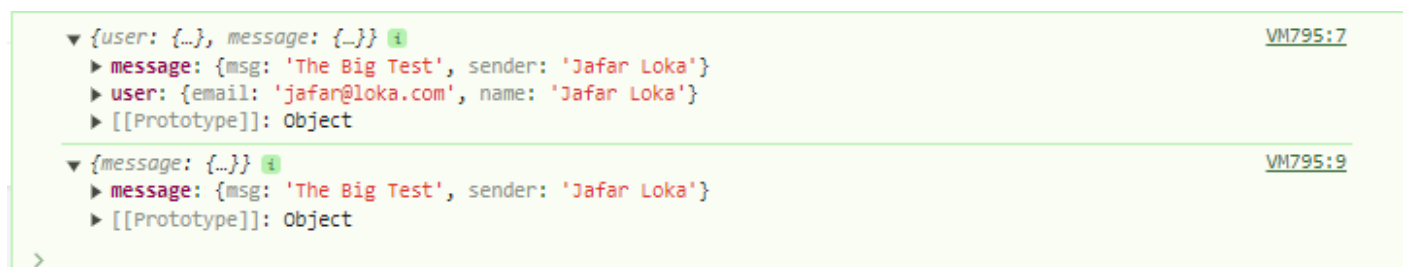


The screenshot shows the phpMyAdmin interface for the 'realtime\_laravel\_basics' database. The 'jobs' table is selected, and the SQL query 'SELECT \* FROM `jobs`' is executed. The table contains two rows of job data.

	id	queue	payload	attempts	reserved_at	available_at	created_at
<input type="checkbox"/>	1	default	{"uuid":"86116b34-6e57-45e5-88eb-8d17b84e5438","di...	0	NULL	1721590636	1721590636
<input type="checkbox"/>	2	default	{"uuid":"3e048152-d388-4440-65629bb2e4a9","di...	0	NULL	1721590636	1721590636



```
G:\Web\Laravel\realtime-laravel-basics>php artisan queue:work
2024-07-21 19:38:51 App\Events\Example\ExampleEvent ..... RUNNING
2024-07-21 19:38:51 App\Events\Example\ExampleEvent ..... 430.27ms DONE
2024-07-21 19:38:52 App\Events\Chat\MessageEvent ..... RUNNING
2024-07-21 19:38:52 App\Events\Chat\MessageEvent ..... 154.80ms DONE
```



```
> {user: {...}, message: {...}} VM795:7
  > message: {msg: 'The Big Test', sender: 'Jafar Loka'}
  > user: {email: 'jafar@loka.com', name: 'Jafar Loka'}
  > [[Prototype]]: Object

> {message: {...}} VM795:9
  > message: {msg: 'The Big Test', sender: 'Jafar Loka'}
  > [[Prototype]]: Object
```

\*\*\*\*\*

```

class MessageEvent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

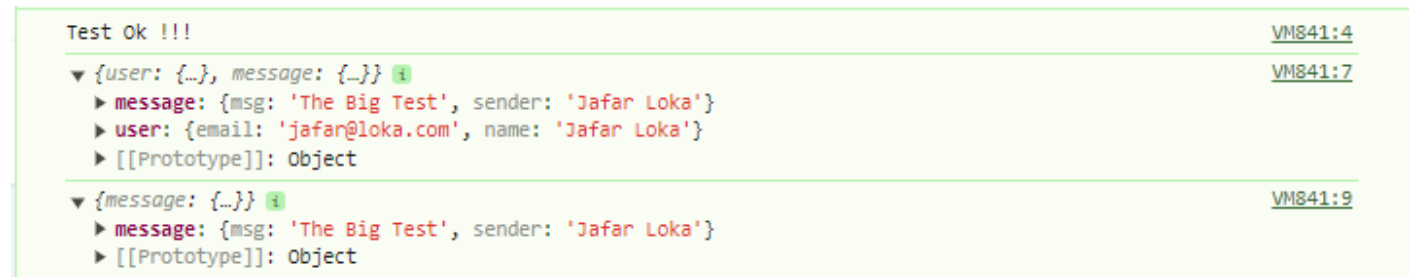
    public string $queue = 'chat';

    ...
}

```



After We Broadcast The Event, The Result will be:



\*\*\*\*\*

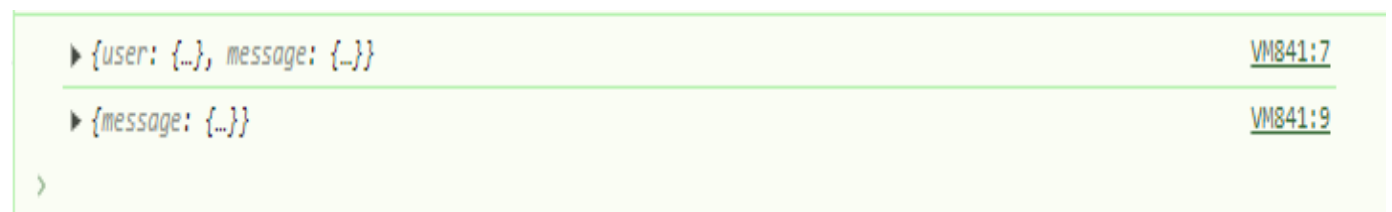
To Define Our Queue Name, In Functional Way We Can use broadcastQueue()-method:

```

public function broadcastQueue() {
    return 'chat';
}

```

Then Broadcast Again, And The Result Will Be:



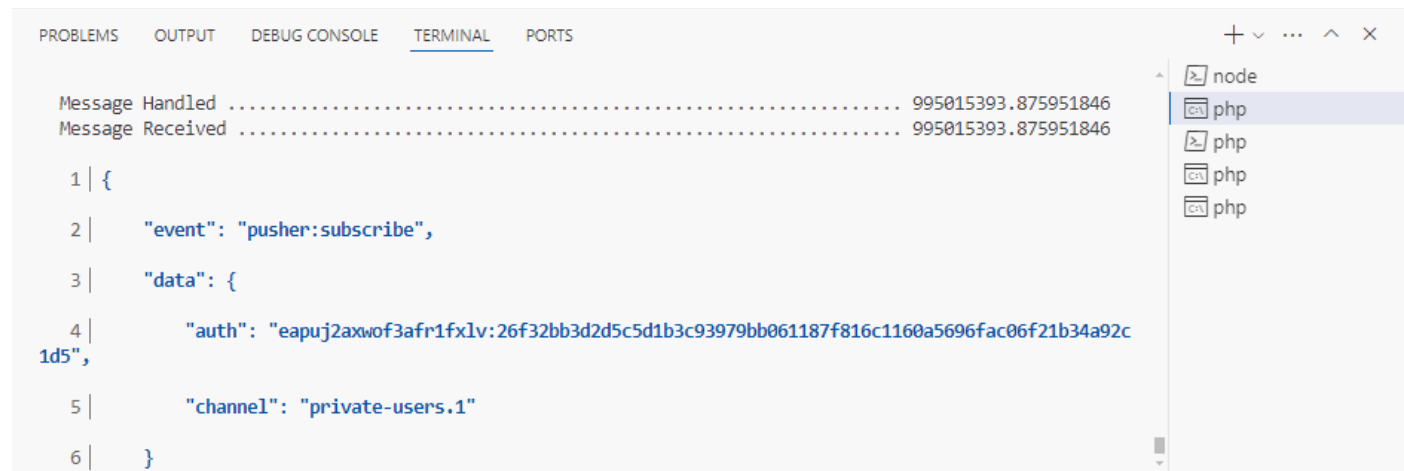
\*\*\*\*\*

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:



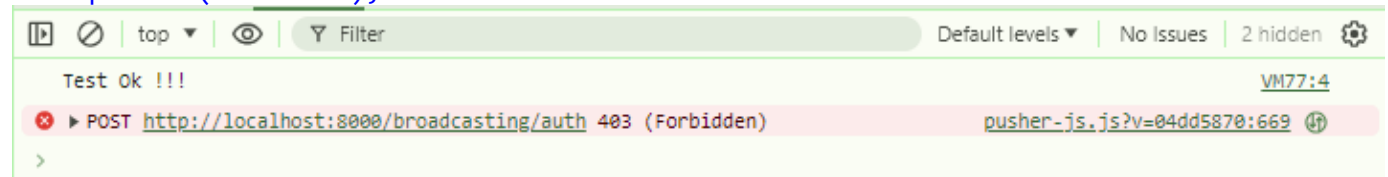
The screenshot shows a VS Code interface with a terminal window open. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing a message received from a Pusher channel. The message is a JSON object with the following structure:

```
1 | {  
2 |   "event": "pusher:subscribe",  
3 |   "data": {  
4 |     "auth": "eapuj2axwof3afr1fxlv:26f32bb3d2d5c5d1b3c93979bb061187f816c1160a5696fac06f21b34a92c1d5",  
5 |     "channel": "private-users.1"  
6 |   }  
7 | }
```

\*\*\*\*\*

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

```
Echo.private('users.2');
```



The screenshot shows a web browser's developer console with a 403 Forbidden error. The error message is:

```
Test Ok !!!  
✖ POST http://localhost:8000/broadcasting/auth 403 (Forbidden) pusher-js.js?v=04dd5870:669
```

\*\*\*\*\*

**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**

```
1 | {  
2 |   "event": "pusher_internal:member_added",  
3 |   "data": {  
4 |     "user_id": "1",  
5 |     "user_info": {  
6 |       "id": 1,  
7 |       "name": "Jafar Loka"  
8 |     }  
9 |   },  
10 |   "channel": "prese...
```

\*\*\*\*\*

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);
"
>
*****
```