27-3-2020

Max Neerken

HAN-AIM

Spotitube

Opleverdocument

Inhoudsopgave

[Inleiding 2](#_Toc35953591)

[1. Package diagram 3](#_Toc35953592)

[2. Deployment diagram 4](#_Toc35953593)

[3. Design keuzes 5](#_Toc35953594)

[Conclusie 6](#_Toc35953595)

# Inleiding

Dit opleverdocument beschrijft het opgeleverde informatiesysteem van de spotitube opdracht voor DEA. Voor deze opdracht moest er een backend gerealiseerd worden voor een bestaande applicatie. De eisen van de backend staan beschreven op <https://github.com/HANICA-DEA/spotitube>.

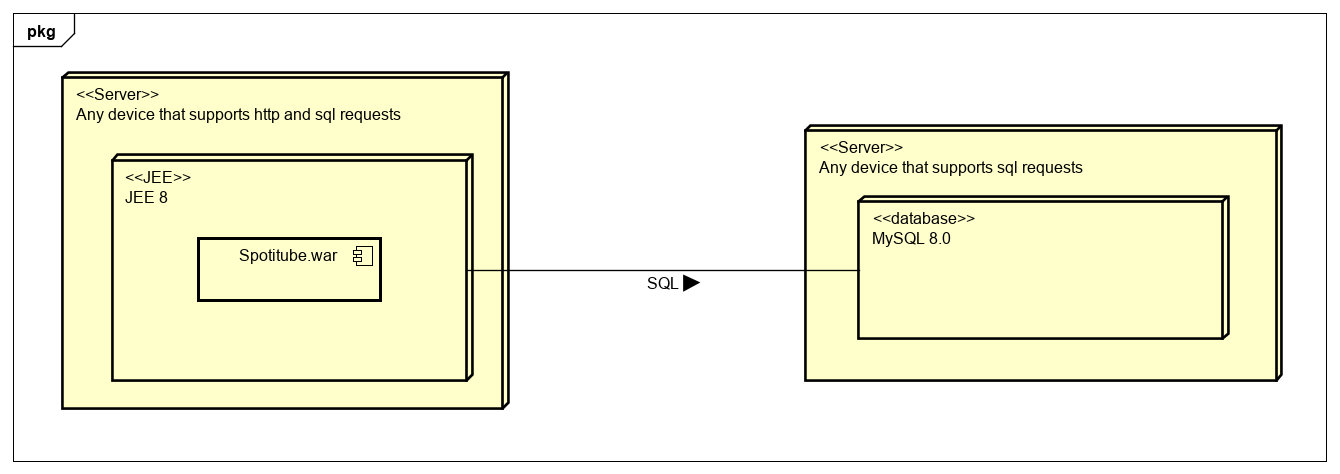
Eerst wordt er ingegaan over de package structuur van de applicatie. Vervolgens wordt er getoond hoe de applicatie juist gedeployed kan worden en tot slot worden er design keuzes toegelicht. s

# 1. Package diagram

*Bron 2: Deployment diagram*

# 2. Deployment diagram

Om de applicatie te deployen, moet er bepaalde hardware en software beschikbaar zijn. Deze hardware en software vereisten worden weergegeven in het onderstaande deployment diagram. (Bron 2)



*Bron 2: Deployment diagram*

3. Design keuzes

Bij het maken van de opdracht zijn er problemen opgedoken die wat denkwerk vereisten om op te lossen. In dit hoofdstuk worden een aantal van deze problemen, met hun oplossing genoemd.

In de datasource laag was er veel code duplicatie bij het aanmaken van prepared statements. Dezelfde code werd voor elke query gebruikt, maar dan met kleine variabelen. Om code duplicatie te verminderen heb ik het builder pattern toegepast voor het aanmaken van prepared statements. Dit heeft als extra effect dat er code uit de data access objects is gehaald, die nu wel getest kon worden.

Ook wordt veiligheid van de applicatie extra gehandhaafd door het gebruik van stored procedures. Aangezien de applicatie alleen deze stored procedures kan aanroepen is er meer controle over handelingen naar de database. Dit heeft als bijgevoegd effect dat er business logica naar de database kon worden verplaatst, zoals het controleren of een token juist is, of het controleren of iemand de eigenaar is van een playlist. Hierdoor kunnen toekomstige applicaties makkelijk dezelfde logica gebruiken.

Verder heb ik generics gebruikt bij de interface voor de datamappers, omdat al deze implementaties precies hetzelfde doen, maar dan voor een ander type object.

Tot slot worden DTO’s gebruikt in de datasource en service laag in plaats van domein objecten. De reden hiervoor is dat er sprake is van een zwak domein en dat de domein objecten niet anders zouden zijn dan de DTO’s. Gezien de aard van de applicatie leek het niet waarschijnlijk dat er later nog domein logica toe zal worden gevoegd die niet in de database of service laag kan worden verwerkt. Hierdoor was het gebruik van DTO’s de logische keuze.  
  
Hiernaast wil ik nog melden dat de stored procedures van de database niet zijn geünit test. De reden hiervoor is dat we van school geen manier hebben gekregen om mySQL databases te unit testen. Ik heb nog gekeken of er een framework was dat ik kon gebruiken en snel genoeg kon leren, maar al gouw bleek dat dit ingewikkelder was dan het leek. Hierom – en omdat het waarschijnlijk buiten de scope van de opdracht licht – is de database alleen met end to end tests getest.

# Conclusie

Voor deze opdracht moest er een backend gerealiseerd worden voor een bestaande applicatie. De gevraagde documentatie is aanwezig. Verder is aan alle eisen voldaan en dat is bewezen door middel van unit tests.