Implementing Passwordless Authentication in Laravel 12

Most users hate remembering passwords. Weak or reused passwords are a top cause of account breaches. A modern alternative is **Passwordless Authentication** — letting users log in using only their email address or phone number with a one-time link or code. This not only improves security but also makes login more user-friendly.

In this tutorial, we'll implement **Passwordless Authentication in Laravel 12** using a "magic link" approach. Users enter their email, receive a secure login link, and access the app without ever setting a password. We'll walk through migrations, controllers, email sending, and securing the login flow step by step.

1 – Why Passwordless?

Passwords are often weak, reused, or forgotten. Passwordless login:

- Improves **user experience** (no need to remember credentials).
- Improves **security** (no weak passwords to crack).
- Reduces **support requests** ("forgot password" flows).

Laravel makes it easy to build this flow using signed URLs and email notifications.

2 – Create Migration for Magic Links

We need a table to store temporary tokens that represent magic login links.

```
php artisan make:migration create_magic_links_tableCode language: Bash (bash)
```

```
// database/migrations/xxxx_xx_xx_create_magic_links_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('magic_links', function (Blueprint $table) {
            $table->id();
$table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->string('token')->unique();
            $table->timestamp('expires_at');
            $table->timestamps();
        });
    }
    public function down(): void {
        Schema::dropIfExists('magic_links');
    }
};Code language: PHP (php)
```

Each link belongs to a user, has a unique token, and an expiration time (e.g., 15 minutes).

3 – Model for Magic Links

```
// app/Models/MagicLink.php
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Carbon;

class MagicLink extends Model
{
    protected $fillable = ['user_id','token','expires_at'];

    public function isExpired(): bool
    {
        return Carbon::now()->greaterThan($this->expires_at);
    }
}Code language: PHP (php)
```

This model includes a helper method to check if the token is expired.

4 – Controller to Handle Requests

We'll create a controller to generate links, send them by email, and log the user in when they click.

```
php artisan make:controller Auth/MagicLinkControllerCode language: Bash (bash)
```

```
// app/Http/Controllers/Auth/MagicLinkController.php
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Models\User;
use App\Models\MagicLink;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Str;
```

```
use Carbon\Carbon;

class MagicLinkController extends Controller
{
    public function requestLink(Request $request)
    {
        $request->validate(['email' => 'required|email']);
        $user = User::where('email',$request->email)->first();

        if (! $user) {
            return back()->withErrors(['email' => 'No account
found.']);
        }

        $token = Str::random(64);

        $magic = MagicLink::create([
            'user_id' => $user->id,
            'token' => $token,
            'expires_at' => Carbon::now()->addMinutes(15),
        ]);

        $url = route('magic.login', $token);

        Mail::raw("Click here to log in: $url", function($m) use
($user) {
            $m->to($user->email)->subject('Your Magic Login Link');
        });

        return back()->with('status','We emailed you a magic login
link!');
    }

    public function login($token)
    {
        $magic = MagicLink::where('token',$token)->first();

        if (! $magic || $magic->isExpired()) {
```

```
            return redirect()->route('login')->withErrors(['email' =>
'Link expired or invalid.']);
        }

        Auth::login($magic->user, remember:true);

        $magic->delete(); // one-time use

        return redirect()->intended('/');
    }
}Code language: PHP (php)
```

Explanation:

- `requestLink`: Validates email, creates token, emails user.
- `login`: Validates token, logs user in, deletes the token so it can't be reused.

5 – Routes

```
// routes/web.php
use App\Http\Controllers\Auth\MagicLinkController;

Route::get('/magic-link', function() {
    return view('auth.magic-link');
})->name('magic.form');

Route::post('/magic-link',
[MagicLinkController::class,'requestLink'])->name('magic.request');
Route::get('/magic-login/{token}',
[MagicLinkController::class,'login'])->name('magic.login');Code language:
PHP (php)
```

We added three routes: a form, a POST request to send the link, and a GET route to log in

via token.

6 – Blade Form

```
<!-- resources/views/auth/magic-link.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container" style="max-width:480px">
  <h1 class="h4 mb-3">Login without a password</h1>

  @if (session('status'))
    <div class="alert alert-success">{{ session('status') }}</div>
  @endif

  <form method="POST" action="{{ route('magic.request') }}">
    @csrf
    <div class="mb-3">
      <label class="form-label">Email</label>
      <input type="email" name="email" class="form-control" required>
    </div>
    <button class="btn btn-primary">Send Login Link</button>
  </form>
</div>
@endsectionCode language: PHP (php)
```

This simple form lets users request a login link by entering their email. No password needed.

7 – Common Errors & Fixes

- **No email received:** Check your mail configuration (`.env`) and use `Mailtrap` for testing.
- **Link expired:** Default expiration is 15 minutes. Adjust `addMinutes()` if you need longer.
- **Token reused:** We delete the token after login to prevent reuse.

Wrapping Up

You just implemented **Passwordless Authentication in Laravel 12**. With magic links, users can log in securely without memorizing passwords. This improves UX and reduces security risks from weak or reused passwords. You can extend this by adding SMS one-time codes, limiting devices, or adding 2FA for sensitive apps.

What's Next

- Implementing Two-Factor Authentication in Laravel — add stronger identity checks.
- How to Build Email Verification in Laravel 12 — confirm real user emails.
- Implementing Password Reset in Laravel 12 Without Packages — alternative login recovery method.