HAIBA

LPR-kontakter, data behandler

Design, arkitektur og udviklerguide

version 1.0

Indhold

1 Formål 3

2 Arkitekturoverblik 4

2.1 Arkitektur 5

2.1.1 Regelhåndtering 6

3 Særlige forhold 7

3.1 Personhenførbare data 7

3.2 Ændring/tilføjelse af regler og datamodel 7

4 Fysiske Datamodeller 8

4.1 LPR Database 8

4.2 Indlæggelsesdatabase 8

5 Opsætning af udviklingsmiljø 10

5.1 Kildekode 10

5.2 Byggemiljø 10

5.2.1 Dependencies 10

5.3 Database setup 10

5.3.1 Indlæggelsesdatabase 10

5.3.2 LPR database 11

5.4 Test 11

5.5 IDE 11

5.5.1 Eclipse 11

5.6 IntelliJ Idea IDE 12

5.7 Distribution 13

6 Tips og tricks 14

6.1 Tomcat out of memory 14

6.1.1 Beskrivelse 14

6.1.2 Løsning 14

7 Referencer og kilder 15

8 Ændringslog 16

# Formål

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

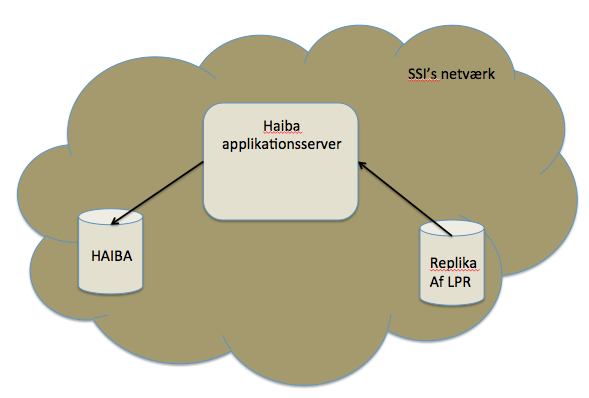
Dokumentet har som formål at give et indblik i det underliggende design, de udstillede snitflader på det overordnede niveau samt særlige forhold vedrørende LPR Databehandleren.

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

# Arkitekturoverblik

Formålet med LPR databehandleren er at importere data fra Landspatientregisteret (LPR), Data transformeres gennem nogle, af NSI/SSI definerede forretningsregler og placeres i en database bestående af HAIBA indlæggelsesdata

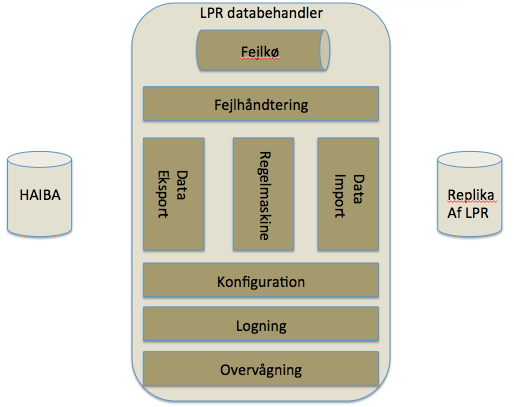
Overordnet set består LPR Databehandleren af en applikation kørende på en JEE applikationsserver (I dette tilfælde Tomcat), som har adgang til to databaser; en med LPR kontakt data hvor der importeres fra, og en med HAIBA indlæggelses data hvor der eksporteres til. LPR databasen er et replika (og subset) af den LPR database sundhedsstyrelsen bruger til afregning af sygehusene.



LPR databehandleren 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:



* Data importeren har til ansvar at hente data ind fra LPR databasen, via batches af konfigurerbar størrelse, og placere det i en intern LPR model baseret på POJO's.

Dataimporteren ligger i pakken: dk.nsi.haiba.lprimporter.importer, klassen ImportExecutor er det skedulerede job som sørger for at importere data når noget nyt dukker op, ved at se i LPR databasen om en ekstra tilføjet kolonne med import tidspunkt er tomt.

* Regelmaskinen sørger for udførelse af de beskrevne forretningsregler i den rækkefølge som står i Løsningsbeskrivelsen afsnit 2.4 (Forretningsregler), data bliver konverteret til en intern HAIBA indlæggelsesmodel baseret på POJO's.

Regelmaskinen ligger i pakken: dk.nsi.haiba.lprimporter.rules, hvor klassen RulesEngine sørger for processeringen, de enkelte forretningsregler ligger også i denne pakke og implementerer alle LPRRule interfacet

* Data eksporteren sørger for at skrive det behandlede data ned i HAIBA databasen ud fra den interne HAIBA model
* Konfigurationsmodulet sørger for opsætning af LPR databehandleren ud fra den eksterne konfiguration (eksempelvis batchstørrelser, tidsintervaller m.m.). Konfigurationsmodulet er baseret på Springs "Configuration" og ligger i pakken: : dk.nsi.haiba.lprimporter.config, hvor LPRConfiguration klassen er den primære indgang. For detaljer om hvor konfiguration skal installeres/ændres se hhv. installationsvejledning og driftsvejleding afsnit 4.1.3 (Konfiguration af LPR databehandler)
* Logningsmodulet sørger for applikationslog til debug og applikationsfejlhåndtering - ikke fejl i forbindelse med databehandling, dette sørger fejlhåndteringsmodulet for.
* Overvågningsmodulet sørger for at udstille en html side, hvor man kan se, eller maskinelt overvåge, LPR databehandlerens tilstand.

Overvågningsmodulet ligger i pakken: : dk.nsi.haiba.lprimporter.status, hvor StatusReporter klassen, er den der bliver ramt når der spørges på status URL'en via en browser

* Fejlhåndteringsmodulet har til ansvar at rapportere datafejl, enten fordi input format ikke kan bruges, eller hvis noget data falder uden for gældende regelsæt og derved ikke kan proceseres. Fejlhåndteringsmodulet skriver til en fejlkø (fil og databasetabel), hvor statistikere eller klinisk personale kan se hvad det er for noget data der er fejl, og hvad fejlen lød på, driftsvejledningen beskriver fil og database logningsformat.

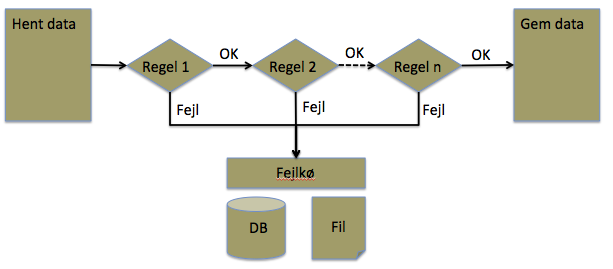
Fejlhåndtering bliver styre at regelmotoren der ligger i pakken dk.nsi.haiba.lprimporter.rules og klassen LPRRulesEngine

## Arkitektur

### Regelhåndtering

LPR Databehandleren sender LPR data gennem en række forretningsregler, som beskrevet i løsningsspecifikationen (afsnit 2.4).

Regelhåndteringen er meget simpel, da reglerne skal udføres i en bestemt rækkefølge, og de alle bliver afbrudt på samme måde i tilfælde af fejl i databehandlingen.

Som det ses ud fra figuren bliver data hentet ind og sendt til den 1. regel, går det godt sende data til 2. regel o.s.v. I alle tilfælde hvis der sker en fejl i databehandlingen vil flowet blive afbrudt og fejl detaljer vil blive sendt til fejlhåndteringsmodulet som sørger for at skrive det til en log-fil, og alternativt en tabel i databasen, hvis dette er konfigureret.

For at implementere dette flow blev Spring integration overvejet, Spring integration er godt til at holde styr på løst koblede komponenter og hvilket beskedflow der skal være mellem dem, samt håndtering af eksterne triggere såsom services, hændelser og lignede der kan igangsætte flowet.

Dog er der i denne prototype ikke behov for hverken avanceret routing, eksterne triggere (der "polles" efter data i LPR) eller afkobling, derfor er det mere optimalt at bygge flowet som enkeltstående klasser, der sørger for at behandle data og sende det videre til næste forretningsregel. Dette vil umiddelbart også gøre fejlfinding nemmere da man kan koble en debugger til og følge dataflowet gennem denne, i stedet for at følge beskeder/hændelser som Spring Integration understøtter.

Skulle der efter prototype fasen (der slutter 1/4-2013) vise sig at være et mere avanceret behov for routing, afkobling eller eksterne trigger, kan forretningslogikken fra ovenstående regler genbruges da de er lavet som enkeltstående klasser, det som skal til er at Spring Integration konfigurationen sættes op.

# Særlige forhold

## Personhenførbare data

Data fra LPR indeholder personnumre, og derfor personhenførbare. Dette personnummer bæres igennem regelbehandling og efterfølgende gemt i HAIBA indlæggelsesdatabasen.

Personnumre vil aldrig blive skrevet i logfiler, alt data der logges er referencenumre fra LPR (Record nummer), som gør at personer med relevant viden og rettigheder kan slå op i LPR og se data og derved sammenholde det med fejlbeskeden.

## Ændring/tilføjelse af regler og datamodel

Skal der laves ændringer eller tilføjelser til forretningsreglerne, skal disse kodes og testes først. Der vil blive lavet konfigurationsparametre hvor det giver mening (såsom justering af tidsintervaller o.l.), men disse kan ikke ændre på det grundlæggende som forretningsreglen implementerer.

Ligeledes hvis der er ændringer til datamodellen, så skal dette også kodes, da den interne model i LPR databehandleren er bundet hårdt op på datamodellen, det er et bevidst valg at der ikke er indført et konfigurerbart afkoblingslag mellem databasen og koden, f.eks. via Hibernate, da dette erfaringsmæssigt giver et stort overhead, og indfører en kompleksitet der projektet mere omkostningstungt, både under udvikling, men også under senere drift.

Dog er der lavet en række views som sammenholder indlæggelsesdata med data fra FGR (Fælles Grunddata), i sådanne tilfælde kan det være muligt man kan nøjes med at justere viewet.

# Fysiske Datamodeller

LPR Databasen indeholder 3 tabeller med data, hvor T\_ADM er den overordnede tabel med de administrative kontakt data.

HAIBA indlæggelsesdatabasen er de behandlede data, hvor kontakter er gjort til indlæggelsesforløb, Indlaeggelsesforloeb tabellen er den overordnede tabel med de behandlede forløb. HAIBA databasen indeholder desuden en række views, som er sat op for at gøre det nemmere for den efterfølgende databehandling via eksempelvis SAS.

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

## LPR Database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| T\_ADM | Indeholder data for den administrative del af en kontakt, heriblandt personnummer, indlæggelses og udskrivningstidspunkt. |
| T\_DIAG | Indeholder diagnose typer og koder |
| T\_PROCEDURER | Indeholder data om undersøgelser og operationer, bl.a. tidspunkter og koder |

## Indlæggelsesdatabase

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| Indlaeggelser | Indeholder processeret data fra T\_ADM tabellen, dette er beriget med metadata såsom navne på sygehuse, afdelinger m.v. |
| Diagnoser | Indeholder processeret data fra T\_DIAG tabellen, ligeledes beriget med metadata såsom navne på diagnoser. |
| Procedurer | Indeholder processeret data fra T\_PROCEDURER tabellen, ligeledes beriget med metadata såsom navne på operationer og undersøgelser. |
| Indlaeggelsesforloeb | Referencetabel, der viser sammenhængen mellem de enkelte indlæggelser i.h.t. forretningsreglerne. |
| LPR\_reference | Referencetabel, der viser sammenhængen tilbage til kontakterne i LPR T\_ADM tabellen. Bl.a. hvis der er kontakter der ikke er gyldige i f.t. forretningsreglerne. |
| Regelfejlbeskeder | Tabel til de fejlbeskeder der kommer fra behandlingen af LPR data, data er en kopi af de data fejlhåndteringsmodulet logger. |

Views:

Der er lavet en række views, som vil gøre det lettere for business intelligence programmer som SAS at hente data, disse er lavet som views for ikke at have data kopieret rundt flere gange i tabellerne.

|  |  |
| --- | --- |
| View | Beskrivelse |
| IndlaeggelsesForloebsOversigt | Indeholder et overblik of et indlæggelsesforløb med første indlæggelsesdata og sidste udskrivningsdata samt personens CPR nummer. |
|  |  |

# 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-LPRimporter.git

## Byggemiljø

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

efter Maven layout konventionen.

For at bygge LPR Databehandler systemet, 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 LPR 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

### Indlæggelsesdatabase

Databaseskema for indlæggelser ligger i database/HAIBA-Indlaeggelser.sql filen, den indeholder de tabeller, views m.m. hvortil data fra LPR skal transformeres og eksporteres

### LPR database

Databaseskema for LPR ligger i database/HAIBA-LPR.sql filen, den indeholder de tabeller som LPR data vil ligge i. Data fra LPR er det grunddata som skal transformeres og eksporteres til indlæggelsesdatabasen.

SSI har lavet et regneark med nogle få testdata der dækker de overordnede forretningsregler.

I pakken dk.nsi.haiba.lprimporter.testdata er der lavet en lille utility som kan generere SQL ud fra CSV filer (som er eksporteret fra regnearket) - disse kan indsættes in LPR databasen og derved har man testdata til udviklingsbehov.

SSI har desuden sørget for anonymiseret produktionsdata som testdata, der er tilgængeligt på testmiljøet, således der kan testes med de rette datamængder-

## Test

Installationen kan verificeres ved at eksekvere LPR 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

LPR 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 LPR 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 LPR 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 LPR 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 LPR 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

LPR 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-LPRimporter/blob/master/doc/Design, arkitektur og udviklerguide-1.0.docx](https://github.com/trifork/HAIBA-LPRimporter/blob/master/doc/Design,%20arkitektur%20og%20udviklerguide-1.0.docx)

| Version | Dato | Ændring | Ansvarlig |
| --- | --- | --- | --- |
| 1.0 | 2013-01-03 | Initielt dokument | Trifork  Kjeld Froberg |
|  |  |  |  |