Ontwikkeling

Bij de aanschaf van onze Autobedrijf & Voertuigen plugin ontvang je een standaard thema, deze kan je als child-theme naast je hoofd thema installeren. Hierdoor kan je je hoofdthema up-to-date houden en verlies je geen instellingen van onze plugin.

Binnen dit thema werk je aan de ontwikkeling van de betreffende pagina's, zoals het overzicht en de detailpagina van de voertuigen. Je schrijft zelf de HTML, CSS, PHP en eventueel extra Javascript om de pagina's vorm te geven.

> Je kan deze pagina's niet bewerken vanuit WordPress, ook niet met page builders zoals Elementor.

Daarnaast worden er veel velden gebruikt in de voorbeeldcode, het kan zijn dat er voor de versie van jouw plugin andere velden in de code gebruikt moet worden. Wegens de doorontwikkeling van verschillende partijen hebben wij een versie 3.x ontwikkeld, in deze versie wordt meer informatie verwerkt zoals informatie over elektrische voertuigen.

> Alle velden voor de versie van jouw plugin vind je onder Velden.

Templates

Het standaard thema is opgedeeld in verschillende templates.

Template	Beschrijving
voertuig/**archive.php**	Wordt door WordPress geladen op de archiefpagina van de voertuigen.
voertuig/**search.php**	Bevat het zoekformulier op de archiefpagina. Aparte template i.v.m. live zoeken.
voertuig/**loop.php**	Bevat de resultaten op de archiefpagina. Aparte template i.v.m. live zoeken.
voertuig/**item.php**	Compacte weergave van een voertuig, gebruikt vanuit *loop.php* om resultaten te renderen.
voertuig/**none.php**	Wordt ingeladen vanuit *archive.php* als er geen zoekresultaten zijn.

Template	Beschrijving
voertuig/**single.php**	Wordt door WordPress geladen wanneer een enkel voertuig wordt bekeken.

Om het template *voertuig/item.php* te renderen vanuit een ander template kan het volgende gebruikt worden:
`<?= Voertuig::template('item'); ?>`

Naast de template bestanden worden er standaard de volgende bestanden automatisch ingeladen: autobedrijfvoertuigen*/functions.php* en *voertuig/functions.php.* De eerst genoemde om code van toepassing op alle onderdelen van de plugin te plaatsen en de tweede functions is specifiek voor de voertuigen.

> WordPress laadt de archive, single en de functions in omdat we via de geavanceerde instellingen van de plugin het pad aangeven naar de templates. Als je het thema anders op wil bouwen moet je daarom de paden aanpassen.

Globals

Binnen de loop worden zonder zoekopdracht alle voertuigen opgehaald en op het moment dat de bezoeker een zoekopdracht instelt worden hier de voertuigen getoond die voldoen aan die zoekopdracht. In deze loop is het huidige voertuig in de iteratie beschikbaar in de variabele `$voertuig`. Het is over het algemeen niet nodig om deze variabele expliciet als `global` te declareren, de plugin zal dit voor zijn rekening nemen. Met deze variabele kan je informatie van het voertuig tonen in de item.php.

> Let op: Wanneer er geen gebruik gemaakt wordt van een template is de globale variabele `$voertuig` leeg. We raden daarom het gebruik van templates aan.

Caching

We raden aan om een WordPress caching plugin te gebruiken om het laden van de site te versnellen. Voor het ophalen van de data voor alle voertuigen zijn veel database query's nodig, wat alles bij elkaar enkele tientallen milliseconden in beslag kan nemen.

Om dit te versnellen, raden we aan om templates waarin gegevens van een voertuig worden opgevraagd te cachen. Bijvoorbeeld binnen de loop tijdens het weergeven van de voertuigen:
`<?= Voertuig::template('item')->cache(); ?>`

> Templates worden gecached op basis van de naam en ID van het huidige voertuig.

Weergave van velden

Het kan voorkomen dat er specifieke eisen zijn om velden op een bepaalde manier weer te geven, soms afhankelijk van andere velden en soms omdat je een prijs als een prijs wil formatten. Om te voorkomen dat deze logica overal door het thema opnieuw geschreven wordt biedt de plugin een aantal manieren om dit op een prettige manier te laten verlopen, waarbij alles op een centrale plek aangepast kan worden.

Een veld uit onze plugin is een object en kan naast het terug geven van zijn waarde nog veel meer informatie terug geven. Zo kunnen de volgende methods worden gebruikt:

```
$voertuig->algemeen->merk; // Het volledige object voor het veld 'merk'
$voertuig->algemeen->merk->render(); // Het merk van het voertuig na transformaties
$voertuig->algemeen->merk->value(); // De waarde van het veld merk
$voertuig->algemeen->merk->label(); // Label van het veld merk
$voertuig->algemeen->merk->is('BMW'); // Is het merk BMW?
$voertuig->algemeen->merk->is('BMW', 'Audi'); // Is het merk BMW of Audi?
$voertuig->algemeen->merk->isnt('BMW'); // Is het merk niet BMW?
$voertuig->algemeen->merk->isEmpty(); // Heeft het merk geen waarde?
$voertuig->algemeen->merk->hasValue(); // Heeft het merk een waarde?
```

> **TIP:** Doordat onze plugin gebruik maakt van Magic Methods (__toString()) kunnen we voor het tonen van het merk ook simpel het volgende schrijven `<?= $voertuig->merk; ?>`. Voor een simpele echo is `->render()` niet nodig.

Veld types

Elk veld heeft een bepaald type zodat het weergeven ervan automatisch gebeurt. Ieder van de types heeft verder specifieke methodes als hulp. Voor de onderstaande veldtypes worden letterlijke voorbeelden in de objecten gezet, er zijn echter geen velden die letterlijk ‘list’, ‘boolean’, ‘date’, ‘dateTime’, ‘integer’ of ‘double’ heten. Deze velden referen naar velden met dat type waarde. De velden die geschikt zijn voor jouw verisevind je allemaal terug onder het hoofdstuk 'Velden'.

Arrays

```
$voertuig->list->render($separator = ', ');
$voertuig->list->count();
$voertuig->list->all();
$voertuig->list->first();
$voertuig->list->last();
$voertuig->list->implode($glue);
```

In plaats van komma gescheiden waardes kan ook gebruik worden gemaakt van een lijst. Verander hiervoor de formatter in `list`, waarbij als standaard een `<ul>` wordt gebruikt. Optioneel kunnen twee argumenten worden opgegeven om de list template en item template aan te passen, in het voorbeeld hieronder zijn de standaard templates opgegeven:
`$voertuig->list->formatter('list')->render('<ul>#{items}</ul>', '<li>#{item}</li>');`

Booleans

```
CustomPost\Formatter\BooleanFormatter::setTrue($true);
CustomPost\Formatter\BooleanFormatter::setFalse($false);
CustomPost\Formatter\BooleanFormatter::setTrueFalse($true, $false);
```

```
$voertuig->boolean->render($true= null, $false = null);
$voertuig->boolean->boolValue();
$voertuig->boolean->isTrue();
$voertuig->boolean->isFalse();
```

Datums

Voor datums wordt gebruik gemaakt van de Carbon library. Voor het weergeven wordt de PHP functie strftime gebruikt en niet de Carbon API, vanwege localization ondersteuning in `strftime`. Naast de volledige Carbon API zijn de volgende methods beschikbaar:

```
$voertuig->date->render($format = '%e %B %Y'); // Velden zonder tijd
$voertuig->dateTime->render($format = '%e %B %Y, %R'); // Velden met tijd
$voertuig->date->date(); // Alias voor `value`
```

Integers

```
$voertuig->integer->render($decimals = 0, $decimal = ',', $thousands = '.');
$voertuig->integer->n('deur', 'deuren'); // Kiest de juiste vorm
```

Doubles

```
$voertuig->double->render($decimals = 2, $decimal = ',', $thousands = '.');
```

Bedragen

```
CustomPost\Formatter\MoneyFormatter::setCurrency($currency);
CustomPost\Formatter\MoneyFormatter::setDecimals($decimals);
```

```
$voertuig->prijs->verkoopprijs_particulier->render($currency = null, $suffix = null, $decimals = null);
```

Weergave aanpassen

Voor ieder veld kan echter de weergave ervan worden aangepast. Als voorbeeld zetten we het veld *merk* om in hoofdletters:

```
Voertuig::formatter('algemeen.merk', function ($value)
{
    return strtoupper($value);
});
```

> Standaard wordt het veld *merk* al slim omgezet in de juiste vorm. Bovenstaande geldt als voorbeeld en kan beter worden bereikt met CSS.

Afhankelijkheden van andere velden

In sommige gevallen is voor het bepalen van de weergave meer informatie over het voertuig nodig. Het volgende voorbeeld toont hoe de prijs wordt weergegeven in combinatie met het btw bedrag:

```
Voertuig::formatter('prijs', function($value, $field)
{
    return $field->formatter('money').' '.$field->post()->bpm_bedrag;
});
```

Waar het eerste argument dus de waarde van het veld is, is het tweede argument het veld zelf. het voertuig is van daaruit op te vragen a.d.h.v. `$field->post()`.

In bovenstaande voorbeeld wordt ook duidelijk dat we terug kunnen vallen op de valuta weergave door gebruik te maken van de formatter 'money' : `$field->formatter('money')`, zoals het *koopprijs* veld origineel zou worden weergegeven.

>

> De volgende formatters zijn beschikbaar:*string, bool, date, datetime, double, float, integer, array, list, money*.

Formatter class

Als niet alleen de waarde van het veld moet worden getransitioneerd maar bijvoorbeeld ook het label moet worden afgeleid van een ander veld, kan een class worden gebruikt om de weergave te beinvloeden:

```
class MerkFormatter extends CustomPost\Formatter\DelegateFormatter
{
    public function label()
    {
        return ucfirst($this->post()->voertuigsoort->render()) ?: parent::label();
    }

    public function render()
    {
        return $this->value.' '.$this->post()->model->render();
    }
}
Voertuig::formatter('algemeen.merk', 'MerkFormatter');
```

De waarde van het veld is beschikbaar via `$this->value` en het voertuig zelf via `$this->post()`.

Label aanpassen

In het vorige voorbeeld wordt een class gebruikt om het label af te leiden van een ander veld. In veel gevallen zal het label echter niet afhankelijk zijn van een waarde en kan een simpelere constructie worden gebruikt:

```
Voertuig::label('prijs.verkoopprijs_particulier', 'Prijs');
```

Values aanpassen

Als een veld een beperkt aantal values heeft waarvan de weergave moet worden aangepast, gebruik dan de volgende constructie:

```
Voertuig::values('geschiedenis.tellerstand_eenheid', array(
    'K' => 'km',
    'M' => 'mijl',
));
```

Hierdoor zal het veld *tellerstand_eenheid* met waarde *K* worden weergegeven als *km*, enz. Waardes waarvoor geen transformatie is opgegeven zullen zonder wijziging worden weergegeven.

> Bovenstaande mapping voor *tellerstand_eenheid* is standaard aanwezig.

Listings

Vaak moet een lijstje van bepaalde eigenschappen worden gegeven. Daarbij moet worden gecontroleerd of iedere eigenschap wel een waarde heeft om zo geen lege rijen te tonen. Dit is waar listings gebuikt kunnen worden:

```
<?= $voertuig->listing('prijs.verkoopprijs_particulier', 'algemeen.datum_deel_1', 'geschiedenis.tellerstand'); ?>
```

Bovenstaande snippet geeft een lijst met de opgegeven velden in een lijst, met hun bijbehorende labels en alleen als de velden een waarde hebben. Optioneel kan er een titel worden weergegeven welke voor de lijst komt te staan:

```
<?= $voertuig->listing(...)
   ->title('<h3>Eigenschappen</h3>'); ?>
```

Lege listings

Wanneer geen van de velden een waarde heeft wordt de listing in zijn geheel niet weergegeven. Om toch bijvoorbeeld een melding te geven kan het volgende worden gebruikt:

```
<?= $voertuig->listing(...)
   ->ifEmpty('<p>Geen eigenschappen bekend</p>'); ?>
```

> Als op deze manier een melding wordt ingevoegd wordt ook de titel getoond.

Templates

Standaard wordt een definition list `<dl><dt>#{label}</dt><dd>#{value}</dd></dl>` structuur gebruikt om de velden weer te geven. Dit kan echter naar wens worden aangepast, als volgt:

```
<?= $voertuig->listing(...)
   ->before('<div>')->between('<hr>')->after('</div>')
   ->item('<div>#{label}: #{value}</div>'); ?>
```

Template globaal aanpassen

Binnen een site zal vaak dezelfde opmaak gebruikt worden. Het is mogelijk om dit globaal te wijzigen:

```
CustomPost\Formatter\Listing::templates(array(
    'title' => '<h2>#{title}</h2>',
));
```

De volgende templates zijn aan te passen:

Template	Standaard waarde
title	`'#{title}'`
before	`'<dl>'`
item	`'<dt>#{label}</dt><dd>#{value}</dd>'`
between	`''`
after	`'</dl>'`
empty	`'#{message}'`

Specifiek per veld

Soms moet een veld op een andere manier worden weergegeven dan de overige velden. Om dit te bereiken kan het `item` template hiervoor specifiek per veld worden opgegeven:

```
<?= Voertuig::listing('accessoires')
    ->before('<ul>')->after('</ul>')
    ->item('<li>#{value}</li>')
    ->field('accessoires', '#{items}')
?>
```

Naast een template string kan ook een render callback functie worden opgegeven, waarbij in de callback alles voor een normaal veld handmatig moet worden gedaan (alles wat normaal via het `item` template wordt bereikt moet dus nu worden herhaald).

Aangepaste labels

We hebben al gezien hoe het label van een veld kan worden gewijzigd. Omdat een aangepast label soms alleen nodig is op bepaalde plekken is het mogelijk om dit per listing aan te passen:

```
<?= $voertuig->listing(array(
    'prijs.verkoopprijs_particulier',
```

```
    'algemeen.datum_deel_1',
    'geschiedenis.tellerstand' => 'Kilometerstand',
)); ?>
```

Voor *lease.leasevorm* zal vervolgens binnen de listing het label *Lease* worden gebruikt, de overige velden behouden hun originele label.

Extra data meegeven

Soms is het nodig om per eigenschap een class op te geven, voor bijvoorbeeld een icoontje. Geef hiervoor in plaats van een enkel label een array op met de benodigde data.

```
<?= $voertuig->listing(array(
    'geschiedenis.tellerstand' => array('label' => 'Kilometerstand', 'class' => 'mileage', 'suffix' => ''),
    'algemeen.aantal_deuren' => array('label' => 'Kamers', 'class' => 'doors', 'suffix' => array(' deur', '
deuren')),
))
->item(
    '<dt><i class="fa fa-#{data.class}"></i> #{label}</dt>
     <dd>#{value}#{data.suffix}</dd>'
); ?>
```

Met de key `'label'` kan het label als voorheen worden overschreven, de overige keys zijn beschikbaar via `#{data.key}`.

> De array notatie voor `'suffix'` geeft de twee waardes op voor enkelvoud en meervoud.

Render argumenten opgeven

Geef aangepaste argumenten op om een veld te renderen door in de lijst met data de key `'render'` op te geven als array van de argumenten.

```
<?= $voertuig->listing(array(
    'geschiedenis.invoerdatum' => array('render' => '%B %Y'),
)); ?>
```

Formatter type aanpassen

Het type formatter kan worden aangepast door in de lijst met data de key `'formatter'` op te geven met de naam van de gewenste formatter.

```
<?= $voertuig->listing(array(
    'geschiedenis.datum_binnenkomst' => array('render' => '%B %Y'),
)); ?>
```

Item template aanpassen

We hebben al gezien hoe `$listing->field($name, $template)` kan worden gebruikt om de template specifiek voor een item aan te passen. Voor gemak kan het ook via de data key `'template'` worden bereikt. In het volgende voorbeeld gebruiken we al het bovenstaande om een lijst weer te geven waarbij alle waardes van het veld *voertuig.voorzieningen* geintegreerd zijn in de lijst zelf:

```
<?= $voertuig->listing(array(
    'geschiedenis.tellerstand',
    'algemeen.aantal_deuren',
    'nl.zoekaccessoires' => array('template' => '#{value}', 'formatter' => 'list', 'render' => '#{items}')
))
->before('<ul>')->after('</ul>')
->item('<li>#{value}</li>')
?>
```

Macros

Vaak worden stukken code vaker gebruikt en is het gewenst om logica uit de templates te houden. Hiervoor kunnen zogenaamde macros worden toegevoegd, welke dan als method op ieder voertuig kunnen worden aangeroepen.

```
/* voertuig/functions.php */
Voertuig::macro('similar', function ($voertuig, $amount = 3)
{
    return Voertuig::search(array($voertuig->merk), array('posts_per_page' => $amount));
});

/* voertuig/single.php */
// Resultaat wordt gecached, herhaaldelijk opvragen geeft identiek resultaat
$similar = $voertuig->similar;

// Voert macro iedere keer uit
$similar = $voertuig->similar();
```

```
// Geef aangepaste argumenten op
$similar = $voertuig->similar(10);
```

Customs

Macros kunnen ook een veld als resultaat geven. In dat geval kan de macro net als ieder ander veld worden gebruikt, het is dus een soort alias naar een ander veld. Zo is het *prijs* veld wat standaard beschikbaar is ook een alias naar *verkoopprijs_particulier*, *verkoopprijs_handel* of *meeneemprijs*, afhankelijk van welke beschikbaar is. Het voordeel hiervan is dat hierdoor `$voertuig->prijs->label()` automatisch aangeeft of het om een koop- of huurprijs gaat.

Bij het gebruik van een macro is het echter van belang dat er altijd een veld als resultaat wordt gegeven, `null` of alleen een waarde als resultaat zal fouten opleveren. Om dit te voorkomen kunnen er custom velden worden gedefinieerd.

In het onderstaande custom wordt direct uitgelezen of het vermogen in PK of in kW gecommuniceerd moet worden. Ook is het gebruik van het veld *vermogen_motor_eenheid* nu overbodig omdat je handmatig de juiste eenheid plaatst.

```
Voertuig::custom('vermogen', function ($voertuig) {
  /* Standaard waarden */
    $vermogen = '';
    $label = '';

    if ($voertuig->motor->vermogen_motor_pk->hasValue()) {
//// Vul de standaard waarden met informatie over het vermogen in PK
        $vermogen = $voertuig->motor->vermogen_motor_pk;
        $label = 'PK\'s'
    } elseif ($voertuig->motor->vermogen_motor_kw->hasValue()) {
//// Vul de standaard waarden met informatie over het vermogen in kW
        $vermogen = $voertuig->motor->vermogen_motor_kw;
        $label = 'kW'
    };

// Controleer of in ieder geval het vermogen een nieuwe value heeft gekregen, zo niet: return null
    return (strlen($vermogen) > 0 ? $vermogen.' '.$label : null);
}, 'string');
```

Het laatste argument, in bovenstaande voorbeeld `'string'`, bepaalt het type van het veld. Omdat `'string'` standaard is kan het in dit geval worden weggelaten.

Querying

Het opvragen van voertuigen is ondersteund via `Voertuig::query()`, waarbij via het eerste optionele argument de `WP_Query` argumenten kunnen worden opgegeven.
`$voertuigen = Voertuig::query(array('posts_per_page' => 5));`

Het resultaat is een Collection van posts.

Sorteren

Sorteer de resultaten door gebruik te maken van de `orderby` optie van `WP_Query.` De beschikbare sorteervolgordes zijn `asc` voor oplopend (A-Z) en `desc` voor aflopend (Z-A) en kunnen worden opgegeven door het veld en de ordering te scheiden met een dubbele punt. Meerdere velden kunnen worden opgegeven door ze te scheiden met een komma.
`$voertuigen = Voertuig::order('orderby', 'verkoopprijs_particulier:desc');`

Zoeken

Om op voertuigen met specifieke eisen te zoeken, kan `Voertuig::where()` worden gebruikt. Vervolgens kan de zoekquery worden opgebouwd via chained method calls. Het is mogelijk om de aanroep tot `where()` weg te laten, alle ondersteunde calls zijn ook direct onder `Voertuig` beschikbaar.

```
// Alle voertuigen van merk Volkswagen
$voertuigen = Voertuig::where('algemeen.merk', 'Volkswagen');
$voertuigen = Voertuig::where('algemeen.merk', '!=', 'Volkswagen');

// Uitgebreid zoeken met meer eisen
$voertuigen = Voertuig::where('algemeen.merk', 'is not null')->where('prijs.verkoopprijs_particulier', '>',
100000)->where('geschiedenis.datum_binnenkomst', '> DATE_SUB(NOW(), INTERVAL 1 WEEK)');
```

De volgende methodes zijn beschikbaar om zoektermen op te geven:

Methode	Beschrijving
`where('veld', '=', $value)`	Geef op dat `veld` een bepaalde waarde heeft. Operator `=` is standaard en kan worden weggelaten.
`whereBetween('veld', $min, $max)`	Geef op dat `veld` tussen `$min` en `$max` moet zijn.

Methode	Beschrijving
`whereNull('veld')`	Geef op dat `veld` geen waarde mag hebben.
`whereNonNull('veld')`	Geef op dat `veld` een waarde moet hebben.
`whereSql('veld', 'sql')`	Gebruik een ruwe SQL clause als zoekopdracht.
`matching($voertuig->field, $operator = '=')`	Zoek op voertuigen met dezelfde waarde als `$voertuig->field`, optioneel kan een operator worden opgegeven.
`without($voertuig, ...)`	Neemt het huidige `$voertuig` niet mee in de resultaten.

De `where` functie ondersteunt alle SQL operators:

Operator	Code
IS NULL	`where('veld', 'null')`
	`where('veld', 'is null')`
IS NOT NULL	`where('veld', 'not null')`
	`where('veld', 'is not null')`
BETWEEN	`where('veld', 'between', $min, $max)`
SQL	`where('veld', 'sql', '> NOW()')`
	`where('veld', '> NOW()')`
=, <>, <, >, ...	`where('veld', $operator, $value)`

Daarnaast zijn de volgende methodes beschikbaar voor het opgeven van WordPress query argumenten, om bijvoorbeeld het aantal resultaten en de sorteervolgorde aan te passen:

Methode	Beschrijving
`amount($n)`	Beperkt het resultaat tot `$n` voertuigen.
`all()`	Geef op dat alle voertuigen opgehaald moeten worden.
`order($field, $direction = 'asc')`	Geef sorteervolgorde op.
`ascending($field)`	Geef oplopende sorteervolgorde op.
`descending($field)`	Geef aflopende sorteervolgorde op.
`arg($key, $value)`	Geef direct een WordPress query argument op.
`args($args)`	Geef direct WordPress query argumenten op.

Nesting

Zoektermen kunnen worden genest om te schakelen tussen `and` en `or`. Standaard moet aan alle zoektermen worden voldaan, maar door eerst `push('or')` en vervolgens `pop()` aan te roepen hoeft alleen aan minimaal één van de zoektermen ertussenin voldaan te worden.

```
// Zoek nieuwe voertuigen die minder dan een week op de site staan, met een merk en waar het model een Golf of een Phaeton is
$voertuigen = Voertuig::where('algemeen.merk', 'is not null'),
  ->push('or')
    ->where('algemeen.model', '=', 'Golf')->where('algemeen.model', '=', 'Phaeton')
  ->pop()
    ->where('geschiedenis.datum_binnenkomst', '> DATE_SUB(NOW(), INTERVAL 1 WEEK)'),
));
```

Soortgelijke voertuigen zoeken

Het is eenvoudig gemaakt om voor het huidige voertuig, dus bijvoorbeeld binnen de template file `single.php`, te zoeken naar voertuigen met hetzelfde merk. Geef hiertoe alleen het veld in:

```
// Zoek drie voertuigen met hetzelfde merk als het huidige voertuig, maar goedkoper
$voertuigen = Voertuig::matching($voertuig->algemeen->merk)->matching($voertuig->prijs->verkoopprijs_particulier, '<')->amount(3);
```

Voertuigen tellen

Soms is het alleen het aantal resultaten nodig.
`$aantal = Voertuig::whereSql('datum_binnenkomst', '> DATE_SUB(NOW(), INTERVAL 1 WEEK)')->total();`

Performance

We ontwikkelen de plugins altijd door, dit heeft er in geresulteerd dat het laden van de plugin tot 2.5 keer sneller is dan voor voorheen. Dit is bereikt door het laden van de module uit te stellen tot het moment waarop het als eerste gebruikt wordt, om zo onnodig werk te voorkomen. Dit vereist dat ook het definieren van formatters en overige acties met betrekking tot velden moet worden uitgesteld tot wanneer een component wordt geladen. Hiervoor moet het manueel includen van de verschillende functions.php bestanden uit *active-theme/functions.php* worden verwijderd zodat deze bestanden vervolgens on-demand worden ingeladen. Let hierbij op dat in deze bestanden de acties, die stonden gepland in de `init` hook, direct uitgevoerd dienen te worden, `init` is op dit moment namelijk al uitgevoerd geweest.

> Wanneer de manuele includes niet verwijderd worden zal alles blijven werken, echter blijft de snelheidswinst dan ook grotendeels uit.

Zoekvelden

Eerder heb je over het instellen van zoekvelden gelezen in het admin gedeelte van onze plugin, in dit gedeelte van de documentatie leggen we uit hoe je de ingestelde zoekvelden kunt koppelen aan de code in de `search.php` en daarnaast leggen we uit hoe je de zoekvelden kunt gebruiken.

Zoekvelden koppelen

In het admin gedeelte van de plugin kan je zoekvelden aanmaken, de naam die je het zoekveld daar geeft zorgt voor de koppeling. Een aantal voorbeelden.

Merk: `<?php Voertuig::form()->dropdown('merk'); ?>`
Model: `<?php Voertuig::form()->dropdown('model'); ?>`
Type: `<?php Voertuig::form()->dropdown('type'); ?>`
Carrosserie: `<?php Voertuig::form()->dropdown('carrosserie'); ?>`

Let er daarom goed op dat je de naam een duidelijke, maar simpele naam geeft.

> Een naam met spaties, hoofdletters of bijzondere karakters kan voor onverwachte problemen zorgen.

Type zoekvelden

Zoekvelden aangemaakt vanuit de admin kunnen op verschillende manieren worden weergegeven.

Dropdown

`<?php Voertuig::form()->dropdown('naam', $options = []); ?>`

De volgende mogelijkheden heb je in de options array:

Optie	Type	Standaard	Beschrijving
`'emptyLabel'`	string	`''`	Label van optie om zoekveld leeg te laten.
`'showCounts'`	bool	`false`	Weergave van aantal resultaten per optie.
`'hideWhenNoResults'`	bool	`false`	Verberg een optie als er 0 resultaten zijn.

Optie	Type	Standaard	Beschrijving
'hideWhenNotExists'	bool	false	Verberg een optie als de waarde nooit resultaten zal geven.
'select'	array	[]	Extra HTML attributen voor de <select>.
'option'	array	[]	Extra HTML attributen voor <option> elementen.
'emptyOption'	array	[]	Extra HTML attributen voor <option> elementen, specifiek voor lege optie.

```
<?php Voertuig::form()->minDropdown('naam', $options = []); ?>
<?php Voertuig::form()->maxDropdown('naam', $options = []); ?>
<?php Voertuig::form()->minMaxDropdown('naam', $options = []); ?>
```

Optie	Type	Standaard	Beschrijving
'emptyLabelMin'	string	''	Label van optie om zoekveld leeg te laten voor minimum dropdown.
'emptyLabelMax'	string	''	Label van optie om zoekveld leeg te laten voor maximum dropdown.

Checkboxes & Radio buttons

```
<?php Voertuig::form()->checkboxes('naam', $options = []); ?>
<?php Voertuig::form()->radios('naam', $options = []); ?>
```

Optie	Type	Standaard	Beschrijving
'showCounts'	bool	false	Weergave van aantal resultaten per optie.
'hideWhenNoResults'	bool	false	Verberg een optie als er 0 resultaten zijn.
'hideWhenNotExists'	bool	false	Verberg een optie als de waarde nooit resultaten zal geven.
'label'	array	[]	Extra HTML attributen voor <label> elementen.
'input'	array	[]	Extra HTML attributen voor <input> elementen.

Optie	Type	Standaard	Beschrijving
'count'	array	[]	Extra HTML attributen voor de om aantal resultaten in te tonen.

Verborgen input

```
<?php Voertuig::form()->hidden('naam', $options = []); ?>
```

Optie	Type	Standaard	Beschrijving
'input'	array	[]	Extra HTML attributen voor de <input>.

> Extra HTML attributen kunnen worden opgegeven via de opties, zoals in de tabellen staat beschreven. Voor nodes die specifiek zijn voor een bepaalde optie worden de placeholders {{label}}, {{value}} en {{count}} ingevuld met de label, waarde en aantal resultaten voor de optie. Dit kan bijvoorbeeld worden gebruikt om extra data-attributen op te geven welke door JavaScript plugins worden gebruikt om de weergave van de zoekvelden aan te passen.

Vrije input

Zoekvelden hebben een vast aantal opties en accepteren geen aangepaste waardes. Om bijvoorbeeld te kunnen zoeken op merk en model moet handmatig de query worden aangepast. Als HTML fragment kan het volgende worden gebruikt:

```
<input type="text" name="merk_model" value="<?= Voertuig::form()->value('merk_model'); ?>">
```

Om vervolgens het merk_model op de query toe te passen moet een event listener worden toegevoegd om de query te wijzigen:

```
Voertuig::listen('search.before', function($form, $query)
{
    if ($search = $form->value('merk_model'))
    {
        $query->compare('algemeen.merk', 'like', '%'.str_replace(' ', '%', $search).'%');
        $query->compare('algemeen.model', 'like', '%'.str_replace(' ', '%', $search).'%');
    }
});
```

De query kan in alle mogelijke constructies worden beinvloed:

Operator	Code
IS NULL	`$query->isNull('veld')`
IS NOT NULL	`$query->isNotNull('veld')`
BETWEEN	`$query->between('veld', $min, $max)`
SQL	`$query->sql('veld', '> NOW()')`
=, <>, <, >, ...	`$query->compare('veld', $operator, $value)`

Een geneste groep kan worden begonnen met `$query->push('and|or')`, waarna opvolgende declaraties binnen die groep worden toegepast. Een groep moet worden afgesloten met `$query->pop()`.

Nieuwe types toevoegen

Voor meer controle over de HTML kunnen nieuwe types worden gemaakt. De standaard types zijn makkelijk aanpasbaar door ze te subclassen en voor de gewenste methods een aangepaste implementatie te schrijven.

```
class CustomRenderer extends CustomPost\Search\Renderers\Dropdown
{
    // ...
}


Voertuig::form()->render('naam', new CustomRenderer($options = []));
```

Door de subclass te registreren onder een bepaalde identifier kan het nieuwe/aangepaste type eenvoudiger worden gebruikt:

```
CustomPost\Search\Form::register('identifier', 'CustomRenderer');


Voertuig::form()->identifier('naam', $options = []);
```

Daarnaast is het ook mogelijk om bestaande types aan te passen zodat je meer vrijheid hebt in de vormgeving. In het onderstaande voorbeeld kan je zien hoe je een `span` toe kan voegen om het label binnen `checkboxes` en `radios`, hiermee kan je bijvoorbeeld met een `::before` of `::after` de input vervangen voor een mooiere vormgeving.

```
// Checkboxes
use \CustomPost\Search\Option;
```

```
class CustomCheckboxes extends \CustomPost\Search\Renderers\Checkboxes {
    protected function renderLabel(Option $option) {
        $option->setLabel('<span>' . $option->getLabel() . '</span>');
        parent::renderLabel($option);
    }
}

// Radios
class CustomRadios extends \CustomPost\Search\Renderers\Radios {
    protected function renderLabel(Option $option) {
        $option->setLabel('<span>' . $option->getLabel() . '</span>');
        parent::renderLabel($option);
    }
}
CustomPost\Search\Form::register('checkboxes', new CustomCheckboxes());
CustomPost\Search\Form::register('radios', new CustomRadios());
```

> Voor meer informatie wordt aangeraden om contact met ons op te nemen. De standaard types bieden ruime mogelijkheden om via CSS aan te passen, alleen in uitzonderlijke gevallen zal een aangepast type noodzakelijk zijn.

Zoekopties vanuit Javascript

In enkele gevallen kan het handig zijn om alle opties van een zoekveld in Javascript beschikbaar te hebben, bijvoorbeeld om afhankelijke velden handmatig bij te kunnen werken als er geen gebruik wordt gemaakt van live bijwerken. Een geneste structuur, vooral geschikt voor een zoekveld met afhankelijkheden, kan in JSON formaat worden verkregen via:

```
Voertuig::form()->options('merk')->json();
```

Dit geeft een JSON object waarbij alle opties onderverdeeld zijn onder de afhankelijke waarde waar de optie bij hoort. Als voorbeeld een zoekveld model waarvan de waardes afhankelijk zijn van het geselecteerde merk:

```
{
   "Audi": [
      {"value": "a5", "label": "A5"},
```

```
        {"value": "r8_spyder", "label": "R8 Spyder"}
    ],
    "BMW": [
        {"value": "x4", "label": "X4"}
    ]
}
```

Het is echter ook mogelijk om alle opties in een vlakke lijst te krijgen:

```
Voertuig::form()->options('merk')->flat()->json();
```

```
[
    {"value": "a5", "label": "A5", "parents": {"merk": "Audi"}},
    {"value": "r8_spyder", "label": "R8 Spyder", "parents": {"merk": "Audi"}},
    {"value": "x4", "label": "X4", "parents": {"merk": "BMW"}}
]
```

> In plaats van de opties in JSON representatie kan ook een PHP array worden opgevraagd via `get()` in plaats van `json()`.

Voorbeeld

Als voorbeeld een manier om een dropdown voor het zoekveld model bij te werken met de opties behorende bij een bepaald merk, nadat een ander merk is geselecteerd. We geven hierbij alle opties in bovenstaande geneste structuur op via een HTML data-attribuut, zodat dit vervolgens vanuit Javascript beschikbaar is.

```
<?php Voertuig::form()->dropdown('model', array(
    'emptyLabel' => 'Model',
    'select' => array(
        'data-options' => Voertuig::form()->options('model')->json(),
    ),
)); ?>
```

Het onderstaande is de minimaal benodigde Javascript om de dropdown bij te werken:

```
$('#merk').change(function() {
    var merk = $(this), model = $('#model');

    model.empty().append($('<option></option>').attr('value', '').text('Model'));
```

```
    $.each(model.data('options')[merk.val()] || [], function(option) {
        model.append($('<option></option>').attr('value', option.value).text(option.label));
    });
});
```

Resultaten sorteren

Binnen het Admin gedeelte van de documentatie heb je kunnen lezen hoe je sorteer opties kunt toevoegen, dit doe namelijk via de geavanceerde instellingen van de plugin. Om de bezoeker van de website de mogelijkheid te geven om te switchen tussen de door jou ingestelde opties moeten we een `orderby` veld vullen met opties.

Een lijst van de ingestelde sorteeropties is beschikbaar via `Voertuig::form()->orderings()` en kan bijvoorbeeld worden gebuikt om een dropdown weer te geven:

```
<select name="orderby">
    <?php foreach (Voertuig::form()->orderings() as $ordering): ?>
        <option value="<?= $ordering['orderby'] ?>" <?= selected($ordering['current']); ?>>
            <?= $ordering['label'] ?>
        </option>
    <?php endforeach; ?>
</select>
```

Als je in plaats van een dropdown buttons wil gebruiken kan bijvoorbeeld een verborgen input worden ingevoegd waarvan de waarde met JavaScript wordt aangepast:

```
<input type="hidden" name="orderby" id="orderby" value="<?= Voertuig::form()->ordering(); ?>">

<?php foreach (Voertuig::form()->orderings() as $ordering): ?>
    <a class="orderby" data-orderby="<?= $ordering['orderby'] ?>"><?= $ordering['label'] ?></a>
<?php endforeach; ?>
```

```
$(document).on('click', 'a.orderby', function() {
    var orderby = $(this).data('orderby'), current = $('#orderby').val();

    if (current !== orderby) {
        $('#orderby').val(orderby).trigger('change');
    }
```

```
}
```

> In de vormgeving ben je vrij, zolang je maar de `name` `orderby` mee stuurt met de zoekopdracht. De plugin sorteert vervolgens de resultaten.

Hulpmiddelen

De plugin bevat ingebouwde hulpmiddelen die je kunnen helpen bij het ontwikkelen van je thema, hier lees je meer over deze hulpmiddelen.

Voertuigen overslaan

Het kan voorkomen dat je bepaalde voertuigen niet wilt importeren, bijvoorbeeld als ze van een merk zijn die je niet op je eigen site wil publiceren. Dit kan worden bereikt door een filter toe te voegen welke bepaalt of het voertuig moet worden overgeslagen.

In het onderstaande voorbeeld worden voertuigen van het merk Renault overgeslagen:

```
Voertuig::filter('updater.skip', function($skip, $voertuig)
{
    // Skip dit voertuig wanneer het merk 'Renault' is
    if ($voertuig->algemeen->merk == 'Renault') {
        return true;
    }

    return $skip;
});
```

De bovenstaande code kan je toevoegen binnen het onderstaande bestand:
`/wp-content/themes/*active-theme*/autobedrijfvoertuigen/functions.php`

Het mechanisme werkt via de WordPress `add_filter` API, het eerste argument `$skip` is standaard `false`. Want zonder dit filter willen we immers alle voertuigen uit de feed importeren. Dit is de reden dat we `return true` doen op het moment dat een voertuig aan onze eisen voldoet.

Post Collections

Ingebouwde query resultaten zijn van het type `QueryCollection`, een gebruiksvriendelijke interface in plaats van het standaard WordPress `WP_Query` object. Dit biedt een aantal voordelen, zo zijn alle methods uit Illuminate's Collection class beschikbaar, en blijven ook alle properties en methods van `WP_Query` direct beschikbaar.

```
$voertuigen = Voertuig::search(...);


$voertuigen->count(); // Aantal voertuigen in resultaat, rekening houdende met pagination
$voertuigen->total(); // Totaal aantal gevonden resultaten, zonder pagination
$voertuigen->has(); // Of er resultaten zijn
$voertuigen->isEmtpty();
$voertuigen->first();
$voertuigen->last();
$voertuigen->random();
$voertuigen->sample(3); // Collection van 3 random voertuigen
```

> Het `WP_Query` object is beschikbaar via `$voertuigen->getQuery()`, al zijn alle properties en methods dus ook beschikbaar op de Collection zelf.

The loop

De Collection biedt een PHP iterator om de WordPress loop constructie niet handmatig uit te hoeven schrijven:

```
while ($voertuigen->next()):
    // ...
endwhile;
```

Dit is equivalent aan de loop constructie:

```
while ($voertuigen->have_posts()): $voertuigen->the_post();
    // ...
endwhile;
wp_reset_postdata();
```

Media

Voor ieder voertuig kan de bijbehorende media eenvoudig worden opgevraagd d.m.v. `$voertuig->media()` met eventueel een argument om query opties in te geven:

```
$media = $voertuig->media(array('posts_per_page' => 3));
```

Ook dit levert een Collection op.

Groepen

Vanuit jouw VMS wordt media onderverdeeld in een aantal groepen. Alle media in een bepaalde groep kan eenvoudig worden opgevraagd, als volgt:

Methode	Ggroepen
`$voertuig->afbeeldingen()`	Gekoppelde afbeeldingen
`$voertuig->documenten()`	Gekoppelde documenten

Voor iedere andere combinatie van groepen kan `$voertuig->groep(...)` worden gebruikt, met als eerste argument een groep of array van groepen.

Caching

Iedere keer als een van bovenstaande methods wordt aangeroepen zal opnieuw het resultaat uit de database worden opgevraagd. Om dit te voorkomen zijn de methods ook beschikbaar als properties, waarbij alleen bij het eerste gebruik de resultaten zullen worden opgehaald.

```
foreach ($voertuig->afbeeldingen as $afbeelding):
    // ...
endforeach;
```

Doordat de `QueryCollection` wordt gebruikt, werkt bovenstaande voorbeeld impliciet de loop af en zal achteraf automatisch de postdata worden herstelt.

Dit voorbeeld gebruikt een `foreach` loop in plaats van `while` zoals bij de Collections documentatie. Dit is omdat hiermee een nieuwe loop gestart wordt, waarbinnen `$voertuig` niet langer beschikbaar is. Bij gebruik van `while ($voertuig->afbeeldingen->next())` zal na de eerste afbeelding `$voertuig` dus `null` zijn.

Overig

Code	Omschrijving
`$voertuig->prijs`	Alias voor `verkoopprijs_particulier`, `verkoopprijs_handel` of `meeneemprijs`.

Javascript

Vanuit de plugin is standaard ondersteuning om het zoeken interactief te maken. Het meegeleverde thema bevat enkele Javascript bestanden, in dit gedeelte van de plugin willen we duidelijk maken hoe deze kunnen worden gebruikt en wat alle opties zijn.

Template	Beschrijving
js/voertuig/**archive.js**	Geregistreerd als `voertuig-archive` en te enqueuen vanuit *voertuig/archive.php*.
js/voertuig/**single.js**	Geregistreerd als `voertuig-single` en te enqueuen vanuit *voertuig/single.php*.

Live bijwerken

Een van de voornaamste features van de plugin is het direct bijwerken van resultaten nadat een zoekveld van waarde is veranderd. Het principe is eenvoudig: vanuit Javascript worden alle wijzigingen opgemerkt en wordt het zoekformulier via AJAX verzonden, waarna het wordt verwerkt op de server. Deze stuurt een JSON response met de templates die vernieuwd moeten worden terug, zodat de nieuwe templates vervolgens in de DOM worden ingevoegd. Ook pagination links worden onderschept en verwerkt op dezelfde manier.

Een bijkomend voordeel van deze methode is dat een lange, lelijke en onduidelijke query string wordt voorkomen, doordat het formuleer wordt geëncodeerd in een hash welke aan de URL wordt toegevoegd. Hierdoor blijft de browsergeschiedenis intact en wordt bij het vernieuwen van de pagina de gewenste zoekopdracht weer uitgevoerd.

De plugin wordt ingeladen door `custompost-liveform` toe te voegen als script dependency, waarna initialisatie als volgt gebeurt:

```
var form = new CustomPost.LiveForm(options);

// Event handlers toevoegen

form.init();
```

Optie	Standaard	Beschrijving
`'container'`	`'#entity-search-form'`	DOM element of selector voor `<form>` container.
`'els'`	*Zie Templates*	Bepaalt welke templates moeten worden bijgewerkt in welke containers.
`'pageLinkEls'`	`'.pagination a'`	Selector voor pagination links.

Templates

Voor het bijwerken van de pagina worden de benodigde templates op de server gerendered en wordt de parent DOM vervangen met de nieuwe content. Via de optie `'els'` kan worden opgegeven welke templates moeten worden bijgewerkt en de selector van de parent element, met een Javacript object met als key de naam van de template en als value de selector:

```
{
   search: '#entity-search',
   loop: '#entity-results'
}
```

Vanuit voertuig/archive.php moeten deze templates dus als volgt worden gebruikt, waarbij de wrapper alleen de template mag bevatten:

```
<div id="entity-search">
   <?= Voertuig::template('search'); ?>
</div>
```

Er kunnen indien nodig dus nog meer templates worden ingeladen en vervangen, door ze in `'els'` op te geven. De template met zoekvelden wordt bijgewerkt om zo het aantal resultaten per optie bij te werken.

Events

Om bepaalde acties uit te voeren wanneer specifieke events optreden kunnen event listeners worden toegevoegd. Zo moeten bijvoorbeeld Javascript widgets welke van toepassing zijn op elementen in de te vervangen templates iedere keer opnieuw worden geinitialiseerd, omdat de volledige DOM structuur vervangen wordt.

Event	Argumenten	Beschrijving
`'load'`	`data|null`	Uitgevoerd bij initialisatie en nadat bijwerken is voltooid.
`'updated'`	`data`	Uitgevoerd nadat bijwerken is voltooid.
`'refresh'`		Uitgevoerd wanneer een refresh wordt gestart.
`'before:replace'`	`data`	Uitgevoerd net voordat de templates worden ingevoegd in hun DOM parents.
`'after:replace'`	`data`	Uitgevoerd direct nadat de templates zijn ingevoegd.
`'request:data'`	`data`, `request`	Geeft de mogelijkheid om extra data aan de request toe te voegen door `data` argument aan te passen.
`'change:page'`		Uitgevoerd wanneer van pagina wordt gewisseld.

De API voor het gebruik van events is als volgt:

```
form.on('event', function(e, ...) {
    // ...
});

form.off('event');
```

DOM Event delegation

We raden aan om DOM event listeners aan document toe te voegen en vervolgens te filteren op het gewenste element. Hierdoor hoeft de listener maar één keer worden toegevoegd en niet bij iedere 'load' op de vernieuwde elementen:

```
$(document).on('click', 'a', function(e) {
    // ...
});
```

Pagination

Links om van pagina te wisselen moeten ook worden onderschept om een pagina reload te voorkomen. Met de optie `'pageLinkEls'` kan de selector voor pagination links worden opgegeven,

waarbij de plugin ook pagina wijzigingen via AJAX in zal laden. Ook wordt het event `'change:page'` gegenereerd om een actie te ondernemen bij het wijzigen van de pagina.

Infinite scrollen

In plaats van pagination is het mogelijk om meer resultaten achteraf in te voegen, mogelijk automatisch wanneer de gebruiker het einde van de resultaten bereikt. Voeg hiertoe alleen een link in naar de volgende pagina, met de class `infinite-results` of een van de parents moet deze class toegekend krijgen. Extra voertuigen zullen vervolgens achteraan in `options.infinite.appendTo` (met standaardwaarde `#entity-items`) worden ingevoegd, waarbij per voertuig de events `before:append` en `after:append` worden uitgevoerd met het DOM element van het voertuig als extra parameter. Vanuit deze events kunnen bijvoorbeeld animaties worden opgegeven of overige widgets worden geinitialiseerd. Alle overige events zijn als volgt:

Event	Argumenten	Beschrijving
`'before:append'`	`item`	Uitgevoerd net voordat een voertuig wordt toegevoegd in de DOM.
`'after:append'`	`item`	Uitgevoerd direct nadat een voertuig is ingevoegd.
`'appended'`	`data`	Uitgevoerd direct nadat alle voertuigen zijn ingevoegd.
`'infinite:end'`	`data`	Uitgevoerd wanneer alle resultaten zijn ingeladen.
`'append:page'`		Uitgevoerd wanneer meer resultaten worden opgehaald.

De events `before:replace`, `after:replace`, `change:page` en `updated` komen te vervallen en verder zal het `load` event alleen bij initialisatie worden uitgevoerd en niet meer voor nieuwe resultaten.

> Als voor een voertuig ongeldige HTML wordt gegenereert zal er geen DOM element aangemaakt kunnen worden, met als gevolg dat er vanuit *liveform.js* een fout optreedt. Ga in dat geval na of alle elementen wel worden gesloten en of dit in de correcte volgorde gebeurt.

Voor ieder resultaat zal de template `item` worden gerendered en worden ingevoegd in de DOM. De template per voertuig kan worden aangepast door `options.infinite.template` te wijzigen in het gewenste template. Om infinite scrollen te bereiken moet een klik-event op de pagination link handmatig worden uitgevoerd, zodat automatisch de juiste pagina aan extra resultaten geladen wordt. De pagination link moet verplicht zijn opgenomen in de DOM maar kan uiteraard verborgen worden indien gewenst. Voorbeeldcode om dit te bereiken is als volgt:

```
var scrolled = false, $window = $(window);

$window.scroll(function() { scrolled = true; });
```

```
setInterval(function() {
    if (scrolled && !form.isLoading() && $window.scrollTop() + $window.height() >= form.$el.offset().top +
form.$el.height()) {
        $('.infinite-results a, a.infinite-results').click();
    }
    scrolled = false;
}, 100);
```

Laad indicatie

Tijdens het wachten op resultaten van de server krijgt de container `<form>` de class `search-loading` toegewezen. Vanuit CSS kan dit bijvoorbeeld worden gebruikt om laad indicaties op het juiste moment zichtbaar te maken.

Widgets

Er zijn standaard een aantal widgets beschikbaar om de functionaliteit van zoekvelden uit te breiden.

Sliders

Om een slider weer te geven worden de opties van een dropdown gebruikt, er wordt een jQuery slider widget toegevoegd in de DOM welke in sync blijft met de dropdown. Hierdoor is het eenvoudig om de dropdown te blijven tonen voor bijv. mobiele devices en de slider alleen zichtbaar te maken voor desktops, aan de hand van responsive CSS rules. Voor gebruik van deze widget, voeg `custompost-slider` toe als script dependency en initialiseer de widget als volgt:

```
new CustomPost.Slider(options);
```

Optie	Standaard	Beschrijving
`'container'`	*Verplicht*	DOM element of selector voor label container.
`'type'`	`'min'`	Bepaalt het type slider, `'min'` of `'max'`.
`'select'`	`'select'`	DOM element of selector voor gekoppelde dropdown.

Optie	Standaard	Beschrijving
`'emptyLabel'`	`null`	Label wanneer geen minimum/maximum is ingesteld, standaard bepaald a.d.h.v. lege optie in gekoppelde dropdown.
`'optionLabelAttribute'`	`null`	Biedt de mogelijkheid om HTML label te gebruiken zoals opgegeven in data attribuut van de `<option>`. Standaard wordt simpelweg de optie's label gebruikt.
`'sliderOptions'`	`{}`	Geef extra opties op voor de jQuery slider widget.

Bij initialisatie wordt een extra wrapper toegevoegd aan de opgegeven container, met de volgende opmaak:

```
<div class="slider-container">
    <div class="slider-values">
        <div class="value low|high"></div>
    </div>
    <div class="slider"></div>
</div>
```

De class van `div.value` is `low` wanneer `'type' = 'min'`, voor `'type' = 'max'` wordt de class `high` gebruikt. Hierin wordt de huidige waarde van het veld weergegeven. Tijdens verplaatsen van de handle krijgt de container de class `tracking` toegewezen, en `tracking-low` / `tracking-high` voor respectievelijk `min` / `max` types.

Range sliders

Naast sliders met een enkele handle kan ook een dubbele handle worden gebruikt, om zowel een minimum als maximum op te geven. Range sliders moeten los van sliders worden opgegeven met `custompost-rangeslider` als dependency.

```
new CustomPost.RangeSlider(options);
```

Optie	Standaard	Beschrijving
`'container'`	*Verplicht*	DOM element of selector voor label container.
`'minSelect'`	`'select[id$="-min"]'`	DOM element of selector voor gekoppelde dropdown voor minimum waarde.

Optie	Standaard	Beschrijving
'maxSelect'	'select[id$="-max"]'	DOM element of selector voor gekoppelde dropdown voor maximum waarde.
'minLabel'	null	Label wanneer geen minimum is ingesteld, standaard bepaald a.d.h.v. lege optie in gekoppelde dropdown.
'maxLabel'	null	Label wanneer geen maximum is ingesteld, standaard bepaald a.d.h.v. lege optie in gekoppelde dropdown.
'optionLabelAttribute'	null	Biedt de mogelijkheid om HTML label te gebruiken zoals opgegeven in data attribuut van de <option>. Standaard wordt simpelweg de optie's label gebruikt.
'sliderOptions'	{}	Geef extra opties op voor de jQuery slider widget.

Nu zullen zowel div.value.low als div.value.high beschikbaar zijn, de eerste toont het label van de minimum waarde en de laatste bevat het label van de maximum waarde.

Show more

Deze widget is bedoeld in combinatie met een lijst van checkboxes/radios, om in eerste instantie slechts een beperkt aantal opties te tonen en de resterende opties pas later weer te geven.

De hoeveelheid altijd zichtbare opties kan worden ingesteld op een vast aantal, of afgeleid worden van de alfabetische volgorde van de opties. Zo kunnen primaire opties op alfabet voor secundaire opties worden geplaatst, waardoor automatisch de grens tussen primair en secundair bepaald kan worden.

De widget wordt ingeladen door custompost-showmore op te geven als script dependency, hoeft verder alleen te worden geinitialiseerd:

```
new CustomPost.ShowMore(options);
```

Optie	Standaard	Beschrijving
'container'	*Verplicht*	jQuery element of DOM selector voor label container.
'name'	*Verplicht*	Unieke naam van de lijst.
'moreText'	'More...'	Tekst welke wordt weergegeven als niet alle opties worden weergegeven.
'lessText'	'Less'	Tekst welke wordt weergegeven als wel alle opties worden weergegeven.

Optie	Standaard	Beschrijving
`'defaultAmount'`	`0`	Het aantal te tonen opties als primair, of `0` om af te leiden van de opties.

De naam is verplicht in verband met het onthouden van de state, waarop wordt teruggevallen bij herinitialisatie tijdens live bijwerken zodat de widget in de juiste state hersteld wordt.

Wat betreft HTML structuur wordt een lijst van labels verwacht, genest in een container. Bij initialisatie wordt een `<a class="show-more-toggle">` element toegevoegd aan het einde van de container, waarvan de click events de secundaire opties tonen/verbergen. Als alleen de primaire opties worden weergegeven wordt de tekst `options.moreText` weergegeven en heeft dit element de class `less`, bij tonen van de secundaire opties verandert de tekst in `options.lessText` en de class in `more`.

Revisie #14
Aangemaakt: 23 november 2020 13:47:10 door Tussendoor
Bijgewerkt: 20 september 2024 08:10:34 door Tussendoor