CHALMERS


# Specificering och implementering av en ny modul till webapplikationen aiai.se

*Master of Science Thesis in the Computer Science and Engineering*

CHRISTIAN SIPOLA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, May 2009

Specificering och implementering av en ny modul till webapplikationen aiai.se

Christian Sipola



Examiner: Jan Skansholm

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2009

## Sammanfattning

Det huvudsakliga målet med modulen var att vidareutveckla webapplikationen Aiai för att kunna rikta sig till fler användare. I detta fallet var det att utvidga Aiai till att klara av att ha företag som registrerade, och inte bara privatpersoner. Modulen uppfyller de krav som ställs på den och den har börjat användas i skarpt läge av två stora företag. Arbetet med modulen kommer att fortsätta efter att detta examensarbeteavslutats för att uppfylla de ytterligare önskemål om funktioner som företagen har.

## Abstract

The main goal of the module was to develop the web application Aiai further to address more users. In this case the task was to expand Aiai to be able to handle companies and not just private persons. The module satisfies the requirements that where set up. The module is now used actively by two big companies. The work with the module will continue after this master thesis is finished to fulfil the additional requests for functions made by the companies.

# Förord

Denna rapport behandlar ett examensarbete i Datorteknik vid Instutionen för Data och Informations- teknik, Chalmers Tekniska Högskola.

Projektet har utförts hos Kaustik AB i Göteborg.

Tack till min examinator Jan Skansholm som var till stor hjälp när jag skulle strukturera upp arbetet.

Tack även till min handledare Erik Ljungberg på Kaustik AB som gett mig goda råd och kritik om olika lösningar.

Ett stort tack även till David på Kaustik AB som varit ett bollplank under hela examensarbetet och hjälpt till med att strukturera databaser och gränssnitt så det passar ihop med resten av Aiai.

Slutligen ett tack till de många utvecklarna av Eclipse och Doctrine som utvecklat två programvaror och släppt dem gratis och med öppen källkod. Utan dessa program skulle implementeringen av de olika iterationerna av modulen aldrig kunnat gått så snabbt och smidigt som det gjorde.

# Innehåll

# Kapitel 1

# Bakgrund

## 1.1 Inledning

Aiai är en webbapplikation för att administrera personlig assistans. Personlig assistans innebär att en person med funktionshinder, kallad brukare i denna rapport, får ersättning från Försäkringskassan för att täcka kostnaderna för att ha personliga assistenter anställda. Detta regleras enligt LSS (Lagen om stöd och service till vissa funktionshindrade) som kom 1994. En brukare blir tilldelad ett visst antal timmar som brukaren kan schemalägga för assistenter. Aiai har utvecklats sedan 2003 och används i nuläget av enskilda brukare som inte tillhör ett assistansföretag. Brukarna kan schemalägga assistenterna, godkänna tidrapporterna som skapas från schemat och även göra lönearbete för assistenterna. Assistenterna i sin tur kan önska pass i schemat och ändra i tidrapporter. Mer om detta finns i nästa kapitel Bakgrund till Aiai.

## 1.2 Syfte

Det finns flera stora assistansföretag i Sverige som administrerar den personliga assistansen åt brukaren. De har tidigare inte kunnat använda Aiai och detta examensarbetets uppgift är att anpassa Aiai för att användas av assistansföretag och uppfylla deras krav. Detta är inte så specifikt och därmed kommer en stor del av arbetet bestå av att analysera vad som behöver ändras och byggas ut för att Aiai ska klara av assistansföretagen och deras krav. Det mest grundläggande kravet som måste uppfyllas är att assistansföretagen liksom enskilda brukare ska kunna schemalägga assistenter och se tidrapporterna för brukarens assistenter.

## 1.3 Avgränsning

En avgränsning är att assistansföretagen inte ska göra lönearbetet via Aiai utan tidrapporterna kommer att matas in manuellt i ett externt löneprogram. Det finns idag flera stora bokförings- och löneprogram som har mycket funktionalitet och det skulle vara

alltför stort för att implementeras i Aiai.

## 1.4 Mål

Mina mål med det här examensarbetet är att:

- Utveckla en webapplikation med en från början fastställd projektmodell.
- Ha en ordentlig planering i början och hålla sig till den. Försöka att fokusera mer på specifikation och mindre på implementering. Med andra ord mer fokus, analys och specificering och mindre fokus på programmering.
- Skapa kravspecifikationer ihop med kund.
- Skriva testfall för de viktigaste egenskaperna och köra de testen efter att ändringar skett i kod.
- Utveckla funktioner som är till för att uppfylla användarnas krav, varken mer eller mindre. Inga extra funktioner om de inte behövs.
- Inte göra saker för modulära, komplexa och utbyggbara i onödan.

Många av de här målen är inspirerade av systemutvecklingsmetodiken Extrem programmering som beskrivs i boken Extreme Programming[2] och i Wikipedia-artikeln om ämnet[8].

# Kapitel 2

# Bakgrund till Aiai

Aiai är namnet på webapplikationen aiai.se som Kaustik AB utvecklar. Adressen till webbapplikationen är https://www.aiai.se. Aiai är ett administrativt verktyg för personlig assistans.

## 2.1 Kaustik AB

Kaustik AB är ett mindre IT-företag i Göteborg som utvecklar webbapplikationer riktade till personer med funktionshinder. Aiai är deras första produkt men det finns planer på andra, bland annat en i samarbete med Allmänna Arvsfonden.

## 2.2 Allmänt om Aiai

Aiai har utvecklats sedan 2003 och har körts i skarpt läge sedan 1 januari 2007. Kaustik började ta betalt för Aiai från och med den 1 januari 2009 och började i och med det att marknadsföra sig i olika sammanhang och öka antalet användare. Användarantalet växer nu stadigt för varje månad. En exakt användarsiffra är dock inte något som Kaustik redovisar utåt för tillfället.

## 2.3 Personlig assistans

Personlig assistans innebär att en person med funktionsnedsättning, kallad brukare, får ersättning från Försäkringskassan eller hemkommunen för att kunna ha assistenter anställda som hjälper till i det dagliga livet. Beroende på funktionsnedsättning erhålls olika antal timmar från Försäkringskassan eller kommunen som kan kan ha assistenter schemalagda. Detta är reglerat enligt LSS-lagstiftningen[6].

Om antalet timmar ligger under viss gräns är det kommunen som bekostar assistansen och administrerar den. Annars är det Försäkringskassan som bekostar assistansen och då är det upp till brukaren själv att välja hur den vill att assistansen ska administreras. Antingen kan brukaren administrera allt själv eller välja att låta ett assistansföretag

mot en kostnad administrera allt eller vissa delar. Om brukaren gör något själv så ska brukaren vara registrerad som en enskild firma och arbetsgivare hos Bolagsverket och Skatteverket och lämna in alla blanketter och deklarationer som hör ett sådant till.

## 2.4 Kunder hos Aiai

Aiais kunder är brukarna, dvs. personerna med funktionshinder. Assistenter använder Aiai gratis och kan vara registrerade utan att vara kopplad till en brukare. Modulen som detta examensarbete beskriver kommer att utvidga Aiai till att även ha assistansbolag som kunder.

## 2.5 Vad kan brukaren och assistenterna använda Aiai till?

En brukare kan använda Aiai till att i princip administrera allt med den personliga assistansen. En brukare har en eget konto på Aiai och brukarens assistenter har också varsitt konto. Dessa assistenter är kopplade mot brukaren. Brukaren kan lägga till och ta bort kopplade assistenter själv. Nedan följer ett vanligt scenario om hur brukarna och assistenterna använder Aiai.

### 2.5.1 Schema

En brukare börjar med att lägga tomma pass i ett schema i Aiai. Assistenterna kan därefter önska hur bra ett pass skulle vara att jobba på och även hur många timmar per månad den vill jobba.

Brukaren kan efter detta gjort använda Aiais automatiska schemaläggare som placerar ut assistenter i schemat anpassat efter assistenternas önskemål. Brukaren kan även göra detta själv eller göra egna ändringar i ett färdig schema. Aiai håller koll på de viktigaste arbetstidsreglerna(ATL) i kollektivavtalen och varnar om man schemalagt någon för många timmar eller utan tillräcklig vilotid mellan passen.

Brukaren och assistenterna kan under hela tiden se schemat i en vecko- eller månadsvy.

### 2.5.2 Tidrapport

Passen från schemat förs automatiskt över till assistenternas tidrapporter. Aiai räknar ut hur många av timmarna som är vanliga timmar och hur många som är OB(obekväm arbetstid) och jour. Efter att en assistent jobbat alla passen i en månad kan den gå in i månadens tidrapport och godkänna den alternativt göra mindre ändringar i den först.

Brukaren kan sedan antingen godkänna en assistants tidrapport eller neka godkännande med ett meddelande till assistenten. Brukaren kan få ut tidrapporterna som PDF och skriva ut dem och lämna in dem underskrivna till Försäkringskassan.

### 2.5.3 Lönearbete

När brukaren godkänt alla assistenters tidrapporter kan de göra lönearbetet för månaden. I lönearbetet skapas lönespecifikationer som skickas ut till assistenterna. Det genereras även utbetalningslistor, en skattedeklaration till Skatteverket och en ifylld assistansersättningsblankett till Försäkringskassan. Aiai genererar även kontrolluppgifter för lönen en gång per år som skickas in till Skatteverket.

Aiai genererar även ett ekonomiskt sammandrag där brukaren kan se hur mycket assistansen kostat per månad och hur många timmar man använt av de som blivit tilldelade från Försäkringkassan.

## 2.6 Teknisk bakgrund

Aiai är skrivet i PHP och körs på en LAMP-plattform. Det betyder PHP och MySQL på en Apache-webbserver i operativsystemet Linux.

### 2.6.1 Ramverk

Aiai körs med ett egenutvecklat ramverk som bygger på strukturen Model View Controller(MVC), eller modell-vy-kontroller på svenska. Allt i ramverket är objektorinterat och det enda som inte är klasser är vyernas mallar som skrivs i XHTML och PHP. Denna uppdelning innebär att alla HTML-kod är samlat i vymallarna, all logik och databashantering finns i modellerna och ihopkoppling av allt sker i kontrollerna. Vymallarna i sin tur kan bestå av flera vymallar och det kan därför vara flera vymallar som skapar ett grafiskt gränsnitt på en sida. Varje vymall kan därmed ses som en grafisk komponent som kan återanvändas i flera olika grafiska gräsnsnitt.

#### Struktur

Ramverket följer i mångt och mycket den struktur som andra moderna PHP-ramverk har. Varje kontroller är en klass och i varje klass finns metoder för att utföra det som behövs för att visa en sida.

Lite förenklat är kontrollerstrukturen uppbyggd enligt vilken URL som anropats. Om URL:en är www.aiai.se/schema/manad så anropas metoden manad i klassen Schema_Controller av ramverket. Denna metod hämtar i sin tur data från modeller, skapar en instans av en av HTML-vyerna och exporterar datan från modellerna till vyerna. Ramverket hämtar sedan vyerna från kontrollern, skapar en ren XHTML-version av vyn och exporterar den som text till browsern.

Ett exempel på en kontroller syns i figur 2.1. I exemplet syns en förenklad version av kontrollern för schemat i Aiai. Först kommer metoden 'vecka' som hanterar veckoschemat. Efter det följer metoden 'manad' som hanterar månadsschemat. I metoden skapas en instans av modellen Shift_Model som hanterar passen (shifts på engelska) i schemat. Från den hämtas en lista med alla pass som sedan sparas som en instansvariabel i en instans av schemavyn. Denna vy skapas av vymallen vars sökväg skickas som argument

till konstruktorn till vy-objektet. Vymallen består som tidigare nämnt av XHTML och PHP.

Slutligen returnerar metoden vy-objektet som nu består av ett komplett månadsschema som kan skickas till browsern och visas upp för användaren.

```
class Schema_Controller extends Controller{

  public function vecka(){
    ...
  }

  public function manad(){

    $shift = new Shift_Model();

    $allShifts = $shift->getAllShifts();

    $view = new View('schema/manad');
    $view->allShifts = $allShifts;

    return $view;
  }

  ...

}
```

Figur 2.1: Exempel på en kontrollerklass i ramverket

### Händelser

En händelse på Aiai kallas det när en användare exempelvis postar ett formulär från browsern eller klickar på en länk som ska utföra något mer än att bara visa en annan sida. Ett exempel på en händelse är när man klickar på logga in på framsidan på Aiai och loggar in. Ett annat exempel är när man klickar på radera symbolen på ett pass i schemat för att ta bort ett pass. Vid en händelse skickas en sträng med i POST-variablen 'action' som beskriver vilken händelse som ska utföras. Händelsen hanteras av en händelsekontroller, kallad Action_Controller i Aiai. Det är en klass med en metod för varje händelse och följer samma struktur som de andra kontrollerna.

### 2.6.2 Namngivning

Internt i Aiai används begreppet 'ass' för assistenter, 'bru' för brukare, 'sam' för assistansföretag (samordnare är ett annat ord för assistansföretag) och 'per' för personal. Dessa benämningar kommer att synas i olika figurer och i källkod.

### 2.6.3 Doctrine

Aiai använder Doctrine[5] som Object-Relational Mapping(ORM). Doctrine kan ses som ett abstraktionslager mellan ramverket och databasen. Det är ett plattformsoberoende lager vilket innebär att det man skriver för Doctrine teoretiskt ska fungera oavsett vilken databasmotor man använder. Denna abstraktionen innebär att Doctrine används för att göra SQL-förfrågningar till databasen och skapa objekt av datan från databasen.

Doctrine består av ett bibliotek av PHP-klasser. Detta bibliotek laddas in i Aiais ramverk och kan användas i alla modeller i Aiai.

Doctrine underlättar kopplingen mot databasen avsevärt eftersom man inte behöver skriva sina egna SQL-förfrågningar utan använder Doctrines egna syntax DQL som är lättare att hantera. Det som framförallt underlättar är att Doctrine skapar objekt av datan som returneras från databasen.

#### Doctrine Query Languange och skapandet av objekt

I Doctrine används DQL, Doctrine Query Language, för att göra queries mot databasen. Nedan följer ett exempel på en DQL och skapandet av ett objekt. Själva förfrågningen och skapandet sker i ett steg:

```
$user = Doctrine_Query::create()
->from('User_Model u')
->where("u.id = 69)
->leftJoin('u.Adr')
->leftJoin('u.UserLevel')
->leftJoin('u.UserPriv')
->fetchOne();
```

I exemplet hämtas användare 69 från databastabellen 'user' och datan används för att skapa ett User_Model-objekt i $user. Adressdata, användarnivåer och rättigheter hämtas via many-to-many joins från adresstabellen, användarnivåtabellerna och rättighetstabellerna och läggs till objektet. Detta görs med de 3 olika leftJoin-anropen.

Kopplingar mot databastabeller finns specificerade i metoder setTableDefinition() och setUp() i klassen User_Model som finns i Appendix D.1.1 och följer den standard som Doctrine satt upp.

#### Dataobjekt

Varje objekt som skapas av Doctrine är en instans av klassen Doctrine_Record, eller har den som förfader. Varje tabell i databasen man skapar är kopplad mot en sådan klass

och varje rad som returneras från en databasfrågan skapar ett dataobjekt av den typen. I objektet finns data från varje kolumn sparad som en instansvariabel i dataobjektet. Ett exempel är att det finns en användartabell 'user' i en databas med kolumner id, username, firstname och lastname. Detta exempel syns i tabell 2.1. I tabellen syns två exempelanvändare och och värdena för varje kolumn.

| user | | | |
|---|---|---|---|
| id | login | firstname | lastname |
| 1 | js | jon | smith |
| 2 | dv | dave | jones |

Tabell 2.1: Databastabellen 'user'

Varje rad som hämtas av Doctrine från den tabellen kommer att bli ett dataobjekt av klassen User_Model och varje dataobjekt kommer att ha id,login, firstname och lastname som instansvariabler som kan läsas.

#### Samlingar av dataobjekt

Doctrine kan även hantera skapandet av en lista av dataobjekt om en databasförfrågan returnerar mer än en rad. Då sparas alla dataobjekt i ett annat objekt kallat Doctrine_Collection. Den kan hanteras som en fält i PHP och det går att iterera igenom samlingen och därmed gå igenom alla dataobjekten.

### 2.6.4 Bogusanvändare

I Aiai finns några användare som är bogusanvändare, dvs. de representerar inga riktiga personer. Dessa används vid testning och utveckling av Aiai och är tänkta att representera användarna på ett bra sätt. De kommer att förekomma i de exempel och mätningar som görs i detta examensarbete.

# Kapitel 3

# Analys

Innan specificering skedde en analys av vilka funktionskrav som assistansföretagen har. Det ingick en kontinuerlig kommunikation med olika assistansföretag vad de tyckte om olika funktioner och vilka krav och synpunkter de har om hur de skulle vilja administrera personlig assistans. Denna kommunikation fortgick genom hela projektet.

Det som främst skulle analyseras var hur de olika användartyperna skulle vara kopplade mot varandra och hur olika uppgifter och rättigheter skulle kunna delegeras till olika användare. Detta var i fokus för det skulle till stor grad bestämma hur nya databastabeller skulle konstrueras och hur de olika modellerna i MVC-strukturen skulle specificeras.

# Kapitel 4

# Metodbeskrivning och utförande

Arbetet bestämdes att utföras i ett iterativt förlopp. De utvecklingsprinciper som används på Kaustik förespårkar snabba implementeringar av ideér och iterativa förbättringar av dem. Aiai är utvecklat i PHP med ett eget MVC-baserat ramverk och Doctrine för Object-relational mapping. Det möjliggjorde snabba implementeringar som kunde testas av vissa assistansföretag för att få en snabb återkoppling.

Arbetet började med kravspecifikation, databasdesign, specifikation av modeller och specificering av HTML-vyer och sedan en implementering av detta. Denna cykel upprepades iterativt 4 gånger under arbetets gång.

## 4.1 Kravspecifikation

Kommunikationen med företagen och analys av arbetet gav upphov till de kravdokument som finns i appendix A. Dessa uppdaterades efterhand som nya iterationer av implementeringen visade upp problem som inte kommit fram tidigare.

## 4.2 Användarkopplingar och delegeringar

Strukturen för hur användare skulle kopplas arbetades fram efter att användarkraven satts ihop med assistansföretagen. Dessa syns senare i resultatförtecking och i appendix A.

## 4.3 Databasdesign

Efter att kravdokumenten och andvändarkopplingarna specificerats började arbetet med att designa databasen. Efter en analys framställdes två olika förslag till databasdesign. De två varianterna hette 'alternativ 1: Fler tabeller' och en annan 'alternativ 2: färre tabeller'. Dessa visas som figurer i Appendix C. Kriterier som användes vid databasdesign var lättast att arbeta med, bäst anpassat för Doctrine, bra namn på tabeller och kolumner, minska risk för att databasen blir inkonsekvent och bra prestanda.

Eftersom databasen körs med MySQL togs även hänsyn till de rekommendationer som finns i MySQL:s referensmanual[3].

## 4.4 Modellspecifikation

Det var ett antal modeller enligt MVC-strukturen som behövde skapas för att hantera datat från databasen.

### 4.4.1 Doctrine-modeller

För att få modulen att passa ihop med Doctrine ska en Doctrine-modell skapas per skapad databastabell. Själva grunden till modellen kan man skapa genom ett verktygsskript i Doctrine som läser tabelldefinitionen från databasen och skapar phpkoden för klassen. Denna klass måste sedan komplemetteras med de kopplingar som finns mot andra tabeller genom att fylla i setUp-metoderna i klasserna. Efter det lades ytterligare metoder och instansvariabler till för att kunna hantera och modifiera datan och utföra algoritmer och dylikt.

### 4.4.2 Andra modeller

När Doctrine-modellerna var specificerade specificerades modellerna för att hantera datan från Doctrine-modellerna till kontrollerna i MVC-strukturen. Målet var att försöka använda så många så många som möjligt av de befintliga modellerna inom Aiai. Alla nya modeller som skapades skulle vara lätta att förstå genom klass- och metodnamn. Ett antal förslag till modeller och namngivning togs fram. Ihop med personal på Kaustik valdes de förlag ut som bäst passade ihop med Aiais befintliga modell- och namngivningsstruktur.

## 4.5 Grafiska gränssnitt

De grafiska gränsnitt som skulle skapas i modulen gjordes för att passa in med resterande grafiska gränssnitt i Aiai. Andra riktlinjer som användes var enligt de designregler som finns i kapitel 7 i Human Computer Interaction[1].

## 4.6 Prestanda

Prestandakravet på modulen i kravspecifikation var att den skulle vara skulle vara likvärdig med resterande Aiai. Prestandamätningar valdes att göras för minnesanvändning på webbservern, exekveringstid på servern och sidstorlek som skickas till browsern. Detta skulle mätas på månadsschemat, veckoschemat och tidrapporterna.

## 4.7 Testning

Från början var tanken att skriva testfall och modeller som de kunde testas med för det mesta av funktionaliteten för modulen. Detta ändrades dock till att bara täcka de funktioner som var kritiska och som var tvungna att fungera för att användarna skulle kunna logga in och ha rätt rättigheter.

# Kapitel 5

# Resultat

## 5.1 Kravuppfyllnad

Modulen uppfyller de krav som ställts upp i början av projektet och som finns i Appendix A.

## 5.2 Användarkopplingar och delegeringar

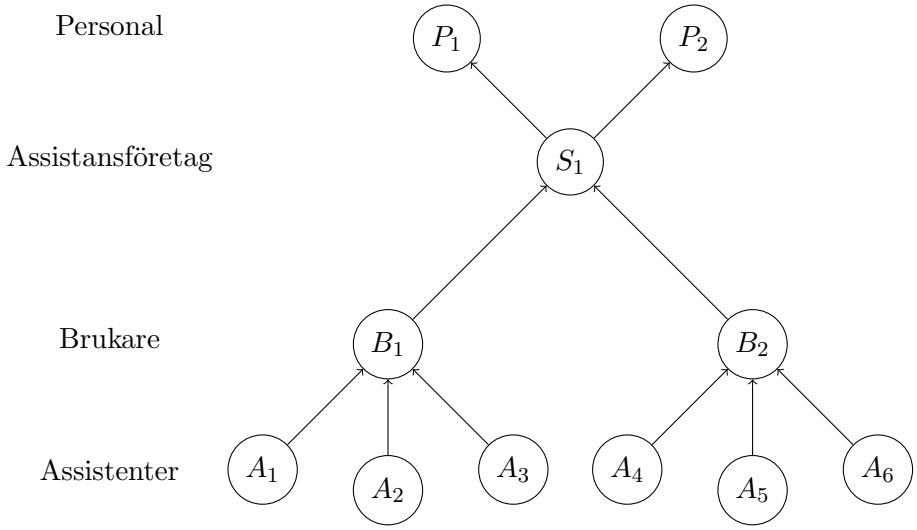
I figur 5.1 syns hur kopplingarna till de nya användartyperna personal och samordnare blev i modulen. Bokstaven 's' i ringarna är en förkortning för samordnare vilket är en intern benämning inom Aiai för ett assistansföretag. Figuren beskriver att assistenter är kopplade till brukare och brukarna i sin tur är kopplade mot ett assistansföretag. Assistansföretaget i sin tur har personal anställd. Man kan se assistansföretaget som de centrala i figuren och de andra användarnivåerna som cirklar utåt.

I figur 5.2 syns hur delegeringarna blir inom den nya modulen. De två exemplen innebär att schema för brukare #1 och brukare #2 sköts av personal #1 och att tidrapport för brukare #1 och brukare #2 sköts av personal #1.

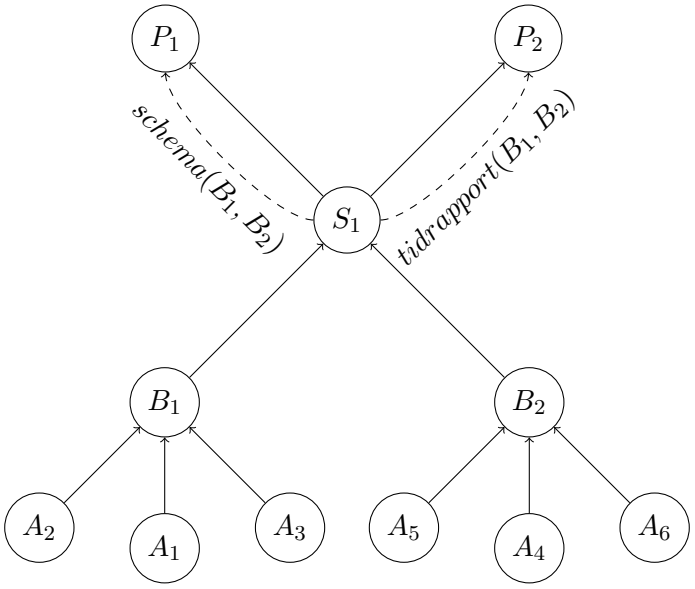
## 5.3 Databasdesign

Det alternativ som valdes för databasdesign var 'alternativ 1: Fler tabeller' och syns i figur 5.3. De databastabellerna som syns används för att realisera de kopplingar och delegeringar som beskrevs i sektion 5.2. I databasen så har varje typ av koppling har en egen tabell. Varje tabell består av ett id och två användarid. Varje rad i tabeller user_bru_sam, user_sam_per beskriver koppling mellan två användare. Varje rad i tabeller user_bru_sam_priv_map, user_sam_per_priv_map beskriver delegering av en rättighet för en användare till en användare.

Ett exempel är att det finns en koppling mellan personal #69 och assistansföretag #33, dvs. en personal jobbar hos ett visst assistansföretag. Då kommer det finnas en rad



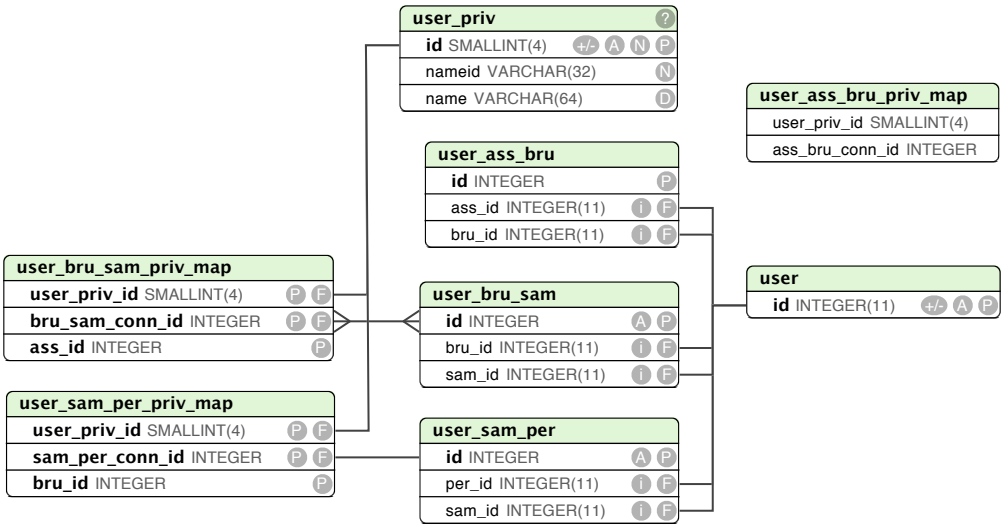

Figur 5.1: Koppling mellan olika användartyper.




Figur 5.2: Delegeringar mellan olika användare och användartyper.

i databastabellen user_sam_per med per_id satt till 69 och sam_id satt till 33 och id satt till en unikt värde för tabellen.

Om det sedan finns en delegering av en brukare till personalen kommer det finns en rad i user_sam_per_priv_map där sam_per_conn_id är satt till id:t som kopplingsraden har i user_sam_per, bru_id är satt till brukarens användarid och user_priv_id är satt till




Figur 5.3: Databas som hanterar data för modulen

id:t som rättigheten 'alla rättigheter' har i tabellen user_priv.

Det fungerar på liknande sätt i de andra tabellerna som beskriver kopplingar och delegeringar mellan assistansföretag och brukare och brukare och assistenter.

## 5.4 Modeller

### 5.4.1 Nya modeller

De viktigaste nya modellerna som skapades var de som syns i tabell 5.4.1. De beskriver de modeller som behövdes för att representera de nya tabellerna som skapades i databasen och även de modeller som behövdes för att hantera alla DQL-kod för de olika användarna. De modeller som hanterar DQL-koden är utility-klasser. Det innebär klasser med bara statiska metoder och en privat konstruktor. Det går därmed inte att skapas instanser av klassen utan den är bara till för att samla ihop metoder i en gemensam fil.

Anledningen till att ingen Sam_Model skapades var att ingen loggar in som ett assistansföretag.

### 5.4.2 Modifierade modeller

Det var ett antal modeller i Aiai som var tvungen att modifieras för att kunna hantera den nya modulen.

#### User_Model

En modell som modifieras var User_Model som representerar användartabellerna, bland annat 'user', i databasen. De nya användarnivåerna fick läggas in i dess olika metoder

| | |
|---|---|
| UserBruSam_Model | Doctrinemodell för att representera och hantera användarkopplingstabellen user_bru_sam. |
| UserBruSamPriv_Model | Doctrinemodell för att representera och hantera användardelegeringstabellen user_bru_sam_priv. |
| UserSamPer_Model | Doctrinemodell för att representera och hantera användarkopplingstabellen user_bru_sam. |
| UserSamPerPriv_Model | Doctrinemodell för att representera och hantera användardelegeringstabellen user_bru_sam_priv. |
| Per_Model | Modell för att hantera alla DQL:s som en personal använder |
| Bru_Model | Modell för att hantera alla DQL:s som en brukare använder |
| Ass_Model | Modell för att hantera alla DQL:s som en brukare använder |

Tabell 5.1: Tabell över nya modeller

och även några nya metoder lades till.

#### ActiveUser_Model

En annan viktig modell som modifierades var ActiveUser_Model som representerar den inloggade användaren i Aiai. Den fick modfieras för att klara av att hantera att även en personal skulle kunna vara en typ av inloggad användare. Modellen fick även anpassas för att kunna hantera att en användare byter mellan två olika användarnivåer.

#### Priv_Model

Modellen som representerar en användares rättigheter, Priv_Model, fick även modifieras. Dels för att kunna hantera de nya rättigheterna som den nya modulen innehåller och Den även för att kunna hantera de nya användarnivåerna assistansföretag och personal. Det innebar att ett några nya klasskonstanter lades till för varje ny rättighet och några metoder ändrades och lades till.

## 5.5 Kontroller

Nya kontroller lades och och flera befintliga kontroller modifierades för att anpassa Aiai till den nya modulen.

### 5.5.1 Nya kontroller

Det skapades en ny kontroller, Administrera_Controller för att hantera administrationssidan där en personal med administrativa rättigheter kan delegera rättigheter till andra brukare.

### 5.5.2 Modifierade kontroller

Flera kontroller i Aiai fick modifieras för den nya modulen. De flesta modifikationer innebar att ett nytt fall fick skapas i en metod som hanterade en viss sida. I detta fall lades kod som hanterar hur modeller och vyer ska sättas upp för att sidan ska kunna visas och användas av en personal. Denna kod tog bland annat reda på vilken brukare som var aktiv brukare för personalen och om personalen har rättighet för att visa den aktuella sidan för den aktuella brukaren.

Nedan följer de de kontrollers där de viktigaste ändringarna gjordes.

#### Installningar_Controller

Denna kontroller modifierades för att admin för Aiai ska kunna skapa ett nytt assistansföretag i Aiai och koppla brukare och personal till den. Den modiferades även för att personal skulle kunna se och ändra samma inställningsidor som en brukare kan se.

#### Schema_Controller

Kontrollern för att hantera schemat modifierades för att en personal ska kunna se månads och veckoschemat för en brukare. Även metoderna som visar sidorna för att skapa, kopiera pass och kontrollera ATL för schemat modifierades för att en personal skulle kunna se och använda dessa sidor.

#### Tidrapport_Controller

Kontrollern för att hantera tidrapporten modiferades på liknande sätt som schemakontrollern. Kontrollerns metoder fick anpassas för att en personal skulle kunna visa och godkänna en aktiv brukares tidrapporter.

#### Header_Controller

Denna kontroller hanterar hur sidhuvudet på Aiai ska se ut, dvs. den översta delen av sidan på Aiai där man ser menyn, vem som är inloggad och aktiv brukare (ifall man är inloggad som en personal).

## 5.6 Prestanda

För att mäta prestanda gjordes prestandamätningar av när vissa sidor visades med modulen jämfört med när samma sidor visades utan att modulen användes. Det betyder att när modulen användes så visades sidan från ett personal-konto med en viss brukare som den aktiva. För att visa en sida med modulen avslagen visades sidan inloggad direkt som den specifika brukaren.

I tabell 5.2 syns prestandaresultat från mätningar som gjordes på de viktigaste sidor på Aiai där modulen används. Mätvärden som redovisas gjordes med bogusbrukaren Jens Jonsson som aktiv brukare och representerar en genomsnittlig brukare på Aiai.

Boguspersonalen Test Personal användes som personalkonto vid testningarna och aktivt företag var bugusföretaget Testföretaget AB. Månaden som testades var mars månad som var genomsnittligt schemalagt.

Mätvärden är ett genomsnitt av 10 mätningar. Mätningarna gjordes på en Macbook Pro 2.4GHz med 2GB ram och med MAMP installerat lokalt med PHP 5.2.5, MySQL 5.0.41 och Apache 2.0 .

| Sida | Exekveringstid(ms) | Sidstorlek(KiB) | Minnesanvänding(MiB) |
|---|---|---|---|
| Månadsschema | 756(719) | 574(575) | 13,75(13,75) |
| Veckoschema | 482(485) | 433(432) | 12,5(12,25) |
| Tidrapport | 2534(2491) | 388(389) | 11,25(11,25) |

Tabell 5.2: prestandaresultat där modulen användes. Värden inom parantes är då modulen inte användes

## 5.7 Grafiskt gränssnitt

Det skapades nya vyer och grafiska gränssnitt för den nya kontroller, Administrera_Controller, som skapades. En hel del vyer modifierades och de hörde till de kontrollers som modifierades. T.ex modifierades schemakontrollern och det innebar att även vyerna för schemat modifierades.

### 5.7.1 Administrera

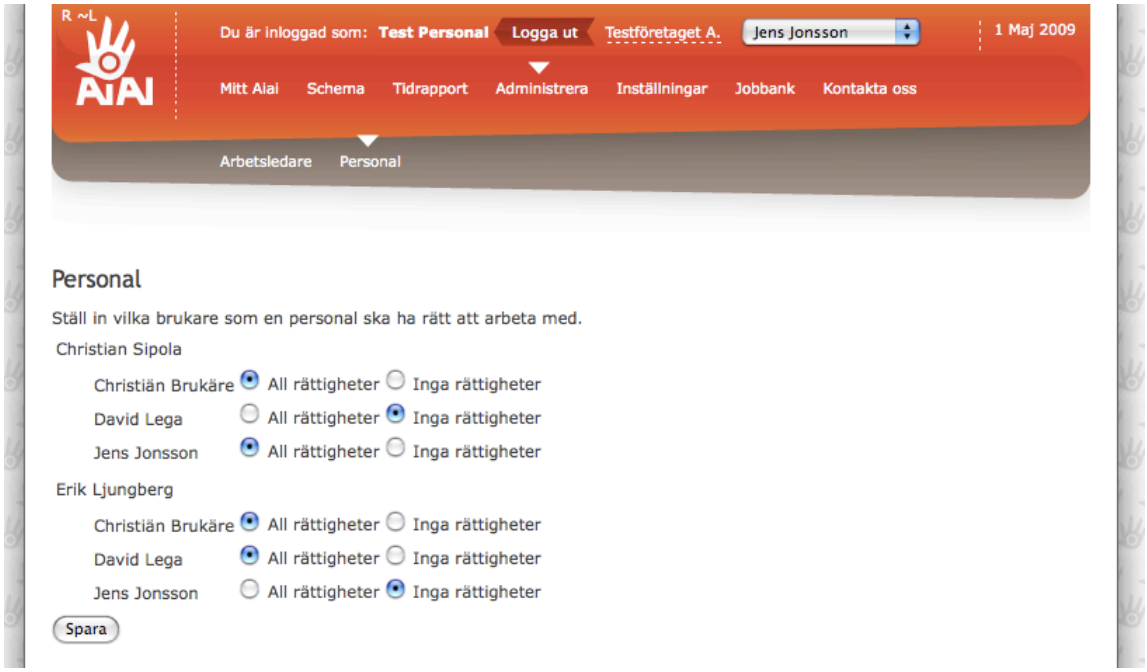
Den nya vyerna och det grafiska gränssnitt de bygger upp som skapades för Administrera_Controller syns i figur 5.4 där en personal med adminrättighet kan delegera vilka brukare de andra i personalen ska administrera.

### 5.7.2 Schema

I figur 5.5 syns hur veckoschemat ser ut när en personal är inloggad. I headern syns vilket företag personalen tillhör och en drowdown-menyn där personalen kan välja aktiv brukare. I schemat syns brukarens, i det här fallet bogusbrukaren Jens Jonsson, pass och vilken assistent som jobbar det passet. Personalen kan i den här vyn bland annat ändra assistent för ett visst pass, ta bort ett pass och ändra start- och sluttid.

### 5.7.3 Admin

Det grafiska gränsnittet för admin anpassades för att kunna administrera företag och dess personal. I figur 5.6 syns en del av vyn där admin kan lägga till personal och brukare till ett företag och välja vilken personal som ska ha huvudansvar.




Figur 5.4: Vy där brukare delegeras till personalen

## 5.8 Testning

Modulen har testats både med de testfall som skrivits och användartester och testen har godkänts. Testfallen finns beskrivna med källkod i appendix D.3.

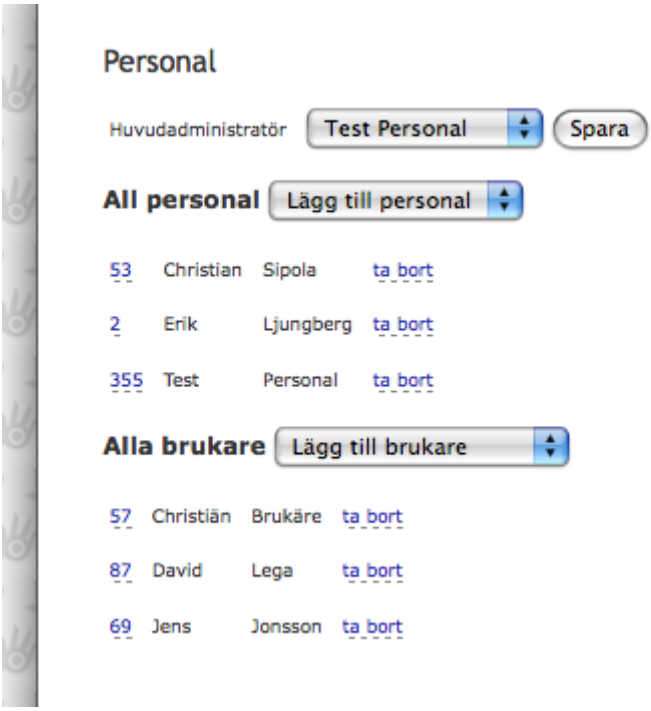
## 5.9 Användning

Ett stort assistansföretag, Stil Assistans, och ett lite mindre assistansföretag, Jevia Assistans, använder modulen skarpt. Stil började använda modulen den 8 april 2009 och Jevia började använda modulen 16 april 2009. Två större assistansföretag till kommer att börja använda modulen under juni 2009.

## 5.10 Användarvänlighet

Enligt den respons som Aiai fått ifrån assistansföretag så verkar de nöjda med modulen och har efter några kortare instruktioner kunnat använda modulen utan problem.


Figur 5.5: Schemavy för personal




Figur 5.6: Administrera ett företag som admin

# Kapitel 6

# Diskussion

## 6.1 Kravuppfyllnad

De modeller och databastabeller som implementerades uppfyllde även kraven på att vara lättanvända och lätta att felsöka. Det alternativ som valdes för att representera användarkopplingarna och delegeringarna i databasen var inte det alternativ som var mest generellt eller det som var mest utbyggbart, men det visade sig vara det som var lättast att arbeta med och som innebar att den ganska komplicerade strukturen ändå var lätt att hålla reda på.

Delegeringen valdes till att ligga per brukare, istället för per rättighet, och det ser ut att ha varit ett bra val eftersom det var det som alla assistansföretag efterfrågar.

## 6.2 Databasdesign

Ordningen BruSam och PerSam i namngivningen(istället för SamBru och PerSam) är för att jag tyckte att det är brukaren som 'ger' administrationen av sina assistenter till assistansföretaget och inte tvärtom. På samma sätt är det assistansföretaget som 'ger' brukaren till sin personal. Eftersom det är standard att man skriver i ordningen från och till blir det då en bra logik bakom valet. Detta ansåg också ansvariga på Kaustik vara en bra konvention.

Det skulle egentligen bara behövts två användarid och ha dem som primary key i databastabellerna user_bru_sam och user_sam_per eftersom de alltid är unika. Däremot finns det delegeringstabeller som använder id:t som foreign key och jag ansåg det vara bättre att ha ett id som foreign key än att ha två användarid i en annan tabell som foreign key. Då slipper man problem vad som händer om det finns två användarid i delegeringstabellen och dessa två inte finns i kopplingstabellen.

### 6.2.1 Val av databasdesign

Det databasalternativ, alternativ 1, som valdes gjordes det eftersom det ansågs uppfylla de villkor som sattes upp i metodbeskrivningen. För att databasförfrågningarna via

Doctrine ska vara lätta att skriva och deras resultat lätta att avläsa ska man inte gör 'left joins' mellan mer än 3 tabeller i taget och detta uppfylldes med det valda alternativet.

Det andra alternativet,alternativ 2, var mer generellt än det valda och hade t.ex inte användarnivåer som kolumnnamn i tabellerna. Om alternativ 2 valts skulle det varit krångligare att skriva databasförfrågningar och läsa resultatet från databasen. Detta eftersom det inte direkt framgått vilken användarnivå ett användarid i en kolumn tillhör och att det då skulle varit lätt att blanda ihop de olika användarid:na i en tabell i databasförfrågningar och resultat från databasen.

Alternativ 2 skulle även varit mer generellt än alternativ 1 eftersom det skulle gått att lägga till en ny användarnivå i Aiai utan att behöva skapa flera nya tabeller, vilket krävs för alternativ 1. Analysen jag gjorde visade dock att det inte finns något behov att skapa nya användarnivåer i framtiden. Om det ändå skulle inträffa är det relativt enkelt att skapa nya tabeller och modeller kopplade till dem genom att kopiera de tabeller och modeller som skapades för denna modulen och bara ändra tabell och kolumnnamn enligt de namnmönster som finns för tabellerna.

## 6.3 Modeller

### 6.3.1 Nya modeller

De nya Doctrine-modellerna som skrevs gjordes enligt den mall som Doctrine har och enligt hur databastabellerna specificerades. Det var inga större problem med att utforma dem när väl databasstrukturen var vald.

De nya modellerna som skrevs för att hantera DQL-kod gjordes medvetet ganska enkla. Detta eftersom jag ansåg att DQL:en i sig går att beskriva på ett sådant generellt sätt att det inte behövdes tillföra någon ytterligare abstraktionlager mellan Doctrine och Aiai. Ett nytt abstraktionslager skulle kunnat underlättat anropen mot databasen och fått en mer konsekvent struktur på den kod som används för att anropa databasen. Jag ansåg dock att DQL placerat i metoder i en utility-klass (en klass med bara statiska metoder) var tillräckligt lätt att arbeta med och tillräckligt konsekvent. Ett nytt abstraktions lager skulle tvärtom kunnat göra arbetet med databasen onödigt komplext.

### 6.3.2 Modifierade modeller

Alla ändringar av Aiais befintliga modeller innebar att resterande Aiai fick genomgå en fullständig testning. Eftersom detta var en tidskrävande och omständligt process hölls antal ändringar på ett minimum. Det mesta i modellerna var skrivna såpass generellt att det gick att behålla mer kod än väntat när nya användarnivåer och rättigheter lades till för modulen. Däremot blev allting ett per nivåer komplexare med de nya användarnivåerna och möjligheten för en användare att ha flera användarnivåer och byta aktiv användarnivå.

## 6.4 Kontroller

### 6.4.1 Nya kontroller

Den nya kontrollern som skrev, Administrera_Controller, innebar inte att några större val behövdes göras. Kontrollern skrevs enligt samma mall som de tidigare fanns på Aiai.

### 6.4.2 Modifierade kontroller

De kontroller som modifierades gjordes alla på ett liknande sätt för att passa för personal. de flesta kontroller och deras vyer var anpassade från början att kunna hantera att brukaren i ett schema var någon annan än den som var inloggad för tillfället, vilket underlättade arbetet att anpassa kontrollerna för personal.

## 6.5 Testning

De test som skrev blev ganska specifika och testar i stort sett att olika användare har de egenskaper och rättigheter som de är tänkta att ha.

## 6.6 Prestanda

Prestandadelen av modulen var inte något högprioriterat men de sidor som använder modulen ligger på sidstorlek och exekveringstid som är på samma nivå som när modulen inte används. Minnesanvändningen ligger också på samma nivåer som de andra sidorna på aiai.

Exekveringstiden är i högsta laget rent generellt på Aiai och är något som kommer att prioriteras högre framöver. En förbättring av prestandan i Aiai kommer att förbättra alla delar av Aiai och även denna modulen.

Anledningen till mätningarna inte gjordes på Aiai riktiga webserver var att det är en server som alltid är belastad och det var svårt att få mätvärden som inte varierade alltför mycket mellan de olika mätningarna. Testservern som användes har samma version av all PHP,MySQL och Apache som den riktiga webservern.

Mätningarna gjordes även för andra brukare men de resultaten skiljde sig inte nämnvärt för de som redovisades.

## 6.7 Grafiska gränssnitt

De grafiska gränsnitten är simpla men de fyller sitt syfte och är lättförståeliga enligt de användare som kör modulen.

# Kapitel 7

# Slutsats

Den arbetsmetod som valdes fungerade bra för att implementera en ny modul till en relativt komplex webapplikation.

jag tycker själv att jag lyckades uppfylla de mål jag satt jag upp och att de val som gjordes under arbets gång var bra gjorda och kommer att stå sig bra.

Det var en stor fördel att stora delar av Aiais modeller var skrivna modulärt och generellt från början vilket underlättade arbetet med att modifiera dem för att även passa för den nya användartypen personal. Överhuvudtaget blev arbetet med modulen lättare än väntat eftersom befintliga delar av Aiai var välstrukturerade och genomtänkta och lätta att modifiera.

## 7.1 Framtiden för modulen

Arbetet med modulen kommer att fortsätta efter detta examensarbete. Det finns ytterligare önskemål om funktioner från företagen som kommer att arbetas vidare med. Två av dem är att antingen kunna exportera lönedata som XML från Aiai till windowsbaserade löneprogram eller kunna använda Aiais befintliga lönedel för ett assistansföretag. En annan funktion som kommer specificeras och implementeras framöver är en översikt över saker som inte står rätt till som t.ex arbetstidskriterier (ATL) som inte är uppfyllda och varningar för månader som inte är schemalagda eller som innehåller fler timmar än vad en brukare har blivit tilldelad från Försäkringskassan.

# Litteraturförteckning


[1] Gregory D. Abowd Russerl Beale Alan Dix, Janet Finlay. *Human-Computer Interaction*. Pearson Education, third edition, 2004.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999.

[3] MySQL. Mysql 5.0 reference manual, http://dev.mysql.com/doc/refman/5.0/en/index.html.

[4] PHPUnit. http://www.phpunit.de/, 05 2009.

[5] Doctrine Project. http://www.doctrine-project.org/, 05 2009.

[6] Socialstyrelsen. Lag om stöd och service till vissa funktionshindrade – lss, http://www.socialstyrelsen.se/publicerat/2007/9456/html.htm.

[7] Sommerville. *Software Engineering*. Addison-Wesley, 8 edition, 2007.

[8] Wikipedia. http://sv.wikipedia.org/wiki/extrem programmering, 05 2009.

# Bilaga A

# Användarkrav

Denna kravspecifikation kommer enbart beskriva kraven för den nya modulen till Aiai, inte kraven för de resterande och befintliga delarna av applikationen. Dock kommer det att finnas en viss gråzon, eller överlapp, mellan den nya modulen och resten av Aiai. Uppdelningen här är gjord enligt den som beskrivs i kapitel 6 i Software Engineering [7].

## A.1 Generella

- Det ska finnas användartyperna personal, assistansföretag, brukare, personal och admin.
- Admin sköts av adminpersonal på Kaustik och är något som Aiais kunder inte har tillgång till.
- En personal kan få adminrättighet för ett assistansföretag av admin.
- Det ska finnas en ihopkoppling mellan personal, assistansföretag, brukare och assistenter enligt figur A.1.
- Det ska finnas möjlighet i databas till delegeringar mellan personal, assistansföretag, brukare och assistenter enligt figur A.2.
- En personal kan bli delegerad att sköta administration för en brukare. Personalen får då rättighet att göra allt för en brukare.
- Det ska inte gå att logga in som en assistansföretag. Istället är det personal med adminrättigheter som sköter assistansföretaget.
- En användare i systemet av typ personal, brukare eller assistenter (alla förutom assistansföretag och admin) ska alltid representeras av en riktig person med personnummer och kontaktuppgifter. Dvs. det ska t.ex. inte finnas en generell löneansvarig användare utan det är en rättighet som i så fall tilldelas till en användare.

### A.1.1 Inställningar - Kopplingar

- Admin ska kunna ändra kopplingarna mellan assistansföretag och brukare och mellan assistansföretag och personal.
- En personal med administrativa rättigheter ska kunna delegera vissa brukare till en personal.
- En personal kan inte ändra vilket assistansföretag den är kopplad mot utan det gör admin.
- En personal kan lista alla brukare och assistenter som den har fått delegerade till sig av sin assistansföretag
- En personal kan välja ut vilka brukare den vill ska vara aktiva

### A.1.2 Möjliga kopplingar

- En assistent kan vara kopplade till 0 eller flera brukare
- En brukare kan vara kopplade till 0 eller flera assistenter
- En brukare kan vara kopplade till 0 eller 1 assistansföretag
- Ett assistansföretag kan vara kopplade tilll 0 eller fler brukare
- Ett assistansföretag kan vara kopplade till 0 eller fler personal
- En personal kan ha vara kopplade till 0 eller fler assistansföretag

### A.1.3 Inställningar - Brukare

Det ska finnas en sida med ett gränssnitt där en personal med adminrättighet kan se alla brukare.

### A.1.4 Generellt för personal

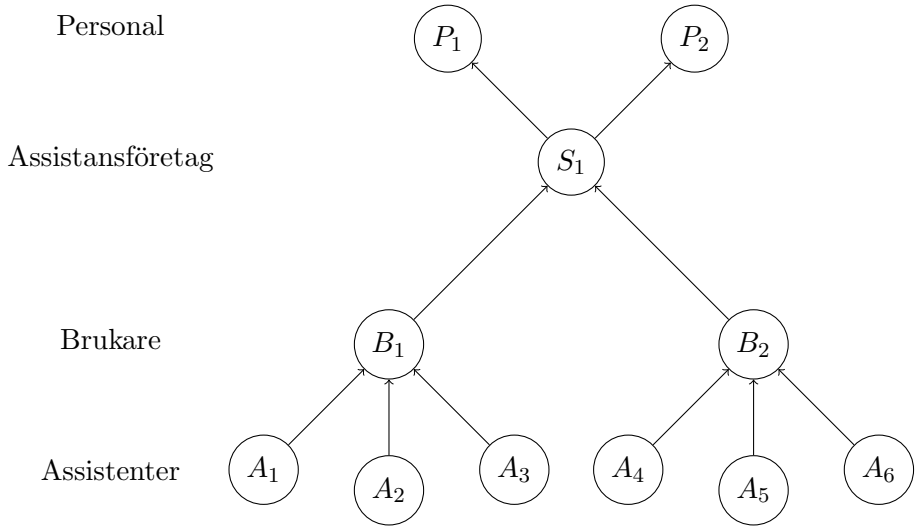
- En personal ska att ha ett visst antal tillgängliga brukare som den blivit delegerad.
- Tillgängliga brukare ska synas i en selectbox i menyn när den loggat in i Aiai.
- Med hjälp av selectboxen ska personal kunna välja aktiv brukare.
- När personal ser på scheman ska schemat för den aktiva brukaren visas.
- När personal ser på tidrapporter ska tidrapporter för den aktiva brukaren visas.

### A.1.5 Schema

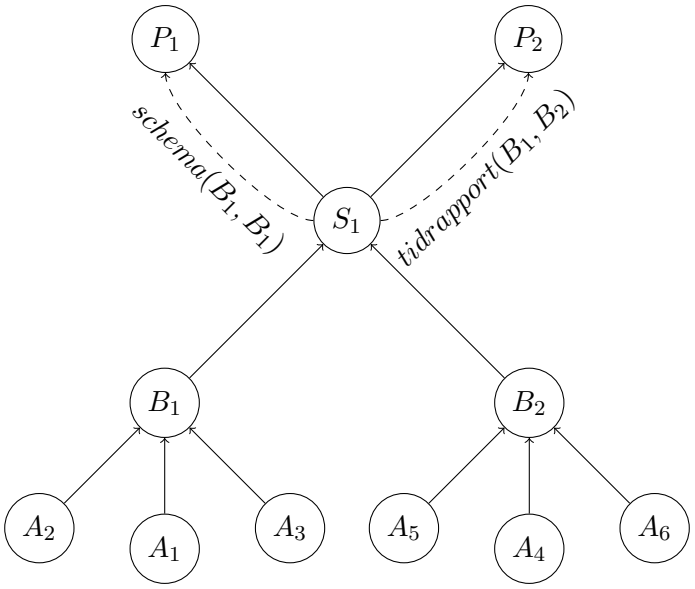
- En personal kan se schema för alla sina aktiva brukare. Man ser schema för en brukare i taget men det går lätt att byta mellan de aktiva brukare genom en popupruta alternativt en select-box.
- En personal kan kontrolla ATL för en brukare i taget.
- En personal kan lägga schema, ändra i schemat på samma sätt som en brukare kan. Allt detta görs för en brukare i taget.

### A.1.6 Tidrapport

- En personal kan se tidrapporter för alla assistenterna för den aktiva brukaren.
- En personal ser summering av antal timmar sammanlagt för den aktiva brukaren.




Figur A.1: Koppling mellan olika användartyper.




Figur A.2: Delegeringar mellan olika användare och användartyper.

# Bilaga B

# Systemkrav

## B.1 Funktionskrav

Kopplingarna och delegeringarna ska lösas genom att använda en databas som beskrivs i figur C.1 i appendix C. Denna databas ska klara av att delegera vissa rättigheter för en brukare till en personal. Detta är för att i framtida versioner kunna göra ett gränssnitt som kan hantera en mer detaljerad delegering.

## B.2 Icke-funktionella krav

### B.2.1 Produktkrav

#### Tillförlitlighet

Modulen ska ha samma driftpålitlighet som resten av Aiai och följa samma SLA-modeller.

#### Effektivitet

Exekveringstiden, minnesanvändning, och sidstorleken för en sidvisning i modulen ska i princip samma som i resterande delen av Aiai då modulen inte används.

#### Personlig integritet

En användare ska bara ha tillgång att se den data som tillhör användaren eller som den har rättighet till. En personal ska bara kunna se schema och tidrapporter för de brukare och assistenter som den blivit tilldelad. En personal med administratörsrättigheter ska bara kunna information om det egna företaget och dess användare. Det ska dock gå att se assistenter i jobbanken som inte tillhör ett företag eller som företaget har tillåtit ska ses utåt. Ingen användare ska kunna se andra andra användares personnummer, förutom administratör för ett företag och de löneansvariga.

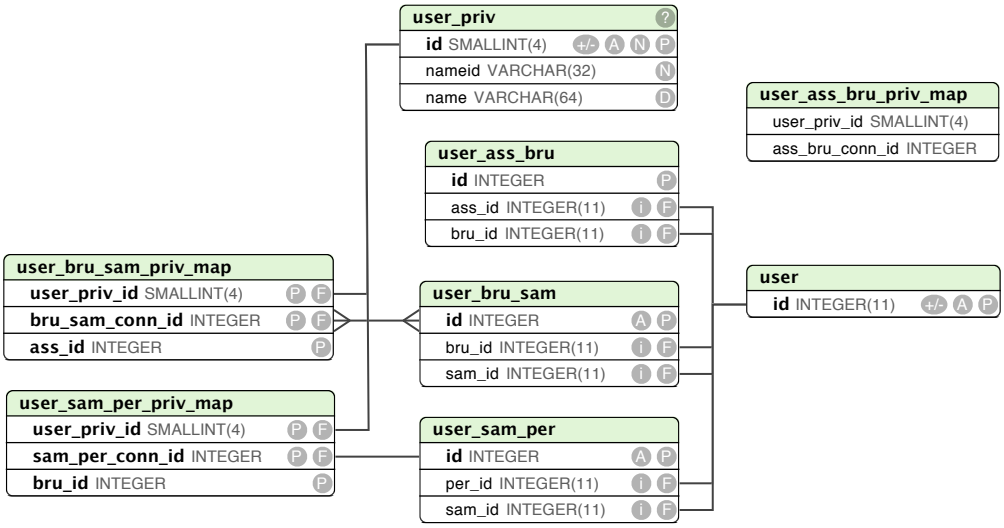
**Säkerhet**

Modulen ska (på samma sätt som resten av Aiai) vara säker mot de vanligaste attackerna på internet. All kod ska vara skriven för att stå emot SQL-injections, XSS-atacker och det ska inte gå att visa en sida eller utföra en åtgärd som man inte har rättighet till. Det krävs inloggsnamn och lösenord för att logga in som en användare.

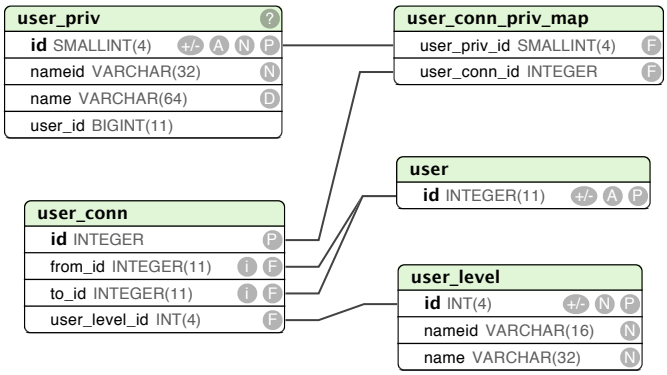
# Bilaga C

# Databas

Nedan följer de två databasvarianter som var förslag till den slutgiltiga databasdesignen. De beskriver den delen av databasen som utgör kopplingen och delegeringen mellan användare på Aiai. Symbolförklaring till databaskolumnerna i de två figurerna finns i tabell C.1.




Figur C.1: Variant 1: Fler tabeller




Figur C.2: Variant 2: Färre tabeller

| Symbol | Förklaring |
|---:|---:|
| P | Primary key |
| F | Foreign key |
| +/- | unsigned |
| N | Not null |
| A | Auto increment |

Tabell C.1: Symbolförklaring för databaskolumner

# Bilaga D

# Källkod

I det här appendixet finns den viktigaste källkoden som utvecklades med. De delar av filerna som är irrelevanta för modulen har klippts bort.

## D.1 Doctrine-modeller

Här följer de Doctrine-modeller som utvecklades. Varje Doctrine-modell är kopplad mot en tabell i databasen. I varje modell finns det definerat hur databastabeller är kopplad mot andra databastabeller.

### D.1.1 User_Model

```
<?
class User_Model extends Doctrine_Record{


  /**
   *
   * @var UserLevel_Model
   */
  public $level = '';


  /**
   *
   * @var Priv_Model
   */
  public $priv;


  /**
```

```
 * Setup DB tables
 */
public function setTableDefinition(){
  $this->setTableName('user');

  $this->hasColumns(
    array(
      'regdate'  => array(
        'type'     => 'timestamp',
        'ntype'     => 'datetime',
        'default'  => '0000-00-00 00:00:00',
        'length'=> null,
        'notnull'  => true,
      ),

      'pnr'  => array(
        'type'     => 'integer',
        'default'  => 0,
        'length'=> 10,
        'notnull'  => true,
      ),

      'username'  => array(
        'type'     => 'string',
        'default'  => '',
        'length'=> 30,
        'notnull'  => true,
      ),

      'password'=> array(
        'type'     => 'string',
        'default'  => '',
        'length'=> 130,
        'notnull'  => true,
      ),

      'firstname'  => array(
        'type'     => 'string',
        'default'  => '',
        'length'  => 50,
        'notnull'  => true,
      ),
```

```
      'lastname'  => array(
        'type'     => 'string',
        'default'  => '',
        'length'  => 50,
        'notnull'  => true,
      )
    )
  );

}


/**
 * Setup DB relations
 */
public function setUp(){
  $this->hasMany(
    'UserPriv_Model as UserPriv',
    array(
      'local'  => 'user_id',
      'foreign'=> 'user_priv_id',
      'refClass'  => 'UserPrivMap_Model'
    )
  );
  $this->hasMany(
    'UserPrivMap_Model as UserPrivMap',
    array(
      'local'  => 'user_id',
      'foreign'=> 'user_priv_id',
    )
  );
  $this->hasMany(
    'UserSamPer_Model as SamPerMap',
    array(
      'local'  => 'id',
      'foreign'=> 'sam_id',
    )
  );
  $this->hasMany(
    'User_Model as PerSam',
    array(
      'local'  => 'per_id',
      'foreign'=> 'sam_id',
```

```
    'refClass'=> 'UserSamPer_Model'
  )
);
$this->hasMany(
  'UserSamPerPrivMap_Model as SamBruPrivMap',
  array(
    'local'  => 'id',
    'foreign'=> 'bru_id',
  )
);
$this->hasMany(
  'UserSamPer_Model as PerSamMap',
  array(
    'local'  => 'id',
    'foreign'=> 'per_id',
  )
);
$this->hasMany(
  'User_Model as SamBru',
  array(
    'local'  => 'sam_id',
    'foreign'=> 'bru_id',
    'refClass'=> 'UserBruSam_Model'
  )
);
/**
 * BRU only has max one SAM
 */
$this->hasOne(
  'UserBruSam_Model as SamBruMap',
  array(
    'local'  => 'id',
    'foreign'=> 'sam_id',
  )
);

/**
 * BRU only has max one SAM
 */
$this->hasOne(
  'UserBruSam_Model as BruSamMap',
  array(
    'local'  => 'id',
```

```
        'foreign'=> 'bru_id',
      )
    );
    $this->hasOne(
      'UserAdr_Model as Adr',
      array(
        'local'=> 'id',
        'foreign'=> 'userid'
      )
    );
    $this->hasMany(
      'UserLevel_Model as UserLevel',
      array(
        'local'=> 'user_id',
        'foreign'=> 'level_id',
        'refClass'=> 'UserLevelMap_Model'
      )
    );
    $this->hasMany(
      'Shift_Model as Shift',
      array(
        'local'=> 'id',
        'foreign'=> 'userid',
      )
    );
    $this->hasMany(
      'ShiftAss_Model as ShiftAss',
      array(
        'local'=> 'id',
        'foreign'=> 'userid',
      )
    );
  }
}
?>
```

### D.1.2 UserSamPer_Model

```
<?
/**
 * @version $LastChangedDate: 2009-03-10 15:12:28 +0100 (ti, 10 mar 2009) $
 *          $LastChangedRevision: 1078 $
 * @since 09 mar 2009
 * @author  $LastChangedBy: christian $
```

```
 */
class UserSamPer_Model extends Doctrine_Record{

    public function setTableDefinition(){

        $this->setTableName('user_sam_per');
        $this->hasColumn('id', 'integer', 4, array('type' => 'integer', 'length' => 4,
                        'primary' => true,'autoincrement'=>true));
        $this->hasColumn('sam_id', 'integer', 4, array('type' => 'integer',
                        'length' => 4));
        $this->hasColumn('per_id', 'integer', 4, array('type' => 'integer',
                        'length' => 4));

        $this->index('sam_per',array('fields'=>array('sam_id','per_id'),
                                     'type'=>'unique')
        );

    }

    public function setUp(){

      $this->hasMany(
        'UserSamPerPrivMap_Model as PrivMap',
        array(
          'local'  => 'id',
          'foreign'=> 'sam_per_conn_id',
        )
      );

      $this->hasOne(
        'User_Model as UserPer',
        array(
          'local'  => 'per_id',
          'foreign'=> 'id',
        )
      );
    }

}
```

### D.1.3 UserSamPerPrivMap_Model

```
<?
/**
```

```
 * @version $LastChangedDate: 2009-04-30 16:24:29 +0200 (to, 30 apr 2009) $
 *          $LastChangedRevision: 1272 $
 * @since 09 mar 2009
 * @author  $LastChangedBy: christian $
 */
class UserSamPerPrivMap_Model extends Doctrine_Record{

  public function setTableDefinition(){

        $this->setTableName('user_sam_per_priv_map');
        $this->hasColumn('user_priv_id', 'integer', 2, array('type' => 'integer',
                                                'length' => 2, 'primary' => true));
        $this->hasColumn('sam_per_conn_id', 'integer', 4, array('type' => 'integer',
                                                  'length' => 4, 'primary' => true));
        $this->hasColumn('bru_id', 'integer', 4, array('type' => 'integer',
                                       'length' => 4, 'primary' => true));

  }

  public function setUp(){

      $this->hasOne(
        'UserPriv_Model as Priv',
        array(
          'local'  => 'user_priv_id',
          'foreign'=> 'id',
        )
      );
      $this->hasOne(
        'UserPriv_Model as Bru',
        array(
          'local'  => 'bru_id',
          'foreign'=> 'id',
        )
      );

      $this->hasOne(
        'UserSamPer_Model as Conn',
        array(
          'local'  => 'sam_per_conn_id',
          'foreign'=> 'id',
        )
      );
```

```
  }

}
```

### D.1.4 UserBruSam_Model

```
<?
/**
 * @version $LastChangedDate: 2009-03-18 15:51:16 +0100 (on, 18 mar 2009) $
 *          $LastChangedRevision: 1116 $
 * @since 09 mar 2009
 * @author  $LastChangedBy: christian $
 */
class UserBruSam_Model extends Doctrine_Record{

  public function setTableDefinition(){

    $this->setTableName('user_bru_sam');
    $this->hasColumn('id', 'integer', 4, array('type' => 'integer', 'length' => 4,
                     'primary' => true,'autoincrement'=>true));
    $this->hasColumn('bru_id', 'integer', 4, array('type' => 'integer', 'length' => 4));
    $this->hasColumn('sam_id', 'integer', 4, array('type' => 'integer', 'length' => 4));

    $this->index('bru_sam',array('fields'=>array('bru_id','sam_id'),'type'=>'unique'));

  }

  public function setUp(){

    $this->hasOne(
      'User_Model as Bru',
      array(
        'local'  => 'bru_id',
        'foreign'=> 'id',
      )
    );

    $this->hasOne(
      'User_Model as Sam',
      array(
        'local'  => 'sam_id',
        'foreign'=> 'id',
      )
```

```
    );

  }

}
```

### D.1.5 UserBruSamPrivMap_Model

```
<?
/**
 * @version $LastChangedDate: 2009-03-10 15:12:28 +0100 (ti, 10 mar 2009) $
 *          $LastChangedRevision: 1078 $
 * @since 09 mar 2009
 * @author  $LastChangedBy: christian $
 */
class UserBruSamPrivMap_Model extends Doctrine_Record{

    public function setTableDefinition(){

        $this->setTableName('user_bru_sam_priv_map');
        $this->hasColumn('user_priv_id', 'integer', 2, array('type' => 'integer',
                         'length' => 2, 'primary' => true));
        $this->hasColumn('bru_sam_conn_id', 'integer', 4, array('type' => 'integer',
                         'length' => 4, 'primary' => true));
        $this->hasColumn('ass_id', 'integer', 4, array('type' => 'integer',
                         'length' => 4, 'primary' => true));

    }

}
```

## D.2 Utilitymodeller med DQL-kod

### D.2.1 Per_Model

```
<?
/**
 * @version $LastChangedDate: 2009-04-30 16:42:57 +0200 (to, 30 apr 2009) $
 *          $LastChangedRevision: 1274 $
 * @since 10 mar 2009
 * @author  $LastChangedBy: christian $
 */
class Per_Model {
```

```
/**
 * Utility class
 */
private function __construct() {}

/**
 * @param $sam_id
 * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
 * @return array|Doctrine_Record|Doctrine_Query
 */
public static function getAllPer($sam_id,$hydrate = Doctrine::HYDRATE_ARRAY){

  $bru = Doctrine_Query::create()
    ->from('User_Model u INDEXBY u.id')
    ->leftJoin('u.PerSam s')
    ->where('s.id = ?',$sam_id)
    ->orderBy('u.firstname')
  ;

  if($hydrate == FALSE){
    return $bru;
  }else{
    $bru = $bru->execute(array(),$hydrate);
  }
  return $bru;

}

/**
 * @param $sam_id
 * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
 * @return array|Doctrine_Record|Doctrine_Query
 */
public static function getAllPerWithPrivAndBru($sam_id,$hydrate = Doctrine::HYDRATE_ARR

  $bru = Doctrine_Query::create()
    ->from('User_Model u INDEXBY u.id')
    ->leftJoin('u.PerSamMap psm')
    ->leftJoin('psm.PrivMap pm')
    ->leftJoin('pm.Priv p')
    ->leftJoin('pm.Bru b')
    ->where('psm.sam_id = ?',$sam_id)
    ->orderBy('u.firstname, p.name')
```

```
  ;

  if($hydrate == FALSE){
    return $bru;
  }else{
    $bru = $bru->execute(array(),$hydrate);
  }
  return $bru;

}

/**
 * Returns array with all sam with id as keys
 * @param $sam_id
 * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
 * @return array|Doctrine_Record|Doctrine_Query
 */
public static function getAllBru($sam_id,$hydrate = Doctrine::HYDRATE_ARRAY){

  $bru = Doctrine_Query::create()
    ->from('User_Model u INDEXBY u.id')
    ->leftJoin('u.BruSamMap s')
    ->where('s.sam_id = ?',$sam_id)
    ->orderBy('u.firstname')
  ;

  if($hydrate == FALSE){
    return $bru;
  }else{
    $bru = $bru->execute(array(),$hydrate);
  }
  return $bru;

}

/**
 * Returns all BRU which a PER has allPrivFor
 * @param $sam_id
 * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
 * @return array|Doctrine_Record|Doctrine_Query
 */
public static function getAllPrivBru($sam_id,$per_id,$hydrate = Doctrine::HYDRATE_ARRA
```

```
  $bru = Doctrine_Query::create()
    ->from('User_Model u INDEXBY u.id')
    ->leftJoin('u.SamBruPrivMap sbpm')
    ->leftJoin('sbpm.Conn c')
    ->where('c.sam_id = ?',$sam_id)
    ->andWhere('c.per_id = ?',$per_id)
    ->orderBy('u.firstname')
  ;

  if($hydrate == FALSE){
    return $bru;
  }else{
    $bru = $bru->execute(array(),$hydrate);
  }
  return $bru;

}

public static function getAllBruAndAss($sam_id,$hydrate = Doctrine::HYDRATE_ARRAY){

  $bru = Doctrine_Query::create()
      ->from('User_Model u INDEXBY u.id')
      ->leftJoin('u.BruSamMap s')
      ->leftJoin('u.BruAssMap bap')
      ->leftJoin('bap.Ass')
      ->where('s.sam_id = ?',$sam_id)
      ->andWhere('bap.assi_status = ? OR bap.user_status = ?',array('accepted','accepte
      ->orderBy('u.firstname')
  ;

  if($hydrate == FALSE){
    return $bru;
  }else{
    $bru = $bru->execute(array(),$hydrate);
  }
  return $bru;

}

/**
 * Return all BRU that a PER has access to
 * @param $sam_id
 * @param $per_id
```

```
   * @param $priv Priv_Model::const
   * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
   * @return array|Doctrine_Record|Doctrine_Query
   */
  public static function getAllBruForPriv($sam_id,$per_id,$priv,$hydrate = Doctrine::HYDR

    //TODO add priv check
    $bru = Doctrine_Query::create()
      ->from('UserBruSam_Model ubs')
      ->leftJoin('ubs.Bru')
      ->where('ubs.sam_id = ?')
      ->execute(array($sam_id),$hydrate)
    ;
    return $bru;
  }

  /**
   * Returns array with all sam with id as keys
   * @param $per_id
   * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
      * @return array|Doctrine_Record|Doctrine_Query
   */
  public static function getAllSam($per_id,$hydrate = Doctrine::HYDRATE_ARRAY){

    $sam = Doctrine_Query::create()
      ->from('User_Model u INDEXBY u.id')
      ->leftJoin('u.SamPerMap usp')
      ->where('usp.per_id = ?',$per_id)
    ;
    if($hydrate == FALSE){
      return $sam;
    }else{
      $sam = $sam->execute(array(),$hydrate);
    }
    return $sam;

  }

}
```

### D.2.2 Bru_Model

```
<?
/**
```

```
 * @version $LastChangedDate: 2009-04-02 16:42:04 +0200 (to, 02 apr 2009) $
 *          $LastChangedRevision: 1183 $
 * @since 02 apr 2009
 * @author  $LastChangedBy: christian $
 */
class Bru_Model {

  /**
   * Utility class
   */
  private function __construct() {}

  /**
   * @param $bru_id
   * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
   * @return array|Doctrine_Record|Doctrine_Query
   */
  public static function getAllAssActive($bru_id,$hydrate = Doctrine::HYDRATE_ARRAY){

    $bru = Doctrine_Query::create()
      ->from('User_Model u INDEXBY u.id')
      ->leftJoin('u.AssBruMap m')
      ->where('m.userid = ?',$bru_id)
      ->andWhere('m.removed = 0')
      ->andWhere('m.inactive = 0')
      #->andWhere('m.assi_status = ? OR m.user_status = ?',array('approved','approved'))
      ->orderBy('u.lastname, u.firstname')
    ;

  if($hydrate == FALSE){
      return $bru;
    }else{
      $bru = $bru->execute(array(),$hydrate);
    }
    return $bru;

  }

  /**
   * @param $bru_id
   * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
   * @return array|Doctrine_Record|Doctrine_Query
   */
```

```
  public static function getAllAssInactive($bru_id,$hydrate = Doctrine::HYDRATE_ARRAY){

    $bru = Doctrine_Query::create()
      ->from('User_Model u INDEXBY u.id')
      ->leftJoin('u.AssBruMap m')
      ->where('m.userid = ?',$bru_id)
      ->andWhere('m.removed = 0')
      ->andWhere('m.inactive = 1')
      ->andWhere('m.assi_status = ? OR m.user_status = ?',array('accepted','accepted'))
      ->orderBy('u.lastname, u.firstname')
    ;

  if($hydrate == FALSE){
      return $bru;
    }else{
      $bru = $bru->execute(array(),$hydrate);
    }
    return $bru;

  }

  /**
   * @param $bru_id
   * @param $hydrate Doctrine::HYDRATE_* or false (returns Doctrine_Query
      * @return array|Doctrine_Record|Doctrine_Query
   */
  public static function getAllSam($bru_id,$hydrate = Doctrine::HYDRATE_ARRAY){

    $sam = Doctrine_Query::create()
      ->from('User_Model u INDEXBY u.id')
      ->leftJoin('u.SamBruMap m')
      ->where('m.bru_id = ?',$bru_id)
    ;
    if($hydrate == FALSE){
      return $sam;
    }else{
      $sam = $sam->execute(array(),$hydrate);
    }
    return $sam;

  }

}
```

## D.3 Test-modell

Testmodellen körs genom PHPUnit[4] och följer deras standard.

### D.3.1 UserPriv_Test_Model

```
<?
class UserPriv_Test_Model extends PHPUnit_Framework_TestCase {

  /**
   * @var ActiveUser_Model
   */
  private $activeUser;

  private static $originalActiveUser;

  public function setUp(){

    Error::$isTest = true; //setting activeUser to SAM generetas notice. suppress it!
    //save for later restoring
    if(is_null(self::$originalActiveUser)){
      self::$originalActiveUser = ActiveUser_Model::getInstance();
    }

  }

  public function testStil(){

    Aiai::$activeUser->id = 366; //Stil
    ActiveUser_Model::renew();
    $this->activeUser = ActiveUser_Model::getInstance();

    $this->assertEquals('SAM',$this->activeUser->whoAmI());
    $this->assertEquals('stil',$this->activeUser['username']);
    $this->assertEquals(8020110709,$this->activeUser['pnr']);
    $this->assertNotEquals('active',$this->activeUser['Acc']['status']); //not active yet

    Aiai::$activeUser->id = 403; //Lennart Ramö
    ActiveUser_Model::renew();
    $this->activeUser = ActiveUser_Model::getInstance();

    $this->assertEquals('BRU',$this->activeUser->whoAmI());
    $this->assertEquals('LRStil',$this->activeUser['username']);
```

```
  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('stil',$user['username']);

  $this->assertFalse($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnema
  $this->assertTrue($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
  #stils wants to see lon
  $this->assertTrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');

  Aiai::$activeUser->id = 422; //Irma Högberg
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('ASS',$this->activeUser->whoAmI());
  $this->assertEquals('wagner',$this->activeUser['username']);

  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('stil',$user['username']);
  $this->assertFalse($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnema
  $this->assertTrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');
}

public function testJevia(){

  Aiai::$activeUser->id = 367; //Stil
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('SAM',$this->activeUser->whoAmI());
  $this->assertEquals('jevia_assistans',$this->activeUser['username']);
  $this->assertNotEquals('active',$this->activeUser['Acc']['status']); //not active yet

  Aiai::$activeUser->id = 241; //Ingegärd Wauge
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('BRU',$this->activeUser->whoAmI());
  $this->assertEquals('jevia1',$this->activeUser['username']);

  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('jevia1',$user['username']);

  $this->assertTrue($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnemar
  $this->asserttrue($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
```

```
  $this->asserttrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');

  Aiai::$activeUser->id = 244; // Jenny ForsmanHasselgren
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('ASS',$this->activeUser->whoAmI());
  $this->assertEquals('jennyfh',$this->activeUser['username']);

  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('jevia1',$user['username']);

  $this->assertFalse($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnema
  $this->assertFalse($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
  $this->assertTrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');
}

public function testErik(){

  Aiai::$activeUser->id = 2; //Erik Ljungberg
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('BRU',$this->activeUser->whoAmI());
  $this->assertEquals('errkki',$this->activeUser['username']);

  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('errkki',$user['username']);

  $this->assertTrue($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnemar
  $this->asserttrue($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
  $this->asserttrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');

  Aiai::$activeUser->id = 25; // Niclas Jonsson
  ActiveUser_Model::renew();
  $this->activeUser = ActiveUser_Model::getInstance();

  $this->assertEquals('ASS',$this->activeUser->whoAmI());
  $this->assertEquals('niclas',$this->activeUser['username']);

  $user = $this->activeUser->getPaymentUser();
  $this->assertEquals('errkki',$user['username']);
```

```
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnema
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
    $this->assertTrue($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');
  }

  public function testMorticia(){

    Aiai::$activeUser->id = 177; //Morticia Bakken
    ActiveUser_Model::renew();
    $this->activeUser = ActiveUser_Model::getInstance();

    $this->assertEquals('BRU',$this->activeUser->whoAmI());
    $this->assertEquals('Morticia',$this->activeUser['username']);

    $user = $this->activeUser->getPaymentUser();
    $this->assertEquals('Morticia',$user['username']);

    $this->assertTrue($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnemar
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');

    Aiai::$activeUser->id = 180; // Tamara
    ActiveUser_Model::renew();
    $this->activeUser = ActiveUser_Model::getInstance();

    $user = $this->activeUser->getPaymentUser();
    $this->assertEquals('Morticia',$user['username']);

    $this->assertEquals('ASS',$this->activeUser->whoAmI());
    $this->assertEquals('tammy80',$this->activeUser['username']);

    $this->assertFalse($this->activeUser->priv->has(Priv_Model::ABONNEMANG),'har abonnema
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::LON),'har lon:');
    $this->assertFalse($this->activeUser->priv->has(Priv_Model::SMS),'har sms:');
  }

  public function testFinal(){

    ActiveUser_Model::setInstance(self::$originalActiveUser);

  }

}
```