



HoGent

Faculteit Bedrijf en Organisatie

Monitoring mogelijkheden in NodeJS toepassingen voor feilloos debuggen in 2019

Michiel Leunens

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Tom Antjon
Co-promotor:
Michiel Cuvelier

Instelling: beSports bvba - Kayzr

Academiejaar: 2018-2019

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Monitoring mogelijkheden in NodeJS toepassingen voor feilloos debuggen in 2019

Michiel Leunens

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Tom Antjon
Co-promotor:
Michiel Cuvelier

Instelling: beSports bvba - Kayzr

Academiejaar: 2018-2019

Tweede examenperiode

Woord vooraf

Backend debug toepassingen zijn vaak slecht onderhouden, incompleet of gewoonweg verschrikkelijk duur. Daarom is deze bachelorproef gemaakt met als doel een open-source debug tool te schenken aan iedereen die hun project in NodeJS en Express ontwikkelt. Mede mogelijk gemaakt dankzij Kayzr, is het de bedoeling om deze tool online te zetten zodat deze effectief in de praktijk gebruikt kan worden. Graag bedank ik ook Dhr. Tom Antjon, voor altijd paraat te staan, snel mijn vragen te kunnen beantwoorden en mij een goed inzicht te geven in het werk dat nog gedaan moest worden.

Ook wil ik Kayzr bedanken, voor een onvergetelijke stage, waar ik als developer EN als fotograaf enorm heb kunnen groeien. Zonder hen had ik nooit deze bachelorproef kunnen verwezenlijken.

Samenvatting

Inhoudsopgave

1	Inleiding	13
1.1	Probleemstelling	13
1.2	Onderzoeksvraag	13
1.3	Onderzoeksdoelstelling	14
1.4	Opzet van deze bachelorproef	14
2	Stand van zaken	15
2.1	Javascript	15
2.1.1	Tijdljn van JavaScript	16
2.1.2	Waarom JavaScript?	16
2.2	Node.js	18
2.2.1	Tijdljn van Node.js	18
2.2.2	Waarom Node.js?	19

2.3	Express	21
2.3.1	Waarom Express?	21
2.4	Testen en Monitoring	23
2.4.1	Agile Testing	23
2.4.2	Soorten Agile Testing	24
2.4.3	Monitoring	25
3	Methodologie	27
4	Conclusie	29
A	Onderzoeksvoorstel	31
A.1	Introductie	31
A.2	State-of-the-art	31
A.3	Methodologie	32
A.4	Verwachte resultaten	33
A.5	Verwachte conclusies	33
	Bibliografie	35

Lijst van figuren

2.1	Voorbeeld van een DOM structuur.	17
2.2	Google searches van PHP (blauw), Apache (rood) en Node.js (geel) in de afgelopen 5 jaar.	20

Lijst van tabellen

1. Inleiding

1.1 Probleemstelling

Het zoeken naar fouten in je software kan een frustrerend, hectisch en vermoeiend proces zijn. Daarom zijn, als ontwikkelaar zijnde, toepassingen die dat proces voor jou kunnen automatiseren of beter analyseren altijd welkom. Kayzr, een Belgische start-up die zich bezighoudt met het mainstream maken van E-Sports in de Benelux en het organiseren van toernooien ervan, wenst zulke software zodat hun productieproces versneld kan worden. Het concreet probleem gaat als volgt: Kayzr's processen werden op verschillende plekken gemonitord. De tools zijn incompleet, en daardoor heeft men een gefragmenteerde collectie aan logs, foutmeldingen, informatie, enz... Per API call moet men foutmeldingen controleren in de terminal, IP-adressen in het Google Cloud platform, andere nuttige informatie staat dan weer eens ergens anders opgeslagen. Hierdoor duurt het lang om fouten te analyseren en te komen tot een mogelijke oplossing. Een dergelijke tool zoals PM2, een monitoringstoepassing met veel meer functionaliteiten, meer dan dat ze nodig hadden, is meteen een kost dat ze zich niet kunnen permitteren. Aan €50 per server per maand, kost hen dit met een vijftigtal servers al te veel. Deze kost zou schalen zeer moeizaam of zelfs onmogelijk maken.

1.2 Onderzoeksvraag

De onderzoeksvraag gaat dus als volgt: bestaat er een mogelijkheid om de monitoringstechnieken van Kayzr, en mogelijks andere bedrijven, op een goedkope manier te stroomlijnen en verbeteren binnen een NodeJS omgeving, zodat hun debugproces geoptimaliseerd kan worden?

1.3 Onderzoeksdoelstelling

Kayzr is tevreden met de toepassing en gaat in hun project hiervan gebruik maken. De gemiddelde tijd om een probleem op te lossen is gedaald en dit proces verloopt efficiënter. De data die per proces is verzameld kan op een goede manier gevisualiseerd worden zodat ze die ook voor andere doeleinden kunnen hanteren.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 volgt een literatuurstudie dat een groot deel van Javascript en NodeJS, Express en middleware gaat omvatten

In Hoofdstuk 3 bestuderen we het huidige proces. Hoe verloopt dit? Wat kan er beter? Wat kan er sneller? Welke functies moet de nieuwe toepassing bevatten? Na deze studie schrijven we software naargelang de requirements, en toetsen we die op verschillende factoren aan de oude processen.

In Hoofdstuk 4, wordt er gekeken of er een oplossing bestaat op deze onderzoeksvragen. We kijken of Kayzr deze gaat gebruiken naar de toekomst toe en wat er gaat gebeuren met de eigendom van dit softwareproject.

2. Stand van zaken

Dit hoofdstuk bestaat uit een zeer uitgebreide literatuurstudie, waarin de kern van het probleem wordt opgesplitst in verschillende delen. In het eerste deel wordt er uitleg gegeven over JavaScript zelf en diens evolutie. Vervolgens volgt er een groot deel over de frameworks Node.js en Express, erna bekijken we het belang van testen en monitoren van software en breiden we uit naar een analyse van zulke monitoringssoftware. In het tweede deel wordt er onderzoek gedaan naar de vraag van Kayzr. Wat wordt er verwacht, welk inzicht moet er gegeven worden, welke features moeten worden uitgewerkt.

2.1 Javascript

Javascript is een programmeertaal gemaakt voor het web, waarmee u statische webpagina's kan omzetten naar dynamische en interactieve websites. Doordat het een enorm krachtige scripttaal is dat speciaal werd ontwikkeld om de functionaliteiten van een doorsnee HTML/CSS-pagina uit te breiden, wordt het bijna onmogelijk om nog iets in te beelden dat niet geïmplementeerd kan worden m.b.v. Javascript. De taal is weakly-typed, functioneel, event-driven en dynamisch. JavaScript bevat een bibliotheek van standaardobjecten samen met specifieke taalelementen zoals operatoren, controlestructuren, enz... Deze kern of 'core' kan makkelijk uitgebreid worden met extra objecten waardoor men JavaScript makkelijk kan specificeren als *Client-side JavaScript* en *Server-side JavaScript*. Client-side JavaScript wordt bijvoorbeeld gebruikt om mooi, visueel, de toeters en bellen van de website mooi te maken, of om te reageren op muiskliks, en nog veel meer. Server-side JavaScript houdt zich eerder bezig met het achterliggende van een site, bijvoorbeeld om een webapplicatie te doen communiceren met een databank. Een zeer bekende en veel gebruikte Server-side JavaScript variant is Node.js, waarover later meer uitleg wordt gegeven.

(„What is JavaScript?”, 2019)

2.1.1 Tijdlijn van JavaScript

Javascript heeft echter een hele evolutie achter de rug. Ontstaan in mei 1995, wordt de taal na vele updates nog steeds dagdagelijks gebruikt en kan het wereldwijde web niet meer ingebeeld worden zonder. Brendan Eich, de auteur van de taal, werkte in 1995 samen met Netscape Communications, de makers van de eerste grote webbrowser genaamd Netscape Navigator, om een taal te implementeren in hun browser waar webontwikkelaars gebruik van zouden kunnen maken. Java, een zeer zware programmeertaal met tal van functionaliteiten was de eerste keuze van Netscape, maar Eich schreef uiteindelijk zijn eigen idee uit van een scripttaal in nog geen 10 dagen en overtuigde Netscape om de lichte, schaalbare en Java-complementerende-taal te adopteren (Rangpariya, 2019). Javascript, toen onder de naam Mocha en vervolgens LiveScript, was geboren en zette de wereld van webontwikkelaars op zijn kop.

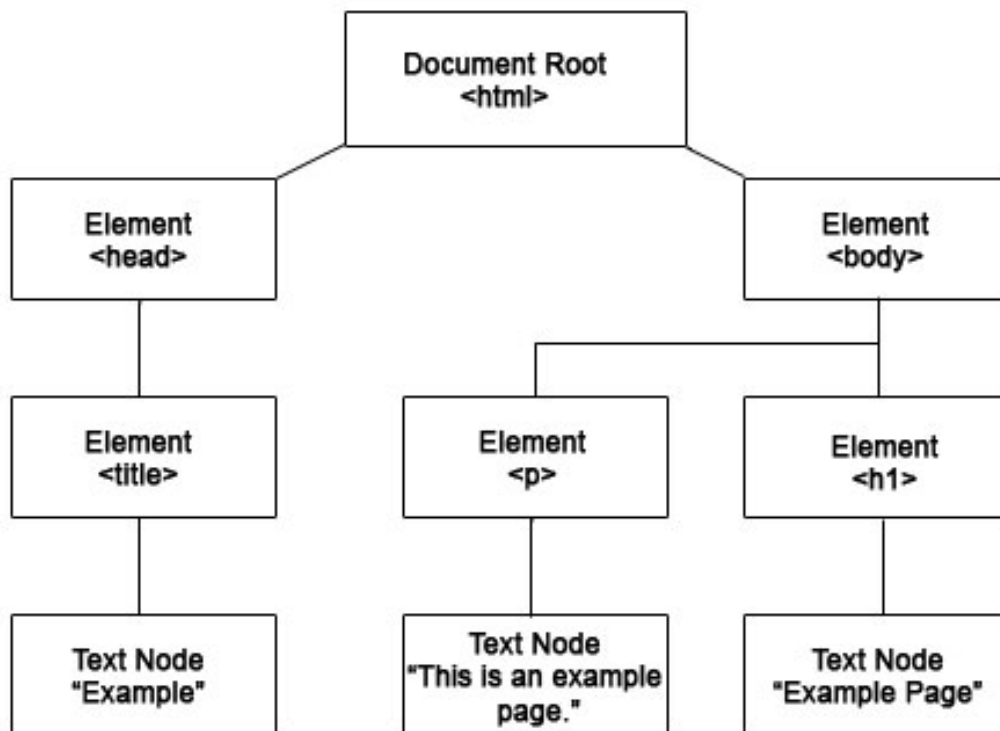
Netscape bracht nieuwere versies van JavaScript uit, maar Microsoft dreigde die te onttrenen. Niet zoveel later, bracht Microsoft Internet Explorer 3 uit met een eigen variant op Javascript, namelijk JScript. Dit was een zware slag voor Netscape, aangezien Microsoft hen hierdoor zou voorbijsteken, maar was wel een grote stap in de evolutie van JavaScript zoals wij het nu kennen. Dit bracht echter problemen met zich mee, want bedrijven zouden telkens eigen versies van JavaScript uitbrengen wat voor veel compatibiliteitsproblemen zou zorgen. Uiteindelijk, in 1997, werd JavaScript 1.1 gestandaardiseerd dankzij het European Computer Manufacturers Association en werd omgedoopt tot ECMAScript. (Wiley, 2016) Deze standaard, namelijk ES1 (versie 1), zou dan vertakkingen verhelpen. Implementaties van ECMAScript, waaronder JScript, ActionScript, maar dus ook JavaScript zelf, zouden dan telkens ECMAScript implementeren waardoor de core van de varianten telkens hetzelfde blijft. Nu, in 2019; is JavaScript nog steeds enorm populair en implementeert ondertussen al de 9e versie van ECMAScript: ES2018. Na ES5 is ECMAScript overgeschakeld naar jaarlijkse releases, startend vanaf ES2015.

2.1.2 Waarom JavaScript?

JavaScript is niets voor niets enorm populair, zelfs nog steeds na al die jaren, dankzij de vele voordelen dat de taal met zich meebrengt. En die lijst van voordelen wordt per iteratie alleen maar groter. Hieronder worden er enkele kernvoordelen opgesomd.

Dynamisch

Een dynamisch getypeerde taal slaat neer op het feit dat het type van waarden kunnen veranderen tijdens uitvoertijd. Dit betekent bijvoorbeeld dat een variable string plots kan gebruikt worden om een som te maken met een getal. Dit maakt uitvoertijd en compileertijd aanzienlijk sneller doordat er niet voortdurend controles moeten plaatsvinden.



Figuur 2.1: Voorbeeld van een DOM structuur.

DOM-manipulatie

De DOM, ofwel Document Object Model, is kortweg het skelet van een HTML-pagina. De structuur van het skelet beschrijft een boomstructuur, waardoor een element een ouder, broer, kind, kleinkind, achterkleinkind, enz... van een element kan zijn. Het HTML element is de wortel van de boom. Client-side JavaScript wordt grotendeels gebruikt om deze boomstructuur te manipuleren. Hierdoor kunnen we HTML elementen toevoegen, verwijderen, stijlen manipuleren, enz... allemaal tijdens uitvoertijd. Kantor (2017)

Prototype-based

In JavaScript kan een object eigenlijk aanzien worden als een Array. Hierdoor kunnen ze heel gemakkelijk aan de eigenschappen van een object aan en kunnen we deze ook makkelijk tijdens uitvoertijd aanpassen. Ook kunnen we op deze manier makkelijk eigenschappen aan bestaande objecten toevoegen. In JavaScript wordt gebruikt van de 'dot-notatie' of van de 'haakjesnotatie'.

```
foo.bar = 10;
foo['bar'] = 10;
const bar = foo.bar;
foo['bar2'] = bar;
```

Event-driven

JavaScript is event-driven, wat betekent dat de code vooral kijkt naar acties van de gebruiker. Bijvoorbeeld: Wanneer een gebruiker op een knop klikt, wil je dat met een animatie de knop heen en weer beweegt, tekst op je scherm verschijnt, en een server-request wordt gestuurd.

Functioneel

Sinds ES2015 is JavaScript niet enkel en alleen een object-georiënteerde taal, maar ook een functionele taal. JavaScript was zeer snel met het introduceren van functioneel programmeren (waaronder de populaire arrow-functions). Dit vergemakkelijkte de taal aanzienlijker, aangezien veel minder code geschreven moest worden om hetzelfde resultaat te bereiken.

Universeel

JavaScript wordt sinds ES5 ondersteund door alle moderne browsers! Of er nu in Chrome, Firefox, Safari, Edge of het oude Internet Explorer¹ wordt gesurft, ze ondersteunen allemaal JavaScript. D.m.v. de console in de browser te openen kan er al geprogrammeerd worden.

2.2 Node.js

Node.js is een open-source JavaScript runtime omgeving dat wordt gebruikt voor Server-side toepassingen. Node.js maakt het mogelijk, (of eerder *makkelijker*), om JavaScript nu ook te gebruiken buiten websites maken. Aangezien JavaScript zo'n krachtige taal is, hadden de ontwikkelaars van Node.js een ding in gedachten: JavaScript niet enkel in de browsers, maar ook als alleenstaande applicatie. JavaScript sindsdien evenaren aan andere scripttalen, zoals Python. Patel (2018)

2.2.1 Tijdlijn van Node.js

Node.js' verhaal start in 2009, toen een ontwikkelaar genaamd Ryan Dahl het niet zo goed stelde met Apache Http Servers.

Zoals beschreven in (Chaniotis, Kyriakou & Tselikas, 2015), was (en is in grote mate nog steeds) Apache HTTP, in combinatie met PHP, jarenlang de go-to-taal om webapplicaties te laten communiceren met databases en server-side functionaliteiten toe te voegen aan webapplicaties, zoals authenticatie, file-en-ftp servers, logging en nog veel meer. PHP is echter nooit in het achterhoofd ontwikkeld geweest om hedendaagse, complexe webapplicaties te schrijven. Ten tweede was Apache incapabel om te schalen naar meerdere processorkernen, waardoor de performantie enorm daalde. PHP in combinatie met Apache was gewoonweg

¹ Internet Explorer wordt niet meer verder ontwikkeld, dus ondersteunt het enkel ES2015.

niet gemaakt voor de volgende generatie webapplicaties. Andere struikelblokken zijn cross-platform mobiele applicaties en real-time communicatie.

Meer en meer applicaties worden gemaakt met cross-platform compatibiliteit in het achterhoofd, aangezien de toestroming van verschillende nieuwe besturingssystemen en apparaten het meer en meer onmogelijk maken om native apps² te ontwikkelen. Hierdoor worden bibliotheken en frameworks ontwikkelt zoals React Native, Flutter, Xamarin, Ionic, PhoneGap, enz... die deze nadelen kunnen overbruggen. Deze zijn echter zo krachtig geworden en schenken zoveel nieuwe voordelen, dat server-side talen en frameworks zoals Apache HTTP/PHP eerder een bottleneck vormden.

Real-time communicatie is nog een factor dat niet ondersteunt wordt door Apache. Apache was niet ontwikkelt om berichten te sturen naar de client-side. In de wereld van vandaag, waarin sociale media nog nooit zo'n grote invloed heeft gehad op ons dagelijks leven, is het haast ondenkbaar dat real-time communicatie niet zou kunnen bestaan.

Dit alles zorgde voor de ontwikkeling van Node.js. Dahl's project werd met open armen ontvangen. Na enkele struikelblokken, kende Node.js in 2011 voor het eerst het licht en wordt nu zo'n tien jaar later, gebruikt in vele moderne applicaties en is nog steeds één van de meest begeerde vaardigheden van een programmeur. (Patel, 2018)

Hoewel Apache en PHP nog steeds overduidelijk op plaats één staan, hebben ze hun populariteit enkel nog te danken aan bedrijven die op verouderde software werken, en hun populariteit bij senior ontwikkelaars. PHP blijft een geliefde taal bij velen, maar we zien dat Node.js populariteit duidelijk die van Apache aan het inhalen is. (SimilarTech, g.d.)

2.2.2 Waarom Node.js?

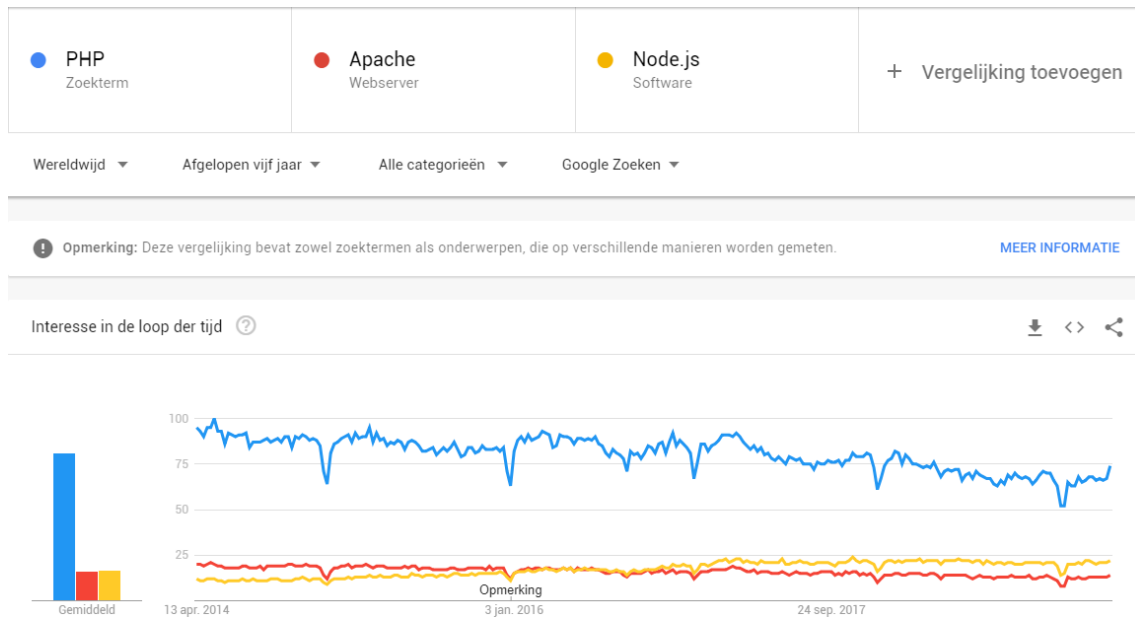
Node.js is reeds een volwassen server-side framework. Youtube, Yahoo, Google, Amazon, Netflix, Ebay, Reddit, LinkedIn, Paypal, Github, Forbes, Walmart, Uber, NASA, Slack,.. zijn slechts enkele van de duizenden websites die overgeschakeld zijn naar Node.js. En dit zijn geen kleine namen.

Illustrator BAT (2016) toont op een illustratieve wijze de voordelen van Node.js aan. LinkedIn kon hun 15 servers reduceren naar slechts 4 en terwijl de verkeerscapaciteit nog eens verdubbelen. Walmart kon al hun Client-side JavaScript processing naar hun servers verplaatsen, en Ebay kon hun verkeerscapaciteit enorm doen verhogen en hun verbruik enorm doen dalen. De voordelen en functionaliteiten worden hieronder nog eens opgesomd zoals aangegeven door Chandrayan (2017).

nieuw

JavaScript is relatief gezien een nieuwe taal dat voortdurend geüpdatet wordt. In tegenstelling tot traditionele server-talen zoals Python, PHP, enzovoort. Vele dialecten van

²Native Apps zijn applicaties die gemaakt worden met één besturingssysteem in het hoofd. Bv: Android, IOS.



Figuur 2.2: Google searches van PHP (blauw), Apache (rood) en Node.js (geel) in de afgelopen 5 jaar.

JavaScript (zoals TypeScript, CoffeeScript, ClojureScript,...) compileren dan gewoonweg in JavaScript waardoor overschakelen zeer simpel wordt. Doordat Node ook in JavaScript geschreven is, maakt dit het makkelijker om de client-server-side-brug over te steken als frontend ontwikkelaar of vice versa. („Express/Node introduction”, g.d.)

Asynchroon

Node.js is sterk afhankelijk van asynchrone en continue programmeerstijl. I/O-bewerkingen worden uitgevoerd door middel van oproepen naar asynchrone functies waarbij een callback moet worden doorgegeven om aan te geven hoe de berekening wordt voortgezet zodra de genoemde I/O-bewerking asynchroon is voltooid. Het Node.js executiemodel bestaat uit een hoofdgebeurtenislus die wordt uitgevoerd op een single-threaded proces. dankzij deze *event loop* moet een Node.js server nooit wachten op antwoord, het kan gewoon blijven doordraaien na een oproep van een API Call en kan dankzij een notificatiesysteem op een later moment het antwoord terugsturen. Dit maakt Node.js enorm schaalbaar, in tegenstelling tot Apache dat slechts een gelimiteerd aantal threads kan starten om handelingen uit te voeren.

Het is daarom niet altijd makkelijk om deze soort frameworks te debuggen, en het wordt uiteindelijk een uitdagende taak. Gelukkig zijn hiervoor dan ook weer verschillende hulpmiddelen en tools uitgebracht om dit te vereenvoudigen. Zoals hierboven vermeld in (Ancona e.a., 2017), kan asynchrone code subtiele bugs opleveren die niet meteen zichtbaar zijn. Hier is nog niet echt onderzoek over gedaan. Waarin er honderden vergelijkende studies bestaan over de frameworks zelf en of Node.js een goede optie is, zijn er geen of

amper studies over de beste manier om dit te debuggen en hoe men dit het best aanpakt. (Ancona e.a., 2017) vertelt ons meer over het identificeren van schaalbaarheidsproblemen en het aanrijken van mogelijke oplossingen. Ze maken gebruik van parametrische uitdrukkingen voor runtime monitoring van Node.js toepassingen, maar ze geven toe dat dit nog maar de eerste stap is en dat hier nog meer onderzoek naar kan gedaan worden. Hier wordt later in dit onderzoek dieper op ingegaan.

Performant

Google Chrome's JavaScript engine, V8, is bijzonder snel. Dahl heeft dit verder ontwikkeld voor Node.js en dit maakt de framework enorm performant en efficiënt in het compileren en uitvoeren van code. Veel sneller dan Python, Ruby of Perl.

npm

Node.js kent een enorm bloeiende community. Deel van Node is zijn Node Package Manager, ofwel npm, en komt meegeleverd bij de installatie. Via npm kan men uit honderdduizenden pakketjes kiezen voor een JavaScript/Node project makkelijk uit te breiden met extra functionaliteiten in één oogwenk. Dagelijks worden er nieuwe pakketjes toegevoegd aan de packet manager, en is makkelijk één van de redenen waarom Node.js zo populair is, aangezien het ontwikkelen van een webapplicatie enorm vereenvoudigd wordt.

2.3 Express

2.3.1 Waarom Express?

„Express/Node introduction” (g.d.) vertelt ons dat echter niet alles rechtstreeks door Node.js wordt ondersteund. Als men bijvoorbeeld routing aan een webapplicatie wenst toe te voegen, zoals HTTP *GET*, *POST*, *PUT*, *DELETE*, enzovoort... of URL-paden met specifieke requests, statische bestanden weergeven... moet men die code zelf schrijven. Herinner echter het npm-systeem van Node.js. Eén van die honderdduizenden pakketjes is namelijk Express, en is makkelijk *het* populairste Node framework en is zelfs de grondlegger van duizenden andere pakketjes. Express wordt om deze reden door bedrijven dan ook vaak in combinatie gebruikt met Node.js, waaronder Kayzr.

Request Handlers

Dankzij Express wordt het zeer gemakkelijk om routing in je applicatie toe te voegen. Indien een GET request gemaakt wordt in de frontend naar een bepaalde URL, is dit zeer gemakkelijk om op te vangen m.b.v. Express. Voor elke soort request en URL kan Express logica uitvoeren en code makkelijk verder delegeren.

Views

Express kan statische bestanden zoals HTML, CSS en afbeeldingen laten weergeven in de browser dankzij render engines. Makkelijk om bijvoorbeeld foutpagina's te weergeven.

Modulair

Modulariteit is een kerneigenschap van Express. Express is *unopinionated*, wat wil zeggen dat er niet echt een gouden regel is om een doel in Express te bereiken. Doelen kunnen op meerdere manieren bereikt worden, waardoor het makkelijker is voor ontwikkelaars om een bepaalde taak uit te voeren. Dankzij modulariteit kan de framework ook volledig ingesteld worden naar wens, van poorten en routers, het instellen van een databank tot het toevoegen van Express middleware.

Middleware

Wellicht de krachtigste functie van Express is de eigenschap tot het gebruik maken van middleware. Middleware wordt voortdurend gebruikt in Express. Wanneer een request binnenkomt, wordt deze doorgestuurd door nul, één of meerdere middleware die elk iets doen met de request, tot deze uiteindelijk wordt afgehandeld. De volgorde waarin een request de middleware doorloopt, is volledig afhankelijk van de programmeur. Net zoals Node.js, kent Express ook een enorm grote bibliotheek aan middleware die men kan gebruiken in een applicatie en is alsook helemaal verkrijgbaar via npm. Hierdoor kunnen ontwikkelaars zelf hun middleware schrijven, bestaande middleware downloaden en deze inpluggen in hun applicatie. Middleware kan gaan van Http-request loggers tot cookie-parsers en veel meer. De mogelijkheden zijn praktisch oneindig. Foutmeldingen zijn een goed voorbeeld van Express Middleware. Wanneer ergens iets fout gaat, roept Express de Error-Handler op, die dan zelf de request afwerkt. Monitoringstoepassingen in Node.js maken vaak gebruik van middleware. Hierover volgt meer info in het volgende hoofdstuk.

Listing 2.1: Express code flow voorbeeld.

```
1  var express = require('express');
2  var app = express();
3
4  // An example middleware function
5  var a_middleware_function = function(req, res, next) {
6    // ... perform some operations
7    // Call next() so Express will call the next middleware function in the
      chain.
8    next();
9  }
10
11 // Function added with use() for all routes and verbs
12 app.use(a_middleware_function);
13
14 // Function added with use() for a specific route
15 app.use('/someroute', a_middleware_function);
16
17 // A middleware function added for a specific HTTP verb and route
18 app.get('/', a_middleware_function);
```


19
20

```
app.listen(3000);
```

Middleware schrijven is op zich niet moeilijk. Men kan een enkele functie schrijven, of een heel takenpakket, die men dan als het ware injecteert in een request afhandeling. In een middleware functie verkrijgt men dan een req object, dat van de router kan komen of van een andere middleware, waar dan iets mee kan gedaan worden. Wanneer alles is uitgevoerd, roept de middleware de next-functie op, die dan het nieuwe res, of result, object doorstuurt naar de volgende middleware. Het kan ook zijn dat de middleware de next-functie niet oproept, wat betekent dat dit de laatste stop was van router.

2.4 Testen en Monitoring

Testen schrijven. Of men nu alleen aan software werkt, of in team in een bedrijf, elke programmeur met een gezond verstand zal kunnen vertellen over het belang van testen schrijven. Iets waar vaak wordt tegenop gekeken, maar het is nodig voor een geslaagd werkend product op te leveren. Meer en meer bedrijven gaan dit dan ook verplichten in hun implementatieproces.

Testen en monitoring zijn twee termen dat dicht bij elkaar horen. Bijna elke programmeur kent dan ook het belang van testen schrijven. D.m.v. testen te implementeren kan men kwaliteitsvoller code schrijven, fouten sneller opsporen en is de software een pak robuuster. Het toevoegen van nieuwe functionaliteiten kan code niet meer breken en daardoor kan software langer meegaan.

2.4.1 Agile Testing

Agile testing is een methodologie, ontstaan rond 1990, waarin software incrementeel en iteratief wordt uitgewerkt en dat op de dag van vandaag in vele bedrijven wordt gehanteerd. Vooraleer er effectieve code wordt geschreven, zullen er eerst testen gemaakt worden, waarop men dan de code gaat schrijven. Zo verkrijgt men code die robuust, aanpasbaar en efficiënt is. Tijdens elke iteratie van het projectproces, zullen er nieuwe testen worden geschreven, oude testen worden aangepast, en zo hoeft men telkens maar kleine gedeeltes code aan te passen of te schrijven. Een goede handhaving van Agile testing doet niet alleen de slaagkansen van het project vergroten, het kan het project ook goedkoper doen uitdraaien en de effectieve loopduur stabiel en klein houden. Per sprint, of ontwikkelcyclus, worden er telkens eerst testen geschreven voordat men dan uiteindelijk begint met programmeren. (Chakravorty, Chakraborty & Jigeesh, 2014)

Dit kan echter ook problemen met zich meebrengen. Agile vergt voorbereiding. Chakravorty e.a. (2014) vertelt ons dat de kost en tijd van het project kunnen uit balans geraken telkens de klant iets wilt veranderen en daardoor komt er veel werk op de programmeurs te liggen. In („Adopting Agile Testing, a Borland Agile Testing White Paper”, 2012) wordt dit ook nog eens samengevat. Veel bedrijven doen aan de Agile methodologie, maar niet

aan Agile Testing. Dit komt omdat in Agile de verantwoordelijkheid om beslissingen te nemen naar het team van ontwikkelaars wordt verschoven. Dit breekt met de traditionele manier van de waterval methode waar de belangrijkste keuzes door de leiding en niet door het team worden genomen. Dit breekt ook met de traditionele hiërarchische structuur waarin bedrijven werken. De leiding verliest een deel van de controle over de richting van hun projecten. Het is dus veilig om te zeggen dat het implementeren van de Agile methodiek niet alleen het werkproces van het bedrijf aanpast maar ook de mentaliteit van het bedrijf. Veranderingen in een bedrijf gaan log vooruit en zeker als het een gehele mentaliteitsverandering inhoud. Daarom dat er veel bedrijven zijn die er voor opteren om de Agile methodiek gedeeltelijk te implementeren. Zo laten ze een paar minder belangrijke componenten van de Agile methodiek achterwege zoals bijvoorbeeld Agile testing. Dit kan verklaren waarom er niet zo veel bedrijven zijn die Agile testing implementeren. Een andere grote hindernis bij Agile testing is dat er niet veel tools bestaan die geavanceerde testing methodes ondersteunen. Er bestaan heel veel verschillende soorten compilers die enorm uitgebreid zijn in de codetalen die ze ondersteunen en die veel verschillende programmeertechnieken ondersteunen. Dit heb je niet bij unit testing en komt grotendeels door de zware natuur van de testing tools.

Wat men ook in de afgelopen tien jaar meer en meer terugziet is de problematiek van het continue releasen van software. Software-reuzen zoals Google, Facebook, Mozilla, enzovoort... kozen om software telkens opnieuw uit te brengen in zeer korte periodes, ook wel *rapid releases* genoemd. Mozilla kende bijvoorbeeld een release-cyclus van maanden, waarin elke grote versie vele nieuwe dingen met zich meebracht. Vanaf Firefox 5.0 zijn ze overgeschakeld naar een rapid release systeem en daardoor brengen ze nieuwe, maar kleinere updates om de 6 weken uit. Dit heeft zijn voordelen, maar kent echter ook zijn nadelen in het test-proces. Aangezien er meer en meer sneller moet uitgebracht worden, moeten testen meer geautomatiseerd worden, wordt er minder getest en is de software minder robuust doordat fouten sneller tevoorschijn komen (Mäntylä, Khomh, Adams, Engström & Petersen, 2013). Kayzr kampt met dezelfde problemen. Daarom is er meer en meer nood aan monitoringstoepassingen die de duur van het debugproces van zo'n rapid release model aanzienlijk kunnen doen verminderen. Hierover volgt later meer info.

2.4.2 Soorten Agile Testing

Er bestaan natuurlijk verschillende manieren om te testen, maar het wordt grotendeels opgesplitst in drie categorieën, namelijk unit testing, integratie testing en systeem testing, maar er zijn er nog anderen. Sanity testing, smoke testing, interface testing, regression testing, end to end testing, performantie testing,... zijn allerlei soorten opgesomd door Pittet (2019). Dit zijn slechts de functionele testen. Men heeft ook niet-functionele testen, die dan betrokken zijn tot andere aspecten van het project, zoals beveiliging en lokalisatie (taal),... Dan wordt er nog eens gekozen of men deze soorten testen wenst te automatiseren of niet. Daarin kruipt meer tijd, maar goed geschreven geautomatiseerde testen zijn een sleutelcomponent tot continue integratie en releases. Hier wordt echter niet verder in detail op ingegaan aangezien dit niet tot de scope van het onderzoek behoort.

2.4.3 Monitoring

3. Methodologie

4. Conclusie

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Node.js is een backend Javascript framework wiens populariteit in de afgelopen jaren hard is toegenomen. Ontwikkelaars genieten van verschillende voordelen. Het werkt asynchroon, het is makkelijk schaalbaar en het is zeer functioneel. Doordat het Javascript is, kan elk besturingssysteem gebruik maken van de krachtige functies die Node.js te bieden heeft. Dit maakt het een uitstekend framework om webapplicaties te ontwikkelen. Node.js is echter niet makkelijk om te debuggen doordat het asynchroon is opgebouwd. Het toepassen van de juiste monitortechnieken kan de slaagkansen van een project echter goed verhogen, net als de levensduur van de applicatie. Op welke manieren kunnen we het monitoren van zulke applicaties aanpakken? Welke software en tools worden hiervoor gebruikt? Welke technieken worden het best toegepast? En kan er ook intern in het proces van Node.js gekeken worden om daaruit nuttige informatie te halen? En zijn al deze technieken drastisch veranderd sinds het framework werd uitgebracht in maart 2009?

A.2 State-of-the-art

Node.js is reeds een matuur framework. Zoals gezegd wordt in (Ancona e.a., 2017): Node.js is sterk afhankelijk van asynchrone en continue programmeerstijl. I/O-bewerkingen

worden uitgevoerd door middel van oproepen naar asynchrone functies waarbij een callback moet worden doorgegeven om aan te geven hoe de berekening wordt voortgezet zodra de genoemde I/O-bewerking asynchroon is voltooid. Het Node.js executiemodel bestaat uit een hoofdgebeurtenislus die wordt uitgevoerd op een single-threaded proces. Het is daarom niet makkelijk om deze soort frameworks te debuggen, en wordt uiteindelijk een uitdagende taak. Gelukkig zijn hiervoor dan ook weer verschillende hulpmiddelen en tools uitgebracht om dit te vereenvoudigen. Zoals hierboven vermeld in (Ancona e.a., 2017), kan asynchrone code subtiele bugs opleveren die niet meteen zichtbaar zijn. Hier is nog niet echt onderzoek over gedaan. Waarin er honderden vergelijkende studies bestaan over de frameworks zelf en of Node.js een goede optie is, zijn er geen of amper studies over de beste manier om dit te debuggen en hoe men dit het best aanpakt.. (Ancona e.a., 2017) vertelt ons meer over het identificeren van schaalbaarheidsproblemen en het aanrijken van mogelijke oplossingen. Ze maken gebruik van parametrische uitdrukkingen voor runtime monitoring van Node.js toepassingen, maar ze geven toe dat dit nog maar de eerste stap is en dat hier nog meer onderzoek naar kan gedaan worden. Doordat hier nog niet veel over staat neergepend, leek me dit een zeer interessant onderwerp dat hand in hand gaat met mijn stageopdracht. Kayzr, mijn stagebedrijf, zoekt namelijk zelf goede manieren om hun Node.js applicaties goed te kunnen debuggen, en deze studie zou zeker een goede bijdrage kunnen zijn aan deze soort doelgroep. Mijn onderzoek zal zich onderscheiden door een goed overzicht te behouden in die ongedocumenteerde zee van beschikbare frameworks, tools en software, waardoor deze studie kan gebruikt worden als een guideline voor best-practices.

A.3 Methodologie

We starten door literatuuronderzoek te doen naar de verschillende mogelijkheden van software en tools. Ook kunnen we literatuuronderzoek doen naar Javascript van ES1 naar ES6, NodeJS en zijn asynchrone processen, monitoring en het monitoren van async processen. Na de literatuurstudie maken we een steekproef van een aantal node-developers door hen te contacteren en via een enquête hen te vragen welke tools zij gebruiken om hun Node.js applicaties te monitoren, uitgebracht tussen 2009 en 2019. We kunnen erna kijken welke tools performanter zijn dan anderen dankzij het gebruik van de servers van Kayzr. We kunnen er uitgebreider onderzoeken door:

- Monitoren van api calls volgens het aantal keren opgeroepen
- Monitoren van api calls volgens duratie tot een response gestuurd wordt
- Het gemak om errors op te slaan en later te debuggen/analyseren
- Monitoren van deze node process en hun taxatie op de server waar ze op draaien (CPU, RAM, netwerk...)

De tools voor de bovenstaande metingen te testen gaan uiteraard de te onderzoeken tools, en hulpprogramma's als taakbeheer zijn. We maken daarna gebruik van R Studio om deze metingen en enquêtes om te zetten tot mooie data waaruit we hopelijk een conclusie kunnen trekken.

A.4 Verwachte resultaten

Ik vermoed dat de resultaten uiteen zullen lopen, en dat verschillende developers zich liever vasthouden aan hun eigen methodes. Ook zal er een breuk zijn tussen ontwikkelaars die liever oudere maar matuurdere tools zullen blijven gebruiken, en developers die liever het nieuwste van het nieuwste wensen te gebruiken. Dit kunnen we dan visualiseren a.d.h.v. een staafdiagram met op de x-as de tool en op de y-as het aantal developers.

Qua effectieve data van de overwogen tools zal elk waarschijnlijk zijn voor en nadelen hebben. Toch vermoed ik dat we er een effectieve winnaar gaan uit kunnen halen die over het algemeen goed scoort. Dit zullen we dan toetsen aan de literatuurstudie om zo best practices te bekomen.

A.5 Verwachte conclusies

De wereld van Javascript is nog in volle groei, maar is toch al een pak maturder dan vroeger. We zien dat de tools beter zijn geworden. De concurrentie is enorm groot aangezien Node.js een enorm populair ontwikkelaarsplatform is. We verwachten dat nieuwere tools meer functionaliteiten gaan bieden maar minder stabiel gaan zijn dan de reeds bestaande. De manier van aanpak zal variëren, maar uit de steekproef kunnen we dat toch veralgemenen.

Bibliografie

- Adopting Agile Testing, a Borland Agile Testing White Paper.* (2012). Micro Focus Limited. Verkregen van https://cs.anu.edu.au/courses/comp3120/public_docs/Borland_Whitepaper_v7_AgileTesting.pdf
- Ancona, D., Franceschini, L., Delzanno, G., Leotta, M., Ribaud, M. & Ricca, F. (2017). Towards Runtime Monitoring of Node.js and Its Application to the Internet of Things. In *Proceedings First Workshop on Architectures, Languages and Paradigms for IoT, ALP4IoT@iFM 2017, Turin, Italy, September 18, 2017.* (pp. 27–42). doi:10.4204/EPTCS.264.4
- BAT, M. (2016). Infographic on Node.js. Verkregen van https://coderwall.com/p/_ukoeg/infographic-on-node-js
- Chakravorty, T., Chakraborty, S. & Jigeesh, N. (2014). Analysis of Agile Testing Attributes for Faster Time to Market: Context of Manufacturing Sector Related IT Projects. *Procedia Economics and Finance*, 11, 536–552. Shaping the Future of Business and Society. doi:[https://doi.org/10.1016/S2212-5671\(14\)00219-6](https://doi.org/10.1016/S2212-5671(14)00219-6)
- Chandrayan, P. (2017). All About Node.js you wanted to know. Verkregen van <https://codeburst.io/all-about-node-js-you-wanted-to-know-25f3374e0be7>
- Chaniotis, I. K., Kyriakou, K.-I. D. & Tselikas, N. D. (2015). Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*, 97(10), 1023–1044. doi:10.1007/s00607-014-0394-9
- Express/Node introduction.* (g.d.). Mozilla. Verkregen van https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
- Kantor, I. (2017). Dom tree. Verkregen van <https://javascript.info/dom-nodes>
- Mäntylä, M. V., Khomh, F., Adams, B., Engström, E. & Petersen, K. (2013). On Rapid Releases and Software Testing. In *2013 IEEE International Conference on Software Maintenance* (pp. 20–29). doi:10.1109/ICSM.2013.13

- Patel, P. (2018). What exactly is Node.js? Verkregen van <https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5>
- Pittet, S. (2019). The different types of software testing. *Software testing*. Verkregen van <https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing>
- Rangpariya, N. (2019). Evolution of JavaScript: Revolution of Web Development. Verkregen van <https://medium.com/swlh/evolution-of-javascript-revolution-of-web-development-5df234a617e6>
- SimilarTech. (g.d.). NodeJs Vs PHP. Verkregen van <https://www.similartech.com/compare/nodejs-vs-php>
- What is JavaScript?* (2019). Mozilla. Verkregen van https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What_is_JavaScript
- Wiley. (2016). What is JavaScript? Verkregen van http://media.wiley.com/product_data/excerpt/88/07645790/0764579088.pdf