

# Laravel - Sending Email

Topics : [Laravel](#)

Written on [December 25, 2023](#)

In Laravel, sending emails is a common task, and Laravel provides a clean and simple API for this purpose. Laravel uses the SwiftMailer library to send emails. Here's a basic guide on how to send emails in Laravel:

## 1. Configuration:

Make sure that your application's mail configuration is set up correctly in the `config/mail.php` file. You may need to configure the `MAIL_DRIVER`, `MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, and `MAIL_PASSWORD` variables in your `.env` file.

## 2. Creating Mailables:

Mailables in Laravel are classes that represent emails. You can generate a new mailable using the `make:mail` Artisan command:

```
php artisan make:mail OrderShipped
```

This will create a new mailable class in the `app/Mail` directory.

## 3. Configuring Mailables:

Edit the generated mailable class to define the email's subject, view, and any data that should be passed to the view.

```
// app/Mail/OrderShipped.php

public function build()
{
    return $this->view('emails.orders.shipped');
}
```

## 4. Creating Email Views:

Create a Blade view for your email. By default, email views are stored in the `resources/views/emails` directory.

```
<!-- resources/views/emails/orders/shipped.blade.php -->
```

<p>Your order has been shipped!</p>

## 5. Sending Emails:

In your controller or wherever you need to send an email, create a new instance of your mailable and call the send method:

```
use App\Mail\OrderShipped;
use Illuminate\Support\Facades\Mail;

public function sendEmail()
{
    $orderShipped = new OrderShipped();

    Mail::to('recipient@example.com')->send($orderShipped);

    return 'Email sent successfully!';
}
```

## 6. Passing Data to Mailables:

You can pass data to your mailable for use in the email view by defining public properties or using the with method.

```
// app/Mail/OrderShipped.php

public $order;

public function __construct(Order $order)
{
    $this->order = $order;
}

public function build()
{
    return $this->view('emails.orders.shipped');
}
```

## 7. Inline Attachments and Attachments:

You can attach files to your emails using the attach and attachData methods. For inline attachments, use the embed method.

```
public function build()
{
    return $this->view('emails.orders.shipped')
        ->attach('/path/to/file')
        ->embed('/path/to/image');
}
```

## 8. Markdown Mailables:

Laravel also supports Markdown for emails. You can generate a Markdown mailable using the `make:mail` Artisan command with the `--markdown` option.

```
php artisan make:mail OrderShipped --markdown=emails.orders.shipped
```

## 9. Queueing Emails:

To send emails asynchronously, you can use Laravel's queue system.

```
Mail::to('recipient@example.com')->queue(new OrderShipped());
```

## 10. Mail Logging:

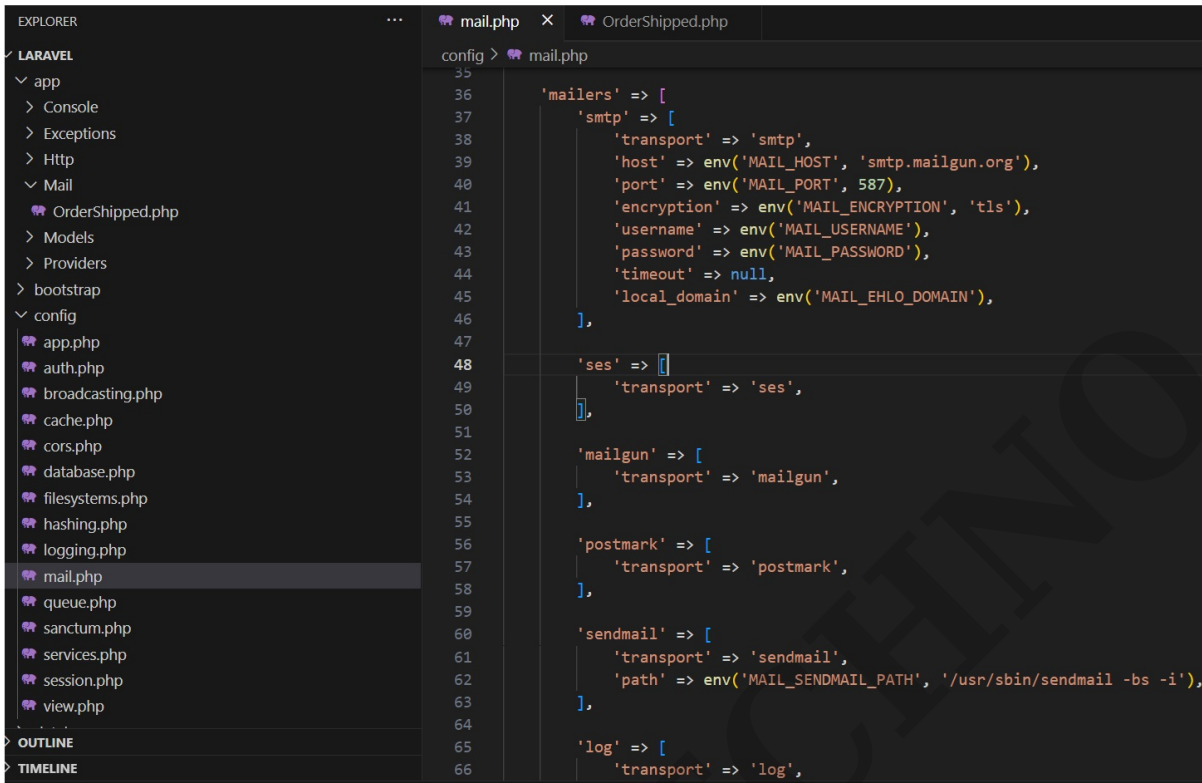
In your local development environment, you might want to log emails instead of sending them. Update your `.env` file:

```
MAIL_MAILER=log
```

This will log all sent emails to the `storage/logs` directory.

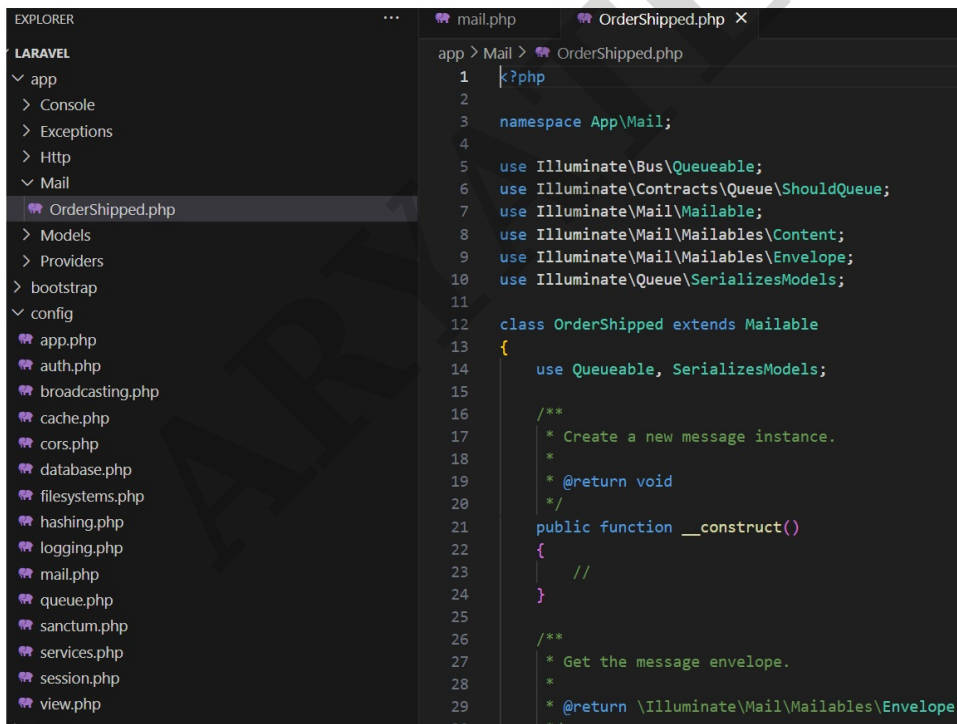
```
D:\xampp\htdocs\laravel>php artisan make:mail OrderShipped
```

```
INFO Mailable [D:\xampp\htdocs\laravel\app\Mail\OrderShipped.php] created successfully.
```



The screenshot shows the VS Code interface with the Explorer pane on the left showing the project structure. The main editor displays the contents of `config/mail.php`. The file defines several mailers and their configurations:

```
35
36 'mailers' => [
37     'smtp' => [
38         'transport' => 'smtp',
39         'host' => env('MAIL_HOST', 'smtp.mailgun.org'),
40         'port' => env('MAIL_PORT', 587),
41         'encryption' => env('MAIL_ENCRYPTION', 'tls'),
42         'username' => env('MAIL_USERNAME'),
43         'password' => env('MAIL_PASSWORD'),
44         'timeout' => null,
45         'local_domain' => env('MAIL_EHLO_DOMAIN'),
46     ],
47
48     'ses' => [
49         'transport' => 'ses',
50     ],
51
52     'mailgun' => [
53         'transport' => 'mailgun',
54     ],
55
56     'postmark' => [
57         'transport' => 'postmark',
58     ],
59
60     'sendmail' => [
61         'transport' => 'sendmail',
62         'path' => env('MAIL_SENDMAIL_PATH', '/usr/sbin/sendmail -bs -i'),
63     ],
64
65     'log' => [
66         'transport' => 'log',
```



The screenshot shows the VS Code interface with the Explorer pane on the left showing the project structure. The main editor displays the contents of `app/Mail/OrderShipped.php`. The code defines the `OrderShipped` class, which extends `Mailable` and implements the `Queueable` and `SerializesModels` interfaces.

```
1 k?php
2
3 namespace App\Mail;
4
5 use Illuminate\Bus\Queueable;
6 use Illuminate\Contracts\Queue\ShouldQueue;
7 use Illuminate\Mail\Mailable;
8 use Illuminate\Mail\Mailables\Content;
9 use Illuminate\Mail\Mailables\Envelope;
10 use Illuminate\Queue\SerializesModels;
11
12 class OrderShipped extends Mailable
13 {
14     use Queueable, SerializesModels;
15
16     /**
17      * Create a new message instance.
18      *
19      * @return void
20      */
21     public function __construct()
22     {
23         //
24     }
25
26     /**
27      * Get the message envelope.
28      *
29      * @return \Illuminate\Mail\Mailables\Envelope
30      */
```

© Copyright **Aryatechno**. All Rights Reserved. Written tutorials and materials by [Aryatechno](#)