Minlog – Splunk opsamling og udstillelse

Guide til udviklere

Indholdsfortegnelse

[1 Formål 3](#_Toc328724813)

[2 Opsætning af udviklingsmiljø 4](#_Toc328724814)

[2.1 Om koden 4](#_Toc328724815)

[2.2 Byg 4](#_Toc328724816)

[2.3 SosiIdCardTool 4](#_Toc328724817)

[3 Test vejledning 5](#_Toc328724818)

[3.1 Coverage 5](#_Toc328724819)

[3.2 Funktionelle tests 5](#_Toc328724820)

[3.3 Testrapport til sammenligning 5](#_Toc328724821)

[4 Performance Tests 6](#_Toc328724822)

[4.1 Opsætning 6](#_Toc328724823)

[4.2 Generering af testdata 7](#_Toc328724824)

[5 Ændringslog 8](#_Toc328724825)

# Formål

Nærværende dokument er en guide til nye udviklere af komponenten. Guiden gennemgår på overordnet plan de aktiviteter, der er nødvendige for at kunne videreudvikle komponenten.

# Opsætning af udviklingsmiljø

Projektet bør køres på en NSP in a box (NIAB)

<https://www.nspop.dk/display/public/NSP+In+A+Box>

Projektet kan i sin nuværende tilstand ikke køre på andre servere, da der er en masse maven dependencies som bliver ekskluderet for ikke at kollidere med NIAB.

Desuden bør man installere Splunk lokalt <http://www.splunk.com/>

For at teste komponenten:

1. Deploy komponenten på NIAB
2. Opsæt splunk på host maskinen (udenfor vmware).
3. Opbyg en fil med en række logninger
4. Lad Splunk indexere denne fil og loade den ind i MySql.
5. Man vil nu kunne tilgå MinLog service fra en webservice klient.

## Om koden

Applikationen er en standard DAO-Service-Controller arkitektur, hvor service-delen er undladt, da kodebasen er ret begrænset.

Controller er implementeret i Spring Webservice frameworket

<http://static.springsource.org/spring-ws/sites/2.0/>

DAO er implementeret i eBean frameworket <http://www.avaje.org/>

## Byg

For bygge skal man have installeret maven og køre "mvn clean install" fra roden af projektet. Artifakten vil efterfølgende ligge under

minlogudtraekservice/target/minlog.war

## SosiIdCardTool

Der ligger et tool til at lave "Den gode webservice 1.0.1" headers, så det er nemmere at teste om servicen virker:

java -Done-jar.main.class=dk.vaccinationsregister.testtools.SosiIdCardUtil -jar SosiIdCardTool.jar

På OS X kan man pipe resultatet over i clipboardet med pbcopy:

Java -Done-jar.main.class=dk.vaccinationsregister.testtools.SosiIdCardUtil -jar SosiIdCardTool.jar | pbcopy

# Test vejledning

Test bliver kørt automatisk når projektet bygges, som beskrevet herover.

## Coverage

For at lave coverage-tests køres

mvn clean install cobertura:cobertura surefire-report:report

fra

minlogudtraekservice/

Coverage resultaterne findes i

minlogudtraekservice/target/site/cobertura/index.html

Svar på unittests kan ses i

minlogudtraekservice/target/site/surefire-report.html

## Funktionelle tests

Formålet med de funktionelle tests er at teste hele vejen gennem komponenten. Fra webserviceklient, igennem webservice, hent data fra MySql, og returner data.

Dette gøres for hver test:

* En ny service oprettes
* En embedded mysql database i en ny process.
* Der populeres data i databasen.
* Der generes en soap security header med saml.
* Der generes en body, med en forspørgelse på test data.
* Svarets body sammenlignes med et statisk body fra en xml.

**NB!** Hvis de funktionelle tests bliver afbrudt, er der en risiko for at man ikke kan starte en mysql server efterfølgende, da mysql vil brokke sig over at der er en server instans der ikke er blevet lukket korrekt. Dette er en uhensigtsmæssighed/fejl i MySql ved denne anvendelse. Problemet løses ved genstart af computer eller ved at rydde op i pid/err i mysqls data-folder.

## Testrapport til sammenligning

Test coverage sitet kan findes under doc/coverage.zip

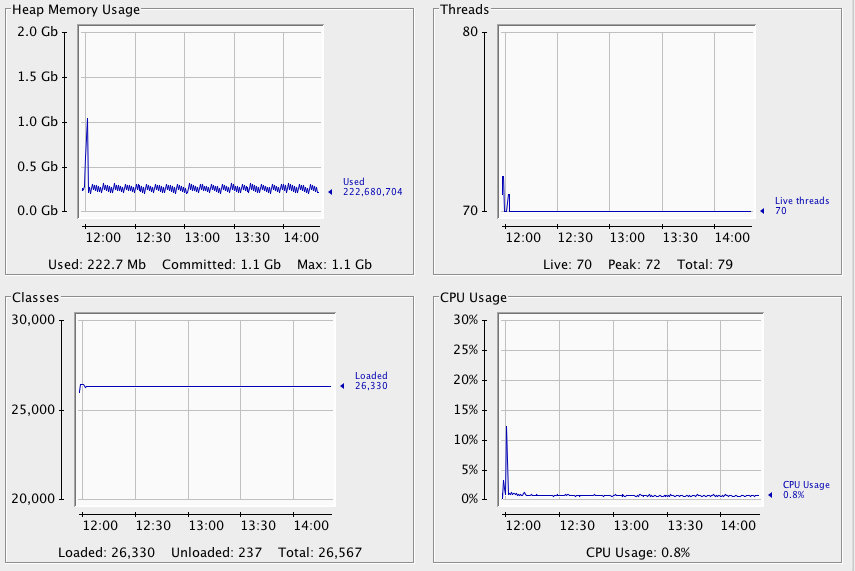
# Performance Tests

Sitet med performances-tests ligger under doc/performance.zip

Dette afsnit skal skrives om når vi har lavet nye performance tests. Der findes benerator scripts til at generere tilfældige testdata. Se senere beskrivelse om generering af testdata.

Det er ikke lykkes at få serveren til at gå ned, men i tests med et forventet throughput på 1000 request/sec bliver der kun laves 200 requests/sec. Dette kan skyldes setup'et eller en indstilling i OS'et.

Denne endurence tests laver 2 requests/sec og er lavet over 2 timer. Grafen spiker kl 12:00. Dette kan skyldes at computeren har et job der bliver eksekveret kl 12:00.  
Derved får garbage collectoren ikke lov til at lave løbende collection.  
Ved 1GB heap rydder garbage collectoren fint op.

[](https://github.com/trifork/nsi-minlog/raw/master/doc/endurence.png)

## Opsætning

Disse tests er kørt på

2GHz Intel core i7

8 GB ram

Harddisk med 5400 rpm

Opsætning af mysql:

innodb\_data\_file\_path = ibdata1:10M:autoextend

innodb\_flush\_log\_at\_trx\_commit = 1

innodb\_lock\_wait\_timeout = 50

innodb\_additional\_mem\_pool\_size=512M

innodb\_buffer\_pool\_size=4096M

innodb\_log\_buffer\_size=128M

innodb\_log\_file\_size=1024M

read\_buffer\_size= 128M

sort\_buffer\_size=4096M

tmp\_table\_size= 1024M

Det antages at databasen minlog er oprettet med adgang fra brugeren minlog og at der er minlog er blevet sat op med sosi.production = 0

## Generering af testdata

Til at genere test-data med er Benerator blevet brugt <http://databene.org/databene-benerator>

Alle kommandoer skal køres fra /performance.

Først køres benerator benerator/cpr.xml som laver CPR numre i data/cpr.csv Der generes 50.000 tilfældige CPR numre.

Dernæst benerator benerator/logentries.xml som laver 450.000.000 logs i filen data/logentries.csv

import.sql tilpasses så den absolute sti passer. **NB!** Dette skal gøres for at slippe for local parameteren til load data som laver en kopi af csv filen.

**NB!** Det kan være en fordel at slette alle indekser på nær primary key inden kørsel. Dette optimerer indsættelsen væsentligt.

# Ændringslog

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Dato** | **Ændring** | **Ansvarlig** |
| 0.1 | 2012-06-28 | Initielt Dokument | Trifork |