Webucator 3.0

Brukerhåndtering og aksesskontroll for DPG 2.0

Masteroppgave

Kristian Skønberg Løvik
Institutt for informatikk
Universitetet i Bergen

18. august 2008

Forord

Denne oppgaven inngår i mitt masterstudium i programvareutvikling ved Institutt for informatikk ved Mattematisk og naturvitenskapelig fakultet ved Universitetet i Bergen.

Oppgaven omhandler brukerhåndtering, sikkerhet og aksesskontroll i Dynamic Presentation Generator og Webucator, som er sentrale systemer for gjennomføringen av fjernundervisningskurs over internett ved Universitetet i Bergen.

Jeg vil benytte anledningen til å takke veilederne mine; Khalid A. Mughal og Torill Hamre for deres innspill og gode råd underveis i masterstudiet. Jeg vil også takke hele IKT kull ’03 for fem fantastiske år som venner og medstudenter. En spesiell takk går til Bjørn Christian Sebak, Bjørn Ove Ingvaldsen og Karianne Berg som har vært gode samarbeidspartnere ved JAFU-prosjektet. Til slutt vil jeg takke familien min og min kjære Marte for utallige oppmuntringer og uforbeholden støtte til arbeidet som er nedlagt i denne oppgaven.

Bergen, juli 2008
Kristian Skønberg Løvik

Innholdsfortegnelse

Liste over eksempler

Liste over figurer

Liste over tabeller

1 Introduksjon

1.1 Historisk bakgrunn

1.1.1 Fjernundervisningen ved UiB

I 1998 startet prosjektet *The Bergen Webucator*[1] som et samarbeid mellom *Institutt for informatikk og Senter for Etter- og Videreutdanning* (*SEVU*). Fra år 2000 har utviklingen vært en del av prosjektet *Java FjernUndervisning* (*JAFU*).

1.1.2 Bergen Webucator

I 1999 fullførte Espen Haagensen sin masteroppgave[2], som det første bidraget til *The Bergen Webucator*. Siden har systemet gjennomgått en rekke restruktureringer som kuliminerte i *Webucator 2.0* i 2006. [3]. Da var *Webucator* redusert til et brukerhåndteringssystem, mens de opprinnelige andre oppgavene til systemet, og nye oppgaver, var blitt fordelt ut i fem andre delsystemer:[3]

- *DPE* – En dynamisk presentasjonsmotor som viser web-innhold i en nettleser basert på presentasjonsmønstre definert i XML.
- *RAT* – Et administrasjonsverktøy for håndtering av de ulike presentasjonene som skal vises i *DPE*.
- *SYSTEKON* – Et system for gjennomføring av interaktive prøver.
- *Innleveringssystemet* – Et system for innlevering av obligatoriske oppgaver.
- *MVNForum* – Et tredjeparts forum utgitt under GNU General Public Licence [4]

1.1.3 Ny implementasjon våren 2008

DPG – Dynamic Presentation Generator er en fellesbetegnelse for systemene *DPE* og *RAT* som er nevnt i avsnittet over.

Gjennom tre masteroppgaver[5][6][7] vil store deler av dette systemet erstattes våren 2008. Det nye systemet vil få navnet *DPG 2.0* og bestå hovedsakelig av følgende delsystemer:

- *DPE* består i sin nåværende rolle men med en helt ny implementasjon, og får navnet *PV (Presentation Viewer)*.
- *RAT* fritas for all brukerhåndtering, og resten av funksjonaliteten blir delt i to nye systemer: [5]
    - *PM (Presentation Manager)* – Et verktøy for å administrere presentasjoner.

- *PCE (Presentation Content Editor)* - Et verktøy for å redigere innholdet i presentasjoner.

Denne oppgaven tar for seg brukerhåndteringen for det nye systemet, som skal implementeres som en selvstendig webapplikasjon, *Webucator 3.0*. Det er også blitt implementert en løsning for autentisering av disse brukerne mot den nye *DPG*-en til fjernundervisningsformål, samt implementert et sikkerhetssystem for de nye systemene.

I de tilfellene det i resten av denne oppgaven vil bli referert til alle de ulike systemene som er beskrevet over, vil samlebetegnelsen *JAFU-systemene* brukes.

1.2 Bakgrunn for oppgaven

Som nevnt i forrige delkapittel har *JAFU-systemene* gjennomgått en kraftig renovering/nyutvikling våren 2008. I forkant av dette ble det avdekket en del problemstillinger som måtte tas hensyn til i utviklingen av de nye systemene. De viktigste av problemstillingene som omhandles i denne oppgaven er beskrevet nærmere i de følgende underavsnittene.

1.2.1 Manglende fungerende brukerhåndtering

Høsten 2007 var det ikke noe fungerende brukerhåndteringsapplikasjon for *JAFU-systemene*. *Webucator 2.0* manglet systemdokumentasjon og komplett kildekode. Det ble derfor besluttet å utvikle et nytt brukerhåndteringssystem fra bunnen av. Det kunne likevel tenkes å bruke deler av det gamle *Webucator*-systemet, om det var mulig. Selv om det som sagt ikke eksisterte komplett kildekode for dette systemet, var *Webucator 2.0* beskrevet i masteroppgaven til Preben Solheim[3], og mye av funksjonaliteten det tilbød var beskrevet der.

1.2.2 Duplisering av ansvarsområder

I *DPG 1.0* var en del brukerhåndteringsfunksjonalitet lokalisert i *RAT*, og en del eksternt i *Webucator*. Dette gjorde systemene vanskelig å drifte og vedlikeholde. En nærmere beskrivelse av denne funksjonaliteten finnes i delkapittel 5.3.2

Det var ønsket en klarere delegering av ansvarsområder mellom de ulike delsystemene. Blant annet medførte dette et ønske om at *Webucator* i fremtiden skulle ha ansvaret for all brukerhåndtering i gjennomføringen av kurs ved Fjernundervisningen ved Universitetet i Bergen.

1.2.3 Sikkerhet

Gjennom *DPG1.0* og *Webucator*s levetid var det blitt avslørt en del, flere av dem alvorlige, sikkerhetshull. Et eksempel kunne være at man, uten noen form for innlogging, kunne få tilgang til en XML-fil med alle systemets

administratorer bare ved å vite URL-en til denne. En nærmere beskrivelse av sikkerhetsproblematikken i *DPG1.0* og *Webucator 2.0* er gitt i kapittel 5.

Det var derfor ønskelig å utvikle *DPG2.0* og *Webucator 3.0* på en slik måte at sikkerheten ble bedre ivaretatt enn tilfellet var med *DPG1.0* og *Webucator 2.0*. I all hovedsak betydde dette at det var ønskelig å få på plass en helhetlig sikkerhetsløsning for JAFU-systemene, gjerne i form av et rammeverk som håndterer sikkerheten i applikasjonene i sin helhet.

1.2.4 Automatisering av kommunikasjon

Et problem med *DPG 1.0* og *Webucator 2.0*, var at de manglet en mekanisme for å utveksle informasjon og data uten manuell "innblanding" fra systemansvarlige.

Et eksempel på dette kunne være at man manuelt måtte eksportere tre XML-filer fra *Webucator 2.0* og importere dem i *DPG 1.0* for å utveksle informasjon om hvilke brukere som skulle ha tilgang til en gitt presentasjon (nettside). Dette ledet igjen til et ønske om en bedre løsning for en ny versjon av systemet. Optimalt sett betydde dette en løsning der systemene kunne kommunisere seg i mellom uten "innblanding" fra systemansvarlige.

1.3 Mål for denne oppgaven

I hovedsak er denne oppgavens mål å prøve å løse problemstillingene som ble omtalt i forrige delkapittel (1.2). Dette betyr i klartekst at det skal:

- Implementeres et nytt brukerhåndteringssystem, *Webucator 3.0*, for å kunne bruke *DPG 2.0* til fjernundervisningsformål,. Systemet skal håndtere kurs og kursinstanser og dets brukere, og tilby ønsket funksjonalitet for å administrere disse.
- Systemet skal utvikles som en selvstendig webapplikasjon og på en slik måte at *DPG 2.0* fritas for alle oppgaver rundt brukerhåndtering ved bruk i fjernundervisningssammenheng.
- Sikkerhetshull avdekket i *DPG 1.0* skal i størst mulig grad unngås og det skal etterstrebes å implementere en helhetlig sikkerhetsløsning for systemet, gjerne ved hjelp av et rammeverk for dette.
- Det skal også implementeres en ny løsning, optimalt sett automatisert, for utveksling av informasjon mellom *DPG 2.0* og *Webucator 3.0*.

1.4 Organisering av dokumentet

Kapittel 2

Kapittel 2 gir en gjennomgang av de eksisterende applikasjonene som fantes ved JAFU-prosjektet før nyutviklingen som har funnet sted våren 2008, og som har relevans for denne oppgaven. Dette gjelder applikasjonene *Dynamic Presentation Generator 1.0* og *Webucator 2.0*. Dette kapittelet tar ikke opp

aksesskontroll for brukere og sikkerhet i disse systemene, da dette blir gjennomgått i kapittel 5.

Kapittel 3

Dette kapittelet gir en kortfattet beskrivelse av den nye utgaven av *DPG*, versjon 2.0, som ble utviklet parallelt med *Webucator 3.0*. Nok en gang er problemstillinger rundt aksesskontroll og sikkerhet utelatt, da det er beskrevet i kapittel 6.

Kapittel 4

Dette kapittelet er viet det nye brukerhåndteringssystemet som ble utviklet i forbindelse med denne oppgaven: *Webucator 3.0*. Kapittelet tar opp systemets arkitektur og oppbygging, samt ser på design av brukergrensesnitt og ulike funksjoner som tilbys. Igjen er aksesskontroll og sikkerhet i hovedsak utelatt og viet et eget kapittel (Se kapittel 6).

Kapittel 5

Dette kapittelet tar for seg den eksisterende løsningen for aksesskontroll av brukere og sikkerhet som fantes i forkant av nyutviklingen av *DPG* og *Webucator* våren 2008.

Kapittel 6

Dette kapittelet tar for seg den nye løsningen for aksesskontroll av brukere og sikkerhet som denne oppgaven presenterer.

Kapittel 7

Dette kapittelet inneholder en kort oppsummering og evaluering av arbeidet som er utført, og gir i tillegg en del forslag til hva det kan jobbes videre med i tilknytning til denne oppgaven.

2 Eksisterende løsning

2.1 DPG 1.0

Dette underavsnittet (2.1) er i store deler utformet i samråd med Karianne Berg, Bjørn Ove Ingvaldsen og Bjørn Christian Sebak. Deres masteroppgaver [7][6][5] *omhandler utviklingen av DPG 2.0. De har i oppgavene et felles bakgrunnskapittel, som beskriver DPG 1.0. Denne beskrivelsen er omfattende. Under følger de delene av bakgrunnskapittelet vi i samarbeid har kommet frem til er relevant i denne oppgaven.*

2.1.1 Historikk

I begynnelsen av JAFU-prosjektet var det ingen verktøy på plass for å kunne holde fjernundervisningskurs. En sentral del av prosjektet ble – og er fremdeles - dermed å utvikle og tilpasse en samling av relaterte applikasjoner for å støtte gjennomføringen av slike kurs. Disse verktøyene ble gradvis utviklet og videreutviklet gjennom ulike master-, hovedfags- og doktorgradsoppgaver, prosjekter i fagene *INF112 – Systemkonstruksjon* og *INF219 – Praktisk prosjekt i programmering,* og betalt arbeid. Ved oppstarten av denne oppgaven hører følgende av verktøyene i Tabell 2-1 til JAFU-prosjektet.

Navn på system	Beskrivelse	Siste arbeid utført
Dynamic Presentation Generator (DPG)	Innholdshåndteringssystem (CMS)	Elektronisk kompendium-prosjektet (finansiert av SEVU) 2006-2007
Webucator 2.0	System for administrasjon av brukere. Ikke fungerende.	Masteroppgaven ”Webucator 2.0 – A course administration system” av Preben Solheim, fullført 2006
SYSTEKON	System for kontrollspørsmål og interaktive prøver	INF112-prosjekt vår 2007
MVNForum	Forumløsning	(ikke utviklet ved JAFU)

Tabell 2-1. Verktøy som hører til JAFU-prosjektet

DPG 1.0 ble skrevet av Yngve Espelid i 2004 som en del av masteroppgaven hans, og ferdigstilt i 2004 [8]. Espelid baserte noe av arbeidet sitt på resultatene til flere andre hovedfagsoppgaver og doktorgradsavhandlinger skrevet ved instituttet, spesielt [9] og [10], som omhandlet lignende prosjekter kalt henholdsvis *Java Presentation Generator* og *Presentasjons-Mønster-Motor*. Mer informasjon om disse prosjektene finnes i kapittel 2 av [8], som inneholder en sammenligning av de to prosjektene, og i de

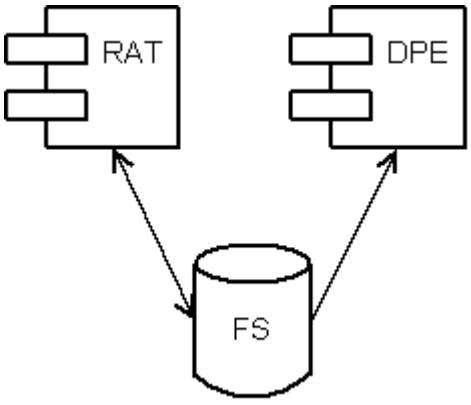
respektive oppgavene til Cruickshanks og Berg. All koden til Espelids *DPG* var imidlertid skrevet fra bunnen av, det var kun ideene fra de foregående systemene som ble brukt og videreutviklet i hans oppgave.

DPG 1.0 har gjennomgått en del vedlikeholdsarbeid etter Espelids oppgave. Sommeren 2005 foregikk det arbeid med utarbeidelse av systemdokumentasjon, retting av noen programfeil, refaktorering av deler av koden, og det ble påbegynt reimplementasjon av presentasjonslaget i MVC-rammeverket *Spring MVC*. Dette arbeidet ble imidlertid ikke videreført, da det senere det året ble bestemt at det skulle foretas en fullstendig reimplementasjon av *DPG*: *DPG 2.0*.

En modifisert versjon av DPG-systemet er også blitt tatt i bruk i andre sammenhenger enn JAFU-prosjektet. Senter for etter- og videreutdanning (SEVU), også ved Universitetet i Bergen, fattet interesse med verktøyet, og ville undersøke om det var mulig å bruke det til fjernundervisningskurs tilknyttet juridisk fakultet. Videreutviklingen av DPG-en startet for fullt i 2007 og ble ferdigstilt til kursstart høsten samme året. Kurset som ble avholdt den høsten var vellykket, og våren 2008 ble det avholdt ytterligere to juridiske nettkurs. Dermed fikk man ytterligere demonstrert nytteverdien med presentasjonsmønstre[8] og generiske systemer, og både sterke og svake sider med den nåværende DPG-implementasjonen ble avdekket.

2.1.2 Overordnet arkitektur

DPG 1.0 er delt inn i to hoveddeler (se Figur 2-1): *Dynamic Presentation Engine (DPE)* og *Repository Administration Tool (RAT)*. Begge disse bruker et sentralt repository som datakilde. Dette repositoriet er en katalog på filsystemet med en forhåndsdefinert katalogstruktur. På grunn av problemer med kompilering av JSP-er, Tomcat og nedlasting av filer for brukere, må denne katalogen ligge inne i applikasjonens katalog i ”webapps”-katalogen til Tomcat.

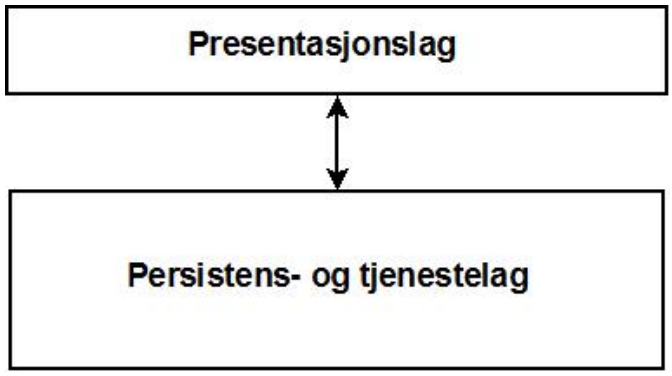


Figur 2-1. Overordnet arkitektur i DPG 1.0

DPE er komponenten som er ansvarlig for å rendere innhold og vise det til vanlige brukere. Det er altså ikke mulig å manipulere innhold ved å gå

gjennom DPE-en i *DPG 1.0*. *RAT* er komponenten som er ansvarlig for å tilby redigering av innhold, presentasjoner og presentasjonsmønstre, samt opprettelse av nye presentasjoner og presentasjonsmønstre. *RAT* kan også håndtere brukere (legge til, fjerne, gi rettigheter til), men denne funksjonaliteten har vært lite brukt.

DPG 1.0 er delt opp i to lag (se Figur 2-2): Et presentasjonslag og et persistens- og tjenestelag. Presentasjonslaget har som ansvarsområde å vise brukeren riktige JSP-sider, å holde orden på hvor i presentasjonen brukeren befinner seg (tilstand), og å delegere hoveddelen av arbeidet til persistens- og tjenestelaget. Dette laget inneholder all forretnings- og persistenslogikken. Lagets ansvarsområder inkluderer blant annet å utføre create-, retrieve-, update-, og delete-operasjoner (CRUD) på presentasjoner, patterns og brukere, å parse XML-data for å bygge objekter og tilstandshåndtering.



Figur 2-2. Den overordnede lagdelingen i DPG 1.0

Presentasjonslaget til DPG-en bruker designmønsteret *Front Controller*[11], både i DPE- og RAT-delen av applikasjonen. Begge disse ble implementert med en servlet som delegerte arbeid videre til resten av applikasjonen. Espelid benyttet ikke noe MVC-rammeverk til implementasjonen av presentasjonslaget. Presentasjonslaget til *DPE* er imidlertid nå implementert i *Spring MVC*[12], som en del av en refaktorering foretatt etter Espelid var ferdig med oppgaven.

I persistens- og tjenestelaget til *DPG 1.0* er designmønsteret *Singleton* [13] mye brukt. Ved oppstart av applikasjonen lastes nemlig alle presentasjonsmønstre og alle presentasjoner inn i minnet, og en slik form for ressursbruk vil man naturligvis at kun skal skje en gang. Dette er i bunn og grunn en primitiv erstatning for caching, og er sannsynligvis gjort for å forbedre ytelsen på applikasjonen. Det er ikke brukt noen form for rammeverk i denne delen heller, verken til tjeneste- eller persistensdelen. Selve persisteringen av dataene skjer i form av flate XML-filer. Ingen form for transaksjonshåndtering er implementert (for eksempel i form av låsing).

2.1.3 Brukere

DPG 1.0 opererer med tre grupper av brukere:

- *Readers*, som har tilgang til å lese informasjon
- *Publishers*, som har tilgang til å endre informasjon
- *Administrators*, som har tilgang til å endre struktur på informasjon (mønster), i tillegg til å administrere de andre brukerne og opprette nye presentasjoner og mønstre.

Tilgang for *Readers* og *Publishers* er på en per-presentasjon-basis. Det vil si at man blir definert som *Reader* eller *Publisher* i en presentasjon, og dermed ikke har tilgang til å vise de andre presentasjonene som finnes i systemet, med mindre man er definert som *Reader* eller *Publisher* i dem. *Administrators* har tilgang til å administrere alle presentasjoner som finnes i systemet. En *Publisher* har alle rettigheter som en *Reader* har i en gitt presentasjon, og en *Administrator* har alle rettigheter som *Publishers* har.

2.2 Webucator 2.0

Selv om det ved denne oppgavens oppstart ikke lenger fantes noen fungerende brukerhåndtering i *JAFU-systemene*, ga Preben Solheims oppgave,[3] som omhandlet *Webucator 2.0*, mange pekepinner på hvordan systemet var ment å fungere, samt hvilke oppgaver det skulle utføre.

Utfordringen ble derfor å se på ideene bak *Webucator 2.0* og legge dem til grunn for implementasjonen av *Webucator 3.0*. Samtidig måtte det gjøres prioriteringer for at *Webucator 3.0* skulle kunne brukes fra denne oppgavens leveringsdato. På den annen side var det også viktig at disse prioriteringene ikke gikk ut over sikkerheten til systemet, eller muligheten til å videreutvikle det. I og med at *Webucator 3.0* har blitt implementert fra bunnen av, var det viktig å legge til rette for at eventuelle fremtidige oppgraderinger kunne gjøres uten å begynne helt på nytt nok en gang.

2.2.1 Hovedmål

Webucator 2.0 hadde i følge oppgaven til Solheim[3] blant annet som hovedmål å:

- Implementere en løsning for administrering av kurs- og brukerhåndtering til bruk i fjernundervisningen.
- Implementere en løsning for å administrere innlevering av obligatoriske oppgaver, samt gradering av oppgaver.

Det andre punktet førte til at *Webucator 2.0* hadde behov for flere typer brukere:

- *User managers* – Hadde ansvaret for å administrere kurs og dets brukere.

- *Reviewers* – Hadde ansvaret for å karaktersette studenters obligatoriske oppgaver.

Administrering av innleveringer av obligatoriske oppgaver er senere skilt ut i et eget system, *Innleveringssystemet*, som ble utviklet i en ny versjon i et prosjekt i faget *INF219 – Praktisk prosjekt i programmering*, høsten 2007. [14]. Denne utskillelsen førte til at behovet for ulike brukere av *Webucator* bortfalt, noe som ble tatt hensyn til i utviklingen av *Webucator 3.0.*

2.2.2 Teknologier

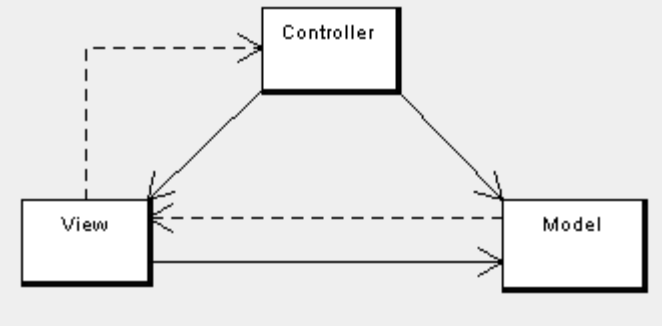
Webucator 2.0 ble implementert for å kjøre mot instituttets sentrale database. Da denne senere ble oppgradert, var ikke lenger *Webucator 2.0* kjørbart. Det fantes svært lite dokumentasjon på hva *Webucator 2.0* krevde av databasekonfigurasjon for å kunne kjøre, og dette var en av hovedgrunnene til at *Webucator 2.0* i dag er et ikke-fungerende system.

Webserveren som ble brukt var *Apache Tomcat*[15], som dekket systemets behov for en JSP og Servlet Container. *Webucator 2.0* var implementert med *Sun Java 5*[16] som programmeringsspråk, *Spring Framework*[12] som applikasjonsrammeverk og *Maven*[17] som applikasjonsbygger.

Oppgaven til Solheim [3] gir ikke informasjon om krav til browserstøtte. Dette kan skyldes at *Internet Explorer* [18] ved oppgavens start hadde ca ¾ markedsandel [19], og at dette ble sett på som tilstrekkelig til at støtte for denne browseren ville dekke de flestes behov.

2.2.3 Arkitektur

Webucator 2.0's arkitektur var basert på design-mønsteret *Model View Controller* [20] (Figur 2-3):



Figur 2-3. Model View Controller**[3]**

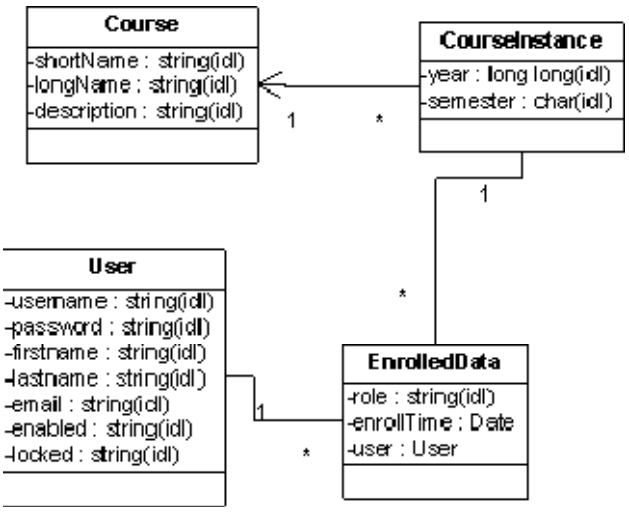
- *Model* – Inneholder forretnings-logikken og datamodellen til applikasjonen.
- *View* – Presenterer applikasjonens tilstand og vil også sende brukerens forespørsler til *Controller*. Vanligvis i web-applikasjoner er *View* HTML-sider.

- *Controller* – Vil svare på forespørsler som kommer fra *View* og oppdatere eller hente data fra *Model*.

Videre gir Solheims [3] oppgave noe mangelfull informasjon om applikasjonens videre oppbygging. Det kan se ut til at det har vært tatt i bruk en form for lagdeling:

- Et presentasjonslag som håndterer systemets kommunikasjon med brukeren.
- Et servicelag som håndterer systemets kommunikasjon med instituttets database.

I tillegg ser det ut til at Solheim har introdusert en domenemodell av Plain Old Java Objects (POJOs)[21], for å innkapsle businesslogikken i systemet. Figur 2-4 sies å være et subsett av denne domenemodellen. Det kan altså se ut til at disse objektene omfatter de grunnleggende begrepene *Webucator 2.0* benytter. I og med at det ikke gis noen komplett beskrivelse verken av objektene i seg selv eller bruken av disse, vil en nærmere analyse av oppbyggingen bli for spekulasjon å regne.



Figur 2-4. Subset av Webucator 2.0's domenemodell **[3]**

2.2.4 Persistering

Persisteringen av domenemodellen som ble omhandlet i forrige avsnitt ser ut til å ha blitt gjort ved hjelp av *Annotations* i Java-koden og rammeverket *Hibernate*[22], som tilbyr persistering i en objekt-orientert kontekst. Eksempel 2-1 viser hvordan *Annotations* er brukt for å beskrive hvordan *Hibernate* skal behandle domeneobjektet med tanke på persistens.

```
@Entity
@Table(name="tbl_user")
public class User implements Serializable , UserDetails {
      private String username;
      private String password;
      private String firstname;
      ...
```

Eksempel 2-1. Persistering av bruker-objekt

Fordelen ved bruk av et rammeverk som *Hibernate* er at man selv slipper å skrive SQL-kode. Det kan også være en fordel å ha brukt et slikt rammeverk ved eventuelt bytte av databaseløsning. Det kan for eksempel være dialektforskjeller mellom SQL-en en *PostgreSQL* database bruker i forhold til en *MySQL* database, som ville måtte endres om man selv skrev SQL. *Hibernate* på sin side støtter de fleste kjente SQL-dialektene. [23]

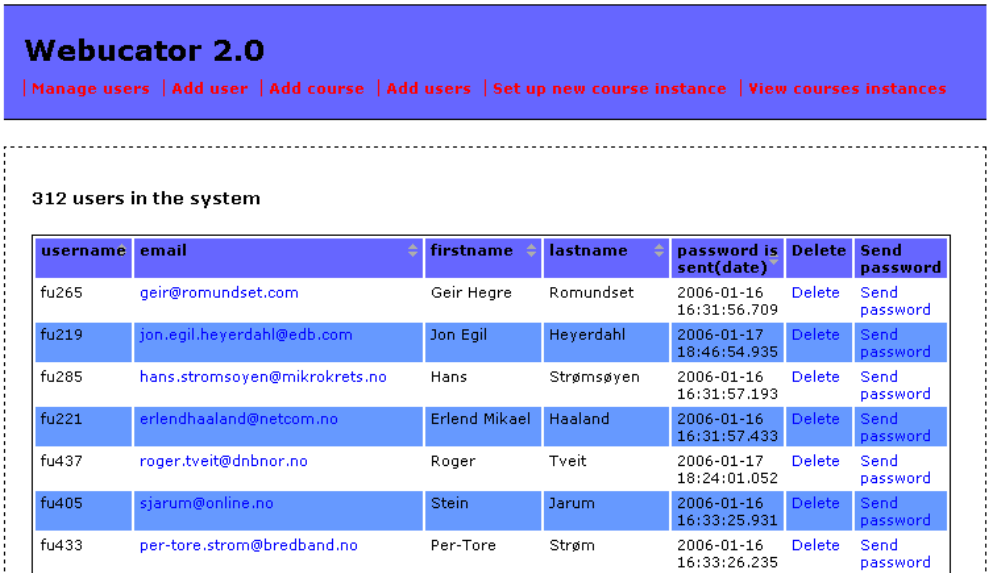
Problemet i tilfellet *Webucator 2.0*, er en svært mangelfull dokumentasjon av hvordan denne løsningen er implementert og hvordan den fungerer i praksis. Dette har vært en av hovedårsakene til at man ikke har fått *Webucator 2.0* opp å kjøre igjen etter at databaseløsningen ved instituttet ble endret.

2.2.5 Brukergrensesnitt

Brukergrensesnittet til *Webucator 2.0* var et at hovedproblemene med systemet da det var i drift. Som vi ser av Figur 2-5 var særlig fargevalget i beste fall dårlig. Forklaringen på dette er at blant annet hovedmenyen med rød skrift på lilla bakgrunn, ble utformet slik for å være sikker på at designet skulle forbedres før systemet ble ferdigstilt og tatt i bruk(!) På grunn av store tidsproblemer med tanke på innleveringsfrist og så videre, ble dette likevel ikke gjort.

Brukerne av systemet ga også uttrykk for mange tungvinte løsninger, særlig med tanke på behandling av mange brukere samtidig. Dette kunne for eksempel være at sletting av brukere eller utsendelse av passord til brukere, begge deler måtte utføres for én og én bruker av gangen.

Grunnet at *Webucator 2.0*, som tidligere nevnt ikke lengre er et kjørbart system, var det ikke mulig å ta en grundig gjennomgang av alle aspektene ved brukergrensesnittet og hva som måtte utbedres.



Figur 2-5. Brukergrensesnitt for Webucator 2.0

3 Dynamic Presentation Generator 2.0

Dette kapittelet gir en kort innføring i *Dynamic Presentation Generator 2.0*, som er en ny implementasjon av et generisk innholdshåndteringssystem basert på ideene bak *Dynamic Presentation Generator 1.0*, som ble utformet som en del av masteroppgaven til Espelid [8], og som er også er beskrevet i delkapittel 2.1.

3.1 Roller

Tilgangen til *DPG 2.0* er rollebasert. Systemet håndterer tre typer brukere (roller):

- *Reader* – Kan lese informasjon i en presentasjon. I fjernundervisningssammenheng er denne brukeren gjerne en student.
- *Publisher* – Kan lese og endre informasjon i en presentasjon. I fjernundervisningssammenheng er denne brukeren gjerne en kursansvarlig eller en studieassistent.
- *Administrator* – Kan lese og endre informasjon i presentasjoner, samt opprette og administrere dem.

En presentasjon kan for eksempel være en nettside for en kursinstans ved fjernundervisningen til UiB, med nyhetsside, fremdriftsplan, ressurser, eksamensplan og annet det er naturlig å finne på et slikt nettsted.

Rollene *reader* og *publisher* er knyttet til en enkelt presentasjon. Det betyr at en og samme bruker for eksempel kan være *publisher* i presentasjonen for *"INF100 – Grunnkurs i programmering våren 2008"* mens vedkommende er *reader* i *"INF101 – Videregående kurs i programmering høsten 2008"*.

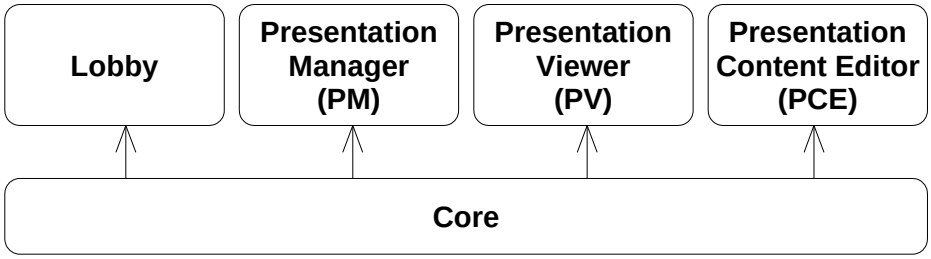
Rollen *administrator* er derimot knyttet opp til hele *DPG2.0*-systemet. Med andre ord har en *administrator* tilgang til alle presentasjoner som finnes i systemet til enhver tid. *Administrator* kan også konfigurere presentasjonene. Med andre ord betyr det at de med denne rollen kan opprette, endre og slette presentasjoner i systemet, samt spesifisere hvilke brukergrupper som skal ha tilgang til systemet.

3.2 Delsystemer

I delkapittel 1.1.3 ble oppbyggingen av den nye *DPG*-en overflatisk beskrevet. I dette delkapittelet vil det gis en mer utfyllende presentasjon av hvilke delsystemer *DPG 2.0* består av.

DPG 1.0 besto som tidligere nevnt av to delsystemer: *DPE* og *RAT*. *DPG 2.0* har fem delsystemer: *Lobby, Presentation Viewer (PV), Presentation Content*

Editor (PCE), Presentation Manager (PM) og *Core*. Oppdelingen kan sees i Figur 3-1. En nærmere beskrivelse av dem følger i de neste underavsnittene.



Figur 3-1. Delsystemene i DPG 2.0

3.2.1 Core

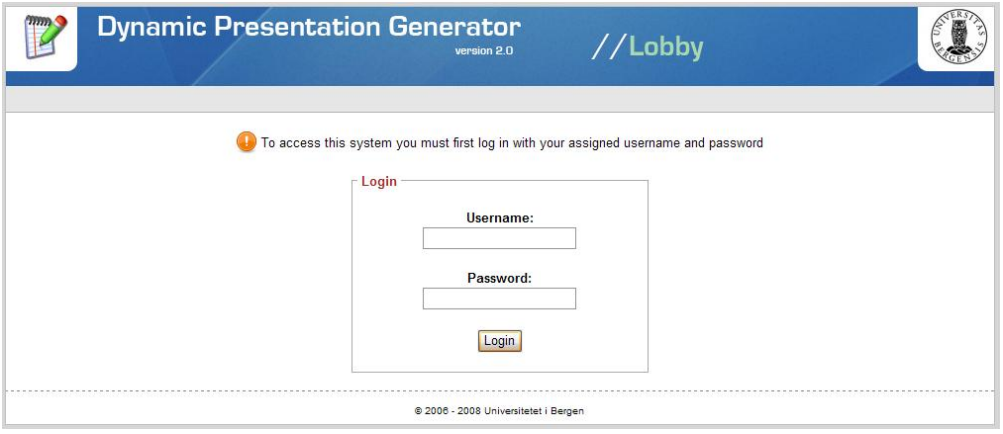
Dette delsystemet inneholder i hovedsak funksjonalitet som er felles for de andre delsystemene. En stor del av *Core* er persistenslaget, som er utformet som en del av masteroppgaven til Karianne Berg, våren 2008 [24].

De resterende delene er i hovedsak utformet av Bjørn Christian Sebak, som en del av hans masteroppgave [5]. Koden i systemet som omhandler ulike former for multimedia, som bilder, videosnutter, lydfiler og så videre er beskrevet i detalj i masteroppgaven til Bjørn Ove Ingvaldsen [6]

Koden som tar hånd om autentisering av brukere og sikkerheten i systemet er beskrevet senere i denne oppgaven (Se kapittel 6).

3.2.2 Lobby

Lobby er det klart minste delsystemet. Dette kan ses på som inngangsportalen til systemet (Se Figur 3-2). Det tar seg blant annet av autentisering av brukere ved hjelp av *remoting* [25] mot *Webucator 3.0* (Se delkapittel 6.4.3).



Figur 3-2. Lobbyen i DPG 2.0 før innlogging.

I tillegg gir *Lobby* etter innlogging en oversikt over hva man kan foreta seg i systemet, for eksempel å vise en presentasjon, eller redigere innholdet av den.

Figur 3-3. Lobbyen i *DPG2.0* etter innlogging.

Informasjonen som vises her vil variere fra hvilken bruker som har logget inn. I listen over presentasjoner (Se Figur 3-3, boks 1), vil brukeren bare finne presentasjoner vedkommende faktisk har tilgang til. For å endre innholdet i en presentasjon (Se Figur 3-3, boks 2), må brukeren ha rolle som *publisher* i presentasjonen, eller eventuelt være *administrator* i systemet. For å konfigurere en presentasjon må brukeren være *administrator* (Se Figur 3-3, boks 3).

I de tilfellene man bruker *Webucator 3.0* som brukerhåndteringssystem for *DPG 2.0*, vil utseendet til listen over presentasjoner også avhenge av statusen til brukergruppen som er tilknyttet presentasjonen. I *Webucator 3.0* samsvarer brukergruppene i *DPG 2.0* som tidligere nevnt med kursinstanser og deres brukere. Disse statusene er aktuelle:

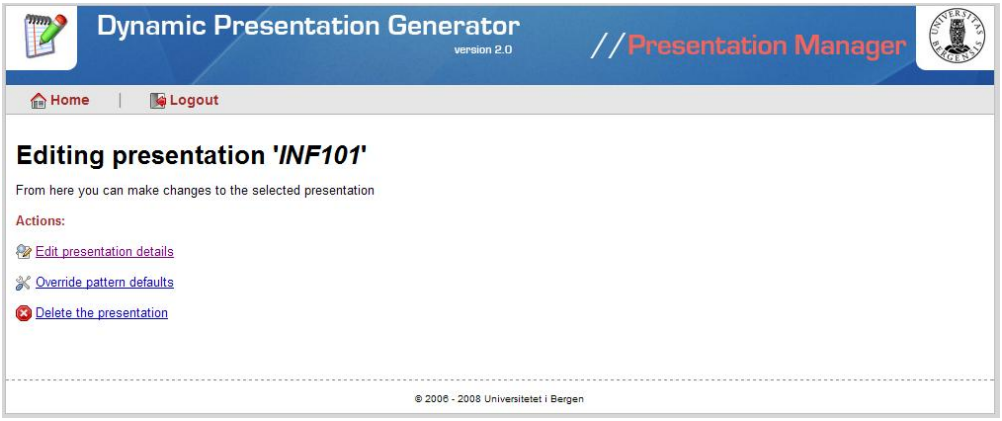
- Om en kursinstans er *enabled* i *Webucator 3.0* vil alle typer brukere kunne se presentasjonene som er knyttet opp til kursinstansen i listen over presentasjoner i *DPG 2.0*.
- Om en kursinstans er *disabled* i *Webucator 3.0* vil bare *publisher*-e i kursinstansen og *administrator*-er i systemet kunne se presentasjonen i listen over presentasjoner.

Bakgrunnen for skillet som er beskrevet over er at man gjerne ønsker å klargjøre en presentasjon (for eksempel for et fjernundervisningskurs), før brukerne skal få tilgang til den. Slik løsningen er implementert, vil man nå kunne opprette et kurs og en kursinstans i *Webucator 3.0*, samt legge til brukere til denne. I *DPG 2.0* kan man så opprette en presentasjon som knyttes til brukergruppen kursinstansen vi akkurat opprettet representerer. Man kan så legge til informasjon og ressurser i presentasjonen. Påmeldte studenter, som har rollen *reader* (Se 3.1), vil nå ikke ha tilgang til denne presentasjonen, og man trenger ikke å bekymre seg for at man utilsiktet

avslører informasjon til dem under dette arbeidet. Når presentasjonen etter hvert er klar til å *"ta i mot"* brukerne, trenger man bare å *enable* kursinstansen med brukergruppen til presentasjonen i *Webucator*.

3.2.3 Presentation Manager (PM)

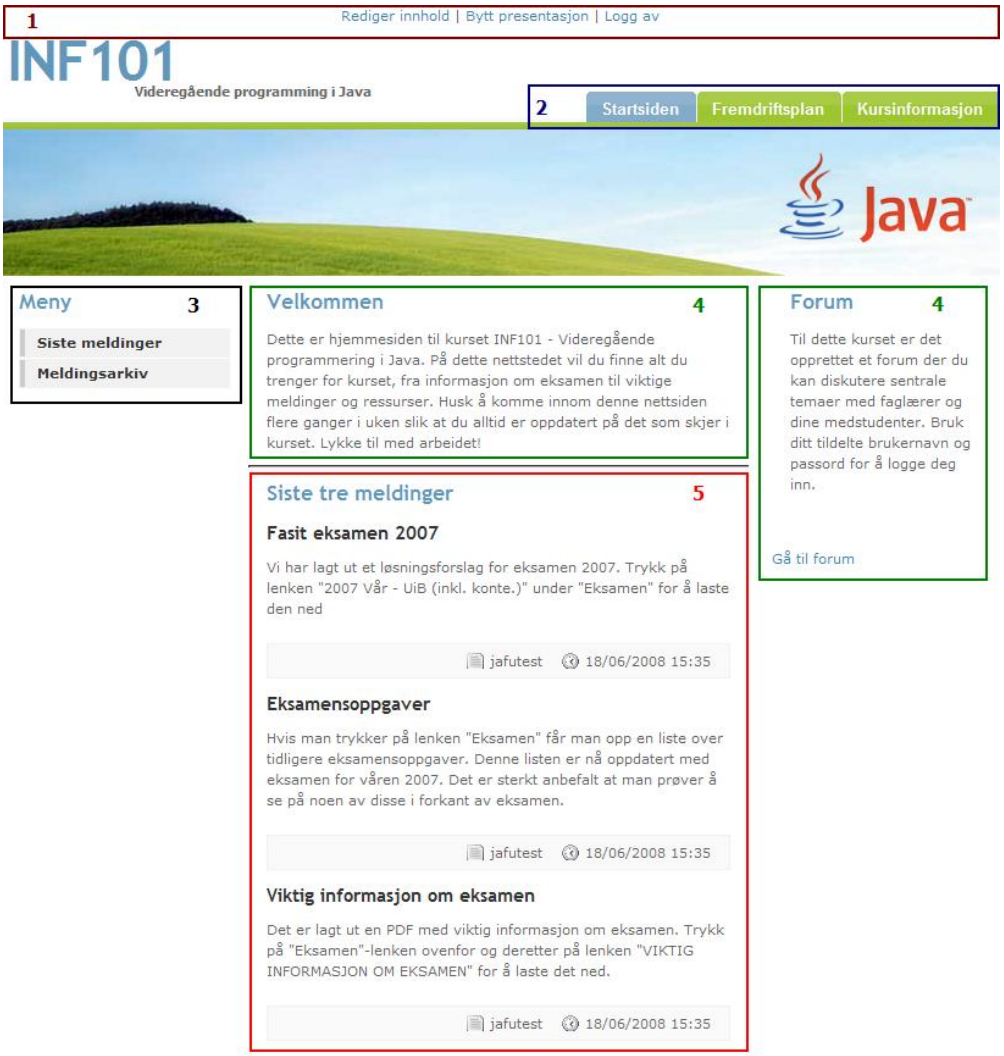
Dette er administratordelen av *DPG 2.0*, og kun brukere med rollen *administrator* har tilgang her. Her har man mulighet til å opprette, kopiere, endre og fjerne presentasjoner og presentasjonsmønstre. Man kan også knytte en presentasjon til en gitt brukergruppe som skal ha tilgang til den. Dette skjer som tidligere nevnt ved hjelp av *remoting* mot *Webucator 3.0*, og er beskrevet i delkapittel 6.4.3



Figur 3-4. *PM* for presentasjonen *"INF100"*.

PM (Figur 3-4) gir også brukeren mulighet til å overkjøre de transformasjoner, sidemaler og stilark som er satt som standard for presentasjonen. De av dem som brukeren eventuelt laster opp har høyere presedens enn standardversjonene. *PV* (Figur 3-5) vil da for eksempel rendere innholdet i en presentasjon med brukerens stilark om vedkommende har lastet opp en egen versjon av dette. Dette gjør at man kan gi presentasjoner forskjellig utseende fra det som er gitt som standard.

3.2.4 Presentation Viewer (PV)



Figur 3-5. Oversikt over elementer i en presentasjon i DPG2.0

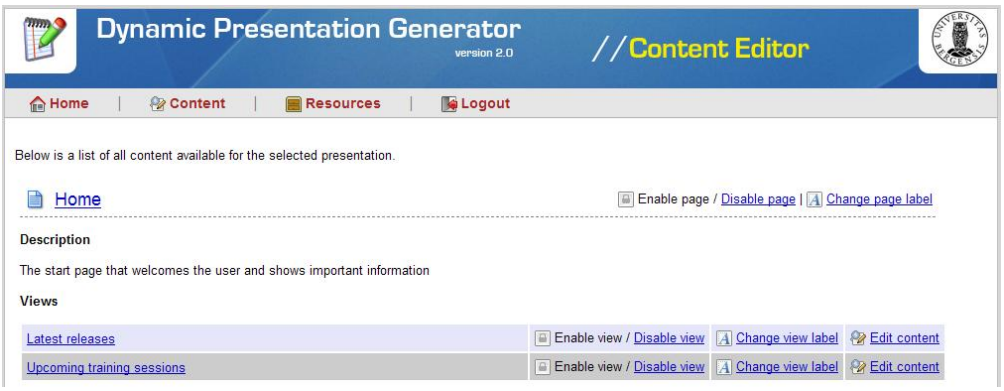
Denne komponenten renderer presentasjonene og viser dem til brukerne som en nettside i en nettleser. Alle autentiserte brukere har tilgang til *PV*, men informasjonen som vises kan i visse tilfeller være forskjellig avhenging av hvilken rolle man har. En *publisher* vil for eksempel kunne få opp en *Edit*-knapp i tekstfeltene på siden, for å ha mulighet til å endre denne. Sidene (*page*) er bygget opp av fire hovedelementer:

- *Navigation bar (*Se Figur 3-5, boks 1) – Denne inneholder gjerne en lenke til lobbyen, og en lenke for å logge ut av systemet. Den kan også inneholde andre lenker basert på hvilken rolle den inneværende brukeren har. En *publisher* vil for eksempel typisk ha en lenke her for å kunne endre presentasjonen som vises.
- *Main menu (*Se Figur 3-5, boks 2) – Denne inneholder lenker til alle *page*-ene i systemet (for den gitte presentasjonen), og muliggjør navigering mellom disse. Tab-en for *page*-en *Startsiden* er nå valgt, og det er denne som vises i presentasjonen.

- *View menu (*Se Figur 3-5, boks 3) – Er en undermeny som viser oversikten over de *views* en *page* består av.
- *Views* er elementer i en *page* (side). De kan merkes `alwaysVisible` og vil da ikke vises som et valg i *view menu*, da de alltid vil være synlig på *page*-en uavhengig av hvilke valg som blir gjort i *view menu*. I Figur 3-5 er boksene merket 4 eksempler på *views* som er `alwaysVisible`, og vi ser at navnet deres ikke finnes i *view menu*. Figur 3-5, boks 5, viser et *view* som ikke er merket `alwaysVisible` og vi ser at navnet til dette *view*-et er et valg i *view menu*. Om det ikke finnes *view* i *page*-en som ikke er `alwaysVisible`, vil *view menu* ikke være synlig.

3.2.5 Presentation Content Editor (PCE)

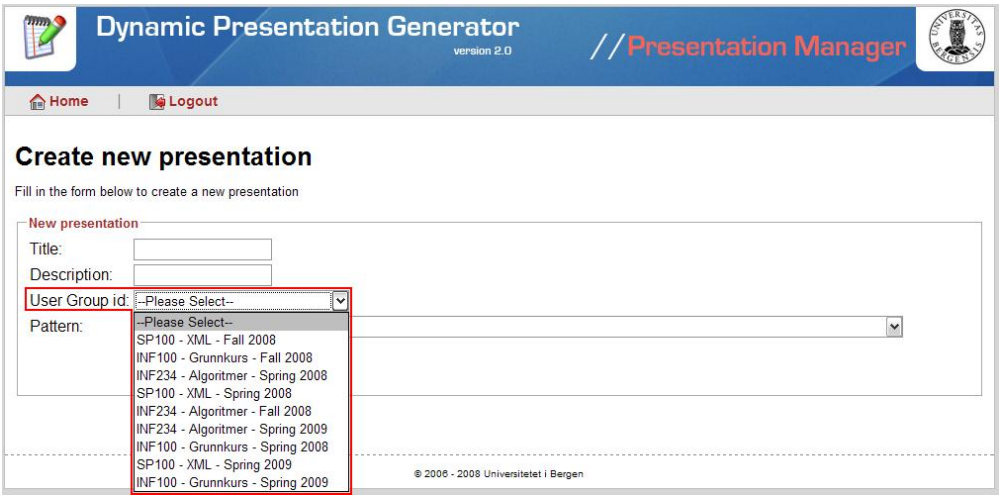
Figur 3-6 viser den siste komponenten som har ansvaret for å tilby redigering av innholdet i presentasjonene. Brukere med rollen *publisher* i presentasjonen som de ønsker å redigere, og *administrator*-er i systemet, har tilgang til denne funksjonaliteten.



Figur 3-6. PCE for en gitt presentasjon.

3.3 Kobling mot Webucator 3.0

Koblingen mellom *DPG 2.0* og *Webucator 3.0*, altså koblingen mellom applikasjonen som håndterer og viser presentasjoner og brukerne som skal ha tilgang til dem, skjer i *Presentation Manager* (Se Figur 3-7).



Figur 3-7. Skjerma for opprettelse av ny presentasjon i DPG 2.0

Når man oppretter en ny presentasjon, er et av valgene å knytte presentasjonen til en brukergruppe, *User group.* (Se Figur 3-7, markert område). I tilfeller der man bruker *Webucator 3.0* som brukerhåndtering for *DPG 2.0,* for eksempel i fjernundervisningssammenheng, vil listen over brukergrupper samsvare med de kursinstansene som til enhver tid finnes i *Webucator 3.0* og brukerne som er knyttet til dem.

Delkapittel 6.4.3 vil beskrive nærmere hvordan kommunikasjonen mellom *DPG 2.0* og *Webucator 3.0* foregår.

4 Webucator 3.0

Dette kapittelet vil gi en gjennomgang av *Webucator 3.0*'s arkitektur og oppbygging. Det vil også bli redegjort for hvordan brukergrensesnittet er bygget opp og hvilke funksjoner det tilbyr.

4.1 Krav til teknologier

Da utviklingen av *Webucator 3.0* startet ble det kartlagt en del krav til hvilke teknologier som skulle brukes. I hovedsak ble dette gjort for å sikre at systemet skulle kunne kjøre på Institutt for Informatikks webservere, samt at applikasjonens oppbygging ikke skulle være veldig ulik den som ble valgt i de andre JAFU-systemene. Det ble også lagt vekt på at systemet skulle bruke teknologi basert på åpen kildekode.

I bunn kjører instituttet en *PostgreSQL8* [26] databaseserver der Webucator 3.0 har fått tildelt en konto. Webserveren kjører *Apache Tomcat 6.0.14*[27] som JSP- og Servlet Container.

Videre bruker *Webucator 3.0* fellesteknologier som ble valgt for JAFU-systemene. Dette innbefatter *Sun Java 6*[16] som programmeringsspråk, *Spring Framework 2.0.7*[12] som applikasjonsrammeverk og *Ant 1.6.5+*[28] som applikasjonsbygger.

For å sikre at de fleste brukere skal kunne bruke applikasjonen var det også påkrevd at Webucator 3.0 støttet de to største nettleserne på markedet (Se Tabell 4-1)Tabell 4-1. **Markedsandel for nettlesere på verdensbasis** , *Microsoft Internet Explorer*[18] og *Mozilla Firefox*[29].

2008	Internet Explorer	Firefox	Opera
Mai	53,8%	39,8%	1,5%
Mars	52,8%	37,0%	1,4%
Januar	53,2%	36,4%	1,4%

Tabell 4-1. Markedsandel for nettlesere på verdensbasis **[19]**

Det er også kjent at den norske nettleseren *Opera* [30]har større markedsandeler her hjemme enn de internasjonale undersøkelsene viser. Blant annet viser undersøkelser for det norske teknologinettstedet *Hardware.no* at én av fem lesere bruker *Opera*[31]. Ved ferdigstillelsen av denne oppgaven støtter Webucator 3.0 alle de tre nevnte nettleserne.

2008	Internet Explorer	Firefox	Opera
Mai	35%	40%	20%

Tabell 4-2. Markedsandel for nettlesere blant leserne til *Hardware.no* **[31]**

4.2 Overordnet struktur

Dette avsnittet gir en gjennomgang av den overordnede strukturen til Webucator 3.0, hvordan applikasjonen er bygget opp og koden er organisert.

4.2.1 Lagdeling

Å dele applikasjoner i lag har mange viktige fordeler, blant annet[32]:

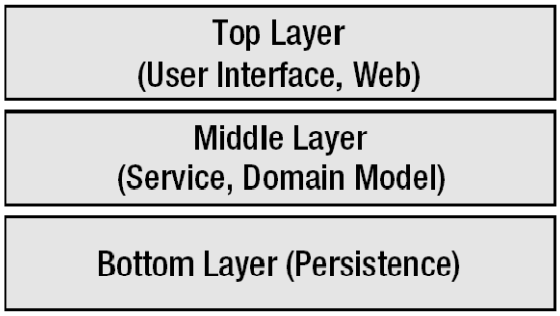
- Man bør kunne fokusere på et enkelt lag som en helhet uten å vite mye om de andre lagene i systemet.
- Man kan bytte ut lag for å endre til alternative implementasjoner. Skulle man for eksempel endre persistenslaget til å bruke en annen datalagringsløsning, vil man kunne gjøre dette og allikevel vite at de andre lagene vil fungere som før.
- Når man har et lag på plass i en applikasjon, kan flere ulike delsystemer i lag lengre oppe i systemet bruke laget. Dette hindrer for eksempel at hvert av delsystemene i en applikasjon må implementere sin egen databaseløsning osv.

På den annen side finnes det også ulemper med bruk av lag, som det er viktig å være oppmerksom på, for eksempel[32]:

- Lag kan ikke innkapsle alle ulike aspekter ved et system. Dette fører noen ganger til vannfallsendringer. En kan for eksempel tenke seg at man trenger å legge til et felt som skal vises i UI, og at dette må ligge i databasen. Man må da legge til feltet i alle lagene i mellom også.
- Lagdeling kan gå ut over ytelsen. Man trenger typisk å transformere data fra en representasjon til en annen når man går fra et lag til et annet. I de fleste tilfeller veier imidlertid ytelsesfordelene ved lagdeling opp for ulempene. For eksempel kan et lag som håndterer transaksjoner optimeres, og på den måten gi en stor ytelsesforbedring.

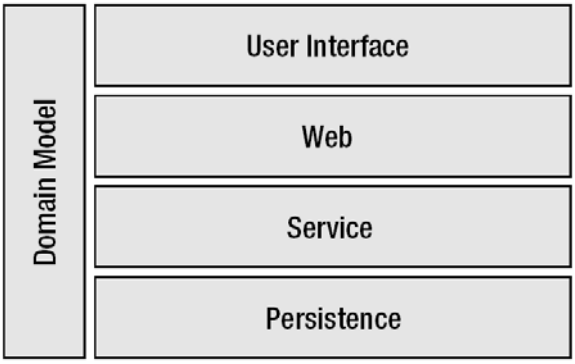
Det er vanlig å dele opp applikasjoner i 3 hovedlag[33] (Se Figur 4-1):

- Et *Top Layer* som håndterer interaksjonen mellom bruker og systemet. Dette laget inneholder typisk brukergrensesnitt og kontrollere.
- Et *Middle Layer* som håndterer logikken i systemet. Dette laget inneholder typisk domeneobjekter og serviceklasser.
- Et *Bottom Layer* som kommuniserer med databaser, transaksjonsmanagere osv.



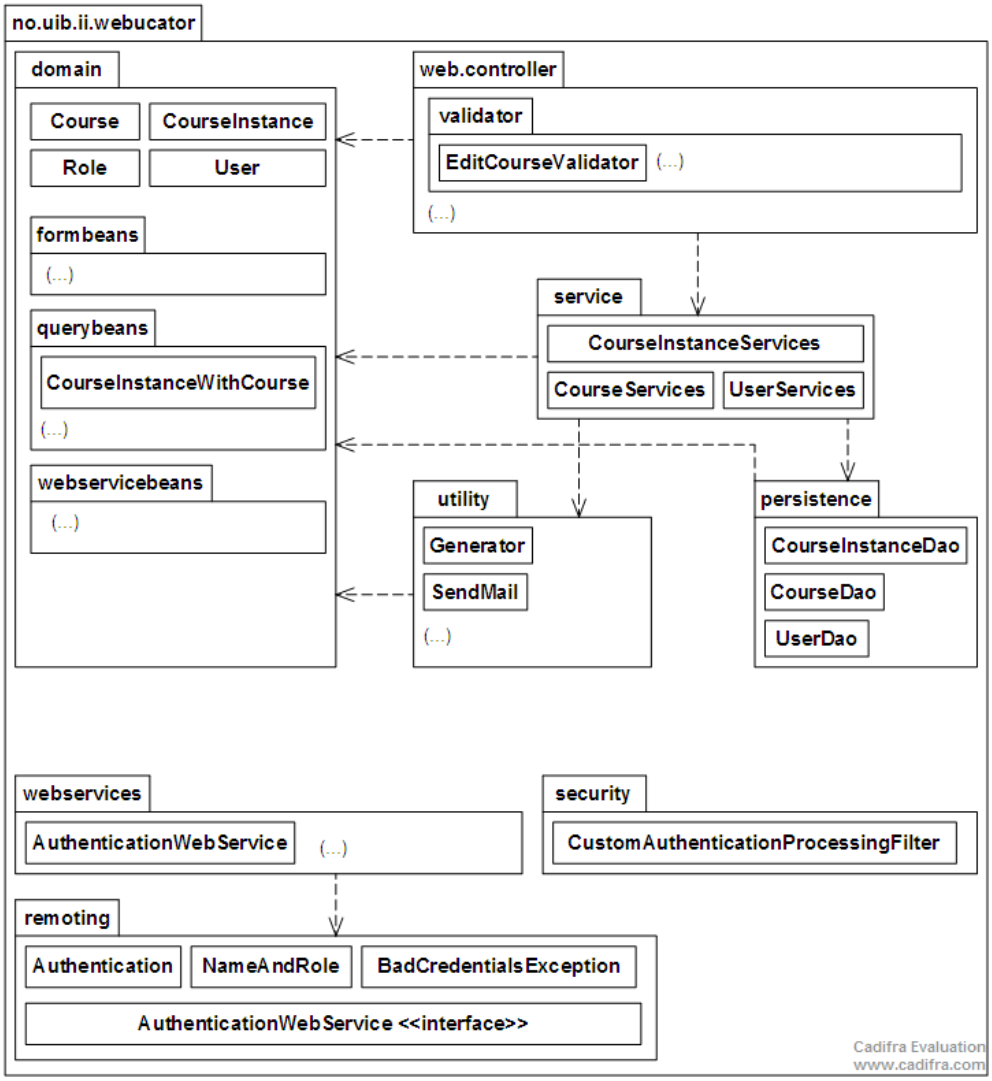
Figur 4-1. Generelle, høynivå lag i en webapplikasjon **[33]**

Som tidligere nevnt er Webucator 3.0 implementert ved hjelp av rammeverket *Spring Framework* [12]. Programmering i dette rammeverket gjør bruk av minst fem lag for abstraksjon[33]. Disse er skissert i Figur 4-2:



Figur 4-2. Spring MVC applikasjonslag **[33]**

4.2.2 Pakkestruktur



Figur 4-3. Oversikt over pakker i Webucator 3.0

Figur 4-3 er en grafisk fremstilling av pakkestrukturen til *Webucator 3.0*. Som vi ser samsvarer den i stor grad med femlagsmodellen som ble beskrevet i avsnitt 4.2.1 og Figur 4-2. En nærmere beskrivelse av strukturen finnes i de følgende underavsnittene.

Brukergrensesnitt

Selve brukergrensesnittet i *Webucator 3.0* blir generert av JSP-sider som vises i en nettleser. Som vi ser er dette ikke en del av pakkediagrammet.

Domain Model

I *domain*-pakken finner vi applikasjonens domeneobjekter. *Webucator 3.0* har fire slike, som representerer de konseptene programmet omhandler; Kurs, kursinstanser, roller og brukere.

I tillegg inneholder *domain*-pakken en del andre pakker som inneholder forskjellige beans. Disse brukes i ulike forms, spørringer osv. der de fire

domeneobjektene ikke er tilstrekkelige. Et eksempel kan være beanen *CourseInstanceWithCourse* som i tillegg til de vanlige feltene domeneobjektet *CourseInstance* (Kursinstans) har, også inneholder en referanse til kurset det instansierer.

Web

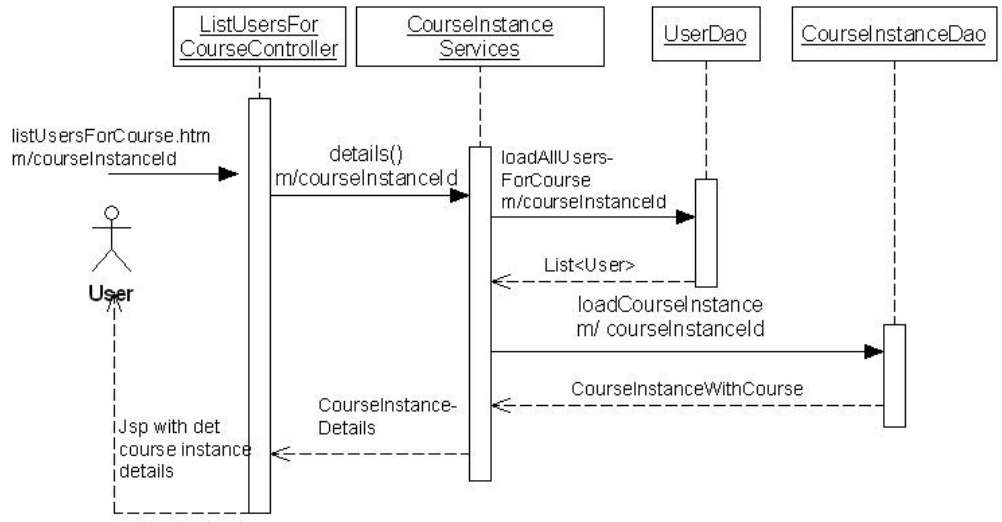
Web-pakken inneholder kontrollerne til systemet. I hovedsak er det kontrollere som utfører ulike brukstilfeller, samt en del for prosessering av data.

I webpakken er det også en validator-pakke som inneholder validatorer til ulike skjema som presenteres for brukeren i brukergrensesnittet. Dette kan for eksempel være en validator som sjekker at man har fylt ut alle postene i et skjema.

Service

Service-pakken har som oppgave å tilby et rent grensesnitt til kontrollerne i web-laget for å utføre ulike oppgaver i systemet. Som vi ser av Figur 4-4, kan service-laget for eksempel skjule underliggende databasekall og prosesseringer, slik at kontrollerne i web-laget bare trenger å gjøre et metodekall for å få utført en spesiell oppgave.

Eksempelet viser en bruker som vil se detaljer om en kursinstans, inkludert en liste over brukere i kursinstansen. Vi ser at kontrolleren bare gjør ett kall ved hjelp av metoden details() til service-laget. Service-laget skjuler her at det faktisk gjøres flere kall mot to ulike DAO-er (og databasetabeller) for å få tak i alle nødvendige data for å vise informasjonen brukeren ønsker.



Figur 4-4. Fremstilling av interaksjon mellom bruker og system.

Persistence

Persistence-pakken inneholder *DAO*-er som kommuniserer med databasen ved hjelp av *JDBC*[34]. Det er både fordeler og ulemper med en slik løsning, noe som er nærmere beskrevet i delkapittel 7.4.3.

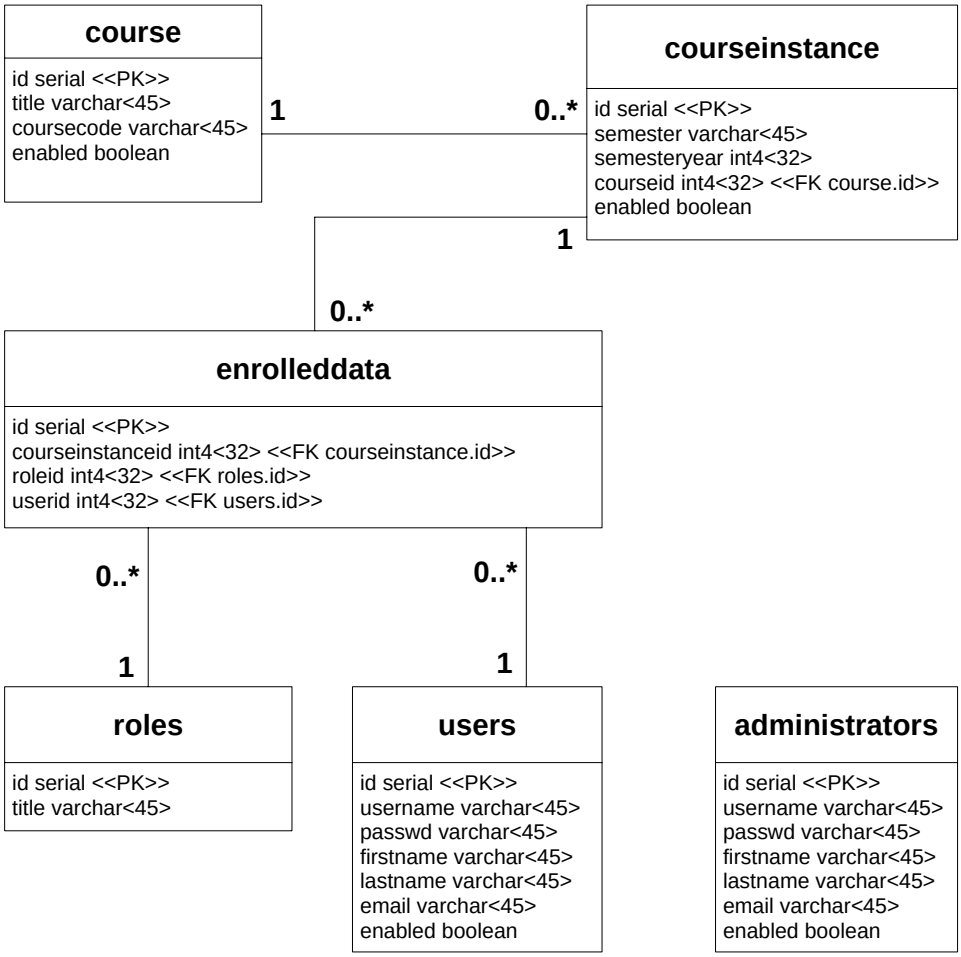
En modell av tabellene i databasen finnes i Figur 4-5. Modellen er basert på den som ble gitt for *Innleveringssystemet* i [14] (Se 1.1.2), som igjen var basert på modellen for *Webucator 2.0* (Se Figur 2-4). Med bakgrunn i en eventuell fremtidig tettere kobling mellom disse systemene ble det sett på som fordelaktig at datamodellen for de to systemene var noenlunde samsvarende.

Den største endringen fra *Webucator 2.0*'s modell er tabellen *administrators.* Det naturlige ville kanskje være en løsning der *administrator* var en rolle i tabellen *roles,* på samme måte som *publisher* og *reader* er det. Dette ble imidlertid sett på som en dårlig løsning implementasjonsmessig. Rollen *administrator* skiller seg fra de andre rollene i systemet ved at den ikke er knyttet opp mot kurs og kursinstanser, men derimot gir tilgang til systemet i sin helhet.

Hadde rollen *administrator* blitt plassert i *roles*, og administratorene i den vanlige *users*-tabellen, ville flere oppgaver i systemet blitt tungvint å implementere. Hvis man for eksempel ville legge til en ny administrator, måtte man ha lagt vedkommende til som bruker med rollen *administrator* i alle kursinstansene i systemet gjennom oppføringer i *enrolleddata*. I tillegg ville man, om man la til en kursinstans i systemet, måtte søke gjennom databasen etter administratorer og legge dem til som brukere av kursinstansen med rollen *administrator*. Dette ble sett på som en dårlig løsning fordi det hadde komplisert implementasjonen, men også fordi det kunne bli en mulig fremtidig ytelsesflaskehals, da tabellen *enrolleddata* ville få mange oppføringer og antall kall mot databasen ville bli betydelig flere enn med løsningen som ble valgt.

Grunnen til den noe ukonsistente navngivingen av tabellene, som *course* i entall og *users* i flertall har den enkle forklaringen at *PostgreSQL*[26] har en del reserverte ord som ikke kan brukes som navn på tabeller, deriblant *user* og *role.* Dette gjenspeiles også i en del feltnavn, som *courseinstance.semesteryear*, som er valgt i og med at *year* også er et reservert ord i *PostgreSQL*.

Feltene som er *primary key* i en tabell er merket <<*PK*>>. På samme måte er felter som er *foreign key* i en tabell merket <<FK tab.felt>>, der *tab* er tabellen det refereres til, og *felt* er feltet det refereres til.



Figur 4-5. Oversikt over databasetabeller som brukes i Webucator 3.0

Andre pakker

I tillegg inneholder Webucator 3.0 en del andre pakker som ikke faller inn under laginndelingen:

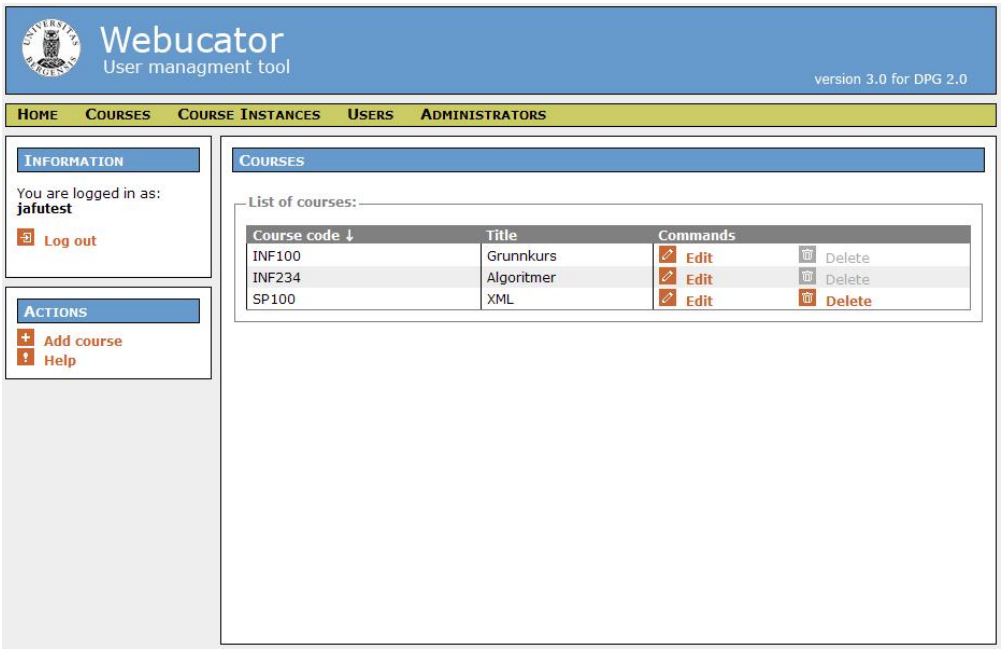
- *Utility* – Inneholder ulike redskaper som generatorer for brukernavn og passord, grensesnitt for å sende e-post fra systemet og så videre.
- *Security* – Tar hånd om sikkerhetsmekanismer som må programmeres.
- *Webservices/Remoting* – Disse pakkene håndterer kommunikasjon med andre webapplikasjoner.

4.3 Funksjonalitet

Webucator 3.0's hovedoppgave er i første rekke å administrere kurs, kursinstanser og brukere, og tilby denne informasjonen til *DPG 2.0* slik at den kan brukes i fjernundervisningssammenheng. I de neste underavsnittene gis en kort gjennomgang av hovedfunksjonaliteten i *Webucator 3.0.*

4.3.1 Kurs

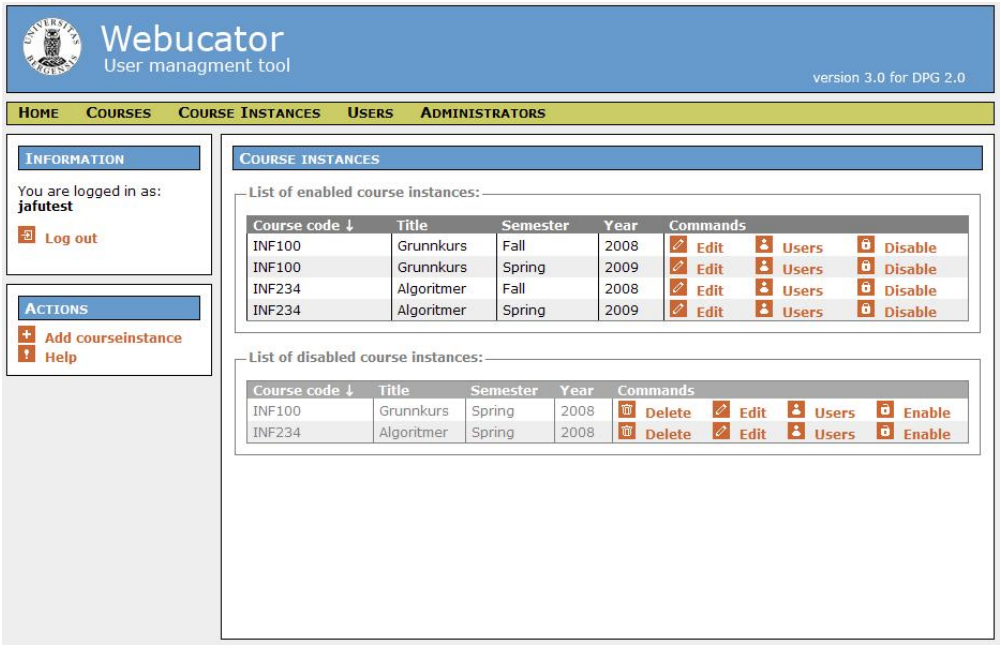
Kurssidene tilbyr en oversikt over de kursene som er lagt inn i systemet, samt mulighet for å legge til, endre og slette dem. Et kurs er representert med en kurskode og et navn. (Se Figur 4-6)



Figur 4-6. Kurssidene i Webucator 3.0

4.3.2 Kursinstanser

Kursinstanser er som navnet tilsier en instans av et kurs. Disse instansene inneholder i tillegg til kursinsformasjonen også informasjon om hvilket semester (vår/høst) og år (for eksempel 2008) det undervises i denne kursinstansen. Kursinstanser har også en liste over brukere som skal ha tilgang til den.



Figur 4-7. Kursinstanssidene i Webucator 3.0

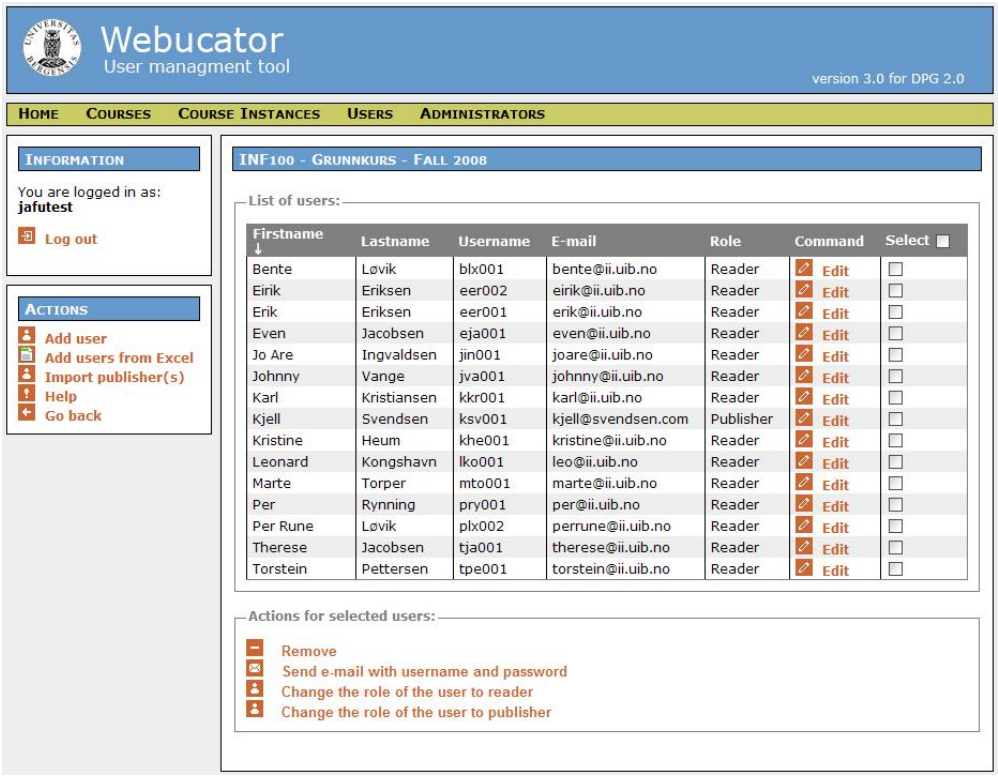
Figur 4-7 viser kursinstanssidene i *Webucator 3.0*. De tilbyr en liste over kursinstansene i systemet. Det er gjort et skille mellom aktiverte og deaktiverte kursinstanser i denne listen. (Forskjellen mellom disse er beskrevet i delkapittel 3.2.2). I tillegg tilbys det funksjonalitet for å legge til, endre, slette, aktivere og deaktivere kursinstanser på disse sidene. I tillegg fins det for hver kursinstans en lenke, *"Users"* som leder til siden for å håndtere brukere for den enkelte kursinstansen. (Se delkapittel 4.3.3).

4.3.3 Brukere

Webucator 3.0 håndterer tre typer brukere (Roller), basert på hvilke tilganger de skal ha ved bruk i *DPG 2.0*. De ulike rollene ble nærmere beskrevet i kapittel 3.1.

Brukere i en kursinstans

Som beskrevet i delkapittel 4.3.2 har hver kursinstans en liste over brukere. Bruker-sidene i en kursinstans håndterer disse. (Se Figur 4-8). Her har man mulighet til å legge til, endre og fjerne brukere. Man kan sende dem e-post med brukernavn og passord som de skal bruke for innlogging til *DPG 2.0*, eller endre rollen de skal ha i kursinstansen mellom *reader* og *publisher*.



Figur 4-8. Brukersidene i en kursinstans i Webucator 3.0

I tillegg til funksjonene som er beskrevet i avsnittet over, finnes også to funksjoner som er skreddersydd for å forenkle en del arbeidsoppgaver ved bruk av *Webucator 3.0* i fjernundervisningssammenheng:

Add users from Excel

Ved oppstart av et fjernundervisningskurs, får man typisk oversendt *Excel*-fil fra *Senter for Etter- og VidereUtdanning (SEVU)* med de brukerne som er påmeldt og har betalt kontingent for kurset. Det var derfor ønskelig med en funksjon som tillot direkte import av brukerne fra denne filen uten at man manuelt måtte legge brukere til i systemet én og én.

Denne funksjonen fantes også i *Webucator 2.0*, og koden som ble skrevet av Preben Solheim [3] var tilgjengelig. Denne ble tilpasset til *Webucator 3.0*, blant annet ved å legge til automatisk generering av brukernavn og passord til brukerne i *Excel*-filen når de ble importert til systemet.

Import publisher(s)

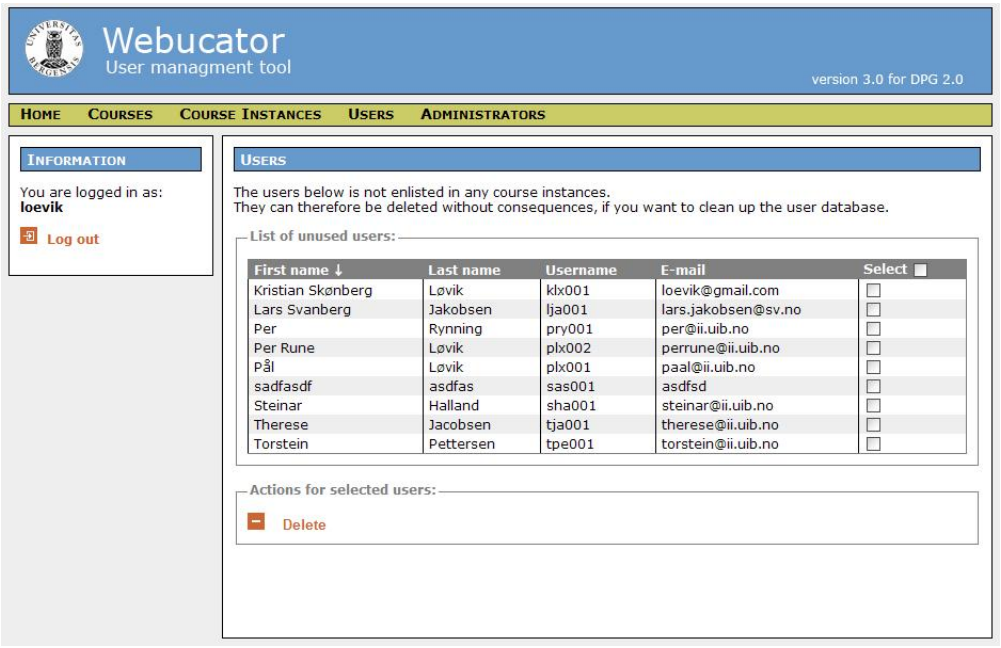
Kursansvarlige og studieassistenter er typiske *"gjenbruksartikler"* i fjernundervisningssammenheng, på den måten at de gjerne dukker opp i flere kursinstanser. Det ville derfor være tungvindt å måtte opprette dem på nytt for hver nye kursinstans de skulle delta i.

Denne funksjonen søker gjennom systemet etter *publisher*-e i de andre kursinstansene i systemet, og tilbyr brukeren en liste over dem, med mulighet til å importere dem til gjeldende kursinstans.

Brukere utenfor kursinstanser

En erfaring fra *Webucator 2.0* var at etter hvert som kursinstanser var gjennomført og ble slettet, ble det liggende mange ubrukte brukere i systemet. Dette gikk etter hvert utover ytelsen, og det var derfor ønskelig å få på plass en løsning for å unngå dette i *Webucator 3.0.* Den finner vi under ”users” på hovedmenyen. (Se Figur 4-9).

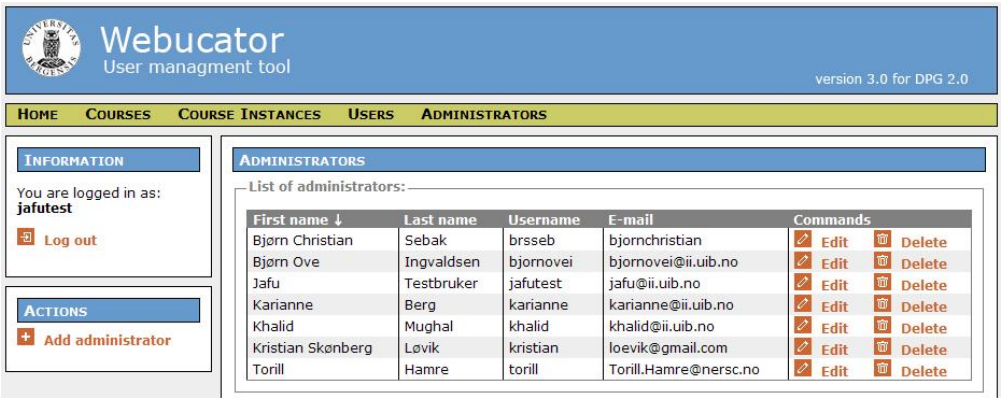
Disse brukersidene søker frem de brukerne som ikke er assosiert med noen kursinstanser og tilbyr mulighet for å vise og eventuelt slette dem.



Figur 4-9. Brukere utenfor en kursinstans i Webucator 3.0

Administratorer

Under *”Administrators”* i hovedmenyen har man mulighet til å legge til, endre og slette administratorer i *DPG 2.0*. (Se Figur 4-10)



Figur 4-10. Oversikt over administratorer for DPG 2.0 i Webucator 3.0

4.4 Design og brukergrensesnitt

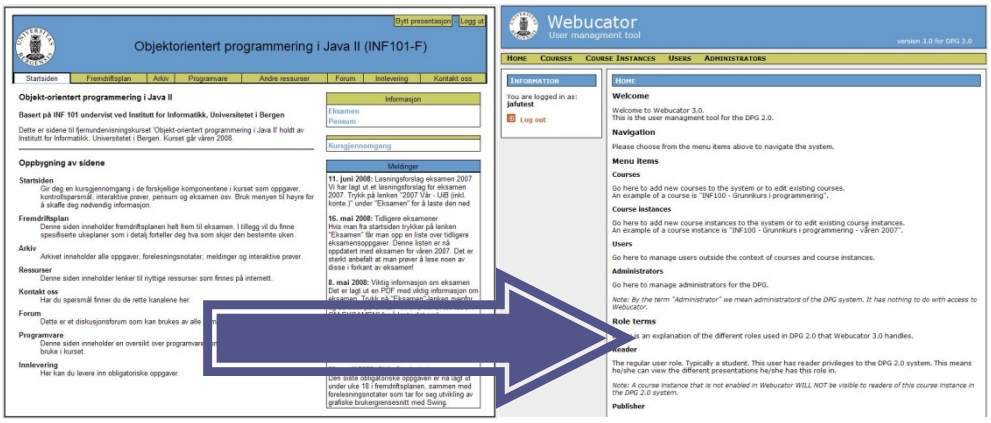
Da *Webucator 2.0* var i drift var et av hovedproblemene at det under utviklingen av systemet ikke hadde vært tilstrekkelig tid til å få på plass et funksjonelt og oversiktlig brukergrensesnitt. Som vi har sett i et tidligere delkapittel (2.2.5) var blant annet enkelte av fargevalgene knapt lesbare. Andre feil kunne være at i lange lister over bruker ble font-størrelsen stadig større jo lenger ned i listen man kom.

Det har derfor under hele utviklingen av *Webucator 3.0* vært lagt mye arbeid ned i selve brukergrensesnittet, og at det skal følge viktige retningslinjer for hva som gjør et brukergrensesnitt godt, som blant annet er beskrevet i [35]. I de neste avsnittene følger en gjennomgang av disse retningslinjene, som gjennomgående er brukt ved designvalg i systemet. Det er også gitt en del konkrete eksempler.

4.4.1 Estetisk

Alle elementene i et brukergrensesnitt konkurrerer i utgangspunktet om brukerens oppmerksomhet. Det er derfor viktig at det er behagelig å arbeide i og bidrar til brukerens forståelse av informasjonen som blir presentert. Hva som ser pent ut, det være seg farger eller ikonvalg, er ofte veldig individuelt. De fremtidige brukerne av *Webucator 3.0* har derfor under hele utviklingsperioden kommet med innspill til hvordan systemet ser ut og hva som eventuelt burde endres.

Som basis for selve utseendet til systemet ble stilarket til fjernundervisningssidene brukt. Dette har fått gode tilbakemeldinger, blant annet senest under evalueringen av fjernundervisningen, våren 2008. Stilarket ble videreutviklet og utvidet til det vi finner i *Webucator 3.0* i dag, gjennom flere tilbakemeldingsrunder med brukerne. Fargevalgene ble i all hovedsak beholdt i tillegg til navigeringsløsningen med hovedmeny øverst. Det ble laget et noe klarere skille mellom de ulike boksene på sidene, og hovedruten ble gjort noe større, da flere av sidene i *Webucator 3.0* inneholder tabeller med mye informasjon, som krever sin plass. En skisse av forsiden til de to systemene kan sees i Figur 4-11.



Figur 4-11. Videreutvikling av stilark fra fjernundervisningen

Det er også gjort grep for at det skal være relativt enkelt å endre utseendet til systemet, om dette er ønskelig. Det er derfor gjennomgående for systemet at det er gjort et skille mellom innhold og presentasjon, ved at presentasjonen er definert i et eget stilark i CSS-format [36].

Et problem man gjerne støter på når man gjennom *CSS* endrer for eksempel fargen på alle linkene i systemet, kan være at denne fargen står dårlig til ikonene som er valgt for de samme linkene. Dette er prøvd unngått ved å ta i bruk såkalte *Sanscons* [37]. Dette er ikoner som kan stilsettes ved hjelp av *CSS*. Vi ser av Figur 4-12 hvordan ikonene kan nedre farge og utforming, uten at det de beskriver endres.

Figur 4-12. Ikoner stilsatt ved hjelp av CSS.

```
<img
src="<c:url value='/graphics/icons/trackback.gif' />"
class="iconlink"
/>
```

```
img.iconlink {
      margin-right: 5px;
      background-color: red;
      background-image: url("/icons/bg_rounded.gif");
}
```

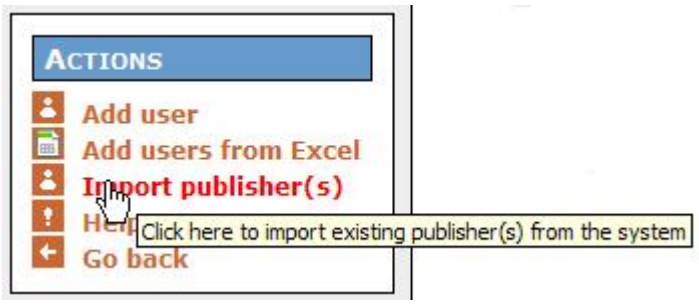
Eksempel 4-1. Stilsetting av ikoner ved hjelp av CSS.

Eksempel 4-1 viser hvordan ikonet blir definert i JSP og ved hjelp av klassen `iconlink` blir knyttet opp til CSS-stilarket som definerer utseendet til denne klassen. I figuren er det kun vist et lite utvalg av hvordan man kan forme ikonet. Deklarasjonen `margin-right` er satt for å lage et mellomrom fra ikonet til eventuell etterfølgende tekst på fem piksler. `background-color` angir ikonets farge, mens `background-image` angir ikonets form.

4.4.2 Tydelig

For å gjøre viktig informasjon åpenbar er det viktig at brukergrensesnittet er tydelig utformet og at det prøver å hindre brukerfeil og bidrar til å forenkle læring og bruk av systemet.

Allerede på velkomstsiden blir brukeren presentert for en kort innføring i hva de ulike delene av systemet inneholder, og ulike termer som brukes. Ved videre navigering i systemet er det lagt vekt på å hjelpe brukeren på veien. Et eksempel på slik hjelp er bruk av såkalte *"titles"* i linker. Det innebærer at man får en utdypning av hva linken gjør ved å holde musepekeren over den (Se Figur 4-13).



Figur 4-13. Titles

Et eksempel på hvordan det er prøvd å hindre brukerfeil er at kurs ikke kan slettes hvis det finnes kursinstanser av dem. Vi ser av Figur 4-14 at *"Delete"*-ikonene for kursene *INF100* og *INF234* er inaktive fordi det finnes kursinstanser av dem.

Figur 4-14. Liste over kurs

4.4.3 Konsistent

Ved å være konsistent i utformingen av brukergrensesnittet kan brukerne bruke tidligere lært kunnskap for å utføre nye oppgaver i systemet. Dette er blant annet sikret gjennom konsistent bruk av termer og ikoner for gitte oppgaver. Ønsker man for eksempel å endre informasjon, ser man etter en *"edit"*-knapp og et tilhørende edit-ikon. Alle linker og ikoner i systemet er stilsatt likt, og kommandoer for en gitt side er samlet i en egen boks på siden, *Actions.* (Se Figur 4-15). Brukeren kan derfor forvente å finne relevante kommandoer for inneværende side i denne boksen gjennom hele systemet.

Figur 4-15. Actions-boks

4.4.4 Enkelt

Det enkle er ofte det beste, også når det gjelder brukergrensesnitt. For eksempel er det viktig å finne balansen mellom å maksimere funksjonaliteten til systemet og samtidig beholde enkeltheten i det ved å gradvis "avsløre" informasjon.

Et viktig konsept i *Webucator 3.0* er håndteringen av kursinstanser og deres brukere. I utgangspunktet dreier dette seg om store mengder informasjon som skal presenteres til den som anvender systemet. Denne informasjonen presenteres derfor gradvis, ved først å vise en liste over kursinstanser, med ulike valg for å behandle dem (Se Figur 4-7). Man har der mulighet for å avsløre brukerne av en gitt kursinstans ved å klikke på ikonet *"Users"* i kolonnen kursinstansen befinner seg. Man blir da ledet til en ny side med kommandoer for å behandle brukerne. (Se Figur 4-8)

4.4.5 Bruker-kontrollert

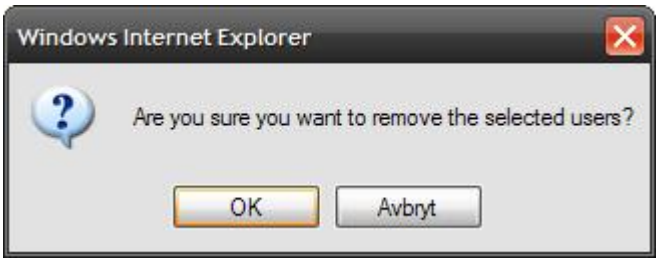
Det er viktig at det er brukeren og ikke systemet som initierer og kontrollerer alle hendelser i systemet. *Webucator 3.0* utfører ingen kommandoer som endrer dataene i systemet uten at brukeren bevisst har bedt om dette.

Dette punktet kan i enkelte tilfeller komme i konflikt med sikkerheten til systemet, og da har sikkerheten blitt prioritert. For eksempel vil *Webucator 3.0* automatisk logges av etter en gitt periode med inaktivitet fra brukerens side, uten at brukeren bevisst logger ut av systemet ved å klikke på linken *"log out"*. Systemet vil likevel i de fleste tilfeller *"huske"* hvor i systemet brukeren befant seg da utloggingen skjedde, og lede brukeren tilbake dit etter ny innlogging.

4.4.6 Tilgivende

Man bør alltid utvikle brukergrensesnitt med tanke på at alle brukere på ett eller annet tidspunkt gjør noe feil. Systemet bør derfor være tilgivende, i den forstand at brukernes handlinger bør være reversible.

Dette er blant annet sikret ved at brukeren ved utførelsen av alle kritiske operasjoner får opp en bekreftelsesdialog. Dette kan være operasjoner som fjerning og sletting av brukere, masseutsendelse av e-post og så videre (Se Figur 4-16).



Figur 4-16. Bekreftelsesdialog

4.4.7 Gi tilbakemeldinger

Brukeren må hele tiden få tilbakemeldinger på handlinger han/hun gjør. Alle handlingene brukeren utfører i systemet blir fulgt av en bekreftelsesside eller en bekreftelsesboks på siden brukeren befinner seg på. Der finnes informasjon om hvilken handling som er utført og eventuelle konsekvenser handlingen har.

For eksempel vil dét at man endrer rollen en bruker har i en kursinstans fra *reader* til *publisher* resultere i at en informasjonsboks angående den utførte handlingen blir synlig over listen over brukere i kursinstansen. (Figur 4-8).

5 Eksisterende sikkerhet og aksesskontroll

Dette kapittelet tar for seg den eksisterende løsningen for sikkerhet og aksesskontroll, før denne oppgaven ble påbegynt. I den forbindelse er det viktig å understreke at det ikke hadde vært implementert en helhetlig løsning for disse spørsmålene i det eksisterende systemet.

Da Yngve Espelid skrev sin oppgave [8] og parallelt utviklet *DPG1.0*, var det ikke hans mål at systemet skulle bli tatt i bruk i fjernundervisningen. Det skulle heller fungere som en prototype for de problemstillingene han tok opp i oppgaven sin. Ingen av dem omhandlet sikkerheten til systemet, men *DPG1.0* hadde likevel implementert en innloggingsmekanisme.

Oppgaven til Preben Solheim [3], som i hovedsak tok for seg *Webucator 2.0* belyste også en del tema knyttet til autentisering av brukere og sikkerhet i *DPG1.0*, men ikke nok til at dette kunne sees på som en helhetlig oversikt.

De neste delkapitlene gir en skisse av de sikkerhetsmekanismene som fantes i det gamle systemet. Det vil også bli gitt en gjennomgang av hvordan brukerinformasjonen fra *Webucator 2.0* ble overført til *DPG1.0.*

5.1 Webucator 2.0

5.1.1 Aksesskontroll

Oppgaven til Preben Solheim [3], gir svært mangelfull informasjon om sikkerhet og aksesskontroll i *Webucator 2.0*, og i og med at det ikke fantes noen kjørende versjon av systemet, eller komplett kildekode, har det ikke vært mulig å lage en komplett oversikt over dette.

Som nevnt i kapittel 2.2.1 håndterte *Webucator 2.0* to typer brukere (roller). Det kan se ut som om selve innloggingen av disse brukerne foregikk ved hjelp av en tidlig versjon av sikkerhetssystemet ACEGI [38], da dette er nevnt i en setning i Solheims oppgave [3].

5.1.2 Database

På let etter ytterligere informasjon, ble databasen til *JAFU-systemene* undersøkt, for å se om den kunne gi bedre svar på hvordan brukerne av *Webucator 2.0* ble håndtert. Det viste seg at brukerinformasjonen var spredd over mange tabeller. Tabellnavn som `temp_roles`, `user_roles`, `user_roles_2` og `tbl_roles` (i samme database) vitnet om at stadig nye utviklere av systemet, grunnet manglende dokumentasjon, hadde måttet lage sine egne løsninger for å få *JAFU-systemene* til å fungere. Det kunne også se ut som om brukerne av *Webucator 2.0* og brukerne av *DPG1.0*

var lagret i samme databasetabell, noe som bidro ytterligere til at databasen var svært uoversiktlig.

5.1.3 Sikkerhetshull

Det ble i *Webucator 2.0*'s levetid ikke avdekket noen sikkerhetshull, men i den forbindelse må det også nevnes at det kun var i bruk i fjernundervisningen våren 2007. I hovedsak ble systemet også bare brukt av én person, som var ansvarlig for fjernundervisningen det semesteret. Vedkommende forlot *JAFU* det påfølgende semesteret, og dette førte til at man ikke lenger hadde tilgang til personer som hadde brukt *Webucator 2.0* aktivt.

5.2 DPG 1.0

5.2.1 Aksesskontroll

Autentiseringen av brukere foregår via Tomcat, satt opp med et *JDBC Realm* [39] og en *PostgreSQL*-database med en tabell som inneholder brukernavn og passord. Oppgaven til denne mekanismen er å finne ut om brukeren har tilgang til systemet i det hele tatt. Etter en eventuell innlogging vil brukeren bli presentert for en liste over presentasjonene som finnes i systemet. En presentasjon kan for eksempel være *"INF100-F – Grunnkurs i programmering"*, altså en kursside for et fag ved fjernundervisningen.

Hvilke presentasjoner brukeren har tilgang til og hva slags tilgang vedkommende har til dem, er definert i XML-filer, basert på de ulike rollene systemet håndterer. (Hva de ulike rollene betyr er beskrevet i kapittel 2.1.3):

- *administrator.xml* – Denne filen inneholder en liste over alle *administrator*ene i systemet.
- *publishers.xml* – Denne filen inneholder en liste over alle *publisher*ene i en gitt presentasjon. I motsetning til *administrators.xml* som det bare finnes en av i hele systemet, finnes det altså en *publishers.xml* for hver presentasjon i systemet.
- *readers.xml* – Denne filen inneholder en liste over alle *reader*e i en gitt presentasjon. På samme måte som for *publisher*e finnes det altså en slik XML-fil for hver presentasjon i systemet.

Eksempel 5-1 viser hvordan XML-filene er utformet. De er **ikke** synkronisert med databasen, og må manuelt oppdateres når nye brukere legges til / slettes i systemet. Dette er nærmere beskrevet i delkapittel 5.3.

```
<?xml version="1.0" ?>
<users>
     <user-name>karianne</user-name>
     <user-name>freejay</user-name>
</users>
```

Eksempel 5-1. Eksempel på filen administrators.xml i *DPG 1.0*

5.2.2 Sikkerhetshull

I Preben Solheims oppgave [3] og i undertegnedes og Karianne Bergs semesteroppgave i faget INF226, høsten 2006 [40] ble det avdekket noen alvorlige sikkerhetshull i *DPG1.0*. I hovedsak skyldtes hullene hvordan applikasjonen var bygget opp.

Som tidligere nevnt var ikke *DPG 1.0* ment som en ferdig implementasjon av en *CMS* [41]. Det var derfor kun implementert en forenklet filhåndtering med flate filer, uten bruk av database. Alle filer en presentasjon trengte ble plassert i en katalog, *repository*. I utgangspunktet kunne man konfigurere hvor denne katalogen skulle ligge på filsystemet gjennom JNDI [42], men for *DPG 1.0*'s del måtte den uansett ligge i *webapps*-katalogen til applikasjonen. Denne katalogen er den som inneholder alle applikasjonene på en *Tomcat*-server. Det er to hovedgrunner til at *repository*-mappen må ligge her. For det første har ikke *DPG 1.0* implementert noen form for strømbasert filnedlasting, slik at filer som skal være tilgjengelig for nedlasting, må være tilgjengelig over *HTTP*. For det andre bruker *DPG 1.0* JSPX-filer for å plassere en del layout-elementer på websidene. Disse filene må kompileres, og for at *Tomcat* skal kunne gjøre dette automatisk, må de ligge i applikasjonens *webapps*-katalog.

De to neste underavsnittene gir en beskrivelse av sikkerhetshullene løsningen over førte til.

Autorisering gjennom skjulte ressurser

Kursrelaterte filer

Vi tenker oss et fjernundervisningskurs i faget *INF101* våren 2008, som blir gjennomført ved hjelp av kurssider i *DPG 1.0*. Slike kurssider har gjerne en fremdriftsplan med spesifiserte ukeplaner for hver uke. De spesifiserte ukeplanene inneholder gjerne linker til forelesningsnotater, relevant kildekode, samt til ukeoppgaver og løsningsforslag til disse. Som beskrevet i avsnittet over vil alle disse ressursene være plassert i *repository*-mappen til applikasjonen på web-serveren og være tilgjengelig over *HTTP*.

Vi kan tenke oss at vi i den spesifiserte ukeplanen for uke 11, finner løsningsforslaget til ukeoppgave 6. Samtidig finner vi oppgaveteksten til

ukeoppgave 7 på samme sted. Linken til løsningsforslaget for ukeoppgave 6 kunne da sett ut som i Eksempel 5-2:

```
http://nettkurs.uib.no/dpg/repository/presentations/inf101fv2008/resources/losningsforslag_ukoeppgave6_inf101.zip
```

Eksempel 5-2. Link til løsningsforslag

En våken student vil da kanskje tenke at siden løsningsforslaget til ukeoppgave 6 ligger der, vil kanskje løsningsforslaget til ukeoppgave 7, som ikke legges ut som link i den spesifiserte ukeplanen før neste uke, også befinne seg der. Vedkommende ville da kanskje prøve å endre linken slik vi ser i Eksempel 5-3:

```
http://nettkurs.uib.no/dpg/repository/presentations/inf101fv2008/resources/losningsforslag_ukeppgave7_inf101.zip
```

Eksempel 5-3. Modifisert link til løsningsforslag

I mange tilfeller ville da brukeren faktisk få lastet ned løsningsforslaget for ukeoppgave 7. Dette skyldes at vanlig praksis ved fjernundervisningen som oftest var å laste opp ressurser tidlig, slik at de ”lå klare” i systemet, til de skulle bli gjort tilgjengelig for studentene gjennom en link i en ukeplan. Dette var mulig gjennom en ”enable”-funksjon i systemet. Etter opplasting måtte ressursene enables for å bli synlig på sidene. Men som vist over var ressursene tilgjengelige, selv om de ikke var synlige.

En enkel løsning på problemet over ville selvsagt kunne være å vente med å laste opp ressurser til de faktisk skulle brukes. Dette ville være noe mer tungvindt for de kursansvarlige, men ville sikre at studentene ikke fikk muligheten til å jukse på ukeoppgaver ved å bruke løsningsforslag før innleveringsfristen. Så enkelt er det imidlertid ikke. Det er nemlig ikke bare kursrelaterte filer som befinner seg i *repository*-mappen.

Systemrelaterte filer

Et nytt eksempel kan være filen *administrators.xml* som ble beskrevet i delkapittel 5.2.1. Den inneholder som kjent en liste over alle administratorene til systemet. Ved å taste inn linken fra Eksempel 5-4 i en hvilken som helst nettleser vil man bli presentert for en liste over administratorene i *DPG1.0*. Selv om filen ikke inneholder informasjon om brukernes passord, ville en eventuell inntrenger være et godt stykke nærmere uautorisert tilgang til systemet ved å få tak i brukernavnene.

```
http://nettkurs.uib.no/dpg/repository/administrators.xml
```

Eksempel 5-4. Eksempel på link til liste over administratorer

Det vil kunne argumenteres for at det skal litt til at man tilfeldigvis havner inn på URL-en i Eksempel 5-4. I den forbindelse er det viktig å nevne at informasjon om *administrators.xml* er gitt i Yngve Espelids oppgave [8], som er et offentlig publisert dokument. Det finnes også flere hacker-verktøy som ved hjelp av ordlister kan lete seg frem til URL-er på servere.

Opphavsrettighetsbeskyttet materiale

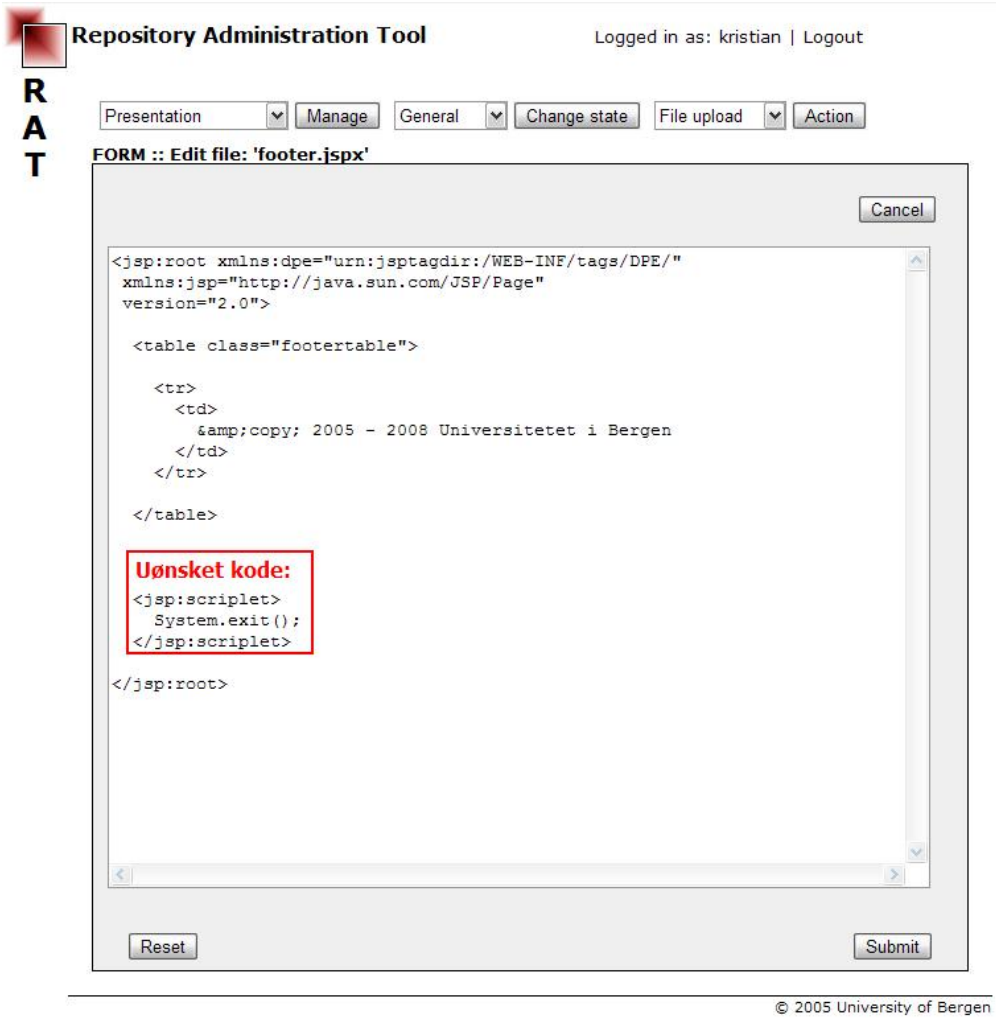
Det er også verdt å nevne at det ofte legges ut opphavsrettighetsbeskyttet materiale i systemet til bruk i fjernundervisningen. At disse ressursene i praksis kan nås fra en hvilken som helst nettleser rundt om på kloden, kan heller ikke sies å være heldig.

Problemene som beskrives i avsnittene over skyldes det man kaller *"Security Through Obscurity"* [43]. Altså at man implementerer sikkerhet gjennom å skjule ulike ressurser. Svakheten er som vi ser at disse ressursene, om de ikke også beskyttes på annen måte, kan oppdages av uvedkommende. Det er ikke kjent at noen har fått tak i ressurser i *JAFU-systemene* på denne måten, men det kan absolutt ikke utelukkes. Flere av studentene som har tatt kurs ved fjernundervisningen og brukt systemene har bred IT-bakgrunn. For å unngå juks grunnet denne sikkerhetsbristen blir det viktig å beskytte systemets ressurser bedre i *DPG 2.0*. Sagt på en mer presis måte, skal det ikke være mulig for en bruker å få tak i ressurser vedkommende ikke er ment å ha tilgang til.

Denial of Service (DoS) angrep ved hjelp av JSPX

Som tidligere nevnt bruker *DPG 1.0* JSPX-filer til å plassere ulike elementer på nettsidene den skal presentere. Disse filene defineres gjennom administreringsverktøyet til *DPG 1.0*, også kalt *RAT*. Det er ikke lagt begrensninger på hvilke tagger som er tillatt å bruke i JSPX-filene.

Som vi ser av Figur 5-1 er det altså ingenting i veien for å bruke *jsp:scriplet*-taggen til å utføre systemkommandoer. Gjennom et forsøk i forbindelse med Preben Solheims oppgave [3], ble det avslørt at innsettelsen av koden i den rød boksen i Figur 5-1 førte til at hele *Tomcat*-serveren ble skrudd av, når siden ble lastet i nettleseren.



Figur 5-1. Uønsket kode i RAT.

Hvis uvedkommende skulle få administratortilgang til systemet, for eksempel gjennom å utnytte sikkerhetshullet som ble beskrevet tidligere i dette delkapittelet, vil det altså være fullt mulig, å utføre et såkalt *Denial of Service* angrep [44] mot systemet. Dette innebærer at systemet blir satt helt ut av spill og ikke har mulighet til å tilby de tjenester det normalt gjør til sine brukere.

Med bakgrunn i den begrensede bruken *JAFU-systemene* har vært utsatt for i de årene det har vært operativt har det ikke vært overhengende fare for et angrep av det slaget som er beskrevet over, og det har heller ikke vært registrert noen slike. I utviklingen av *DPG 2.0* vil det allikevel være viktig å ta høyde for at systemet i fremtiden kan få betydelig større utbredelse. Det bør derfor enten settes en begrensning på hvilke tagger som skal kunne brukes i utformingen av JSPX-filer, eller eventuelt finne en helt annen løsning for å plassere elementer på sidene, uten bruk av JSPX.

5.3 Utveksling av brukerinformasjon

5.3.1 Tilordning av brukere til en ny presentasjon

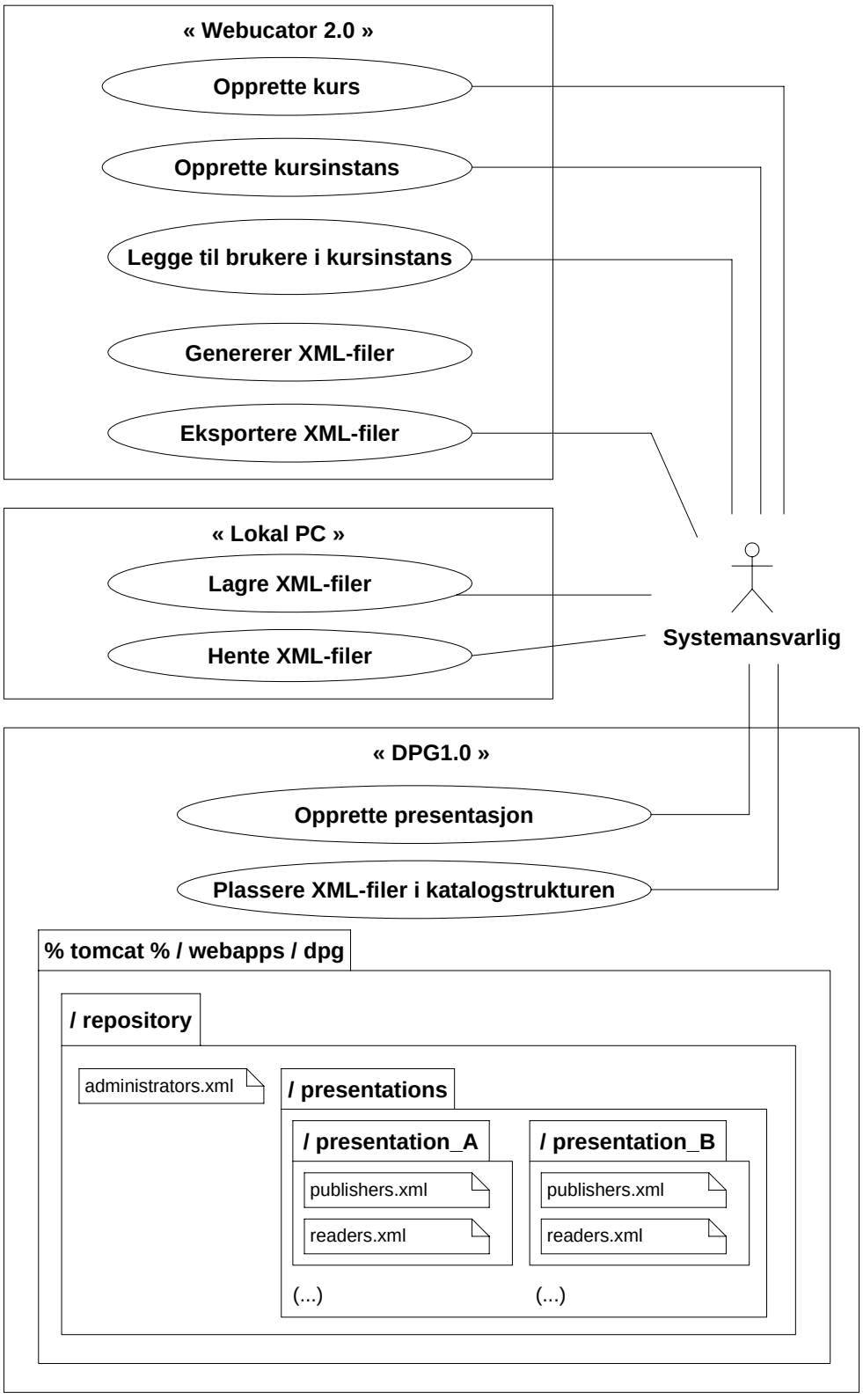
Figur 5-2 viser hvordan systemansvarlig kunne opprette en presentasjon *DPG 1.0,* og tilordne brukere til presentasjonen blant annet ved hjelp av *Webucator 2.0.*

Etter å ha opprettet en kursinstans (For eksempel *"INF100 – Grunnkurs – våren 2009"*) og lagt til brukere i denne, måtte systemansvarlig hente XML-filene *readers.xml* og *publishers.xml*, som inneholder informasjon om hvilke brukere som har hvilken rolle i kursinstansen (og den tilhørende presentasjonen i *DPG 1.0*). Disse filene ble generert av *Webucator 2.0,* og måtte eksporteres og lagres på en lokal PC. Deretter måtte systemansvarlig plassere disse XML-filene i katalogstrukturen til *DPG 1.0* på webserveren. Hvis systemansvarlig i tillegg hadde opprettet en ny administrator, måtte også XML-filen *administrators.xml* eksporteres til en lokal PC og derfra kopieres til *repository*-mappen til applikasjonen på webserveren.

Det er flere ulemper med løsningen over. For det første trenger systemansvarlig inngående kunnskaper om filstrukturen til systemet for å kunne lage nye presentasjoner og knytte brukere til disse. I tillegg kan det for eksempel være lett å plassere XML-filene i feil presentasjon og dermed gi brukerne gale rettigheter i systemet.

Et annet problem med løsningen er at den kan bli relativt tidkrevende og komplisert ved behandling av flere presentasjoner. Vi kan for eksempel se for oss en situasjon i fjernundervisningssammenheng. *DPG 1.0* skal ha 3 administratorer. Videre skal det være 5 presentasjoner med ca 2 *publisher*e og ca 20 *reader*e i hver. Disse er knyttet opp til 5 forskjellige kursinstanser i *Webucator 2.0.* Vi trenger altså én *administrators.xml*, 5 *publishers.xml* og 5 *readers.xml*, totalt 11 XML-filer som alle skal plasseres riktig i DPG1.0's katalogstruktur.

Det blir også åpenbart problematisk å legge til brukere til kursinstansene/presentasjonene i ettertid. Man må altså da i tillegg til å legge dem til kursinstansen(e) brukeren skal ha tilgang til i *Webucator 2.0* også oppdatere XML-filene der det trengs. Det vil bli sett nærmere på dette i neste delkapittel.



Figur 5-2. Tilordning av brukere til en presentasjon i DPG 1.0

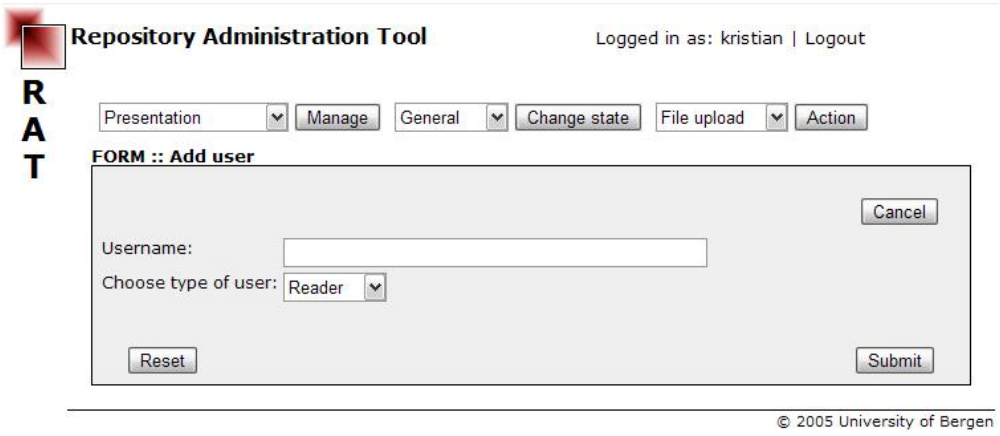
5.3.2 Legge til ekstra brukere i en presentasjon

I fjernundervisningssammenheng er det svært vanlig at man trenger å legge til brukere til en presentasjon i flere omganger. Et eksempel på dette kan

være at studenter som er etterinnmeldt på et kurs ikke har fått tilgang ved kursstart, og at man derfor trenger å gi dem det på et senere tidspunkt.

For å løse problemstillingen over må man legge til brukeren i kursinstansen vedkommende skal ha tilgang til i *Webucator 2.0*. For å gi *DPG 1.0* "beskjed" om at brukeren skal ha tilgang til en gitt presentasjon har man nå to valg:

- Man kan oppdatere XML-filene i presentasjonen brukeren skal ha tilgang til. For eksempel *readers.xml* i presentasjonen for *"INF100 – Grunnkurs – våren 2009"*, hvis brukeren er student i den kursinstansen. Man må da altså eksportere den oppdaterte XML-filen fra *Webucator 2.0* og lagre den lokalt på en PC. Videre må man plassere filen på rett plass i *DPG1.0*'s katalogstruktur som tidligere beskrevet.
- Man kan i stedet bruke DPG 2.0's innebygde løsning for å legge til brukere. Denne funksjonen finner man i *DPG*-ens administrasjonsverktøy *RAT* (Se Figur 5-3). Her kan man legge til én og én bruker i en presentasjon og gi vedkommende rollen *reader* eller *publisher*. *RAT* vil da legge til brukeren i riktig XML-fil i katalogstrukturen til *DPG*-en. Administratorer kan ikke legges til på denne måten.



Figur 5-3. Funksjon for å legge til en ny bruker i RAT

Ved første øyekast kan det se ut som om funksjonen som er vist i Figur 5-3 forenkler oppgaven med å legge til ekstra brukere i en presentasjon mye. Men det er viktig å huske på at brukeren likevel må legges til i en kursinstans i *Webucator 2.0* for å få tilgang til systemet i det hele tatt. XML-filene styrer bare tilgang til hver enkelt presentasjon.

Muligheten for å legge til brukere til en presentasjon i *RAT* henger egentlig igjen fra gammelt av. Det er viktig å huske på at det ikke fantes noen *Webucator 2.0* da *DPG1.0* ble ferdigstilt. Da *Webucator 2.0* tok på seg ansvaret med brukerhåndtering for *DPG1.0* burde man kanskje ha koblet ut

denne funksjonen. For det første er den med på å duplisere ansvarsområdene til de to applikasjonene, men kanskje mer alvorlig er det at den kan gjøre informasjonen som er tilgjengelig i de to systemene usynkron og dermed korrupt.

Vi kan tenke oss brukeren Student_A. Han er bruker i kursinstansen *"INF100 – Grunnkurs – våren 2009"* i *Webucator 2.0*, og har rollen *reader*, brukernavnet hans er derfor tilstede i *readers.xml* i presentasjonen med samme navn i *DPG1.0*. Neste semester blir Student_A etterinnmeldt på kurset *"INF101 – Videregående kurs – høsten 2009"*. Systemansvarlig som skal gi Student_A tilgang til presentasjonen for denne kursinstansen bruker *RAT*'s funksjon for å legge ham til som *reader* der. Systemansvarlig gjør ingenting i *Webucator 2.0*. (Student_A har allerede tilgang til selve systemet i og med at han var student i *"INF100(…)"*, så han får logget inn.)

Konsekvensen av det som står i avsnittet over er at Student_A nå har tilgang til presentasjonen *"INF101(…)"* i *DPG1.0*, men vi finner ham ikke i listen over brukere til den tilhørende kursinstansen i *Webucator 2.0*, og informasjonen der er dermed korruptert ved at den ikke gir oversikt over alle brukerne til kursinstansen.

5.4 Oppsummering

Gjennom analysen av den eksisterende løsningen for sikkerhet og aksesskontroll i *DPG 1.0* og *Webucator 2.0* ble det avslørt en del problemer som må løses, i den grad det lar seg gjøre, ved implementasjonen av *DPG2.0* og *Webucator 3.0*

- Man bør tette de sikkerhetshullene som ble beskrevet i dette kapittelet. (Se delkapittel 5.2.2)
- Man bør ha en mer helhetlig tilnærming til sikkerhet i systemene. Bruk av et sikkerhetsrammeverk kan være et godt utgangspunkt for en slik tilnærming.
- Brukere av *Webucator* og *DPG* bør lagres forskjellige steder for å unngå forvirring og for å skape en løsere kobling mellom systemene.
- Man bør lage en ny løsning for utveksling av informasjon mellom systemene. Ideelt sett bør systemene kunne kommunisere med hverandre uten innblanding fra en systemansvarlig.
- Man bør unngå duplisering av arbeidsoppgaver i systemene og på den måten sikre at brukerinformasjon ikke kan korrupteres.

6 Ny løsning for sikkerhet og aksesskontroll

6.1 Valg av sikkerhetsløsning

Erfaringene fra *Webucator 2.0* og *DPG 1.0* viste med all tydelighet behovet for en mer helhetlig sikkerhetsløsning for *JAFU*-systemene. I og med at begge de nye systemene (*Webucator 3.0 og DPG 2.0*) er implementert ved hjelp av *Spring Framework*[12], ble det derfor naturlig å se hva det rammeverket kunne tilby av sikkerhetsfunksjonalitet. *Spring* tilbyr et eget sikkerhetsrammeverk *Spring Security* [38]. Dette var tidligere kjent under navnet *ACEGI* før det ble innlemmet i *Spring*-familien. Denne innlemmelsen fant ikke sted før et stykke ut på vårparten i 2008. I denne oppgaven vil det derfor refereres til systemet som *ACEGI*. Versjonen som i skrivende stund benyttes er 1.0.5.

ACEGI's tette integrasjon med *Spring* gjør at det egentlig ikke er noen reelle alternativer til dette rammeverket når man skal implementere sikkerhet i dette rammeverket. Enkelte ville peke på *SecurityFilter* [45] som et alternativ som i noen tilfeller kan tilby et noe enklere oppsett av sikkerhetsfunksjonalitet enn ACEGI. Dette systemet var likevel ikke noen reell kandidat, da det sist ble oppdatert i desember 2004. Liten aktivitet på brukerforumet vitnet også om at det kunne bli vanskelig å få hjelp om en støtte på problemer under implementasjonen. *ACEGI* er etter hvert blitt veldig utbredt og aktiviteten på brukerforaene er betydelig større enn for *SecurityFilter* sin del. Utbredelsen, og innlemmelsen under *Spring*-paraplyen har også ført til at dokumentasjonen for systemet begynner å bli bra. Det har nemlig vært et velkjent problem for OpenSource-prosjekter at dokumentasjonen ofte kan være mangelvare.

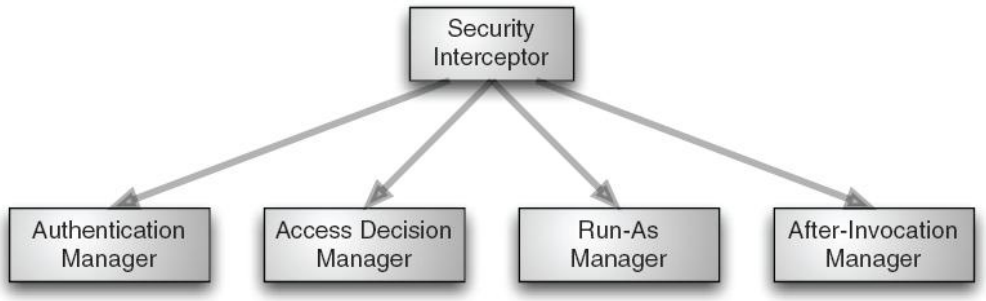
En siste faktor som spilte inn var at det allerede fantes en del kompetanse på *ACEGI* på *JAFU*-prosjektet, da Bjørn Christian Sebak brukte dette systemet i et fjernundervisningsprosjekt i samarbeid med Senter for Etter- og VidereUtdanning (SEVU) i 2006/2007. Erfaringene derfra viste at selv om *ACEGI* er meget omfattende og relativt komplekst, var det mindre problemfylt å sette opp enn man først skulle tro. I tillegg ble det bekreftet at det var god hjelp å få i ulike brukerforum de gangene man stod fast.

6.2 ACEGI

6.2.1 Oppbygging

Figur 6-1 gir en grov oversikt over hvordan *ACEGI* er bygget opp. ACEGI tilbyr standardimplementasjoner av de fleste av systemkomponentene, men det er også mulig for utviklere å lage egne løsninger ved å implementere

ulike *interface* [46] som *ACEGI* tilbyr. Videre er det gitt en nærmere beskrivelse av *ACEGI*'s hovedkomponenter.



Figur 6-1. *ACEGI*: oversikt over hovedkomponenter**. [47]**

Security Interceptor

Security Interceptor kan sees på som en dør som beskytter en gitt ressurs i systemet. I bunn og grunn gjør ikke *Security Interceptor* noe annet enn å avskjære forespørsler fra å nå en ressurs. Ansvaret for å bestemme om brukeren bak forespørselen skal få tilgang til ønsket ressurs blir delegert til de ulike managerne vi ser i Figur 6-1.

Authentication Manager

Denne manageren har ansvaret for å finne ut hvem som prøver å få tilgang til en ressurs. Den krever at det oppgis *principals* (typisk et brukernavn) og *credentials* (typisk et passord) for å avgjøre dette spørsmålet. *Principal* definerer da hvem vedkommende er mens *credential* underbygger at man er den man gir seg ut for å være. Hva *Authentication Manager* sjekker *principals*/*credentials* opp mot kommer alt an på hvordan denne er implementert. *ACEGI* tilbyr ferdig implementerte løsninger for de fleste behov, som for eksempel sjekk av *principals*/*credentials* opp mot en database.

Access Decisions Manager

Access Decisions Manager avgjør, som navnet tilsier om vedkommende som er autentisert gjennom *Authentication Manager* skal få tilgang til ressursen vedkommende ønsker. Igjen er det mulighet for utallige implementasjoner av denne manageren, og igjen tilbyr *ACEGI* standardiserte løsninger, for eksempel basert på roller. Hvis en ressurs kun skal kunne nås av brukere med rollen *administrator* vil *Access Decisions Manager* sjekke om den autentiserte brukeren har denne rollen før brukeren får tilgang til ønsket ressurs.

Run-as Managers

Disse managerne er valgfrie og kan tilby beskyttelse av ulike objekter i en gitt ressurs. Vi kan se for oss at *Access Decisions Manager* har gitt en bruker tilgang til en nettside, men at en del av nettsiden krever ytterligere autentisering for at brukeren skal kunne bruke denne. Det kan for eksempel

være et forum som krever et annet brukernavn og passord enn det som ble gitt til *Access Decisions Manager*. Da kan *Run-as Managers* brukes til å erstatte brukerens autentisering med en autentisering som gir tilgang til beskyttede objekter dypere ”nede” i systemet.

After-invocation Manager

På samme måte som *Run-as Managers* er denne manageren valgfri. Den fungerer også på en litt annen måte enn de andre managerne. Den tilbyr nemlig sikkerhet **etter** at en gitt ressurs er aksessert. Den kan evaluere returverdien fra ressursen og avgjøre om brukeren som aksesserte ressursen er autentisert for å få tilgang til denne returverdien.

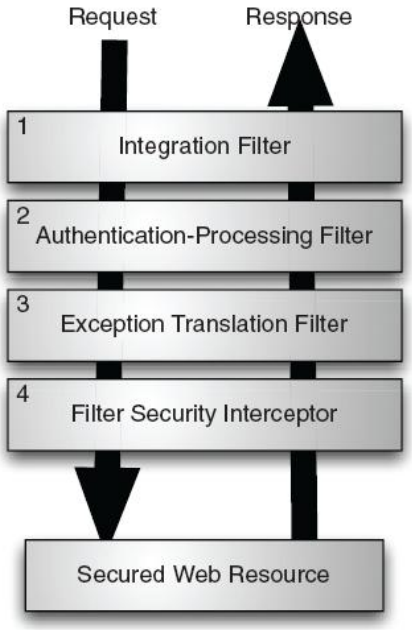
6.2.2 Filtre

Spring sikrer webapplikasjoner gjennom servlet-filtre [48]. Filtrene fanger opp servlet-forespørsler for å autentisere dem og framtvinge sikkerhetshåndtering.

Som vi ser av Figur 6-2 er filtrene toveis. En forespørsel blir sendt gjennom filtrene i en retning, mens responsen blir sendt tilbake gjennom de samme filtrene, men i motsatt rekkefølge.

Standardfiltre

Videre er det nå gitt en kort beskrivelse av de ulike filtrene som er vist i Figur 6-2. Dette er de fire filtrene som er **påkrevd**. Spring tilbyr minst 17 ulike filtre, i tillegg er det muligheter for utviklere å definere egne filtre.



Figur 6-2. Filtre i *ACEGI* **[47]**

Integration Filter

Dette filteret er ansvarlig for å hente eventuell eksisterende autentiserings-informasjon. I de fleste tilfeller er denne informasjonen, om den finnes, lagret i *HTTP session*. Dette filteret er nødvendig fordi HTTP er tilstandsløs og ikke har noen mulighet til å ”huske” en bruker fra en forespørsel til den neste.

Authentication-Prosessing Filter

Dette filteret avgjør om en forespørsel er en autentiseringsforespørsel. Hvis så er tilfelle henter filteret ut brukerinformasjonen (typisk brukernavn og passord) fra forespørselen og videresender informasjonen til *Authentication Manager* for å avgjøre brukerens identitet. Hvis det ikke dreier seg en om autentiseringsforespørsel utfører filteret ingen handlinger og bare videresender forespørselen videre nedover i filterkjeden.

Exception Translation Filter

Den eneste oppgaven til dette filteret er å oversette ulike unntak som kan bli kastet til egnede HTTP-responser. Hvis det blir kastet et `AuthenticationException` vil forespørselen bli videresendt til et såkalt *authentication entry point*. Dette kan for eksempel være en login-side til systemet. Hvis et `AccessDeniedException` blir kastet er standardhandlingen at det blir returnert en HTTP 403-feil til nettleseren.

Filter Security Interceptor

Det siste obligatoriske nødvendige filteret har ansvaret for å avgjøre om en bruker bak en forespørsel har nødvendige rettigheter til å få aksess til ønsket ressurs. Filteret "jobber sammen med" *Authentication Manager* og *Access-desicion Manager* for å gjøre dette.

Rekkefølgen på filtrene

I hvilken rekkefølge man ønsker at forespørsler og responser skal passere gjennom de ulike filtrene definerer man i en filterkjede, i ACEGI kalt `filterChainProxy`. Denne kjeden er, sammen med de ulike filtrene definert i en XML-fil som brukes til å spesifisere alle de ulike innstillingene ACEGI skal ha i en applikasjon. I JAFU-systemene heter denne filen *applicationContext-security.xml*. Vi ser et utdrag av denne, som viser filterkjeden til *DPG 2.0* i Eksempel 6-1. Vi kjenner igjen de fire standardfiltrene[1] (markert i rødt), i tillegg til to andre ACEGI-filtre.

```
<bean id="filterChainProxy"
      class="org.acegisecurity.util.FilterChainProxy">
  <property
      name="filterInvocationDefinitionSource”>
    <value>

      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /**=httpSessionContextIntegrationFilter,
      logoutFilter, authenticationProcessingFilter,
      securityContextHolderAwareRequestFilter,
      exceptionTranslationFilter,
      filterInvocationInterceptor

    </value>
  </property>
</bean>
```

Eksempel 6-1. Filterkjeden til *ACEGI* i *DPG 1.0*.

[1] `httpSessionContextIntegrationFilter` er det integrasjonsfilteret som brukes i webapplikasjoner.

6.3 Webucator 3.0

6.3.1 Beskyttelse av ressurser

I *Webucator 3.0* er alle sidene (det vil si kontrollerne som genererer dem) beskyttet ved hjelp av *ACEGI*, bortsett fra innloggingssiden. Som vi ser av Eksempel 6-2 gjøres dette ved at kontrollerne som skal beskyttes legges i et URL-mønster med *"/admin/"* før referansen til selve kontrolleren. Vi ser også at login-siden ikke ligger under *"/admin/"*.

```
<bean name="/login.htm"
 class="no.uib.ii.webucator.web.controller.LoginController"
/>

<bean name="/admin/index.htm"
 class=
 "no.uib.ii.webucator.web.controller.FrontpageController"
/>
```

Eksempel 6-2. Definisjon av kontrollere i *applicationContext-servlet.xml*.

Webucator 3.0 har bare én type autentisert bruker, og den gis rollen *admin* i systemet. For å sikre at kun brukere med rollen *admin* får tilgang til den beskyttede delen av systemet (kontrollerne med *"/admin"*-prefiks) må vi få ACEGI til å filtrere ut forespørsler mot disse kontrollerene og autentisere brukeren. Som vi så i delkapittel 6.2.2 er det standardfilteret *FilterSecurityInterceptor* som utfører denne oppgaven. Eksempel 6-3 viser hvordan dette filteret ved hjelp av egenskapen *objectDefinitionSource* angir at kun brukere med rollen *admin* skal ha tilgang til kontrollere med prefikset "admin" (Markert i rødt).

```
<bean id="filterInvocationInterceptor"
 class= "org.acegisecurity.intercept.web.
 FilterSecurityInterceptor">
 (...)

  <property name="objectDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /admin/**=ROLE_ADMIN
    </value>
  </property>
</bean>
```

Eksempel 6-3. Definisjon av *FilterSecurityInterceptor*.

Som beskrevet i delkapittel 6.2.2 får *FilterSecurityInterceptor* "hjelp" av blant annet *Authentication Manager* (Se 6.2.1) til å avgjøre om brukeren som sender en forespørsel mot en beskyttet kontroller har rollen som er nødvendig

for å få tilgang. Det neste delkapittelet tar for seg hvordan dette er satt opp i *Webucator 3.0*

6.3.2 Aksesskontroll

Authentication Manager benytter ulike *providere* for å autentisere en bruker. En *provider* tilbyr en oversikt over hvordan *Authentication Manager* skal gå frem for å finne ut hvem som har tilgang til et system og hvilken rolle brukeren har. Som vi ser av Eksempel 6-4 bruker *Webucator 3.0 provider*en `daoAuthenticationProvider`. Egenskapene `userDetailsService` og `passwordEncoder` gir informasjon om henholdsvis hvor brukerinformasjonen kan finnes og hvilken passord-enkoding som er benyttet.

```
<bean id="daoAuthenticationProvider"
 class="org.acegisecurity.providers.
 dao.DaoAuthenticationProvider"
>

  <property name="userDetailsService"
    ref="userDetailsService"
  />
  <property name="passwordEncoder"/>
    <bean class=”org.acegisecurity.providers.
          encoding.ShaPasswordEncoder”
    />
  </property>
  (...)
</bean>
```

Eksempel 6-4. Definisjon av *daoAuthenticationProvider*.

Eksempel 6-5 viser definisjonen av *userDetailsService* for *Webucator 3.0*. Vi ser at informasjonen om brukerne er definert i en *InMemoryDaoImpl*. Det vil si at de er direkte oppført i XML-filen (markert i rødt). Vi ser brukernavnene etterfulgt av passordet (kodet med valgt *passwordEncoder*) og rollen brukeren har.

Denne løsningen er i hovedsak valgt for å skille brukerne av *Webucator 3.0* fra dem i *DPG2.0* (som er plassert i en database). I de gamle systemene var denne informasjonen lagret i samme databasetabell, noen som bidro til at systemene var uoversiktlige å håndtere. Den nye løsningen gjør det også relativt enkelt å legge til brukere i *Webucator 3.0*, uten bruk av SQL-spørringer mot en database.

Om det i fremtiden skulle bli så mange brukere av *Webucator 3.0* at denne løsningen ikke lenger er hensiktsmessig, vil man med enkle grep for eksempel kunne gå over til autentisering mot en egen *Webucator 3.0*-database, ved å bytte ut *InMemoryDaoImpl* med en *JdbcDaoImpl* og konfigurere denne til å autentisere mot riktig database.

```
<bean id="userDetailsService"
 class="org.acegisecurity.userdetails.
 memory.InMemoryDaoImpl"
>
  <property name="userMap">
    <value>

brsseb=917689847d186cacbfbf2ef61c658433cf3d165a,ROLE_ADMIN
loevik=fc188ba892720a1873adfc5164e377a7f7baab2d,ROLE_ADMIN
jafu=e67d5ab3c076777aed47ffcde7f6030af42a7ab8,ROLE_ADMIN

    </value>
  </property>
</bean>
```

Eksempel 6-5. Definisjon av *userDetailsService*.

6.4 DPG 2.0

6.4.1 Begrensning av tilgang til delsystemer

Som det ble beskrevet i delkapittel 3.2 består *DPG 2.0* av ulike delsystemer, der fire av dem interakterer med brukeren: *lobby, PV, PCE* og *PM*. Tilgangen til sidene (kontrollerne som genererer dem) er begrenset på samme måte som i *Webucator 3.0* (Se Eksempel 6-6).

```
<property name="objectDefinitionSource">
  <value>

    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT

    /lobby/presentations.htm=ROLE_AUTHENTICATED_USER,
                             ROLE_AUTHENTICATED_ADMIN
    /pv/**=ROLE_AUTHENTICATED_USER,
           ROLE_AUTHENTICATED_ADMIN
    /pce/**=ROLE_AUTHENTICATED_USER,
            ROLE_AUTHENTICATED_ADMIN
    /pm/**=ROLE_AUTHENTICATED_ADMIN

  </value>
</property>
```

Eksempel 6-6. Definisjon av *objectDefinitionSource*.

Vi ser at delsystemet *Lobby* (som blant annet inneholder innloggingsskjemaet) i utgangspunktet er åpent, men at kontrolleren *presentations.htm* er beskyttet. Dette er kontrolleren som genererer en liste over tilgjengelige presentasjoner i *DPG* **etter** innlogging.

I denne sammenhengen representerer rollen *Authenticated_User* både *publishere* og *readere*, mens *Authenticated_Admin* representerer rollen *administrator*. Dette kan virke noe uoversiktlig ved første øyekast, men har sin naturlige forklaring: Det vi her snakker om er altså tilgangen til de ulike

delsystemene. I den forbindelse er det snakk om to typer brukere. *Authenticated_Admin* har tilgang til alt, inkludert *PM*, mens andre autentiserte brukere i utgangspunktet har tilgang til det meste, utenom *PM*.

Noen spør seg da kanskje om hvorfor ikke alt kan begrenses på del-system-nivå. Altså at *publishers* har tilgang til *PCE* og *PV*, mens *readers* kun har tilgang til *PV*. Dette er ikke mulig fordi brukere kan være *publishere* i én presentasjon og *readere* i en annen. Det er altså ikke tilstrekkelig å styre tilgangen på del-system-nivå. For *readere* og *publishere* må tilgangen også begrenses på per-presentasjon-nivå. Hvordan denne problemstillingen er løst er beskrevet i det neste delkapittelet.

6.4.2 Aksesskontroll

Som det har vært nevnt blant annet i delkapittel 6.3, har *Authentication Manager* en viktig rolle når det gjelder å autentisere brukere. Det ble også vist i delkapittel 6.3.2, hvordan *Authentication Manager* i *Webucator 3.0* brukte en *provider* for å hente brukerinformasjon. For *DPG 2.0* er dette derimot ikke en tilstrekkelig løsning. Som det ble nevnt i forrige delkapittel, holder det ikke at en bruker blir autentisert med en gitt rolle til systemet. *DPG 2.0* trenger også informasjon om hvilke presentasjoner brukeren skal ha tilgang til, og om brukeren skal ha *reader*- eller *publisher*-rolle i hver enkelt presentasjon. Dette har ikke *ACEGI* forståelig nok noen ferdig implementasjon for, men som tidligere nevnt er det sjelden noe problem å integrere egne løsninger med *ACEGI*-rammeverket.

Eksempel 6-7 viser at *DPG 2.0* bruker en *Authentication Manager* som er plassert i *Webucator 3.0*. Som beskrevet i tidligere kapitler er det i *Webucator* informasjonen om hvilke brukere som har hvilke roller i de ulike presentasjonene, befinner seg. Fordelen med denne løsningen er beskrevet nærmere i delkapittel 6.4.3.

```
<bean id="authenticationManager"
 class="org.springframework.remoting.httpinvoker.
 HttpInvokerProxyFactoryBean"
>
  <property name="serviceUrl"
   value="http://jafutest.ii.uib.no:8080/
   webucator3.0/authenticationWebService.htm"
  />

  <property name="serviceInterface"
   value="org.acegisecurity.AuthenticationManager"
  />
</bean>
```

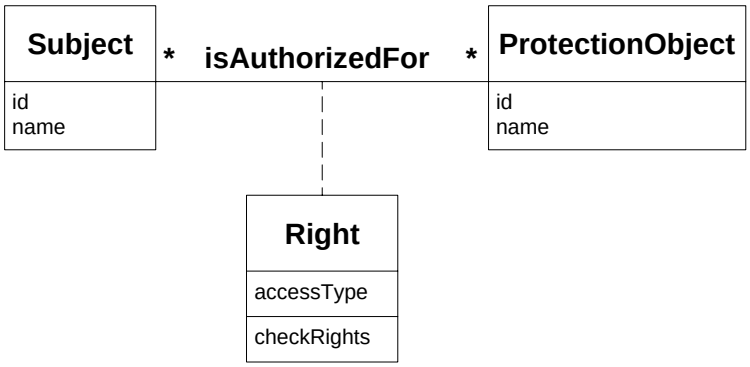
Eksempel 6-7. Remote *Authentication Manager* i *DPG 2.0.*

Bruk av designmønstre

Authorization

Selve autoriseringen av brukerne er basert på design-mønsteret [49] *Authorization: "Dette mønsteret beskriver hvem som er autorisert for tilgang til spesifikke ressurser i et system, i et miljø der det fins ressurser som trenger slik aksesskontroll. Det indikerer for hver aktive entitet som kan aksessere ressurser, hvilken ressurs den kan aksessere og på hvilken måte den kan aksessere den.*

Måten dette gjøres på er som Figur 6-3 beskriver at man når for eksempel en bruker (som Subject) ønsker å aksessere en ressurs i et system (ProtectionObject) , må den ha den rette autorisasjonen for dette (Right).

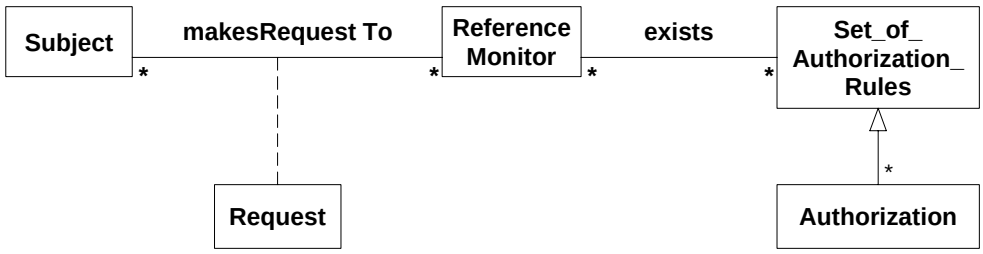


Figur 6-3. Klassemodell for *Authorization*.

Reference Monitor

Selv om *Authorization* sier noe om på hvilken måte en bruker kan aksessere en beskyttet ressurs, trenger man en måte å forsikre seg om at systemet tvinger brukeren til å aksessere beskyttede ressurser på denne måten. I den forbindelse er det vanlig å implementere en form for *Reference Monitor* [49]: *"En implementasjon av dette mønsteret skal håndheve deklarerte aksessrestriksjoner når en aktiv entitet (Subject) forespør en ressurs (ProtectionObject). Mønsteret beskriver hvordan man definerer en abstrakt prosess som avskjærer alle forespørsler og sjekker om de har nødvendig autorisasjon for å aksessere ønsket ressurs."*

Figur 6-4 viser hvordan forespørsler blir sjekket mot et autoriseringsobjekt gjennom en *Reference Monitor*. Som beskrevet tidligere i oppgaven blir alle forespørsler i *DPG 2.0* allerede avskjært ved hjelp av *ACEGI*'s ulike filtre. Filteret *Security Filter Interceptor* bruker deretter blant annet *Authentication Manager* for å avgjøre om brukeren bak en forespørsel skal få tilgang til ønsket ressurs. For at løsningen skal fungere i henhold til *Reference Monitor* mønsteret, trenger vi et objekt som *ACEGI* kan sjekke autentiseringsinformasjon opp mot (Authroization-objektet i Figur 6-4).

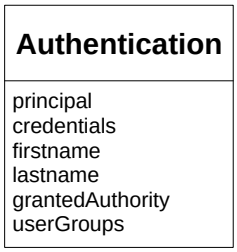


Figur 6-4. Klassemodell for *Reference Monitor*.

Interfacet `AuthenticationManager` i *ACEGI* spesifiserer en metode `autenticate(…)` som returnerer et `Authentication`-objekt. *Autentication Manageren* som brukes i *DPG 2.0* implementerer dette interfacet. `autenticate(…)`-metoden er derfor implementert på en slik måte at `Authentication`-objektet som returneres fra denne metoden inneholder nok informasjon til at *Autentication Manageren* kan avgjøre om en bruker skal få tilgang til ønsket ressurs. `Authentication`-objektet vil dermed fungere som *Authorization*-objektet som er angitt i Figur 6-4.

Figur 6-5 viser de ulike feltene i `Authentication`-objektet:

- principal/credentials som spesifiserer henholdsvis brukernavn og passord.
- firstname/lastname, fornavnet og etternavnet til brukeren, er strengt tatt ikke nødvendig for selve autoriseringen, men kan være nyttig å ha tilgang til, for eksempel å generere en velkomstmelding til vedkommende etter innlogging.
- grantedAuthority spesifiserer hvilken tilgang brukeren skal ha til de ulike delsystemene. Som vi så i delkapittel 6.4.1 kan dette være to ulike verdier: AUTHENTICATED_USER eller AUTHENTICATED_ADMIN.
- userGroups spesifiserer hvilken tilgang brukeren har til de ulike presentasjonene i systemet. Dette feltet inneholder en liste over kursinstansene brukeren er medlem av, og hvilken rolle brukeren har i hver av dem, enten *reader* eller *publisher*. Hvis brukeren er en *administrator*, vil userGroups inneholde en liste over alle kursinstansene i systemet.



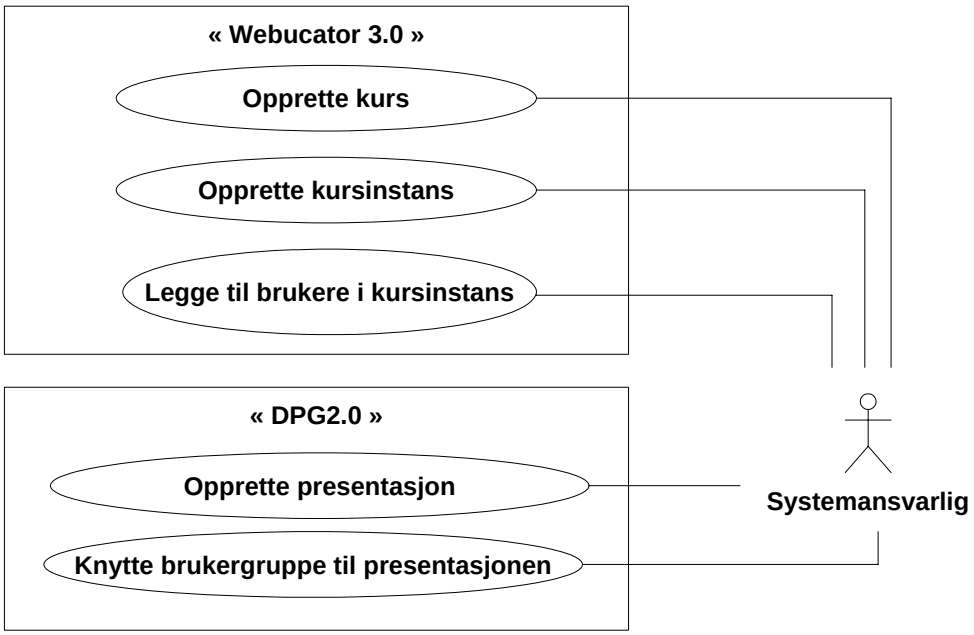
Figur 6-5. Innhold i *Authentication*-objektet.

Som beskrivelsen av `Authentication`-objektet viser, finnes dataene som trengs for å opprette dette objektet i *Webucator 3.0*. Det ble også nevnt i delkapittel 6.4.2 at *Authentication Manager*en som *DPG2.0* bruker befinner seg i *Webucator 3.0*. Hvordan autentiseringsinformasjonen opprettes i *Webucator 3.0*, overføres til *DPG 2.0* og brukes ved innlogging og autorisering av ressurser er beskrevet i det neste delkapittelet.

6.4.3 Utveksling av brukerinformasjon

Et av problemene i den gamle løsningen med *Webucator 2.0* og *DPG 1.0*, var at en systemansvarlig blant annet måtte flytte filer manuelt i filsystemet for å spesifisere hvilke presentasjoner de ulike brukerne skulle ha tilgang til i *DPG 1.0*, og hvilken rolle de skulle ha i presentasjonene. Dette ble beskrevet i delkapittel 5.3 og vist i Figur 5-2.

Den nye løsningen med *Webucator 3.0* og *DPG 2.0* vil altså kutte hele mellomleddet, der en systemansvarlig må eksportere XML-filer fra Webucator til en lokal PC for så å plassere dem i *DPG*-ens katalogstruktur, ved at denne informasjonen overføres automatisk mellom systemene ved hjelp av *remoting* der *DPG 2.0*'s *Authentication Manager* er plassert i *Webucator 3.0* (Se delkapittel 6.4.2). Vi ser i Figur 6-6 hvordan systemansvarlig har fått færre arbeidsoppgaver, samtidig som vedkommende ikke lenger trenger å ha kunnskap til *DPG*-ens filstruktur.

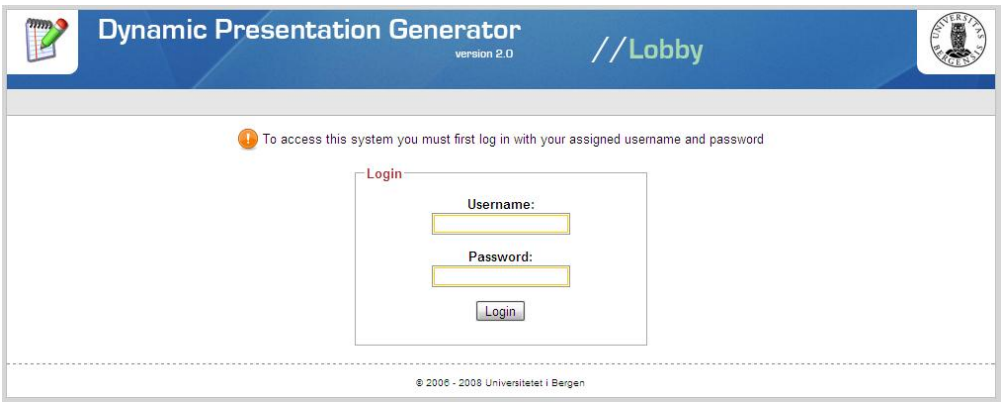


Figur 6-6. Tilordne brukergruppe til en presentasjon i DPG 2.0.

De videre underavsnittene viser hvordan brukerinformasjonen blir utvekslet mellom systemene i forbindelse med innlogging av en bruker.

Før innlogging

En bruker ønsker tilgang til *DPG 2.0*. *ACEGI*'s filtre har ingen autentiseringsinformasjon for vedkommende og *DPG 2.0* viser derfor login-skjemaet (Se Figur 6-7) slik at brukeren kan autentisere seg. Når brukeren sender skjemaet med informasjonen vil *DPG 2.0*'s *Authentication Manager* sjekke dette opp mot informasjonen som er lagret i systemet. Som beskrevet befinner *DPG2.0 Authentication Manager*-en seg i *Webucator 3.0*.



Figur 6-7. Login-skjerm til *DPG 2.0*.

Sjekk i Webucator 3.0

Hvis brukerens brukernavn og tilhørende passord ikke finnes i databasen til *Webucator 3.0*, vil brukeren få beskjed om at brukernavn og passord ikke er godkjent, og mulighet til å prøve på nytt.

Om informasjonen derimot finnes i databasen til *Webucator 3.0* vil det bli opprettet et `Authentication`-objekt som beskrevet i delkapittel 6.4.2. I *userGroups*-feltet til objektet vil det legges inn en liste over alle kursinstansene brukeren er medlem av og hvilken rolle vedkommende har i dem. Om brukeren har rollen *administrator* vil som tidligere nevnt alle kursinstansene legges inn i brukergruppelisten. `Authentication`-objektet plasseres nå i *DPG 2.0*'s *HTTP session*, og kan derfra nås hver gang brukeren ønsker tilgang til en beskyttet ressurs i systemet. På den måten unngås det, at for hver gang brukeren ønsker tilgang til en beskyttet ressurs, må *DPG 2.0* undersøke via *remoting* mot *Webucator 3.0*, om brukeren har de nødvendige rettighetene til dette.

6.5 Kort om navngiving

Det kan kanskje virke forvirrende at det brukes ulike termer på for eksempel grupper av brukere gjennom systemene. For eksempel så vi i delkapittel 3.3 hvordan man knyttet opp en presentasjon i *DPG 2.0* mot en kursinstans i *Webucator 3.0*. Vi så da at listen over tilgjengelige kursinstanser var navngitt som "User Groups", et begrep vi fant igjen i forbindelse med `Authentication`-objektet i delkapittel 6.4.2. Dette skyldes at *DPG 2.0* er ment å være et generisk system, i den forstand at det skal kunne presentere helt andre ting enn presentasjoner for kursinstanser i fjernundervisningssammenheng. Bjørn Ove Ingvaldsen har blant annet demonstrert dette i sin masteroppgave [6], der det vises til alt fra presentasjoner av fotballklubber til aviser. *DPG 2.0* er altså løst koblet mot *Webucator 3.0* slik at det er mulig å bytte ut *Webucator* med en annen brukerhåndtering om man ønsker det.

For å oppsummere så vil *"User Group"* når *DPG 2.0* brukes i fjernundervisningssammenheng, med *Webucator 3.0* som brukerhåndtering, samsvare med de kursinstansene som finnes i *Webucator 3.0*.

7 Evaluering og videre arbeid

7.1 Måloppnåelse

Med bakgrunn i oversikten gitt i delkapittel 1.3, er alle målene for denne oppgaven oppnådd. *Webucator 3.0* er implementert som en selvstendig webapplikasjon, og *DPG 2.0* er fritatt for alle oppgaver rundt brukerhåndtering. Sikkerhetshullene som ble avdekket i DPG 1.0 er tettet, og det er implementert en ny, automatisert løsning for utveksling av informasjon mellom *DPG 2.0* og *Webucator 3.0*.

All Java-kode som er levert er kommentert ved hjelp av *Javadoc* [50], noe som er ment å bidra til å lette en eventuell videreutvikling av systemet, ved at det er nøye beskrevet hva de ulike klassene og metodene i systemet utfører.

Rammeverket *Spring Framework* er brukt uten unntak gjennom hele applikasjonen, og er ment å bidra ytterligere til bedre forståelse av systemet. Beskrivelsen av lagene i rammeverket i denne oppgaven og i annen systemdokumentasjon gir også nyttig informasjon for vedlikehold og videre utvikling.

Webucator 3.0 er gjennom utviklingsperioden blitt deployet på en web-server og testet av ulike brukere. Tilbakemeldinger fra dem har gjennom hele utviklingen bidratt til at feil har blitt rettet, og at endringer er blitt gjort for å møte brukernes krav og forventninger. På den annen side har det i denne omgang ikke vært tid til å få implementert tilstrekkelig med automatiserte tester av systemet, noe som er beskrevet nærmere i delkapittel 7.4.1.

7.2 Teknologiske utfordringer

Bruken av de ulike teknologiene og rammeverkene i forbindelse med denne oppgaven har fungert relativt smertefritt. Den eneste teknologien som bød på større tekniske utfordringer var sikkerhetsrammeverket *ACEGI (Spring Security)*. I tillegg til at dette er relativt omfattende og komplekst, var hovedproblemet manglende dokumentasjon rundt hvordan systemet fungerer. Dette skyldes nok i hovedsak *ACEGI* 's unge alder, og er allerede bedret etter at ACEGI ble innlemmet i "*Spring*-familien" våren 2008.

En annen teknologi som bød på en del mindre problemer var *Ant*, som ble brukt for å bygge applikasjonene. *Ant* tilbød ingen god mekanisme for å holde styr på alle de ulike bibliotekene applikasjonene trengte tilgang til for å fungere. Det ble derfor etter hvert ganske vanskelig å holde oversikt over hvilke biblioteker som var i bruk til enhver tid. Et alternativt byggesystem til

videre utvikling som det bør sees nærmere på vil derfor være *Maven*[51] (Se delkapittel 7.4.2).

7.3 Utviklingsmiljø

All utvikling av *Webucator 3.0* ble gjort i *Eclipse*[52]. Dette open-source-prosjektet har de siste årene nærmest utviklet seg til en industristandard, og tilbyr et utall ulike plugins som utvider basisfunksjonaliteten ytterligere. Det kan ofte være vanskelig å finne fram i jungelen av plugins og avhengighetene mellom disse.

EasyEclipse[53] tilbyr en løsning på dette problemet. Det er ferdige distribusjoner som inneholder både selve *Eclipse*-applikasjonen og mange ulike plugins, som dekker de fleste behov. Distribusjonen *EasyEclipse Server Java 1.2.2.2*, var tilstrekkelig for all utviklingen av *Webucator 3.0*. Den inneholdt blant annet nyttige plugins for *Apache Tomcat*, *Spring Framework*, *Ant* og flere av de andre teknologiene som ble brukt. Det ble gjennom arbeidet med denne oppgaven ikke oppdaget noen mangler ved utviklingsmiljøet som påvirket det endelige resultatet.

7.4 Videre arbeid

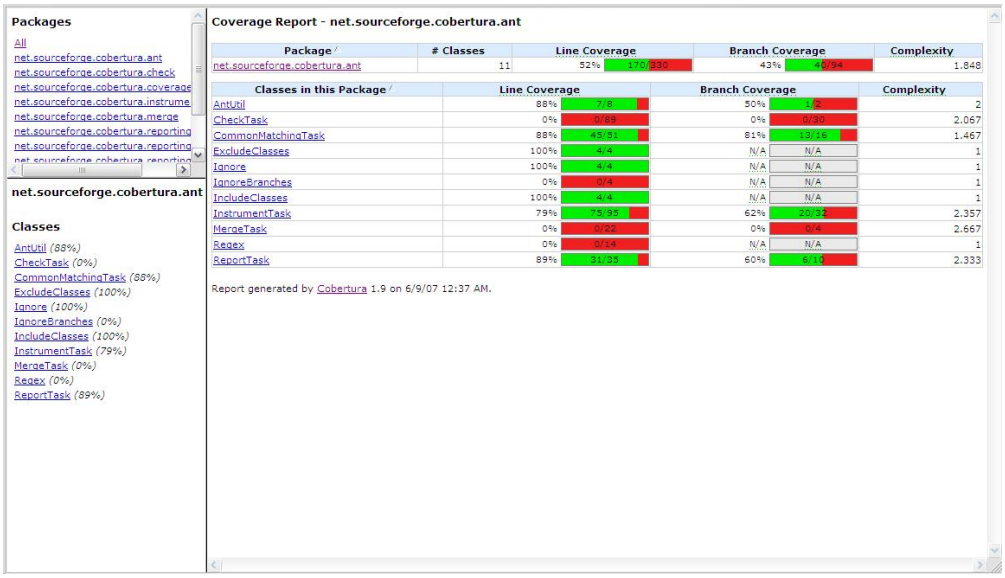
Dette delkapittelet gir forslag til hva det kan jobbes videre med i tilknytning til *Webucator 3.0*.

7.4.1 Testing

Før *Webucator 3.0* eventuelt videreutvikles bør det implementeres tester av funksjonaliteten til systemet, det være seg Unit-tester, integrasjonstester og så videre. Disse testene vil så kunne brukes for å sikre at eksisterende funksjonalitet forblir operativ, når ny funksjonalitet legges til.

For å lette dette arbeidet, er *Cobertura*[54] integrert i applikasjonen. Dette verktøyet overvåker programkoden, og gir beskjed om hvilke deler av den som er testet. Figur 7-1 viser et eksempel på en *Cobertura*-rapport. Man kan for eksempel klikke på linker i rapporten for å få mer informasjon om hvilke kodelinjer i en klasse som er testet.

Før arbeidet med å skrive tester starter, bør det naturligvis sees på om det finnes andre verktøy til å støtte dette arbeidet, som er bedre egnet enn *Cobertura*.



Figur 7-1. Eksempel på rapport fra *Cobertura* .

7.4.2 Applikasjonsbygging

Før en eventuell videreutvikling anbefales det å bytte applikasjonsbygger, fra *Ant* til *Maven*. Dette vil lette arbeidet med å holde rede på applikasjonens biblioteker, siden *Maven* bruker ulike *repositories* som automatisk henter og oppdaterer de biblioteker som trengs for å kjøre applikasjonen. Informasjon om *Maven*, gode tutorials og annen dokumentasjon finnes på http://maven.apache.org/users/index.html.

7.4.3 Persistens

Som beskrevet i delkapittel 4.2.2 er persistenslaget i *Webucator 3.0* implementert ved hjelp av *JDBC-DAO*er. Dette innebærer at SQL-spørringer er skrevet direkte inn i Java-koden. Grunnen til at denne løsningen ble valgt var i hovedsak at det ikke var tid nok til å sette seg inn i andre løsninger.

Ulempen med løsningen som nå er implementert er at den kan være sårbar for endring av databasetype eller versjon. Et alternativ kan for eksempel være å ta i bruk et rammeverk i persistenslaget som for eksempel baserer seg på objekt-orientert kommunikasjon mot databasen. Et eksempel på et slikt rammeverk er *Hibernate* [22]. Informasjon om *Hibernate*, tutorials og annen dokumentasjon finnes på http://www.hibernate.org/5.html.

Det bør også sees på persistensløsningen som ble utviklet for *DPG 2.0* i forbindelse med Karianne Bergs masteroppgave, og om det kan være aktuelt å bruke den også i *Webucator 3.0.*

8 Bibliografi

[1] E. Haagensen, K. A. Mughal, and S. Nysæther, "Bergen Webucator: Web-based tools for distributed learning," Universitetet i Bergen, 2000.

[2] E. Haagensen, "Verktøy for internettbasert undervisning," Universitetet i Bergen, 1999.

[3] P. Solheim, "Webucator 2.0: A Course Administration System," Universitetet i Bergen, 2006.

[4] GNU General Public Licence. (2008, Apr.) The GNU Operating System. [Online]. http://www.gnu.org/

[5] B. C. Sebak, "Dynamic Presentation Generator 2.0," Masteroppgave, Institutt for informatikk, Universitetet i Bergen, Bergen, 2008.

[6] B. O. Ingvaldsen, "Multimedia i Dynamic Presentation Manager 2.0," Masteroppgave, Universitetet i Bergen, Bergen, 2008.

[7] K. Berg, "Persistensproblematikk i Dynamic Presentation Generator," Masteroppgave, Universitetet i Bergen, Bergen, 2008.

[8] Y. Espelid, "Dynamic Presentation Generator," 2004.

[9] K. Cruickshanks, "Verktøy for generering av XML-baserte presentasjonar: JPGen - Java presentasjons generator," Masteroppgave, 2004.

[10] C. M. Berg, "Statusrapport for presentasjonsmønstermotor (PMM)," 2004.

[11] M. Fowler, *Patterns of Enterprise Application Architecture*. Pearson Education, Inc, 2003.

[12] SpringSource. (2008, May) Spring Framework. [Online]. http://springframework.org/

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[14] B. C. Sebak and K. Berg, "Innleveringssystemet," Universitetet i Bergen Oppgave i INF219, 2007.

[15] Apache Software Foundation. (1999-2006) Apache Tomcat 6.0 - Realm Configuration HOW-TO. [Online]. http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html

[16] Sun Microsystems. (2008, May) Sun Java. [Online]. http://java.sun.com/

[17] Apache Software Foundation. (2008, Jun.) Apache Maven Project. [Online]. http://maven.apache.org/

[18] Microsoft Corporation. (2008, May) Microsoft Internet Explorer. [Online]. http://www.microsoft.com/norge/windows/ie/default.mspx

[19] Refnes Data. (2008, Jun.) W3Schools - Browser stats. [Online]. http://www.w3schools.com/browsers/browsers_stats.asp

[20 Wikipedia. (2008, Jun.) Wikipedia - Model-view-controller. [Online].

] http://en.wikipedia.org/wiki/Model_view_controller

[21] Wikipedia. (2008, Jun.) Wikipedia - Plain Old Java Object. [Online]. http://en.wikipedia.org/wiki/Plain_Old_Java_Object

[22] Hibernate. (2008, Jun.) Hibernate. [Online]. http://www.hibernate.org/

[23] Hibernate. (2008, Jun.) Supported Databases. [Online]. http://www.hibernate.org/80.html

[24] Miscellaneous: TO DO.

[25] Spring Source. (2008, Jul.) Spring Remoting and Web Services. [Online]. http://static.springframework.org/spring/docs/2.0.x/reference/remoting.html

[26] The PostgreSQL Global Development Group. (2008, May) PostgreSQL. [Online]. http://www.postgresql.org/

[27] The Apache Software Foundation. (2008, May) Apache Tomcat. [Online]. http://tomcat.apache.org/

[28] The Apache Software Foundation. (2008, May) Apache Ant. [Online]. http://ant.apache.org/

[29] Mozilla Foundation. (2008, May) Mozilla Firefox. [Online]. http://www.mozilla-europe.org/no/products/firefox/

[30] Opera Software. (2008, Jul.) Opera. [Online]. http://www.opera.no

[31] Hardware.no. (2008, Jul.) Hardware.no. [Online]. http://www.hardware.no/artikler/firefox_oker_i_europa/51454

[32] M. Fowler, *Patterns of Enterprise Application Architecture*. Pearson Education, Inc, 2003.

[33] S. Ladd, D. Davison, S. Devijver, and C. Yates, *Expert Spring MVC and Web Flow*. Apress, 2006.

[34] Sun Microsystems. (2008, May) JDBC. [Online]. http://java.sun.com/javase/technologies/database/

[35] The SCO Group. (2008, Jun.) What makes a good user interface?. [Online]. http://ou800doc.caldera.com/en/SDK_vtcl/vtclgN.style_goodui.html

[36] W3. (2008, Jun.) Cascading Style Sheets. [Online]. http://www.w3.org/Style/CSS/

[37] S. R. Dude. (2008, Jul.) Sanscons. [Online]. http://www.somerandomdude.net/srd-projects/clearbits/

[38] ACEGI. (2008, Jul.) ACEGI Security. [Online]. http://www.acegisecurity.org/

[39] Apache Tomcat Foundation. (2008, Jul.) Working with JDBC Realm. [Online]. http://tomcat.apache.org/tomcat-3.3-doc/JDBCRealm-howto.html

[40] K. Berg and K. S. Løvik, "Dynamic Presentation Generator - Del III," Universitetet i Bergen Semesteroppgave INF226, 2006.

[41] Wikipedia. (2008, Jul.) Content Managment System. [Online]. http://en.wikipedia.org/wiki/Content_management_system

[42] Sun Microsystems. (2008, Jul.) Java Naming and Directory Interface (JNDI). [Online]. http://java.sun.com/products/jndi/

[43] S. H. Huseby, *Innocent Code*. Chichester, England: John Wiley & Sons, 2004.

[44] SecurityFilter. (2008, Jul.) SecurityFilter Project. [Online]. http://securityfilter.sourceforge.net/

[45] Sun Microsystems. (2008, Jul.) Java Interface. [Online]. http://java.sun.com/docs/books/tutorial/java/concepts/interface.html

[46] C. Walls, *Spring in Action*, 2nd ed.. Manning Publications Co., 2008.

[47] Sun Microsystems. (2008, Jul.) Servlet Filters. [Online]. http://java.sun.com/products/servlet/Filters.html

[48] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns*. John Wiley & Sons Ltd., 2006.

[49] Sun Microsystems. (2008, Aug.) Javadoc. [Online]. http://java.sun.com/j2se/javadoc/

[50] Apache Foundation. (2008, Aug.) Maven. [Online]. http://maven.apache.org/

[51] Eclipse. (2008, Aug.) Eclipse.org. [Online]. http://www.eclipse.org/

[52] Easyeclipse. (2008, Aug.) Easyeclipse. [Online]. http://www.easyeclipse.org/site/home/

[53] Cobertura. (2008, Aug.) Cobertura. [Online]. http://cobertura.sourceforge.net/