Terence Koch

Fontys

2019

Game Reviewer

DPI6 individuele opdracht

# Introductie

In dit document worden de requirements en specificaties van het project Game Reviewer concreet voorgesteld. Dit project staat in thema van het vak DPI6 dat door het lespakket van Fontys aangeboden wordt.

De requirements worden beschreven en de architectuur wordt in detail besproken. De keuze voor JMS en ActiveMQ wordt ook toegelicht.

Inhoud

[Introductie 1](#_Toc4611671)

[Product A: Analyse Document 2](#_Toc4611672)

[Requirements 2](#_Toc4611673)

[Non-functionele requirements 2](#_Toc4611674)

[Leerdoelen 2](#_Toc4611675)

[Architectuur 3](#_Toc4611676)

[Product B: Keuze messaging 4](#_Toc4611677)

[Algemeen JMS en message services 4](#_Toc4611678)

[Alternatieven voor JMS 4](#_Toc4611679)

[Voor- en nadelen JMS en alternatieven 4](#_Toc4611680)

[RabbitMQ vs ActiveMQ vs Kafka 5](#_Toc4611681)

[Grote gebruikers 5](#_Toc4611682)

[Topics vs Queues 5](#_Toc4611683)

[Beveiliging 6](#_Toc4611684)

[Schaalbaarheid 6](#_Toc4611685)

[Conclusie 6](#_Toc4611686)

[Bronnen 7](#_Toc4611687)

[Product C: Oplevering 8](#_Toc4611688)

[Gebruikte tools 8](#_Toc4611689)

[Werking 9](#_Toc4611690)

[Patronen 9](#_Toc4611691)

[Veel bezochte bronnen 10](#_Toc4611692)

# Product A: Analyse Document

## Requirements

1. Als gebruiker moet ik de naam van een spel kunnen opgeven zodat ik via verschillende bronnen de gegevens over het spel overzichtelijk kan inzien
2. Minimaal 2 bronnen moeten toegankelijk zijn.

## Non-functionele requirements

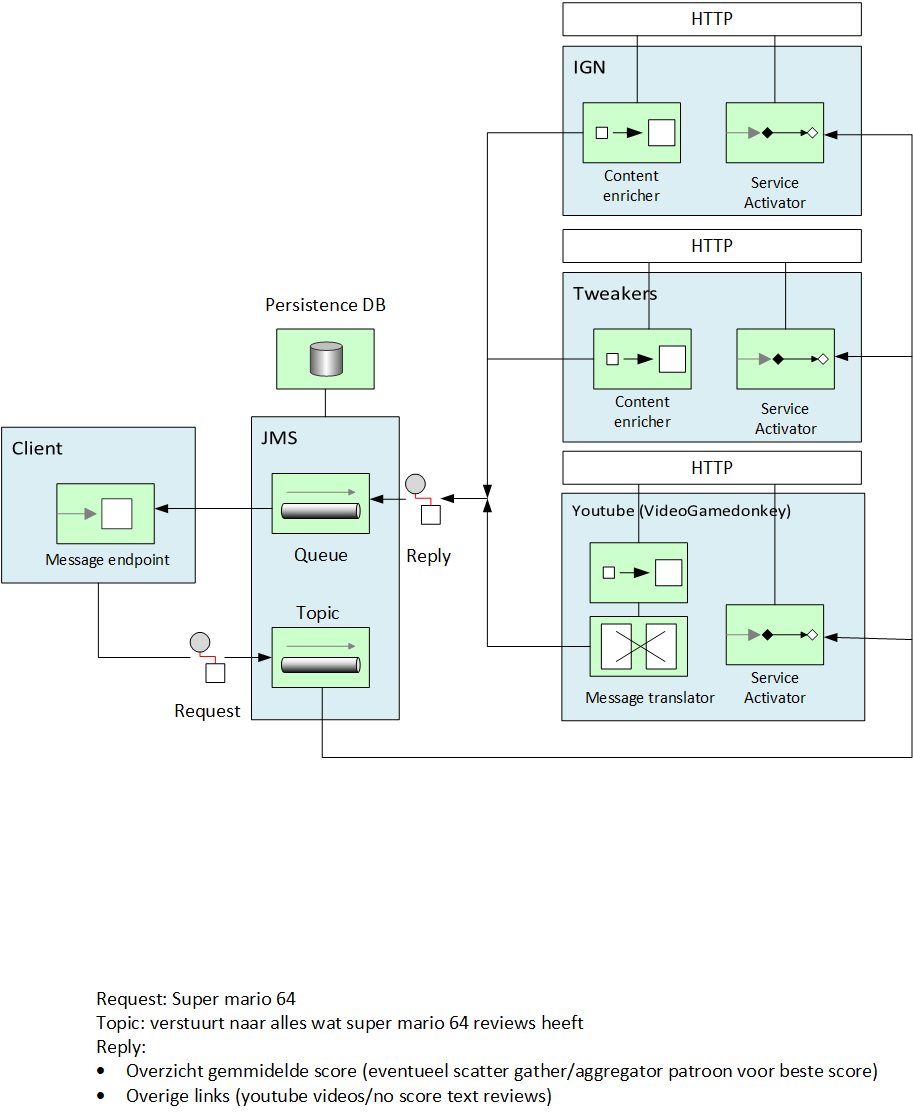
1. Het systeem moet gebruik maken van JMS-topics en queues.
2. Het systeem moet gebruik maken van Java beans.
3. De architectuur bevat de onderdelen Endpoints, Channels, Messages, Transformations en Routing.

## Leerdoelen

1. Bruikbaar ontwerp kunnen maken voor asynchrone messaging tussen applicaties.
2. Een goed advies kunnen geven voor te gebruiken (JMS-messaging) software voor communicatie (met messaging) tussen applicaties.
3. Functionele, onderhoudbare software kunnen ontwikkelen voor asynchrone messaging tussen applicaties.

|  |  |
| --- | --- |
| Activiteiten (eindbeoordeling voldoende): | * JMS * Topics en Queues * Meerdere endpoints - minimaal 2 bronnen en een client * Documentatie en architectuur in orde * System output interface |
| Extra activiteiten  (Eindbeoordeling goed): | * Nieuwe patronen (message filters, message translator, publish/subscribe, …) * Web/applicatie interface |
| Naast extra activiteiten nieuw studiemateriaal ontwikkelen  (Eindbeoordeling excellent): | * Nog extra patronen (competing consumers, …) * Beveiliging |

## Architectuur



Er zijn 2 type berichten: Request en Reply. Een Request bestaat uit een query (bv: Super Mario 64). Deze wordt doorgegeven aan het topic-kanaal die weet welke bronnen informatie over deze game heeft (bv. er is 1 video review en 3 geschreven review met eindcijfer/verdict). Via data mining en eventuele API’s zullen de gegevens op een eerlijke manier de resultaten verzamelen. De bronnen worden ook telkens vermeld. Met een message translator en met de content enricher worden alle gegevens goed vormgegeven. Aangezien de het loop veel tijd in beslag neemt (scrapen is duur) worden bepaalde replies opgeslagen zodat deze op een later tijdstip sneller opgeroepen kan worden. De reply bestaat uit een id, gemiddelde score (indien van toepassing), aantal bekeken bronnen, links naar de verschillende bronnen en links naar video reviews (indien van toepassing).

Er is in totaal sprake van 1 client applicatie en verschillende “bron-applicaties”. De bron applicaties zijn YouTube (specifieke kanalen), Polygon, IGN, MetaCritics, Tweakers enzovoort. Elke bron applicatie zal net iets anders zijn afhankelijk van hoe de data verwerkt zal worden.

# Product B: Keuze messaging

## Algemeen JMS en message services

JMS is een mechanisme dat asynchrone requests behandeld. ActiveMQ is een JMS-broker implementatie en staat in als een programmeer omgeving die eenvoudig en efficiënt omgaat met de techniek.

Omdat JMS asynchroon werkt is het grootste voordeel om lange requests hiermee te behandelen. Een andere mogelijkheid is dat verschillende partijen interesses hebben in de request. Dit stelt ook elke partij (eventueel via eigen clients) in staat om de request op een eigen manier te behandelen. Daarnaast is het ook een eenvoudig systeem om berichten tussen verschillende programma’s te sturen.

*Voorbeeld 1:* In kader van dit project zou de request van de gebruiker (naam van een game) door de verschillende bron clients (YouTube, IGN of metacritics) op een unieke wijze behandeld worden zodat elke bron client een verwacht antwoord geeft.

*Voorbeeld 2:* In de dpi-practicum kon er een loonvraag gedaan worden en met behulp van regels kregen verschillende banken, afhankelijk van de loonaanvraag, een bericht waarop ze antwoord konden geven. Dit soort aanvragen duren lang en de omdat het asynchroon is opgezet is, zijn lange processen geen probleem voor de clients.

## Alternatieven voor JMS en andere brokers

Er zijn nog andere messaging protocollen en service waaronder AMQP (.Net messaging service) en MQTT (voornamelijk voor IOT gebruikt). In een andere categorie heb je Kafka wat een message streaming service is (message broker en message service tegelijk). Kafka is dus een broker en service tegelijk maar wordt in het hoofdstuk hieronder verder besproken.

Web sockets zijn niet gericht op berichten sturen. De JMS-implementatie doet onder water een hoop om dubbele berichten slim af te handelen. En werkt goed asynchroon tussen gedistribueerde programma’s. Web sockets zijn vaak iets meer error-vatbaar en werken niet zo eenvoudig asynchroon. Web sockets zijn niet java-exclusief, beveiligd via een SOAP (ook als WS-security gekend).

Andere jms brokers zijn IBM MQ, MS MQ, MQTT, XMPP, AMQP, ZeroMQ, RocketMQ, Qpid, Artemis, NSQ.

## Voor- en nadelen JMS en alternatieven

|  |  |  |
| --- | --- | --- |
|  | **Voordelen** | **Uitleg** |
| **JMS** | **Asynchroon** Betrouwbaarheid Losse koppelingen Schaalbaar | **Moet niet wachten op antwoorden** Bijvoorbeeld once-and-only-once delivery Bijvoorbeeld met database of totaal andere systemen Beloofd en aangetoond in verschillende use cases |
| **MQTT** | Betrouwbaar Schaalbaar Beveiliging Flexibele publish/subscribe | Met extra focus voor IoT Quality of Service-uitbreidingen Met extra focus voor IoT |

## RabbitMQ vs ActiveMQ vs Kafka

Message technologieën zoals RabbitMQ, ActiveMQ en Kafka stellen de gebruiker in staat om asynchroon te communiceren van losgekoppelde processen. Deze worden elke gebruikt om schaalbaar, veilig en betrouwbaar berichten te verzenden tussen losgekoppelde systemen. Hun doel is hetzelfde maar de manier waarop verschilt. Hieronder een korte vergelijking.

|  |  |  |
| --- | --- | --- |
| **RabbitMQ** | **Kafka** | **ActiveMQ** |
| - Erlang | + Java | + Java |
| + Eenvoudig te monitoren | + High throughput | + Snel opzetten |
| + Eenvoudig te begrijpen | + Broker en message service |  |

Hieronder enkele puntjes toegelicht

|  |  |  |
| --- | --- | --- |
|  | **Voordelen** | **Uitleg** |
| **RabbitMQ** | Flexible Routing Multiple client Goede interface | Kan routing eenvoudig aanpassen Werkt met JMS en andere message services Zorgt voor duidelijke monitoring (tracing, plugins, |
| **Kafka** | Goede doorgang Geen message broker  Lagere performance | Kan snel en efficiënt grote volumes data doorsturen Kafka kan werken als message brokers en service (geen nood aan JMS dus) Grote berichten worden standaard niet gecompresst |

## Grote gebruikers

*ActiveMQ* en JMS worden door verschillende gebruikers goed ervaren waaronder in grote industrieën. STG Technologies heeft de technologie gebruikt in de luchtvaartindustrie, University of Washington gebruikt deze in verschillende gedistribueerde applicaties in hun berichten laag. Ook grote websites zoals wework gebruiken ActiveMQ en JMS. ActiveMQ is ook geïntegreerd in Amazon MQ. Dit alles geeft al snel het beeld dat de technieken goed bruikbaar zijn voor groot en klein.

*RabbitMQ*wordt gebruikt door Reddit, 9GAG, Vine Labs, MIT, BitBucket, Sendgrid en andere grote bedrijven.

*Kafka*wordt gebruikt door Uber, Spotify, Shopify, WePay en andere grote instanties.

## Topics vs Queues

Topics en queues zijn kanalen die het bericht (de request) uitdelen aan de juiste ontvanger. De keuze in dit project is gedaan voor topics. Deze techniek maakt gebruik van verschillende “subscribers” op een bepaalde “topic”.

*Voorbeeld 1:* 3 bronnen hebben informatie van een bepaalde game. Als er een request wordt gedaan over deze game, dan zouden de bronnen een antwoord kunnen sturen.

*Voorbeeld 2:* Queues kunnen bij de proftaak rekeningrijden gebruikt worden om gestolen auto’s door te sturen naar het politiesysteem. Alle gestolen auto’s kunnen zo eenvoudig getraceerd worden door de politie.

## Beveiliging

ActiveMQ beveiligt het systeem met behulp van JAAS (Java Authentication and Authorization Service). Via configuratie bestanden kun je een wachtwoord en gebruikersnaam opzetten en via de methode createConnection kun je authentiseren. Met behulp van ACL (Acces control list) kun je permissies uitdelen binnen ActiveMQ voor de geautoriseerde gebruikers.

## Schaalbaarheid

Er zijn verschillende manieren hoe JMS en ActiveMQ gebruikt kunnen worden om meerdere berichten tegelijk te behandelen of om grote hoeveelheden berichten te beheren. Een enterprise integration pattern is “competing consumers”.

ActiveMQ en JMS worden door verschillende gebruikers goed ervaren waaronder in grote industrieën. STG Technologies heeft de technologie gebruikt in de luchtvaartindustrie, University of Washington gebruikt deze in verschillende gedistribueerde applicaties in hun berichten laag. Ook grote websites zoals wework gebruiken ActiveMQ en JMS. Dit geeft al snel het beeld dat de technieken goed bruikbaar zijn voor groot en klein.

## Conclusie

JMS is kent veel voordelen bij het opzetten van de communicatie laag in gedistribueerde systemen:

* Berichten (queues) kunnen gebruikt worden om asynchroon requests te behandelen.
* Opdrachten die veel tijd in beslag nemen kunnen bijgehouden worden.
* Berichten kunnen naar verschillende systemen verstuurd worden. Ook als de systemen helemaal los van elkaar staan.
* De technieken bieden goede schaalbaarheid en beveiliging. Verschillende grote bedrijven gebruiken deze technieken.

Voor grote oplossingen zijn RabbitMQ en Kafka bekender dan ActiveMQ. De non-functional requirements van de game reviewer moet JMS gebruiken dus is Kafka niet de beste oplossing. RabbitMQ werkt op erlang/python en er moet gebruikt worden van java in dit project. De opgedane ervaring en non-functionele requirements maken ActiveMQ dus een logische oplossing. Wegens de integratie van JMS in payara is deze message queue (gekend als openMQ) ook een goede oplossing.

Hoofdzaken die gecontroleerd moeten worden bij het kiezen van een passend message queue zijn:

* Lag: omgang met grote aantalen data
* Management: hoe monitoren

## Bronnen

<https://www.enterpriseintegrationpatterns.com/>

<https://discovery.hgdata.com/product/apache-activemq>

<http://activemq.apache.org/>

<https://en.wikipedia.org/wiki/Java_Authentication_and_Authorization_Service>

<https://stackoverflow.com/questions/5576415/jms-topic-vs-queues>

<https://stackoverflow.com/questions/14152648/what-is-the-responsibility-of-a-jms-broker-in-the-jms-eco-system>

<https://stackshare.io/stackups/activemq-vs-kafka-vs-rabbitmq>

<https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/>

<https://www.ct.nl/achtergrond/iot-protocol-mqtt-betrouwbaar-data/>

<https://www.oreilly.com/library/view/analytics-for-the/9781787120730/db65d957-cf17-459c-a203-4b8234a14261.xhtml>

<https://www.rabbitmq.com/features.html>

<https://dzone.com/articles/what-activemq>

<https://stackoverflow.com/questions/855210/jms-vs-webservices>

<https://blogs.oracle.com/theaquarium/jms-over-websocket>

<https://medium.com/linagora-engineering/how-to-choose-a-message-queue-247dde46e66c>

# Product C: Oplevering

In dit hoofdstuk wordt de oplevering van het programma toegelicht.

Een demo filmpje is hier beschikbaar: <https://youtu.be/uhkr03vw8hQ> (0:33).

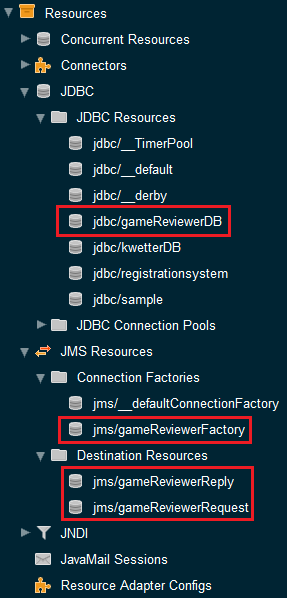
## Gebruikte tools

De hele applicatie is gemaakt met behulp van services uit Payara. Persistentie, de webapplicatie en de JMS-messaging gebeuren hiermee.

Binnen Payara is een mySQL8 database met JNDI-naam: “jdbc/gameReviewerDB” aangemaakt om resultaten van query’s in op te slaan. Dit zorgt ervoor dat een gebruiker niet telkens een query moet invoeren en dan moet wachten op een antwoord.

Er is een JMS connection factory binnen Payara opgezet genaamd “jms/gameReviewerFactory”. Deze is gelinkt aan 2 “destination resources” (zoals deze in Payara genoemd worden). Deze 2 heten “jms/gameReviewerReply” en “jms/gameReviewerRequest”. Reply is een jms-queue en en request is een jms-topic. De benaming hiervan kan telkens veranderd worden in de message driven beans van de client en de verschillende deelprogramma’s.

Payara werd gebruikt om de webapplicaties en database in te zetten. De JMS-functies kreeg ik ook snel aan de praat. Er is daarom gekozen om ook de JMS binnen Payara te houden en er is verder geen gebruik gemaakt van ActiveMQ of een andere broker.



Figuur : Extra settings in Payara

Om het data van het internet te formateren is jsoup gebruikt. De interface van de webapplicatie is met JSF en PrimeFaces gemaakt. Github is gebruikt als versiebeheer. Iedere applicatie heeft een eigen branch.

## Werking

1. Gebruiker voert een query in, sessionbean wordt aangeroepen (btnPressed methode in SessionBean.java binnen de client applicatie).
2. De query wordt naar een request queue gestuurd.
3. De bron-applicaties (IGN, MetaCritic) ontvangen het bericht (in de message driven bean).
4. De bron-applicaties begin het internet te doorzoeken en met de query de benodigde informatie te verzamelen (in LookupBean.java).
5. Zodra de informatie verzameld is wordt het antwoord op de reply queue gezet.
6. Zodra de client het antwoord ontvangen heeft wordt deze toegevoegd aan een lijst (lookupResults).
7. De client voegt de resultaten toe aan de webpagina en database.

Hieronder de request en reply messages:

|  |  |
| --- | --- |
| **RequestMessage** | **ReplyMessage** |
| Query: String | Score: Double  Link: String  Source: String  (JMSCorrelationId) |

## Patronen

*Message Dispatcher:* De topic wordt aangemaakt in MessageDispatcher (van de client). Deze heeft de naam “jms/gameReviewerRequest”. Iedere bron applicatie luistert naar deze topic. Elk werkt asynchroon met behulp van de @Asynchronous methode “postToJMS” (nieuwe thread).

*Service Activator:* Zodra een request van de client naar de topic gestuurd wordt, dan worden de bron-applicaties (MDB) geactiveerd. Dit gebeurt via de @ActivationConfigProperty annotatie. Hierin worden de topics en id’s gedefinieerd waar de bron applicatie naar luistert. De bron applicaties luisteren hier dus naar een request – in dit geval een query dat via de client webapplicatie is opgegeven en voeren vervolgens hun service uit.

*Content enricher:* Omdat een query op zich niet alle data bevat wordt er gebruik gemaakt van jsoup om de gegevens van websites te halen. Afhankelijk van de bron applicatie wordt er data opgehaald van verschillende sites en asynchroon verwerkt. Deze reply wordt met behulp van json terug naar de client verzonden.

*Message translator:* Deze wordt gebruikt om mediafiles om te zetten naar een tekstformaat wat zoals de andere berichten terug verzonden kan worden.

*Scatter-gather:* Verschillende ontvangers (IGN, Metacritic, …) zetten het bericht om naar 1 vorm.

*Publish-subscribe:* De ontvangers zijn gesubscribeerd aan een topic (jms/gameReviewerRequest) en de verzender stuurt berichten naar deze topic (publish).

*Request/Reply:* Client stuurt uniek sessionId mee -> reply zet sessionId als correlationId -> client haalt berichten op waar zijn sessie gelinkt is aan dit sessionId. Op deze manier krijgt iedere sessie (client) enkel zijn eigen resultaten terug. Omdat JSF niet een tab als nieuwe client beschouwd wordt er gebruik gemaakt van de localhost link en de directe ip link (127.0.0.1/gameReviewer) om 2 clients te simuleren.

## Veel bezochte bronnen

<https://www.enterpriseintegrationpatterns.com>

<https://blog.payara.fish/connecting-to-activemq-with-payara-server>

<http://theopentutorials.com/content/tutorials/java-ee/ejb3/mdb/>

<https://www.tutorialspoint.com/jsf/index.htm>

<https://docs.oracle.com/javaee/7/api/index.html>

<https://jsoup.org/>

<https://developer.jboss.org>

<https://stackoverflow.com/>

(Reparatie bronnen)

<https://www.enterpriseintegrationpatterns.com/patterns/messaging/RequestReplyJmsExample.html>

<https://stackoverflow.com/questions/23342116/ibm-websphere-mq-request-reply-scenario/23359217#23359217>

Zoekresultaten