Onderzoeksrapport

Cache in Spotitube

—

VERSIE **1**
DATUM **10 - 01 - 2022**
COURSE **OOSE - DEA**

STUDENT
Thomas Beumer
585380

DOCENT
Meron Brouwer

Inhoudsopgave

1 - Inleiding

In dit document wordt onderzocht hoe je effectief kan cachen in een JEE applicatie. Voor het onderzoek zijn drie deelonderzoeken opgesteld met hoofd- en deelvragen;

“Cache” is een algemeen concept. In het hoofdstuk “Onderzoek cache” is de betekenis van het concept cache onderzocht binnen een JEE applicatie of nog specifieker: Spotitube.

Cache kun je op verschillende manieren inpassen. Het is mogelijk om zelf een systeem te ontwikkelen of een bestaand systeem te integreren. In het hoofdstuk “Onderzoek cache library’s” is onderzocht of er een library wordt toegepast en wat vervolgens de meest geschikte is.

In het hoofdstuk “Onderzoek toepassing” wordt de integratie onderzocht en (technische) haalbaarheid aangetoond door middel van uitwerkingen in de Spotitube applicatie.

Het doel van dit onderzoek is om een bestaand stukje technologie te vervangen met een nieuwe technologie. Dat doel wordt behaald als caching is ingepast op de bestaande architectuur van Spotitube binnen de gestelde criteria.

1.1 - Hoofdvraag en deelvragen

Om de doelstelling te behalen, wordt een onderzoek opgesteld aan de hand van de hoofdvraag; “**Hoe kan je caching effectief toepassen binnen een JEE REST app?**”. De hoofdvraag is opgedeeld in drie deelvragen;

1. Wat zijn de criteria om te bepalen welke cache technologie wordt toegepast?
2. Welke eisen worden gesteld aan een cache bibliotheek voor een JEE applicatie?
3. Hoe integreer je caching in een JEE applicatie?

1.2 - Methode

Voor het verzamelen van data zijn de volgende onderzoeksmethode toegepast (*ICT, 2021*);

Literature study	Toegepast voor algemeen onderzoek naar informatie over caching.
Available product analysis	Toegepast voor onderzoek naar bestaande cache bibliotheken.
Community research	Toegepast voor onderzoek naar gezamenlijke cache gerelateerde probleemstellingen.
Prototyping	Toegepast om (technische) haalbaarheid aan te tonen.

Tabel #1 - Onderzoeksmethode

2 - Onderzoek caching

Om kwantitatief te meten hoe caching effectief wordt toegepast, is criteria onderzocht en opgesteld aan de hand van de hoofdvraag;

- “**Wat zijn de criteria om effectief te cachen in een JEE applicatie?**”

Om op de hoofdvraag antwoord te geven, zijn de volgende deelvragen opgesteld:

1. Wat is cache?
2. Welke cache technologieën zijn beschikbaar?
3. Welke cache eigenschappen zijn relevant?
4. Wat zijn de verschillende eigenschappen, features en kenmerken tussen X, Y en Z?
5. Wat zijn de criteria om te bepalen welke cache technologie wordt toegepast?

2.1 - Wat is cache?

Cache is een “*high-speed data storage layer which stores a subset of data*” (*Amazon, z.d.*). Dit betekent dat een (toekomstig) data-verzoek sneller geladen wordt dan dat van de oorspronkelijke databron. Een gecachet verzoek is sneller omdat resultaten uit (historische) data-verzoeken zijn opgeslagen en hergebruikt. Een normaal (ongecachet) verzoek moet het resultaat eerst berekenen. Bovendien kan je cache opslaan op “fast-access” hardware zoals RAM, om data sneller te lezen en schrijven (*Amazon, z.d.*).

2.2 - Welke cache technologieën zijn beschikbaar?

Het primaire doel van “cache” is om de performance van data-verzoeken te optimaliseren door het belang van de primaire datastore laag te reduceren. Dat doel kan je op de volgende manieren bereiken (*Amazon, z.d.*);

1. *HTTP Cache Headers*
   Ook wel “cache control header” genoemd, wordt toegepast om het browser cache beleid te specificeren in de response headers. Het beleid heeft effect op de manier waarop gecachet wordt, waar gecachet wordt en op de maximale leeftijd voordat de cache verloopt (*Imperva, 2019*).

2. *Content Delivery Network (CDN)*
   Een CDN is een geografische verzameling van een groep (proxy)servers die content sneller leveren. Een verzoek voor (statische) content (dat kan zijn: HTML, JS, CSS, foto’s en video’s) wordt via een CDN door de meest optimaal geografisch gelegen server geleverd (*Cloudflare, z.d.*).

3. *Reverse Proxies*
   Een reverse proxy server vangt inkomende verzoeken van de client op en verstuurd die naar een (web)server. De proxy is dus een abstracte laag die je kan toepassen voor (*NGINX, 2022*);

   A. *Load balancing*
      Een reverse proxy kan inkomende verzoeken distribueren over de servers en verzekert dat een van de servers niet overbelast raakt. Als server “A” op maximale capaciteit draait, dan verbindt de proxy de client met server “B” om “A” te ontlasten., bijvoorbeeld. Hierdoor wordt de algehele server capaciteit verdeeld over de servers en wordt data sneller geladen omdat er meer rekenkracht beschikbaar is (*NGINX, 2022*).

   B. *Web acceleration*
      Een web accelerator kan inkomende en uitgaande data comprimeren om verzoeken sneller te leveren. Bovendien is het mogelijk om een verzoek met SSL te encrypten om de (web)server te ontlasten (*NGINX, 2022*).

4. *Key/value stores*
   Een key/value store is een datastructuur die bestaat uit twee stukken data (oftewel een “data pair”). De “key” is een unique identifier die een koppeling maakt met de “value”. Een key/value datastructuur wordt al langer gebruikt, denk aan databases. Toch zijn er enkele voordelen ten opzichte van een row-column-based structuur. Het data formaat maakt schrijf- en leesacties snel omdat er geen afhankelijkheden/relaties zijn. Ook bevat een key/value store geen placeholders zoals “null” waardes die onnodig ruimte reserveren (*Hazelcast, 2019)*.

2.2.1 - Wat zijn de verschillen tussen X, Y en Z?

* Complexiteit wordt bepaald door de combinaties van het aantal knopen met het aantal verbindingen (*Theunissen T., z.d.*).

** De beschikbaarheid van informatie wordt kwantitatief gemeten met behulp van de Google keyword planner tool. De resultaten indiceren de populariteit van de desbetreffende technologie. Op grondt van een educated guess is “voldoende” +1M resultaten en “goed” +10M resultaten.

	Header	**CDN**	**Reverse Proxy**	**Key/value stores**
Geografisch	-	x	x	-
Local disk store	x	-	-	x
Remote cache store	-	x	x	x
In-mem store	-	-	-	x
Client-side	x	x	-	x
Server-side	-	x	x	x
Eist remote server	-	x	x	-
Eist service provider	-	x	-	-
Database cache	-	-	-	x
Google-resultaat **	Goed	Provider afh.	Voldoende	Goed
Google-trends	Toenemend	Gestagneerd	Gestagneerd	Gestagneerd
Complexiteit *	Laag	Gemiddeld - Hoog	Gemiddeld - Hoog	Laag - Gemiddeld

Tabel #2 - Verschillen in X, Y en Z.

	Gem. maandelijkse verzoeken 2020	Gem. maandelijkse verzoeken 2021	Gem. Trend	Resultaten afg. jaar
“Cache headers”	100 – 1K	1K – 10K	+1000%	+16.4M
“CDN”	100K – 1M	100K – 1M	+0%	+2580M
“Reverse Proxy”	10K – 100K	10K – 100K	+0%	+6.8M
“Key/value stores”	1K – 10K	1K – 10K	+0%	+2730M

Tabel #3 - Google zoekterm populariteit

2.3 - Wat zijn de criteria om te bepalen welke cache technologie wordt toegepast?

** zie tweede alinea, hoofdstuk 2.2.1.*

Algemene criteria die van belang zijn voor het introduceren van een nieuwe techniek zijn;

1. **Er is voldoende * documentatie beschikbaar online.**
2. **Online documentatie is Engels of Nederlandstalig.**
3. **Er is geen neerwaartse trend in populariteit.**

Spotitube is een server-side REST app die op 1 omgeving draait en is ontwikkeld in het kader van een schoolopdracht. Daarmee zeggende staan de volgende criteria vast voor het type technologie;

4. **Er is geen externe server toegepast.**
5. **Server-side cache is integreerbaar.**
6. **Query resultaten zijn cachebaar.**
7. **Heap / in-memory storage cache is ondersteund**
8. **Offheap / disk storage cache is ondersteund**

Met een duidelijk beeld van cache is het zaak om de technologieën te filteren aan de hand van de criteria.

2.3.1 - Cache headers

Omdat Spotitube een server-side applicatie is, valt het gebruik van HTTP cache headers af. Deze technologie wordt client-side toegepast door de browser.

2.3.2 - CDN

Een CDN maakt gebruikt van externe servers die door een service provider gehost wordt. De benodigde financiën en architectuur is hiervoor niet aanwezig. Tevens is het best practice om vanaf een CDN alleen statische assets te leveren. Als je dynamisch data verstuurt, dan wordt het verzoek door het CDN van de primaire server opgehaald en neemt de latentie af (Jetha H., 2021).

2.3.3 - Reversed proxy

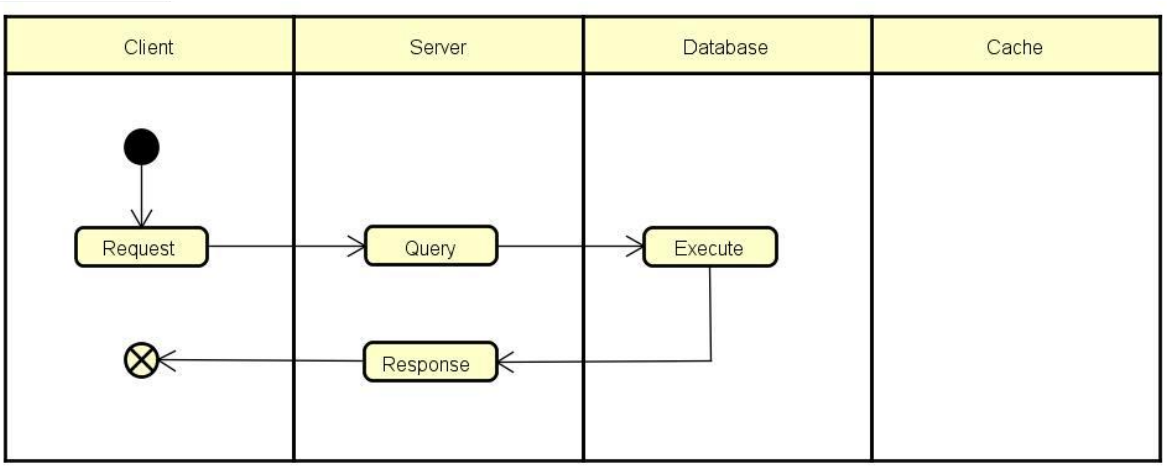
Een reversed proxy staat (fysiek) voor de server. Low level (in-memory) cache implementatie is daardoor niet mogelijk (Cloudfare, z.d.). Tevens vormt een proxy server een risico dat gebaseerd is op het Single Point of Failure principe.

2.3.4 - Key/value stores

Een key/value store voldoet aan de gestelde criteria. Dat komt mede omdat het concept breed inzetbaar is. Client-side, server-side en zelfs op database niveau wordt het concept toegepast voor cache.

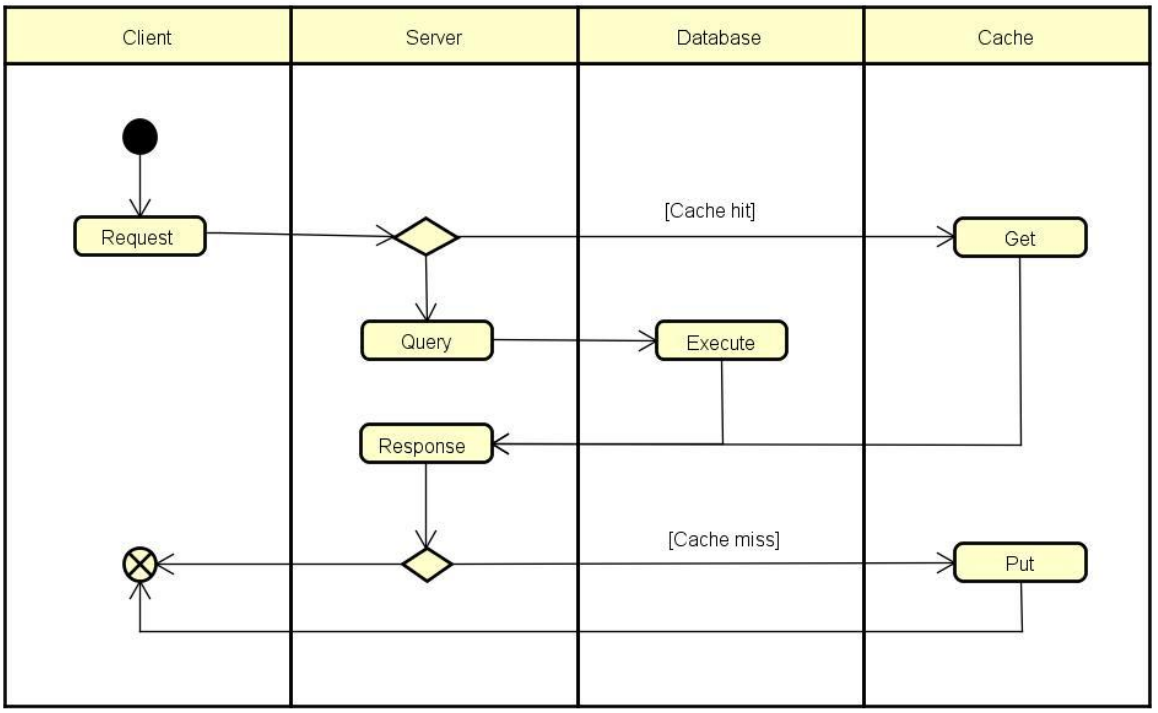
2.4 - Conclusie

Het filteren van cache technologieën heeft tot één kandidaat geleid en dat is de key/value store. In afbeelding #1 zie je dat bij de huidige situatie elk HTTP verzoek van de client, via de server de database wordt aangesproken.



Afbeelding #1 - Huidige situatie zonder key/value store

In de gewenste situatie wordt bij ieder verzoek van de client door de server gecontroleerd of er sprake is van een “cache hit”. Als dat het geval is, dan wordt het resultaat verpakt in een response en teruggestuurd naar de client. Als er sprake is van een “cache miss”, dan wordt de query uitgevoerd op de database en wordt het resultaat vanaf de server in de cache gezet. Het toekomstige zelfde verzoek wordt uit de cache gehaald in plaats van database.



Afbeelding #2 - Gewenste situatie met key/value store

3 - Onderzoek cache library's

Cache kan je op verschillende manieren integreren. Het is mogelijk om zelf een laag te ontwikkelen of je gebruikt een bestaand systeem. Die vraag wordt onderzocht aan de hand van de hoofdvraag;

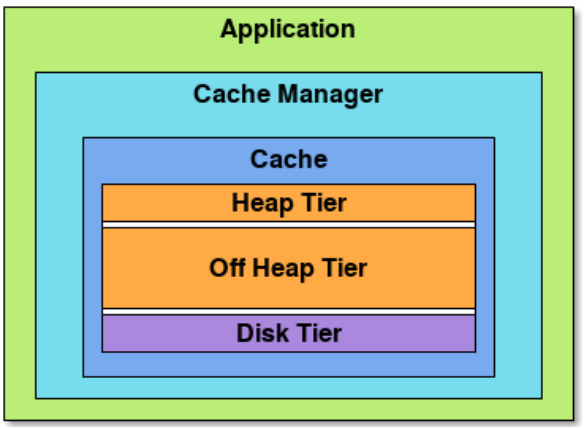
- “**Welke eisen worden gesteld aan een cache library voor JEE applicaties?**”

Om op de hoofdvraag antwoord geven zijn de volgende deelvragen opgesteld:

1. Waarom een cache library?
2. Welke cache library’s zijn geschikt voor JEE applicaties?
3. Wat zijn de criteria voor een geschikte cache library?

3.1 - Waarom een cache library?

Java heeft een ingebouwde bibliotheek `java.nio` waarmee je “unmanaged” geheugen kan wegschrijven en lezen naar RAM geheugen. Met deze bibliotheek kan je als programmeur zelf een cache store laag ontwikkelen. Een beperking van dit systeem is dat geheugen buiten JVM wordt gemanaged. Heap (in-memory) opslag is daardoor niet mogelijk en laat dat nou net het snelste geheugen zijn. De cache op de heap staat in het geheugenmodel en kan daardoor direct aangeroepen worden vanuit de stack in tegenstelling tot unmanaged geheugen, waar eerst een lees actie plaatsvindt gevolgd door deserialisatie. Een risico bij heap opslag is dat JVM op een gegeven moment een “Full GC” ondergaat. Als de heap groter is dan 1GB, dan zorgt de garbage collector voor performance issues met pauzes in de applicatie als gevolg. Als programmeur wil je de heap daarom zo min mogelijk belasten om kostbaar geheugen te sparen. In de praktijk wordt cache in de heap meestal toegepast om kleine frequente hoeveelheid data te cachen zoals gebruikerssessies. De offheap wordt gebruikt voor grotere minder frequente vormen van data zoals query resultaten. (*Moeller R., 2011*)



Afbeelding #3 - Multi-tier cache (EHCACHE, z.d.)

Het combineren van geheugenlagen zoals de heap, offheap en disk wordt gezien als een “multi-tier storage” architectuur. Het ontwikkelen van een multi-tier storage architectuur vereist veel technische kennis over geheugen. “Geheugen” kan men als een vak apart beschouwen vanwege de diepgang, omvang en complexiteit. Het geheel zelf ontwikkelen van multi-tier cache store laag, past daarom niet binnen de scope van de opdracht. *(EHCACHE, z.d.)*

3.2 - Welke cache providers zijn geschikt voor Java applicaties?

Omdat Java nog steeds een populaire taal is (al neemt de trend af), zijn er online veel cache providers te vinden. Dit onderzoek is gericht op een selectie die samengesteld is door Nicolas Fränkel, een Java enthousiast met 15+ jaar ervaring. De selectie bestaat uit 9 relatief bekende cache providers met recente releases. (*Fränkel N., 2021*)

3.2.1 - Wat zijn de verschillen tussen X, Y en Z?

Voor dit onderzoek zijn relevante eigenschappen, kenmerken en features opgenomen die je kan opdelen in drie categorieën; onderhoud, techniek en documentatie.

S1 = Disk	S2 = Offheap	S3 = Heap	S4 = Clusters
O = Onvoldoende	V = Voldoende	G = Goed	

	#	Updated	Stores	JCache	Multi-T	Ontzetting	Maven	Doc
Java Caching	2.17.1	Dec 31, 2021	S1, S2, S3, S4	x	-	LRU	x	G
Guava	31.0.1	Sep 27, 2021	S1, S2, S3	-	-	FIFO	x	V
Caffeine	3.0.5	Dec 03, 2021	S3	x	-	FIFO	x	V
Ehcache	3.9.6	Sep 24, 2021	S1, S2, S3	x	x	LRU LFU FIFO	x	G
Infinispan	12.1.6	Jul 03, 2021	S3	x	-	LFU	x	G
Coherence	21.12	Dec 11, 2021	S1, S4	x	-	LRU LFU MFU MRU	x	V
Ignite	2.11.1	Dec 21, 2021	S1, S3, S4	x	x	LRU FIFO	x	V
Geode	1.14.2	Dec 10, 2021	S3	-	-	LRU	x	G
Hazelcast	5.0.2	Dec 17, 2021	S3	x	-	LRU LFU	x	V

Tabel #4 - Verschillen tussen X, Y en Z

3.3 - Wat zijn de criteria voor een geschikte cache library?

Algemene criteria die van belang zijn voor introduceren van een library;

1. **Er is voldoende documentatie beschikbaar online.**
2. **Online documentatie is Engels of Nederlandstalig.**
3. **De bibliotheek wordt onderhouden.**

Om een cache store laag in te passen zijn de volgende technische criteria opgesteld;

4. **Er is een Maven dependency beschikbaar.**
5. **Multi-tier cache architectuur is ondersteund met;**
    a. **Heap storage**
    b. **Offheap storage**
    c. **Disk storage**

3.4 - Conclusie

Na filter van kandidaten aan de hand van het criterium “Multi-tier cache architectuur wordt ondersteund.” resteren 2 kandidaten; “Ehcache” en “Ignite”. Een probleem met Ignite is dat er niet duidelijk gedocumenteerd wordt waar in het interne geheugen (naast in-memmory en disk geheugen), cache wordt weggeschreven. Daarentegen voldoet Ehcache wel aan alle gestelde criteria en is hun documentatie duidelijk. Tevens is Ehcache expliciet gespecialiseerd in cache en claimen zij “*JAVA’S MOST WIDELY-USED CACHE*” te zijn. (*EHCACHE, z.d.*)

4 - Onderzoek toepassing

Om EhCache succesvol in Spotitube te integreren dien je een aantal stappen te volgen. Allereerst voeg je een EhCache Maven dependency toe aan het project via de `pom.xml`.

```
<dependency>
      <groupId>org.ehcache</groupId>
      <artifactId>ehcache</artifactId>
      <version>3.9.9</version>
</dependency>
```

Code #1 - dependency

4.1 - Cache configuratie

Nu is het zaak om het cachebeleid te specificeren. Dat kan in de code of vanuit een XML bestand. Deze uitwerking gaat in op het XML bestand omdat in de applicatie statische configuratie bestanden los opgeslagen staan. Tevens is het robuuster omdat een wijziging niet direct in de code plaatsvindt en een configuratiebestand frequent wordt aangepast. “code #2” toont een voorbeeld van het XML stramien waarin het beleid gespecificeerd wordt.

```
<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 dir here →
      <!-- cache templates here →
      <!-- cache alias here →
</config>
```

Code #2 - XML basis

Omdat cache (onder andere) wordt weggeschreven naar de schijf moet een `persistence directory` aangewezen worden. De cache wordt ontzet in die map. In het voorbeeld “Code #3”, wordt de environmentvariabele `${java.io.tmpdir}` toegepast om een tijdelijke folder te genereren.

```
.<persistence directory="${java.io.tmpdir}/spotitube-cache"/>
```

Code #3 - persistence directory

Het config bestand kan meerdere cache templates bevatten. Een template specificeert hoe en waar er gecachet wordt. Templates zijn handig om het config bestand leesbaar te houden vanwege het DRY principe. Ook draagt het bij aan robuustheid doordat een wijziging op 1 positie plaatsvindt. In het voorbeeld “Code #4” staat een multi-tier cache architectuur geconfigureerd onder de naam `tiered-cache`. De app probeert cache uit de `heap` te halen. In het geval van een cache-miss, wordt de `offheap` aangesproken en daarna de `disk`. Als in één van tiers een cache hit plaatsvindt, dan wordt data in de top tier geplaatst. Als alle tiers een cache miss retourneren wordt via de gebruikelijke route de database aangesproken en het resultaat in de cache geplaatst.

```
<cache-template name="tiered-cache">
     <resources>
          <heap unit="MB">100</heap>
          <offheap unit="MB">500</offheap>
           <disk unit="MB" persistent="true">1000</disk>
     </resources>
</cache-template>
```

Code #4 - Cache template

Nu een cache template gespecificeerd is, kan het gekoppeld worden aan een key/value store. In het voorbeeld “Code #5”, krijgt de `cache` tag een `alias` en `uses-template` attribuut. Het `alias` attribuut wordt gebruikt om een koppeling te maken vanuit de code. Het `uses-template` attribuut verwijst naar een template naam. De store moet twee child tags bevatten;

1. De `<key-type>` tag verwijst naar het datatype van de unique identifier. Dat is meestal een `java.lang.Integer` maar kan ook een `java.lang.String` zijn, in het geval van een token, bijvoorbeeld.

2. De `<value-type>` tag verwijst naar het datatype van de waarde die gekoppeld is aan de key. In het voorbeeld “Code #5” is de value een DTO (oftewel een object). Ehcache kan met deze informatie data (de)serialiseren.

```
<cache alias="tracks" uses-template="tiered-cache">
     <key-type>java.lang.Integer</key-type>
     <value-type>han.ica.dea.domain.dto.TracksDto</value-type>
</cache>
```

Code #5 - Cache configuratie

Spotitube werkt met tokens die gekoppeld zijn aan sessies. Een token komt daarom in ieder HTTP verzoek voor. Omdat het bij “sessies” om een frequente kleine hoeveelheid data gaat, kan je die het beste op heap plaatsen (waarom, zie hoofdstuk “Waarom een cache library?”). In het voorbeeld “Code #6” wordt aan de cache een “time-to-live” (`ttl`) tag toegevoegd van 1 dag (oftewel `86400` seconden). De server kan aan de hand van de cache bepalen of de sessie nog actief is. Zoniet, dan wordt er opnieuw geauthenticeerd. Een verschil ten opzichte van voorbeeld “Code #4” is dat het formaat van de heap niet in MB’s is uitgedrukt maar in `entries`. Op de heap van de `session` store kunnen maximaal 1000 sessies (`entries`) actief zijn.

```
<cache alias="session">
      <key-type>java.lang.String</key-type>
      <value-type>ica.dea.domain.dto.LoginResponseDto</value-type
      >
      <expiry>
            <ttl unit="seconds">86400</ttl>
      </expiry>
      <heap unit="entries">1000</heap>
</cache>
```

Code #6 - Heap sessie cache

4.2 - Cache implementatie

Nu het cache beleid is gespecificeerd in het configuratiebestand, wordt daadwerkelijk de cache geïmplementeerd. Om dit SOLID te implementeren wordt een `CacheHelper` class opgezet als singleton. In deze class wordt het `cacheConfig` bestand uitgelezen en geïnitialiseerd aan de hand van het voorbeeld, “Code #7”;

```
private CacheHelper() {
      String cfgFile = “cacheConfig”;
      URL url = getClass().getClassLoader().getResource(cfgFile);
      Configuration config = new XmlConfiguration(url);
      cacheManager = CacheManagerBuilder.newCacheManager(config);
      cacheManager.init();
}
```

Code #7 - cacheConfig init

De `CacheHelper` class bevat een methode `create` die de configuratie uit het `cacheConfig` XML bestand ophaalt. Voor ontwikkelgemak kan je een `enum` en `switchcase` toepassen in een factory patroon om een store te koppelen. Wat van belang is dat de `getCache` parameters overeenkomen met de `alias`, `key-type` en `value-type` uit het `cacheConfig` XML bestand.

```
public Cache create(CacheAlias type) {
      switch (type) {
            case SESSION:
            return cacheManager.getCache(
                  "session", String.class, LoginResponseDto.class
            );
            case TRACKS:
            return cacheManager.getCache(
                  "tracks", Integer.class, TracksDto.class
            );
      }

      return null;
}
```

Code #8 - create cache

Met de `CacheHelper` class wordt vanuit `services`, cache daadwerkelijk ingepast. Dat kan door in de constructor van een `service` class, de `create` methode aan te roepen en te zetten. De methode retourneert een instantie van de `EhCache.Cache` class die gebruikt wordt voor CRUD operaties.

```
private Cache<Integer, TracksDto> trackCache;

public TrackService() {
      trackCache = cache.create(
            CacheHelper.CacheAlias.TRAKCS_NOT_IN_PLAYLIST
      );
}
```

Code #8 - Service constructor

Nu is het zaak, om bij ieder verzoek te controleren of de cache aanwezig is in de desbetreffende store. Dat doe je met de `get(id)` methode (zie “Code #9”). Als de methode een waarde retourneert, dan is er sprake van een cache hit en wordt het resultaat door de `service` methode geretourneerd aan de controller. Als de methode `null` retourneert, is er sprake van een cache miss en wordt via de database het resultaat opgehaald en toegevoegd aan de cache via de `put(id, value)` methode. Hetzelfde toekomstige verzoek wordt dan uit de cache opgehaald in plaats van database.

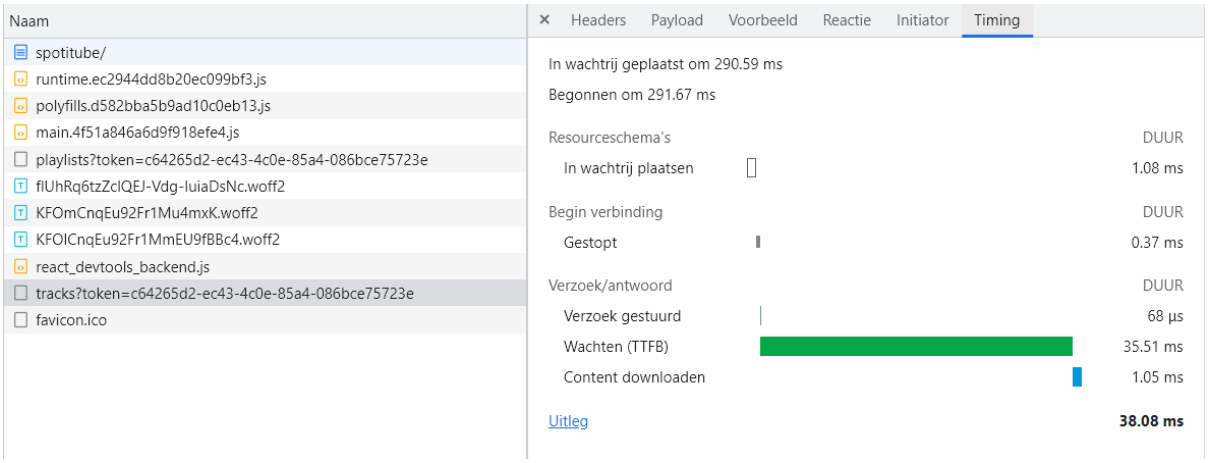
```
public TracksDto getTracksNotInPlaylist(int playlistId) {
      TracksDto tracksDto = trackCache.get(playlistId);
      if(tracksDto != null) {
            return tracksDto;
      }

      tracksDto = trackDao.getTracksNotInPlaylist(playlistId);
      trackCache.put(playlistId, tracksDto);
      return tracksDto;
}
```

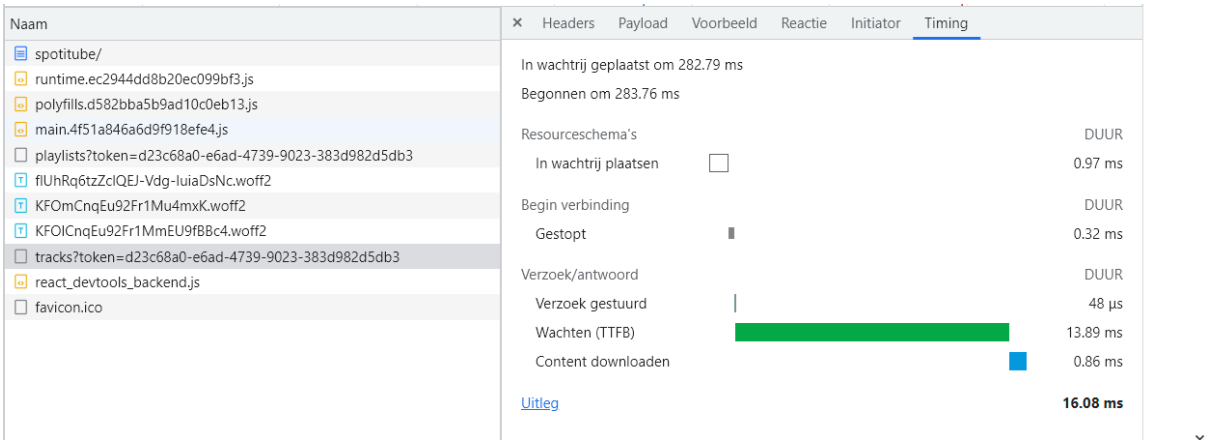
Code #9 - Service constructor

4.3 - Conclusie

In afbeelding #4 kost de totale rendertijd van een ongecached verzoek 38.08 ms. In afbeeding #5 kost hetzelfde gecached verzoek 16.08 ms. Dat betekent dat het gecached verzoek 57,77% sneller is gerenderd. Daarmee zeggende is het mogelijk om EhCache succesvol te implementeren en is de technische haalbaarheid aangetoond. Voor de volledige uitwerking, zie externe bijlage, “SpotitubeWithCache”.



Afbeelding #4 - Cache miss



Afbeelding #5 - Cache hit

Bronvermelding

1. Moeller, R. (2011, 22 mei). *Difference between “on-heap” and “off-heap”*. Stack Overflow. Geraadpleegd op 12 januari 2022, van https://stackoverflow.com/questions/6091615/difference-between-on-heap-and-off-heap

2. Hazelcast. (2019, 9 november). *Key-Value Stores Explained. Advantages & Use Cases*. Geraadpleegd op 11 januari 2022, van https://hazelcast.com/glossary/key-value-store/

3. McKeever, G., Hasson, E., Rossi, E., Hasson, E., B.L., Hasson, E., B.L., Weaver, P., Lynch, B., & Weaver, P. (2019, 30 december). *What is Cache-Control and How HTTP Cache Headers Work | CDN Guide | Imperva*. Learning Center. Geraadpleegd op 11 januari 2022, van https://www.imperva.com/learn/performance/cache-control/

4. *Methods - ICT research methods.* (2021, 5 mei). ICT Research Methods. Geraadpleegd op 10 januari 2022, van https://ictresearchmethods.nl/Methods

5. Frankel, N. (2021, 31 oktober). A list of cache providers. A Java Geek. Geraadpleegd op 12 januari 2022, van https://blog.frankel.ch/choose-cache/2/

6. Jetha, H. (2021, 28 december). U*sing a CDN to Speed Up Static Content Delivery. DigitalOcean*. Geraadpleegd op 11 januari 2022, van https://www.digitalocean.com/community/tutorials/using-a-cdn-to-speed-up-static-content-delivery

7. *Getting Started with Caching: Turbocharging Your Application Workloads.* (z.d.). Amazon Web Services, Inc. Geraadpleegd op 10 januari 2022, van https://aws.amazon.com/caching/

8. NGINX, Inc. (2022, 3 januari). *What is a Reverse Proxy Server?* NGINX. Geraadpleegd op 11 januari 2022, van https://www.nginx.com/resources/glossary/reverse-proxy-server/

9. *What is a CDN?* (z.d.). Cloudflare. Geraadpleegd op 11 januari 2022, van https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

10. What is caching. (z.d.). Cloudflare. Geraadpleegd op 11 januari 2022, van https://www.cloudflare.com/learning/cdn/what-is-caching/

11. Theunissen, T. (z.d.). *Diepgang, complexiteit, omvang. Google Docs*. Geraadpleegd op 11 januari 2022, van https://docs.google.com/presentation/d/1hdksuuAMo3TnexLc28KOUmSWzAAyqFeTlvOlNkqKvrQ/edit#slide=id.g1f4d0305ab_0_56

12. EHCACHE. (z.d.). *Concepts Related to Caching*. Geraadpleegd op 12 januari 2022, van https://www.ehcache.org/documentation/3.9/caching-concepts.html#storage-tiers