

Hoe de acties van gebruikers achterhalen in Laravel?

09/02/2022

Opgesteld door: Morgane Van Velthoven

Hoe de acties van gebruikers achterhalen in Laravel?

Inhoudsopgave

INLEIDING.....	3
VOORAFGAANDE KENNIS & ONDERZOEK.....	4
HET IN PRAKTIJK BRENGEN.....	5
AJAX.JS	5
AJAXCONTROLLER.PHP	6
FOUTMELDINGEN OPLOSSEN	7
ACTIONSCONTROLLER.PHP	8
LARAVEL.LOG.....	8
CONCLUSIE.....	9
BRONNEN.....	10

Lijst met figuren

Figuur 1: Code ajax.js	5
Figuur 2: linken van scripts in master.blade.php	5
Figuur 3: Foutmelding "404 (Not Found)"	6
Figuur 4: Code AjaxController.php	6
Figuur 5: Route naar /ajaxRequest in web.php	7
Figuur 6: Foutmelding "419 (unknown status)"	7
Figuur 7: X-CSRF-TOKEN meegeven in de Ajax header in ajax.js	7
Figuur 8: HTML meta tag in master.blade.php	7
Figuur 9: Functie processForm	8
Figuur 10: Info over aangeklikt element in laravel.log	8
Figuur 11: Contactformulier weergegeven in laravel.log	8

Inleiding

Het is altijd handig om te weten wat gebruikers doen met een website, welke acties ze ondernemen. Je hebt hiervoor dan ook verschillende tools om dit te monitoren, denk maar aan Google Analytics. Deze tools zijn niet altijd duidelijk of vergen veel tijd om te begrijpen. Dit heb ik dan ook gemerkt met de website van mijn moeder, gemaakt in Laravel, die Google Analytics gebruikt. We kijken naar Google Analytics om de website te optimaliseren, maar het is niet erg duidelijk en mijn moeder begrijpt er niets van wat de statistieken aantonen. Om deze reden heb ik besloten om dit op een makkelijkere manier te doen dat ook voor haar begrijpbaar is.

Na onderzoek en opzoekingswerk heb ik besloten om via de logs van Laravel bij te houden wat elke gebruiker doet. Dit wil zeggen dat wanneer een gebruiker ergens op klikt, dit wordt weergegeven in het log bestand. Op deze manier begrijpt ze wel wat elke gebruiker doet en vindt ze dat alles overzichtelijker is.

In deze tutorial zal ik uitleggen en demonstreren hoe ik dit heb gedaan. Om alles begrijpbaar te houden ben ik een mini website ontwikkeld in Laravel.

De volledige website is beschikbaar op mijn GitHub repository:

<https://github.com/MorganeVV/usersactionstracker>.

Voorafgaande kennis & onderzoek

Om deze tutorial tot een goed einde te brengen en te weten welke weg ik moest volgen, heb ik mijn kennis van Laravel en jQuery gebruikt. Natuurlijk is een website gemaakt met Laravel een must.

Mocht je zelf nog geen Laravel website hebben geeft de documentatie van Laravel Homestead je alles om dit te ontwikkelen en te laten draaien. ([Laravel Homestead](#), z.j.)

Kennis van Laravel was dan ook nodig voor het maken van de website, maar ook om de verschillende manier van gegevens op te slaan te begrijpen. Gegevens in Laravel kan je opslaan in een database, je kan werken met storage of logs ([Writing Log Messages](#), z.j.). Laravel documentatie, maar ook W3Schools en de jQuery documentatie hebben mijn geholpen bij het maken van de juiste beslissing.

De keuze is om te gaan werken met logging. Achteraf gezien was het ook de meest logische keuze als je acties van een gebruiken wil loggen. Werken met de storage zelf was ook mogelijk, hier ging ik dan een bestand aanmaken ik de storage waar ik alle acties zou opslaan. Ik heb hiervan afgezien omdat je met Laravel kan werken met Logs.

Alle acties zullen dus worden gelogd in het logbestand van Laravel. De volgende stap is om te weten hoe ik de communicatie tussen client en server ging aanpakken.

Na wat rondzoeken heb ik Ajax ontdekt. Met Ajax kan je data lezen van de web client en deze verzenden naar de server in de achtergrond, dus zonder de pagina te herladen. Wat voor dit ideaal is. ([What is Ajax?](#), z.j.)

Het plan is om alles dat klikbaar is een klasse "klikker" te geven. Met deze klasse kan je bij iedere klik een Ajax post request sturen. Het request bevat de sessie-ID, pagina naam en het geklikte onderdeel. Dit request wordt opgevangen door een Laravel controller die de data in het logbestand zet. Bij een formulier zal de controller dit doet i.p.v. het Ajax script.

Met al deze informatie kon ik alles in praktijk brengen!

Het in praktijk brengen

Alles van de voorafgaande kennis en het onderzoek zal hier in praktijk worden gebracht. Dit zal in verschillende stappen gebeuren.

Stap 1: alle klikbare elementen krijgen de klasse "klikker", behalve de submit buttons van een formulier.

Stap 2: een Ajax script, ajax.js, schrijven die bij elke klik van een element met klasse "klikker" een post request stuurt naar */ajaxRequest*.

Stap 3: een controller, AjaxController, maken die de Ajax post request van */ajaxRequest* opvangt en de data naar het logbestand stuurt.

Stap 4: voor formulieren zal de controller die het formulier verwerkt en valideert, de data naar het log bestand sturen.

ajax.js

ajax.js zal ervoor zorgen dat bij elke klik van een element met klasse "klikker" data wordt verstuurd met een post request. Dit post request zal verstuurd worden naar */ajaxRequest* met de bijhorende data.

Om met Ajax een post request te sturen, heb je bepaalde parameters nodig. Het type 'POST' is, de url */ajaxRequest* en de data die de pagina naam en het geklikte element bevat. Het geklikte element wordt hier het target genoemd.

Wanneer je dit gecodeerd hebt, zou je iets moeten hebben dat overeenkomt met figuur 1.

Dit script link je dan met de master blade, maar vergeet niet om een jquery script ervoor te linken (zie figuur 2).

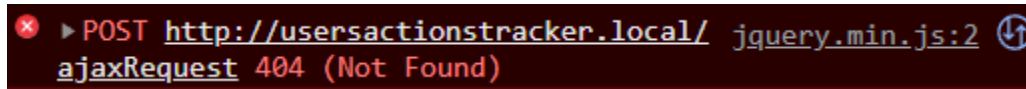
```
$(".klikker").click(function (e) {  
    element = e.target;  
    var target = element.outerHTML;  
    var page = window.location.href;  
  
    $.ajax({  
        type: 'POST',  
        url: "/ajaxRequest",  
        data: { page: page, target: target }  
    });  
});
```

FIGUUR 1: CODE AJAX.JS

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  
<script type="text/javascript" src="{{ asset('js/ajax.js') }}"></script>
```

FIGUUR 2: LINKEN VAN SCRIPTS IN MASTER.BLADE.PHP

Wanneer je deze code uittest geeft dit de foutmelding "404 (Not Found)" (zie figuur 3). Dit komt omdat we zeggen tegen Ajax, stuur deze data met een post request naar de pagina `/ajaxRequest`, maar deze pagina bestaat niet. Deze error zal je na het maken van de controller oplossen.



FIGUUR 3: FOUTMELDING "404 (NOT FOUND)"

AjaxController.php

AjaxController.php bevat de functie waarmee je de post request opvangt. Je kan deze controller aanmaken met het commando: `php artisan make:controller AjaxController`. Omdat we gebruiken maken van de Log, moeten we ook `use Log` meegeven.

Je maakt een functie, `ajaxRequestPost`, aan waaraan een parameter wordt meegegeven. Deze parameter is een Request. Deze request zal dan verstuurt worden naar het logbestand. Vergeet ook niet de sessie-ID mee te geven. Deze kan je verkrijgen met `session()->getId()` en voeg je toe aan de request `$request['session_id']=$session_id`.

De input data is volledig en kan worden verstuurt naar het log bestand. `Log::info($input)`.

Figuur 4 toont de volledige code van deze controller.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Log;

class AjaxController extends Controller
{
    public function ajaxRequestPost(Request $request)
    {
        $session_id = session()->getId();
        $request['session_id'] = $session_id;
        $input = $request->all();

        Log::info($input);

        return response()->json(['success'=>'Ajax request ontvangen.']);
    }
}
```

FIGUUR 4: CODE AJAXCONTROLLER.PHP

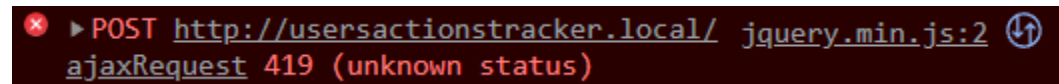
Foutmeldingen oplossen

Telkens er een post request wordt verstuurd naar `/ajaxRequest` wil je dat de functie `ajaxRequestPost(Request $request)` wordt uitgevoerd. Dit doe je met een route toe te voegen aan `web.php` (zie figuur 5).

```
Route::post('/ajaxRequest', [AjaxController::class, 'ajaxRequestPost']);
```

FIGUUR 5: ROUTE NAAR /AJAXREQUEST IN WEB.PHP

Dit lost ook de foutmelding (zie figuur 3) op. Als je deze code test zal je merken dat het inderdaad verdwenen is. Echter is er een andere foutmelding in de plaats gekomen (figuur 6). Dit komt omdat Laravel controleert op de X-CSRF-TOKEN request header. Deze token kan je bewaren in een HTML meta tag (zie figuur 7). Daarna kan je in `ajax.js` ervoor zorgen dat er in de request header een X-CSRF-TOKEN wordt meegegeven (zie figuur 8). ([X-CSRF-TOKEN](#), z.j.).



✖ POST http://usersactionstracker.local/ jquery.min.js:2 ↕
ajaxRequest 419 (unknown status)

FIGUUR 6: FOUTMELDING "419 (UNKNOWN STATUS)"

```
$.ajaxSetup({  
  headers: {  
    'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')  
  }  
});
```

FIGUUR 7: X-CSRF-TOKEN MEEGEVEN IN DE AJAX HEADER IN AJAX.JS

```
<meta name="csrf-token" content="{{ csrf_token() }}" />
```

FIGUUR 8: HTML META TAG IN MASTER.BLADE.PHP

Door het toevoegen van deze tokens verdwijnt nu ook de 419 foutmelding (zie figuur 6).

ActionsController.php

Wanneer je website ook formulieren bevat, heb je de submit buttons geen klasse "klikker" gegeven, omdat dan de data van het formulier niet gevalideerd wordt. In de functie in de controller die het formulier valideert voeg je een lijn code toe die de gevalideerde data naar het logbestand verstuurd als json (zie figuur 9 op de volgende pagina). In dit voorbeeld is dat de functie *processForm(Request \$request)*.

```
public function processForm(Request $request)
{
    $validatedData = $request->validate([
        'naam' => ['required', 'max:50'],
        'email' => ['required'],
        'reden' => ['required', 'max:255'],
    ]);

    Log::info("Contactformulier", $validatedData);

    return response()->json(['Het contactformulier is doorgestuurd. Uw ingevulde gegevens:', $validatedData]);
}
```

FIGUUR 9: FUNCTIE PROCESSFORM

Laravel.log

In laravel.log verschijnt er bij elke klik die er gebeurt een entry met de juiste informatie (zie figuur 11). Deze informatie is een array bestaande uit een pagina naam, target en sessie_id. De datum en tijd van het moment dat het in het logbestand komt verschijnt ook. Dit kan handig zijn om de tijd tussen acties weer te geven voor latere statistieken.

Van het contactformulier wordt de entry getoond als json formaat met de meegegeven titel (zie figuur 11). Deze entry is op één regel, maar voor het nog duidelijk te houden heb ik deze opgesplitst over meerdere regels.

```
[2022-01-17 10:49:00] local.INFO: array (
    'page' => 'http://usersactionstracker.local/',
    'target' => '<input type="button" value="Klik knop" class="klikker">',
    'session_id' => '5PCczqG9CRYP7xWqtHikYTadWbrINFd0RYZAWW0H',
)
```

FIGUUR 10: INFO OVER AANGEKLIKT ELEMENT IN LARAVEL.LOG

```
[2022-02-11 20:28:49] local.INFO: Contactformulier {"naam":"Morgane","email":"morganevv@live.be",
    "reden":"De gevalideerde data wordt weergegeven in laravel.log",
    "session_id":"z03nURmLQdw3Amyg0a5KoP9ekIdl7k05LMqDkB6A"}
```

FIGUUR 11: CONTACTFORMULIER WEERGEGEVEN IN LARAVEL.LOG

Conclusie

Het praktische gedeelte, het programmeren, nam een paar dagen in beslag. De meeste tijd ging naar het opzoekingswerk en onderzoeken welke methodes het beste waren. Dit kwam vooral omdat bij het uitwerken van de theorie, problemen naar boven kwamen waar toen niet was aan gedacht.

Zoals toen bij de foutmelding van "419 (unknow status)" (zie figuur 6). Dit kwam, na weer wat opzoeking, omdat Laravel bij een post request gebruikt maakt van een CSRF token. Met deze token te bewaren in een HTML meta tag en mee te geven als header aan Ajax was de foutmelding verdwenen en werkte het.

Ik heb geleerd dat er niet direct iets hoeft te werken om juist te zijn. Dit werd vooral getoond met het feit dat `ajax.js` als eerste werd opgesteld en een foutmelding gaf (zie figuur 3), omdat de pagina `/ajaxRequest` niet bestaat en er dus ook geen post request naar verstuurd kan worden. Na het maken van de functie `ajaxRequestPost` en in `web.php` de route te definiëren, waar ook de pagina in "bestaan" komt, verdween de foutmelding. Je moet gewoon het proces vertrouwen.

Het was eens leuk om een eigen tutorial te maken, vooral omdat het gaat over een onderwerp dat mij interesseert. Ook kan ik dit gebruiken voor de website van mijn moeder.

Waar ik echter heel veel moeite mee had, was het uitschrijven van deze paper. In mijn hoofd weet ik wat ik wil zeggen, maar het schrijven ervan was heel moeilijk. Wat mij geholpen heeft was om het te vertellen tegen iemand en dit op te nemen. Met erna terug te luisteren kon ik gewoon typen wat ik zei, dit gaf mij de structuur en de manier waarop ik het kon uitschrijven.

Voor de toekomst kan je de data uit het logbestand halen en deze analyseren voor o.a. statistieken.

Bronnen

Laravel. (z.j.) Writing Log Messages. Geraadpleegd op 15 januari 2022 via <https://laravel.com/docs/9.x/logging#writing-log-messages>.

jQuery.post().(z.j.) Ajax | jQuery API Documentation. Geraadpleegd op 15 januari 2022 via <https://api.jquery.com/jQuery.post>

W3Schools. (z.j.) What is AJAX?. Geraadpleegd op 15 januari 2022 via https://www.w3schools.com/whatis/whatis_ajax.asp

W3Schools. (z.j.) jQuery - AJAX get() and post() Methods. Geraadpleegd op 15 januari 2022 via https://www.w3schools.com/jquery/jquery_ajax_get_post.asp

Laravel. (z.j.) X-CSRF-TOKEN. Geraadpleegd op 15 januari 2022 via <https://laravel.com/docs/9.x/csrf#csrf-x-csrf-token>