HAIBA

FGR-import

Design, arkitektur og udviklerguide

Indhold

1 Formål 3

2 Arkitektur- og designoverblik 4

3 Særlige forhold 5

3.1 Personhenførbare data 5

4 Fysiske Datamodeller 6

4.1.1 Database 6

4.1.2 Meta-database 6

5 Opsætning af udviklingsmiljø 7

5.1 Kildekode 7

5.2 Byggemiljø 7

5.2.1 Dependencies 7

5.3 Database-setup 7

5.3.1 Database 7

5.4 Test 7

5.5 IDE 8

5.5.1 Eclipse 8

5.6 IntelliJ Idea IDE 9

5.7 Distribution 10

6 Tips og tricks 11

6.1 Tomcat out of memory 11

6.1.1 Beskrivelse 11

6.1.2 Løsning 11

7 Referencer og kilder 12

8 Ændringslog 13

# Formål

Dette dokument giver et overblik over FGR-importeren med fokus på design og arkitektur. Dokumentet er også en guide til udviklere. Guiden gennemgår på overordnet plan de aktiviteter, der er nødvendige for at kunne videreudvikle på servicen.

Dokumentet har som formål at give et indblik i det underliggende design, de udstillede snitflader på det overordnede niveau samt særlige forhold.

Både udviklere, aftagere samt driftsleverandør kan med fordel læse dette dokument.

# Arkitektur- og designoverblik

FGR-importerens formål er at hente stamdata fra eksterne filer og importere disse i nogke klassifikationstabeller.

Overordnet set består importeren af en applikation kørende på en JEE applikationsserver (I dette tilfælde Tomcat).

Importeren er bygget op via Spring version 3.x (se <http://www.springsource.org/> ) som en standard JEE webapplikation. Kendskab til følgende Spring undermoduler vil være en fordel (MVC, batch og Integration).

Til at bygge systemet bruges Maven 3.x.

Modulerne er strukturerede på følgende måde:

* Dataimporteren har til ansvar at hente stamdata ind fra eksterne filer (over ftp eller http) og placere det i en intern model baseret på POJO's.

Dataimporteren ligger i pakken: dk.nsi.haiba.fgrimporter.importer, klassen ImportExecutor er det skedulerede job som sørger for at importere data periodisk.

* Data hentes for SKS, SHAK og SOR, hvor SKS og SHAK er defineret i samme format, se SHAK-tabelstruktur.pdf - [http://www.ssi.dk/Sundhedsdataogit/National Sundheds-it/TerminologiOgKlassifikationer/~/media/Indhold/DK - dansk/Sundhedsdata og it/NationalSundhedsIt/Klassifikationer/SygehusAfdelingsKlassifikationen/SHAK-tabelstruktur.ashx](http://www.ssi.dk/Sundhedsdataogit/National%20Sundheds-it/TerminologiOgKlassifikationer/~/media/Indhold/DK%20-%20dansk/Sundhedsdata%20og%20it/NationalSundhedsIt/Klassifikationer/SygehusAfdelingsKlassifikationen/SHAK-tabelstruktur.ashx).
* For SHAK gælder det, at shakregion.xml hentes efter import fra shakcomplete.txt. Rækker med sygehuse i klass\_shak beriges med ’Ejerforhold’, ‘Institutionsart’ samt ‘Regionskode’ fra shakregion.xml.
* Degradering: Hvis databehandlingen afbrydes utilsigtet eks. pga. kodefejl eller fysisk afbrydelse, kan database godt ende i en ufærdig tilstand, eks. halvdelen af en fil er importeret. Det tager dog pt. ca. 20 minutter at hente og processere alle 3 filer.

# Særlige forhold

## Personhenførbare data

Data fra FGR indeholder ikke personnumre

# Fysiske Datamodeller

FGR-databaserne indeholder tabeller med data, hvor Header er den overordnede tabel med prøvesvar. CaseDef indeholder de prøvesvartyper, der skal hentes prøvesvar for.

Detaljer omkring feltværdier, datatyper og lignende kan ses i SQL skemaerne som ligger på følgende URL: <https://github.com/trifork/HAIBA-FGRimporter/tree/master/database>

### Database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| Klass\_sks | Indeholder data for diagnoser og procedurer, fundet i skscomplete.txt. Kolonnen ’type’ adskiller diagnoser fra procedurer. |
| Klass\_SOR | Indeholder data fra sor.xml og mapper mellem sor-identifikation og shak-identifikation. |
| Klass\_shak | Indeholder data fra shakcomplete.txt om sygehuse og sygehusafdelinger samt udvalgte felter fra shakregion.xml |

### Meta-database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| FGRImporterStatus | Indeholder information om kørsler; start- og sluttid samt udfald. Desuden en kort tekstbesked om evt. fejl. Denne information bruges af statussiden. |

# Opsætning af udviklingsmiljø

Opsætningen af udviklingsmiljøet for LPR-databehandleren forudsætter, at følgende elementer allerede er installeret på udviklerens maskine:

* Java Developer Kit 6.0\_x
* Maven 3.x [MAVEN]
* Git 1.7.x
* MySQL 5.5.x
* Tomcat 7 (Udviklet og testet på version 7.0.34)

Installationsvejledningen indeholder detaljer omkring opsætning af ovenstående komponenter.

## Kildekode

Kildekoden er placeret i et github-repositorie og kan checkes ud på følgende måde:

git clone git@github.com:trifork/HAIBA-FGRimporter.git

## Byggemiljø

Databehandleren anvender Maven som byggesystem. Strukturen følger de generelle anbefalinger for Maven projekter, og er struktureret

efter Maven layout-konventionen.

For at bygge databehandlersystemet, skal man gøre følgende:

mvn install

Projektet indeholder udover unittests også integrationstests, der kræver en kørende database, for at undlade udførelsen af integrationstests kan parameteren -DskipITs tilføjes mvn install kommandoen.

### Dependencies

For at kunne hente NSI-specifikke afhængigheder (bl.a. nsp-util) i binær form i stedet for at skulle bygge alle afhængigheder selv på det lokale udviklingsmiljø, indeholder pom'en en reference til nexus.trifork.com, som er et artefaktrepository der er placeret hos Trifork. Binære releases af databehandleren findes også i nexus.trifork.com.

Repository’et bør, når det er muligt, udskiftes med et artefaktrepository der er driftet hos NSI. Når et sådant er etableret

## Database-setup

### Database

Databaseskema for prøvesvar mm. ligger i database-folderen.

## Test

Installationen kan verificeres ved at eksekvere databehandlerens test suite.

Testsuiten benytter JUnit og Mockito til test.

Test suiten afvikles ved at udføre følgende kommando i projektroden:

mvn test

Installationen kan yderligere verificeres (Code coverage, kode konventioner o.l.) ved at udføre kommandoen, rapporterne ligger i <projekt rod>/target/site:

mvn verify

## IDE

Databehandleren kan principielt udvikles i enhver Java IDE, der forstår Maven projekters opbygning.

I dette dokument beskrives kort opsætning for to af de pt. mest udbredte Java IDE’er: Eclipse og IntelliJ.

### Eclipse

Eclipse er ikke født med Maven support, og det anbefales derfor, at man installerer m2eclipse inden databehandleren hentes ind i Eclipse, pluginet kan hentes her:

http://www.eclipse.org/m2e/

Herefter importeres projekterne i Eclipse via ”import”:

Alternativt kan man importere projektet ved at udføre følgende kommando:

mvn eclipse:eclipse

Og herefter importere projektet på normal vis i Eclipse.

Kommandoen genererer Eclipse projektfilerne (.project og .classpath) for databehandleren. Denne metode kræver dog, at kommandoen udføres hver gang man ændrer i pom filerne.

## IntelliJ Idea IDE

IntelliJ Idea er født med Maven support, og databehandleren kan derfor direkte importeres. Projektet importeres i IntelliJ ved under ”Create new project” at vælge ”Import project from external model”. Herefter udvælges roden af databehandleren, hvorefter projektet importeres.

Det anbefales i den sammenhæng, at man krydser af i ”Import Maven projects automatically”, hvorefter IntelliJ selv detekterer nye moduler i projektet.

Alternativt kan man importere projektet ved at udføre følgende kommando:

mvn idea:idea

Herefter kan projektet importeres på normal vis i IntelliJ.

*Obs! Denne metode kræver dog, at kommandoen udføres hver gang man ændrer i pom filerne.*

## Distribution

databehandleren kan bygges til distribution eller lokal test ved at udføre:

mvn package

Dette generer en WAR fil, der efterfølgende kan deploye's lokalt eller på et testmiljø.

Til produktion bør Maven release plugin bruges, da det får tag'et bygget og automatisk får opdateret versionsnumre m.v.

Først køres:

mvn release:prepare

Går det godt køres:

mvn release:perform

# Tips og tricks

I det følgende beskrives problemer man som udvikler kan støde på, og forslag til løsning af samme.

## Tomcat out of memory

### Beskrivelse

I Tomcats log, logger den noget i stil med ”out of memory” og nævner “permgenspace”, dette kan ske ved at der hot-deployes ofte, som man typisk gør under udvikling

### Løsning

Forøg Tomcat permgen space ved at sætte følgende miljø variabel:

JAVA\_OPTS="-Xms2048m -Xmx2048m -XX:MaxPermSize=512m"

# Referencer og kilder

|  |  |  |
| --- | --- | --- |
| Reference-id | Indhold / Overskrift | Henvisning |
| [MAVEN] | Welcome to Apache Maven | <http://maven.apache.org/> |

# Ændringslog

Kilden til dette dokument kan findes på:

[https://github.com/trifork/HAIBA-FGRimporter/blob/master/doc/Design, arkitektur og udviklerguide.docx](https://github.com/trifork/HAIBA-FGRimporter/blob/master/doc/Design,%20arkitektur%20og%20udviklerguide.docx)

| Version | Dato | Ændring | Ansvarlig |
| --- | --- | --- | --- |
| 1 | 2014-02-13 | Initielt dokument | Trifork  Aksel Schmidt |
|  |  |  |  |