Onderzoek naar monitoring mogelijkheden van NodeJS toepassingen in 2019.

Onderzoeksvoorstel Bachelorproef

Michiel Leunens[1]

Samenvatting
De wereld van Javascript is in volle groei. Honderden frameworks vervangen ondertussen verouderde frameworks, en velen worden elk jaar op hun beurt weer vervangen. Een van die frameworks die nu al even bestaat en er in uitblinkt, is Node.js. Als een backend Javascript framework met een uitstekende open-source community, goede ondersteuning, schaalbaarheid en snelle asynchrone bouwstenen, is dit de favoriet van vele backend webontwikkelaars. Het debuggen van zo'n framework, kan echter een ambetante zaak zijn. Kunnen die Javascript frameworks wel goed gemonitord worden? Deze studie omvat een totaal onderzoeken en uitwerkingen van de mogelijkheden om NodeJS toepassingen te monitoren. Dit enerzijds op gebied van bestaande tools en wat de mogelijkheden ervan zijn, alsook zelf een onderzoek doen naar wat er bekenen kan worden intern in het proces van NodeJS. De doelstelling van dit onderwerp zal de lezer helpen met het debuggen van NodeJS applicaties door te onderzoeken wat de best practices zijn. Uit een steekproef bestaande uit verschillende webontwikkelaars zal gekeken worden welke tools er het populairst zijn. Deze tools zullen dan aan verschillende testen onderdaan worden. Die hulpmiddelen gaan we toetsen in de praktijk aan de hand van het monitoren van API calls en andere testen. Ook zal een uitgebreide literatuurstudie aantonen welke methodieken er het best toegepast worden bij het analyseren en debuggen van zulke frameworks. Dit zal getoetst worden aan de praktijk. Er wordt verwacht dat uit de overvloed aan hulpmiddelen er een paar gaan uitspringen. Het zal variëren van software die al minstens vijf jaar meegaat tot software die nog geen jaar op de markt verschenen is. De conclusie zal ontwikkelaars een goed overzicht geven van best-practices en software die het beste gebruikt worden. Natuurlijk zal dit onderzoek geen jaren standhouden, aangezien javascript in de komende jaren nog veel evoluties zal meemaken. Maar het is eerder de bedoeling om toch een tijdelijk goed overzicht te geven.

Sleutelwoorden
Onderzoeksdomein. Webapplicatieontwikkeling — NodeJS — Monitoren

Co-promotor
Michiel Cuvelier[2] (KAYZR)

Contact: [1] michiel.leunens.y7743@student.hogent.be; [2] michiel@kayzr.com;

Inhoudsopgave

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 portabel. 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?

2. State-of-the-art

Node.js is reeds een matuur framework. Zoals gezegd wordt in (**Runtime2017**): 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 (**Runtime2017**), 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.. (**Runtime2017**) 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.

3. Methodologie

Onderzoek doen naar de verschillende mogelijkheden van software en tools. Een steekproef maken van een aantal node-developers door hen te contacteren en via een enquete vragen welke tools zij gebruiken om hun Node.js applicaties te monitoren, uitgebracht tussen 2009 en 2019. We kunnen erna kijken welke tools beter performeren 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 enquetes om te zetten tot mooie data waaruit we natuurlijk een conclusie kunnen trekken.

Ook kunnen we literatuuronderzoek doen naar Javascript van ES1 naar ES6, NodeJS en zijn asynchrone processen, monitoring en het monitoren van async processen.

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 duidelijk een breuk zijn in ontwikkelaars die liever blijven met hun vertrouwde tools die al een aantal jaar meegaan, en developers die liever het nieuwste van het nieuwste wensen te gebruiken. Dit zal het overzichtelijkst zijn in een bar chart 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.

5. Verwachte conclusies

De wereld van Javascript is nog in volle groei, maar is toch al een pak matuurder dan vroeger. We zien dat de tools beter zijn geworden. De concurrentie is enorm groot aangezien Node.js een enorm populair ontwikkelaarsplatform is. Tool X zal over het algemeen de beste tool zijn om te gebruiken. De manier van aanpak gaat variëren, maar uit de steekproef kunnen we dat toch kunnen veralgemenen.