hogeschool Leiden

12-1-2017

Ontwerp Ontwikkelstraat

Groep 4

Auteurs:
Dennis van Bohemen
Philip Wong
Roy Touw

Module:
IpsenH

Opleiding:

Klas:	INF3B
Opleiding:	Informatica
School:	Hogeschool Leiden

Versie:
Versie 1.0 Final

Versie	Datum	Omschrijving	Opgeslagen door:
0.1 Ontwikkel	08-12-2016	Initiële versie	Roy Touw
0.2	11-01-2016	Toevoeging hoofdstuk twee, drie en vier.	Roy Touw
0.3	12-1-2017	Layout fix	Dennis van Bohemen
1.0	13-01-2017	Afronding document	Roy Touw

INHOUDSOPGAVE

1 inleiding 1

2 Applicatiestructuur 2

3 angular 2 1

4 testarchitectuur 3

1 INLEIDING

In dit document wordt de architectuur van de applicatie TravelPlanner beschreven. In hoofdstuk twee wordt vanaf het hoogste niveau beschreven hoe de abstracte componenten zich onderling verhouden en hoe deze communiceren.

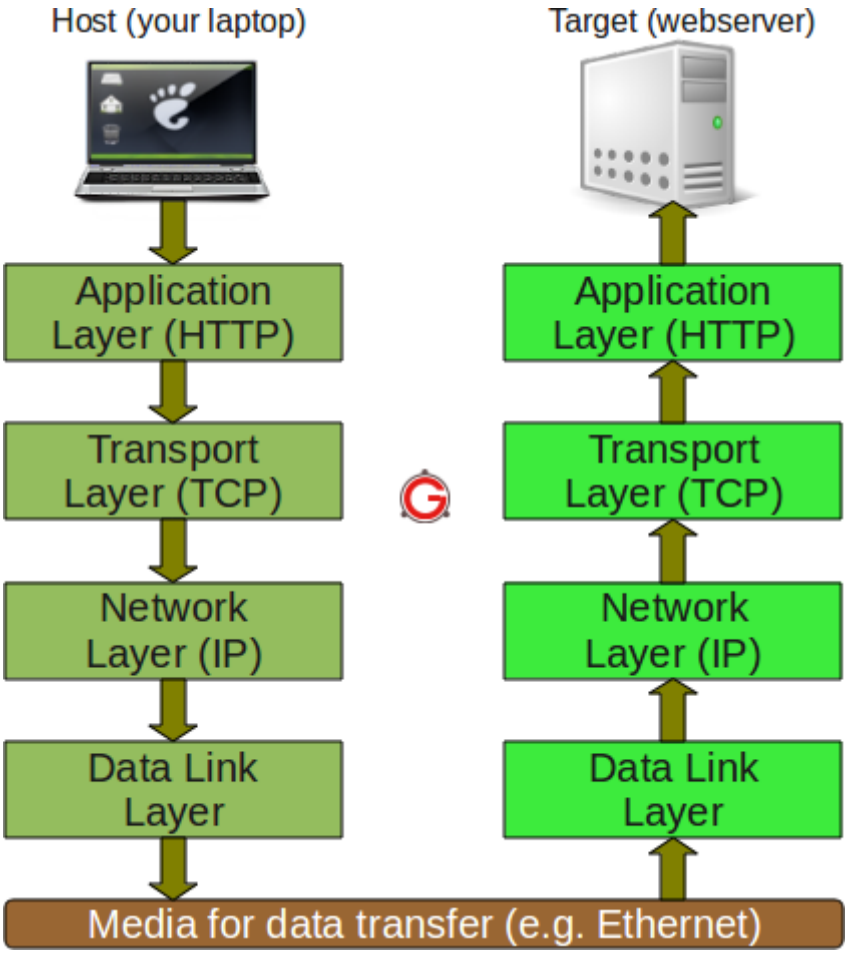
In hoofdstuk drie wordt het Angular2 framework wat wij gebruikt hebben beschreven. In het vierde hoofdstuk wordt gekeken naar de invloed van het testen op de applicatiearchitectuur.

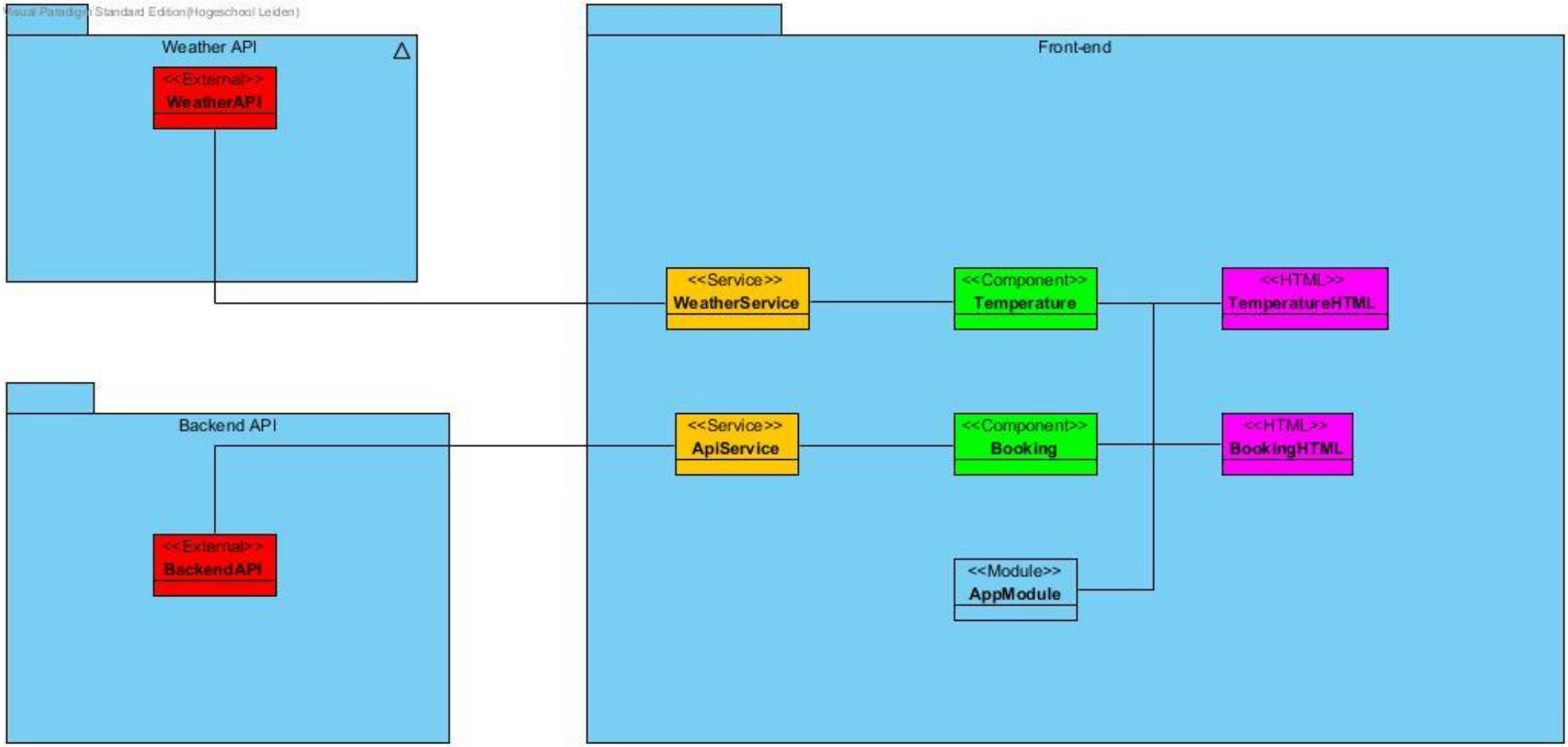
2 APPLICATIESTRUCTUUR

In dit hoofdstuk wordt vanaf het hoogste niveau naar de applicatiestructuur gekeken. Zoals te zien in het diagram op de volgende pagina bestaat de applicatie uit drie aparte systemen zijnde:

- Front-end, de webapplicatie.
- Backend, het systeem wat onder andere communiceert met de database.
- Weather API, het systeem van een externe partij waar de weerdata vandaan gehaald wordt.

Kijkende naar het diagram valt op te merken dat de front-end verbonden is met beide andere systemen, de communicatie tussen de front-end en de backend systemen gebeurt via desbetreffende de WeatherService en de APIService. Deze services communiceren met de andere systemen gebruikmakend van het HTTP protocol.





Er is gekozen om de communicatie met de externe systemen via de services te laten verlopen omdat dit het meest flexibel is. De services zijn binnen onze applicatie als singleton opgezet, dit heeft als meerwaarde dat er meerdere componenten met een instantie van de service kunnen communiceren, met als resultaat dat de service eenmaal de data ophaalt en beide componenten deze data kunnen benaderen.

Als voorbeeld hiervan het ophalen van het weer:
Het booking component geeft nadat er een locatie is geselecteerd door de gebruiker door aan de WeatherService dat de weerinformatie van deze bestemming opgehaald kan worden, vervolgens gaat de gebruiker verder naar de volgende pagina. Op de volgende pagina staat het weather component, deze vraagt vervolgens de weer informatie van de reeds geselecteerde bestemming op aan de WeatherService en toont deze aan de gebruiker. Er is nu maar een externe call gedaan en de data wordt netjes binnen het systeem gedistribueerd.

De rol van de componenten is de bussineslogic, hier worden de front-end berekeningen gedaan en vraagt de services data op te halen of weg te sturen. De rol van de HTML elementen is enkel het grafische aspect.

3 ANGULAR 2

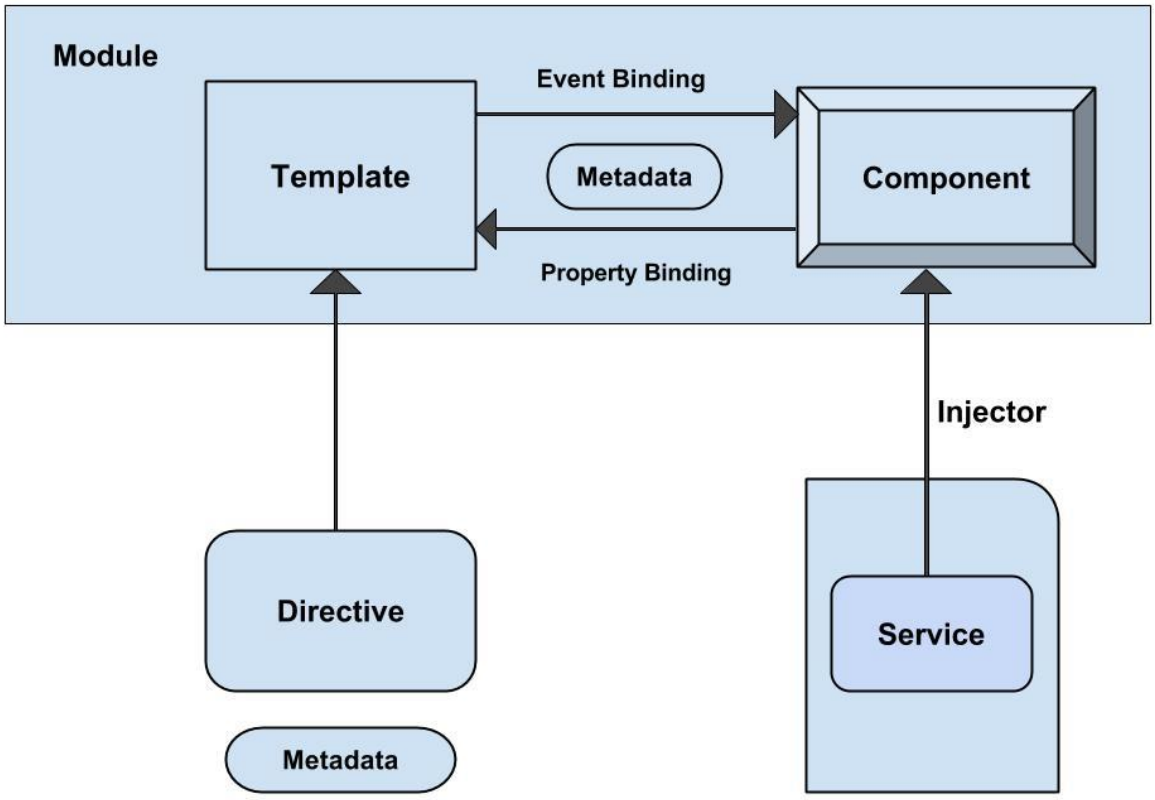
In dit hoofdstuk wordt beknopt uiteengezet hoe wij het Angular2 framework gebruikt hebben en hoe onze ervaring daarmee is geweest.

De front-end applicatie is gemaakt met het framework Angular 2, hiervoor is gekozen omdat dit de voorkeur van de opdrachtgever is.

Het voordeel van Angular2 is dat het door de modulaire structuur erg goed te testen is, per service of component wordt er een file gemaakt met eenzelfde naam als het desbetreffende component of service gevold door ".spec". In deze .spec files komen de unit tests voor het bijbehorende component of service, hierdoor heb je kleine testbestanden wat zorgt voor een grote mate van overzichtelijkheid.

Het nadeel van het gebruik van Angular2 is dat het nog erg nieuw is, en dat er binnen het Angular2 framework ook een aantal versies zijn geweest. Hierdoor is het lastig om informatie en voorbeelden te vinden omdat je dan of helemaal niks vindt, of een verouderde oplossing.

In onderstaand diagram is de architectuur van het Angular2 framework te zien:



Gebruikmakend van het Angular2 framework begin je met een module, dit is het grootste te onderscheiden element binnen een applicatie. Een applicatie bestaat altijd een minimaal een module, en bij een grotere applicatie uit meerdere modulen, aangezien onze TravelPlanner tamelijk klein is bestaat die uit een module.

Binnen deze module gaat de hoofdrol naar de componenten, een component representeert een proces of functie binnen de applicatie, zoals het Booking component het boekingsproces representeert. Binnen dit component heb je een klasse met hierin de attributen en methoden die gebruikt worden binnen het proces.

Een component heeft een Template, dit kan intern maar meestal wordt hiervoor een extern HTML bestand gebruikt om aan het SRP principe te voldoen. Het HTML bestand is zoals elk ander HTML bestand met als toevoeging dat er variabelen uit het bijbehorende component gebruikt kunnen worden, en dat er events functies uit het component aangeroepen kunnen worden. Dit komt door de property binding en event binding tussen deze twee elementen.

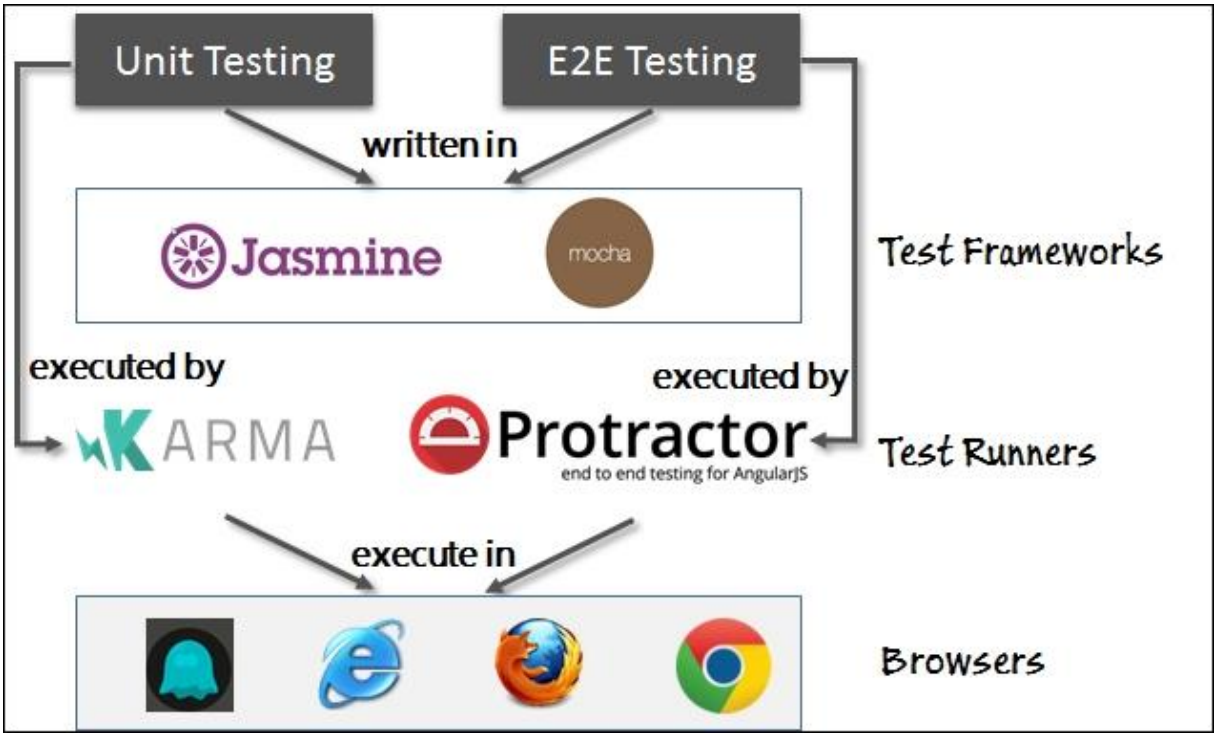
Naast de componenten die een concreet doel hebben zijn er ook services die een abstractere taak hebben. Deze services zijn in Angular2 niet per se een singleton, maar meestal is dit wel waar je naar streeft zodat er gelijke data is tussen de componenten die met de service communiceren. Binnen de service staan de globalere functies en variabelen die door de gehele module gebruikt kunnen worden, zoals bijvoorbeeld het opvragen van het weer, wat vervolgens meerdere componenten kunnen gebruiken.

Als laatste is er nog een directive, dit is feitelijk een custom HTML element, deze heben wij bij dit project niet zelf gemaakt, wel hebben we enkele directives uit de standaard Angular2 bibliotheek gebruikt.

4 TESTARCHITECTUUR

Over de testarchitectuur is al een klein stuk geschreven in hoofdstuk twee, in dit hoofdstuk wordt hier verder op ingegaan.

In het onderstaande diagram is te zien hoe de tests binnen onze applicatie zijn vormgegeven.



Het eerste wat er valt op te maken uit het diagram is dat er twee soorten tests zijn, unit tests en e2e tests dan wel ui tests. De unit test zijn tests geschreven voor kleine stukken logica, waar de e2e test geschreven zijn om de werking vanuit het gebruikersperspectief te testen.

Beide tests zijn geschreven in het Jasmine framework, dit heeft als voordeel dat er maar een framework geleerd hoeft te worden om beide tests te kunnen schrijven en lezen.
De unit tests worden uitgevoerd door Karma waar de e2e tests uitgevoerd worden door Protractor. Voor het uitvoeren van de tests is een webbrowser nodig, hiervoor hebben wij PhantomJS gekozen, dit is een webbrowser zonder grafische weergave. Wij hebben voor PhantomJS gekozen omdat de tests uitgevoerd worden in de ontwikkelstraat, en daar heeft de grafische weergave geen meerwaarde, waar dit het testen wel trager maakt.

De resultaten van de tests worden vervolgens in een xml bestand geplaatst welke verwerkt wordt door de ontwikkelstraat, voor de werking hiervan zie het ontwikkelstraat ontwerp document.