PHP

T 3.1 - Forms

Steven De Ryck

december 2020

Inhoud

[1 Opbouw van HTML forms 4](#_Toc58272583)

[1.1 Eenvoudige inputs 4](#_Toc58272584)

[1.2 Keuzelijsten 5](#_Toc58272585)

[1.3 Aankruisvakje (checkbox) 6](#_Toc58272586)

[1.4 Radiobuttons 7](#_Toc58272587)

[1.5 Knoppen 8](#_Toc58272588)

[1.6 Andere invoervelden 8](#_Toc58272589)

[1.7 Metadata toevoegen 9](#_Toc58272590)

[1.8 CSRF token toevoegen 10](#_Toc58272591)

[1.9 Formulier verzenden 12](#_Toc58272592)

[1.9.1 Method 12](#_Toc58272593)

[1.9.2 Action 12](#_Toc58272594)

[2 Verwerking van het formulier op de server 13](#_Toc58272595)

[2.1 Sanitizing values 14](#_Toc58272596)

[2.2 Validatie 15](#_Toc58272597)

[2.2.1 Validaties aan de hand van eigenschappen opgeslagen in de databank 15](#_Toc58272598)

[2.2.2 Andere validaties 16](#_Toc58272599)

[2.3 Opslaan in de databank 17](#_Toc58272600)

[2.4 Redirect na INSERT of UPDATE 19](#_Toc58272601)

[2.5 Redirect na een error 20](#_Toc58272602)

**Zie ook**

**W3Schools** heeft een beknopte tutorial over HTML Forms in PHP. Het is zeer nuttig om die eerst door te nemen.

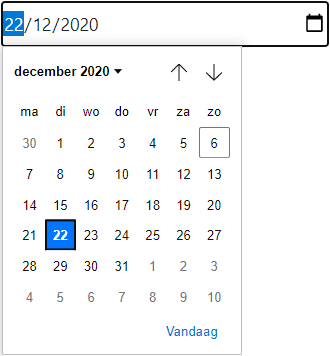
Zie <https://www.w3schools.com/php/php_forms.asp>

# Opbouw van HTML forms

## Eenvoudige inputs

Bij het maken van HTML forms, let je best op het volgende:

* Gebruik voor elk input element het **juiste HTML type**: een **number** voor een getal, **text** voor een string, **date** voor een datum …  
    
  Hoewel dit niet strict noodzakelijk is voor een correcte verzending – je kan alle elementen als **text** instellen en verzenden – is het toch **handig** dat je als gebruiker **increment/decrement** **pijltjes** krijg bij numbers, een **datepicker** bij dates, …   
    
  

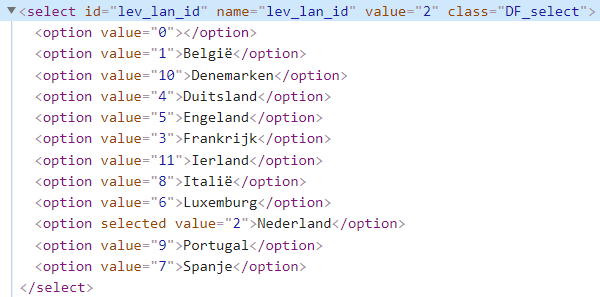


Een deel van de **client-side validatie** is hiermee ook al ingebouwd: in een **number** veld kan je enkel getallen invoeren, …   
  
Opgelet: de **server-side validatie** valt hierdoor niet weg!

* Voeg aan al je inputs het attribuut **name** toe. **Enkel inputs met een name** worden verzonden. Een veel gebruikte conventie voor het **name** attribuut is de naam van het veld in de databank: **img\_id**, **img\_title**, …
* De waarde van een veld zet je – bij update – in het **value** attribuut; niet in de **placeholder**: die is bedoeld om extra info te geven bij het invullen van de velden
* Gebruik geen **disabled** attribuut: disabled elementen worden **niet mee verzonden**. Je kan wel **readonly** gebruiken.

## Keuzelijsten

Een **foreign key** in je tabel – bijvoorbeeld **lev\_lan\_id** – wordt vaak een **keuzelijst** in je formulier. In het voorbeeld hieronder: het **land van een leverancier**.



Als je keuzelijsten voorziet in je formulier, bouw deze dan zo op:

* in de **select** tag zet je de naam van het **foreign key veld** in de attributen **id** en **name**, de numerieke waarde ervan in **value**
* de **option** tags bouw je op met een PHP **loop** die alle records in de tabel **land** overloopt; de **value** is de waarde van de **primary key**, tussen de **option** begin- en eindtag staat de **omschrijving**
* de **option** die de waarde heeft van de **foreign key**, krijg het attribuut **selected**
* voeg eventueel een **0-waarde zonder omschrijving** toe als het veld optioneel is

Je kan dit alles in een functie **MakeSelect()** stoppen met als argumenten:

* **$fkey**: de naam van het foreign key veld, bv. **lev\_lan\_id**
* **$value**: de waarde van het foreign key veld, bv. **2**
* **$sql**: de selectie om de options list samen te stellen, bv. **select lan\_id, lan\_naam from land order by lan\_naam**

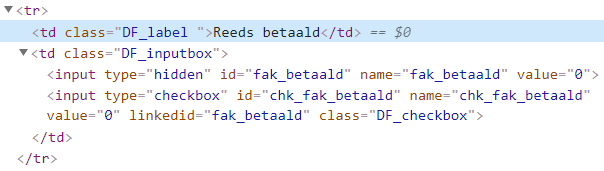
## Aankruisvakje (checkbox)



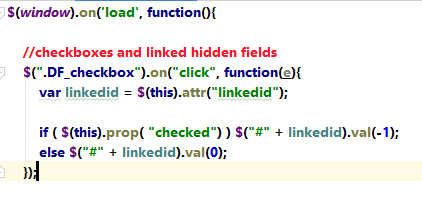
Een checkbox input heeft een attribuut **checked** als hij aangevinkt is, dat wegvalt als het vinkje wordt weggehaald. Enkel **checked checkboxes** worden mee naar de server gestuurd bij verzending van het formulier.

Dat is een beetje vervelend aan de serverzijde; daarom wordt aan een checkbox vaak een **hidden input** toegevoegd, die via Javascript de waarde 0 of -1 krijgt (of 0 en 1) wanneer men de eigenlijke checkbox aan- of uitvinkt.

Het is de **hidden input die altijd mee naar de server verzonden wordt**, en die als **id** en **name** de eigenlijke veldnaam bevat, bv. **fak\_betaald** in het voorbeeld.



Via **JQuery** wordt hieronder de hidden input op 0 of -1 gezet als men op de bijhorende checkbox klikt. Je kan dit ook in **zuiver** (‘vanilla’) **Javascript** schrijven.



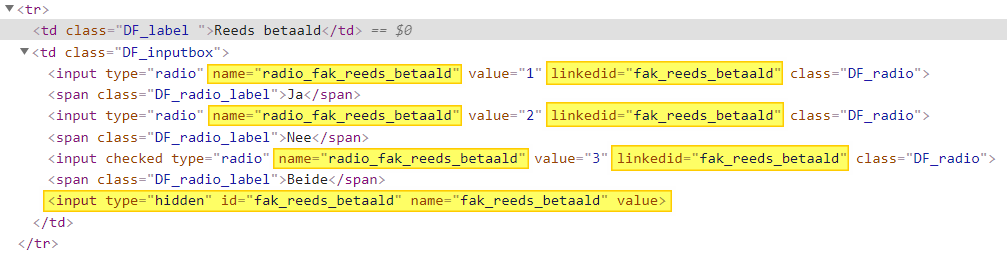
## Radiobuttons

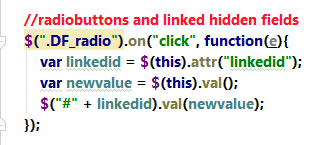


Bij radiobuttons heb je een beetje hetzelfde probleem.

Het is makkelijker om **één veld te verzenden** naar de server dat de waarde van 1 van de opties heeft.

In het voorbeeld hieronder zie je het **hidden input** veld **fak\_reeds\_betaald**, dat via **JQuery** de waarde van de aangeduide optie krijgt.





## Knoppen

Je kan meer dan één knop toevoegen aan je form. Hieronder een voorbeeld van hoe je – na verzending – bepaalt **op welke knop** de gebruiker geklikt heeft:



## Andere invoervelden

Naast de reeds genoemde input types zijn er natuurlijk nog andere soorten: **textarea**, **color**, **month**, …

Ook voor deze types kan je natuurlijk **aangepaste PHP functies** creëren.

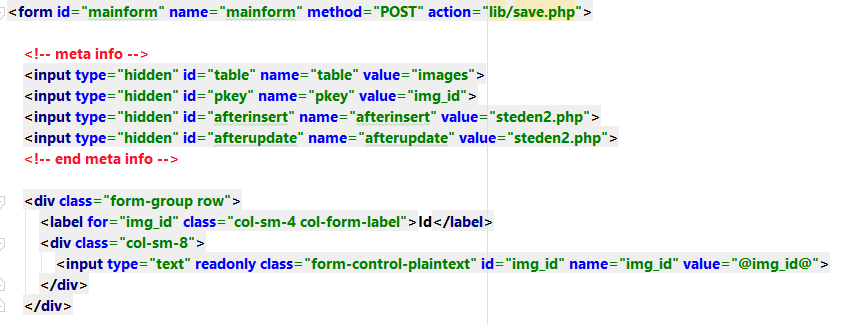
## Metadata toevoegen

Het kan nuttig zijn om extra informatie, ‘**metadata**’, toe te voegen aan je formulieren.

Bijvoorbeeld:

* de naam van de **(hoofd)tabel** waaruit de data afkomstig zijn: dat laat je toe om het uit te voeren SQL statement hierop te baseren
* de naam van de **primary key** van deze tabel
* **links** naar scripts die uitgevoerd moeten worden na opslaan (insert of update)
* **csrf token**: zie verder
* …

Metadata zet je best in **hidden inputs** – met een **name** attribuut – zodat ze mee verzonden worden.



## CSRF token toevoegen

**CSRF** staat voor **Cross Site Request Forgery**, en is een techniek waarbij hackers je formulier – of beter de ‘**action**’ van je formulier, het script dat de verwerking doet – misbruiken om bepaalde (ongewenste) commando’s naar de server te verzenden.

Voor meer info over dit soort kwetsbaarheid in je applicatie, zie <https://en.wikipedia.org/wiki/Cross-site_request_forgery> of Google naar ‘CSRF attack’ of ‘CSRF token’.  
  
Om je formulier te beveiligen tegen **CSRF aanvallen**, voeg je best een **CSRF token** toe. Dat is een **unieke code** die bij elke http-request vernieuwd wordt.

* **security.php**: dit script bevat een functie **GenerateCSRF()** om een CSRF token aan te maken, en deze op te slaan in de **sessie** van de gebruiker**:**

**<?php***session\_start***()**;  
  
**function** GenerateCSRF**( string $formname** = **"noformname" )**: **string  
{  
 $csrf\_key** = *bin2hex***(** random\_bytes**(**32**) )**;  
 **$csrf** = *hash\_hmac***( 'sha256', 'PHP1CURSUS SECRET KEY '** . **$formname, $csrf\_key )**;  
  
 **//store CSRF token in SESSION  
 $\_SESSION['lastest\_csrf']** = **$csrf**;  
  
 **return $csrf**;  
**}**

* **stad\_form.html**: in het formulier wordt een hidden input voorzien voor de CSRF token:

**<!--security-->  
<input type="hidden" name="csrf" value="@csrf\_token@">  
<!--end security-->**

* **stad\_form.php**: roept de functie op om de token aan te maken, en voegt hem in in het formulier:

**<?php  
 if (** ! *is\_numeric***( $\_GET['img\_id']) ) die("Ongeldig argument "** . **$\_GET['img\_id']** . **" opgegeven")**;  
  
 **//get data  
 $data** = GetData**( "select** *\** **from images where img\_id="** . **$\_GET['img\_id'] )**;  
  
 **//add CSRF token to data  
 $data[**0**]["csrf\_token"]** = GenerateCSRF**( "stad\_form.php" )**;  
  
 **//get template  
 $template** = *file\_get\_contents***("templates/stad\_form.html")**;  
  
 **//merge  
 $output** = MergeViewWithData**( $template, $data )**;  
 **print $output**;  
**?>**

Het resultaat is dat in je formulier nu een hidden input staat met een **unieke code**:



Die code wordt dus mee met de andere data verzonden naar onze server. De code is maar **éénmaal geldig**. De server controleert de geldigheid.

* **save.php**: bij de verwerking van het formulier, controleren we of er een geldige CSRF token mee verzonden werd, en maken de laatst gebruikte token ongeldig:

**function** SaveFormData**()  
{  
 if ( $\_SERVER['REQUEST\_METHOD']** == **"POST" )  
 {  
 if (** ! *key\_exists***("csrf", $\_POST)) die("Missing CSRF")**;  
 **if (** ! *hash\_equals***( $\_POST['csrf'], $\_SESSION['lastest\_csrf'] ) ) die("Problem with CSRF")**;  
  
 **$\_SESSION['lastest\_csrf']** = **""**;

…

## Formulier verzenden

### Method

Formulieren verzend je met de **POST methode**.

Dat heeft verschillende voordelen, en is ook **overeenkomstig de specificaties van het HTTP protocol**:

* **POST** is om **nieuwe gegevens** toe te voegen (of te wijzigen) op de server
* **GET** is enkel voor het **opvragen** van informatie

Zie <https://www.w3schools.com/php/php_forms.asp>

### Action

Het script waarnaar je verzendt (de ‘**action**’ van je formulier), kan je zelf kiezen:

* **hetzelfde script** dat het formulier genereert (dan hoef je geen **action** op te geven)
* een algemeen ‘**verwerkingsscript**’, zoals ‘**save.php**’
* een **specifieke pagina** (bv. **welcome.php** na inloggen)
* …

Belangrijk is om ervoor te zorgen dat je zo weinig mogelijk ‘**code duplication**’ op de server toelaat, en **algemeen bruikbare functies of classes** voorziet: hierdoor kan je je website of applicatie sneller ontwikkelen, sneller aanpassen en beter onderhouden.

Gebruik **geen variabelen** voor het **action** attribuut. Dat is het veiligst. Als je dat toch wil doen, lees dan alle documentatie over mogelijke misbruiken daarvan.

Wees je er ook altijd van bewust dat het script dat de verwerking doet (**save.php** bijvoorbeeld), door kwaadwillige zielen gebruikt zou kunnen worden om gegevens te verzenden, en dat het dus uiterst belangrijk is de nodige beveiligingen in te bouwen:

* ga na of het script de **identiteit van een ingelogde gebruiker** controleert; is dat het geval, dan is het aantal potentiële aanvallen al heel wat beperkter, vermits ze afkomstig moeten zijn van ingelogde gebruikers
* is dat niet het geval – heb je met andere woorden een **publiek toegankelijk ‘endpoint’** – voorzie dan een **beveiliging tegen robots** die voortdurend nieuwe posts proberen uit te voeren, en allerlei rommel in je databank kunnen pompen; hier bestaan verschillende systemen voor (zie Google)

# Verwerking van het formulier op de server

Vooraleer de data die verzonden werden vanuit een formulier, opgeslagen kunnen worden in de databank, dienen er een aantal zaken te gebeuren:

* de doorgestuurde waarden ‘**opkuisen**’ (**sanitize**): overbodige witruimte verwijderen, ongewenste karakters zoals quotes verwijderen of converteren, …
* de geldigheid van de data **controleren** (**validatie**): getallen, strings, e-mails, …
* eventueel bepaalde gegevens **loggen**
* …

Uiteindelijk zal je de gegevens willen **opslaan in de databank** van je applicatie

## Sanitizing values

Overbodige witruimte – spaties, tabs vooraan en achteraan een stuk tekst – kan je best verwijderen met **trim**.

|  |  |
| --- | --- |
| trim | verwijdert overbodige witruimte vooraan en achteraan een string |

Strings waarin tekens voorkomen die in SQL een **speciale betekenis** hebben - zoals <, >, “, ‘, … - dienen geconverteerd te worden. Daarvoor zijn er verschillende functies.

|  |  |
| --- | --- |
| htmlspecialchars  mysql\_real\_escape\_string  htmlentities | deze functies converteren tekens die problemen kunnen veroorzaken bij de uitvoering van het SQL statement waarin ze voorkomen |

Mogelijk wil je nog een aantal zaken in een **standaard formaat** opslaan:

* telefoonnummers als +32 A BBB BB BB
* gsm nummers als +32 4AA BB BB BB
* wachtwoorden dienen **geëncrypteerd** (‘gehashed’) opgeslagen te worden
* …

Hiervoor dien je eventueel zelf de nodige functies te schrijven. Je kan natuurlijk op het net kijken wat er mogelijk al bestaat, onder andere hier: <https://www.php.net/manual/en/filter.examples.sanitization.php>

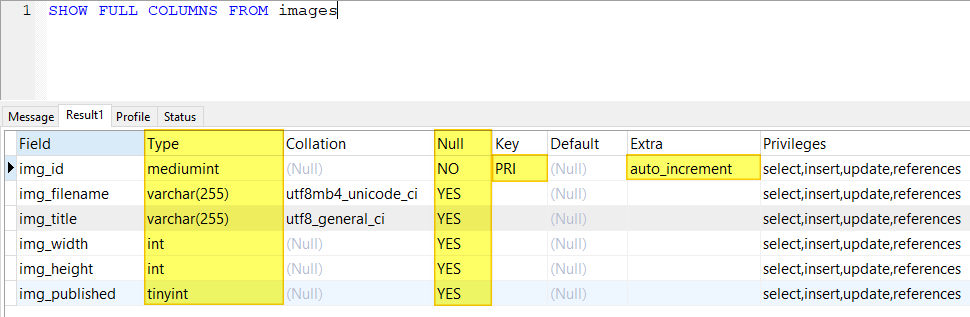
## Validatie

### Validaties aan de hand van eigenschappen opgeslagen in de databank

In een **numerieke** kolom (integer, mediumint, double, float, …) in de databank, kan je geen **string** opslaan. Het **type van de doorgestuurde gegevens** moet dus kloppen met het **veldtype in de databank**.

Je kan de info over de kolommen van een tabel – zoals het veldtype – opvragen uit de databank, en controleren of de doorgestuurde waarde daarmee compatibel is.

*SHOW FULL COLUMNS FROM images*



**Verplichte velden** kan je ook detecteren ( **Null** kolom hierboven gelijk aan **NO** ). Als een verplicht gegeven niet aanwezig is, kan je dus ook een foutmelding genereren.

Je kan ook de **primary key** en **autoincrement** kolom vinden.

* Met de **primary key** bouw je de WHERE-clause van je SQL statement bij UPDATE.
* Een **autoincrement** veld laat je (meestal) leeg in een INSERT-statement.

### 

### Andere validaties

Niet alle validaties kan je uit de databank halen:

* een **percentage** moet tussen 0 en 100 liggen
* een **e-mailadres** moet een **@** en een **punt** bevatten (en nog een aantal voorwaarden)
* **bankrekeningnummers** en **btw nummers** bevatten een **checkdigit** die moet kloppen
* een **datum voor een nieuwe afspraak** moet in de toekomst liggen
* een **nieuwe afspraak mag niet overlappen** met een bestaande afspraak
* een gegeven uit een **keuzelijst** moet één van de waarden uit die keuzelijst zijn
* een gegeven uit een reeks **radiobuttons** moet ook één van de toegelaten waarden zijn
* …

Voor sommige formulieren heb je daarom een **op maat gemaakte** validatieprocedure (-functie) nodig. Dit kan je op verschillende manieren implementeren.

## Opslaan in de databank

Uiteindelijk moeten de verzonden gegevens in de databank opgeslagen worden.

Het handigst is een **universele procedure/functie** die de input van **alle mogelijke formulieren** in je applicatie kan verwerken, een **SQL statement** kan opbouwen, en dat kan **uitvoeren** in de databank.

Als je de nodige metadata voorziet in je formulier, is dat niet zo moeilijk.

Hieronder een stukje voorbeeldcode – zonder **sanitization** en **validation** – dat gegevens van verschillende formulieren kan opslaan, voor INSERTS en UPDATES van gegevens:

**<?php  
require\_once "pdo.php"**;  
**require\_once "security.php"**;  
**require\_once "strings.php"**;  
  
SaveFormData**()**;  
  
**function** SaveFormData**()  
{  
 if ( $\_SERVER['REQUEST\_METHOD']** == **"POST" )  
 {  
 //controle CSRF token  
 if (** ! *key\_exists***("csrf", $\_POST)) die("Missing CSRF")**;  
 **if (** ! *hash\_equals***( $\_POST['csrf'], $\_SESSION['lastest\_csrf'] ) ) die("Problem with CSRF")**;  
  
 **$\_SESSION['lastest\_csrf']** = **""**;  
  
 **//controle metadata  
 $table** = **$pkey** = **$update** = **$insert** = **$where** = **$str\_keys\_values** = **""**;  
  
 **if (** ! *key\_exists***("table", $\_POST)) die("Missing table")**;  
 **if (** ! *key\_exists***("pkey", $\_POST)) die("Missing pkey")**;  
  
 **$table** = **$\_POST['table']**;  
 **$pkey** = **$\_POST['pkey']**;  
  
 **//insert or update?  
 if ( $\_POST["$pkey"]** > 0 **) $update** = **true**;  
 **else $insert** = **true**;  
  
 **if ( $update ) $sql** = **"UPDATE $table SET "**;  
 **if ( $insert ) $sql** = **"INSERT INTO $table SET "**;  
  
 **//make key-value string part of SQL statement  
 $keys\_values** = **[]**;  
  
 **foreach ( $\_POST as $field** => **$value )  
 {  
 //skip non-data fields  
 if (** *in\_array***( $field, [ 'table', 'pkey', 'afterinsert', 'afterupdate', 'csrf' ] ) ) continue**;  
  
 **//handle primary key field  
 if ( $field** == **$pkey )  
 {  
 if ( $update ) $where** = **" WHERE $pkey = $value "**;  
 **continue**;  
 **}  
  
 //all other data-fields  
 $keys\_values[]** = **" $field = '$value' "** ;  
 **}  
  
 $str\_keys\_values** = *implode***(" , ", $keys\_values )**;  
 **$sql** .= **$str\_keys\_values**;  
 **$sql** .= **$where**;  
  
 **$result** = ExecuteSQL**( $sql )**;

…

## Redirect na INSERT of UPDATE

Als de gegevens correct opgeslagen zijn in de databank, bepaal je wat de gebruiker vervolgens te zien krijgt:

* **hetzelfde formulier** met de ingevulde gegevens (bv. **stad\_form.php**)
* de pagina van waaruit hij oorspronkelijk op het formulier terecht gekomen is (bv. **steden2.php**)
* een **andere pagina**, bijvoorbeeld wanneer de gebruiker verschillende pagina’s in een procedure doorloopt, bv. bij registratie
* een **welkomstpagina** na registratie
* …

Vaak hangt dit af van het formulier zelf. Vandaar dat er in de **metadata** in het eerdere voorbeeld **2 links** voorzien zijn, die bepalen wat er gebeurt na **insert** en na **update**:

**<!-- meta info -->  
<input type="hidden" id="table" name="table" value="images">  
<input type="hidden" id="pkey" name="pkey" value="img\_id">  
<input type="hidden" id="afterinsert" name="afterinsert" value="steden2.php">  
<input type="hidden" id="afterupdate" name="afterupdate" value="steden2.php">  
<!-- end meta info -->**

Je kan de gebruiker makkelijk naar deze pagina’s sturen met de **header** functie:

**if ( $insert AND $\_POST["afterinsert"]** > **"" )** *header***("Location: ../"** . **$\_POST["afterinsert"] )**;  
**if ( $update AND $\_POST["afterupdate"]** > **"" )** *header***("Location: ../"** . **$\_POST["afterupdate"] )**;

* **Kijk het pad na**: de voorbeeldcode staat in het script **save.php** dat zich in de map **lib** bevindt. Daarom staan er **twee puntjes en een schuine streep ( ../ )**, om eerst naar het hogergelegen niveau te gaan, waar het script **steden2.php** staat.

## Redirect na een error

Als het opslaan niet lukt, kan je de gebruiker ook redirecten naar een foutpagina (bv. **error.php**).

Hetzelfde kan je doen wanneer de **sanitization** of **validatie** mislukt. Maar in dat geval toon je waarschijnlijk beter het formulier zelf, met vermelding van de fouten.

De fouten kan je opslaan in een **sessievariabele** ( bv. **$\_SESSION[‘errors’][‘img\_title’] = “Veld is te lang”** ) en toevoegen aan de **dataset** waarmee je je template samenvoegt. In de **template** van je formulier moet je ook de nodige elementen voorzien om de fouten in te tonen.