Samenvatting

Inhoudsopgave

Inhoudsopgave

[Voorwoord 1](#_Toc528323564)

[1 Inleiding 1](#_Toc528323565)

[2 Theoretisch kader 1](#_Toc528323566)

[3 Wat is de meest geschikte vorm van caching voor Java EE-applicaties die gebruik maken van REST services? 2](#_Toc528323567)

[4 Hoe verhouden beschikbare caching bibliotheken voor Java-applicaties zich tot elkaar? 3](#_Toc528323568)

# Voorwoord

todo

# 1 Inleiding

De doelstelling van dit onderzoek is het vervangen van een deel van de bestaande Spotitube applicatie door een alternatieve technologie.

Uit de genoemde doelstelling is de hoofdvraag, “hoe kunnen we caching goed inzetten binnen een Java REST API applicatie?” voortgekomen.

Die hoofdvraag is op zichzelf in te delen in enkele deelvragen, dit zijn:

1. Wat is de meest geschikte vorm van caching voor Java EE-applicaties die gebruik maken van REST services?
2. Hoe verhouden beschikbare caching bibliotheken voor Java-applicaties zich tot elkaar?
3. Hoe is één van gevonden bibliotheken te integreren binnen Spotitube?

Om tot een uiteindelijke aanbeveling te komen voor de opdrachtgever, wordt er gebruik gemaakt van de onderzoeksmethoden behorende tot het DOT framework (Fontys, 2018), literaire studie, beschikbare product analyse en prototypering.

Voor het komen tot een conclusie op de eerste deelvraag wordt de literaire studie toegepast. Vervolgens wordt er een analyse van beschikbare producten uitgevoerd om een geschikte bibliotheek te vinden, die te integreren is binnen het huidige systeem. Nadat daarmee een antwoord is gegeven op de tweede deelvraag, wordt geprobeerd de geselecteerde bibliotheek daadwerkelijk te integreren in de applicatie doormiddel van prototypering.

# 2 Theoretisch kader

Caching is het tijdelijk opslaan van recentelijk gebruikte informatie (Citrix, 2018) of zoals door Tawara (2017) concreter gesteld “Een cache is een geheugenbuffer die gebruikt wordt om tijdelijk frequent opgevraagde data op te slaan”, hierdoor kan een systeem sneller toegang verkrijgen tot opgevraagde informatie.

Door toepassing van caching wordt performance verbeterd doordat langzamer communicatieverkeer wordt vervangen door een communicatiestrategie met hoger aantal IOPS (Input/Output operations per second). Iets soortgelijks is te bereiken door het vervangen of verbeteren van de hardware architectuur, maar dit verhoogt de kosten en verbeterd de latency niet in die mate als caching dit doet (AWS,2018).

Maar buiten de performanceverbetering en mogelijke ontlasting van systeemcomponenten door minder bevragingen heeft caching nog een ander nut. Uit onderzoek blijkt dat 53% van de gebruikers met mobiele devices, een app die over het web communiceert verlaat als de laadtijd meer dan 3 seconden bedraagt (Babich, 2018). Uit het oogpunt van user experience wordt zelfs aangeraden laadtijden van een verzoek onder de 100 milliseconden te houden (Basques, Kearney, Miller & Osmani, 2018). Dat hieruit mogelijk financiële besparing volgen is te zien als er wordt gekeken naar onderzoek verricht door Amazon en Google. Een vertraging in de laadsnelheid zou Amazon jaarlijks 1,6 miljard in verkopen kosten. Bij Google bleek dat het vertragen van de laadsnelheid met een vierde van een seconde, zou leiden tot 8 miljoen minder zoekopdrachten per dag (Eaton, 2012).

Bij caching ontstaat alleen een probleem op het moment dat de cache vol is en er een nieuw record wordt toegevoegd. Belady (1966) hanteert in zijn onderzoek naar virtuele opslag binnen computers de term optimal replacement algotirthm, momenteel ook bekend onder de namen, optimal page replacement algorithm, en Belady’s optimal page replacement policy, dit algoritme beschrijft wat er moet gebeuren op het moment dat de cache vol is en er een nieuwe record aan wordt toegevoegd. In een blogpost beschrijft Ali (2009) een twaalftal mogelijke implementaties voor dit algoritme.

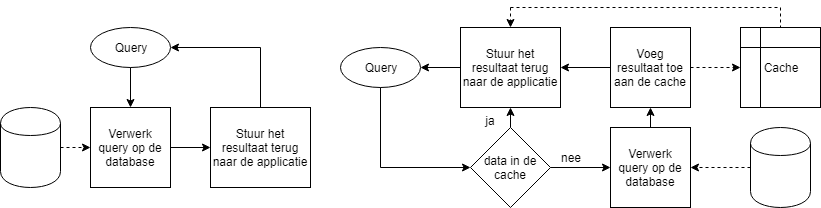
Uit bovenstaande is op te maken dat caching kan bijdragen aan verbetering op verschillende vlakken maar dat het van belang is op te letten bij de wijze waarop caching wordt geïmplementeerd, omdat verschillende algoritmes de voorkeur hebben in specifieke situaties.

# 3 Wat is de meest geschikte vorm van caching voor Java EE-applicaties die gebruik maken van REST services?

Voor Java EE applicaties die communiceren over HTTP met een achterliggende database, is een verscheidenheid aan caching methoden beschikbaar. Zo is er te denken aan HTTP Cache Headers, CDN (content delivery network), reverse proxies, web accelerators, database buffers, en key/value (data) stores (AWS, 2018).

Met dit beeld van mogelijke caching opties is het zaak, de methoden er uit te filteren die niet implementeerbaar zijn in Java of op databaseniveau. Hierdoor valt het gebruik van een CDN af. Een CDN vereist meerdere systemen die over een geografisch gebied zijn verspreid en gezamenlijk de gehoste content beschikbaar stellen. De benodigde architectuur hiervoor is niet aanwezig. Reverse proxies verrichten verzoeken voor systemen met verminderde ondersteuning voor het HTTP protocol. Hiervoor wordt doorgaans een eigen systeem opgezet. Gezien de volledige applicatie waarvoor in dit document een verbeteringsonderzoek wordt gedaan op één systeem draait is ook dit geen implementatieoptie. HTTP cache header vallen buiten de mogelijkheden gezien dit een techniek is die op de clientside wordt toegepast en in het geval van een REST applicatie geen toegevoegde waarde biedt, omdat er reeds sprake is van minimale informatieoverdracht. Ditzelfde argument sluit ook toepassing van web accelerators uit.

Hoewel de meeste databases ondersteuning bieden voor database buffers, waaronder de gebruikte MySQL implementatie (MySQL, 2018), is dit in de huidige situatie een inferieure techniek ten opzichte van een key/value (data) store, die de noodzaak tot communicatie met de database in zijn geheel ontneemt en zo bijdraagt aan een nog grotere verbetering van de IOPS.



Figuur 1 toont een systeem zonder (links) key/value (data) store en een systeem met (rechts) key/value (data) store.

In figuur 1 wordt de werking van een key/value (data) store getoond. Daar waar in een situatie zonder caching telkens de database wordt aangesproken bij het verwerken van een query, wordt dit bij een systeem met een key/value (data) store pas gedaan nadat er een cache miss is opgetreden. In het geval van een cache hit haalt het systeem de data op uit de cache, hierdoor verlagen de retrieval costs.

Hoewel het te beargumenteren is om caching te laten plaatsvinden op het niveau van de database als er sprake is van een centrale database, dit in verband met één single point of definition, is er binnen de doelarchitectuur geen sprake van meerdere applicaties die gebruik maken van dezelfde data.

Omdat er slecht sprake is van één applicatie die de data aanspreekt (aan de serverzijde) is het sneller om de communicatie met de database geheel te vermijden daar waar dit kan.

Na evaluatie van de mogelijke caching technieken binnen een Java EE applicatie, heeft er rekening houdend met de huidige architectuur, een filtratie plaatsgevonden die tot twee mogelijke kandidaten heeft geleidt. Uit die twee kandidaten komt het gebruik van een key/value (data) store naar voren als de meest geschikte kandidaat voor een Java EE applicatie die gebruik maakt van REST services.

Het is aan te bevelen om in de volgende deelvraag uit te gaan van implementaties van caching bibliotheken die een key/value (data) store implementeren.

# 4 Hoe verhouden beschikbare caching bibliotheken voor Java-applicaties zich tot elkaar?

Met behulp van google is een beknopte lijst samengesteld van aangeboden bibliotheken.

* Java Caching System (Apache, 2018)
* Caffeine (ben-manes, 2018)
* Ehcache (Ehcache, 2018)
* OSCache (nicklockwood, 2016)
* Cache4J (Stepovoy, 2006)
* ShiftOne (Drost, 2004)
* WhirleyCache (Whirleycott, 2013)
* SwarmCache (Watkinson, 2003)

Uit overleg zijn enkele SMART gedefinieerde meetcriteria naar voren gekomen.

1. Er is een Maven dependency beschikbaar.
2. De bibliotheek wordt nog onderhouden.
3. Er is functionaliteit aanwezig voor afhandeling van invalidaties.
4. De cache grootte is configureerbaar.
5. Er is kwalitatief goede en voldoende documentatie online vindbaar.
6. Geïmplementeerde verschoningsalgoritmes.

Om een completer beeld te schetsen staan in de resultaattabel ook de laatste releaseversie en releasedatum opgenomen.

Documentatiebeoordeling is opgedeeld in : 1 = slecht, 2 = middelmatig, 3 = goed.

Configureerbaarheid van de cache grootte van een bibliotheek is opgedeeld in:

C1. Op basis van aantal objecten.

C2. Op basis van gebruikte device opslag.

C3. Op basis van gebruikt RAM geheugen.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Meest recente release | Meest recente releasejaar | Er is een Maven dependency beschikbaar | De bibliotheek wordt nog onderhouden | De cache grootte is configureerbaar | Er is kwalitatief goede en voldoende documentatie online vindbaar | Geïmplementeerde verschoningsalgoritmes |
| Java Caching System | 2.2.1 | 2018 | ja | ja | C1, C2 | 3 | FIFO, LRU, MRU |
| Caffeine | 2.6.2 | 2018 | ja | ja | Onbekend | 2 | Onbekend |
| Ehcache | 3.6.1 | 2018 | ja | ja | C1,C2,C3 | 3 | LRU, LFU, FIFO, Custom |
| OSCache | 2.4.1 | 2011 | ja | nee | Onbekend | 1 | Onbekend |
| Cache4J | 0.4 | 2006 | nee | nee | C1 | 1 | LFU, LRU, FIFO |
| ShiftOne | 2.0b | 2004 | ja | nee | Onbekend | 1 | LFU, LRU, FIFO |
| WhirleyCache | 1.0.1 | 2011 | ja | nee | Onbekend | 1 | Onbekend |
| SwarmCache | 1.0 RC2 | 2003 | ja | nee | C1 | 2 | LRU, Automatic (op basis van de JVM garbage collector, TimeOut, Hybrid |

Bibliotheken die geen (Engelstalige) documentatie leveren, zijn als niet geschikt beoordeeld, doordat het implementeren hierdoor vermoeilijkt. Ook bibliotheken die geen recente releases kennen, zijn ongeschikt geacht omdat niet met zekerheid is te zeggen of hij implementeerbaar is binnen huidige Java versies.

Na toepassing van bovenstaande filters, komen de bibliotheken Java Caching System en Ehcache naar voren als mogelijke kanshebbers. Ehcache meer configureerbaard dan Java Caching System en is om die reden gekozen als bibliotheek om verder mee te experimenteren.

# Toepassing

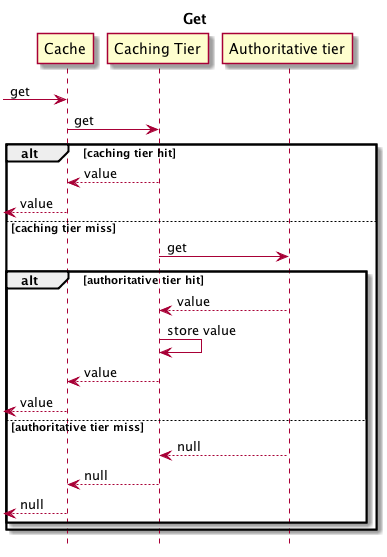
De bestaande applicatie cachet met twee caches binnen drie klassen, cache worden dus gedeeld tussen klassen.

De klasse CacheFactory biedt de mogelijkheid tot het krijgen van de verschillende caches, dit gebeurd op basis van het cache-config.xml bestand. Hierdoor heeft alleen de klasse CacheFactory weet van het configuratiebestand en hoe deze ingelezen moet worden. Dit had ook op basis van code gekund in plaats van een XML-bestand. Andere klassen kunnen een cache opvragen op basis van een de enum CacheType.

Het configuratiebestand beschrijft een cache template, namelijk: three-tiered-cache. Dit template beschrijft op welke media er gecachet wordt, en hoeveel cache er op elk medium mag staan. (<http://www.ehcache.org/documentation/3.6/xml.html>, <http://www.ehcache.org/documentation/3.6/getting-started.html>) Offheap (ook wel big memory) is anders dan heap, doordat offheap niet door de garbage collector wordt beïnvloed en doordat objecten in byte vorm worden opgeslagen. Hierdoor kan de offheap groter zijn zonder dat de JVM crasht vanwege een geheugen limiet of vanwege de garbage collector die te lang bezig is. De offheap is wel langzamer dan de heap vanwege de serialisatie en deserialisatie die plaats vinden bij het ophalen en opslaan van gegevens (http://www.ehcache.org/documentation/2.8/configuration/bigmemory.html). (https://www.voxxed.com/2015/04/a-look-at-the-ehcache-storage-tier-model-with-offheap/)Klassen die gecachet worden in de offheap of op de disk dienen de interface Serializable te implementeren. (http://www.ehcache.org/documentation/3.6/serializers-copiers.html)

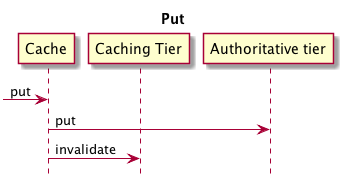
Dit cache template maakt ook gebruik van diskstorage, hiervoor moet een persistence directory worden aangewezen, in dit geval cache. Disk kent het attribuut persistent, als dit attribuut op true staat, is het cache dat op de disk stond na een restart weer beschikbaar, als dit op false staat niet. (http://www.ehcache.org/documentation/3.6/tiering.html)

Het configuratiebestand beschrijft ook twee caches, namelijk session en track-in-playlist met de bijbehorende key- en valuetypes. Bij het beschrijven van deze types is het van belang om te vermelden uit welke packages de types komen.



(http://www.ehcache.org/documentation/3.6/images/Get.png)

Ehcache haalt bij een multi-tier cache data op, op de volgende manier, het probeert eerst in het snellere cache te kijken, is de waarde hier opgeslagen dan wordt dit gebruikt, is dit niet het geval dan wordt er naar het volgende (langzamere) tier gekeken. Als in het volgende tier de waarde wordt gevonden wordt dit opgehaald én opgeslagen in de snellere tier. Dit principe herhaald zich voor het aantal tiers dat gedefinieerd is. Als er bij het laatste tier geen resultaat wordt gevonden is er sprake van een cache miss en moet er dus als nog naar de database worden gekeken.



TODO: stukje bijschrijve over het plaatsen van gegevens in de cache

De documentatie van Ehcache is soms misleidend, er zijn vrij grote verschillen tussen versie 2.x en 3.x, maar in de documentatie is het verschil alleen te zien in de URL. Bovendien is niet alle documentatie beschikbaar voor de nieuwste versie.

TODO: juiste bronvermelding (APA)

TODO: alle tekst een beetje aan elkaar knopen.

TODO: verwijzingen naar code voorbeelden.

# Code voorbeelden

## CacheFactory

**public class** CacheFactory {  
  
 **private static** CacheFactory *instance* = **new** CacheFactory();  
 **private** CacheManager **cacheManager**;  
  
 **private** CacheFactory() {  
  
 URL url = getClass().getClassLoader().getResource(**"cache-config"**);  
 Configuration config = **new** XmlConfiguration(url);  
 **cacheManager** = CacheManagerBuilder.*newCacheManager*(config);  
 **cacheManager**.init();  
 }  
  
 **public static** CacheFactory getInstance() {  
 **return** *instance*;  
 }  
  
 **public** Cache createCache(CacheType type) {  
 **switch** (type) {  
 **case *SESSION***:  
 **return cacheManager**.getCache(**"session"**, String.**class**, SessionExpirationDTO.**class**);  
 **case *TrackInPlaylist***:  
 **return cacheManager**.getCache(**"track-in-playlist"**, Integer.**class**, TrackCollectionDTO.**class**);  
 }  
 **return null**;  
 }  
  
 **public enum** CacheType {  
 ***SESSION***,  
 ***TrackInPlaylist*** }  
}

## Toepassing in code

### Aanmaken van cache

Cache<String, SessionExpirationDTO> **cache** = CacheFactory.*getInstance*().createCache(CacheFactory.CacheType.***SESSION***);

### Toevoegen aan cache

**cache**.put(token, sessionToExtend);

### Opvragen uit cache

**cache**.get(token)

## cache-config.xml

<**config  
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'  
 xmlns='http://www.ehcache.org/v3'  
 xsi:schemaLocation="http://www.ehcache.org/v3 http://www.ehcache.org/schema/ehcache-core.xsd"**>  
  
 <**persistence directory="cache"**/>  
 <**cache-template name="three-tiered-cache"**>  
 <**resources**>  
 <**heap unit="MB"**>1</**heap**>  
 <**offheap unit="MB"**>10</**offheap**>  
 <**disk unit="MB" persistent="true"**>200</**disk**>  
 </**resources**>  
 </**cache-template**>  
  
 <**cache alias="session" uses-template="three-tiered-cache"**>  
 <**key-type**>java.lang.String</**key-type**>  
 <**value-type**>nl.knaake.erik.crosscuttingconcerns.dtos.SessionExpirationDTO</**value-type**>  
 </**cache**>  
  
 <**cache alias="track-in-playlist" uses-template="three-tiered-cache"**>  
 <**key-type**>java.lang.Integer</**key-type**>  
 <**value-type**>nl.knaake.erik.crosscuttingconcerns.dtos.TrackCollectionDTO</**value-type**>  
 </**cache**>  
</**config**>

Bronnen

Ali <http://javalandscape.blogspot.com/2009/01/cachingcaching-algorithms-and-caching.html>

Apache https://commons.apache.org/proper/commons-jcs/

AWS <https://aws.amazon.com/caching/>

Babich https://uxplanet.org/mobile-web-when-speed-matters-d371a961cf46

Belady <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.641.9013&rep=rep1&type=pdf>

Ben-manes https://github.com/ben-manes/caffeine

Basques, Kearney, Miller & Osmani <https://developers.google.com/web/fundamentals/performance/rail>

Drost http://jocache.sourceforge.net/

Eaton <https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales>

Ehcache http://www.ehcache.org/

Fontys https://www.fhict.nl/vakken/DOTframework/

Citrix <https://www.citrix.nl/glossary/caching.html>

MySQL <https://dev.mysql.com/doc/refman/8.0/en/innodb-buffer-pool.html>

Stepovoy http://cache4j.sourceforge.net/

Tawara <https://dzone.com/articles/introducing-amp-assimilating-caching-quick-read-fo>

Watkinson http://swarmcache.sourceforge.net/

Whirleycott https://github.com/whirlycott/Whirlycache

Bijlagen