Alegemene info:  
  
Multipage application samenvatting informatie:  
  
MPA -> elke request laad een compleet nieuwe pagina in (multipage application)  
SPA -> elke request laad dezelfde pagina maar heringericht (singlepage aplication)   
  
MVC = Model

View

Controller -> frameworks 3 laags opgebouwd  
  
Dependency management -> beheren van groot aantal bibliotheken   
- Composer = tool voor PHP  
- Mava = tool voor Java  
- Nuget = tool voor C#  
  
Templating = gestructueerd pagina’s opbouwen in applicatie  
  
Frameworks worden ondersteund door database -> Voorbeeld hiervan = ORM (object relational mapping), de code dient hier als communicatie middel met de database.   
  
**Voor opdracht, codebase in PHP binnen het framework Laravel**YT series voor uitleg: [https://laracasts.com/series/laravel-8-from-scratch](https://laracasts.com/series/laravel-6-from-scratch) -> let op: videos zijn op basis van oudere versie Laravel   
  
  
Tip voor lezen van dit bestand:

* Klein lettertype (en ingetabt) = video samenvatting
* Blauw lettertype = code
* Schuin = extra
* Iedere pagina heeft een eigen onderwerp, lijkt veel tekst maar valt echt mee

Framework info:

Framework = vooropgezette structuur voor het oplossen van complexe problemen door een verzameling van herbruikbare code te bundelen

Plus punten van framework:

* Focus op gevraagde criteria ipv basis herbouwen
* Door betere focus sneller opleveren van producten
* Betere/ simpelere communicatie tussen ontwikkelaars
* Verhoogde kwaliteit en verbeterde betrouwbaarheid
* Applicatie wordt beter onderhoudbaar
* Populaire frameworks voldoen aan markt standaarden van dat moment (best practices)

Min punten framework:

* Populair framework hoeft niet perse functioneel te zijn
* Kan “overkill” zijn voor sommige situaties
* Bij te weinig kennis van de ontwikkelaar worden plus punten zoals hogere productieve tijd en snellere levering teniet gedaan
* Vraagt vaak een bewuste keuze tussen meerdere complexe frameworks

API info:  
  
API = Application programming interface, grote term samengevat:

* Correcte API is vooraf volledig en eenduidig beschreven -> gedocumenteerde applicatie of library
* API maakt deel uit van library die bepaalt hoe deze zal communiceren met externe code
* API is een interface naar een library waardoor herbruikbare code gekoppeld kan worden aan een applicatie
* API kan gebaseerd zijn op een architectuur (zoals een REST API)
* API kan meerdere versies omvatten

MPA info:  
  
Multi page application:

MPA maakt gebruik van meerdere pagina’s om front-end (web client) met backend (server) te laten communiceren (MVC of MPA architectuur)

Flow MPA:  
1. Gebruiker roept webpagina via url/ hyperlink aan (request)

2. Frontend stuurt request naar controller

3. Controller haalt view op (renderen) en stuurt deze als response terug

4. Frontend wordt opnieuw geladen en de view in de response wordt de web browser opgebouwd

Algemene kenmerken MPA:  
- Voor elk request worden de HTML, CSS en scripts van een pagina opnieuw opgehaald en ingeladen  
- Bewaren van 'state' door het gebruik van sessions en cookies(waarin het session id wordt opgeslagen)  
- Kan makkelijker voor zoekmachines worden geoptimaliseerd door het gebruik van verschillende pagina's  
- Frontend en backend zijn verweven; views worden door controllers gerenderd  
- Eenvoudiger te doorgronden door natuurlijke structuur van pagina's  
  
Dependency Management:  
Dependency Management is het organiseren/beheren van packages/libraries binnen een project die (meestal) van derde partijen afkomstig zijn.   
  
Composer wordt door de meeste moderne PHP frameworks gebruikt als dependency manager. De 'composer.json' file wordt gebruikt om packages/libraries toe te voegen aan een project, waarin de namen en de versies van de binnen te halen packages/libraries worden bijgehouden.

* *Packagist (https://packagist.org) is de default site waarvan packages/libraries worden gedownload naar de lokale computer. Het is ook mogelijk om je iegen gemaakte packages/libraries te uploaden naar packagist en ze dus voor anderen beschikbaar te maken via composer.*

COMPOSER TUTORIAL PART 1 SAMENVATTING (van video):

Hoe werkt composer?

Composer werkt met twee bestanden: composer.json en composer.lock. Het composer.json-bestand bevat de vendor naam van het package dat je wilt installeren en de versie ervan. Je kunt composer vragen om automatisch de nieuwste beschikbare versie te downloaden. Het composer.lock-bestand bevat de exacte versie van het package dat momenteel is geïnstalleerd.

Hoe installeer je composer?

Je kunt composer downloaden en installeren van getcomposer.org. Zorg ervoor dat je het beschikbaar maakt op globaal niveau zodat je het kunt gebruiken voor je eigen toepassingen.

Hoe gebruik je composer?

Om een package te installeren, gebruik je de composer require opdracht en de naam van het package. Composer zal automatisch het composer.json-bestand aanmaken en up-to-date houden. Om de packages te installeren die in het composer.json-bestand staan, gebruik je de composer install opdracht. Als er geen composer.lock-bestand bestaat, zal composer de nieuwste beschikbare versie downloaden. Als er al een composer.lock-bestand bestaat, zal composer de versies installeren die in het bestand staan.

Om de nieuwste beschikbare versie van een package te installeren, gebruik je de composer update opdracht. Composer zal dan de nieuwste beschikbare versie downloaden en het composer.lock-bestand up-daten.

Een voorbeeld:

Laten we bijvoorbeeld het intervention/image package installeren om een afbeelding te bewerken. Voer de volgende opdracht uit in de terminal:

--

*composer require intervention/image*

--

Composer installeert het package en zorgt voor het automatisch laden van de klassen. Nu kun je de volgende code gebruiken om een afbeelding te verkleinen:

--

*<?php*

*require 'vendor/autoload.php';*

*$image = Image::make('dino.jpg');*

*$image->resize(300, null, function ($constraint) {*

*$constraint->aspectRatio();*

*});*

*$image->save('dino-small.jpg', 100);*

*echo '<img src="dino-small.jpg">';*

*echo '<img src="dino.jpg">';*

*?>*

--

En dat's alles! Je hebt nu een basisbegrip van composer en hoe je dependencies kunt beheren in je PHP-projecten

Templating info:  
  
Templating (Blade):   
Een templating engine library is een tool om op een nette en gestructureerde manier code toe te voegen aan views.  
In Laravel gebruik je Blade als engine -> Blade gebruikt een verzameling tags in de views die PHP handmatig schrijven overbodig maakt en dus overzichtelijker   
  
Voorbeeld lijst van deze tags:

* {{ $var }} - Echo content
* {{ $var or 'default' }} - Echo content with a default value
* {{{ $var }}} - Echo escaped content
* {{-- Comment --}} - A Blade comment
* @extends('layout') - Extends a template with a layout
* @if(condition) - Starts an if block
* @else - Starts an else block
* @elseif(condition) - Start a elseif block
* @endif - Ends a if block
* @foreach($list as $key => $val) - Starts a foreach block
* @endforeach - Ends a foreach block
* @for($i = 0; $i < 10; $i++) - Starts a for block
* @endfor - Ends a for block
* @while(condition) - Starts a while block
* @endwhile - Ends a while block
* @unless(condition) - Starts an unless block
* @endunless - Ends an unless block
* @include(file) - Includes another template
* @include(file, ['var' => $val,...]) - Includes a template, passing new variables.
* @each('file',$list,'item') - Renders a template on a collection
* @each('file',$list,'item','empty') - Renders a template on a collection or a different template if collection is empty.
* @yield('section') - Yields content of a section.
* @show - Ends section and yields its content
* @lang('message') - Outputs message from translation table
* @choice('message', $count) - Outputs message with language pluralization
* @section('name') - Starts a section
* @stop - Ends section
* @endsection - Ends section
* @append - Ends section and appends it to existing of section of same name
* @overwrite - Ends section, overwriting previous section of same name

Link naar blade: [https://laravel.com/docs/10.x/blade](https://laravel.com/docs/8.x/blade)  
  
  
  
  
  
  
  
  
  
  
ORM (object relational mapping) info:  
  
ORM:  
Een ORM tool is als een brug tussen een database en een programmeermodel, zoals in ons geval MVC frameworks. Het vertaalt database-informatie naar programmeerobjecten zodat je gemakkelijk databasebewerkingen kunt uitvoeren.

In Laravel gebruikt men Eloquent als ORM. Je kunt een model maken en het laten erven van Eloquent's mogelijkheden met behulp van het "extends" keyword. Zoals: (Illuminate\Database\Eloquent\Model). -> [https://laravel.com/docs/10.x/eloquent](https://laravel.com/docs/8.x/eloquent)

Andere bekende ORM is “Doctrine” -> <http://www.doctrine-project.org/>

OBJECT RELATIONAL MAPPING SAMENVATTING (video):  
  
Object-relational mapping (ORM) is een techniek voor het opslaan, ophalen, bijwerken en verwijderen van data in een relational database vanuit een object-georiënteerd programma. Het is een manier om de verschillen tussen objecten en relational data te overbruggen, elk met hun eigen manier van organiseren en opslaan van data.

In een ORM-systeem zorgt een data layer ervoor dat de vertaling tussen objecten en relational data plaatsvindt. Deze data layer is vaak een bibliotheek die in de specifieke object-georiënteerde taal (bijvoorbeeld Java of C#) is geschreven en samenwerkt met het web framework waarmee de rest van de toepassing wordt geschreven.

Een voorbeeld van data in een object-georiënteerd programma is een klasse met eigenschappen als id (integer), naam (string) en beschrijving (string). In een relational database zou deze data worden opgeslagen in een tabel met kolommen voor id, naam en beschrijving.

Er zijn vier belangrijke operaties die uitgevoerd kunnen worden op data in een database: create (of save), read, update en delete (crud). Een ORM-systeem maakt het mogelijk om deze operaties uit te voeren zonder dat je je moet bekommeren om de details van de vertaling tussen objecten en relational data.

Als je data uit de database wilt ophalen, maakt de data layer een nieuw object van het juiste type en vul de eigenschappen ervan in op basis van de data in de database. Zo kan je in je code gewoon met objecten werken alsof ze rechtstreeks in het geheugen staan, terwijl ze in feite uit een database komen.

Het kan echter wel complex worden als je werkt met relaties tussen objecten en data in verschillende tabellen. Een ORM-systeem kan je helpen om deze complexiteit te verwerken en ervoor zorgen dat je toch gemakkelijk met je data kunt werken.

Routing info:  
  
Routing verwijst naar hoe webapplicaties specifieke taken afhandelen op basis van URL's. Een http-request vraagt een server om een bestand terug te geven of een actie uit te voeren, gevolgd door een respons. Deze aanvraag bestaat uit een methode (zoals GET of POST), een URL en eventuele extra informatie.

Vroeger werden URL's direct naar specifieke scriptbestanden geleid. Nu zijn URL's gestructureerd rond het type data en de taak die uitgevoerd moet worden. Routing is de techniek die bepaalt welke script, functie of methode een http-request afhandelt op basis van de URL.

Moderne URL's bevatten geen scriptbestandsnaam meer, maar de server weet welk bestand de request moet afhandelen door middel van 'request rewriting'. Dit wordt geregeld in de public map van de server, waar een configuratiebestand genaamd .htaccess staat.

In Laravel, het framework, definieert de ontwikkelaar expliciet welke URL's welke acties moeten afhandelen in het routes/web.php-bestand. Laravel probeert vervolgens de binnenkomende URL te matchen met een patroon en voert de bijbehorende actie uit. (index.php blijft in alle gevallen wel het start punt, en is dus het start-script voor alle taakgerichte http requests)

Samenvattend moet een ontwikkelaar begrijpen hoe routing werkt, de documentroot van de server kunnen instellen, .htaccess kunnen aanpassen indien nodig, routes definiëren in routes/web.php en bekend zijn met reguliere expressies voor het definiëren van patronen.

Als dev moet je dus:

* een goed inzicht hebben in de routing zoals hierboven uitgelegd;
* de documentroot van een te gebruiken http-server kunnen zetten op de public map van de webapplicatie. Eventueel moet je een virtual host kunnen opzetten;
* eventueel de .htaccess kunnen controleren en aanpassen. Elk framework kan anders routen;
* de routing kunnen definiëren in routes/web.php; [https://laravel.com/docs/10.x/routing](https://laravel.com/docs/8.x/routing/)
* inzicht hebben de mogelijke patronen in de vorm van regular expressions

LARAVEL FROM SCRATCH PART 3 SAMENVATTING (Video):  
  
Zojuist hebben we onze Laravel-project op onze lokale machine gezet op (eerdere videos). Nu gaan we aan de slag met routes en controllers. We willen een pages controller hebben om pagina's zoals over ons, diensten en onze homepage te creëren. Als we onze routes-bestand openen en naar web.php gaan, hebben we routes die de soort verzoek, URL en functie aangeven. We kunnen bijvoorbeeld "return 'Hello World'" plaatsen en zien we deze tekst verschijnen op onze homepage. We kunnen ook dynamische waarden toevoegen aan de URL, zoals in de route "/users/{id}".

Nu gaan we over naar de controller. We kunnen de artisan-opdracht uitvoeren om een nieuwe controller aan te maken. We noemen deze "PagesController" en plaatsen deze in de map app/Http/Controllers. We voegen een index-methode toe aan de controller en wijzigen onze route zodat deze verwijst naar deze methode. In de index-methode kunnen we vervolgens "return view('pages.index')" plaatsen om onze index-blade-sjabloon te laden.

Tot slot gaan we een basislayout maken die we kunnen uitbreiden voor onze andere bladen. We maken een nieuw bestand aan in de map resources/views en noemen dit "layout.blade.php". We kunnen hier onze HTML-opmaak plaatsen, inclusief kopteksten en scripts. Vervolgens kunnen we onze andere bladen uitbreiden met deze layout.

Middleware info:  
  
Middleware in Laravel is een stukje PHP-code dat wordt uitgevoerd voordat een verzoek wordt afgehandeld door de juiste methode of functie. Het wordt gebruikt om te controleren wie toegang heeft tot bepaalde bronnen of functies in een webapplicatie.

Stel je voor dat een website verschillende soorten gebruikers heeft, zoals bezoekers, geregistreerde gebruikers, beheerders, enzovoort. Dit worden 'rollen' genoemd. Afhankelijk van de rol willen we verschillende gebruikers toegang geven tot verschillende delen van de website. Middleware kan helpen bij het filteren van verzoeken, zodat alleen geautoriseerde gebruikers toegang krijgen tot bepaalde functies of gegevens.

Het gebruik van middleware in Laravel gebeurt in twee stappen:   
1. Je moet de middleware code definiëren, wat betekent dat je bepaalt welke controles moeten worden uitgevoerd voordat een verzoek wordt afgehandeld.  
2. Vervolgens moet je de middleware algemeen registreren of toekennen aan specifieke groepen van routes, zodat de controles worden toegepast op de juiste verzoeken.  
  
  
Laravel link: [https://laravel.com/docs/10.x/middleware](https://laravel.com/docs/8.x/middleware)

Authentication en authorization info:  
  
Authentication is het proces waarbij een applicatie controleert of een gebruiker echt is door hun gebruikersnaam en wachtwoord te controleren. In Laravel is authenticatie al standaard ingesteld, maar je kunt het aanpassen zoals je wilt.  
Laravel maakt gebruik van 'guards' om te bepalen hoe een gebruiker wordt geauthenticeerd bij elk verzoek, en 'providers' om gebruikersinformatie op te halen, bijvoorbeeld uit de database.

Autorisatie is het proces waarbij een applicatie controleert welke rechten een gebruiker heeft om toegang te krijgen tot bepaalde delen van de applicatie. Deze rechten kunnen worden toegekend aan individuele gebruikers of aan groepen waar een gebruiker lid van is.  
In Laravel worden hiervoor 'gates' en 'policies' gebruikt. Deze concepten kunnen op verschillende manieren worden toegepast in Laravel, afhankelijk van de specifieke behoeften van de applicatie.  
  
Authentication link: <https://laravel.com/docs/10.x/authentication>  
  
Authorization link: <https://laravel.com/docs/10.x/authorization>