|  |  |
| --- | --- |
|  | Webshop |
|  |  |
| 6/20/2019 | Documentatie beroepsproduct |
|  | Door: Rick Wubs (626010 Klas: ITA-1DA-D (SEB A DT) Docent: M. van der Maas |

Inhoudsopgave

[Opdrachtomschrijving 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153314)

[Globale flow (tussen de schermen) 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153315)

[Schermontwerpen 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153316)

[Schermafbeelding 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153317)

[Mogelijke acties 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153318)

[Mogelijke gevolgen bij de acties 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153319)

[Softwarebeschrijving 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153320)

[Globale softwarestructuur 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153321)

[Tabs 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153322)

[Functies 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153323)

[Programma flow (samenhang tussen methodes) 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153324)

[Methoden en variabelen 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153325)

[Namen, Parameters, Returnwaarden, Toelichting (tabel) 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153326)

[Events 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153327)

[Samenhang schermontwerp en variabelen 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153328)

[Testplan 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153329)

[Testcase 1 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153330)

[Testcase 2 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153331)

[Testcase 3 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153332)

[Testcase 4 3](file:///C:\Users\rickw\OneDrive\Documents\git\han\SEB\GP\Beroepsproduct%201\Beroepsproduct%20GP.docx#_Toc4153333)

# Casusbeschrijving

De casus betreft het ontwerpen en implementeren van een webshop, welke geschreven dient te worden met behulp van PHP, HTML en CSS.

De volgende eisen zijn gesteld:

1. Er wordt gebruikt gemaakt van templates.
2. De webshop dient werkend te zijn en uitgevoerd te zijn met PHP-code. Er dient een koppeling met een database tot stand zijn gebracht.
3. Er dienen tenminste de volgende pagina’s aanwezig te zijn:
   1. Homepage
   2. Productenoverzicht
   3. Individuele productpagina
   4. Winkelwagen
   5. Registreren
   6. Over Ons
4. Er dient tenminste de volgende functionaliteit aanwezig te zijn:
   1. Een bezoeker kan producten zoeken
   2. Een bezoeker kan op categorie filteren
   3. Een ingelogde gebruiker kan artikelen in de winkelwagen leggen
   4. Een ingelogde gebruiker kan artikelen uit de winkelwagen verwijderen
   5. Bij het betalen van de artikelen wordt de voorraad bijgewerkt.
5. Verder dienen er minimaal 3 HTML/CSS-pagina’s te zijn, uitgewerkt als template.
6. De dynamische PHP-pagina’s dienen dezelfde styling als de templates te hebben.
7. Tot slot mogen er geen frameworks gebruikt worden.

Van een aantal van de bovengenoemde eisen is afgeweken; de beslissingen hieromtrent zijn in de relevante secties toegelicht.

# Gebruik

Om de website te gebruiken, kan de code gedownload worden van <https://github.com/rickwubs/webshop>. Om deze goed te gebruiken, dienen echter wel een aantal zaken eerst geregeld te worden:

1. XAMPP dient geïnstalleerd te worden (op deze wijze wordt compatibiliteit gegarandeerd)
2. Er dient een aanpassing te geschieden aan de configuratie van de Apache webserver: de DocumentRoot dient te worden aangepast. De aanpassing is beschreven in de sectie Globale Flow -> Apache.
3. Er dient in phpmyadmin een database genaamd ‘webshop’ aangemaakt te worden, met utf8\_general\_ci encoding. Vervolgens dient het meegeleverde .sql script geïmporteerd te worden.
4. Er dient een gebruiker ‘rickwubs’ met wachtwoord ‘x8SJ36Dm’ aangemaakt te worden, ofwel de instellingen in de Config file van de website dienen aangepast te worden. Zonder deze aanpassingen komt er geen verbinding tot stand.
5. Bij mogelijke problemen kan de poort van apache gewijzigd worden naar localhost:8000.

# Globale flow

Bij het ontwerp van de webshop is rekening gehouden met beveiliging. De aangeleverde database bevatte bijvoorbeeld in plain-tekst opgeslagen wachtwoorden. Tevens was er geen rekening gehouden met “onthoud-mij” functionaliteit of cross-site scripting attacks. Vanwege deze tekortkomingen is besloten de database hieromtrent aan te passen teneinde dergelijke functionaliteit te garanderen.

Verder is vanwege de veiligheid gekozen om de daadwerkelijke inhoud van de pagina’s niet toegankelijk te maken. Het toestaan van toegang tot de pagina’s kan beveiligings-issues met zich meebrengen, niet in het minst omdat op deze manier de gehele structuur van de webshop zichtbaar is.

## MVC

Binnen het ontwerp van de webshop is gekozen voor een MVC (Model-View-Controller) structuur. Dit zorgt ervoor dat er een duidelijke deling is tussen de verschillende vormen van functionaliteit en voorkomt “spaghetti-code”.

Het Model zorgt voor de koppeling van het object in de vorm van een klasse en de betreffende data uit de database.

De Controller zorgt voor de afhandeling van de business-logica. Er wordt hier gekozen welke pagina onder welke condities wordt getoond; of er eventuele error-berichten worden getoond, en regelt de koppeling tussen het Model en de View.

De View houdt zich puur bezig met de presentatie van een model, op basis van de koppeling door de Controller. Hier bevindt zich zo min mogelijk logica. De aanwezige logica richt zich alleen op front-end validatie en presentatie logica.

## Htaccess

RewriteEngine On  
RewriteBase /  
RewriteCond %{REQUEST\_FILENAME} !-f  
RewriteCond %{REQUEST\_FILENAME} !-d  
RewriteRule ^(.\*)$ index.php?$1 [L,QSA]

Er is een .htaccess bestand opgenomen. Hier zal kort worden uitgelegd wat de regels in dit bestand precies betekenen.

RewriteEngine On

Deze regel verteld de server om alle regels welke beginnen met Rewrite\* (tot de volgende RewriteEngine) als 1 logische groep te zien.

RewriteBase /

Deze regel definieert de basis vanuit waar alle relatieve referenties worden uitgevoerd. De “/” houdt in dit geval in dat de basis de public html route betreft.

RewriteCond %{REQUEST\_FILENAME} !-f

Deze regel betekent dat een conditie waar moet zijn voordat de volgende RewriteRule uitgevoerd wordt. {REQUEST\_FILENAME} is een variabele welke door de server wordt aangemaakt en welke de opgevraagde URL bevat. De “-f” flag betekent dat de conditie waar is als het eerste argument resolved naar een bestaande file. De “!” flipt deze beslissing, waardoor “!-f” nu waar is als het eerste argument **niet** naar een bestaande file verwijst.

RewriteCond %{REQUEST\_FILENAME} !-d

Deze regel is vergelijkbaar met bovenstaande regel, behalve dat er nu wordt gekeken naar valide mappen (door middel van de “-d” flag).

RewriteRule ^(.\*)$ index.php?$1 [L,QSA]

Deze regel geeft aan dat er gererouted dient te worden naar index.php. De [L] flag geeft aan dat er na deze regel geen andere regels uitgevoerd dienen te worden. De [QSA] flag zorgt ervoor dat query strings samengevoegd worden. Als voorbeeld:

1. Er is een request naar /products/13?naam=test
2. Zonder de [QSA] flag verwijst deze naar /product.php?product=13. De bestaande query string zal dus verwijderd worden.
3. Met de [QSA] flag verwijst deze naar /product.php?products=13&naam=test.

## Apache

Er is gekozen om gebruik te maken van een zogenoemde “front-controller”. Hiervoor is een wijziging in de Apache Webserver aangebracht (in de httpd.conf file):

DocumentRoot “C:/xampp/htdocs”  
<Directory “C:/xampp/htdocs”

Is aangepast naar het volgende:

DocumentRoot “C:/xampp/htdocs/public”  
<Directory “C:/xampp/htdocs/public”>

Dit zorgt ervoor dat de root folder van de server niet langer *htdocs* is, echter nu de *htdocs/public* folder. Hiervoor is gekozen zodat enkel de bestanden in deze *public* folder zichtbaar zijn voor buitenaf. De folders en bestanden in andere mappen buiten deze *public* folder kunnen gebruikt worden voor logica, door middel van de .htaccess rules en het instellen van de front controller en router.

## Front controller en Routing

De front controller is de pagina waarop alle requests binnenkomen. Deze regelt vervolgens de mapping naar andere bestanden en folders.

require\_once dirname(*\_\_DIR\_\_*) . '/vendor/autoload.php';

Hier wordt gebruik gemaakt van een autoloader, welke ervoor zorgt dat aangeroepen classes automatisch included worden. Dit versimpelt de aanroepen welke hiervoor gedaan dienen te worden en zorgt voor schonere code.

error\_reporting(*E\_ALL*);  
set\_error\_handler('Core\Error::errorHandler');  
set\_exception\_handler('Core\Error::exceptionHandler');

Dit zet de error reporting en geeft aan welke class moet zorgen voor de foutafhandeling.

session\_start();

Hier wordt de sessie gestart. Aangezien een request altijd initieel via de front controller binnenkomt, zijn we op deze wijze gegarandeerd dat er altijd een sessie gestart is.

$router = new Core\Router();

Deze regel geeft aan waar de router zich bevindt.

$router->add('', ['controller' => 'Home', 'action' => 'index']);  
$router->add('{controller}/{action}');  
$router->add('{controller}/{id:\d+}/{action}', ['controller' => 'Products', 'action' => 'description', 'id' => 'id']);  
$router->add('password/reset/{token:[\da-f]+}', ['controller' => 'Password', 'action' => 'reset']);  
$router->add('signup/activate/{token:[\da-f]+}', ['controller' => 'Signup', 'action' => 'activate']);  
$router->add('login', ['controller' => 'Login', 'action' => 'new']);  
$router->add('logout', ['controller' => 'Login', 'action' => 'destroy']);  
$router->add('signup', ['controller' => 'Signup', 'action' => 'new']);

Hier worden verschillende routes aangegeven welke door de router klasse worden gebruikt om een request (gebruik makend van de .htaccess rewrite rules om alle parameters uit de request te halen) de koppelen aan een specifieke methode binnen een klasse.

$router->dispatch($\_SERVER['QUERY\_STRING']);

Deze regel tot slot roept de dispatch functie aan binnen de router, gebruik makende van de query string request welke de server heeft ontvangen. Binnen deze functie wordt deze dan uitgesplitst naar een ‘controller’ en een ‘action’. Hier wordt de *controller* gemapt naar de folder welke zich in App/Controllers bevindt, en de *action* naar een specfieke methode binnen deze controller. Verder wordt in de Core/Controller klass (waar de overige controllers van overerven), gezorgd dat achter de *action* aanroep het woord ‘Action’ geplaatst wordt. Hierdoor wordt bijvoorbeeld een aanroep als products/new gemapt naar:

Bestand :App/Controllers/Products  
Functie: newAction

Hierdoor wordt gezorgd dat dergelijke functies alleen aangeroepen kunnen worden als deze *action* hier achter staat. Tevens wordt op deze manier gezorgd dat er niet een arbitrair request gedaan kan worden (er kan niet zomaar met de URL gerommeld worden waardoor een niet-valide functie uitgevoerd wordt).

# Template engine

Er is gebruik gemaakt van een template engine, genaamd Twig. Dit zorgt voor schonere en eenvoudiger code in de views. Het vervangt de <?php ?> tags welke normalerwijs nodig zijn. Tevens zorgt dit ervoor dat de bestandsnamen eenvoudig .html kunnen blijven en niet .php genoemd dienen te worden, welke ervoor zorgt dat de scheiding tussen business-logica en view-logica duidelijker gescheiden is. Dit zou opgelost kunnen worden door html SSI’s. Echter is op deze wijze tevens eenvoudig een bepaald object of array mee te geven aan de templates, zoals verderop beschreven.

Omdat het eigenlijk niet de bedoeling is dat er frameworks gebruikt worden (Al is dit een template engine en niet een framework), is er tevens een sectie pagina’s opgenomen welke hier geen gebruik van maakt. Deze pagina’s zijn te vinden in de folder App/Views/Home. Hier betreft het php files.

Ter vergelijking:

Met PHP code:

<?php if (isset($\_SESSION['user\_id'])) {?>  
<li class="nav-item">  
 <a href="/profile/show" class="nav-link">Profile</a>  
</li>  
<li class="nav-item">  
 <a href="/logout" class="nav-link">Log out</a>  
</li>  
<?php }?>  
<?php if (!isset($\_SESSION['user\_id'])) {?>  
<li class="nav-item">  
 <a href="/signup" class="nav-link">Sign up</a>  
</li>  
<li class="nav-item">  
 <a href="/login" class="nav-link">Log in</a>  
</li>  
<?php } ?>

Met Twig code:

{% if current\_user %}  
<li class="nav-item">  
 <a href="/profile/show" class="nav-link">Profile</a>  
</li>  
<li class="nav-item">  
 <a href="/logout" class="nav-link">Log out</a>  
</li>  
{% else %}  
<li class="nav-item">  
 <a href="/signup" class="nav-link">Sign up</a>  
</li>  
<li class="nav-item">  
 <a href="/login" class="nav-link">Log in</a>  
</li>  
{% endif %}

Tevens is hiertoe gekozen aangezien ik op mijn werk al bezig was om voor een klant een website met een login systeem te bouwen en ik hiervoor al Twig heb gebruikt. Gezien het feit dat dit een deeltijdstudie betreft waar een relevante werkplek een eis is en er gewerkt wordt met leerdoelen acht ik deze koppeling tussen werk en studie belangrijk, aangezien op deze manier de koppeling tussen beiden duidelijk aantoonbaar aanwezig is. De website is te vinden op <http://www.colombiapostcard.co>. Het eruit slopen van Twig zou relatief veel tijd kosten en in principe niet iets opleveren, omdat Twig code gewoon vervangen kan worden door PHP code (en andersom).

Tevens is door het gebruik van deze engine relatief makkelijk om bepaalde objecten of arrays mee te geven aan het template, welke hierin beschikbaar komen, zonder dat de betreffende klasse hier geinclude dient te worden. Dit zorgt voor een duidelijker scheiding tussen de controller logica en de view logica.

# Models

In deze sectie volgt een beschrijving van de gebruikte modellen.

## Product

Dit model maakt een Product object en zorgt ervoor dat de juiste gegevens uit de database worden opgehaald, aangepast, ingevoerd of verwijderd. De methoden zijn als volgt:

1. \_\_constructor
   1. Maakt het product object, en vult deze met attributen welke meegegeven worden als array.
2. getUser
   1. Maakt gebruik van de Auth klasse, en haalt hier een betreffende user vandaan.
3. Save
   1. Haalt de gegevens vanuit een $\_POST[], valideert deze gegevens en zorgt voor de opslag van de betreffende gegevens in de database.
4. Validate
   1. Valideert de gegevens uit het ingezonden formulier.
5. getAll
   1. Verkrijgt alle producten als associatief array uit de database.
6. getByCategory
   1. Verkrijgt alle producten als associatief array uit de database, waar de categorie gelijk is aan de opgegeven categorie in het filter menu.
7. Search
   1. Verkrijgt alle producten als associatief array uit de database, waar de zoekstring onderdeel is van de naam uit de database. (Gebruikmakend van wildcards).
8. getList
   1. Verkrijgt alle producten als associatief array uit de database, waarbij het id in de shopping cart zit.
9. findByMostRecent
   1. Verkrijgt het meest recent aangemaakte product.
10. findById
    1. Verkrijgt een enkel product als associatief array waarbij de id gelijk is aan het ingezonden id.
11. removeProduct
    1. Verwijderd een specifiek product uit de database waarbij het id gelijk is aan het opgegeven id.
12. doesNotExist
    1. Returned *true* als het product niet bestaat, anders *false*.
13. Payment
    1. Verminderd de hoeveelheid *stock* van het product met 1 wanneer er een betaling wordt gedaan.

## RememberedLogin

Dit model regelt zaken omtrent *onthoud\_mij* functionaliteit en bevat de volgende functies:

1. findByToken
   1. Vindt een onthouden login model bij een bepaald token. Hier wordt gebruik gemaakt van de Token uit de database, welke is aangemaakt en vervolgens gehashed omwille de beveiliging. (Een dergelijke login token staat in principe gelijk aan zowel authenticatie en autorisatie, waardoor deze niet in plain-tekst opgeslagen dient te worden ter beveilging.
2. getUser
   1. Maakt gebruik van de findByID functie uit de User klasse en geeft dit model terug.
3. hasExpired
   1. Bekijkt of de *onthoud\_mij* token nog valide is, of dat deze te lang geleden is aangemaakt.
4. Delete
   1. Verwijderd het *onthoud\_mij* token uit de database.

## User

Dit model regelt zaken omtrent User functionaliteit, is verantwoordelijk voor het maken van de User objecten en het ophalen van de relevante gegevens uit de database. Deze klasse bevat de volgende functies:

1. \_\_construct
   1. Maakt het user object, en vult deze met attributen welke meegegeven worden als array.
2. Save
   1. Valideert de ingezonden gegevens, maakt een gehashede versie van het wachtwoord, een nieuw gehashed activatie token en slaat deze op in de database.
3. Validate
   1. Valideert de gegevens welke door het formulier zijn meegestuurd.
4. emailExists
   1. Bekijkt of een gebruiker al bestaat, op basis van het emailadres. Maakt hiervoor gebruik van de findByEmail functie.
5. findByEmail
   1. Vindt een user model op basis van een meegegeven emailadres. Geeft deze terug als klasse.
6. Authenticate
   1. Vindt eerst een user op basis van email en stored deze in de user variabele. Als deze bestaat en deze zijn account geactiveerd heeft, wordt een password check uitgevoerd. Wanneer deze ook correct is bevonden wordt het user object teruggegeven.
7. findByID
   1. Vindt een user model in de database op basis van een meegegeven id. Deze wordt teruggegeven als class.
8. rememberLogin
   1. Onthoudt de login door middel van een gehashede token welke in de remember\_logins tabel wordt ingevoerd.
9. sendPasswordReset
   1. Vindt een user op basis van email, en als een user wordt teruggegeven roept dit de startPasswordReset functie aan. Als deze true teruggeeft, wordt een email verzonden door middel van de sendPasswordResetEmail functie.
10. startPasswordReset
    1. Begint de wachtwoord reset procedure door het invoeren van een gehashede token en verlooptijd in de users tabel.
11. sendPasswordResetEmail
    1. Verstuurt een email. Maakt hierbij gebruik van de View klasse en de Mail klasse. Voor de View wordt zoals reeds eerder vermeldt Twig gebruikt, voor de Mail klasse een Mailgun sandbox account. Hierdoor wordt vanuit de server daadwerkelijke emails verstuurd aan gebruikers (mits opgenomen in de accept-list van Mailgun). Erg handig om daadwerkelijke mailfunctionaliteiten te testen.
12. findByPasswordReset
    1. Vindt een user model op basis van een token welke in de email is verstuurd. Geeft een user model terug als deze bestaat, valide is en de verlooptijd niet is overschreden.
13. resetPassword
    1. Reset een wachtwoord, op in principe dezelfde wijze als dit in de save functie gebeurt.
14. sendActivationEmail
    1. Stuurt een activatie email met een activatie token. Pas wanneer de gebruiker zijn of haar emailadres via deze link heeft gevalideerd, wordt de gebruiker als actief bestempeld. Tot deze tijd kan er geen gebruik gemaakt worden van de login functionaliteiten.
15. Activate
    1. Activeert een gebruiker op basis van een token welke in de email is toegestuurd en zet deze gebruiker in de database als *actief*.
16. updateProfile
    1. Updatet een profiel, en veranderd alleen het wachtwoord als er een nieuw wachtwoord is ingevuld.

# Controllers

In deze sectie volgt een beschrijving van de gebruikte controllers.

1. Account
   1. validateEmailAction
      1. Valideert of een email beschikbaar is voor een nieuwe gebruiker. Aangezien het emailadres als login wordt gebruikt dient deze check uitgevoerd te worden alvorens een account aan te maken.
2. Authenticated
   1. Before
      1. Dit is een optioneel filter welke aangebracht kan worden. Deze functie wordt uitgevoerd voordat een andere functie wordt uitgevoerd en hier kan dus bepaalde functionaliteit, zoals het eerst dienen in te loggen, worden uitgevoerd. Verder niet geimplementeerd; deze is gebruikt voor de functionaliteit welke nodig was op mijn werk en is erin gelaten om aan te tonen hoe dit precies te werk kan gaan.
3. Home
   1. indexAction
      1. Zorgt ervoor dat een request naar home/index terechtkomt bij de view home/index.html en rendert deze template.
   2. aboutAction
      1. Zorgt ervoor dat een request naar home/about terechtkomt bij de view home/about.php. Hier is gekozen voor .php om de functionaliteit van Twig en PHP met elkaar te kunnen vergelijken en aan te tonen dat de PHP kennis tevens beheerst is, zoals reeds eerder besproken.
4. Items
   1. indexAction
      1. Nog over van de bouwfase. Deze is inmiddels wel gemapped naar de Product klasse en rendert de betreffende view.
   2. newAction
      1. Rendert een view op basis van een items/new request welke de items/new.html file toont.
   3. showAction
      1. Nog over van de bouwfase. Geeft alleen nog maar een echo weer.
5. Login
   1. newAction
      1. Rendert een template behorende bij de request /login en mapt deze naar /login/new.html.
   2. createAction
      1. Authenticeert een user op basis van een ingezonden formulier. Als deze *true* teruggeeft, wordt er ingelogd. Tevens wordt er dan een Flash bericht (een bericht welke slechts eenmaal wordt weergegeven) getoont. Vervolgens wordt er teruggekeerd naar de hiervoor opgevraagde pagina met behulp van de Auth::getReturnToPage functie. Wanneer de user *false* teruggeeft, wordt het login formulier nogmaals getoont en met behulp van de Twig template wordt het reeds eerder ingevulde veld email nogmaals automatisch ingevuld.
   3. destroyAction
      1. Logt een user uit door middel van de Auth::logout functie en toont vervolgens de login/show-logout-message pagina.
   4. showLogoutMessageAction
      1. Toont een Flash bericht en redirect vervolgens naar de homepagina.
6. Password
   1. forgotAction
      1. Rendert de password/forgot.html view.
   2. requestResetAction
      1. Stuurt een wachtwoord reset email en rendert vervolgens de view password/reset\_requested.html.
   3. resetAction
      1. verkrijgt de token uit de geleverde route en rendert vervolgens de password/reset.html view en geeft deze token mee.
   4. resetPasswordAction
      1. verkrijgt het token uit het ingezonden formulier. Hierna wordt er ofwel een user verkregen, ofwel een view password/token\_expired.html weergegeven. Wanneer de user verkregen is wordt de melding “password valid” weergegeven.
   5. getUserOrExit
      1. Verkrijgt een user op basis van de User::findByPasswordReset functie. Als deze een user teruggeeft wordt deze user teruggegeven door de functie, anders wordt de view password/token\_expired.html weergegeven en wordt de functie beeindigt.
7. Products
   1. indexAction
      1. Verkrijgt alle producten uit de database en toont deze vervolgens in de products/index.html view, waarbij deze producten worden meegegeven.
   2. filterAction
      1. Verkrijgt alle producten welke onder een bepaalde categorie vallen. Deze categorie wordt vanuit een formulier meegeven. Vervolgens wordt een view products/category weergeven en deze producten worden hieraan meegegeven.
   3. newAction
      1. Vindt een gebruiker op basis van de sessie en verkrijgt alle producten. Als de user een admin is, wordt de products/new.html view gegenereerd en deze producten hieraan meegeven. Wanneer een user geen admin is, wordt een Flash bericht weergegeven en wordt de user geredirected naar de login pagina (gemakshalve is er even vanuit gegaan dat de gebruiker dan een niet-ingelogde admin betreft, aangezien een niet-admin gebruiker geen links ziet om van deze functie gebruik te maken. Dus ofwel het is een niet-ingelogde admin, ofwel een gebruiker welke specifiek probeert op deze pagina te komen door te rommelen met de URL. In beide gevallen is verwijzen naar login de veiligste oplossing voor deze situatie).
   4. detailsAction
      1. Verkrijgt een product id uit de URL, vindt vervolgens het product behorende bij dit id en toont de products/details.html view waarbij dit product wordt meegegeven.
   5. createAction
      1. Er wordt een nieuw product aangemaakt op basis van het toegestuurde formulier. Als de hierop aangeroepen save functie werkt, wordt het meest recente product gevonden (aangezien dit het product betreft welke zojuist is aangemaakt) en wordt deze getoond. Als dit niet werkt, wordt dezelfde pagina getoond en het product wederom meegegeven aan deze view.
   6. editAction
      1. Verkrijgt een user op basis van de id uit de sessie. Verkrijgt vervolgens alle producten. Als de user admin is, wordt de products/edit.html view weergegeven en het product meegegeven. Zo niet, dan wordt er wederom naar de login pagina verwezen zoals bij de newAction.
   7. searchAction
      1. Verkrijgt een product op basis van een meegegeven zoekterm. Vervolgens wordt de products/search.html view weergegeven, waarbij de producten worden meegegeven.
   8. deleteAction
      1. Verwijderd een product uit de database. Deze optie wordt alleen weergeven als de user admin is.
   9. shoppingAction
      1. Maakt een *cart* sessie variabele aan als deze nog niet bestaat. Als het huidige id nog niet in de *cart* array zit, wordt deze hieraan toegevoegd. Vervolgens wordt een Flash bericht ‘Product added to cart’ getoont. Zo niet, dan wordt een Flash bericht ‘Product already in cart’ getoont. In beide gevallen wordt hierna naar de products/index actie verwezen.
   10. removeAction
       1. Doorzoekt de *cart* sessie variabele naar een een product met het betreffende id. Als deze bestaat, wordt deze variabele uit de sessie array gehaald. Er wordt dan een Flash message weergegeven en terugverwezen naar de shopping cart.
   11. cartAction
       1. Verkrijgt alle producten uit de shopping cart. Als deze niet leeg is, wordt voor alle producten de prijs opgeteld. Vervolgens worden zowel de producten als de totaalprijs meegegeven aan het products/cart.html view. Als de cart leeg is, wordt een Flash message weergegeven en doorverwezen naar de hom/index.html view.
   12. confirmationAction
       1. Verkrijgt alle producten uit de shopping cart. Toont vervolgens de products/confirmation.html view en geeft hierbij de producten mee.
   13. paymentAction
       1. Als een user is ingelogd, verkrijgt alle producten uit de shopping cart. Als de payment functie *true* teruggeeft, wordt de shopping cart leeggemaakt. Vervolgens wordt een Flash message ‘payment successful’ getoont. Als de functie *false* teruggeeft, wordt een Flash message ‘payment unsuccesful’ getoont. In beide gevallen wordt vervolgens de products/payment.html pagina getoont. Als een user wel *null* geruggeeft, wordt een Flash message weergegeven waarbij gevraagd wordt in te loggen en doorverwezen naar de login functie.
8. Profile
   1. Before
      1. Hier wordt gebruik gemaakt van de Authenticated klasse. Ten eerste wordt deze before functie aangeroepen, vervolgens wordt via de Auth klasse de user opgevraagd.
   2. showAction
      1. Toont de view profile/show.html waarbij de user wordt meegegeven.
   3. editAction
      1. Toont de profile/edit.html view waarbij de user wordt meegegeven.
   4. updateAction
      1. Updatet een user profiel op basis van een toegestuurd formulier. Als deze is geslaagd wordt een positieve Flash message weergegeven en doorgestuurd naar de profile/show actie. Als deze niet is geslaagd, wordt terugverwezen naar de profile/edit.html view en de user wederom meegegeven.
9. Signup
   1. newAction
      1. Rendert een view signup/new.html
   2. createAction
      1. Maakt een nieuwe user aan op basis van een toegestuurd formulier. Als de user is opgeslagen wordt vervolgens een activatie email toegestuurd en doorverwezen naar de /signup/success functionaliteit. Zo niet, dan wordt terugverwezen naar de signup/new.html view en de user meegegeven (hierdoor hoeft de gebruiker niet wederom alle velden in te vullen).
   3. successAction
      1. Rendert de signup/success.html view.
   4. activateAction
      1. Activeert een gebruiker op basis van de token in de URL en redirect vervolgens naar de /signup/actived action.
   5. activedAction
      1. Toont de signup/activated.html view.

# Overige klassen

Er zijn een aantal klassen welke zogenoemde *helper-klassen* zijn. Deze zijn niet gerelateerd aan een specifieke opgevraagde Route en staan zodoende niet in een aparte map als *Controllers*, *Model*, of *Views*. Het betreft de volgende klassen en functionaliteit:

1. Auth
   1. Login
      1. Genereert een nieuw sessie id en verwijderd de oude sessie. Zet vervolgens de $\_SESSION[‘user\_id’] gelijk aan de user id uit het User model. Als de *onthoud\_mij* functionaliteit is aangezet, wordt er tevens een *remember\_me* cookie aangemaakt en gelijk gezet aan de remember\_token van de gebruiker, voor de duratie welke is aangegeven in de expiry\_timestamp en over de gehele website toegankelijk gemaakt.
   2. Logout
      1. Maakt het volledige $\_SESSION array leeg. Vervolgens verkrijgt het alle cookie parameters en past deze aan door de tijd in het verleden te zetten. Hierdoor wordt de cookie verwijdert. Vervolgens wordt de sessie vernietigt en de forgetLogin functie aangeroepen.
   3. rememberRequestedPage
      1. Zet de sessie variabele *return\_to* gelijk aan de opgevraagde URI vanuit de server.
   4. getReturnToPage
      1. verkrijgt de URL vanuit de sessie *return\_to* variabele, of zet deze anders gelijk aan ‘/’, welke naar de homepagina verwijst.
   5. getUser
      1. Verkrijgt een user, ofwel op basis van de sessie variabele als deze bestaat, anders vanuit het login cookie.
   6. loginFromRememberCookie
      1. Verkrijgt een user op basis van de *remember\_me* cookie. Vervolgens wordt de user ingelogt en deze teruggegeven.
   7. forgetLogin
      1. Verkrijgt ofwel de waarde vanuit de *remember\_me* cookie, als deze bestaat, anders *false*. Wanneer de *remember\_me* cookie *true* teruggeeft, wordt er een remembered\_login teruggegeven op basis van de token vanuit de cookie. Als deze bestaat, wordt deze verwijderd. Vervolgens wordt de cookie ingesteld om in het verleden te verlopen, waardoor deze wordt verwijderd.
2. Config
   1. DB\_HOST
      1. Geeft de hostname aan welke in de database connectie gebruikt dient te worden.
   2. DB\_NAME
      1. Geeft de naam van de database aan welke gebruikt dient te worden.
   3. DB\_USER
      1. Geeft de gebruiker van de database aan welke gebruikt dient te worden.
   4. DB\_PASSWORD
      1. Geeft het wachtwoord van de gebruiker aan welke gebruikt dient te worden in de database connectie.
   5. SHOW\_ERRORS
      1. Een flag welke gebruik wordt om ofwel errors op de pagina weer te geven, ofwel naar een log file toe te schrijven.
   6. SECRET\_KEY
      1. De secret key welke gebruikt wordt om mee te hashen. Deze wordt bijvoorbeeld gebruikt door de Token in het hash algoritme.
   7. MAILGUN\_API\_KEY
      1. De API key voor het sandbox domein van Mailgun, waarmee test emails vanuit de server naar de gebruiker verstuurd kunnen worden.
   8. MAILGUN\_DOMAIN
      1. Het sandbox mailgun domein van waaruit de mails doorgestuurd worden naar de gebruiker, mits deze op de whitelist staat.
3. Flash
   1. SUCCESS
      1. Een type bericht, welke gebruikt kan worden voor styling.
   2. INFO
      1. Een type bericht, welke gebruikt kan worden voor styling.
   3. WARNING
      1. Een type bericht, welke gebruikt kan worden voor styling.
   4. addMessage
      1. Zet de sessie variabele *flash\_notifications* als deze nog niet bestaat en voegt het Flash bericht toe aan dit array.
   5. getMessage
      1. Verkrijgt een Flash bericht uit de *flash\_notifications* sessie variabele als deze bestaat. Maakt deze vervolgens leeg en geeft het bericht terug.
4. Mail
   1. Send
      1. Gebruikt de mailgun client om een bericht via de Mailgun API te versturen.
5. Token
   1. \_\_construct
      1. Maakt een nieuwe token aan op basis van een meegegeven waarde. Als geen waarde wordt meegeven, wordt een nieuw hexadecimaal token van 16 bytes aangemaakt.
   2. getValue
      1. Geeft de waarde van de token terug.
   3. getHash
      1. Maakt een gehashede waarde van een token waarde aan, op basis van de geheime sleutel uit Config::SECRET\_KEY

# Core

Dit zijn de core klassen welke de functionaliteit teweeg brengen. Tevens zien we hier de basis (abstracte) klassen Controller & Model en de klasse View, van waaruit de betreffende typen klassen overerven. Deze map bevat de volgende klassen en functionaliteiten:

1. Controller
   1. \_\_construct
      1. Maakt een nieuw controller object aan en zet de route\_params gelijk aan de meegegeven parameters.
   2. \_\_call
      1. Dit is een methode welke wordt uitgevoerd als een object is aangeroepen met een methode welke niet bestaat. Vervolgens wordt de methode gelijk gezet aan de meegegeven naam, met ‘Action’ hieraan toegevoegd. Dit zal voor alle aanroepen het geval zijn. Hierdoor garanderen we dat alleen de door ons besloten aanroepen uitgevoerd worden. Als deze methode bestaat wordt deze aangeroepen, anders wordt een nieuwe exceptie gethrowed. Tevens wordt hier eerst de before functie aangeroepen en na het uitvoeren van de opgevraagde functie de after functie.
   3. Before
      1. Een lege functie, welke door de kinderen van deze abstracte klasse kunnen worden geimplementeerd. Wordt uitgevoerd voor de standaard functie aanroep.
   4. After
      1. Een lege functie, welke door de kindereren van deze abstracte klasse kunnen worden geimplementeerd. Wordt uitgevoerd na de standaard functie aanroep.
   5. Redirect
      1. Stuurt een header met een meegegeven url, waardoor er een redirect plaatsvindt. Vervolgens wordt de functie afgesloten.
   6. requireLogin
      1. Checkt of de Auth::getUser functie *false* teruggeeft. Als dit het geval is, wordt een Flash message getoont, de opgevraagde pagina onthouden en doorverwezen naar de login pagina. Wanneer de gebruiker heeft ingelogd, wordt hierna weer terugverwezen naar de origineel aangevraagde pagina (maar dat gebeurt binnen de vervolgens aangeroepen functie, niet binnen deze).
2. Error
   1. errorHandler
      1. Converteert errors naar excepties.
   2. exceptionHandler
      1. Verkrijgt de exceptie code. Zorgt ervoor dat er 2 opties zijn: ofwel 404, ofwel 500. Zet vervolgens de http\_reponse\_code gelijk aan deze code. Als de Config::SHOW\_ERRORS op *true* staat, geeft de pagina de foutmelding in de browser weer. Als deze op *false* staat, wordt een nieuw log aangemaakt in de *logs* folder. Vervolgens wordt de stack trace hier naar toe geschreven en een pagina met de betreffende foutmeldingscode als view gegenereerd.
3. Model
   1. getDB
      1. First sets the $db variabel to null. If this is null, it creates a new PDO object with the provided values from the Config::$variable constants. Throws an error if it occurs, and finally returns the PDO object.
4. Router
   1. Add
      1. Herschrijft de route op basis van preg\_replace met behulp van regular expressions. Een regular expression wordt binnn /…/ gevat. De ‘^’ is de basis van de url. De ‘[‘ en ‘]’ geeft een groep aan. De ‘+’ geeft aan dat deze meerdere elementen mag bevatten. a-z- geeft aan dat hij kleine letters en hyphens mag bevatten. De ‘\/’ escaped de backslash, wat nodig is omdat het een speciaal karakter betreft. De ‘$’ geeft het einde van de string aan. De ‘(‘ en ‘)’ geven een *capture group* aan, waarmee een bepaalde expressie makkelijker uit de string gehaald kan worden. De ‘{‘ en ‘}’ worden gematched in de router. De ‘?P<\1>’ geeft bijvoorbeeld een identifier voor de capture group. Vervolgens slaat hij deze op in de routes variabele.
   2. getRoutes
      1. Geeft de routes variabele terug.
   3. Match
      1. Haalt alle parameters uit de route op en kijkt of de $route de $url matched. Als dit het geval is, en elke match een string betreft, wordt de $params variabele gelijkgesteld aan de eerder gedane $match. Vervolgens wordt er *true* teruggegeven. Anders wordt er *false* teruggegeven.
   4. Dispatch
      1. Haalt eerst de query string variabelen uit de url met behulp van de removeQueryStringVariables functie. Vervolgens wordt de url gematched. Vervolgens wordt de controller variabele gelijkgesteld aan de ‘controller’ uit de parameters variabele. Hierna wordt het geconverteerd met behulp van convertToStudlyCaps. Tot slot wordt hier eerst de juiste namespace voor gezet.
      2. Wanneer dit is gedaan wordt gekeken of deze controller bestaat als klasse. Als dit het geval is, wordt hier een nieuw object van gemaakt en worden de params meegegeven als parameters. Vervolgens wordt het action object aangemaakt op basis van de *action* variabele uit de params array. Deze worden geconverteerd met behulp van de convertToCamelCase functie.
      3. Als deze functie aanroepbaar is, wordt deze uitgevoerd.
      4. Als een van deze stappen niet slaagt, wordt een nieuwe exceptie gethrowed.
   5. convertToStudlyCaps
      1. Haalt hyphens uit de string, maakt de eerste letter een hoofdletter en haalt vervolgens spaties weg. Op deze manier matcht het de naamconventie van een klasse.
   6. convertToCamelCase
      1. Gebruikt convertToStudlyCaps, maar maakt vervolgens de eerste letter van het woord een kleine letter. Op deze manier matcht het de naamconventie van een methode.
   7. getParams
      1. geeft de params array terug.
   8. removeQueryStringVariables
      1. Haalt de ‘&’ tekens weg uit de url en geeft deze terug.
   9. getNamespace
      1. Zet de standaard namespace voor een {controller} aanroep gelijk aan ‘App\Controllers\’. Als de params array de key ‘namespace’ bevat, wordt de ‘namespace’ waarde uit het params array toegevoegd aan de namespace. Vervolgens wordt de namespace teruggegeven.
5. View
   1. Render
      1. Haalt de argumenten uit het meegegeven array, en voorkomt collisies. Vervolgens wordt de $file variabele gelijkgezet aan het /App/Views/$views bestand. Als dit een leesbaar bestand is, wordt deze ingevoegd door middel van require. Anders wordt er een exceptie gethrowed.
   2. renderTemplate
      1. Gebruikt de getTemplate functie en echo’d deze.
   3. getTemplate
      1. zet de $twig variabele naar null. Als deze null is, wordt met behulp van een Twig hulper functie de bijbehorende /App/Views geloaded. Vervolgens wordt hiervoor een nieuwe Twig Environment aangemaakt. Tot slot worden er twee globale variabelen (*user defined variabels*) toegevoegd, namelijk ‘current\_user’ welke de huidige gebruiker ophaalt via de Auth::getUser functie, en de ‘flash\_messages’, welke via Flash::getMessages() wordt opgehaald. Tot slot wordt een template gerendert en deze teruggegeven.

Zoals hierboven weergegeven en uit de code blijkt, is het zowel mogelijk een view met als zonder Twig te realiseren. Twig zorgt enkel voor schonere code in het gebruik van de View code, alsook het makkelijker kunnen meegeven van variabelen aan deze View render. Zoals reeds eerder aangegeven is er ook een PHP-only sectie meegegeven om aan te tonen hoe Twig-code zich verhoudt tot PHP-code.

# Vendor

Er zijn een aantal vendor libraries gebruikt om extra functionaliteit te garanderen. Zo is er gebruik gemaakt van guzzlehttp, welke gebruikt wordt in de mailgun api, welke gebruikt wordt in de Mail klasse.

Tevens is er gebruik gemaakt van een psr autoload library, welke klassen dynamisch inlaadt. De overige Vendor mappen en bestanden zijn ofwel required bij deze libraries, of nodig voor de composer package management en dependency resolution oplossing.

Tot slot zijn er nog een aantal andere script geinclude:

1. jquery: Deze helpt bij de front-end validatie van een formulier. Dit betreft extra functionaliteit en staat verder los van de back-end database en back-end php validatie.
2. Popper: Dit is onderdeel van bootstrap.
3. ShowHidePassword: dit zorgt voor een knop welke de zichtbaarheid van het wachtwoord ofwel aan, ofwel uit zet. Dit is wederom extra front-end functionaliteit.

# Conclusie

Hierboven is duidelijk de structuur en functie van de verscheidene onderdelen van de opgeleverde webshop beschreven. Tevens zijn keuzes onderbouwd. Hier wordt in deze sectie nogmaals op teruggekomen.

Ten eerste is gebruik gemaakt van een MVC patroon bij het ontwerpen van deze webshop. Dit patroon zorgt voor een duidelijke scheiding van de verschillende vormen van logica binnen de verscheidene klassen, waardoor het geheel overzichtelijker en makkelijker aanpasbaar wordt.

Vervolgens is gekozen om met een front controller te werken. Hiertoe is besloten aangezien dit de beveiliging van de webshop ten goede komt; we hebben op deze manier veel meer controle over waarop de gebruiker binnenkomt. Hierdoor wordt de *attack surface* kleiner.

Vervolgens is er gebruik gemaakt van Twig. Hiertoe is zoals reeds eerder aangegeven voor gekozen omdat ik via mijn werk met eenzelfde project bezig was en ik graag de inzichten van school hierop wilde loslaten, alsook andersom. Verder zorgt dit ervoor dat ik met schonere code beter de gewenste functionaliteit op kan leveren. Aangezien dit wellicht een punt kan zijn, heb ik hierom tevens een kleine sectie gemaakt waarbij geen gebruik gemaakt wordt van Twig, maar van “ruwe” php code. Hiermee wordt de equivalentie van deze twee opties aangetoond.

Tot slot ben ik ervan overtuigd met het geleverde aan te tonen de leerdoelen (het begrijpen en kunnen gebruiken van HTML, CSS, PHP en PDO in combinatie met een database, goed onder de knie te hebben. Tevens is gezorgd voor een ruime overdaad aan extra functionaliteiten, zoals onder andere:

1. Specifieke functionaliteit voor een gebruiker welke Admin is.
2. Beveiliging tegen XSS en XSRF aanvallen.
3. Gebruik van patronen zoals het POST-Redirect-GET patroon.
4. Veilige opslag van gegevens, onder andere door middel van gehashede wachtwoorden.
5. Het aanbieden van *onthoud\_me* functionaliteit door middel van sessie en cookies.
6. Het daadwerkelijk versturen van emails en het veilig afhandelen van de registratie- en activatiescenario’s.
7. Database validatie van de input.
8. PHP backend validatie van de input.
9. Front-end validatie van de input.
10. Geavanceerde error handling.
11. Het weergeven van Flash messages.
12. Het maken van een goed uitbreidbaar geheel.
13. Het aantonen zowel met “ruwe” php code als een template engine als Twig te kunnen werken.
14. Het aantonen zowel met “ruwe” css code als libraries als Bootstrap te kunnen werken.
15. Het aantonen met libraries zoals jquery te kunnen werken.
16. Het gebruiken van deze libraries puur bij extra functionaliteiten (met uitzondering van Twig, welke is toegelicht in het kader van de koppeling van samenlopende werk en studie projecten).