|  |
| --- |
| Fontys Hogescholen |
| Personal Development Report |
| Kevin Tinnemans S62-3 |

|  |
| --- |
| Tinnemans,Kevin K.J.H.  19-6-2021 |

# Samenvatting

In dit document zal ik voor elke learning outcome zoals deze beschreven staan in de Canvas course laten zien wat ik heb gedaan om aan et tonen dat ik hieraan gewerkt heb semester. Dit zal gebeuren door middel van screenshots, links en tekst. Ook zal de feedback van de docenten die ik door het semester heen heb ontvangen meegenomen worden om te laten zien wat zij van mijn gedane werk vonden.

Voor elke learning outcome heb ik mezelf ook beoordeeld. Ik vind dat ik elk leerdoel met een proficient heb afgerond op het leerdoel Cloud Services na, deze heb ik naar mijn mening op advanced staan. Ik ben tot deze beoordelingen gekomen door te kijken hoe ik elk leerdoel aan moet tonen in Canvas en door te kijken naar de implementaties van mijn medestudenten.

Aan het einde van het document zal ik terugblikken op het semester en in de bijlagen bevinden zich nog een aantal documenten die de informatie uit de hoofdstukken doet aanvullen.

Inhoudsopgave

[Samenvatting 1](#_Toc75122412)

[Introductie 3](#_Toc75122413)

[Leerdoelen 4](#_Toc75122414)

[Developing Enterprise Software as a Team Effort 5](#_Toc75122415)

[Context Based Research 9](#_Toc75122416)

[Preparation for Life-Long learning 12](#_Toc75122417)

[Scalable Architectures 14](#_Toc75122418)

[Development and Operations (DevOps) 21](#_Toc75122419)

[Cloud services 27](#_Toc75122420)

[Security by Design 34](#_Toc75122421)

[Distributed Data 38](#_Toc75122422)

[Terugblik 40](#_Toc75122423)

[Conclusie 41](#_Toc75122424)

[Bijlagen 42](#_Toc75122425)

[Bijlage 1: Functional requirements 42](#_Toc75122426)

[Bijlage 2: Bcrypt hashes 45](#_Toc75122427)

[Bijlage 3: Security by design 46](#_Toc75122428)

[Bijlage 4: Performance testen 54](#_Toc75122429)

[Bijlage 5: Non-functional requirements 57](#_Toc75122430)

# Introductie

In dit document wordt mijn persoonlijke ontwikkeling in semester 6 gedocumenteerd. Dit document zal vier keer worden uitgebreid met nieuwe informatie en nieuwe inzichten die ik heb verkregen in zowel het individuele project als het groepsproject.

Voor het individuele project ga ik de Kwetter casus maken.

Kwetter is een webapplicatie die het voor gebruikers mogelijk maakt snel alles te delen wat in ze opkomt. Een gebruiker kan namelijk een bericht plaatsen van maximaal 160 tekens. Deze tweet wordt dan gedeeld op de tijdlijn van iedereen die deze persoon volgt. Ook kan een gebruiker een andere gebruiker taggen in een tweet. Alle andere gebruikers kunnen dan een tweet liken of erop reageren.

Ook heeft elke gebruiker een profielpagina. Hierop staan, naast een overzicht van alle tweets van deze gebruiker, ook nog andere gegevens als de locatie, website en een korte biografie. Een gebruiker kan ook een profielfoto toevoegen aan zijn/haar account.

Middels de zoekbalk kunnen alle tweets gevonden worden die ooit geplaatst zijn. Door een zoekopdracht in te voeren met het @-teken ervoor kan gezocht worden op andere gebruikers. Je kan er dan voor kiezen deze gebruiker te volgen of de profielpagina te bezoeken.

De frontend van de applicatie gaat gebouwd worden in React/JS. De backend zal bestaan uit verschillende (micro)services. Deze worden geschreven in Java met behulp van het Springboot framework. Voor het direct plaatsen van tweets op de tijdlijn van andere gebruikers zal er eventueel gebruik worden gemaakt van websockets.

Voor de CI/CD van het project wordt gebruik gemaakt van Gitlab. Meer informatie hierover komt een latere sprint.

Voor het groepsproject gaan werken aan de Connected Airlines casus. Wij zullen hierbij het werelddeel Oceanië op ons nemen.

Connected Airlines is een webapplicatie waarop gebruikers tickets kunnen boeken voor hun vlucht. In de applicatie kunnen klanten een tickets kopen, een stoel reserveren en extra bagage inchecken. Ook kunnen de klanten een hotel boeken of een huurauto reserveren, voor op hun vakantiebestemming. Er zit een betaalsysteem verwerkt in de applicatie waarmee klanten hun bestelling af kunnen rekenen.

Er zal ook een flight tracker komen waarop iedereen de verschillende vluchten live kan volgen. Wanneer dan een vlucht geselecteerd wordt, zal er meer informatie over deze vlucht weergegeven worden.

Medewerkers van de luchtvaartmaatschappij krijgen de mogelijkheid vluchten in te plannen en de prijzen van vluchten aan te passen. Ook kunnen zij passagiersinformatie opvragen en klanten overboeken naar andere vluchten.

Ook zal er een Interpol systeem aangehangen worden. Hiermee kunnen medewerkers van Interpol aangeven dat een crimineel een ticket geboekt heeft. De luchtvaartmaatschappij kan dan meer informatie geven over de vlucht waarop de crimineel zit.

# Leerdoelen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Fase 1** | **Fase 2** | **Fase 3** | **Fase 4** |
| **Developing Enterprise Software as a Team Effort** | Undefined | Beginning | Proficient | Proficient |
| **Context Based Research** | Undefined | Orienting | Beginning | Proficient |
| **Preparation for Life-Long Learning** | Undefined | Orienting | Proficient | Proficient |
| **Scalable Architectures** | Undefined | Beginning | Beginning | Proficient |
| **Development and Operations (DevOps)** | Undefined | Beginning | Proficient | Proficient |
| **Cloud Services** | Undefined | Orienting | Proficient | Advanced |
| **Security by Design** | Undefined | Beginning | Beginning | Proficient |
| **Distributed Data** | Undefined | Beginning | Proficient | Proficient |

## Developing Enterprise Software as a Team Effort

Dit leerdoel beslaat eigenlijk het ontwikkelen van software dat voldoet aan alle eisen en protocollen. Er moet rekening gehouden worden met eisen van de opdracht, wetgeving en testen. Ook moet er rekening gehouden worden met het ontwikkelen van software op een agile manier. Verder is het voor dit leerdoel belangrijk om goed te kunnen samenwerken en ook professionaliteit.

**Fase 1**

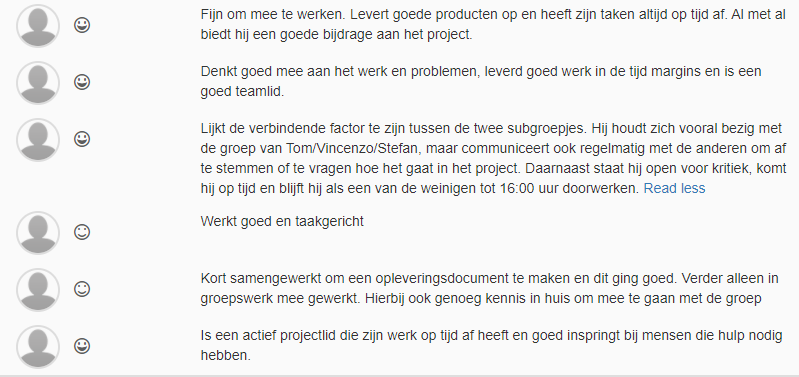
Op dit moment heb ik al redelijk wat kennis in het ontwikkelen van software. Toch moet ik hier nog grote stappen in zetten om dit op een zo professioneel mogelijke manier te doen. Ook het rekening houden met de AVG-wet en ethische vraagstukken zullen nog een uitdaging zijn.

**Fase 2**

Uit de retrospective van sprint 1 is gebleken dat we steeds beter bezig zijn met het werken op een Agile manier. We starten elke dag met een daily standup en ook de tools die zorgen voor overzicht op het project worden goed gebruikt. Steeds vaker allemaal de camera’s aan te hebben draagt ook bij aan de professionaliteit van de groep en mij in het individueel.

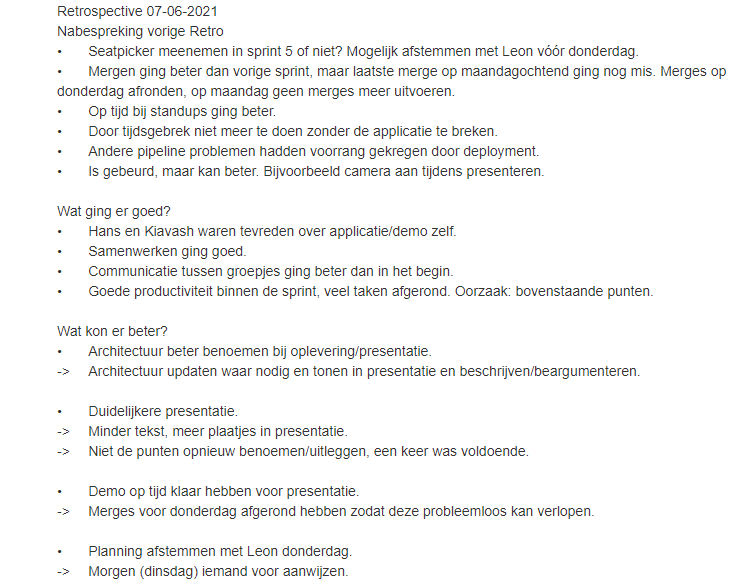
**Fase 3**

Het scrum proces wordt binnen onze projectgroep steeds beter nageleefd. We nemen de verschillende aspecten van dit proces serieus en op veel punten zijn we als groep ook al de goede kant op gegaan. De retrospectives geven ons nieuwe inzichten om ons nog meer te verbeteren en er wordt ook goed omgegaan met de feedback. We hebben de afgelopen week ook een peer review gedaan. Hieruit blijkt dat mijn groepsgenoten zeer tevreden zijn met werkzaamheden. Zo word ik door verschillende projectleden meerdere malen gecomplimenteerd over mijn kennis en bereidheid om bij te springen. Ook heb ik van een lid de zeer positieve feedback gekregen dat ik de “verbindende factor tussen de twee subgroepjes”. Hoewel het nooit fijn is om binnen een groep met twee subgroepen te zitten, is het toch fijn om te horen dat mijn peers erkennen dat ik er alles aan doe om deze kloof te overbruggen.



**Fase 4**

Uit de laatste retrospectives van de proftaak komt steeds beter naar voren dat we goed aan het samenwerken zijn. We krijgen steeds meer af en zijn ondanks het lastige begin toch goed op weg om een mooi eindproduct neer te zetten. De belangrijkste punten van de retrospectives hebben we ook altijd gedeeld met de docenten op Feedpulse. Een voorbeeld van deze punten:



Mijn toevoeging aan het project:

* Het aanmaken van vluchten (samen met Tom)
* Een overzicht geven van alle vluchten (samen met Tom)
* Real-time tracker maken voor vliegtuigen (samen met Tom)
* Policereport service inclusief messaging (samen met Vincenzo)
* Opstellen security by design document (samen met Vincenzo)
* Deployment van de microservices
* CI/CD pipelines opstellen in Azure DevOps
* Creëren van een roadmap (samen met volledige groep)
* Context based research DevOps (samen met Stefan, Vincenzo en Tom)
* Maken en geven verschillende presentaties
* Verschillende case studies

Aan het einde van elke sprint hebben we altijd twee opleveringen gegeven. Een oplevering wordt gegeven aan onze docenten Hans en Kiavash. De feedback uit de laatste oplevering (sprint 4) is als volgt:



Uit de feedback van Hans en Kiavash blijkt dat ze weinig te klagen hebben. Bij een aantal opleveringen hebben we te maken gehad met merge problemen vlak voor de opleveringen, dit hebben we vanaf sprint 4 voorkomen door op tijd te mergen en de presentatie goed voor te bereiden. De inhoudelijke technische feedback van Hans en Kiavash gaat eigenlijk alleen over het deployen van de microservices, iets waar we in de laatste sprint hard aan hebben gewerkt

De tweede oplevering geven we elke sprint aan product owner Leon en docente Monique. De feedback van de 4de oplevering is als volgt:



Vanwege het ontbreken van Monique hebben we alleen feedback gekregen van Leon. Uit deze feedback is gebleken dat we een redelijk goed product op kunnen gaan leveren. Hoewel niet alle functionaliteiten optimaal geïmplementeerd inzitten hebben we wel een goede werkende Enterprise applicatie. In de laatste sprint moeten we vooral gaan letten op het netjes maken van de code en een vlekkeloze oplevering. De laatste user story die we echt nog toe hebben moeten voegen is de implementatie van de API’s van andere groepjes.

**Conclusie**

Ik vind dat ik heb aangetoond dit leerdoel gehaald heb. In het begin van het semester zijn we verplaatst van de ene klas naar de andere. Hierdoor hebben wij een lastig begin gekend betreffende de samenwerking van een groep waarmee we samengevoegd zijn. Naarmate het semester vorderde hebben we laten blijken dat ondanks de verschillende meningen en aanpak methodes we toch een goed product neer kunnen zetten. Uiteindelijk hebben we alle docenten tevreden gesteld met het door ons opgeleverde product. We hebben een product neergezet waar we trots op mogen zijn.

## Context Based Research

Context Based Research houdt in dat er grondig onderzoek uitgevoerd wordt in combinatie met je eigen ethische normen. In de onderzoeken wordt gebruik gemaakt van breed geaccepteerde onderzoeksmethoden zoals de ictresearchmethods.

**Fase 1**

Ik heb al vaker onderzoek gedaan naar onderwerpen die bepalend zullen zijn voor de ontwikkeling van software. Zowel op stage als in eerdere semesters heb ik hier al aan gewerkt. Toch denk ik dat ik mijn onderzoeken nog grondiger uit kan voeren en zo tot betere resultaten kan komen.

In deze fase heb ik samen met Vincenzo, Tom en Stefan gewerkt aan de case study over Enterprise Architecture & User Expectations. We zijn hier dieper ingegaan op het gebruik van event storming en het modelleren van software met behulp van het C4-model. Aangezien we in deze week tussen twee klassen inzaten hebben we hier helaas geen feedback op ontvangen.

**Fase 2**

Uit de retrospective van sprint 1 is gebleken dat wij als groep tevreden zijn over de manier waarop de case studies worden uitgevoerd. Vanuit docenten wordt nog iets meer diepgang verwacht bij de onderzoeken. Dit zal ik dus aan moeten gaan tonen bij het onderzoek over Azure DevOps wat we binnenkort uit gaan voeren.

Voor de case study hebben we met dezelfde groep als vorige keer de “shaping your technical architecture” case study onder handen genomen. We hebben hier verschillende architectuurstijlen onderzocht. We leggen het verschil uit tussen Web-Queue-Worker, microservices en N-Tier architectuur en in welke projecten welke architectuur het beste gebruikt kan worden. We hebben onze bevindingen besproken met Kiavash en zijn feedback verder verwerkt in ons onderzoek. Kiavash heeft ons nog een aantal extra linkjes gegeven om ons onderzoek uit te breiden, dit hebben we dan ook gedaan.

**Fase 3**

Deze fase zijn we begonnen met ons grote onderzoek. Dit onderzoek gaat over DevOps en ik merk dat wij als groep hier een mooi onderzoek neer gaan zetten. Persoonlijk ben ik bezig met de deelvraag “wat zijn de best practises voor CI/CD?” te beantwoorden en ik zal hierbij gebruik maken van de “best good and bad practises en community research” methodes van ictresearchmethods. Het onderzoek zal afgerond worden in een van de volgende weken en de feedback die wij dan krijgen zal ik meenemen in de laatste fase van dit rapport.

Ook heb ik veel onderzoek gedaan voor bepaalde technieken die ik gebruik in mijn individuele project. Zo heb ik ontdekt dat er een GUI bestaat voor Kubernetes, genaamd Kubesail. Deze GUI maakt het werken met Kubernetes een stuk makkelijker en Hans was zeer te spreken hierover. Hij had namelijk nog geen idee dat zoiets bestond en was dan ook zeer content met het onderzoek wat ik hiervoor heb verricht.

Deze sprint hebben we gewerkt aan de case study over secutiry. In deze case study hebben we alle fases van softwareontwikkeling doorlopen en hoe security kan worden toegepast in elk van deze fases. Onze uitkomsten hebben we besproken met Hans. De feedback van Hans was als volgt:



**Fase 4**

In deze fase hebben we onze laatste case study gedaan. Deze ging over ethiek en GDPR. We kregen hier 3 scenarios waarbij we voor elk scenario een goede oplossing moesten vinden voor een probleem in een bedrijfscontext waarbij we rekening houden met zowel ethiek als wetgeving. Vanwege het missen van een reactie van Kiavash hebben wij geen feedback kunnen krijgen op deze case study.

**Conclusie**

We hebben de afgelopen maanden veel gewerkt met onderzoeken, zogenaamde case studies. Hieruit is nieuwe informatie naar boven gekomen die we gebruikt hebben in de proftaak, maar ook in ons individueel project. Als ik eerlijk ben en kijk naar de tijd die we erin hebben gestoken ben ik niet echt onder de indruk van hoeveel nieuwe informatie het heeft gebracht. Misschien hadden wij zelf onze onderzoeken groter aan kunnen pakken waardoor we nog meer geleerd hadden, maar vanwege de grote tijdsdruk dit semester zijn we hier niet aan toegekomen.

We hebben wel bij alle case studies gebruikt gemaakt van de *seven step approach*. Dit is een techniek om in te delen hoe je een probleem op gaat lossen, het probleem in dit geval een onderzoeksvraag bij de case study. Deze techniek is mij goed bevallen en zorgt ervoor dat ieder teamlid nieuwe informatie meebrengt.

Om de case studies uit te voeren hebben we vooral gebruik gemaakt van de Literature study methode uit de Library strategie. We hebben namelijk vooral veel informatie opgezocht over nieuwe onderwerpen waar we weinig vanaf wisten.

## Preparation for Life-Long learning

In dit leerdoel wordt gevraagd up-to-date te blijven van de laatste trends in de softwareontwikkeling industrie en deze toe te passen in de projecten waar deze van toegevoegde waarde zijn. Trends van dit moment zijn bijvoorbeeld Domain-Driven Design, Blockchain, Programming paradigms, Machine Learning, en Quantum Computing. Dit leerdoel gaat ook over het voorbereiden op het afstuderen en je loopbaan na het afstuderen.

**Fase 1**

Ik heb op dit moment nog niet de kans gehad om nieuwe trends te implementeren in een van mijn projecten. Ik heb in het eerste semester gewerkt met genetic algorithms en machine learning, maar dit was op kleine schaal.

**Fase 2**

In de proftaak zijn we op dit moment nog bezig met de basisopzet van de architectuur. We hebben daarom nog niet de luxe gehad om gebruik te maken van de nieuwste trends van softwareontwikkeling. Wel proberen we ons te houden aan de codestandaarden en werken we met innovatieve frameworks.

**Fase 3**

Ik heb de afgelopen fase grote stappen gezet bij het kiezen van een stageplek. Ik zal deze gaan lopen bij Infoland. Ik heb deze keuze gemaakt na meerdere gesprekken bij verschillende bedrijven. Infoland heeft een opdracht die op dit moment goed aansluit bij mijn kennis en waren ook zeer enthousiast over mijn toekomstplannen om software engineer te combineren met online marketeer. Ze willen mij, als de stage bevalt, graag ondersteunen met verschillende cursussen om zo veel mogelijk uit mijn carriere te halen. Wat betreft de sprint 5 onderwerpen lijkt het mij heel gaaf om meer te leren over machine learning. Ik heb in mijn allereerste semester op deze opleiding al eens gewerkt met genetic algorithms om een AI op te zetten die leert over verschillende iteraties en ik zou dit graag nog eens willen proberen met al mijn opgedane kennis van de afgelopen jaren.

**Fase 4**

Ik ben het hele semester in de veronderstelling geweest dat wij in sprint 5 een van de onderwerpen daadwerkelijk gingen toepassen in een project. Helaas bleek dit puur te gaan over een case study. Hierom heb ik ervoor gekozen om het onderwerp Blockchain te behandelen. Hier heb ik mindere theoretische kennis over en is sinds de opkomst van Bitcoin een van de grootste trends in de IT-wereld. Ook de opkomst van NFT’s (non fungible tokens) staat nauw samen met Blockchain. In de case study gaan we dan ook dieper in op de Blockchain technologie en hoe dit werkt in combinatie met Bitcoin. Helaas hebben we voor de deadline van het inleveren van dit PDR nog geen feedback ontvangen op ons onderzoek en kan ik hiervan dus niet de feedback meenemen.

**Workshops**

Door het semester heen heb ik verschillende workshops bijgewoond die me meer inzicht hebben gegeven in het leven van een software engineer. Hoewel ik, zoals ik in fase 3 heb aangegeven, niet de intentie heb om mijn hele leven te werken als software engineer ga ik hier natuurlijk wel de eerste jaren van mijn carrière mee aan de slag. De workshops die ik heb bijgewoond die me hier meer informatie over hebben gegeven zijn:

* Presentatie carrière software engineer
* Presentatie dag in het leven van een software engineer
* Presentatie voorlichting afstuderen

Doordat ik deze presentaties gevolgd heb, heb ik een beter beeld gekregen van hoe het leven van een software engineer eruit ziet. Een van de belangrijkste dingen die ik geleerd heb is dat veel dingen die we op Fontys leren niet altijd even belangrijk zijn in het bedrijfsleven. Ook gebruikt het bedrijfsleven veel tools waar we op de Fontys geen informatie over gekregen hebben. Dit betekent dat ik ook na mijn opleiding veel te leren heb en dat je als software engineer nooit uitgeleerd bent. Volgens de sprekers is de wens van de klant het aller belangrijkste en moet alles wijken voor dit doel. Hoewel dingen als testen en kwaliteit van de code wel belangrijk zijn, zetten bedrijven dit graag opzij als de klant dan beter zijn wensen vervult krijgt.

In de presentatie over de carrière van een software engineer gaf de spreker ook inzage in de arbeidsvoorwaarden die een software engineer kan verwachten. Een mooi salaris en goede secundaire arbeidsvoorwaarden lijken de standaard te zijn. Ook gaf hij aan dat wanneer je ambitie toont er ook veel doorgroeimogelijkheden zijn. Dit vind ik zelf fijn om te horen.

Ik vond deze informatie erg interessant aangezien ik op deze manier een goed beeld heb gekregen van wat ik mag verwachten van mijn carrière in de toekomst. In de wandelgangen hoor je natuurlijk al veel over goede arbeidsvoorwaarden, maar wanneer een echte software engineer dit ook aangeeft zorgt dat voor veel vertrouwen in de toekomst.

**Conclusie**

Ik heb een goed beeld van hoe mijn carrière er uit gaat zien. Ik heb een stage geregeld waar ik veel zin in heb om aan de slag te gaan en waar ik waarschijnlijk later ook de kansen ga krijgen om me door te ontwikkelen naar een nog mooiere functie.

## Scalable Architectures

Scalable Architectures gaat over het ontwikkelen van software dat om kan gaan met veel gebruikers en veel communicatie tussen verschillende componenten van de applicatie. Met behulp van microservices kan er veel communicatie afgehandeld worden en kan de software stabieler en beter worden gemaakt.

**Non-functional requirements**

In het begin van het project heb ik een lijst opgesteld met functional en non-functional requirements. Aan de hand van deze requirements heb ik de architectuurontwerpen gemaakt voor de applicatie. De non-functional requirements voor mijn applicatie zijn als volgt:

|  |  |
| --- | --- |
| **Categorie** | **Omschrijving** |
| Prestaties | Het ophalen van data duurt maximaal 1 seconde. |
| Prestaties | Het laden van een pagina duurt maximaal 1 seconde. |
| Prestaties | De website moet minimaal 500 gelijktijdige gebruikers aankunnen zonder impact te hebben op de prestaties. |
| Prestaties | Een tweet die geplaatst wordt door een andere gebruiker moet binnen 30 seconden zichtbaar zijn voor zijn volgers |
| Veiligheid | Gebruikersgegevens worden veilig opgeslagen & wachtwoorden worden versleuteld. |
| Veiligheid | Gegevens worden verstuurd over een beveiligde (HTTPS) verbinding. |
| Veiligheid | De persistentie van het systeem moet voldoen aan de AVG / GDPR. |
| Schaalbaarheid | De website moet mee kunnen schalen op basis van desktop en mobiele schermformaten. |
| Bruikbaarheid | Een nieuwe klant moet binnen 5 minuten het proces van een account creëren en een tweet plaatsen snappen |
| Onderhoudbaarheid | De broncode is geschreven volgens de SOLID-principes. |
| Onderhoudbaarheid | De systeemprestaties en uptime moeten automatisch gemonitord kunnen worden via een monitoringssysteem. |
| Beschikbaarheid | De applicatie moet een uptime hebben van minimaal 99.9% |
| Beschikbaarheid | Wanneer een van de functionaliteiten uitvalt moet de rest van de applicatie blijven werken |

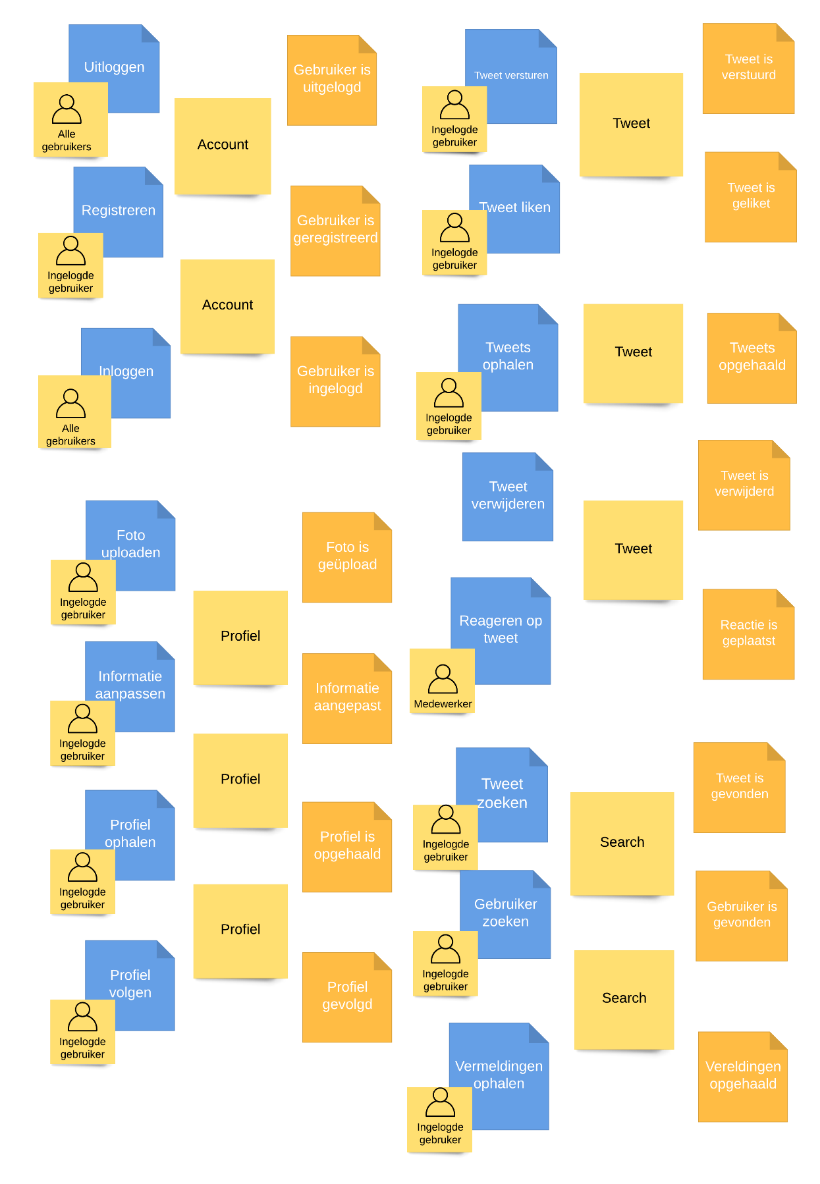
Naast de non-functional requirements zijn de functional requirements natuurlijk ook belangrijk. Deze zijn terug te vinden in bijlage 1.

Uit de non-functional requirements kunnen een aantal conclusies worden getrokken die de basis vormen van de architectuur van de applicatie. Voor het scalable architectures leerdoel is dat het volgende:

De applicatie zal worden opgebouwd via een microservice structuur. Deze structuur zorgt ervoor dat elke microservice verantwoordelijk is voor een bepaalde functionaliteit van de applicatie. Alle services moeten los van elkaar draaien en hebben een eigen dataopslag. Dit om aan de requirements van de categorie “beschikbaarheid” te voldoen.

**Fase 1**

Uit de non-functionals is gebleken dat de applicatie zal bestaan uit verschillende micro services. Om te bepalen welke microservices er komen en welke functionaliteiten elke microservice heeft heb ik event storming toegepast.



Uit event storming is gebleken dat ik in eerste instantie vier verschillende microservices zal hebben. De accountservice is verantwoordelijk voor het registreren en inloggen van gebruikers. De profielservice zorgt ervoor dat profielen geladen en aangepast worden. Ook zorgt deze service voor het volgen van gebruikers. De tweetservice heeft als nut dat tweets geplaatst, geladen en geliket kunnen worden. De searchservice draagt zorg voor het zoeken van tweets, gebruikers en vermeldingen.

**Fase 2**

Om te zorgen dat elke microservice toegang heeft tot een dataopslag heb ik een onderzoek uitgevoerd waarin ik heb gekeken welke oplossingen er zijn. De opslagmethode moet voldoen aan een aantal voorwaarden:

* Online bereikbaar zijn (i.v.m. deployment van applicatie)
* Uitbreidbaarheid van resources (variabele dataopslag, variabele bandbreedte)
* Makkelijke connectiviteit met Java

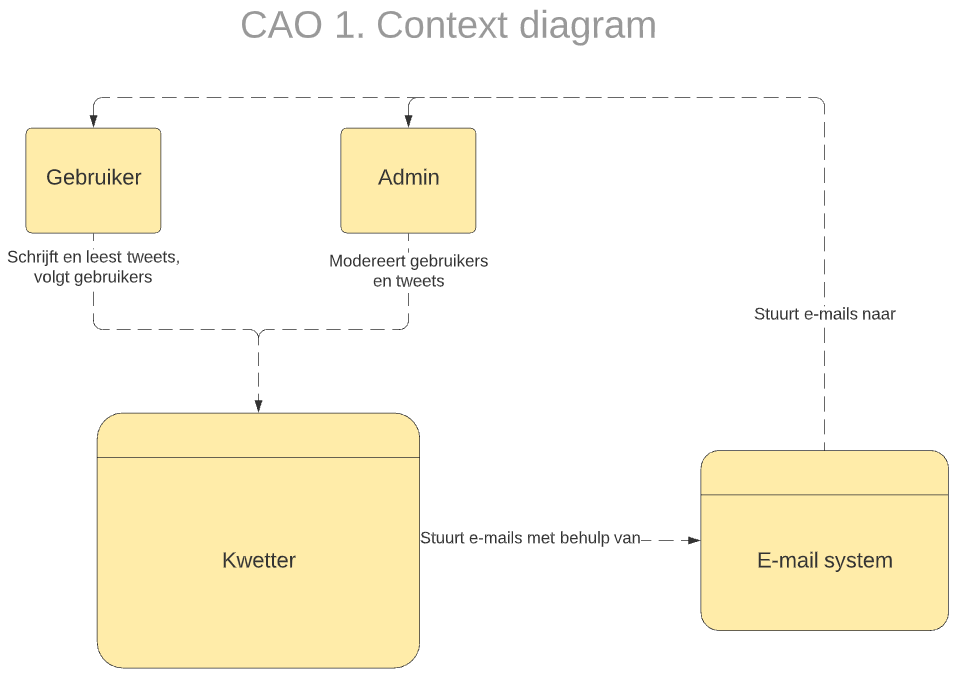
Mijn onderzoek bracht me al snel bij services als AWS, Azure en Google Cloud.

Ik heb de keuze gemaakt om Azure te gebruiken aangezien hier vanuit school al een account voor is opgesteld en ik hier ook al mijn ci/cd pipelines op heb staan. Meer informatie over hoe ik gegevens opsla in de database is te vinden onder het kopje “Security by Design”.

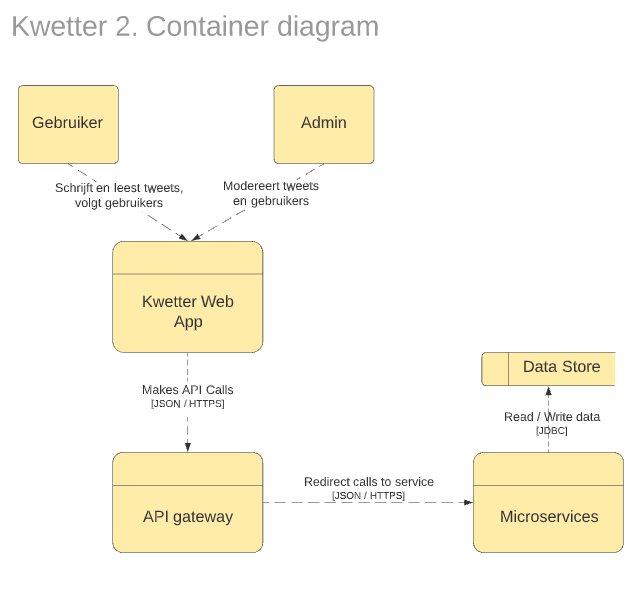
**Fase 3**

Ik ben de afgelopen fase bezig geweest met het opstellen van non-functional requirements en ik heb deze besproken tijdens de feedbackrondes. Ook heb ik voor elke functional requirement een definition of done opgesteld die bestaat uit meerdere punten. Uit mijn gesprek van een aantal weken geleden is gekomen dat ik dit leerdoel goed aan kan tonen door een extra microservice te maken die zonder code aan te passen van de andere microservices volledig meedraait met de rest van het systeem. Dit heb ik gedaan door een notificatieservice te maken die communiceert met een aantal andere microservices. Deze service heb ik binnen een dag opgezet. Hiermee heb ik laten zien dat de architectuur van mijn applicatie goed schaalbaar is.

Ook heb ik deze fase gewerkt aan de c1, c2 en c3 modellen.



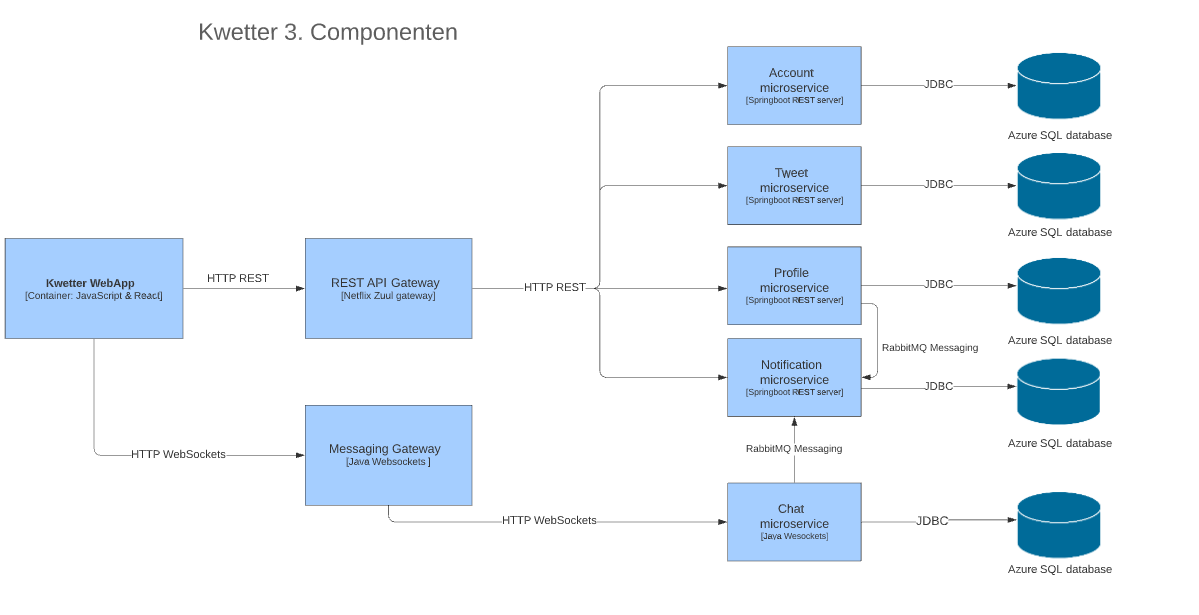
In het C1 model geef ik weer welke gebruikers er verwacht kunnen worden in de applicatie. In dit diagram wordt rekening gehouden met twee verschillende gebruikers: gebruikers en admins. De gebruikers kunnen een account aanmaken, profielen bekijken en tweets plaatsen en lezen. De admins zijn verantwoordelijk voor het modereren van gebruikers en van tweets. In de database zijn deze twee rollen ook gespecificeerd. Vanwege tijdsdruk heb ik helaas niet de mogelijkheid gekregen om de adminrol ook te implementeren in de applicatie zelf. Wel is deze applicatie makkelijk uit te breiden (non functional requirement onderhoudbaarheid) zodat deze rol geïmplementeerd kan worden.



Het C2 model geeft de verschillende containers weer. De web applicatie maakt API calls naar een gateway. Deze gateway is tussen de frontend en backend geplaatst om te zorgen dat alle calls uitkomen bij de goede microservice. Hierdoor hoeft er bij de frontend niet gespecifieerd te worden welke microservice de call afhandelt. De gateway is namelijk verantwoordelijk voor het distribueren van de calls naar de desbetreffende services. De gateway doet ook al een eerste stukje beveiliging van de backend, hiervoor meer onder het kopje “security by design”.

De gateway stuurt de calls dus door naar de verschillende microservices. Alle verzoeken worden verstuurd over http en worden geserialiseerd naar JSON. Ik heb hiervoor gekozen omdat het een industriele standaard is en daardoor zowel de frontend als de backend makkelijk de JSON weer om kunnen zetten in data en objecten.

De microservices slaan data op in de data store met behulp van de JDBC driver, hierover meer bij het volgende diagram.

Het C3 model geeft een beter beeld van de verschillende services en de technieken en protocollen. De bovenste 4 microservices zijn op dit moment geïmplementeerd in de applicatie. Alle microservices werken met Springboot. Deze keuze heb ik gemaakt omdat we in de proftaak al veel gebruik maken van Quarkus en Spark en ik iets nieuws wilde leren. Verder zou Springboot heel handig zijn met het deployen en implementeren van extra functionaliteiten als messaging.

Ik heb messaging toegepast zodat wanneer er meer drukte is er een wachtrij ontstaat voor de taken die notification service moet doen. Door messaging toe te passen kunnen requests tussen microservices asynchroon worden uitgevoerd. Hierdoor zal de applicatie gewoon blijven werken wanneer een microservice eruit ligt, dit is heel belangrijk in de microservice structuur. Voor messaging maak ik gebruik van RabbitMQ. Ook deze keuze heb ik gemaakt na een kleinschalig onderzoek. We hebben over RabbitMQ een workshop gekregen in het begin van het semester en toen ik online rond ging kijken voor alternatieven kwam ik allen Kafka tegen als alternatief met veel documentatie en gebruikers. Mijn beetje voorkennis met RabbitMQ heeft de doorslag gegeven om toch voor RabbitMQ te gaan.

Voor de databases gebruik ik dus Azure SQL databases. In fase 2 heb ik toegelicht waarom ik voor deze databases heb gekozen.

**Fase 4**

In de laatste fase heb ik na overleg met Hans besloten om mijn applicatie online beschikbaar te maken via een publieke url (<http://kwetterreact.azurewebsites.net/>). Hiervoor heb ik gebruikt gemaakt van een Kubernetes cluster welke deployed is binnen de Azure omgeving. Onder het kopje cloud services geef ik aan hoe ik dit heb gedaan en welke problemen ik heb moeten oplossen om de deployment werkend te krijgen.

**Feedback over dit leerdoel Hans en Kiavash**





## Development and Operations (DevOps)

Dit leerdoel beslaat het proces van het continue uitvoeren van unit testen en het continue deployen van de applicatie. Met de hulp van een of meerdere pipelines kan er gezorgd worden voor een stabiele applicatie met relevante goed dekkende testen.

**Fase 1**

Ik heb al vaker gewerkt met Jenkins en Travis-CI. Hierdoor heb ik een goed beeld van hoe ik een pipeline op kan zetten voor een enkele applicatie. Ik moet wel nog meer leren over hoe dit in zijn werk gaat bij microservices.

**Fase 2**

Om te zorgen dat ik alle services gemakkelijk vanaf een server kan draaien ga ik gebruik maken van Docker. Docker zorgt ervoor dat meerdere applicatie gemakkelijk uitgevoerd kunnen worden vanaf een pc of server. Met behulp van Dockerhub kan ik zorgen dat ik elke keer de nieuwste versie van een service in Docker krijg. Ik heb mijn Dockerhub namelijk zo ingesteld dat ik bij iedere push naar een bepaalde branch automatisch een nieuwe Docker Image binnen krijg. Deze kan ik dan gemakkelijk uitvoeren.

Elke service heeft een eigen repository binnen Dockerhub. Dit om elke service zo min mogelijk afhankelijk te maken van de andere services. De verschillende repositories staan hieronder gelinkt.

<https://hub.docker.com/repository/docker/kevints98/profileservice>

<https://hub.docker.com/repository/docker/kevints98/tweetservice>

<https://hub.docker.com/repository/docker/kevints98/accountservice>

<https://hub.docker.com/repository/docker/kevints98/notificationservice>

De volledige Git repository met alle code is terug te vinden op:

<https://github.com/jeBoyKevin/kwetter>

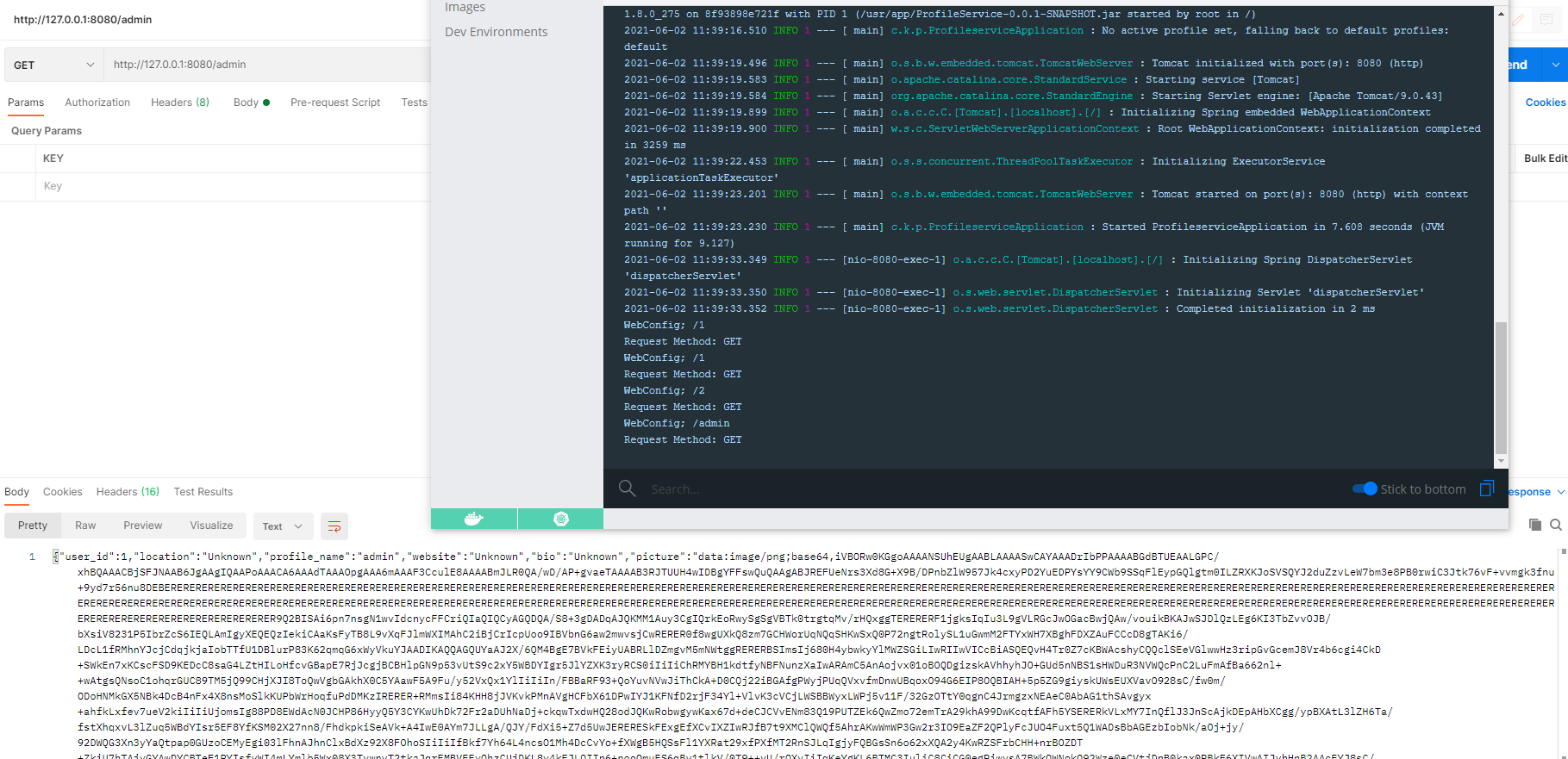
**Fase 3**

Verder heb ik deze fase de user stories beter uitgewerkt met zogenaamde definition of dones. Aan de hand van deze definition of dones kan ik beter testen schrijven om te kijken of een functionaliteit goed geïmplementeerd is. Elk item bij de definition of done kan ofwel als unit test of als integratie test geschreven worden. Wanneer ik elk item van de definition of done heb getest weet ik zeker dat een user story goede werkt.

Nu de non-functionals bekend zijn is het belangrijk dat ik mijn testen aan laat sluiten om te kijken of ik deze requirements ook haal. Om deze requirements aan te tonen zal ik vooral veel systeemtesten en unit testen schrijven. Ook kan ik de functional requirements inclusief *definition of done* meenemen voor het schrijven voor unit testen en integratietesten. Deze testen worden (te laat) in fase 4 geschreven, evenals een testrapport.

Om te zorgen dat de containers die Docker runt mee kunnen schalen met het aantal gebruikers heb ik ook Kubernetes geconfigureerd. Kubernetes kan ervoor zorgen dat een container meer geheugen en CPU-cores toegewezen krijgt wanneer dit nodig is, bijvoorbeeld bij veel gelijktijdige gebruikers. Wanneer deze resources niet nodig zijn, worden ze ook niet gebruikt. Dit is belangrijk voor de non-functional requirements van de categorie prestaties.

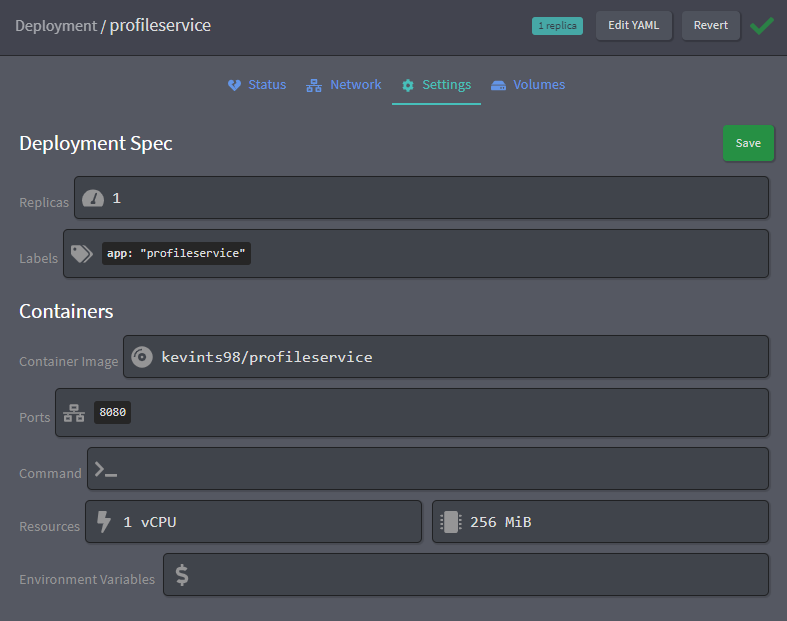
Om Kubernetes voor mezelf makkelijker te begrijpen te maken heb ik Kubesail gebruikt. Dit is een GUI die gratis te gebruiken is en het een stuk makkelijker maakt om Kubernetes op te zetten voor mijn Docker containers. Hans was te spreken over mijn ontdekking van deze software.

Hieronder een voorbeeld van een werkende microservice in Docker die aangeroepen wordt middels Postman en de verwachte response geeft.

Hier een overzicht van alle vier microservices hoe ze in Docker staan.



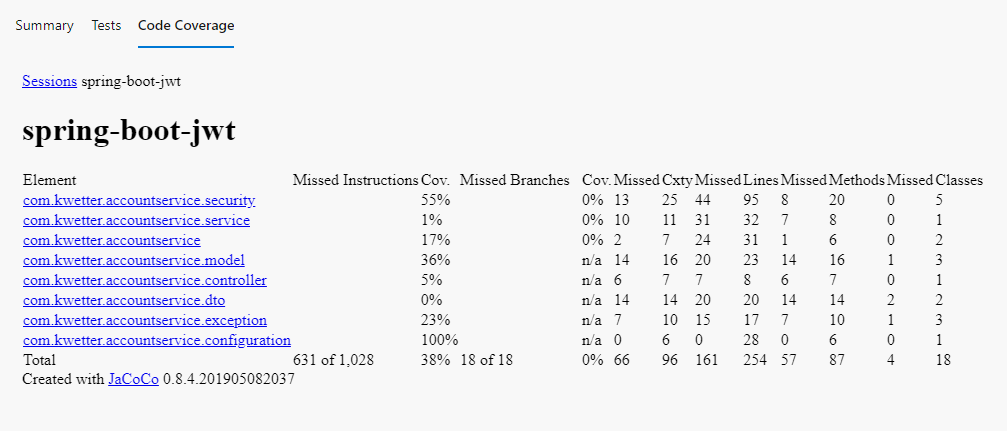
Ook een voorbeeld van een van de services in combinatie met de GUI van Kubesail die het gemakkelijk maakt Kubernetes in te stellen voor Docker containers.



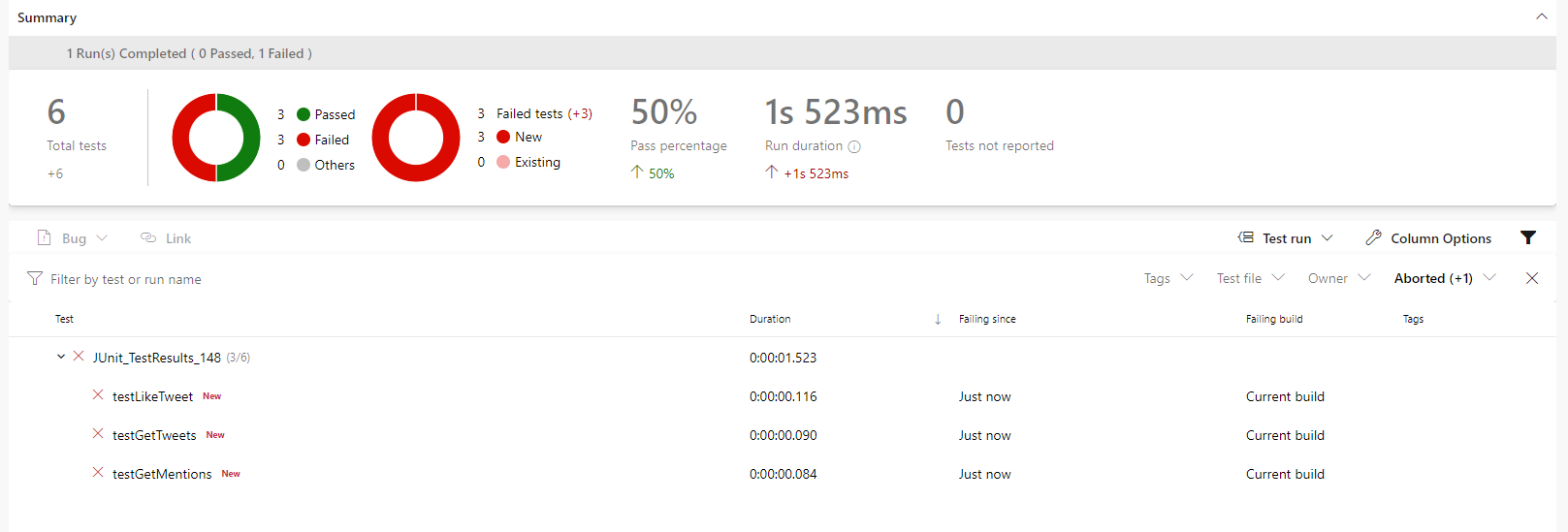
Ik begrijp dat Docker en Kubernetes te maken hebben met zowel DevOps als Cloud services. Ik heb gekozen hier de uitleg over deze twee technieken te geven, onder het kopje cloud services staan mijn databases en messaging beter uitgelegd.

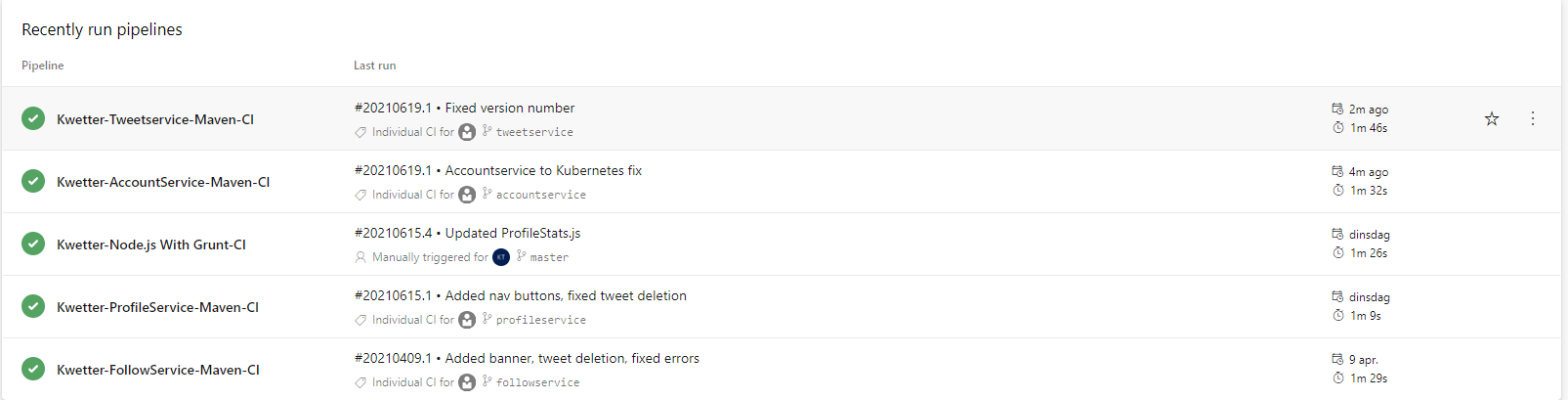
Ook heb ik deze fase gewerkt aan het inzichtelijk maken van de code coverage van mijn project. Hoewel er op dit moment nog te weinig testen zijn geven mijn CI/CD pipelines wel alvast een rapport bij elke run met het aantal geslaagde en gefaalde tests met ook een percentage hoeveel van mijn methodes daadwerkelijk bereikt worden met de huidige testen.

Dit rapport wordt bij elke push naar mijn Git repository automatisch gecreëerd in mijn pipeline.



Een voorbeeld van een geslaagde job met testrapport:



Een overzicht van alle pipelines:

De volledige CI/CD omgeving is terug te vinden met de volgende link

<https://dev.azure.com/382763/Kwetter>

**Fase 4**

In de laatste fase heb ik geen tijd meer gehad om nog veel aandacht te besteden aan de DevOps omgeving van mijn project. Alle pipelines voeren automatisch de testen uit in de Azure omgeving en Dockerhub zorgt ervoor dat aan de hand van de Dockerfiles de nieuwe Docker images gemaakt worden. Deze images worden daarna door binnengehaald in de Azure container registry en middels een Kubectl command worden de nieuwste images dan online gehost op hun desbetreffende URL’s. Meer over dit proces is te lezen onder het kopje Cloud services.

**Testen**

Helaas ben ik er niet meer aan toe gekomen om meer testen te schrijven voor mijn verschillende services. Hoewel de volledige DevOps omgeving staat en klaar is om de testen automatisch uit te voeren ben ik door de tijdnood me gaan richten op andere belangrijke onderdelen binnen dit project. Wel heb ik in de laatste fase performance testen uitgevoerd met behulp van de K6-tool. Meer hierover is terug te vinden in bijlage 4: Performance testen. Ook ga ik in bijlage 5 nog een keer over mijn non-functional requirements heen. Ik doe dit omdat ik vanwege tijdsnood geen tijd heb gehad om de testen te schrijven die normaal de non-functionals aantonen.

**Conclusie**

Al met al denk ik dat ik dit leerdoel heb aangetoond met een proficient. Ik heb volledig werkende pipelines voor zowel de frontend als de backend. Ook worden de nieuwste versies van de code automatisch omgezet naar Docker Images en zijn deze online te bereiken zodat Azure hier werkte Kubernetes pods van kan maken die draaien in mijn Azure Kubernetes cluster. Ik vind het jammer dat ik te weinig unit- en integratietesten heb geschreven, maar ik heb voor de tweetservice wel verschillende testen geschreven om aan te tonen dat ik weet hoe dit moet en als test voor mijn DevOps.

**Feedback Hans en Kiavash over dit leerdoel**







## Cloud services

Dit leerdoel gaat over de verschillende cloud-services die beschikbaar zijn voor databases, authentication, logging, monitoring, virtuele machines en Docker. Er moet onderzoek gedaan worden naar de verschillende middelen ene kosten die nodig zijn voor de applicatie.

**Fase 1**

Ik heb nog geen ervaring met het gebruik van Docker of het onderzoek doen naar cloud-services die nodig zijn voor het goed laten werken van een applicatie. Ik zal hier in de loop van het semester veel over moeten leren.

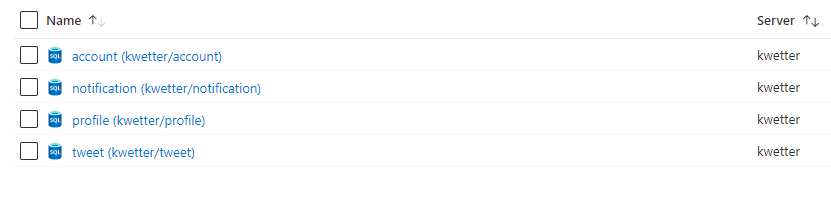
**Fase 2**

Ik heb me de afgelopen weken bezig gehouden met mijn projecten te laten werken binnen de Azure omgeving. In het vorige stukje heb ik al uitgelegd hoe ik dat heb gedaan met de CI/CD pipelines, maar ik heb ook al de databases draaien in de Azure omgeving. Ik moet wel nog kijken naar messaging frameworks (Rabbitmq) en naar Docker.

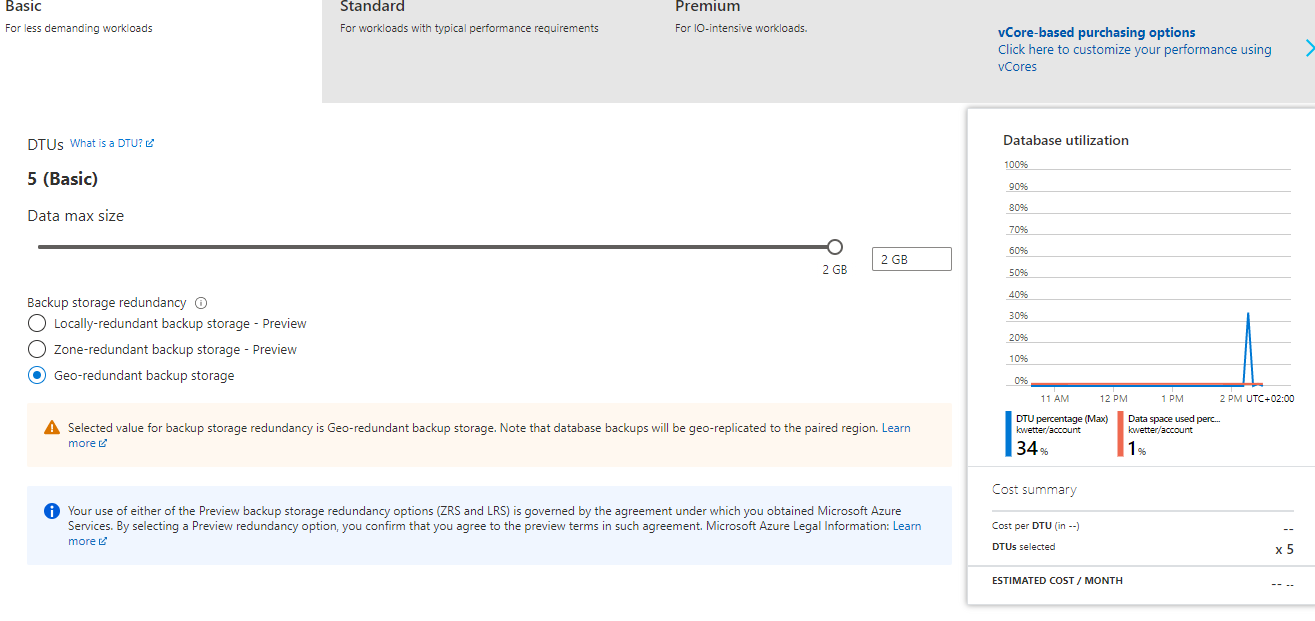
**Fase 3**

In het vorige leerdoel heb ik al laten zien dat ik mijn applicatie uitrol via Kubernetes, maar ik heb ook andere dingen gedaan om te zorgen dat ik me meer verdiep in cloud services.

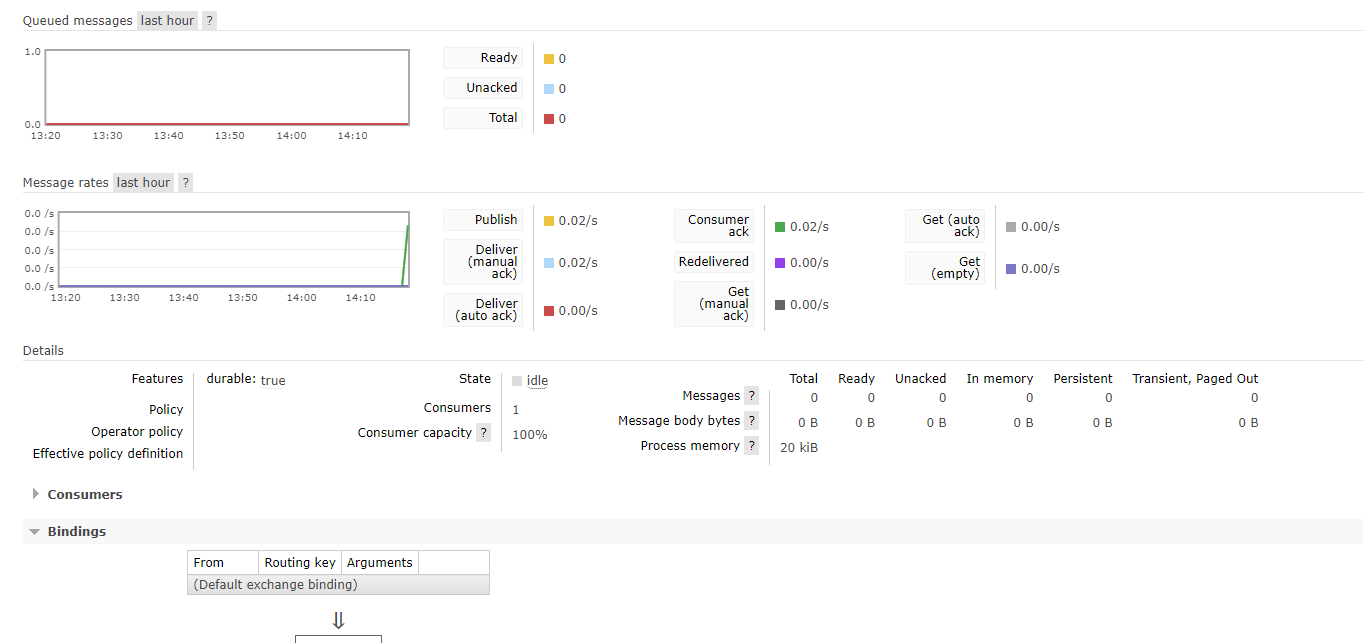
Vorige sprint ben ik bezig geweest met het implementeren van de Azure databases. Hierdoor kan ik de bandbreedte en de grote van de databases eenvoudig opschroeven.

Een overzicht van alle databases die ik in Azure heb staan:

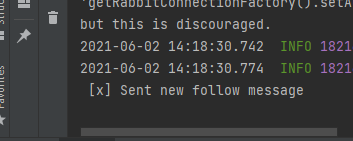
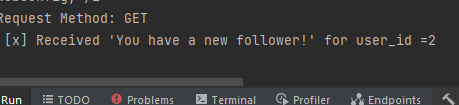
Zoals ik in het leerdoel van scalable architectures heb uitgelegd is het heel makkelijk om de resources van een database uit te breiden. Hieronder het scherm wat dat binnen Azure mogelijk maakt.



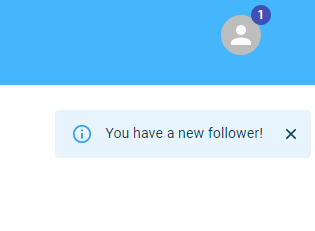
Ik heb in mijn gesprek met Hans mijn onderzoek wat ik heb gedaan voor messaging besproken met Hans en ook mijn implementatie hiervoor uitgelegd en verantwoord. Ik heb er rekening mee gehouden dat ik een proces kies wat een niet-vitale functie heeft voor het doorlopen van het product en dat er goed gebruik kan worden gemaakt van de cue die een messaging oplossing geeft. Op basis van deze voorwaarden heb ik gekozen mijn notificatieservice een messaging functionaliteit te geven van RabbitMQ.



Dit is het dashboard van RabbitMQ met een overzicht van het aantal berichten dat is gestuurd het afgelopen uur. Hier is te zien dat er een bericht is verstuurd en dat deze ook direct aangekomen is, anders zou de grafiek in de queue omhoog zijn gegaan.



De linker afbeelding geeft de profileservice weer die aangeeft een bericht verstuurd te hebben. Aan de rechterkant een melding van de notificatieservice dat er een bericht is binnengekomen.

Wanneer de gebruiker nu inlogt krijgt hij de notificatie binnen dat hij een nieuwe volger heeft.

**Fase 4**

Deze laatste fase heb ik me volledig gericht met het deployen van mijn applicatie naar een publieke URL. Dit heb ik gedaan door middel van een Azure Kubernetes cluster.

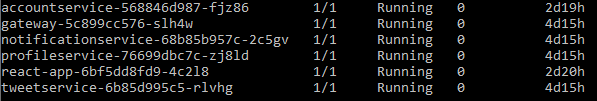
Onder het kopje DevOps heb ik al laten zien dat al mijn images automatisch naar Dockerhub gepusht worden. Doordat al mijn services daar staan kan ik deze gemakkelijk binnenhalen in de Azure Container Registry. Na het koppelen van je lokale powershell of cmd aan je Azure subscription kan elke service gepusht worden naar Azure’s registry door middel van de volledige command:

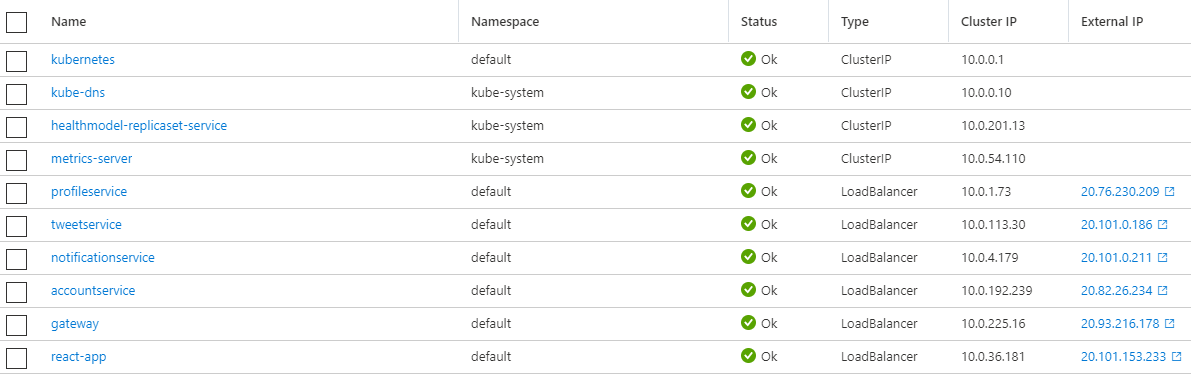
docker push <acrLoginServer>/<Docker image>:latest

Hierna heb ik een yaml geschreven voor elke service (die redelijk hetzelfde is) die dient als input voor Kubernetes. Een voorbeeld van de yaml is als volgt:

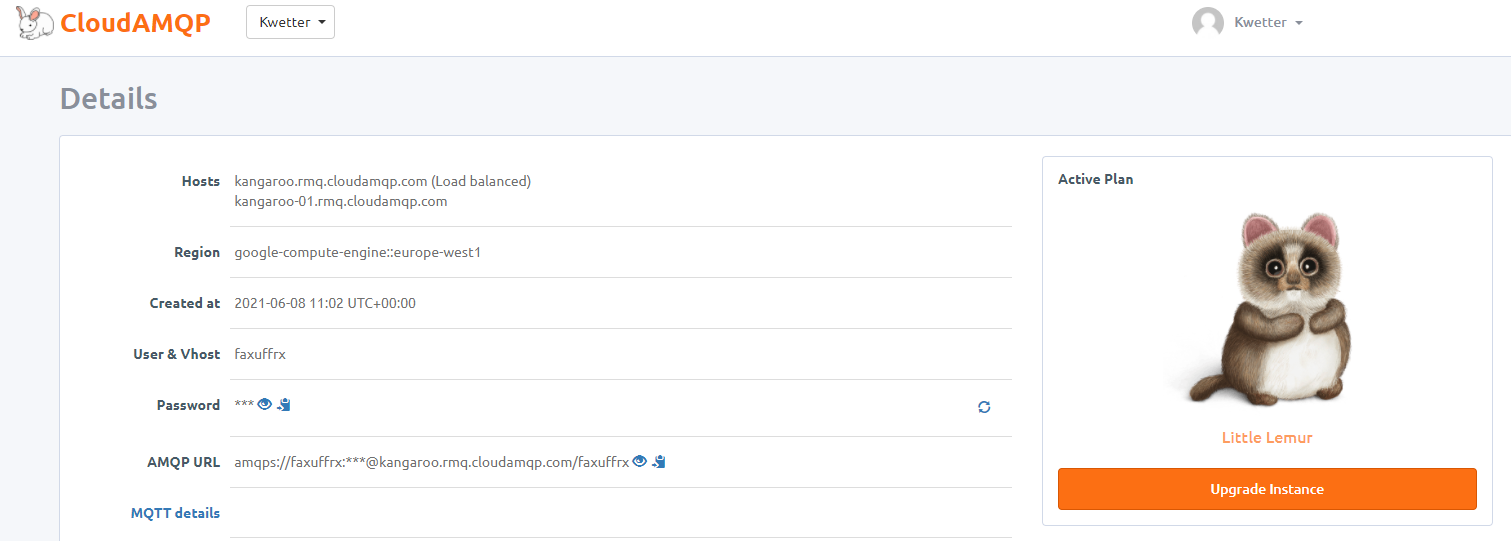
Met deze yaml file maakt Kubernetes een pod aan in mijn Kubernetes cluster. Elke microservice heeft een deploment waarvoor hij de image uit mijn Azure Container registry haalt en waarin ik aangeef hoeveel resources hij op mag nemen. Ook heeft elke microservice een loadbalancer.

Door deze yaml file toe te voegen aan een microservice met een simpele kubectl apply command wordt de er automatisch een Kubernetes pod gemaakt. Wanneer dit voor elke service gedaan is kan ik een overzicht opvragen van alle pods die ik heb draaien in mijn cluster.

Kubectl get pods

****In Azure (of via een command) kan ik dan de URL’s zien voor elke pod.

De external IP’s zijn door iedereen te bereiken via Postman. Mocht u het willen proberen, let wel op dat Azure de pods na een tijd van inactiviteit in een soort slaapstand zet waarna ze ongeveer een minuut nodig hebben om wakker te worden. Dit gebeurt automatisch na een request naar het IP-adres. Ook zal mijn Azure subscription rond 22-06 niet genoeg saldo meer hebben waardoor alle services ook niet meer bruikbaar worden.

Om te zorgen dat mijn messaging service ook via een publieke URL werkt heb ik gebruik gemaakt van CloudAMQP. Deze service host een RabbitMQ service online zodat deze vanuit overal te bereiken is. Deze service werkt perfect en biedt alle nodige functionaliteiten aan in het gratis plan.

De frontend staat op een gratis plan, de microservices niet. De frontend zal als het goed is gewoon door blijven werken, de services zullen niet meer te bereiken zijn. Dit zult u dus al snel merken wanneer u probeert de applicatie te gebruiken.

Tot die tijd is de volledige applicatie te gebruiken via de volgende URL:

<http://kwetterreact.azurewebsites.net/>

**Conclusie**

Ik vind dat ik dit leerdoel zeker goed heb aangetoond. Ik heb de applicatie volledig online werkend gekregen en dat bijna volautomatisch. Bij de demo die ik aan Hans en Kiavash heb gegeven kreeg ik de feedback dat ik heb aangetoond een goed werkende Enterprise applicatie heb opgeleverd. Dit leerdoel heeft daar zeker aan bijgedragen.

**Feedback Hans en Kiavash bij dit leerdoel**







## Security by Design

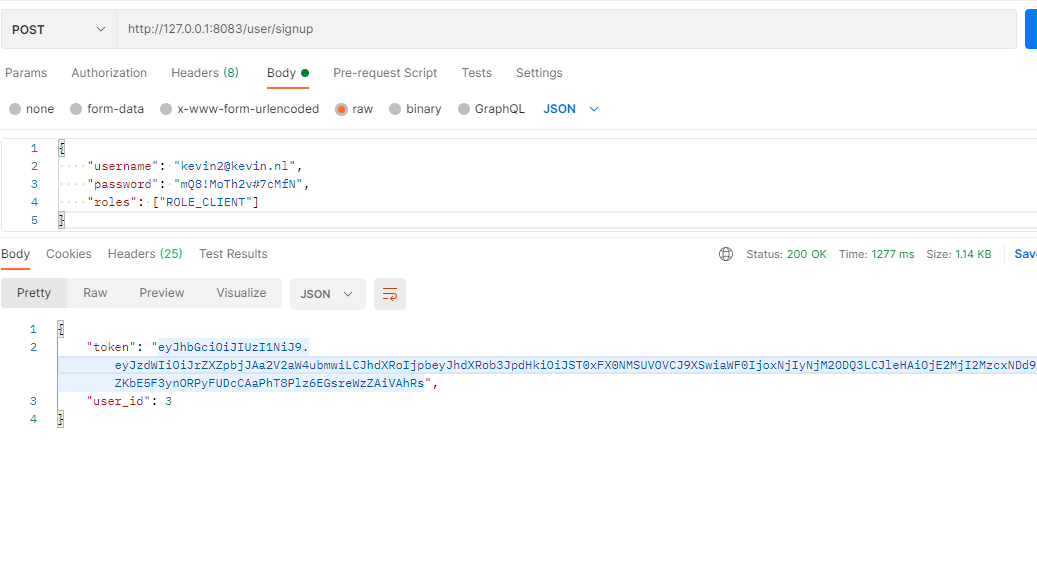
In dit leerdoel moet je aantonen dat je software kan ontwikkelen met het oog op beveiliging. Dit in tegenstelling tot het beveiligen van een applicatie wanneer deze al gebouwd is. Autorisatie en authenticatie zal toegepast moeten worden en er moet onderzoek gedaan worden naar de OWASP 10 risico’s en deze zullen verholpen moeten worden.

**Fase 1**

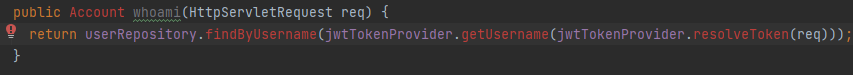
Ik heb al wel eens authenticatie toegepast op zowel de backend als frontend van mijn applicaties. Ook heb ik al eens onderzoek gedaan naar de OWASP 10 bij een React applicatie. Ik moet wel meer rekening gaan houden met de security van mijn applicatie bij de ontwikkeling hiervan, niet pas achteraf.

**Fase 2**

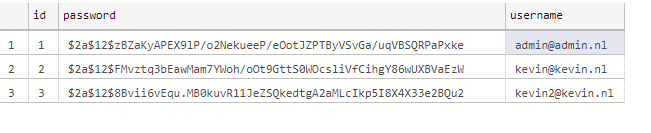
Om dit leerdoel aan te tonen moet ik vanaf het beginpunt bezig zijn met de beveiliging van mijn applicatie. Daarom heb ik nu al enkele technieken toegepast om de veiligheid te verbeteren. Zo maak ik gebruik van bearer tokens om de authenticatie van mijn project te regelen. Dit heb ik gedaan om ervoor te zorgen dat de bepaalde functionaliteiten alleen toegankelijk zijn voor gebruikers die hier het recht toe hebben. Zo mogen alleen ingelogde gebruikers volgen en tweets versturen. Een voorbeeld van een request wat verstuurd wordt naar de accountservice is als volgt:



Met de token die in de response staat kan de gebruiker zich dan verifiëren wanneer hij een actie uit wil voeren waarvoor je ingelogd dient te zijn. De backend heeft hier een *whoAmI* methode voor:



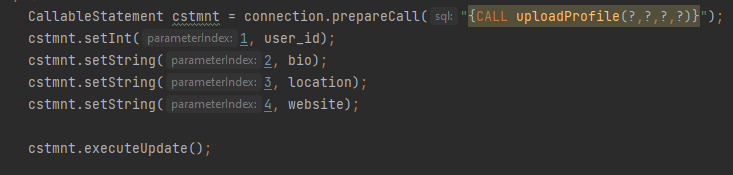
Als de username die bij de token hoort hetzelfde is als de username van de gebruiker die het verzoek verstuurde, is alles in orde en mag deze gebruiker de gekozen functionaliteit uitvoeren



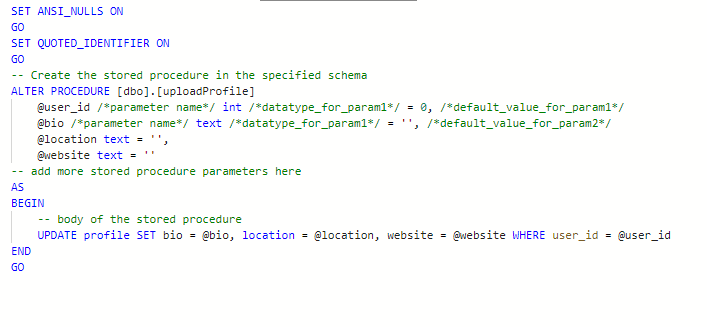
De non functional requirement van het veilig opslaan van wachtwoorden en persoonlijke gegevens is van groot belang binnen dit project. Daarom heb ik ervoor gekozen om alle wachtwoorden te hashen voor ze worden opgeslagen. Ook zorg ik ervoor dat ik geen gegevens opsla in de database die niet nodig zijn voor een goede werking van de applicatie. Meer hierover in het hoofdstuk Distributed data.

Drie voorbeeldaccounts die gehasht worden en met een salt worden opgeslagen in de database. Ik maak gebruik van de Password encoder die standaard in het Spring Boot framework zit. Deze password encoder maakt gebruik van Bcrypt hash. In bijlage 2 wordt uitgelegd uit welke onderdelen zo een hash is opgebouwd.

Door gebruik te maken Stored procedures in combinatie met geparametiseerde queries wordt het onmogelijk voor een hacker om gebruik te maken van SQL Injectie aanvallen om de applicatie binnen te dringen.



Dit is de code in de backend die ervoor zorgt dat een stored procedure aangeroepen kan worden. In tegenstelling tot variabelen direct opnemen in de query zorgen geparemetiseerde queries ervoor dat de gebruiker door bepaalde strings in te vullen niet commando’s uit kan voeren op de database.



Een stored procedure wordt opgeslagen op databaseniveau. Dit heeft niet alleen voordelen wat betreft security, de code hoeft namelijk niet eens meer leesrechten te krijgen tot de database, maar ook voor de uitbreidbaarheid van de code. In de code staan nu geen uitgebreide queries meer, maar wordt alleen het uitvoeren voorbereid en de response afgehandeld.

**Fase 3**

Uit mijn vorige gesprekken met Hans en Kiavash blijkt dat ik al een aantal goede stappen heb gezet op het gebied van security. Authenticatie en autorisatie is toegepast, de datalayer is beschermd door middel van stored procedures en geparametriseerde queries. Ook worden alle gegevens versleuteld opgeslagen in de database. Ze willen dat ik alleen nog aantoon dat ik de security van mijn applicatie al in mijn achterhoofd heb gehouden bij het opzetten van de architectuur van de applicatie. Daarom ben ik bezig met het opstellen van een security document waarin ik in alle lagen van het project ga doorlopen op welke manieren ik voorkom dat er beveiligingsproblemen zijn. Ook ga ik de OWASP top 10 door om te kijken of er bij het opzetten van de architectuur van het project misschien toch fouten zijn geweest die van invloed zijn op de veiligheid van de applicatie. Dit document zal ik in een volgend gesprek met Hans en Kiavash bespreken.

**Fase 4**

In deze fase ben ik met name bezig geweest met het schrijven van een security by design document. In dit document onderzoek ik welke risico’s het project kent op het gebied van bedreigingen. Ook ga ik naar welke gevaren er afkomstig zijn van verschillende groeperingen en welke informatie ze buit willen maken. In het misuse diagram ga ik na welke aanvallen er mogelijk zouden kunnen zijn, in dit diagram wordt ook aangegeven van wie deze bedreigingen afkomen en welke oplossingen er zijn voor deze gevaren. Als laatste loop ik de volledige OWASP top 10 door en kijk ik of de aanvallen betrekking hebben tot het Kwetter project en wat ik doe om de applicatie te beveiligen hiertegen. Het volledige security by design document is terug te vinden in bijlage 3.

**Conclusie**

Ik heb al een achtergrond in security aangezien ik beide delen van de Cyber Security specialisatie gelopen heb. Ik heb deze kennis proberen zoveel mogelijk mee te nemen bij het ontwikkelen van deze applicatie. Door verschillende voorbeelden te noemen van hoe ik mijn applicatie beveilig en door het schrijven van een security by design document vind ik dat ik genoeg heb gedaan om dit leerdoel aan te tonen op proficient.

## Distributed Data

Dit leerdoel betreft het onderzoek doen naar welke data nodig is om op te slaan en het veilig opslaan hiervan. Er moet gekeken worden naar de privacy van de gebruikers, alsmede ethische vraagstukken. Er mag niet onnodige data gevraagd worden en er moet voldaan worden aan de wetgeving (AVG, GDPR).

**Fase 1**

Ik moet onderzoek doen naar welke data precies benodigd is voor de eindgebruikers van mijn Kwetter applicatie. Ik ga kijken wat de Nederlandse wetgeving aangeeft als eisen en ik ga rekening houden met de ethische vraagstukken bij deze kwestie. Ik moet hiervoor nog veel onderzoek doen.

**Fase 2**

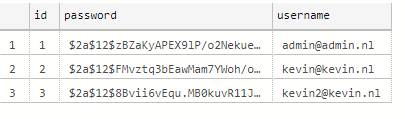
Bij het opzetten van mijn databases heb ik er rekening mee gehouden dat ik alleen data opsla die ik echt nodig heb om de software te laten draaien. Zo vraag ik bij het registeren alleen om een gebruikersnaam en wachtwoord en sla ik ook zo min mogelijk cookies op. Alle wachtwoorden die in de database worden gezet worden gesalt en daarna gehasht. Hierdoor is het zelfs wanneer de database in verkeerde handen komt onmogelijk voor de aanvaller om de inloggegevens van de gebruiker buit te maken.

**Fase 3**

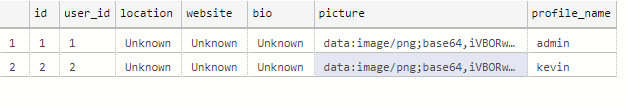
In het hoofdstuk security by design heb ik al aangetoond dat ik gegevens veilig opsla en dat ik ook de toegang tot de database voor iedereen ontzeg behalve ip-adressen die op het subnet mask zitten van mijn eigen netwerk. Ook heb ik onder het kopje cloud services al aangetoond dat mijn database schaalbaar is en de resources makkelijk zijn uit te breiden wanneer dit nodig is.

In dit hoofdstuk wil ik nog ingaan op de gegevens die ik opsla, waarom ik ze opsla en nog een stuk over waarom mijn database openstaat voor uitbreidingen van de applicatie.

Ik heb twee tabellen waar persoonlijke gegevens van gebruikers in zouden kunnen staan. In de *account* tabel gaat het om de volgende gegevens:



De wachtwoorden worden, zoals eerder al aangegeven veilig opgeslagen en zelfs bij een data lek is het knap lastig voor hackers om iets te doen met deze gegevens. Verder sla ik in deze tabel alleen de e-mailadressen op van gebruikers. Dit doe ik zodat ik de gebruikers een e-mail kan sturen met een vervangend wachtwoord, mochten ze deze vergeten zijn. Ook gebruik ik de e-mailadressen om de gebruikers in te lichten over eventuele beveiligingsinbreuken.



Bovenstaande tabel zijn een aantal records uit de *profiel* tabel. Op een profiel kan een gebruiker zelf gegevens over zichzelf plaatsen, zoals zijn woonplaats, favoriete website en een bio. Deze gegevens staan standaard op “Unknown” en de gebruiker wordt ook nooit verplicht deze gegevens in te vullen. Verder kiest de gebruiker bij het aanmaken van een profiel ook een profielnaam. Deze profielnaam is het enige wat zichtbaar is naar andere gebruikers toe en kan dus gewoon een bijnaam of iets willekeurigs zijn in plaats van een echte naam. De gebruiker heeft dus altijd in eigen hand hoeveel hij of zij weg wilt geven over zichzelf. Het is alleen mogelijk om een account aan een profiel te koppelen wanneer deze inzicht heeft in beide tabellen van de database. Iets wat zeer moeilijk wordt gemaakt door de manier waarop ik de database beveiligd heb (zie security by design).

Het is belangrijk dat ik niet data opsla die ik niet nodig heb en dat ik ook geen data onveilig op sla. Dit doe ik om te voldoen aan de wetgeving, maar ook om de privacy van de gebruikers te waarborgen.

**Fase 4**

Ik ben van mening dat ik alles wat ik heb moeten doen in de vorige fases al heb aangetoond. Zo heb ik mijn keuze voor de databases al uitgelegd onder het kopje Scalable Architectures en heb ik onder het kopje Cloud Services al laten zien dat de database schaalbaar zijn. In de vorige fase van dit hoofdstuk heb ik uitgelegd waarom ik bepaalde gegevens op sla en wat ik doe om de privacy te waarborgen.

**Conclusie**

Door te laten zien dat mijn databases schaalbaar zijn en waarom ik voor de Azure SQL databases gekozen heb, heb ik laten zien dat ik dit leerdoel beheers. Ook sla ik alleen de belangrijke gegevens op en ga ik veilig met de gegevens om. Hierdoor vind ik dat ik dit leerdoel proficient heb aangetoond.

# Terugblik

Ik heb dit ervaren als het meest uitdagende semester van de hele opleiding. Gedeeltelijk door te stof waarbij er veel van mij als student gevraagd wordt, maar ook een gedeelte door de lastige situatie nu in de coronatijd. Vanuit school waren veel dingen nogal onduidelijk en zijn er genoeg opdrachten geweest die tijdrovend waren in een al volgepakt semester. Enkele voorbeelden hiervan zijn de case studies, de sprint opleveringsdocumenten en de dubbele oplevering aan het einde van elke sprint.

Persoonlijk gezien ben ik aan het begin van het semester heel gemotiveerd geweest om te zorgen dat ik dit semester tot een succes zou brengen. Ik ben snel aan de slag gegaan met de functionele vereisten. Nu ik terugkijk had ik beter meer tijd kunnen steken in de leerdoelen en dit PDR. Hoewel ik van mening ben dat ik uiteindelijk elk leerdoel heb aangetoond en ik ook de bevestiging en feedback heb gekregen van beide docenten dat dit zo is had het me veel stress bespaard als ik vanaf het begin mijn tijd beter had ingedeeld.

Al met al ben ik tevreden met mijn eindproduct en de opgedane kennis. Ik had een aantal dingen zeker nog wat groter aan willen pakken, maar met één dag in de week werken aan je individuele project is dat simpelweg onmogelijk.

# Conclusie

Dit semester heb ik veel dingen geleerd over het ontwikkelen van een applicatie in een Enterprise context. Het schrijven van code zelf is was dit semester minder belangrijk dan alle zaken er omheen. Het opzetten van pipelines, deployen van applicaties, onderzoek doen naar verschillende platformen en services en het onderhouden van databases hebben mij veel nieuwe dingen geleerd. Ook komen oude dingen zoals architectuur en security weer terug en ik heb gevoel dat ik deze dingen goed onder de knie heb en heb laten zien in dit document en in de verschillende feedbackgesprekken.

Daarom heb ik het gevoel dat ik dit semester heb gehaald.

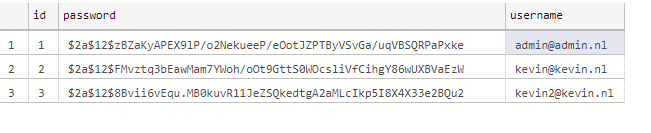
# Bijlagen

## Bijlage 1: Functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Functionaliteit** | **Gebruikersrol** | **User Story ID** | **Omschrijving** |
| Tweet sturen | Gebruiker | 1 | Gebruiker kan een tweet sturen |
| **Definition of done:**   * Ingelogde gebruiker krijgt een invoerveld te zien * Invoerveld accepteert alleen tekst en andere unicode karakters * Er wordt gebruik gemaakt van input filtering (indien nodig, onderzoek) * Er wordt een melding getoond wanneer de tweet verstuurd is of als er iets fout gaat * De tweet wordt in de database opgeslagen | | | |
| Tweet verwijderen | Gebruiker | 2 | Gebruiker kan een tweet verwijderen |
| **Definition of done:**   * Ingelogde gebruiker krijgt een “tweet verwijderen” knop te zien * Alleen de eigenaar van een tweet kan de tweet verwijderen * De gebruiker krijgt een melding dat de tweet verwijderd is * De tweet wordt volledig uit de database verwijderd (niet middels een property niet weergegeven) | | | |
| Tweet liken | Gebruiker | 3 | Gebruiker kan een tweet liken |
| **Definition of done:**   * Ingelogde gebruiker krijgt een “tweet liken” knop te zien (kan ook een duim zijn of iets dergelijks) * De eigenaar van een tweet kan niet zijn eigen tweet liken * De liker van de tweet krijgt visuele confirmatie dat de tweet geliket is * Een tweet kan nooit meerdere malen geliket worden door een gebruiker * De eigenaar van de tweet krijgt een notificatie dat de tweet een like heeft * De combinatie tweet-liker van de tweet wordt opgeslagen in de database * Het aantal likes van de tweet wordt in de database met 1 verhoogd | | | |
| Gebruiker volgen | Gebruiker | 4 | Gebruiker kan een andere gebruiker volgen |
| **Definition of done:**   * Ingelogde gebruiker krijgt een “profiel volgen” knop te zien op een profielpagina * Een gebruiker kan nooit zijn eigen profiel volgen * De gebruiker krijgt een melding dat hij nu het andere profiel volgt * De gebruiker die hoort bij het profiel wat gevolgd wat krijgt een notificatie dat hij een nieuwe volger heeft * De volgers en volgende worden op beide profielen getoond * De combinatie volgende-volger wordt opgeslagen in de database | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Gebruiker ontvolgen | Gebruiker | | 5 | | Gebruiker kan een andere gebruiker ontvolgen |
| **Definition of done:**   * Ingelogde gebruiker krijgt een “profiel niet meer volgen” knop te zien als hij een profiel bezoekt dat hij al volgt * Deze knop zal nooit zichtbaar zijn wanneer een gebruiker zijn eigen profiel bekijkt * De gebruiker krijgt een melding dat hij het profiel niet meer volgt * De combinatie volgende-volger wordt verwijderd uit de database * Het is altijd mogelijk om toch weer opnieuw een gebruiker te volgen | | | | | |
| Tijdlijn inzien | Gebruiker | | 6 | | Gebruiker kan alle tweets zien van gebruikers die hij volgt |
| **Definition of done:**   * De gebruiker krijgt op de hoofdpagina een overzicht met alle tweets * De lijst bestaat uit de laatste x (100?) tweets * De lijst is chronologisch gesorteerd * De lijst bestaat alleen uit tweets van mensen die gevolgd worden door de gebruiker * De lijst kan altijd uitgebreid worden met oudere tweets door naar beneden te scrollen | | | | | |
| Zoeken door tweets | Gebruiker | | 7 | | Gebruiker kan een specifieke tweet terugzoeken |
| **Definition of done:**   * De gebruiker krijgt een invoerveld te zien * Invoerveld accepteert alleen tekst en andere unicode karakters * Er wordt gebruik gemaakt van input filtering (indien nodig, onderzoek) * De gebruiker krijgt een overzicht van alle tweets die de zoekterm bevatten (max 100) * Tweets van mensen de gebruiker volgt staan bovenaan, verder wordt er chronologisch gesorteerd | | | | | |
| Mentions inzien | Gebruiker | | 8 | | Gebruiker krijgt een overzicht van al zijn mentions |
| **Definition of done:**   * De gebruiker ziet de knop “alle mentions inzien” * De gebruiker ziet de laatste 100 keer dat hij genoemd is in een tweet * De lijst van alle tweets wordt chronologisch gesorteerd | | | | | |
| Reageren op tweet | Gebruiker | | 9 | | Gebruiker kan reageren op een tweet |
| **Definition of done:**   * De gebruiker ziet een “reageren op tweet” knop (Kan ook een pijltje terug zijn of iets dergelijks) * De tweet krijgt automatisch de gebruikersnaam van de eigenaar van de oorspronkelijke tweet * De tweet wordt verder afgehandeld als een normale tweet * De eigenaar van de oorspronkelijke tweet krijgt een melding dat hij een reactie heeft ontvangen | | | | | |
| Inloggen | Gebruiker | | 10 | | Gebruiker kan inloggen |
| **Definition of done:**   * De gebruiker kan inloggen met zijn geregistreerde account d.m.v. email en wachtwoord. * De gebruiker krijgt een melding te zien als er verkeerde credentials ingevoerd zijn. * Bij succesvol inloggen wordt de gebruiker doorverwezen naar de homepagina * De ingelogde gebruiker krijgt een (bearer) token toegewezen die gebruikt kan worden om zichzelf te verifiëren bij alle aankomende api calls | | | | | |
| Uitloggen | Gebruiker | | 11 | | Gebruiker kan uitloggen |
| **Definition of done:**   * De gebruiker kan uitloggen middels de uitlog knop * De toegewezen sessie token wordt verwijderd * De inlogknop is nu opnieuw zichtbaar | | | | | |
| Profiel aanpassen | Gebruiker | | 12 | | Gebruiker kan profiel aanpassen |
| **Definition of done:**   * De gebruiker dubbelklikt op het attribuut wat aangepast moet worden * De gebruiker krijgt een invoerveld en vult de juiste gegevens in * Invoerveld accepteert alleen tekst en andere unicode karakters * Er wordt gebruik gemaakt van input filtering (indien nodig, onderzoek) * Een gebruiker kan alleen zijn eigen profiel aanpassen | | | | | |
| Profiel inzien | Gebruiker | | 13 | | Gebruiker kan een profiel inzien |
| **Definition of done:**   * De gebruiker klikt op de naam van een gebruiker en wordt naar het bijbehorende profiel geleidt * Het profiel geeft de volgende persoonlijke informatie weer: gebruikersnaam, woonplaats, biografie en website * Verder worden de tweets, volgers en volgende van de gebruiker weergegeven * Op deze pagina zal de “gebruiker volgen” knop zichtbaar zijn | | | | | |
| Registreren | | Gebruiker | | 14 | Gebruiker kan een account aanmaken |
| **Definition of done:**   * Klant kan zelfstandig een account registreren. * Klant moet zijn persoonsgegevens kunnen invullen. * Er wordt een melding getoond als het ingevoerde wachtwoord niet voldoet aan de eisen. * Alles gegevens zijn op een veilige manier opgeslagen * De gebruikersnaam en het e-mailadres mogen maar een keer voorkomen in de database * Wanneer de gebruiker een al bestaand e-mailadres of gebruikersnaam gebruikt krijgt hij hiervan een melding | | | | | |

## Bijlage 2: Bcrypt hashes



**$2a$:** hieraan kan men identificeren dat Bcrypt gebruikt wordt

**12:** de zogenaamde cost parameter. Hoe hoger dit getal, hoe meer werk een CPU uit moet voeren om een wachtwoord te hashen. Dit is met name om brute force aanvallen tegen te gaan. 12 betekent dat de cost voor het hashen van dit wachtwoord 2^12 = 4096 rondes is.

**zBZaKyAPEX9lP/o2Nekuee:** De salt is een willekeurige string van 22 karakters die ervoor zorgt dat zelfs wanneer wachtwoorden gelijk zijn, ze toch anders worden opgeslagen in de database. Dit maakt het weer moeilijker voor hackers om meerdere accounts tegelijk binnen te dringen.

**P/eOotJZPTByVSvGa/uqVBSQRPaPxke:** Het daadwerkelijke wachtwoord wat wordt gekozen door de gebruiker, maar dan gecodeerd in een aangepaste versie van Base64. Dit is de string die daadwerkelijk vergeleken wordt met de invoer van een gebruiker bij het inloggen van het account.

## Bijlage 3: Security by design

**Security by design Kwetter project**

**Risk Analysis**

Risicoanalyse is het proces van het identificeren en analyseren van mogelijke problemen die een negatieve impact kunnen hebben. Dit proces wordt gedaan om organisaties te helpen deze risico's te vermijden of te beperken. Het uitvoeren van een cyberbeveiliging risicoanalyse helpt bij het identificeren, beheren en beveiligen van gegevens, informatie en activa die kwetsbaar kunnen zijn voor een mogelijke cyberaanval.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bedreiging** | **Type** | **Risico** | **Impact** | **Security level** |
| **Gebruikersgegevens lek** | Leak/Theft | Medium | High | High |
| **Admin inloggegevens lek** | Leak/Theft | Medium | High | High |
| **onveilige connectie met 3rd party** | Vulnerability | Medium | Medium | Medium |
| **Outdated libraries of SDK** | Vulnerability | Medium | Medium | Medium |

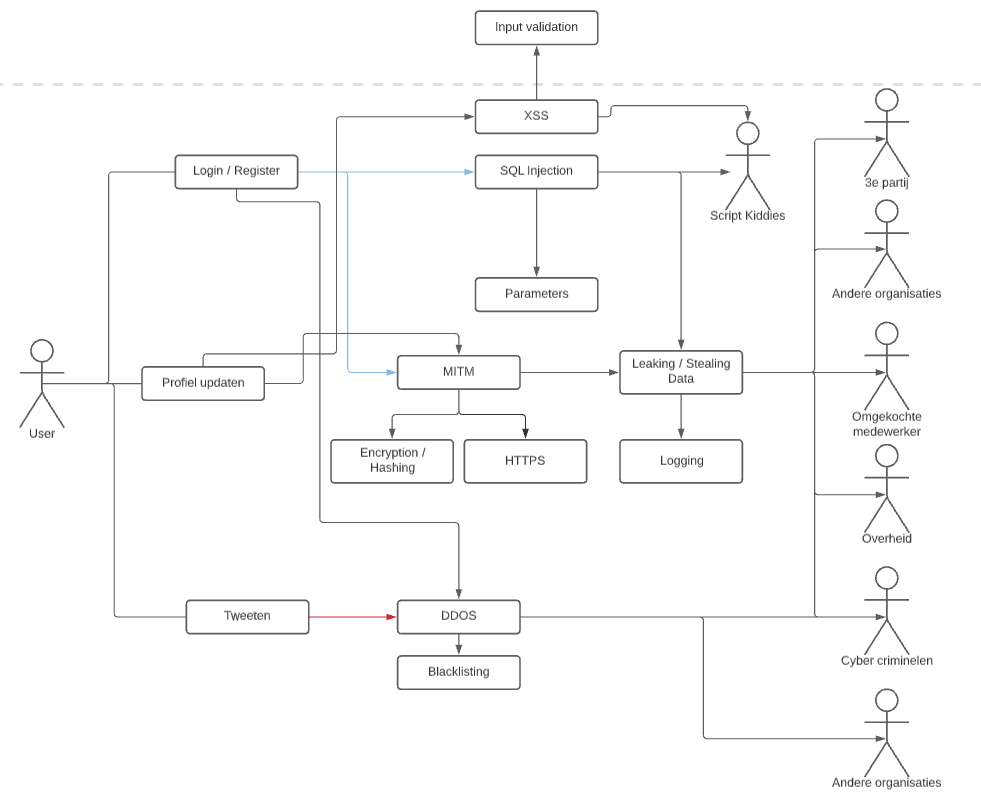
**Threat analysis**

Er wordt een threat analyse gemaakt om de mogelijke bedreigingen en hun potentiële schade voor een organisatie of project in kaart te brengen. In onderstaande tabel zijn de mogelijke bedreigingen voor het project opgesomd en hun mogelijke motivatie en misbruik activiteiten gedefinieerd.

|  |  |  |
| --- | --- | --- |
| **Threat Actor** | **Motivation** | **Misuse activities** |
| **Rogue medewerker** | Bemachtigen van gegevens, Blackmail material, stalking | Access |
| **Cybercriminelen** | Bemachtigen van gegevens, Blackmail material, stalking | Access, hacking tools |
| **Script Kiddies** | Learning, Fun, Thrill | Hacking tools |
| **Overheid** | Locaties, informatie, gegevens | Group hacking, Hired hackers |
| **Andere organisaties/concurrentie** | Bemachtigen gegevens, blackmail, faishiment | Hacking, access, rogue medewerker, hacker groep |
| **3rd party** | Bemachtigen data, blackmail, wraak | Hacking, access, hacking groep |
| **Onbekend** | Dit is een onbekende gebruiker die mogelijk in de toekomst een rol kan spelen | Dit is een onbekende gebruiker die mogelijk in de toekomst een rol kan spelen |

**Misuse cases diagram**

Een misbruik case diagram wordt gebruikt om mogelijke bedreigingen in kaart te brengen en zien hoe deze kunnen worden uitgevoerd op het project dat zal worden gemaakt. Het misbruik geval heeft de gebruikers en de functies die ze kunnen uitvoeren, en de dreiging actoren met de dreigingen die ze kunnen uitvoeren op de bestaande functies.



**OWASP top 10**

De OWASP top 10 is een lijst met de top tien meest voorkomende security kwetsbaarheden. Sinds dat onze applicatie een webapplicatie is zullen we kijken naar de OWASP top 10 voor webapplicaties. We zullen per kwetsbaarheid in de OWASP kijken wat er gedaan kan worden om de kwetsbaarheid te voorkomen in de applicatie.

1. **Injectie**

De meest gebruikte injectie om ongewenst toegang te krijgen door hackers, is de SQL injectie. Met een SQL injectie voert de hacker een query direct uit op de database, waarmee diegene data verkrijgen of zichzelf toegang verlenen. Het kan ook zijn dat mede door een SQL injectie de hele applicatie niet meer werkt en alle data verloren gaat.

De bekendste injectiemethoden zijn:

* SQL injectie
* OS commando injectie
* LDAP injectie

**Injectie binnen het project**

Om te voorkomen dat hackers gebruik kunnen maken van injectie binnen het Kwetter heb ik een aantal stappen genomen. Zo heeft elke backend een eigen database met stored procedures. Hierdoor staat de query los van de code en kan deze op databaseniveau aangepast worden. Ook worden alle stored procedures aangeroepen door middel van zogenaamde “geparametriseerde queries”. Dit zorgt ervoor dat eventuele parameters niet in de query worden meegenomen die daardoor kwaadaardig kan zijn.

Feitelijk gezien is het ook mogelijk om via invoercontrole SQL injectie tegen te gaan, maar dit is totaal niet efficient en waterdicht. Ik kies er daarom voor om alleen op de backend SQL injectie tegen te gaan.

1. **Authenticatie fouten**

Wanneer je een webapplicatie met een gebruikersaccount hebt, dan is het zeer aannemelijk om te zeggen dat er een authenticatiesysteem wordt gebruikt. Het terugsturen van foutmeldingen die een hacker potentieel kunnen helpen bij het vaststellen of een gebruikersaccount bestaat, vormt een behoorlijk risico. Door een foute implementatie van deze authenticatiesysteem kunnen gebruikersgegevens zoals gebruikersnamen en wachtwoorden buitgemaakt worden.

**Authenticatie binnen het project**

Voor het Kwetter project wordt er gebruik gemaakt van een authenticatie systeem. Dit systeem kan twee meldingen teruggeven aan de gebruiker: inloggen geslaagd of inloggen niet geslaagd. Hierdoor kan een gebruiker niet direct zien of een gebruikersnaam wel al bestaat of niet. Ook wordt er op de frontend geen controle gedaan of de gebruikersnaam en wachtwoorden wel kloppen, alleen of deze ingevuld zijn. Bij het afhandelen van persoonlijke gegevens is het belangrijk dat hier zo min mogelijk inzichtelijk is voor een eventuele hacker. Hoewel dit zal zorgen voor extra requests naar de backend zorgt dit er wel voor dat er zeker geen validatiefouten zitten in de frontend die een hacker zou kunnen gebruiken.

1. **Gevoelige data**

Vrijwel alle webapplicaties werken met zeer gevoelige data. Het is dus belangrijk om deze goed af te schermen, door middel van een goede encryptie. Inmiddels hebben wij hier ook een wetgeving dat ons verplicht om goed met gevoelige data om te gaan, genaamd de AVG wetgeving.

**Gevoelige data binnen het project**

Er zijn een aantal manieren om te zorgen dat gevoelige data van gebruikers van de applicatie niet zomaar in de handen van kwaadwillenden kan vallen. Zo is het belangrijk om alleen data op te slaan die relevant is voor de werking van de applicatie. Wanneer telefoonnummers niet gebruikt worden in de applicatie moeten deze ook niet worden opgeslagen. Data die niet wordt opgeslagen kan ook niet worden gelekt.

Ook is het belangrijk dat data die wel opgeslagen wordt, versleuteld wordt. Door gebruik te maken van nieuwe encryptietechnieken kan een hacker nog steeds niks met de data ook bij een volledige lek van de database. Door te zorgen dat de wachtwoorden aan bepaalde eisen voldoen worden aanvallen als brute-force ook een stuk minder bruikbaar.

1. **Externe XML entities (XXE)**

Hackers kunnen door middel van een XML bestand zichzelf toegang verlenen tot bestanden en programma’s op het besturingssysteem van de server. Dit komt met name door gedateerde XML verwerkers. Het zou ook mogelijk kunnen zijn dat de hacker data aan koppelingen, zoals bijvoorbeeld met een boekhoudpakket, door kan sturen met foutieve data en daardoor geld kan overmaken naar een bankrekening.

**Externe XML entities binnen het project**

Binnen het Kwetter project worden er geen XML parsers gebruikt. Deze aanval is daarom niet relevant voor het project.

1. **Rollen en rechtensysteem kwetsbaarheden**

Wanneer een gebruiker foutieve rechten krijgt bij zijn rol, kan de gebruiker eventueel meer handelingen verrichten dan gewenst is. Het is daarom erg belangrijk dat het rollen- en rechtensysteem goed op elkaar afgestemd en getest wordt.

**Rollen en rechtensysteem kwetsbaarheden binnen het project**

Het is belangrijk dat de verschillende rollen binnen een systeem goed beschreven staan. Zo moet een administrator veel meer mogen dan een “gewone” gebruiker. Dit moet getest worden met unit- en systeemtesten. Dit voorkomt dat een gewone gebruiker bij functionaliteiten komt waar hij of zij niet bij hoort te mogen.

Ook moeten de endpoint van een API goed afgebakend worden met het gebruik van tokens. Dit wordt binnen ons project dan ook gedaan. Bij elke afgebakende API call wordt eerst gecheckt of de gebruiker een valide token heeft met de juiste rechten. Dit voorkomt ook rechtstreekse calls naar de API, aangezien een valide token alleen uitgegeven kan worden door de API.

Binnen het Kwetter project zijn er twee mogelijke rollen: gebruiker en administrator. Op dit moment is er in de applicatie nog geen mogelijkheid om iemand een administrator te maken. In de database is deze scheiding er wel. Wanneer dit wel wordt geïmplementeerd zou hier een duidelijke scheiding tussen gedocumenteerd moeten worden.

1. **Foutieve configuratie**

Een ontwikkelaar gebruikt configuratiebestanden om zijn server en frameworks in te stellen, die absoluut niet inzichtelijk mogen zijn voor het openbare publiek. Het komt verrassend vaak genoeg voor dat een ontwikkelaar zijn configuratie niet op orde brengt voor het besturingssysteem van de server en van de gebruikte frameworks, wat de hacker in staat kan stellen om toegang te krijgen.

**Foutieve configuratie binnen het project**

Binnen het Kwetter project zorg ik ervoor dat alle Dockerfiles alleen de poort exposen die ze gebruiken om requests af te handelen. Ook heb ik ervoor gezorgd dat de database alleen IP-adressen van mijn eigen subnetmask doorlaat. Andere IP-adressen worden standaard geblokkeerd en moeten toestemming krijgen van mij.

1. **Cross-Site Scripting (XSS)**

Als een webapplicatie onvoldoende validatie toepast op de te verwerken data, kan het voorkomen dat een hacker code injecteert in de webapplicatie. Hiermee kan de hacker bijvoorbeeld onveilige bestanden plaatsen op de computer van een andere gebruiker, maar ook de webapplicatie volledig vernietigen.

**Cross-Site Scripting (XSS) binnen het project**

XSS is een van de meest voorkomende en dus belangrijke manieren voor een hacker om een applicatie binnen te dringen. Dit gebeurt met name door invoervelden niet te valideren waardoor een kwaadwillende Javascript code uit kan voeren voor elke gebruiker. Wanneer een gebruiker een account maakt en hier als naam <script>alert(“Ik ben een hacker”)</script> invult en dit niet gevalideerd wordt, krijgt iedere gebruiker een pop-up venster te zien wanneer de naam van die specifieke gebruiker geladen wordt. Dit kan makkelijk voorkomen worden door invoervalidatie en wordt in de meeste frameworks al automatisch geïmplementeerd. Ook binnen het Kwetter project zorgt React.JS ervoor dat XSS niet kan worden uitgevoerd.

1. **Onveilige serialisatie**

Er zijn webapplicaties die objecten omzetten naar tekst, al voor deze in de database opgeslagen worden. Wanneer een webapplicatie hiervan gebruik maakt en niet voldoende controles uitvoert op de ingevulde data, kan de hacker commando’s uitvoeren op het besturingssysteem van de gebruikte server.

**Gebruik van serialisatie in het project**

In het Kwetter project worden alleen profielfoto’s van gebruikers opgeslagen in de databsae. Hiervoor worden de base64 strings opgeslagen en ingeladen wanneer iemand een profiel bezoekt. Deze strings worden direct ingeladen in een <img> tag en zijn daardoor niet schadelijk voor de beveiliging van de applicatie.

1. **Gebruik van software met bekende kwetsbaarheden**

Vaak maakt een ontwikkelaar gebruik van open-source frameworks of stukken software dat door andere ontwikkelt worden. Als deze niet tijdig geüpdatet worden door de makers en/of de gebruikers, kunnen veiligheidsrisico’s benut worden door hackers.

**Gebruik van software met bekende kwetsbaarheden in het project**

In het project gebruik ik verschillende libraries of software pakketten. Het pakket wat ik gebruik om de wachtwoorden te hashen wordt standaard ondersteund door Springboot en is op het moment van spreken nog niet te hacken. Verder houd ik het zoveel mogelijk bij standaard frameworks en libraries.

1. **Onvoldoende logging en monitoring**

Feitje, een lek wordt gemiddeld pas na 200 dagen gemeld bij de makers van de webapplicatie. Als de makers van de webapplicatie onvoldoende of zelfs helemaal geen logbestanden bijhoudt, waarin vastgehouden wordt wat er precies gebeurd binnen de applicatie kan het heel lang duren voor de makers het lek boven water krijgen.

Met behulp van monitoring heeft een ontwikkelaar tevens veel sneller in de gaten of een webapplicatie nog wel bereikbaar is en kan hierop beter reageren wanneer dit niet het geval is.

**Monitoring in het project**

Binnen het Kwetter project ben ik aan het onderzoeken hoe ik monitoring het beste in kan zetten. Dit is een cloud service die ik in de laatste weken van het project nog wil toevoegen om te zorgen dat bepaalde acties van gebruikers en beheerders worden opgeslagen. Op dit moment worden alle API calls en responses al gelogd, maar niet opgeslagen.

## Bijlage 4: Performance testen

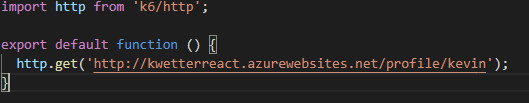
Een Enterprise applicatie wordt normaliter door vele gebruikers tegelijkertijd gebruikt. Mede daarom is gekozen voor een microservice architectuur. Het is wel belangrijk om te zorgen dat deze microservices ook wel de hoge workload aan kunnen. Dit kan getest worden onder de noemer Performance testen.

**K6 testen**

K6 is een opensource load testing tool, ook wel stresstesten genoemd. De tool stuurt verschillende virtuele gebruikers op een URL af en geeft dan inzage in:

* Hoe snel dit gaat
* Hoe vaak er een response komt binnen de afgesproken tijd
* Hoevaak er iets mis gaat

Om aan te geven waar de requests heen gestuurd worden maak ik een simpel Javascript-bestand.



Dit script stuurt een GET-request naar mijn frontend en haalt daar het profiel op van de gebruiker “kevin”. Door mijn opdrachtprompt te openen en het volgende commando in te voeren zal de K6-tool zijn werk gaan doen.

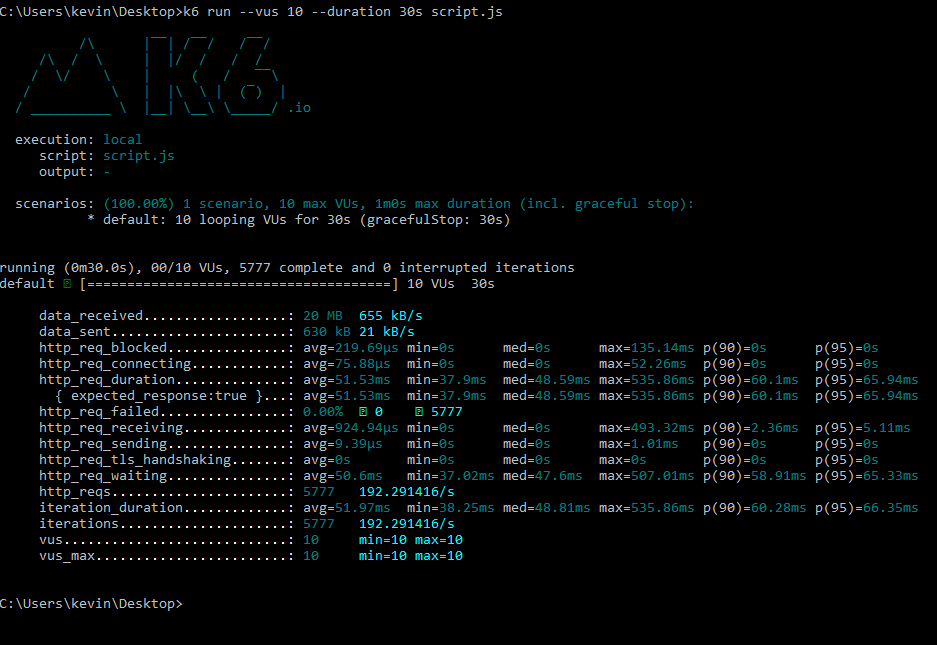
**K6 run –vus 10 –duration 30s script.js**

In dit commando worden twee variabelen meegegeven.

**--vus:** Het aantal *virtual users* dat tegelijkertijd proberen de request uit te voeren. Een *virtual user* kan vergeleken worden met een daadwerkelijke eindgebruiker.

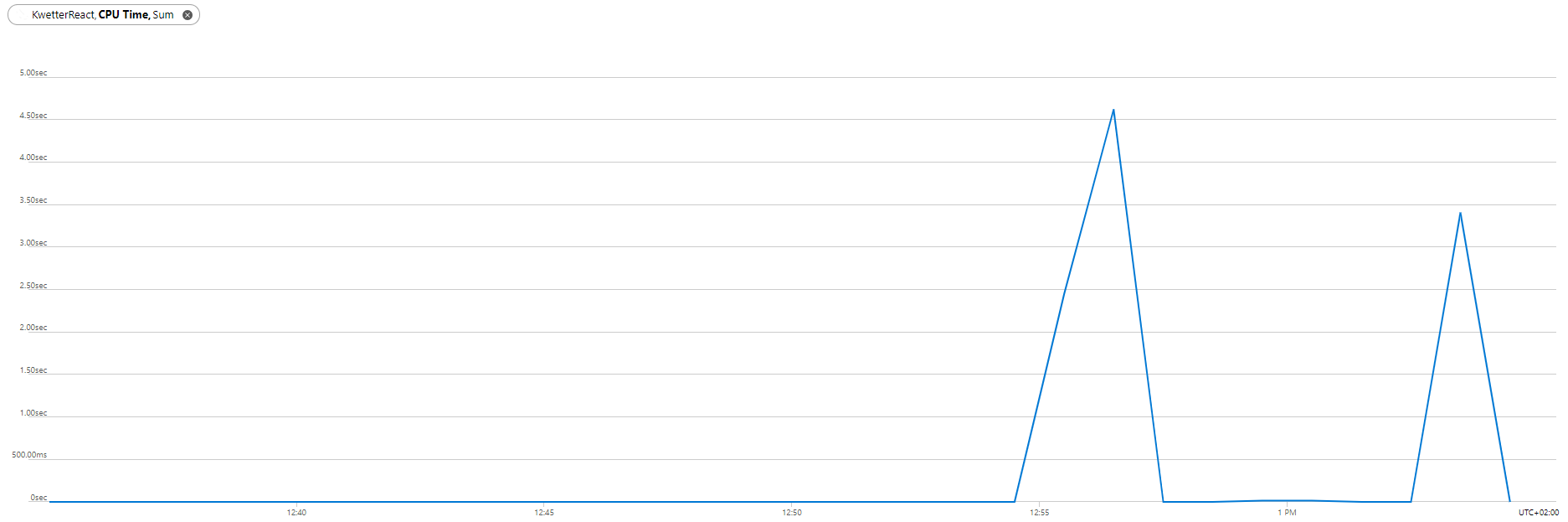
**--duration:** Dit geeft aan hoelang de tool moet runnen in seconden.

Het resultaat van dit commando wordt op de volgende pagina weergegeven.



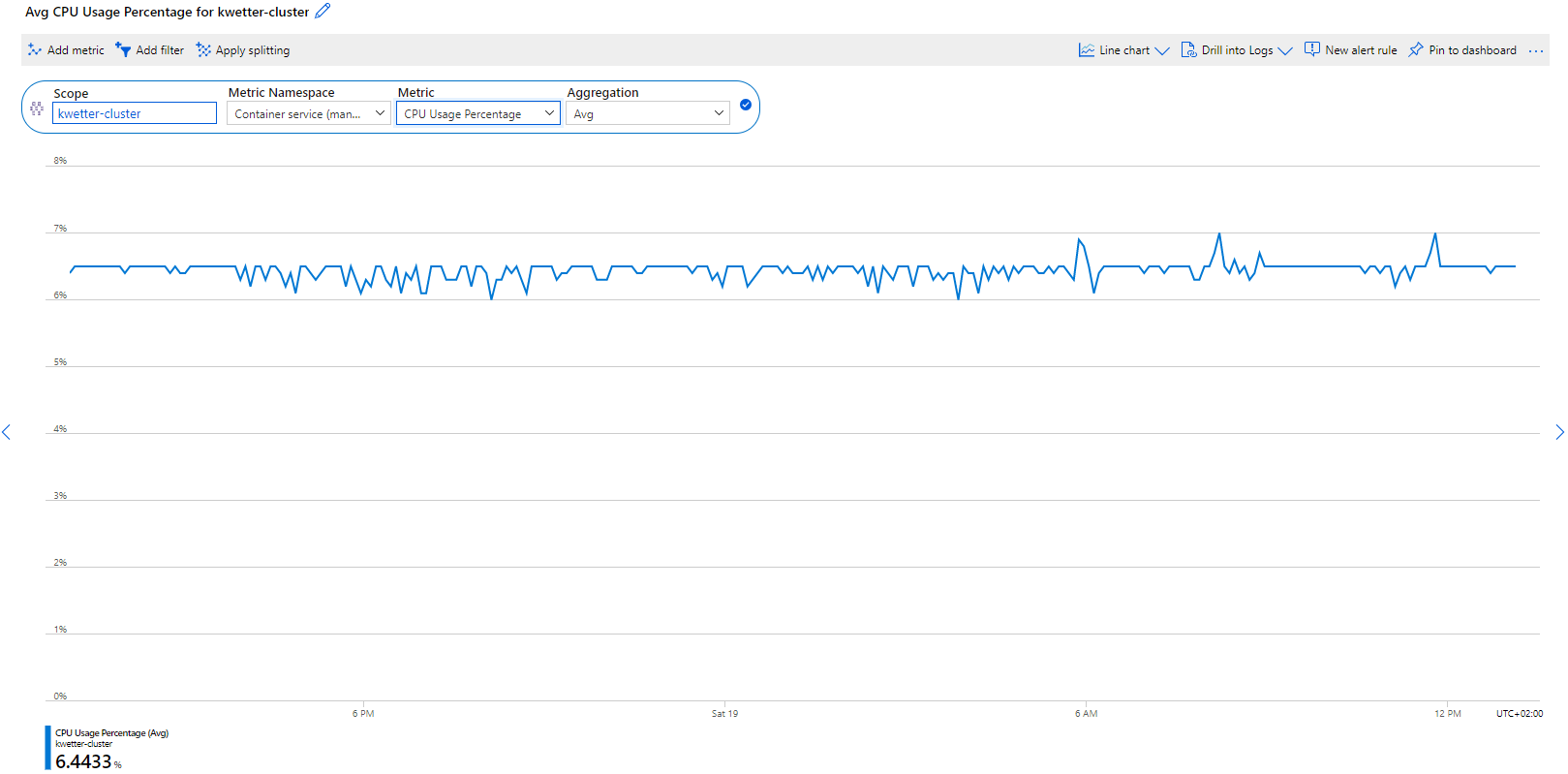
De frontend kan binnen 30 secondes door 10 virtual users 5777x worden aangeroepen. De gemiddelde reactietijd is 51ms en geen enkele request heeft geen response ontvangen.

Hoeveel deze gegevens natuurlijk prima klinken is het wel belangrijk om te kijken in hoeverre de frontend belast werd door deze requests. Hiervoor heeft Azure een service gemaakt genaamd Azure Monitoring. Elke service die bij Azure draait kan hier mee gemonitord worden.



Dit is de grafiek van de CPU tijd die de frontend nodig heeft om de requests af te handelen. Azure gebruikt deze metric om te kijken of applicatie niet boven een bepaald quotom gaat. Wanneer dit wel gebeurt zul je moeten upgraden naar een duurder plan.

Ook de verschillende microservices, die in een Kubernetes cluster draaien hebben deze metrics. Om te kijken hoe dit cluster de klap van de 5777 requests opgevangen heeft kijken we naar CPU usage percentage metric.



Aangezien ik een betaald plan heb voor mijn Kubernetes cluster, heeft deze veel meer resources en kan deze zo een klap dus ook makkelijk opvangen. Dit is een goed bewijs dat mijn applicatie ook in het echt veel gebruikers tegelijkertijd aan kan.

## Bijlage 5: Non-functional requirements

Normaal gesproken kunnen de non-functionals aangetoond worden door middel van unit- en integratietesten. Aangezien ik in mijn project niet genoeg tijd heb gehad om deze volledig te implementeren zal ik deze in deze bijlage nogmaals doorlopen en bekijken of ik deze aan heb getoond.

|  |  |
| --- | --- |
| **Omschrijving** | **Resultaat** |
| Het ophalen van data duurt maximaal 1 seconde. | Uit de resultaten van de K6 performance testen blijkt dat bij 20 gelijktijdige gebruikers de gemiddelde laadtijd van data ~50ms bedraagt |
| Het laden van een pagina duurt maximaal 1 seconde. | Bij het laden van een profielpagina is de laatste resource na 857ms geladen. Dit valt dus nog net binnen de seconde. Mocht het langer dan 1 seconde hebben geduurd zou het op te lossen zijn door een duurder Kubernetes cluster in te zetten met meer resources. |
| De website moet minimaal 500 gelijktijdige gebruikers aankunnen zonder impact te hebben op de prestaties. | Volgens de K6 performance testen vertraagt de website tot een response van 855ms bij 500 gelijktijdige gebruikers. Ook dit is op te lossen door middel van meer resources kopen bij Azure. |
| Een tweet die geplaatst wordt door een andere gebruiker moet binnen 30 seconden zichtbaar zijn voor zijn volgers | Deze functionaliteit is niet volledig geïmplementeerd, maar een tweet is binnen 1 seconden al zichtbaar op de profielpagina van de gebruiker. |
| Gebruikersgegevens worden veilig opgeslagen & wachtwoorden worden versleuteld. | Dit wordt gedaan. Zie kopje security by design |
| Gegevens worden verstuurd over een beveiligde (HTTPS) verbinding. | Dit wordt ook gedaan |
| De persistentie van het systeem moet voldoen aan de AVG / GDPR. | Dit wordt gedaan. Zie kopje Distributed data |
| De website moet mee kunnen schalen op basis van desktop en mobiele schermformaten. | Uit systeemtesten blijkt dat de website op apparaten met een resolutie van 1024x1366(tablets) nog steeds goed meeschaalt. Bij kleinere apparaten kan dit niet gegarandeerd worden |
| Een nieuwe klant moet binnen 5 minuten het proces van een account creëren en een tweet plaatsen snappen | Een kleinschalig onderzoek bij klasgenoten heeft uitgewezen dat dit binnen 1 minuut al lukt. |
| De broncode is geschreven volgens de SOLID-principes. | Dit is gedaan |
| De systeemprestaties en uptime moeten automatisch gemonitord kunnen worden via een monitoringssysteem. | Dit kan via het monitoringssysteem van Azure |
| De applicatie moet een uptime hebben van minimaal 99.9% | Dit ligt volledig aan de uptime van Azure. [In de documentatie](https://docs.microsoft.com/en-us/azure/aks/uptime-sla) garandeert Microsoft een uptime van 99.95% |
| Wanneer een van de functionaliteiten uitvalt moet de rest van de applicatie blijven werken | Door gebruik te maken van de microservices architectuur is ook dit gerealiseerd. |