How to Expire User Sessions Automatically in Laravel

When building secure applications, controlling **how long a user session stays active** is critical. If sessions never expire, users might remain logged in for days, weeks, or even indefinitely — which increases the risk of stolen sessions, unauthorized access, or compliance issues in industries like finance or healthcare.

In this guide, you'll learn **how to expire user sessions automatically in Laravel 12**. We'll go step by step through Laravel's built-in session lifetime configuration, how to implement absolute session timeouts with middleware, and even how to manage session expiration at the database level. Along the way, we'll also cover common mistakes and how to fix them.

1 – Understanding Laravel Sessions

Laravel uses **sessions** to keep track of logged-in users across multiple requests. By default, session data is stored in files (`storage/framework/sessions`), but you can also use databases, Redis, or Memcached. Each session has an `expiration time`, which defines how long the user can stay logged in without activity.

There are two types of expiration you may want to configure:

- **Idle timeout (sliding expiration):** The session ends after X minutes of inactivity. Each request resets the timer.
- **Absolute timeout:** The session ends after X minutes, no matter how active the user is.

Laravel provides idle timeout out of the box via `SESSION_LIFETIME`. For absolute timeout, we'll implement custom middleware.

2 – Configure Idle Timeout in `.env`

By default, sessions last 120 minutes (2 hours). To reduce this to 30 minutes of inactivity, set the following in your `.env` file:

```
# .env
SESSION_LIFETIME=30Code language: Bash (bash)
```

This means if a user is inactive for 30 minutes, their session will automatically expire. Once they try to interact with the app again, they'll be redirected to the login page.

Don't forget to clear and cache your configuration after changing the `.env` file:

```
php artisan config:clear
php artisan config:cacheCode language: Bash (bash)
```

3 – Implement Absolute Session Timeout

Some applications require sessions to expire after a fixed duration, even if the user is actively browsing. For example, a banking app may force a logout after 1 hour for compliance reasons. Laravel doesn't provide this out of the box, but you can implement it with custom middleware.

```
php artisan make:middleware SessionTimeoutCode language: Bash (bash)
```

```
// app/Http/Middleware/SessionTimeout.php
namespace App\Http\Middleware;
```

```
use Closure;
use Illuminate\Support\Facades\Auth;

class SessionTimeout
{
    protected $timeout = 3600; // 1 hour (in seconds)

    public function handle($request, Closure $next)
    {
        if (Auth::check()) {
            $lastActivity = session('last_activity_time');
            $currentTime = time();

            if ($lastActivity && ($currentTime - $lastActivity) >
$this->timeout) {
                Auth::logout();
                $request->session()->invalidate();
                $request->session()->regenerateToken();

                return redirect()->route('login')
                    ->withErrors(['email' => 'Your session has
expired. Please log in again.']);
            }

            session(['last_activity_time' => $currentTime]);
        }

        return $next($request);
    }
}Code language: PHP (php)
```

This middleware tracks the last activity timestamp and forces logout if the session has exceeded one hour. You can adjust the `$timeout` value as needed.

Register the middleware in `app/Http/Kernel.php`:

```
// app/Http/Kernel.php
protected $middlewareGroups = [
    'web' => [
```

```
        // ...
        \App\Http\Middleware\SessionTimeout::class,
    ],
];Code language: PHP (php)
```

Now every request will check if the session has exceeded the absolute timeout limit. If so, the user is logged out and redirected to the login page.

4 – Store Sessions in the Database (Optional)

By default, Laravel stores sessions in files. If you want more control — like forcing logouts across devices — switch to the `database` driver. This way, you can manage sessions with SQL queries.

```
php artisan session:table
php artisan migrateCode language: Bash (bash)
```

This creates a `sessions` table. Now update `.env` to use it:

```
# .env
SESSION_DRIVER=databaseCode language: Bash (bash)
```

Advantages of database sessions:

- You can see who is logged in by querying the `sessions` table.
- You can force logouts by deleting rows.
- You can implement additional expiration rules at the database level.

5 – Auto-Expire Sessions on Browser Close

If you want sessions to expire immediately when the browser closes, set the `expire_on_close` option in `config/session.php` to `true`:

```
// config/session.php
'expire_on_close' => true,Code language: PHP (php)
```

With this enabled, the session cookie is deleted when the browser closes, requiring the user to log in again next time they open the site.

6 – Common Errors & Fixes

- **"Session expired too quickly":** Check `SESSION_LIFETIME`. It counts minutes of inactivity, not absolute time.
- **Users not logged out after timeout:** Make sure your custom middleware is registered in `Kernel.php` and applied to the `web` group.
- **Session not persisting after login:** Ensure `APP_URL` and `SESSION_DOMAIN` are set correctly in `.env`, especially if using subdomains.
- **Database sessions not working:** Confirm the `sessions` table exists and you've set `SESSION_DRIVER=database`.

Wrapping Up

You now know multiple ways to **automatically expire user sessions in Laravel 12**. You can configure idle timeouts with `SESSION_LIFETIME`, implement absolute timeouts with middleware, or even store sessions in the database for maximum control. Combined with

features like email verification and two-factor authentication, this helps you build a secure, professional-grade Laravel application.

What’s Next

- Implementing Two-Factor Authentication in Laravel — add another layer of login security.
- How to Prevent CSRF, XSS, and SQL Injection in Laravel Apps — protect against common vulnerabilities.
- Laravel Authentication: Redirect Users After Login & Logout — improve user experience.