Software Architecture Document (SAD)

Project: Lisk Delegates

Project team: BBB-BC

Team: Joeri Berman (2834499)

Luc Urlings (3071081)

Stijn Baltessen (3064565)

Wouter Vandenboorn (3099571)

Floris Feddema (3188256)

Davy de Haas (3099695)

Opdrachtgever: Bartosz Paszkowski

Version: 0.6

Version Date: 23 maart 2020

Status: Concept

# Document Revision

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Changes** | **Author** | **Date** |
| 0.1 | Setup for the semester 6 architecture case study | Wouter Vandenboorn | 17-02-2020 |
| 0.2 | Added Introduction and c4 models, database and api descriptions. | Wouter Vandenboorn,  Stijn Baltessen | 21-02-2020 |
| 0.3 | Document dupliceert en aangepast voor groepsproject Lisk Delegates.   * Introductie gemaakt * Architectuur design opzet gemaakt * C1 diagram schets gemaakt | Floris Feddema  Stijn Baltessen | 02-03-2020 |
| 0.4 | Document verder uitgewerkt, vertaling gedaan, Document Revisie aangepast. C1 beschrijving gemaakt, C2 opzet gemaakt. | Floris Feddema  Stijn Baltessen | 03-03-2020 |
| 0.5 | Nieuwe versie C1 en C2 uitwerken | Floris Feddema  Stijn Baltessen | 17-03-2020 |
| 0.6 | Container diagram uitgewerkt met behulp van feedback van Bartosz | Stijn Baltessen | 23-03-2020 |
| 0.7 |  | Davy de Haas, Luc Urlings, Stijn Baltessen | 14-04-2020 |
| 0.8 | Verantwoordelijkheden van iedere service, 1e versie toegevoegd | Luc Urlings | 24-04-2020 |

# Table of Contents

[**Document Revision**](#_bu2l4jqkimrr) **1**

[**Table of Contents**](#_w4z8y7a5kuom) **2**

[**Introductie**](#_8anxnj4hrgfu) **3**

[**System Context (C1)**](#_v5og0y2f4s6s) **4**

[**Container Diagram (C2)**](#_wpwo1meelztp) **5**

[**Specification of API’s**](#_ged4l8nquj62) **7**

[**Middleware**](#_gtbh74onvqfm) **8**

[Message queue](#_8l3ocfijtatv) 8

[Process verschil](#_fyixelg3ocdz) 8

[Implementatie](#_mccqnlm2dvww) 9

[**Authenticatie (JWT Authenticatie)**](#_w3ca9631t95g) **10**

[**References**](#_45zouvuimj0b) **11**

# 

# Introductie

In dit document zal het de architectuur van het Lisk Delegates project worden uitgelegd en visueel weergegeven. Dit zal gedaan worden door middel van diagrammen die dit op verschillende niveaus weergeven.

Het doel van het project is om een marktplaats te ontwikkelingen voor Lisk sidechains.

Hierdoor kunnen sidechain applicaties worden gemaakt en hoeven mensen niet zelf de sidechain op delegates te hosten. Dit maakt het veel aantrekkelijker om Lisk te gebruiken voor deze applicaties.

Ook voor mensen met een delegate account is het aantrekkelijk om hier gebruik van te maken. Op de Lisk mainchain mogen alleen de 101 hoogste delegates blocks signen. De rest van de delegates verdient hier niks aan, wanneer er sidechains zijn en deze delegates hier, voor een fee, op kunnen signen hebben alle twee de partijen er voordeel van. De delegates kunnen signen en krijgen een fee. De eigenaren van de applicaties hebben delegates die de sidechain bijhouden.

# 

# System Context (C1)

In dit hoofdstuk wordt het product op het hoogste level van abstractie weergeven en uitgelegd. Ook worden de relaties van gebruikers en externe systemen toegelicht.

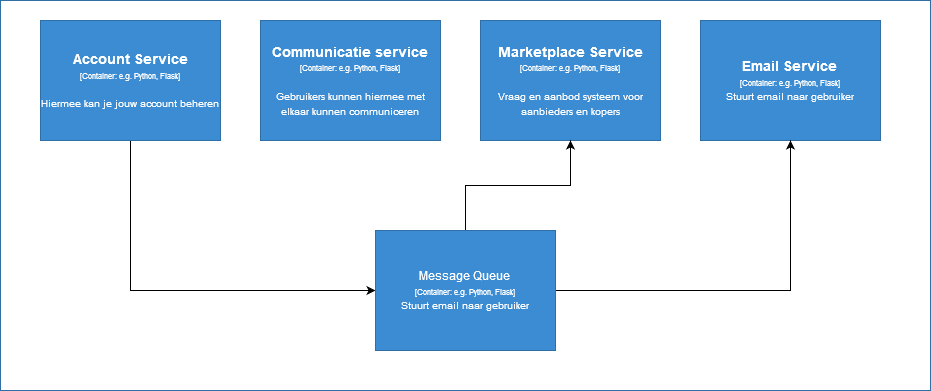
|  |
| --- |
|  |
| *figuur.1.1 C1 model* |

Ons systeem is de Delegate Market. Het doel van dit systeem is om de vraag van dApp eigenaren om delegates en wederzijds te beantwoorden in een veilige en stabiele omgeving.   
Hier zijn de verschillende gebruikers te zien. Een DApp eigenaar kan een DApp aanbieden en chatten met delegate aanbieders. Een delegate eigenaar kan een delegate aanbieden en chatten met DApp aanbieders. Wanneer een gebruiker niet is ingelogd kan deze alleen de aanbiedingen inzien.

# 

# Container Diagram (C2)

Hierboven staan de verschillende gebruikers met die gebruik maken van de frontend. Deze frontend zal op zijn beurt de de verschillende backend services aan spreken. Elke backend service behalve de email service heeft een eigen datastore om gegevens in op te slaan.



Hierboven staan alle services genoteerd samen met de Message Queue. Hier is de communicatie van en naar deze MQ te zien. De accountservice kan berichten op de queue plaatsen en de marketplace of email service zal deze ontvangen, dit is afhankelijk van op welk kanaal het bericht is geplaatst.

# 

# Specification of API’s

Er is een [klein document](https://docs.google.com/document/d/1hiugPMmvub7cUsoA5_NLRB-uUW-TU7lwTUCa-og2DmM/edit?usp=sharing) geschreven over hoe het geïnstalleerd wordt op .NET Core

# Middleware

Tijdens de study case: Messaging, is de term middleware gevallen. Middleware is software tussen services staat of integrated kan zijn in services om de kwaliteit en efficiëntie van het gehele systeem te bevorderen.

Middleware behaalt geen business goals van een applicatie maar zorgt ervoor dat de structuur van het systeem beter verloopt. Business value van middleware is dus de snelheid en kwaliteit waarborgen.

Voor microservices is juiste middleware essentieel om de services goed uit elkaar te kunnen trekken.

In dit hoofdstuk kun je vinden welke middleware technieken we in ons project hebben toegepast. En waarom deze een groot verschil kunnen maken op de performance van een systeem op enterprise niveau.

## Message queue

Om ervoor te zorgen dat asynchrone communicatie tussen verschillende services juist kan verlopen kan je een message queue (mq) gebruiken, hierdoor kan je een bericht sturen naar de message queue met juiste parameter.   
  
Je moet een service zo inrichten dat het messages van de mq kan oppakken en verwerken.

### Process verschil

Message queue



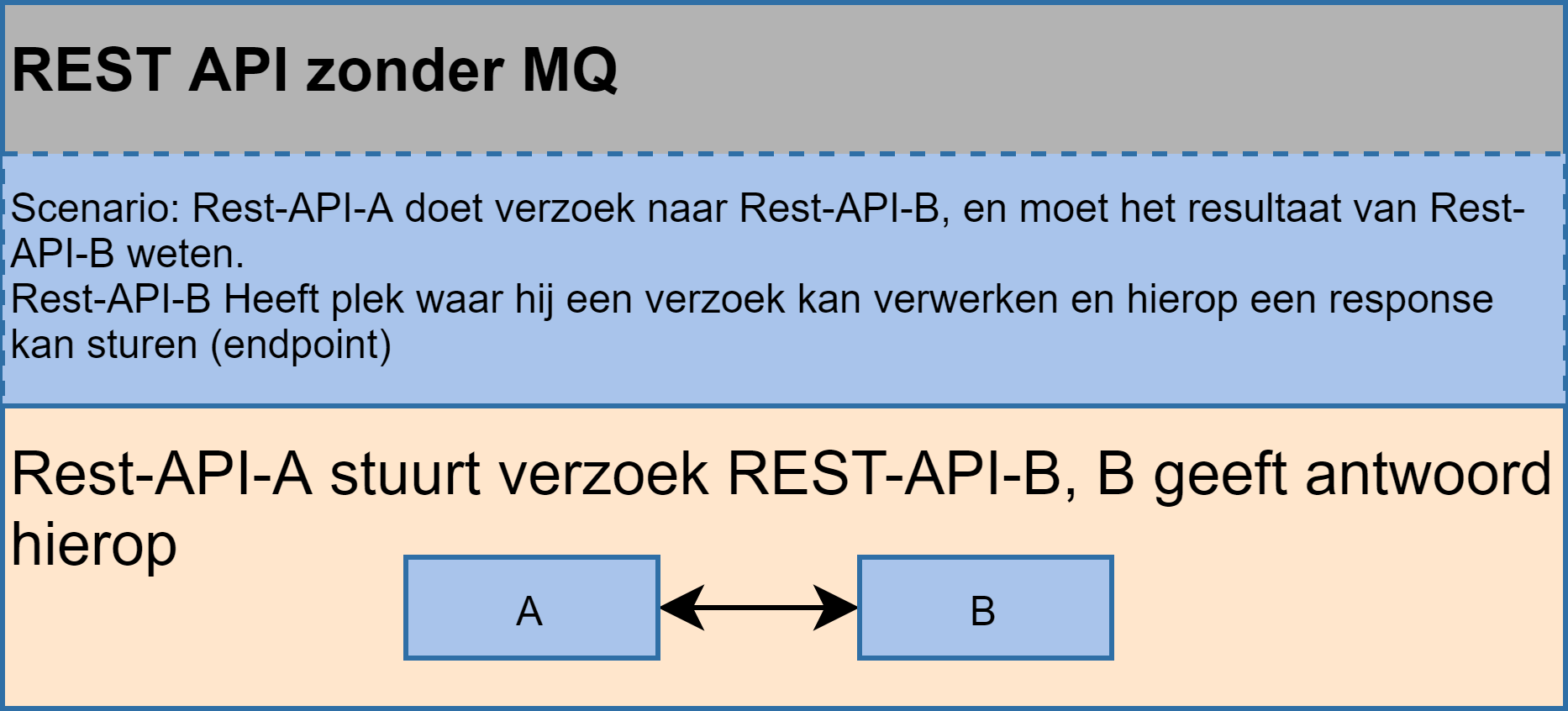
Kenmerken:

* async

Handig voor:

* data dat geduplicate wordt in andere services door te updaten via een vaste mq.
* verzoeken waarvan resultaat niet belangrijk is doorvoeren.
* Geen dataverlies wanneer de ontvanger service offline is.

Rest Api communicatie



Kenmerken:

* sync

Handig voor:

* verzoeken waar resultaat belangrijk is voor de verzoeker

### Implementatie

Om rabbit queue te implementeren hebben we RabbitMq opgezet op een remote ubuntu server. Vervolgens hebben we een “exchange” aangemaakt, een exchange is een weg waar berichten belanden. In de exchange heb je verschillende queues.

Een applicatie verbindt met een exchange en kan dan bepalen naar welke queues die luistert. Als verstuurder van bericht in een exchange geef je de queue aan.

Om snel te differentiëren hebben we een header nog verder toegevoegd waarin je methodes in de code kan indelen.



* “AuthenticationSerivce” is de queue,
* “RegisterUser” is header waarop we diferrentiëren



Voorbeeld implementatie van messagehandler van queue.

## Authenticatie (JWT Authenticatie)

De authenticatie in onze applicatie wordt gedaan door middel van JWT. Na succesvol in te loggen wordt een token gegenereerd en teruggegeven aan de gebruiker. In deze token staan de volgende encrypted gegevens:

* Het id van de gebruiker
* Verloop datum van token.

Deze gegevens worden ge-encrypt met een secret, dit secret zorgt ervoor dat de authenticatie op andere services ook kan worden gebruikt mits ze dezelfde secret gebruiken.

In de token hoe we hem nu hebben ingericht, kan je ervan uitgaan dat de userId klopt wanneer de token is geverifieerd. Dit zorgt ervoor dat je gebruikers kan identificeren.

# References

* <https://c4model.com/>
* <https://fhict.instructure.com/>
* <https://swagger.io/>