SAD Semester 6

Naam: Rienk Engbrenghof

Studentnummer: 429059

Klas: S6-RB03T

# Introductie

In dit document zullen voornamelijk de technische keuzes omtrent mijn project gedocumenteerd staan. Hierbij moet worden gedacht aan een document met alle vereisten, keuzes en verschillende diagrammen die deze keuzes ondersteunen.

# Versiebeheer

|  |  |
| --- | --- |
| 21/2/2022 | Opzet gemaakt. |
| 23/2/2022 | Architectuurdocument toegevoegd. |
| 19/3/2022 | Aanpassingen aan stories. |
| 20/4/2022 | Feedback Marc toegepast. |
| 23/4/2022 | Scenario’s toevoegen. |
| 23/5/2022 | Scenario’s aangepast. |

# Repositories

Link naar repository: https://github.com/Pretter-S6

# Inhoudsopgave

[Introductie 2](#_Toc104206786)

[Versiebeheer 3](#_Toc104206787)

[Repositories 3](#_Toc104206788)

[Inhoudsopgave 4](#_Toc104206789)

[1. SAD 5](#_Toc104206790)

[1.1 Inleiding 5](#_Toc104206791)

[1.2 Userstories 5](#_Toc104206792)

[1.3 Usecases 6](#_Toc104206793)

[1.4 Non-functional requirements 9](#_Toc104206794)

[1.5 Diagrammen 10](#_Toc104206795)

[2. Voortgang 12](#_Toc104206796)

[2.1 Sprint 1 12](#_Toc104206797)

[2.2 Sprint 2 15](#_Toc104206798)

[2.3 Sprint 3 16](#_Toc104206799)

[3. Scenario’s 17](#_Toc104206800)

[3.1 Scalable Architectures 17](#_Toc104206801)

[3.2 Development and Operations 18](#_Toc104206802)

[3.3 Cloud services 19](#_Toc104206803)

[3.4 Security by design 20](#_Toc104206804)

[3.5 Distributed data 21](#_Toc104206805)

[4. Conclusie 22](#_Toc104206806)

# SAD

## 1.1 Inleiding

Het architectuurdocument wordt opgesteld om een duidelijk beeld te creëren over de applicatie. Hierin staan de keuzes gedocumenteerd. De eisen staan gedocumenteerd als userstories, deze worden vervolgens usecases.

De functionele en non-functionele eisen dienen allemaal uitgewerkt te worden. Op deze manier wordt de best mogelijke applicatie gemaakt.

## 1.2 Userstories

1. Als gebruiker wil ik kunnen inloggen, zodat iedereen een persoonlijk account heeft en geïdentificeerd kan worden.

2. Als gebruiker wil ik kunnen registeren, zodat ik een account kan aanmaken en hiermee kan inloggen.

3. Als gebruiker wil ik iemand kunnen volgen, zodat ik alle updates van deze persoon binnenkrijg op mijn feed.

4. Als gebruiker wil ik iemand kunnen ontvolgen, zodat ik zijn/haar updates niet meer krijg.

5. Als gebruiker wil ik mijn profiel kunnen beheren, zodat ik hierop aanpassingen kan doen en mijn informatie kan inzien.

6. Als gebruiker wil ik een post kunnen doen, zodat mensen die mij volgen kunnen zien wat ik post.

7. Als gebruiker wil iemand anders zijn profiel kunnen zien, zodat ik kan kiezen of ik deze persoon wil gaan volgen of niet.

8. Als gebruiker wil ik een foto toevoegen aan mijn post, zodat er een visuele representatie aan de post gekoppeld is.

9. Als gebruiker wil een post kunnen liken, zodat ik kan interacteren met posts van andere personen.

10. Als gebruiker wil ik kunnen reageren op een post van iemand anders, zodat ik mijn mening over de post kan achterlaten.

11. Als gebruiker wil ik kunnen uitloggen, zodat ik niet meer verbonden ben met mijn account.

12. Als gebruiker wil ik kunnen zoeken naar andere gebruikers, zodat ik hen kan gaan volgen.

## 1.3 Usecases

|  |  |
| --- | --- |
| Use case 01 | Inloggen |
| Actor(s) | Gebruiker |
| Aannames | Actor heeft al een account aangemaakt. |
| Basic flow | 1. Actor navigeert naar de login pagina. Voert zijn/haar emailadres en wachtwoord in. En klikt op inloggen. 2. Het systeem autoriseert de gebruiker en geeft bij de juiste combinatie toegang tot het systeem. Vervolgens wordt de actor naar de juiste pagina genavigeerd. |

|  |  |
| --- | --- |
| Use case 02 | Registreren |
| Actor(s) | Gebruiker |
| Aannames | Actor heeft nog geen account bij Pretter. |
| Basic flow | 1. Actor navigeert naar de register pagina. Voert zijn/haar gebruikersnaam, emailadres en wachtwoord (tweemaal) in. En klikt op registreer. 2. Het systeem controleert de gegevens. Wanneer deze correct zijn maakt het systeem het account aan. En wordt de actor naar de juiste pagina genavigeerd. |

|  |  |
| --- | --- |
| Use case 03 | Iemand volgen |
| Actor(s) | Gebruiker |
| Aannames | Actor volgt gebruiker nog niet. |
| Basic flow | 1. Actor navigeert naar iemand zijn pagina (Use case 07) en klikt op de knop om deze persoon te volgen. 2. Het systeem registreert deze handeling en zorgt ervoor dat de actor de gebruiker volgt. |

|  |  |
| --- | --- |
| Use case 04 | Ontvolgen |
| Actor(s) | Gebruiker |
| Aannames | Actor volgt gebruiker. |
| Basic flow | 1. Actor navigeert naar iemand zijn pagina (Use case 7) en klikt op de knop om deze persoon te ontvolgen. 2. Het systeem registreert deze handeling en zorgt ervoor dat de actor de gebruiker ontvolgt. |

|  |  |
| --- | --- |
| Use case 05 | Profiel beheren |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor navigeert naar zijn/haar eigen pagina. En kiest ervoor om 1 van zijn gegevens aan te passen. Bijvoorbeeld zijn gebruikersnaam. 2. Het systeem registreert de verandering, slaat deze op en voert deze door. |

|  |  |
| --- | --- |
| Use case 06 | Posten |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor navigeert naar de homepagina. Voert een tekst in voor de post die hij wil doen. En klikt vervolgens op: ‘plaatsen’. 2. Het systeem controleert de post, slaat deze op en voert het door voor de actor. Het systeem toont deze post aan de actor. |

|  |  |
| --- | --- |
| Use case 07 | Profiel inzien |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor navigeert naar iemand zijn pagina. Door middel van zoeken bijvoorbeeld (Use case 12) 2. Systeem laadt de pagina en toont deze aan de actor. |

|  |  |
| --- | --- |
| Use case 08 | Foto toevoegen |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor plaatst een post (Use case 06) en kiest ervoor om een foto te uploaden, gelinkt aan de post. 2. Het systeem slaat de foto en de post op. Vervolgens wordt de post getoond aan de actor. |

|  |  |
| --- | --- |
| Use case 09 | Post liken |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor ziet een post van een gebruiker die hij/zij volgt op zijn homepagina en klikt op ‘like’. 2. Het systeem slaat dit op en voert het door. |

|  |  |
| --- | --- |
| Use case 10 | Reageren op post |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor ziet een post van een gebruiker die hij/zij volgt op zijn homepagina en typt zijn reactie. Vervolgens klikt hij/zij op ‘reageer’. 2. Het systeem slaat dit op en voert het door. |

|  |  |
| --- | --- |
| Use case 11 | Uitloggen |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor navigeert naar zijn profielpagina en klikt hier op uitloggen. 2. Het systeem logt de gebruiker uit en stuurt de actor door naar de juiste pagina. |

|  |  |
| --- | --- |
| Use case 12 | Zoeken naar gebruikers |
| Actor(s) | Gebruiker |
| Aannames |  |
| Basic flow | 1. Actor bevindt zich op de homepagina en typt een naam van een gebruiker in de zoekbalk. Vervolgens klikt de actor op ‘zoek’. 2. Het systeem zoekt naar de naam en geeft de gevonden resultaten terug. 3. De actor klikt op een van de resultaten. 4. Het systeem stuurt de actor door naar het profiel waarop geklikt is. |

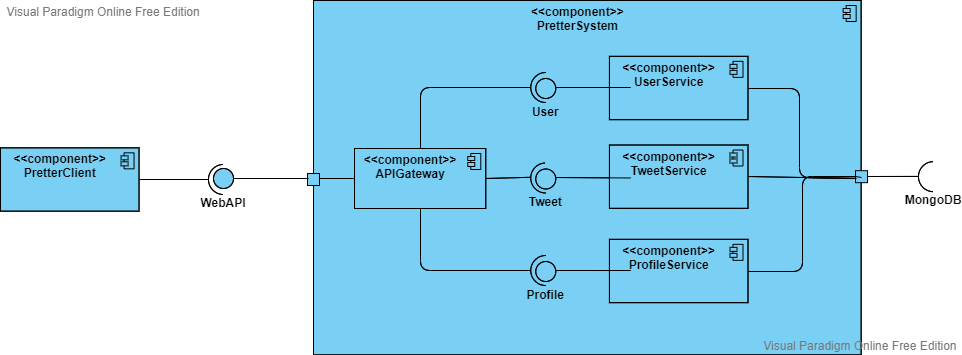
## 1.4 Non-functional requirements

|  |  |  |
| --- | --- | --- |
| NF 01 | Schaalbaarheid | Tijdens het designen en opzetten van de architectuur wordt er rekening gehouden met de mogelijkheid om de applicatie makkelijk uit te kunnen breiden.  Hiervoor worden bepaalde keuzes gemaakt in de architectuur. |
| NF 02 | Performance | De snelheid van de applicatie is ook van belang. Deze wordt ook grotendeels gewaarborgd door een juiste architectuur. |
| NF 03 | Robuustheid | De applicatie gaat juist om met errors. Hij vangt deze af en zorgt ervoor dat het allemaal toegankelijk is voor de gebruiker.  Het is ook van belang dat binnen het systeem errors juist worden afgevangen. Zodat de uptime van het systeem gegarandeerd kan worden. |
| NF 04 | Gedistribueerde data | In het design en in de architectuur moet rekening worden gehouden met het juist opslaan van data. Hierbij moet je denken aan hoe data wordt opgeslagen en hoe deze wordt gedistribueerd. |
| NF 05 | Security | In het onderzoek wordt onderzocht welke risico’s het meest voorkomen bij soortgelijke applicaties. En vervolgens wordt er bij het design en de architectuur rekening gehouden met deze risico’s. Het systeem wordt zo geïmplementeerd dat deze risico’s beperkt worden. |

## 1.5 Diagrammen

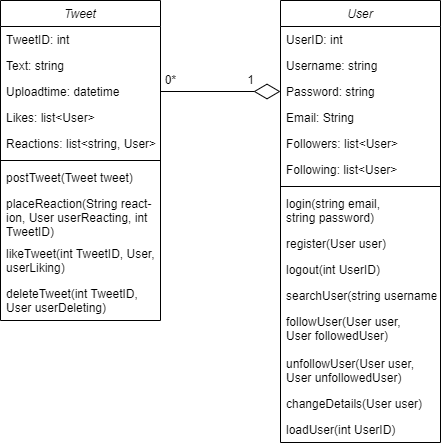
**Componentdiagram**

Dit diagram bevat een overzicht van de opdeling van de client (frontend) dat via de gateway de verschillende services aanroept. Deze services hebben vervolgens communicatie met de DB.



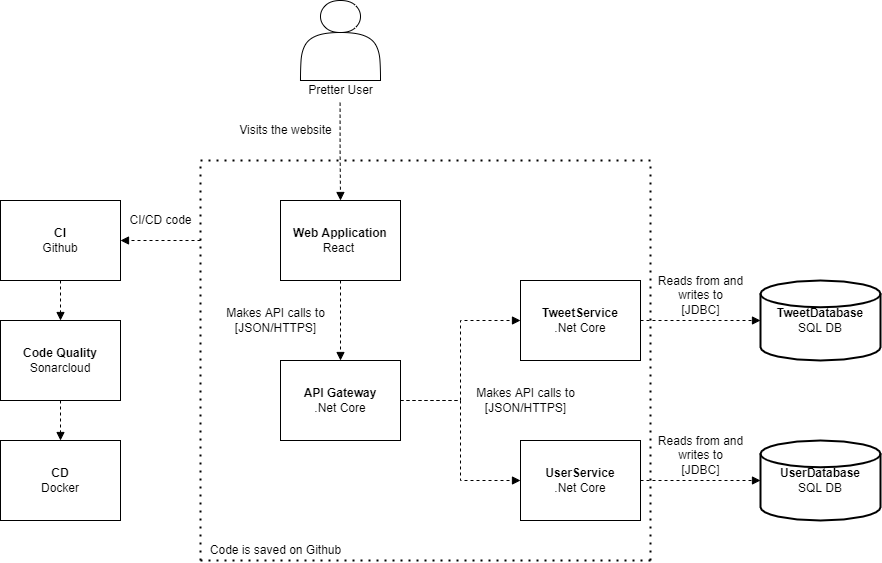
**ERD Diagram**

Dit diagram bevat een overzicht van hoe de modellen binnen de database en binnen de backend eruit gaan zien.



**Containerdiagram**

Dit diagram bevat alle componenten die zullen worden gerealiseerd voor mijn applicatie. Hierbij is alles binnen de gestippelde lijn geprogrammeerd door mij. Alles erbuiten zijn integraties.

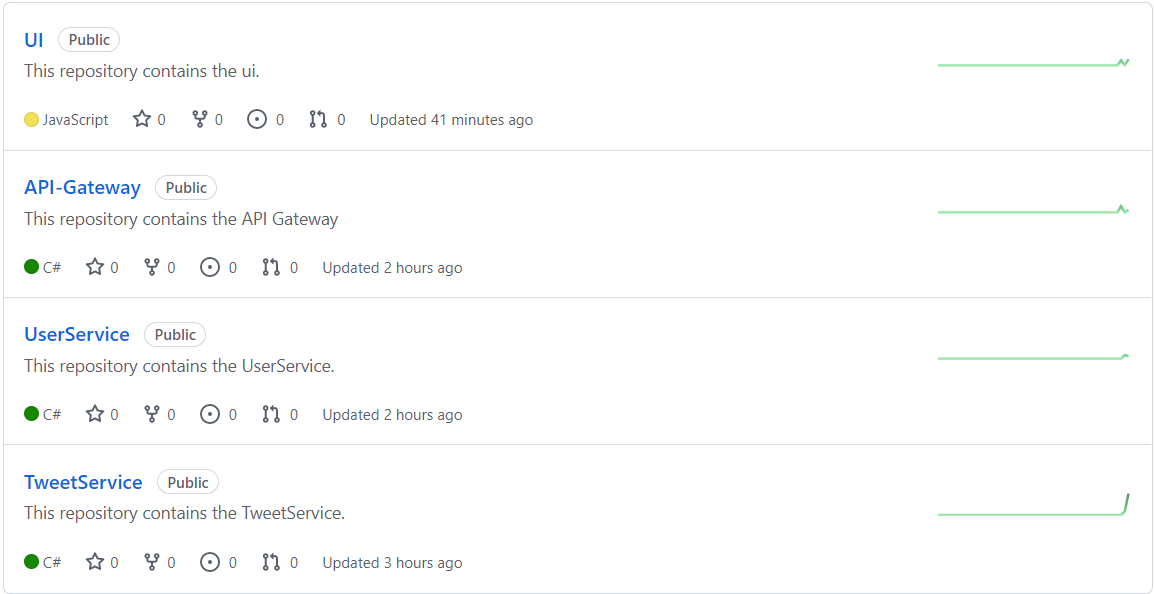
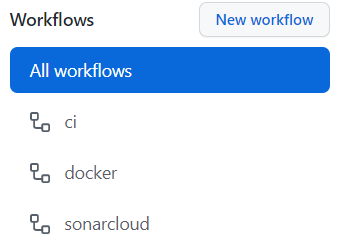


# Voortgang

## 2.1 Sprint 1

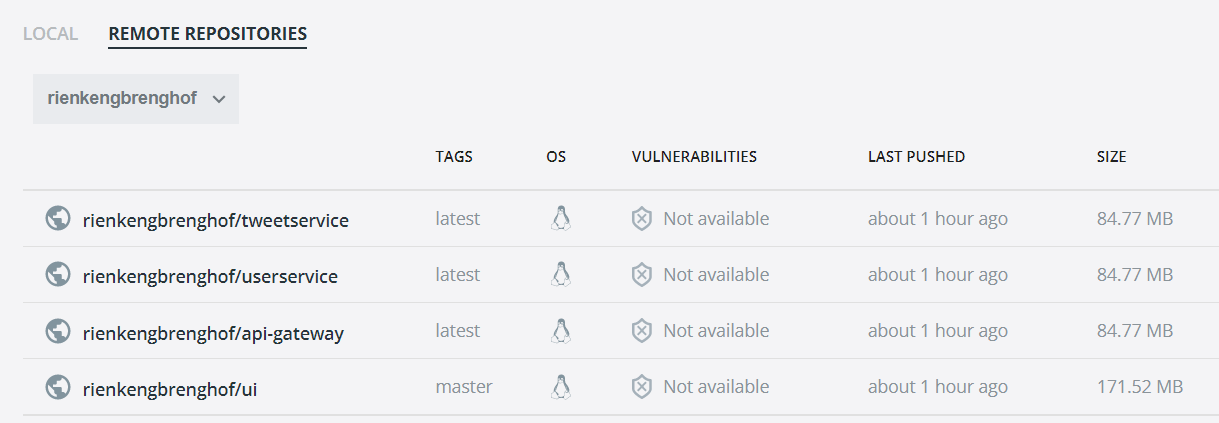
Binnen het groepsproject stond al vast welke technieken er worden gebruikt voor het behalen van de requirements. Hier is gekozen voor een VUE frontend, een API-gateway die gebruik maakt van het ocelot framework (.NET Core) en services die ook draaien op .NET Core. Deze keuzes staan al vast en zijn al goedgekeurd door de stakeholders.

Binnen mijn eigen project maak ik gebruik van een soortgelijke architectuur met dezelfde technieken. Hier gebruik ik zoals te zien in mijn architectuurdocument: react frontend, .NET Core gateway en .NET Core services. Op basis hiervan en het idee van mijn applicatie, heb ik verschillende eisen voor mijn applicatie opgesteld.

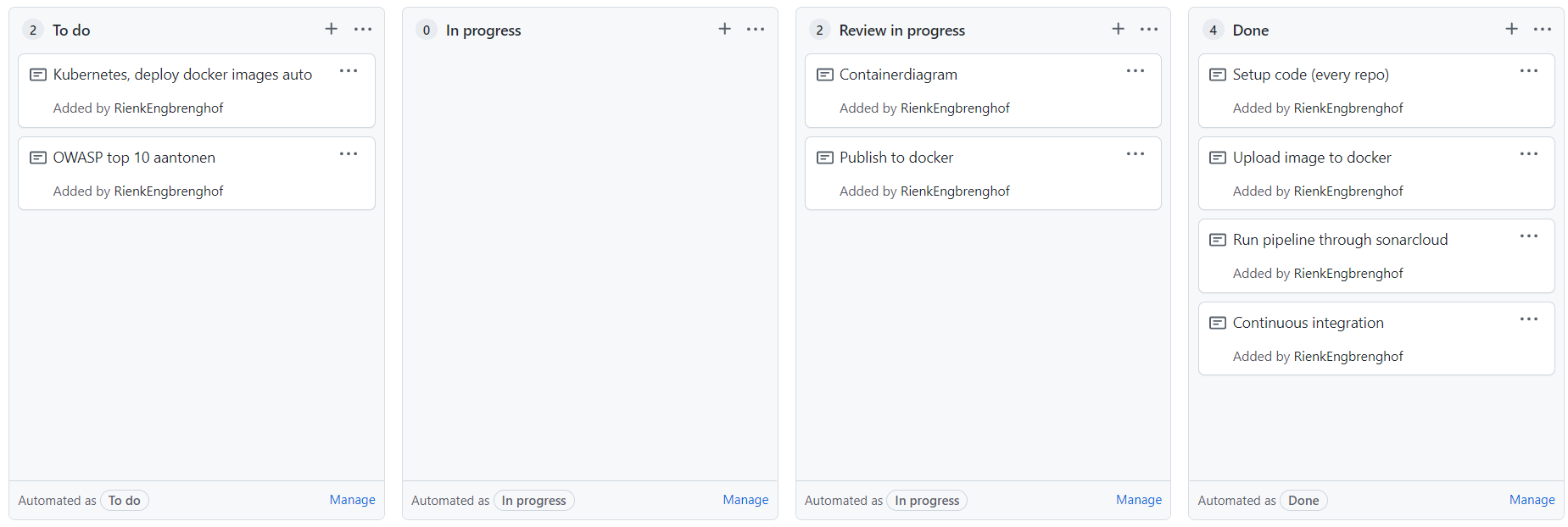


Bovenstaande foto laat mijn architectuur goed zien. Hier heeft elke service ook een toegewijde database. Elke repository bevat al een applicatie met code. En elke repository bevat CI, Code quality check en een image push naar Docker:

Hierbij heb ik mijn architectuur gedocumenteerd, aangepast wanneer nodig en is dit de initiële architectuur waartoe ik ben gekomen (walking skeleton).



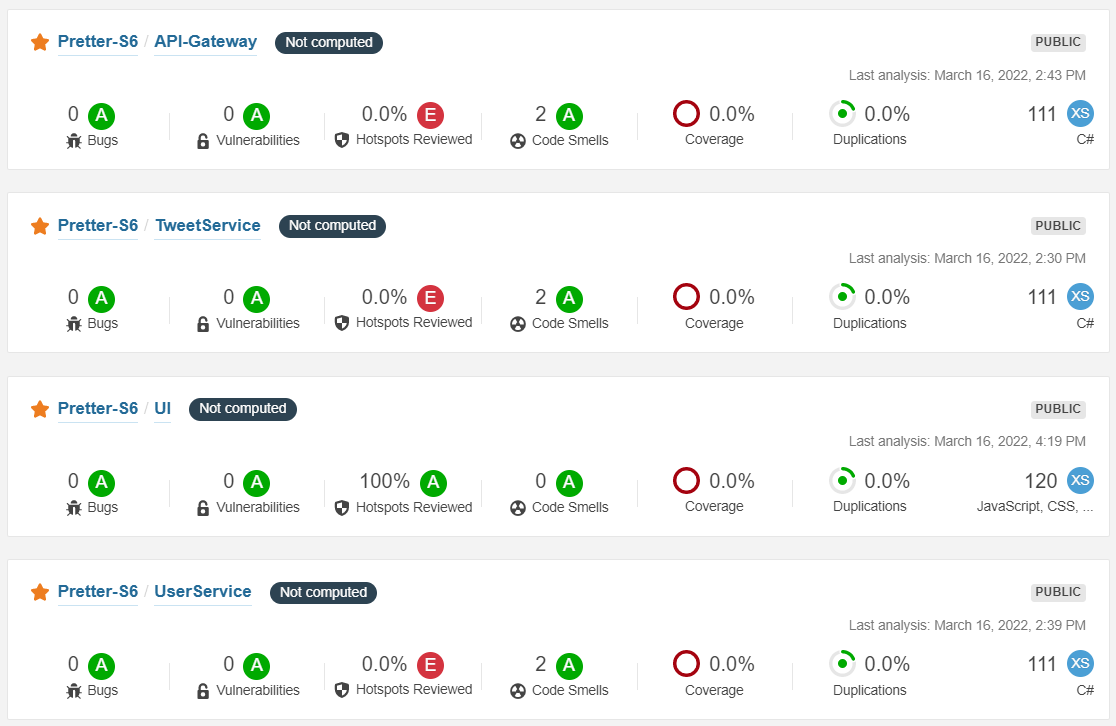
Heel mijn project is opgezet in Github, hierin hou ik ook bij welke taken nog moeten worden voltooid. Om het overzichtelijk te houden voor mezelf. Door middel van Github creëer ik voldoende flexibiliteit en onderhoudbaarheid.



The snelheid van de ontwikkeling binnen het project is afhankelijk van mijn individuele inzet en of de opzet juist is. Deze opzet (walking skeleton) bespreek ik dan ook uitvoerig met mijn docenten.

Omdat alles via CD online wordt gezet is het programma breed bruikbaar. En ook de architectuur en opzet van het project zorgen ervoor dat het een zeer bruikbaar programma wordt.

Zoals in 2.1 Enterprise Architectuur al is besproken, zijn alle repositories met code opgezet. Hierbij zijn de frontend, api-gateway en de microservices opgedeeld.



Als estimate voor total user base zou bij een beginnende situatie 1000 gebruikers een goede inschatting zijn. Vervolgens zouden hiervan ongeveer 100 een actieve gebruiker kunnen worden. Wanneer je 100 actieve gebruikers hebt dan zouden hiervan ongeveer 20 gelijktijdig de applicatie kunnen gebruiken. Hierbij zal de userservice minder worden gebruikt dan de tweetservice. De ui en api-gateway zullen het meeste worden gebruikt (highest load).

Hierbij is de keuze voor het opdelen van mijn architectuur dus gerechtvaardigd.

De gekozen technieken zijn eerder ook al verantwoord, ze zijn al breed in gebruik en zijn geschikt voor de Enterprise architectuur.

Load generation en het hierbij behouden van de juiste performance zijn erg van belang dit semester. Hiervoor maken we gebruik van kubernetes en docker. Die samen de load verdelen waar het nodig is. Er zijn ook verschillende tools die controleren of de performance op het juiste niveau blijft.

De non-functional requirements staan omschreven in hoofdstuk [1.3 Non-functional requirements](#_1.3_Non-functional_requirements).

Schaalbaarheid zal worden bereikt door de architectuur juist op te zetten. Dit voornamelijk door de applicatie op te delen in delen. Ook door het verspreiden van de load kan de applicatie makkelijk worden opgeschaald.

Performance zal voornamelijk worden gewaarborgd door het juist implementeren van technieken en het aanhouden van de juiste architectuur. Zoals gezegd is het ook mogelijk om tools te gebruiken om te controleren of de performance prima is.

Robuustheid kan ik bereiken door de applicatie veel te testen. Hierbij moet het afvangen van errors juist gaan en moet de applicatie up blijven tijdens onenigheden.

Gedistribueerde data zal worden bereikt door de data online, in delen, weg te schrijven.

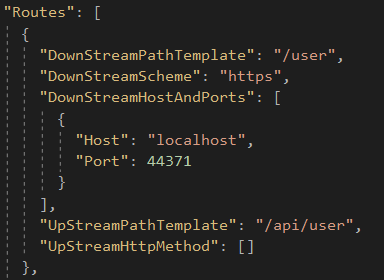
Security kan worden gewaarborgd door de juiste keuzes in de technieken en architectuur te maken. Ook kan er worden getest op verschillende zwaktes binnen een systeem.

Moet nog beginnen met het schrijven van (geautomatiseerde testen)

Sonarcloud is toegevoegd om de kwaliteit van het project te waarborgen. Omdat er nog niet veel code is geschreven is sonarcloud nog niet echt van toepassing. Wanneer er meer code staat dan zal sonarcloud aangeven wanneer er een zwakte waargenomen wordt.

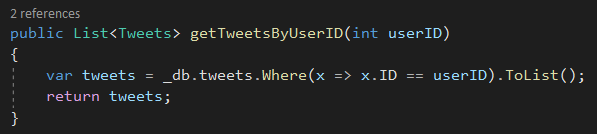
## 2.2 Sprint 2

Binnen sprint 2 ben ik bezig geweest met verscheidene dingen waaronder het aanpassen van de api-gateway naar een ocelot gateway. Hierbij wordt er geen gebruik meer gemaakt van de controller maar wordt alles geregeld via een JSON file:



Deze api-gateway werkt volledig en reroute alle incoming requests naar de juiste services.

Binnen de services ben ik bezig geweest met het aanpassen van het framework. Hier maak ik nu gebruik van het entity framework. Dat mij vooral helpt bij het ophalen en wegschrijven van data.



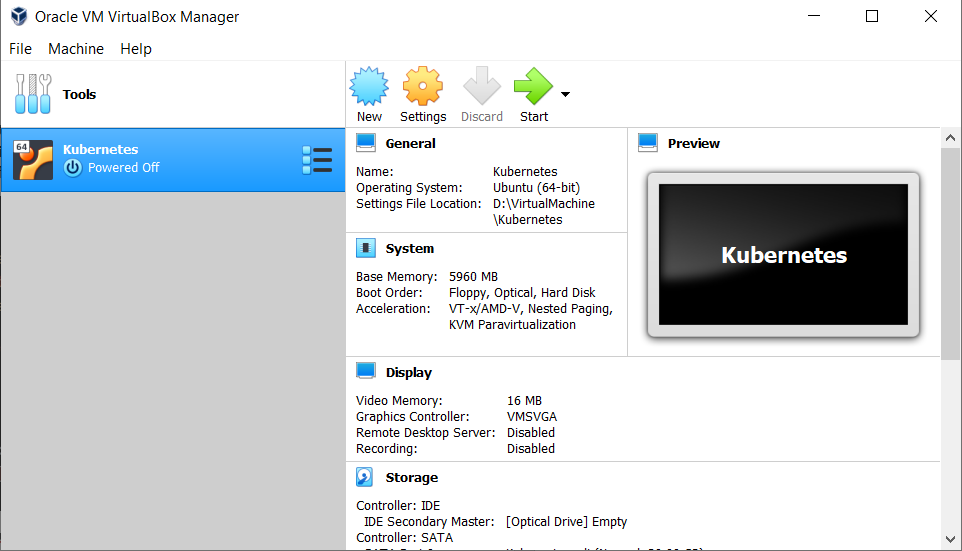
Dit entity framework werkt ook en ik ben nu vooral bezig met het implementeren van dit framework. Zodat mijn backend alle functionaliteiten bevat die ze nodig heeft.

Binnen mijn documentatie heb ik een testfase toegevoegd. Dit om de risico’s in kaart te brengen en ze waar nodig in te perken.

Deze sprint had ik graag meer werk verzet. Maar doordat ik verschillende aanpassingen binnen de structuur van mijn applicatie heb gedaan. Lijkt het alsof er minder werk is verzet. Echter zijn deze aanpassingen van belang voor de applicatie en de NFR’s van de applicatie.

## 2.3 Sprint 3

In deze sprint ben ik vooral bezig geweest met het opzetten van kubernetes. Ik heb het lokaal op een vm gezet en hierin al mijn services gehost. Hij pullt de images van docker en runt deze vervolgens op kubernetes.



Hiernaast heb ik alle documentatie bijgewerkt en online gezet in github.

Ook ben ik bezig geweest met het programmeren, zodat er nu een inlogscherm staat dat kan communiceren met de backend en de juiste methode. Hierdoor heb ik een integratie van frontend tot en met database.

Verder ben ik bezig geweest met het aanpassen van de docker images en de bijbehorende ip adressen voor de productie.

# Scenario’s

## 3.1 Scalable Architectures

**Algemeen**

Als developer, wil ik een schaalbare architectuur, zodat ik wanneer het nodig is kan upscalen, downscalen of kan switchen van instantie.

**Analyse en design**

* Binnen proftaak onderzoek gedaan naar de verschillende architectuur patterns die er zijn.
* Binnen het SAD zijn verschillende keuzes gemaakt die de schaalbaarheid van de applicatie en de architectuur waarborgen.

**Implementatie**

* Zoals in het SAD en de implementatie (Github) te zien, heb ik de microservice architectuur juist toegepast. Hierbij wordt schaalbaarheid gewaarborgd.
* Zoals in de Github repo van de API-Gateway te zien, heb ik Ocelot toegepast. Dit zorgt voor een (makkelijk) schaalbare API-Gateway.
* Zowel in Ocelot als in Kubernetes zit loadbalancing, hierbij heb ik op Kubernetes voor elke instantie 2 replicas aangemaakt. Hierdoor kan Kubernetes switchen wanneer dit nodig is. Kubernetes kent ook een dashboard waar je overzichtelijk kan zien wat de instanties doen.

**Scenario’s**

|  |  |
| --- | --- |
| **Scenario** | **Pretter** |
| Het verkeer naar de UserService wordt twee keer zo groot. | Kubernetes zal de load van de service balancen en wanneer nodig verkeer over een andere replica laten gaan. |
| De eerste replica van de TweetService is niet meer beschikbaar. | Kubernetes zal de load verschuiven naar de andere beschikbare replica. |

## 3.2 Development and Operations

**Algemeen**

Als developer, wil ik een goede DevOps omgeving, zodat ik het agile en automatiseringsproces kan ondersteunen. En om een zo kortmogelijke release tijd en hoge software kwaliteit te garanderen.

**Analyse en design**

* Het proces is ingericht volgens OTAP (ontwikkeling, test, acceptatie en automatisering). Hierbij wordt eerst de code ontwikkeld, vervolgens geautomatiseerd getest en in geval van acceptatie gedeployed.

**Implementatie**

* De DevOps omgeving is opgezet in GitHub. Hier wordt de code en de taken opgeslagen. Ook de CI/CD pipeline bevindt zich in GitHub. De pipeline test eerst de code, vervolgens controleer SonarQube de kwaliteit en eventuele bugs. En hierna wordt de code eventueel gepusht naar Docker.
* Alle delen van de applicatie zijn individueel te deployen. Ze worden ook allemaal individueel als Docker images gepusht. In de virtuele omgeving van Kubernetes worden deze images automatisch gepulled.

**Scenario’s**

|  |  |
| --- | --- |
| **Scenario** | **Pretter** |
| Er is een visuele aanpassing gedaan op de frontend. | Wanneer de push wordt ontvangen. Zullen allereerst de test worden gerund, vervolgens zal sonarcloud de code scannen en tot slot zal bij goedkeuring er een docker image gebouwd worden. Wanneer dit gebeurd dan pullt Kubernetes deze en heeft de deployment de nieuwste versie. |
| Een test van de frontend push faalt. | De gefaalde test zal ervoor zorgen dat de pipeline faalt, hierdoor zal er eerst moeten worden voldaan aan deze test. |

## 3.3 Cloud services

**Algemeen**

Als developer, wil ik services in de cloud hosten, zodat ik de voordelen van cloudservices (schaalbaarheid, eenvoudigheid, betrouwbaarheid en veiligheid) kan benutten.

**Analyse en design**

* In het SAD zijn verschillende keuzes gemaakt over waarom sommige onderdelen wel of niet in de cloud gerund worden. Zo kon sonarcloud bijvoorbeeld ook lokaal gerund worden.

**Implementatie**

* Voor het controleren van de kwaliteit van de code wordt gebruik gemaakt van de sonarcloud service. Hierbij ontvangt de pipeline een report en indien er iets niet correct is wordt de pipeline gecanceld.
* Voor development wordt gebruik gemaakt van een online database. Alle images worden gedeployed naar Kubernetes.
* Er wordt ook gebruik gemaakt van Docker hub, hierbij worden mijn Docker images naar de cloudservice van Docker gepusht. Vervolgens zijn ze online beschikbaar voor Kubernetes.

**Scenario’s**

|  |  |
| --- | --- |
| **Scenario** | **Pretter** |
| Sonarqube heeft problemen door een windows update. | Mijn applicatie heeft hier geen last van omdat het gebruik maakt van de cloud service. |
| Er is erg veel dataverkeer. | Door het gebruik van een online database is er voldoende dataverkeer mogelijk. |

## 3.4 Security by design

**Algemeen**

Als developer, wil ik security implementeren vanaf het design stadium. Zodat ik eventuele beveiligingsrisico’s kan voorkomen en niet op het einde nog alles voor security hoef te doen.

**Analyse en design**

* Gebruikte frameworks geanalyseerd en hiervan de best practices gebruikt.
* OWASP implementeren.
* Testplan opgezet in documentatie en automatisch testen toegevoegd in de pipeline. Ook test sonarcloud op eventuele veiligheidsgebreken.

**Implementatie**

* In het juist opzetten van de architectuur heb ik voor veiligheidsredenen en best practices ervoor gekozen om de API-Gateway als enige punt van aanspraak te houden. Hierdoor is de attack surface een stuk kleiner.
* In de applicatie is authenticatie toegevoegd. Hierdoor kunnen onbevoegden niet alles doen.
* Binnen de applicatie worden errors juist afgevangen. Hierbij wordt er gezorgd dat er veilig gefaald kan worden.
* Sonarcloud toont beveiligingsrisico’s.
* Wachtwoorden zijn encrypted.

**Scenario’s**

|  |  |
| --- | --- |
| **Scenario** | **Pretter** |
| Er wordt van buitenaf een endpoint van de UserService aangesproken. | Endpoints van de services zijn alleen beschikbaar en bereikbaar voor de API-Gateway. |
| Een methode krijgt een error omdat er geen verbinding met de database is. | Errors worden afgevangen, de applicatie zal de data niet kunnen ophalen maar zal niet crashen. |

## 3.5 Distributed data

**Algemeen**

Als developer, wil ik mijn data distribueren, zodat ik de grote hoeveelheid aan data en de gevoeligheid ervan kan waarborgen. Ook de beschikbaarheid van de data is hierbij een belangrijke factor.

**Analyse en design**

* In het SAD is te zien dat er een opdeling van data plaatsvind. Hierbij is ook overwogen of er gebruik moest worden gemaakt van meerdere databases. Binnen de productie is gebruik gemaakt van meerdere replica’s van de database.
* Analyse over de data regulaties waar normaliter aan voldaan moet worden gedaan.

**Implementatie**

* In productie is de data opgedeeld in een User en Tweet database. Hierbij worden in Kubernetes meerdere instanties van de database aangemaakt. Dit ook voor beschikbaarheid en schaalbaarheid.
* De data is encrypted en volgens de geanalyseerde richtlijnen opgeslagen.

**Scenario’s**

|  |  |
| --- | --- |
| **Scenario** | **Pretter** |
| Er gaat een grote hoeveelheid aan data naar de UserDB op de productie. | Dit zou geen probleem moeten zijn, aangezien er meerdere replica’s van de database zijn. Kubernetes kan verkeer hiernaar omzetten. |
| Er vindt een datalek plaats in de UserDB. | Wanneer alles volgens richtlijnen is ingericht, zal er tijdens een lek ook volgens de richtlijnen gehandeld moeten worden. Maar risico´s zijn al kleiner omdat de richtlijnen in acht zijn genomen. |

# Conclusie