

Goeiemorgen iedereen,

Ik ben Frederik Bouillon en ik kom jullie vandaag iets meer vertellen over React & Flux.

Voor de mensen die mij nog niet goed kennen, ik ben 28 jaar en afkomstig uit geel. Ik werk bij euricom sinds begin dit jaar waarin ik gestart ben in het tweede frontend bootcamp.

Sinds begin juni ben ik actief bij de klant Digipolis. Digipolis staat in voor alle IT-software van de provincie antwerpen waaronder software voor brandweer, politie, enz. Zelf werk ik mee aan een project voor politie antwerpen.

Dit is meteen ook de reden waarom ik hier kom vertellen over React & flux, namelijk omdat we deze technologieen gebruiken voor mijn project op digipolis.

**AGENDA**

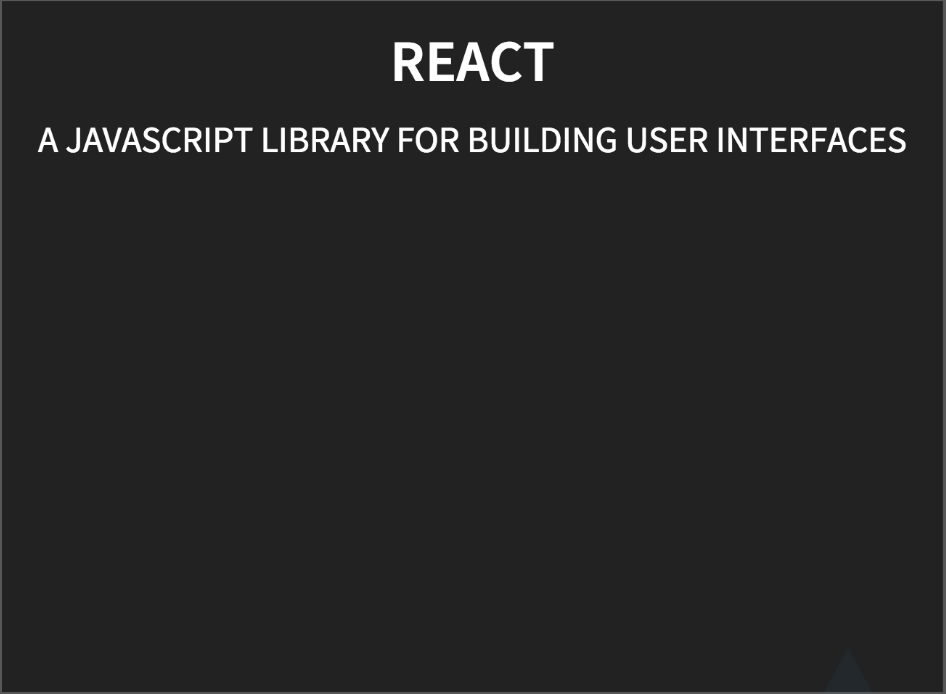
Ik ga beginnen met wat theorie over react gevolgd door een live demo.

Vervolgens ga ik verder met Flux waarbij ik het concept zal toelichten waarna ook een voorbeeld volgt waarbij ik met jullie doorheen de code zal gaan.

**POSITIONERING REACT EN FLUX**

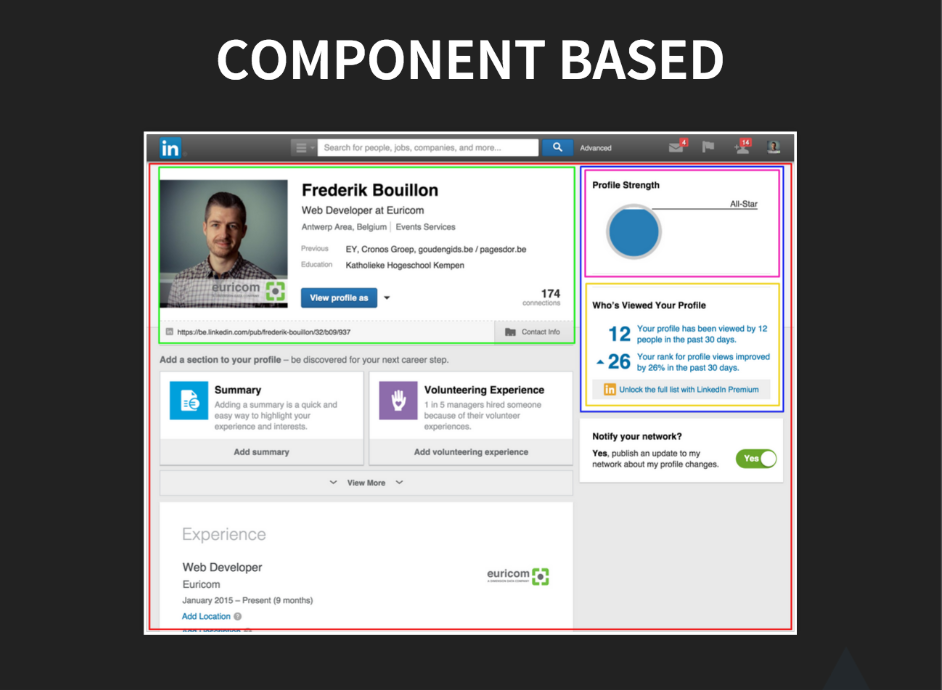
Voor dat ik dieper in ga op React wil ik jullie eerst even snel een kort overzicht geven van wat de rollen precies zijn van react en flux.

De opzet van een applicatie gebouwd in react en flux volgt niet de MVC structuur maar als we de vergelijking zouden maken kunnen we zeggen dat React aanschouwd wordt als de V. De rol van flux speelt zich meer af in door meer structuur te brengen in onze applicatie (en wat meer separation of concern) en kunnen we daardoor meer vergelijking met de M en de C.



React is een open-source Javascript library voor het creëren van single-page applicatie web applicaties. Zoals ik daarnet al vertelde wordt React aanschouwd als de V van in MVC. React is dus geen framework zoals Angular.

React werd ontwikkeld door Facebook voor het oplossen van 1 probleem. Namelijk voor het bouwen van grote applicaties met data die met verloop van tijd zal veranderen.



In react draait alles rond componenten. Ze zijn in feite de bouwstenen van react. Je kan deze vergelijken met directives van angular. Een react component is een verzameling van HTML, CSS en Javascript.

In dit voorbeeld heb ik mijn linked-in profiel pagina even logisch onderverdeeld in components. Zoals je kan zien kunnen deze components gebruikt worden als container die op hun beurt weer andere components bevatten.

Wat react zo speciaal maakt voor het bouwen van web interfaces is dat je zelf kan bepalen of een component al dan niet data bevat. Zo heb je bv een react component die kan dienen als een soort van container of met andere woorden parent voor andere components. Op deze parent wordt dan meestal de data bijgehouden welke deze op zijn beurt kan doorgeven aan zijn onderliggende components.

In dit voorbeeld zouden we kunnen zeggen dat de blauwe component verantwoordelijk is voor de statistieken van mijn profiel pagina. Dit wordt hier dan onderveeld in 2 stukken namelijk enerzijds hoe sterk mijn profiel is en anderzijds hoeveel keer mijn pagina bezocht is. In dit vb zou bv onze blauwe container verantwoordelijk zijn voor het beheren van deze data. Vervolgens zal ze deze data simpleweg doorgeven aan de onderliggende componenten.



Om jullie een idee te geven hoe een component er nu in code uitziet zien we hier een hello world voorbeeld.

De aanmaak van een react component gebeurt door de createClass functie van de react library. Deze functie verwacht een object waarin de component opgebouwd word. Het resultaat van deze functie slagen we op in een variabele HelloWorld.

Een vereiste voor elke component is de render method. Deze method bevat de template van de component. In dit voorbeeld zal de tekst ‘hello world’ getoond worden wanneer deze component gerendered word.

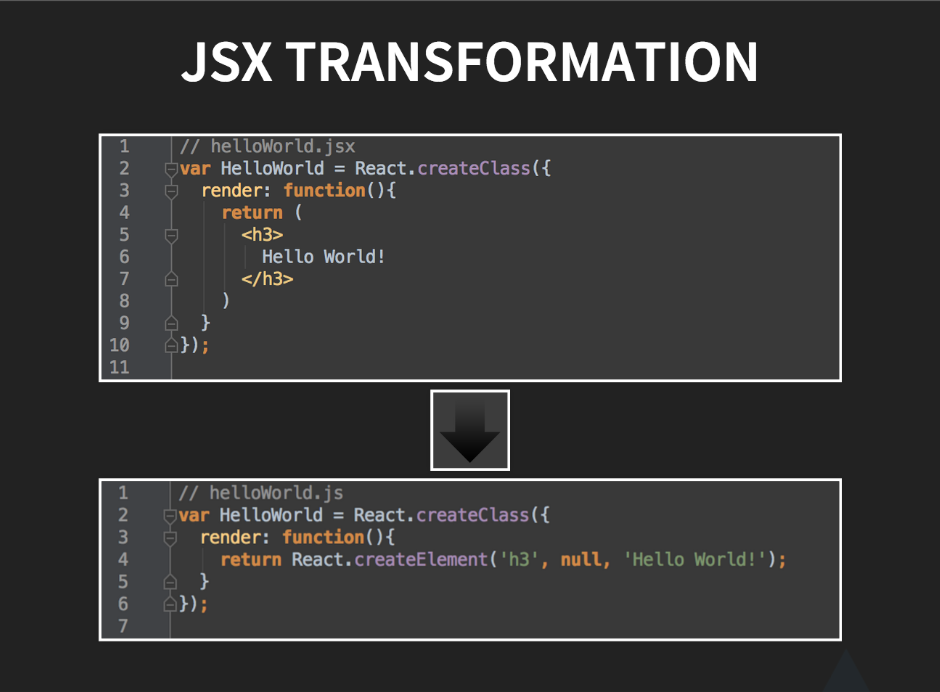
Eens de component correct is opgebouwd dienen we hem nog te renderen. Dit doen we met React.render. Deze functie verwacht 2 argumenten. Als eerste de component die je wil renderen (HelloWorld) en als tweede argument het htlm element waarop je de component wil renderen. In dit voorbeeld zeggen we aan React om onze component te renderen op het element met ‘app’ als ID.

Door de parent/child relatie van react componenten hoeven we deze react.render maar 1 keer uit te voeren. Door de hoogste parent component te renderen worden al zijn child components ook gerendered.

### JSX

Zoals je misschien al hebt opgemerkt heb ik de HTML (of toch iets wat er op lijkt) van deze component rechtstreeks in mijn javascript staan. Zo wordt er in React niet verwezen naar een .htlm file die de template bevat. Dit zou namelijk ook niet lukken aangezien de HTML die je ziet staan in de render method eigelijk geen echte HTML is. Het stukje code wat lijkt op de HTML is eigenlijk JSX. JSX is een soort van XML extentie op javascript.

JSX laat ons toe om onze layout te maken in onze vertrouwde HTML syntax die dan later achter de schermen getransformeerd zal worden naar javascript objecten. Waarom javascript objecten vragen jullie zich waarschijnlijk af. Omdat React vervolgens een virtuele DOM zal opbouwen met deze objecten. Of met andere woorden een javascript presentatie van de actuele DOM.



Hier zie je een voorbeeld van onze react component na de transformatie van de JSX.

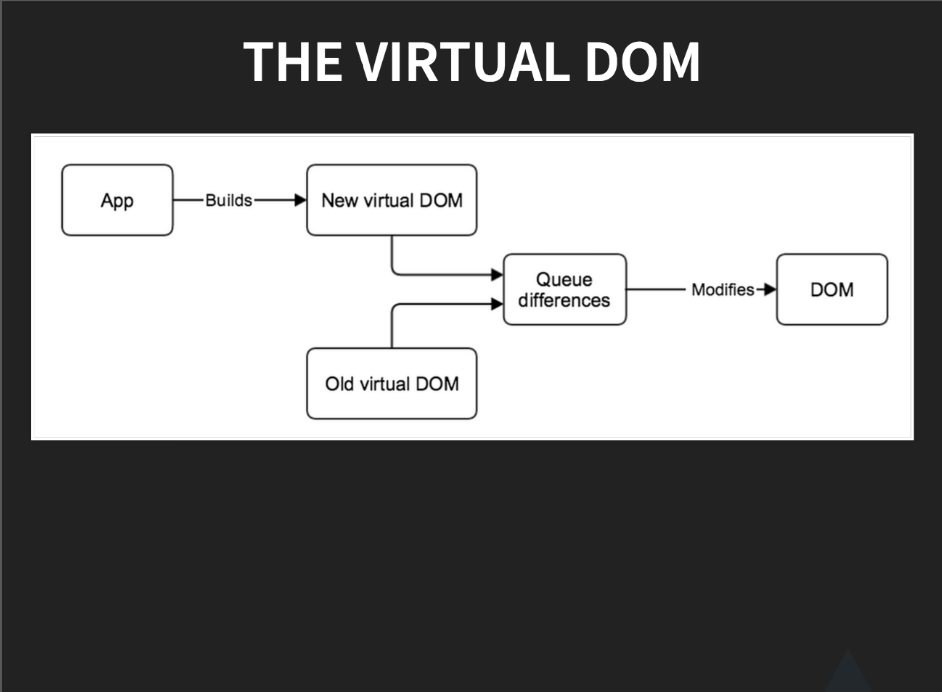
Je zou je react componenten dus ook op deze manier kunnen definieren. Maar zoals je kan zien is dit minder leesbaar en meer vatbaar voor typos of fouten.

Het transformatie process van JSX naar JS gebeurt meestal in het build process.

### HTML attributen en events

Al onze vertrouwde HTML attributen en events zijn eveneens beschikbaar in JSX. Het enige verschil is dat deze in camelcase moeten gedefinieerd worden in plaats van de lowercase stijl van HTML.

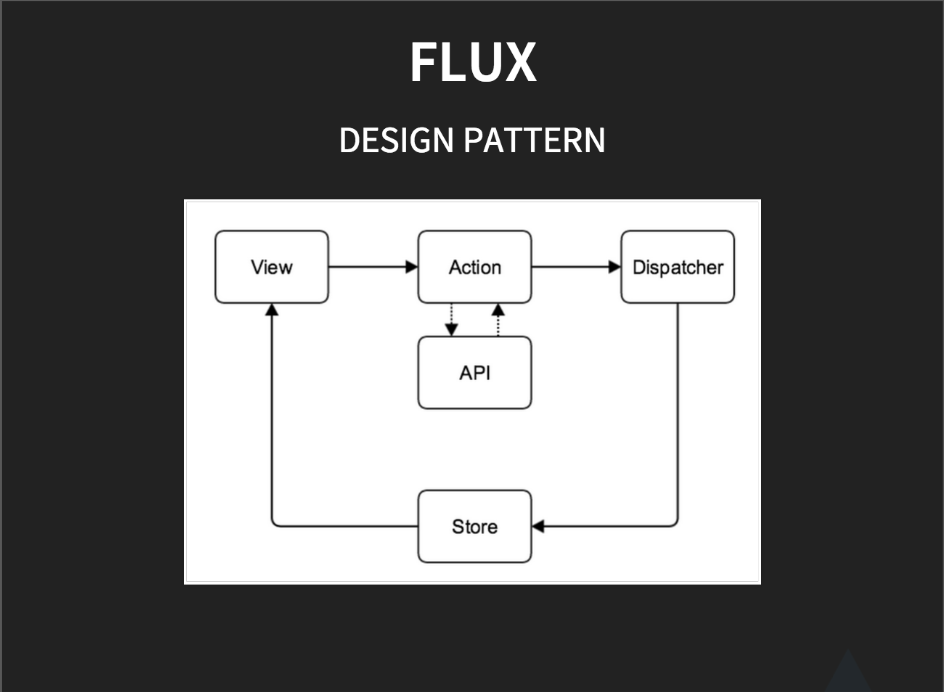
Bijkomend schrijven we HTML elementen zonder hoofdletter en onze react componenten wel met hoofdletter. Op deze manier maakt react het onderscheid tussen onze custom componenten en html tags



De reden waarom React zo performant werkt is doordat het nooit rechtstreeks met de DOM praat. React onderhoud namelijk zelf een zeer snelle in-memory voorstelling van de DOM, ook wel virtuele DOM genoemd.

Bij elke data wijziging zal react namelijk een nieuwe virtuele DOM aanmaken en deze vergelijken met huidige virtuele DOM die de data wijzigingen nog niet bevat. React zal deze 2 versies van de virtuele DOM met elkaar vergelijken en de actuele DOM enkel updaten met deze verschillen. Op deze manier zal niet steeds de ganse DOM geupdate worden bij een data wijziging maar enkel de zaken die hierdoor geaffecteerd zijn.

Het updaten van de DOM gaat in veel gevallen vroeg of laat voor performantie problemen binnen een web applicatie, zeker wanneer deze uitgroeit tot een grote app. Dit komt meestal omdat we heel onze pagina of bepaalde views volledig opnieuw moeten inladen bij een data wijziging. React minimaliseert deze issue door met zijn virtuele dom enkel de gewijzigde elementen van de actuele DOM aan te passen.



Flux is een design pattern dat net zoals react, ontwikkeld en gebruikt wordt door Facebook. Het geeft ons een structuur die zeer goed samenwerkt met React. In een react-flux verhaal wordt geen gebruik gemaakt van het MVC patroon maar eerder van een one way data flow.

Hoewel Flux aanzien wordt als een design pattern maakt het toch gebruik van een javascript library. Namelijk een dispatcher library. De dispatcher is een soort van global pub/sub handler die zaken (payloads) zal broadcasten naar geregistreerde callbacks. Bijkomend vinden we in een typische flux opzet ook de eventEmitter module van nodeJS terug. Deze module geeft ons de mogelijkheid om zelf custom events aan te maken waarop we vervolgens listeners voor kunnen aanmaken.

In volgend voorbeeld zien we een visuele voorstelling van de werking van Flux. We kunnen deze opsplitsen in 4 stukken:

### VIEW

De view in dit diagram stellen onze react componenten voor.

Een belangrijke verandering in onze component is dat data niet meer geinitialiseerd zal worden in de component. Aan het bijhouden van state veranderd uiteraard niets maar de data zal niet meer leven in onze componenten maar deze zal nu leven in de stores.

Wanneer een component zich in de DOM mount zal het zijn initiele state gaan opvragen bij een of meerdere stores.

### ACTIONS

Alle functionaliteiten (zoals het updaten van onze state) van onze react component gaan we niet meer definieren in onze component zelf. In plaats hiervan gaan we actie aanspreken. Een actie bestaat meestal maar uit 2 dingen. Ten eerste een unieke naam die beschrijft wat het doel is van de actie en ten tweede de bijbehorende data indien nodig.

### DISPATCHER

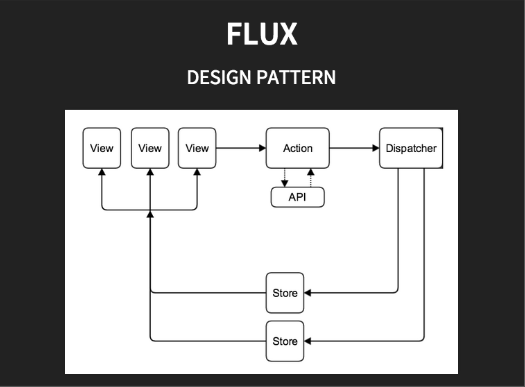
Een actie gaat een methode aanspreken van onze dispatcher. De dispatcher neemt zo de actie binnen en zal niets anders doen dan deze broadcasten.

### STORE

Een store gaan we gebruiken voor de opslag van onze application data.

Een store dient zichzelf ook steeds te registreren bij onze dispatcher. Op deze manier kunnen alle acties die onze dispatcher uitstuurt worden opgevangen in onze store. Vervolgens definieren we in onze store op welke acties we willen dat hij actie moet ondernemen. En met actie ondernemen bedoel ik dat onze store zijn data zal wijzigen naargelang welke actie hij verwerkt.

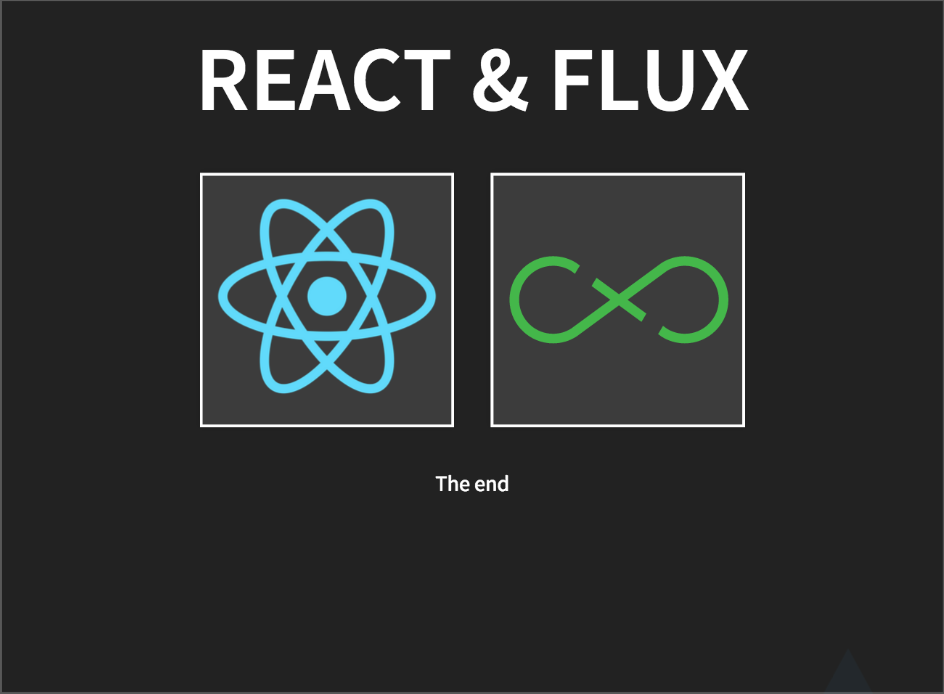
Wanneer onze store een actie verwerkt heeft en zijn data heeft gewijzigd, gaat onze store ook een change event uitsturen. Het zijn vervolgens onze react componenten die door middel van een listener zullen luisteren naar dit change event. Op deze manier zullen onze react componenten verwittigd worden wanneer onze store nieuwe data bevat, vervolgens kunnen ze dan deze nieuwe data gaan opvragen bij de store om dan hiermee hun state te gaan updaten waardoor er een re-render zal plaatsvinden van onze views.



Het voorbeeld van daarnet was een simpele visuele voorstelling van de werking van flux.

Dit voorbeeld toont meer de kracht achter flux. Zo kunnen we zien dat er meerdere stores zich kunnen inschrijven bij onze dispatcher. Deze verwerken vervolgens de acties op welke ze willen reageren.

Vervolgens zullen al onze componenten die geinteresseerd zijn in de gewijzigde stores hierop reageren door zichzelf te re-renderen.



Dit was mijn presentatie van react en flux. Ik hoop dat het voor iedereen duidelijk was en dat jullie het interessant vonden.

Indien het voor sommige toch nog niet helemaal duidelijk of logisch lijkt, kan ik alleen maar zeggen dat dit normaal is. In principe is het allemaal heel simpel maar je hebt wel een bepaalde klik nodig. Ik moet zelf toegeven dat ik zelf in het begin niet direct deze klik had. Het is pas nadat ik door zelf voorbeelden begon uit te werken de klik heb gekregen.

De mensen met extra interesse kan ik alleen maar aanraden om dit zelf eens in de praktijk toe te passen. Indien er nog vragen zouden zijn mag me hier uiteraard steeds over aanspreken.

Als laatste wil ik jullie nog danken voor jullie aandacht!