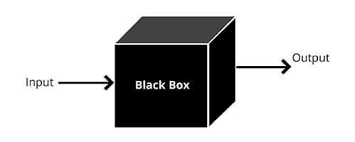
|  |  |
| --- | --- |
|  |  |
| **Project 7 Laravel** |  |
|  |  |
|  |  |

**Les 5 – Het MVC model**

We gaan het vandaag hebben over **het MVC model**

1. Wat is het?
2. Waarom gebruiken we het?
3. Voorbereiden met de opdrachten naar een eigen crud

## Samenvatting van de eerdere lessen



### Routes

We hebben het eerder gehad over de ‘input’ gedeelte, welke wij routes noemen. Elke route moet binnen Laravel eerst gedefinieerd zijn voordat deze werkt. Wanneer je een 404 fout ziet verschijnen, betekent het al snel dat je ‘route’ in je applicatie niet bekend is.

### Templates

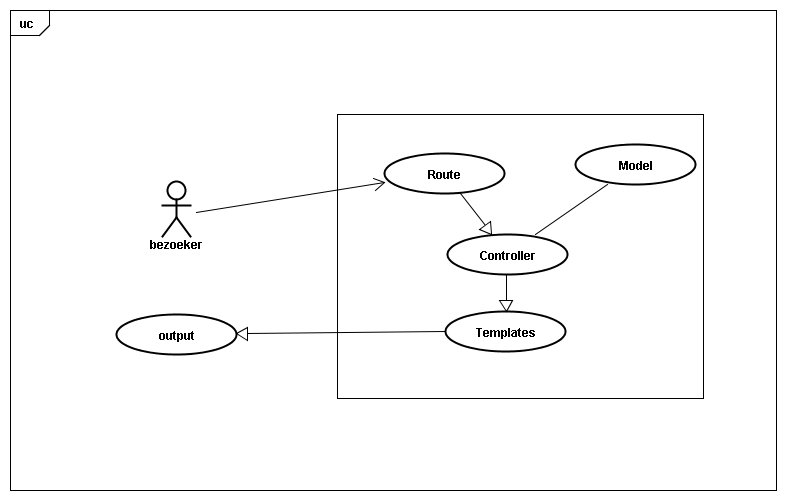
Het gedeelte ‘output’ hebben we eerder in de lessen gehad met als belangrijkste taak, volledige verantwoordelijkheid van de output. Dit kan bijvoorbeeld een webpagina zijn in een bepaalde opmaak.

## Blackbox

#### In deze les gaan we bekijken wat er hier gebeurt wanneer er een ‘route’ ofwel een pagina via de browser opgevraagd wordt.

#### 

# Het MVC model



## Controller

Een controller is een belangrijk onderdeel binnen Laravel.

Tijdens de lessen zul je zien dat elk onderdeel een geheel eigen functie heeft. En dat is ook heel erg belangrijk om het zo te houden binnen het object georiënteerde principe (OO) .

Zoals de naam een beetje laat merken is een ‘controller’ een controleur te noemen. Wanneer een ‘route’ binnen komt, dan gaat het door naar een controller. De functie van een controller is het verder aanknopen, regelen en eigenlijk alles doorgeven aan de views zodat de bezoeker weer wat ziet.

In de volgende voorbeeld willen wij een applicatie maken om het weersinformatie op te halen. Bijvoorbeeld wat is de temperatuur vandaag? Gaat het regenen of niet?

In theorie gaat het er meer om dat je begrijpt wat de logische plekken zijn binnen Laravel om de juiste stukjes code uit te werken. En zodat het in zijn geheel een werkende applicatie is.

Voor het uitwerken van een applicatie heb je een API nodig waar je weersinformatie kan opvragen. Je kan dit natuurlijk niet zelf bepalen en hier heb je dus externe data nodig. Je hebt iets van een weerservice module nodig die je op een of andere manier kan gebruiken.

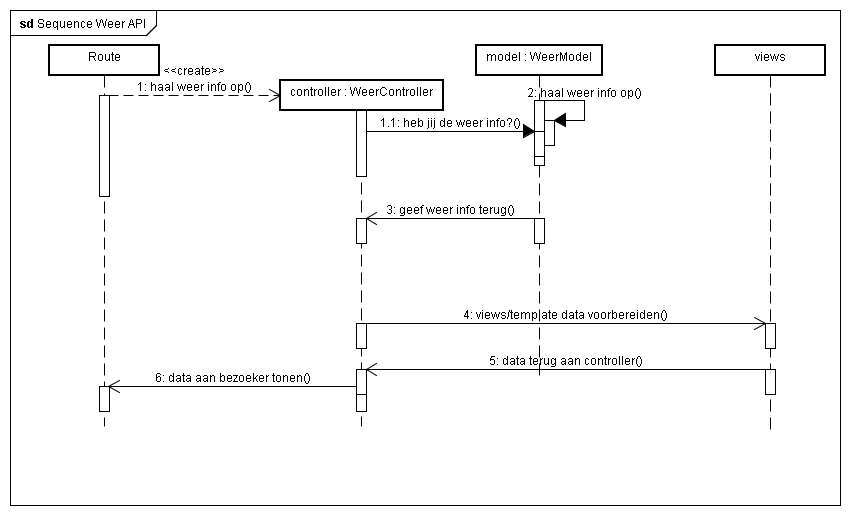
Een API is een dienst/service waarbij je (tegen betaling of gratis) een dienst kan opvragen. Er zijn best wat API’s beschikbaar en via simpele commando’s kun je dus een vraag/antwoord-service oprichten waarbij je de data kunt gebruiken in je eigen voordeel.

Stel dat je elke dag met de fiets naar school gaat, en je zou best wel willen weten of het vandaag gaat regenen; endus een regenpak meenemen of niet?

Je bedenkt zelf, stel nu dat ik voor mezelf een applicatie maak die elke ochtend voor mij de weersinfo aanlevert en een bericht op mijn mobiel weergeeft wanneer het daadwerkelijk gaat regenen? Nou, dat zou toch super zijn! En dat maak je ook gewoon zelf, hoe leuk is dat niet?

Een voorbeeld van een sequence of state diagram ziet er als volgt uit.

**Voorbeeld van een functie uitwerking**



### UITLEG SEQUENCE DIAGRAM (de stappen zijn genummerd zie afbeelding)

1. Route: ‘/check’
   1. Via de route wordt er een controller aangehaakt om het verzoek verder af te maken
      1. De controller weet zelf niet de inhoudelijke logica, dus alle bedrijfslogica en formules staan altijd in models
2. Models ‘Weer Model’ is verantwoordelijk om met een API te praten
3. Models geeft deze info weer terug aan de controller
4. Controller bereidt de data voor om aan de views te geven, je templates dus
5. Wanneer de template klaar is, is de controller nog steeds de baas of deze output al door mag of niet
6. Op het einde, wanneer de controller klaar is, zal de output daadwerkelijk zichtbaar zijn naar de bezoeker toe

Wat valt hier nog op? De controller heeft een hele belangrijke functie. Wellicht zou je in de toekomst met een nieuwe weerAPI willen ‘praten’. En volgens onze OO (object georiënteerde manier) wijze, hoef je dus alleen een nieuwe model te maken, in de controller vervolgens de nieuwe model te instantiëren en vervolgens gedraagt de applicatie verder dezelfde. Cool hè?

In het gehele traject behoudt de controller de controle!

Een controller heeft soms meer info nodig en dan komt het vaak voor dat je met meerdere modellen ‘praat’.

Dat is heel normaal, maar in alle wijsheid: houdt voor je zelf altijd aan dat een functie en dus een ‘route’ maar 1 ding doet. In praktijk zie je vaak dat je veel meer wilt gaan doen dan dat de bedoeling was. Doe dit niet, en houdt het gewoon simpel voor jezelf.

Dat maakt het voor in de toekomst altijd makkelijker om routines te veranderen.

Stel dat je een crud maakt van een tabel; houdt het simpel.

Zorg voor de volgende routes:

* tabelnaam \Index à dit is je overzicht gedeelte, dus toon alle records van je tabel
* tabelnaam \ bekijk \ id \ {id} à dit is een specifieke record die je wilt bekijken, dus maar 1
* tabelnaam \ bewerk \ id \ {id} à dit is een bewerking op een bestaande record, je wilt hier dus iets gaan aanpassen
* tabelnaam \ verwijder \ id \{id} à je bent klaar met een record en je wilt deze verwijderen
* tabelnaam \ nieuw à je wilt een nieuwe invoer maken

Stel dat je tabel ‘todoes’ heet, dus mogelijk een notitietabel voor in je applicatie, dan vervang je ‘tabelnaam’ met ‘todoes’.

Dit geldt dus ook voor de overige tabellen die op zich zelf staan, daarmee bedoel ik nog geen relaties met andere tabellen.

## Model en het MVC uitgelegd

**Models is het hart van je applicatie!**

Alle logica die in de models beschreven zijn, moet gezien worden als het centrale punt waarmee de routes à controllers mee samen werken.

In het MVC model staat M voor Model.

Stel nu in het vorige voorbeeld van een applicatie die voor je elke dag het weer controleert op regen.

Het stuk logica van het inhoudelijke moet dus in de model staan.

Hoewel er varianten dat het in de controller ook opgelost kan worden is een ander verhaal.

Voor ons geldt dat logica van je applicatie in je models beschreven dienen te worden.

Waarom? Waarom zouden we dat nu willen?

Het is een ‘best-practice’ om het om deze manier te doen.

Wanneer je de logica centraal houdt, kun je altijd in de toekomst overstappen naar een andere framework met minieme ontwikkelingskosten. In praktijk worden applicaties dusver afhankelijk gemaakt dat je eigenlijk niet zo goed over kunt stappen.

Dat is jammer en dat komt eigenlijk door je startmoment van keuze en je ontwikkelteam dat je vast raakt.

Voor ons tijdens de lessen houden we aan dat business logica (een nette woord voor alle intelligentie in je applicatie) in je models horen. Afgesproken?

Controllers (de controleurs ) die hebben alleen als taak om te koppelen à data verzamelen à z.s.m. te gaan naar een output, dus je templates opbouwen.

Een laatste en een voornaamste reden om een MVC model framework te gebruiken is dat je in een team op diverse plekken direct (tegelijkertijd) aan de slag kunt gaan.

In de meeste webbureau’s kennen we de volgende teams binnen een project:

* Frontenders (design)
* Backenders
* Testers
* Code reviewers
* Deploymentteam

GA NAAR DE OPDRACHTEN