BACHELOROPPGAVE:

ALUMNI COMMUNICATION SYSTEM

FORFATTER(E):

Lasse Krøger Vanebo
Oddbjørn Undseth Bakke
Petter Andersen Busterud

Dato:

Gjøvik 20.05.20

SAMMENDRAG AV BACHELOROPPGAVEN

Tittel:	Alumni Communication System	Dato : 20.05.10
Deltaker(e):	Lasse Krøger Vanebo	
	Oddbjørn Unseth Bakke	
	Petter Andersen Busterud	
Veileder(e):	Harald Liodden (Prosjekt)	
	Simon McCallum (Teknisk)	
Oppdragsgiver:	Høgskolen i Gjøvik	
Kontaktperson:	Hilde Bakke	
Stikkord (4 stk)	Social Networking, CakePHP og Database	

Antall sider: 98	Antall bilag: 6	Tilgjengelighet (åpen/konfidensiell): kildekode = konfidensiell, ellers åpent.

Kort beskrivelse av bacheloroppgaven:

Alumni er latinsk og betyr ”tidligere studenter ved en skole”. Et Alumnisystem er en plattform for å opprettholde kontakten mellom skolen og deres tidligere studenter. Dette har vært mer populært blant engelsktalende land, hvor universitet mottar mindre offentlige midler og blir dermed avhengig av å motta gaver/donasjoner fra private sektorer.

Vårt Alumni Communication System er utviklet for Høgskolen i Gjøvik som lenge har ønsket en slik løsning. Noe som flere ganger har blitt tatt opp i styret, men tid til utvikling og kostnader har gjort at dette aldri har blitt realisert.

Webfrontenden for vårt system er utviklet i PHP under CakePHP rammeverket. Databasen er konstruert og implementert ved bruk av MySQL og MySQL Workbench.

Innhold

1 Introduksjon

Alumni er latin og betyr tidligere student ved en skole. Mange skoler utenfor Norden har hatt lang tradisjon for å holde tett kontakt med sine alumner og danner alumniorganisasjoner, spesielt i USA og Storbritannia er dette veldig utbredt.

Alumniorganisasjonene utgjør store ressurser for skolene og er kritisk for finansiering av skoler som ikke kan basere seg på statsstøtte. Skolene kan vise til hva deres alumni jobber med og trekker ofte frem alumni med spesielt stor suksess når de reklamerer for seg selv. Alumniorganisasjonene jobber gjerne for å hjelpe skolen med foredragsholdere, donasjoner, pengeinnsamlinger og rekruttering. Mange alumni fungerer og som mentorer for nyutdannete studenter i arbeidslivet, alumniorganisasjonen tilbyr dermed studenter ved skolen et stort kontaktnettverk når de kommer ut i arbeidslivet.

Nordiske skoler har tradisjonelt vært statsfinansierte og har derfor ikke hatt økonomisk behov for alumniorganisasjoner. De Nordiske skolene ser nå behovet for alumniorganisasjoner på grunn av fordelene de gir med tanke på markedsføring, tilgang til foredragsholdere og nettverk i arbeidslivet som deres studenter kan dra nytte av. Med dagens teknologi er den naturlige og mest effektive måten å organisere alumni på med et social-networking-system. Slike alumnisystem er i bruk i dag hos mange skoler, men det er behov for å lage egne system for Norge der fokuset ikke er på finansiering, men på den sosiale verdien et alumninettverk gir i seg selv for elever og skole.

1.1 *Målgrupper*

1.1.1 For prosjektet

Målgruppen for prosjekter er først og fremst Høyskolen i Gjøvik, men også et eventuelt Innlands Universitet og andre utdanningsinstitusjoner

1.1.2 For rapporten

Målgruppe for denne prosjektrapporten prosjektets veileder og sensor, studenter, lærere og ledelse ved Høgskolen i Gjøvik.

1.2 *Prosjektmål*

Gruppens mål i dette prosjektet var å utvikle en prototype/beta løsning til et alumnisystem for Høgskolen i Gjøvik. Målet var å få opp alle kritiske elementene til et alumnisystem: databasen, webfrontenden for administratorer/alumnier med et innloggingssystem. Dette inkluderer registrering av nye brukere, oppdatering av profil og hente ut informasjon om andre brukere. Nøye dokumentasjon var en viktig del av prosjektet, for å sikre mulighetene for videreutvikling.

Systemet skal være «grunnmuren» til et norskutviklet alumnisystem, som kan bygges videre på å lanseres som et ferdig produkt til Høgskolen i Gjøvik (HiG). Effekten som ønskes å oppnås i dette prosjektet, er at HiG skal få et system som vil hjelpe de ansatte å opprettholde kontakten med tidligere studenter. Noe som vil hjelpe til å effektivisere

kontakten med arbeidsplasser samt gjøre det lettere å skaffe mentorer/foredragsholdere til skolen. Skape et slikt nettverk er viktig både for skolen og studentene, både under og etter studietiden.

1.3 *Prosjektorganisering*

1.3.1 Ansvarsforhold og roller

Lasse K. Vanebo var prosjektets leder og har hatt ansvar for den ukentlige tidsdisponeringen. Har hatt hovedansvar for kravspesifikasjon og kundekontakt.

Petter A. Busterud har ansvar for å holde orden på rapport og dokumentasjon. Han fungerte som gruppens sekretær, og skal førte ukentlig logg for gruppen. Han hadde hovedansvar for databasen.

Oddbjørn U. Bakke har hatt ansvar for websiden til gruppen, konfigurasjonsstyring, samt gruppens programvare og utstyr. Han hadde hovedansvar for webfrontend.

1.3.2 Arbeidsgiver

Høyskolen i Gjøvik er en offentlig norsk høyskole med i overkant av 2400 studenter og 270 ansatte. Den er lokalisert på Kallerud i Gjøvik. Høgskolen tilbyr studier blant annet innen informatikk, medieteknikk, informasjonssikkerhet, elektro-, byggingeniør og sykepleierstudier. HiG er spesielt kjent for sitt tunge fagmiljø innen informasjonssikkerhet.

Høgskolen i Gjøvik er også en høgskole med et internasjonalt miljø. Dette gjelder særlig for masterstudiene høgskolen tilbyr.

1.4 *Prosjektgruppens bakgrunn*

Prosjektgruppas medlemmer går alle ingeniørfag data studier og har sådan et bredt grunnlag innen informatikk, programmering og utvikling uten spesiell spesialisering. Et medlem på grunne våres hadde en del erfaring med PHP fra før. To av gruppens medlemmer hadde erfaring med Apache webserver fra fagskole. Alle hadde erfaring innen databaser og systemutvikling.

Under prosjektet har satt oss omfattende inn i CakePHP rammeverket og lært oss mer om PHP. Vi har også satt oss inn bruken av jQuery som JavaScript rammeverk, og ACL til autentisering.

1.5 *Utviklingsmodell*

Prosjektet vårt vil bestå av tre hovedelementer; Database, Webfrontend og Facebook applikasjon. Hvor utviklingen av grunnsystemet vil foregå i denne rekkefølgen på grunn av disse modullenes funksjonelle krav til hverandre. Vi har sett for oss at vi ønsker å jobbe med prosjektet i faser, hvor vi vil først definere mål som skal utføres gjennom en periode, disse utføres i samsvar med valgt utviklingsmodell før vi går over på nye mål.

I møter med oppdragsgiver og veileder er det også kommet fram at det er ønsket at vi kan utgi prototyper av systemet med jevne mellomrom, slik at de kan komme med tilbakemelding om eventuelle tilleggfunksjoner, endring av enkelte løsninger eller andre generelle ønsker de kunne tenke seg i systemet. Dette gjør at vi må ha en fleksibel utviklingsmodell, slik at vi kan være mer åpne for endringer underveis. En slik løsing vil også være til hjelp for oss som prosjektgruppe, ved at vi her får konkrete mål som må utføres innenfor en tidsramme.

Vi så for oss SCRUM som en passende kandidat som utviklingsmodell, men de ”strenge” kravene til rollefordeling og hvor lange periodene hver sprint skulle være på gjorde at den ikke passet inn i vår prosjektgruppe på 3 med en tidsramme på 3-4 måneder.

Vi så og litt på XP, men kom fort fram til at den ikke passet ved våres gruppestørrelse og ikke hjalp oss i noen særlig grad med å dele prosjektet inn i fornuftige faser.

Fossefall og lignende modeller har vi og sett bort fra grunnet at de ikke tilrettelegger for å dele prosjektet inn i faser og at de ikke gir noen god måte å håndtere endringer på.

Valget falt til slutt på spiralmodellen. Denne løsningen gir oss det vi ønsker i en systemutviklings modell. Hvor vi starter med å fastsette mål ved hver nye fase vi går inn i, og ender med at oppdragsgiveren kan gi sin vurdering av prosjektet ved hver fase slutt. Det vil også bli gjort korte risikoanalyser gjennom fasene noe som gjør at vi får redusert risikoer som eventuelt kan dukke opp i hver enkelt fase.

1.6 *Organisering av rapporten*

Rapporten er inndelt i følgende 7 kapitler. Kildekoden til prosjektet vil ikke ligge som vedlegg i rapporten som blir offentlig tilgjenglig via biblioteket ved Høgskolen i Gjøvik, men denne vil bli frigjort til personer som ønsker å fortsette utvikling av prosjektet.

1. **Introduksjon**
   Introduksjon til prosjektet, med prosjektets mål og introduksjon av prosjektgruppe og arbeidsgiver.
2. **Kravspesifikasjon**
   Detaljering og modellering av krav som systemet skulle oppfylle.
3. **Design**
   Dette kapitelet inneholder designet for systemet vi lagde og forklaringer bak dette designet.
4. **Programvare**
   Vi har skilt ut en egen del som gir en introduksjon i programvaren vi brukte i dette prosjektet fordi den ble en veldig stor del av prosjektet og forståelse av den er påkrevd for å forstå implementeringsdelen.
5. **Implantasjon**
   Her beskriver vi implementering av designet, hvilke problemer vi støtte på og generelt hvordan programmet utviklet seg etter hvert som vi implementerte det.
6. **Testing**
   Dette kapitelet skildrer hvordan systemet ble testet og hva som må gjøres av fremtidige tester.
7. **Presentasjon av prosjektet for Høgskolestyret ved HIG**
   Omhandler en presentasjon vi hadde av prosjektet for Høgskolestyret ved HIG
8. **Avslutning**
   Her drøfter vi oppgaven og løsningen vår av den. Vi beskriver hva som må gjøres av videre utvikling og hvordan dette kan skje. Til slutt kommer vi med en konklusjon rundt prosjektet og hva vi har lært.

• Bibliografi - Kilder og referanser.
• Vedlegg

2 Kravspesifikasjon

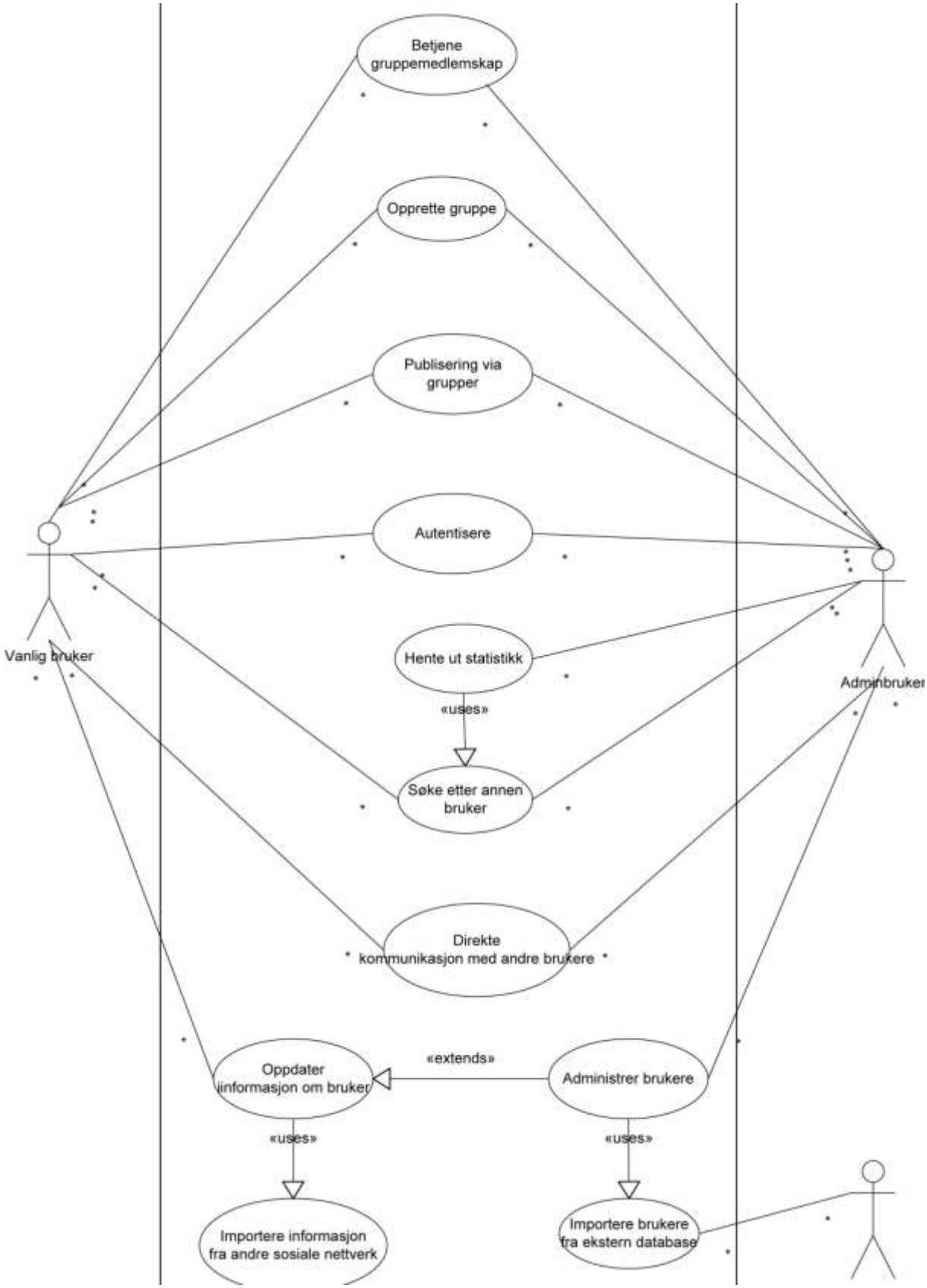
2.1 *Systemets brukere*

Systemets brukere er delt i to hovedgrupper. Den ene er administratorer som jobber for skolen. Disse bruker systemet som et verktøy for å hente ut informasjon om andre brukere for skolen og sende ut informasjon fra skolen, i tillegg vil de hjelpe andre brukere med deres behov og vedlikeholde brukerdatabasen.

Den andre hovedgruppen er vanlige brukere. Her har vi tatt et desingvalg som divergerer fra den klassiske definisjonen av alumnisystem, ved at vi i tillegg til alumner også gir nåværende studenter tilgang. Disse brukerne registrerer sin informasjon, leser informasjon fra skolen og hverandre. Alumner og studenter er begge vanlige brukere og har samme use case, studentene krever derfor ingen ekstra funksjonalitet, men studenter er skilt ut i persona modelleringen fordi de vil ha andre motiver enn alumni for sin bruk.

2.2 *Funksjonelle krav*

2.2.1 Use Case Diagram



Figur 1

2.2.2 Overordnede use case beskrivelser

Use case	Betjene gruppemedlemskap
Aktør	Vanlig bruker eller administrator
Mål	Brukeren kan endre hvilke grupper han/hun er med i
Beskrivelse	Brukere melder seg inn i de grupper de har lyst på informasjon fra. Brukere står fritt til å endre på alle gruppemedlemskap med unntak av hovedgruppe, avdeling og klasse. Administratorer er de eneste med myndighet til å endre medlemskap i disse gruppene. Alle endringer i gruppemedlemskap oppdateres med databasen og deles med venner av brukeren.

Use case	Opprette gruppe
Aktør	Vanlig bruker eller administrator
Mål	En bruker eller en administrator oppretter en gruppe og setter attributter for denne
Beskrivelse	Brukere kan opprette private grupper der andre kan melde seg inn for å dele informasjon. Administratorer kan i tillegg til dette opprette offisielle grupper og melde brukere inn i dem.

Use case	Publisering via grupper
Aktør	Vanlig bruker eller administrator
Mål	Skolen og brukere sender ut informasjon til en gruppe brukere
Beskrivelse	Gruppemedlemskap brukes her som nyhetsabonnementer som får informasjon fra gruppene på sin egen startside. Gruppene har satte regler for hvem som får publisere informasjon gjennom dem

Use case	Hente ut statistikk
Aktør	Administrator
Mål	Skolen henter ut informasjon om brukerne og bruken av systemet
Beskrivelse	Administratoren kan hente ut spesiell informasjon om alle brukerne innenfor en gruppe eller som har visse felles karakteristikk ved å bruke logikken i søkefunksjonen. Administratoren kan også få informasjon om systemet og brukernes bruk av det.

Use case	Søke etter annen bruker
Aktør	Bruker, administrator, <Hente ut statistikk>
Mål	Finne en bruker ut i fra forskjellige typer data
Beskrivelse	Brukere skal kunne være søkbar ut fra blant annet fornavn, etternavn, tidligere navn, klasse, avdeling, adresser, telefonnummer, årskull. Søkene skal kunne begrenses av flere typer data samtidig.

Use case	Direkte kommunikasjon med andre brukere
Aktør	Bruker eller administrator
Mål	Brukere og administratorer sender beskjeder til hverandre enten manuelt eller automatisk.
Beskrivelse	Når kontaktinformasjonen om en bruker oppdateres kan dette automatisk sende melding til brukerens venner. Brukere kan også sende hverandre meldinger.

Use case	Importere info fra andre sosiale nettverk
Aktører	<Oppdater informasjon om bruker>
Mål	Automatisk importering av informasjon fra andre sosiale nettverk, i første omgang Facebook.
Beskrivelse	Brukeren kan velge å få sin informasjon importert fra Facebook og få den automatisk oppdatert derifra.

Use case	Oppdaterer informasjon om bruker
Aktør	Bruker
Mål	Brukeren oppdaterer sin personlige informasjon
Beskrivelse	Brukerne administrerer sin egen kontaktinformasjon, jobbinformasjon og lignende, bortsett fra noe informasjon som kun kan endres av administrator. Oppdateringer her blir oppdatert i serveren og genererer automatiske meldinger til venner.

Use case	Administrere brukere
Aktør	Administrator
Mål	Administrator oppretter brukere, samt setter og endrer grunninformasjon om brukere
Beskrivelse	Kun administratorer kan opprette brukere og gjør dette manuelt eller ved å importere fra ekstern database. Det er og kun administratorer som kan endre basisinformasjon om brukerne som fornavn, etternavn, foreldre og lignende.

Use case	Importere brukere fra ekstern database
Aktør	<Administrer brukere>
Mål	Automatisk overføring av kontaktinformasjon om brukere fra allerede eksisterende databaser.
Beskrivelse	Den eksterne databasen dumper informasjonen i et passende format. Informasjonen importeres så automatisk til systemet og nye brukere opprettes. Hvis det er hull i informasjonen som må være på plass for å skape en bruker, må dette legges inn manuelt av administrator.

2.2.3 Persona

Persona er en metode for kravspesifikasjon som fokuserer veldig på brukeren. Metoden går ut på at du lager ”profiler” for personer som representerer typiske brukere av systemet. Denne profilen beskriver personen ganske detaljert. Målet et her å lage flere ”personer” som er så godt beskrevet og så realistisk at programmereren kan sette seg inn i denne personens situasjon og behov. Det er et poeng å lage disse personaene slik at de sammen representerer de ulike personlighetene man har i brukermassen.
Personaene brukes som verktøy. Programmereren lever seg inn i personaen sin situasjon og tenker på hvilke behov denne personen har. Dette er nyttig for å sjekke ette svakheter i systemet ved å leve seg inn i hvilke problemer personaen vil støte på og for å få et bedre bilde av hvilke funksjoner som er viktige.

2.2.3.1 Primær, administrator: Torild Bakke

Navn: Torild Bakke
Alder: 45
Nasjonalitet: Norsk
Sivilstatus: Gift
Bosted: Gjøvik
Jobb: Alumniadministrator og administrasjonssekretær.
Hobbyer: Langrenn, kafébesøk og lesing

Sitat: «*Jeg vil gjerne ha et verktøy som gjør at jeg lettere kan gjøre jobben med å sende ut e-post og post til alumnene, samt holde orden på deres adresser. Går det an å lage?*»

Bakgrunn:
Torild er en ekte urinnvåner fra Gjøvik der hun har bodd hele livet. Hun gikk ut av videregående når hun var 18 år og søkte seg så jobb som sekretær. Hun fikk de første årene kun vikarjobber i forskjellige bedrifter rundt om i Gjøvik. Dette ga henne erfaring slik at hun fikk fast jobb som administrasjonssekretær ved Gjøvik Videregående(GVGS) når hun var 22 år. Det var og her hun møtte sin mann Kjell, som var lærer ved skolen. Hun jobbet ved GVGS til hun var 30 og har siden jobbet som administrasjonssekretær ved HiG.

Kjell og Torild har 3 barn sammen, den eldste er flyttet ut, men de to andre går på videregående. I sin ungdom var hun en aktiv skiløper, men forsømte denne sporten når hun fikk barn. Hun har i de siste årene tatt opp igjen langrenn, sammen med sin mann, for å mosjonere etter at hun hadde en midtlivskrise og ville gjøre noe med helsen sin. Hun har siden hun begynte å jobbe på skoler utviklet en stadig større interesse for lesing og leser mye skjønnlitteratur.

Typisk dag:
På arbeidsdager står hun opp 06.00 for å passe på at ungene kommer seg opp og kjøre dem på skolen før hun drar på jobb. Hun møter alltid opp 07.45, et kvarter før hun egentlig starter fordi hun vil være sikker på å være tidsnok og for å få tid til en kaffe med de andre i administrasjonen. Hun jobber med administrasjonssekretær relaterte oppgaver 3 dager i uka og med alumniadministrasjon 2 dager i uka.. Hun jobber til 16.00, men har en times lunsj fra 11.00 til 12.00 der hun koser seg med matpakke og kaffe sammen med de andre i administrasjonen mens konversasjonen går løst.

Etter jobb drar hun hjem til middag med familien. På kveldene går hun 3 ganger uken på langrenn i lysløypa og koser seg ellers med en god bok sammen med mannen. I helgene liker hun å dra på langrennsturer til DNT-hytter sammen med mannen (hun får ikke med seg ungene) og å gå på kafé sammen med venninner.

Datakunnskaper:
Hun har brukt datamaskin som sitt fremste arbeidsverktøy i over 10 år nå og mestrer vanlige brukeroppgaver bra. Fra jobb er hun vant til å bruke Office programmer, samt tekstbaserte administrasjonsløsninger. Hun bruker alltid å ha litt vanskelig for å lære seg nye systemer, men gjør dette når arbeidet hennes krever det. Privat bruker hun datamaskin til å handle bøker, sjekke hva venninner gjør på Facebook og aksesser et litteraturforum hvor hun diskuterer bøker. Hun har nettopp fått seg en Kindle i gave fra eldste barnet og er fort blitt veldig glad i den.

Holdning til IKT:
Vegrer seg litt for å lære nye ting, men blir veldig glad i IKT-verktøy som gjør jobben og hverdagen hennes lettere.

Mål:
Hun vil ha et program som gjør det lettere for henne å lage mailinglister og holde orden på kontaktdata.

2.2.3.2 Primær, Alumni: Maxime Dubois

Navn: Maxime Dubois
Alder: 26
Nasjonalitet: Belgisk
Sivilstatus: Singel
Bosted: Wallonia, Belgia
Jobb: Prosjektingeniør i Belgacom SA
Hobbyer: Skøyting og dataspill

Bakgrunn:
Maxime er oppvokst i Wallonia regionen i Belgia i en storbyfamilie som enebarn. Barndommen hans var preget av to foreldre som var veldig opptatt med sine egne karrierer og at sønnen deres skulle få seg en god utdannelse. Han jobbet derfor hardt på skolen fra en ung alder og deltok på ekstraundervisning som foreldrene meldte han på.

Faren hadde en forkjærlighet for skøyter og lærte han derfor å gå på skøyter fra en ung alder. Maxime ble veldig glad i skøyting, både på grunn av at han likte sporten, men også fordi det gjorde faren stolt og det var en av de få aktivitetene der han og faren fikk kontakt. Han var ikke en veldig sosial gutt og fant dataspill som en fin måte å slappe av og underholde seg når han ikke jobbet med skole.

Han tok bachelorgrad innen informatikk i Belgia og søkte seg så inn på master i HiG for å oppleve noe nytt og på grunn av mulighetene for skøyting i regionen. På HiG var han en flittig student som holdt seg litt for seg selv, men fikk seg noen venner blant de andre informatikkstudentene og noen spillevenner. Han utviklet en forkjærlighet for Mjøsa når han var her og gikk mange skøyteturer på den.

Etter mastergraden på HiG dro han tilbake til Belgia der han fikk en høyt ansett jobb i Belgacom som prosjektingeniør.

Vanlig dag:
Maximes dag varierer litt etter hvilket prosjekt han jobber på, men som oftest involverer den å jobbe 10 timer hver dag mandag til lørdag. På kveldene slapper han av med spilling og er for tiden veldig aktiv i en MMORPG kalt EVE sammen med noen av spillevennene han møtte på HiG. I helgene er han og står skøyter i den lokale skøytehallen.

I perioder mellom prosjekter bruker han å ta noen fridager som han bruker til å reise rundt i verden. Han har utviklet en smak for kultur og liker å besøke fremmede kulturer. Han planlegger og en tur til Gjøvik for å gå på skøyter på Mjøsa igjen.

Datakunnskaper:
Maxime har en meget kompetent databruker og programmerer med mastergrad i informatikk og bruker datamaskin over 8 timer daglig.

Holdninger til teknologi:
Maxime er en teknokrat og nerd, han elsker teknologi og er totalt avhengig av den.

Mål:
Maxime ønsker å vite hva som skjer faglig på skolen. Han er meget interessert i forskning og utvikling innen IMT. Han kunne tenke seg å besøke skolen når han er i Gjøvik for å gå på skøyter og vil ha en kommunikasjonskanal for å ordne et slikt besøk. Han kunne gjerne tenke seg å holde et foredrag for informatikkstudenter.

Maxime kunne og tenkt seg en mulighet for å lage grupper for skøyter og dataspill sammen med andre HiG Alumni og studenter.

2.2.3.3 Sekundær, administrator Kalle Lindgren

Navn: Kalle Lindgren
Alder: 35
Nasjonalitet: Svensk
Sivilstatus: Singel
Bosted: Hamar
Jobb: Alumniadministrator
Hobbyer: Hockey, Star Trek og Tetris.

Bakgrunn:
Kalle jobbet 7 år ved Midtuniversitet i Sverige som Alumnikoordinator der. Han var med fra starten, med oppsett av database, testing, initial utrullering og brukerregistrering. Han har jobbet mye med å dokumentere bugs i systemet de hadde der i videreutviklingen av systemet, samt at han skrev brukermanualen for administratorer av alumnisystemet de bruker der. Han ble headhuntet til Innlandsuniversitetet for å være alumniadministrator her på grunn av hans erfaring.

Typisk dag:
Kalle jobber fra 08.00 til 16.00 likt som de fleste andre ved Universitetet, men den første timen av dette går gjerne med til prating rundt kaffebordet. Etter 09.00 jobber han med vedlikehold av databasen, oppdatering av innhold sendt ut fra universitetet, hente ut data til administrasjonen på skolen og å behandle e-post fra brukere. Når han er ferdig med dette, jobber han med å forbedre brukermanualen for systemet, samt dokumentere bugs og nye behov. På sikt ved utvidelser av systemet vil han stå for opplæringen av nye alumniadministratorer.

Datakunnskap:
Kalle er en avansert bruker, han har gode generelle kunnskaper om datamaskiner og bruk av disse, men også inngående kunnskap relatert til alumnisystemer. Han har grunnlegende databasekurs, kan bruke kommandolinjer og skrive enkle SQL-spørringer.

Holdninger til teknologi:
Kalle liker datamaskiner og er veldig opptatt av at ting skal være effektivt bygd opp. Han er skeptisk til ting som er bygd med for mye fokus på GUI. Han elsker å ha snarveier for ting og å kunne gjøre ting via færrest mulig trykk. Han savner når datamaskinene var mer tekstbaserte og ikke hadde så tungrodde GUI som nå.

Mål:
Kalle vil ha et alumnisystem som lar han gjøre jobben sin så raskt og effektivt som mulig. Han vil samtidig at det skal være intuitivt og lett forklart slik at hans jobb med å lære andre opp i systemet og lage brukermanual blir enklest mulig.

2.2.3.4 Sekundær, alumni: Are Moan

Navn: Are Moan
Alder: 30
Nasjonalitet: Norsk
Sivilstatus: Gift
Bosted: Oslo
Jobb: Ingeniør ved Oslo Vannverk.
Hobbyer: Fotball, filmer og Metallica.

Sitat: «*Hvorfor skal jeg gidde å engasjere meg i Alumni? Jeg bryr meg ikke om hva som skjer på skolen eller har ikke tenkt å ta noe mer utdanning.»*

Bakgrunn:
Are vokste opp på Hønefoss på 80-tallet. Han har spilt fotball aktivt siden faren hans lærte han det omtrent før han kunne gå. Det ble for det meste løkkefotball, der han var kjent for sine røffe taklinger. Tross iherdig innsats ble han aldri god nok til å spille profesjonelt. Ellers var han med i en veldig sammensveiset gjeng som for det meste så på actionfilmer og hørte på Metallica når de ikke spilte fotball.

Han var aldri glad i skolen og etter videregående var han skole lei og tok seg jobb på det lokale slakteriet. Etter å ha jobbet der et år ble han kalt inn i førstegangstjeneste i 2. bataljon i Bardu. Are trivdes svært godt i forsvaret og dannet seg flere gode, nye venner, som han sammen med var grenader i 2 år, et halvt år av dette i Afghanistan.

Are fant gjennom sin arbeidserfaring ut at han ikke kunne tenke seg å jobbe lavt i et hierarki eller kun jobbe med repetitive oppgaver. Han overvant skoletrettheten og søkte seg inn på byggingeniør ved HiG. Han likte det sosiale på HiG og traff sin kone, Linda, der. Hun gikk informasjonssikkerhet første året der når han gikk siste året. På det faglige meldte skoletrettheten seg fort og han slet seg gjennom med noen stryk og lave karakterer. Han fikk jobb i Oslo Vannverk året han fullførte utdannelsen og har jobbet der siden. Linda flyttet til Oslo sammen med han når hun ble ferdig med sin grad og de bor nå i en leilighet på Grünerløkka og har en sønn, Tord, på 3 år.

Typisk dag:
Linda står alltid opp først ettersom hun starter på jobb tidlig. Are er treg å stå opp og Linda må ofte mase på han og ha kaffen klar til han. Etter å ha spist en kjapp frokost kjører Are Tord i barnehagen før han drar på jobb. Kjøreturen til jobb tar ofte opp i 40 minutt grunnet rushtrafikk, noe som irriterer Are grenseløst.

På jobb sitter Are i en administrativ stilling og koordinerer vedlikehold av vannledninger. Jobben er noe monoton, men han er i stor grad sin egen sjef så han syns det er greit. Han jobber fra 08.30 til 17.00, med en halvtime lunsj imellom som han bruker til å spille Football Manager ettersom han syns praten med kollegene sine er ganske kjedelig.

Etter jobb drar han hjem til middag og middagshvil. 2 av dagene i uka spiller han fotball i den lokale klubben ellers er han hjemme og ser på tv sammen med Linda, men bruker og mye tid

på å leke med Tord. I helgene drar han ofte ut på bar og fotballkamper med kompisene, pizza med Linda og bekjente av dem. Han gleder seg til Tord blir eldre og kan spille fotball.

Datakunnskap
Tord bruker datamaskiner aktivt i sitt arbeid og sin fritid, og bruker dem så aktivt i sin hverdag at han kan vedlikeholde datamaskiner bra.

Holdninger til IKT
Han er ikke interessert i datamaskiner i seg selv og anser slik interesse som nerdete, men han lærer seg gjerne teknologi han trenger og er opptatt av å ha best mulig tv og lydanlegg.

Mål:
Og lettere kunne holde kontakt med kompisene sine fra HiG, samt få vite når det er gjenforeninger og lignende.

2.2.3.5 Tertiær, student: Jan Øverås

Navn: Jan Øverås
Alder: 20
Nasjonalitet: Norsk
Sivilstatus: Singel
Bosted: Gjøvik, Nordbyen
Jobb: Informatikkstudent ved HIG
Hobbyer: Foto, jibbing og tekniske gadgets.

Sitat: *"Har lagt ut ny edit fra helgas session i Hafjell. Mye fete shots tatt med mitt nye HD helmet-cam. Blir ikke nye edits på en stund, har en del innleveringer som har sneket seg inn"*

Bakgrunn:
Jan vokste opp sammen med 4 eldre brødre og to kjærlige foreldre i Fredrikstad. Han var alltid en aktiv gutt, om noe uregjerlig og prøvde seg på et utall hobbyer gjennom årene. Dette har og hatt utslag i at han har vær noe ukonsentrert på skolen og sluntret unna arbeid, men han har alltid hentet seg inn til eksamen gjennom skippertak. Han fant virkelig sitt rette i jibbing (hopping og triksing på twintip-ski) og har siden han var 12 drevet meget aktivt med dette. Han er og veldig interessert i tekniske gadgets og er veldig glad i avanserte telefoner og spillmaskiner. Det siste året har han startet med foto slik at han kan ta bilder av og filme jibbingen sin, dette kombinert med hans gadgetfiksering gjør at han kjøper seg mye dyrt fotoutstyr.

Han søkte seg inn på HiG til dels på grunn av at hans teknofili fikk han til å tenke informatikk var noe for han og til dels for å være nært mange alpinanlegg med gode parker hvor han kan jibbe.

Typisk dag:
Jan går første året informatikk og har en del forelesninger, men det er variabelt hvilke han går på, mest på grunn av han ikke har rutine og struktur i hverdagen. Mange dager bruker han i Hafjell til jibbing, litt ettersom når han har noen å dra med. Han skulle gjerne dratt dit mer, men kjenner ikke så mange enda så har ikke så mange å dra med. De kompisene han har fått seg til nå er de han går i klasse med og de deler hans tekniske interesse, men ikke jibbingen. Han er med i fotogruppa på skolen, men de sliter med rekrutteringen og å komme ut til sine medlemmer og er derfor ikke så aktiv.

Datakunnskaper:
Jan er oppvokst med data og er ganske interessert, men for det meste i underholdningsapplikasjoner og foto- og filmredigeringsprogram.

Holdninger til teknologi
Jan er teknofil og elsker teknologi

Mål
Komme i kontakt med personer som har de samme interessene som han. Han skulle og gjerne hatt et verktøy som hjalp han strukturere hverdagen sin slik at han kom seg på de riktige forlesningene, blir minnet på innleveringer og kan planlegge hverdagen sin.

2.3 *Supplementær spesifikasjon*

2.3.1 Åpen kildekode

Alumni Communication System er utviklet for Høgskolen i Gjøvik og andre læringsinstitusjoner som ikke har midlene til å betale dyre lisenser for programvare. Derfor skal systemet nyttegjøre seg av åpen kildekodeprogramvare og frie lisenser der mulig.

2.3.2 Standarder

Grunnet HiG sine eksisterende systemer har skolen satt krav om at applikasjonen skal kunne kjøres på Apache webserver, bruke MySQL og støtte PHP versjon 5. Skolen hadde krav om at applikasjonen skulle vises likt i de mest brukte nettleserne.

3 Design

3.1 *Systemet*

Når vi startet dette prosjektet var det ikke planlagt at vi skulle få et ferdig produkt, en prototype for å overbevise administrasjonen til Høgskolen i Gjøvik, om at dette systemet var noe å satse videre på.

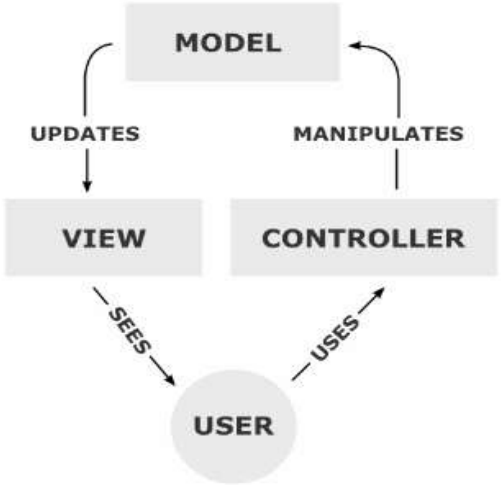
Vi hadde tidlig i prosjektperioden satt opp ukentlige møter med teknisk veileder Simon McCallum. Grunnen til at vi da så på et modulbasert rammeverk, var for og enkelt kunne jobbe mot de moduler Simon så på som kritiske for å få en best mulig prototype, uten å få større opphold ved å måtte kode en del annet i mellom for å få systemet til å fungere slikt det var ønskelig.

Siden prosjektet kun skulle være en prototype, så vi på en modulbasert variant som mye enklere å jobbe videre for eventuelt andre i ettertid. Siden noen muligens skal ta over prosjektet, har vi også måtte bygge systemet opp med en solid grunnstruktur for å gjøre det enklere å kode "byggeklosser", og legge disse på systemet i ettertid.

Vi ønsket også et veldig dynamisk system, både for at man da slipper mer spesialtilpassing om man ønsker å gjøre endringer i serveroppsettet, og fordi vi da kunne bruke de tilbakemeldinger vi fikk til både å reparere og fikse eldre moduler, samtidig som vi utviklet nye moduler for ny funksjonalitet. Denne dynamikken var noe vi enkelt kunne oppnå ved å velge CakePHP, og rammeverkets Modell View Controller baserte struktur.

3.1.1 Model View Controller

MVC er en programvarearkitektur som ble introdusert av nordmannen Trygve Reenskaug. Arkitekturen går ut på at domenelogikken i applikasjonen isoleres fra input og brukergrensesnitt. Dette gjør at moduler for domenelogikk, input og input kan programmeres uavhengig av hverandre.



Modellen [1] viser en enkel MVC kobling.

Figur 2

[1] *Bilde hentet fra http://en.wikipedia.org/wiki/Model–view–controller (mai 2010)*

Modellen i MVC holder orden på informasjon og observerer når denne informasjonen endres. View lager et brukergrensesnitt av model som for interaksjon med brukeren. Controller mottar input og sender forespørsler til modell for å endre informasjon.

Implementering av MVC-arkitekturen vi har brukt fungerer på den måten at selve kodingen skjer i kontrolleren, mens det er selve modellen som kommuniserer med databasen, og som setter opp regler på hvordan informasjonen skal struktureres i databasen. For å gjøre en spørring, må man derfor spørre modellen om informasjonen, også tar modellen seg av resten.

Den informasjonen som hentes ut fra modellen blir så sendt videre til et view, hvor det så genereres en nettside som vises for brukeren ut i fra spørringen.

Fordelen med å gjøre det på denne måten, er at siden man kun koder mot modellen, og ikke mot selve databasen, er det MVC systemet sin jobb å ta seg av alle problemer relatert mellom PHP og det valgte databasesystemet. På denne måten får utvikleren mer frihet til å kode et system, uten å måtte ta mye hensyn til hvilken plattform systemet skal kjøre på.

3.1.2 Systemet slik det er nå

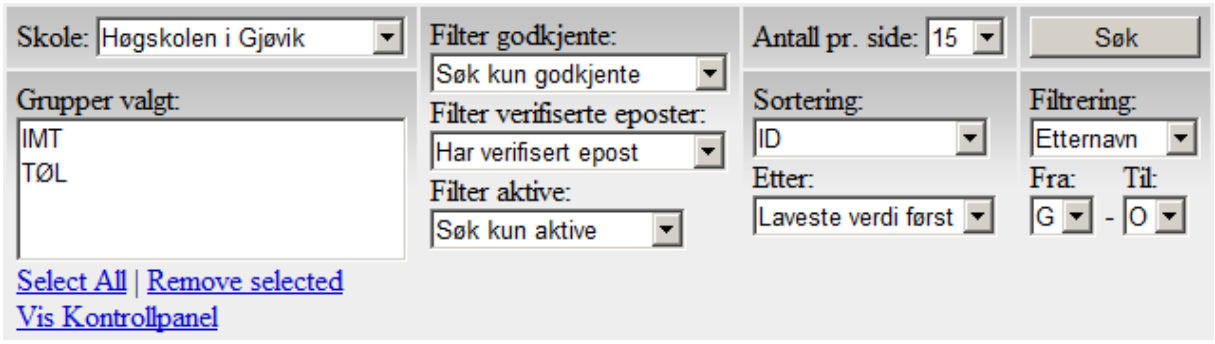
Det vi har nå er et system basert rundt åpne, frie og ikke-lisenskostnadsbelagte rammer(MIT/BSD/LGPL lisenser).

Applikasjonen vi har laget er basert rundt en god og solid grunnstruktur ved å bruke CakePHP rammeverket, samtidig som systemet er enkelt å jobbe videre med og utviklere videre på. Det har også vært viktig at vi fra bunnen av, har lagt opp systemet for å kunne håndtere flere brukere med behov for en god innloggingsprosedyre, samt et godt rettighetssystem for å håndtere riktige rettigheter til brukerne.

Applikasjonen er også bygget for at man enkelt skal kunne hoppe mellom ulike designer om ønskelig, samt at systemet også er lagt opp til at man skal kunne ha støtte flerspråklige brukere.

Vi har i dette systemet også laget flere administrasjonsverktøy for å kunne hente ut lister over brukere, søke etter brukere, godkjenne brukere, godkjenne nye bedrifter, opprette grupper, opprette grupper, skrive artikler, og legge til ekstra funksjoner.

Dersom man lister opp brukere, kan man filtrere ut uønskede brukere, og kun få ut de ønskede brukerne:



Figur 3

Ved søking vil treff som passer med søket automatisk komme fortløpende mens man skriver inn selve teksten i søkefeltet. Når man så ser den brukeren man ønsker å velge, velger man bare denne fra lisen som kom.

Vi har også lagt inn "What you see is what you get" funksjonalitet. Dette vil si at når administratoren skriver en artikkel, vil man få mye av samme funksjonalitet som ved bruk av tekstbehandlere rike på funksjonalitet som Word, Writer, og lignende.

Applikasjonen har også lagt inn funksjonalitet for at administratoren skal kunne gå gjennom grupper, funksjoner, og lignende, ved hjelp av en oversiktelig, pen og ryddig trestruktur.

3.1.3 Funksjonalitet for videre utvikling

Systemet mangler mulighet for å eksportere lister over brukere. Det var tiltenkt eksportering mot Excel og lignende programvare. Grunnen til at vi ønsker å eksportere lister på denne måten er for at administratoren enkelt skal kunne dobbeltsjekke listen, før man importerer denne inn i annen programvare. Ved å gjøre det på denne måten, kan man enkelt sende ut masseutsendelser på e-post, brevpost, lage statistikker eller andre opperasjoner.

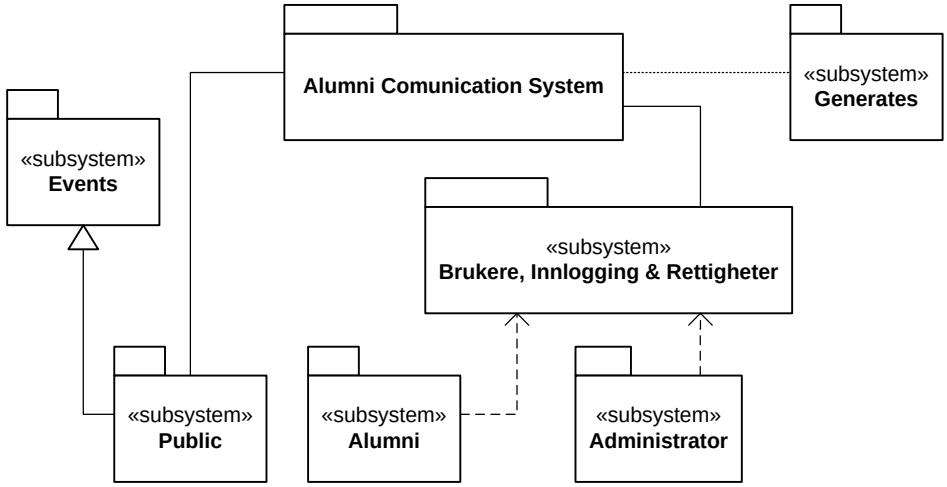
Vi har også sett på mulighetene ved å koble applikasjonen mot Feide, og hente ut relevant studentinformasjon på en god, sikker og lovlig måte. Dette er noe vi kommer tilbake til under designsdelen av rapporten(s26).

Applikasjonen har noe støtte for masseregistrering ved å generere brukere ut av informasjonen fra en annen database. Dette er noe vi hadde ønsket å kunne ha flere rutiner for, som for eksempel importering fra lister fra Excel ark.

Det gjenstår en del på gruppelogikken. Slik applikasjonen er nå kan brukeren kun være medlem av en gruppe. I videre utvikling av systemet må det legges in støtte for at brukere kan være medlemmer av flere grupper. Publiseringssystemet for nyheter mangler funksjoner for å filtrere hvilke nyheter til hvilke brukere etter hvilke grupper de er medlem av. En utvidelse av gruppesystemet som er ønskelig, men som ikke er en nødvendig, er at brukere skal kunne starte egne interessegrupper, hvor de kan melde seg inn og formidle informasjon seg imellom.

Utover dette er det blitt noen skjønnhets- og småfeil som burde rettes opp samt omfatende testing av systemet med flere brukere.

3.1.4 Systemets hoveddeler



Figur 4

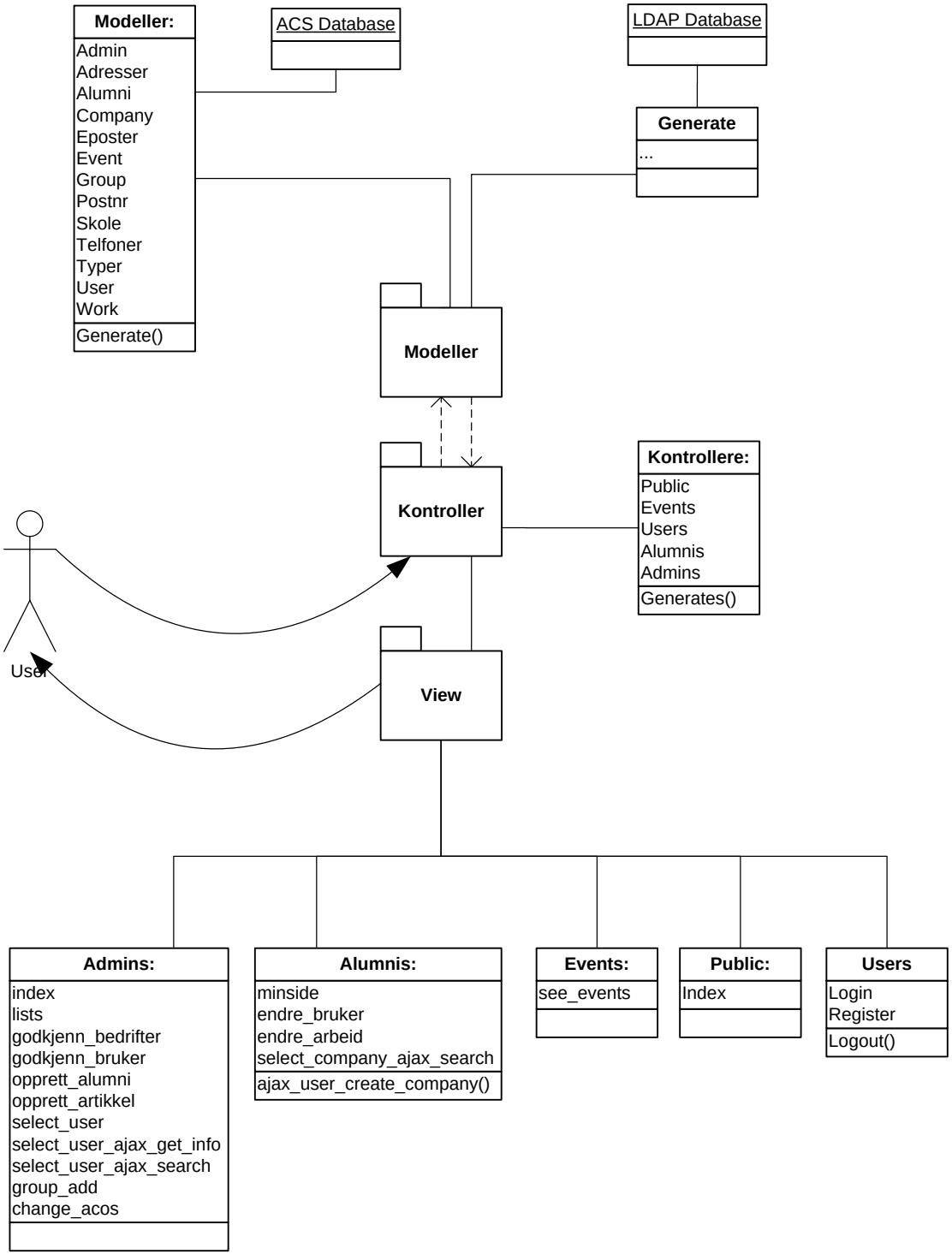
Systemets hoveddeler kan i enkle trekk vises som på figuren over.
Her kan man se at systemet er delt inn i en ”Public” del, der alle kan gå inn å se på innholdet. ”Public” delen er også knyttet mot ”Events” delen. ”Events” delen inneholder artikler lagt inn av administratorene, slik systemet er nå er disse artiklene synlig for alle.

Vi har også en generator del. Denne delen har alle i utgangspunktet tilgang til, men er deaktivert for kjøring med en ”die” funksjon. Dette er gjort for å unngå å måtte sette opp midlertidige rettigheter for å få kjørt dette verktøyet. Generator skriptet blir brukt for å generere testbrukere ut fra en annen database krever en utvikler som koder om skripter for at det skal kunne kjøre.

Vi har også en alumni- og en administratordel som er jobber separat fra hverandre. Begge disse delene av systemet har sine egne verktøy. Disse verktøyene kan kun brukes av de riktige brukerne ved hjelp av brukerinnlogging, tilgang- og rettighetskontroll, noe man kan se noe nærmere etter på neste side.

3.1.5 Kompleks visning av systemet

Figuren under er en mer kompleks visning av systemet, den viser hvor de ulike delene i systemet hører til i MVC modellen.



Figur 5

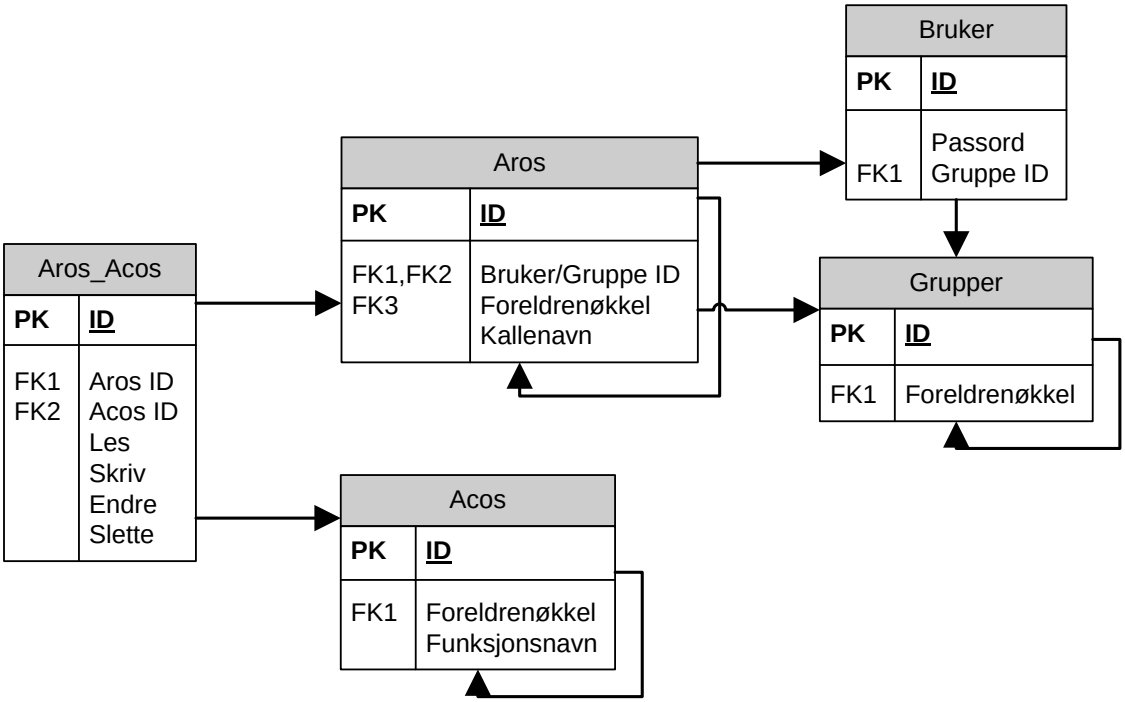
Her vil listen over kontrollere, inneholde selve kontrollerne, mens view listen innholder funksjoner i kontrollerne.

Det er her to spesialtilfeller, ”logout” og ”ajax_user_create_company”. Disse er listet annerledes fordi det ikke finnes noe view knyttet mot funksjonene, og all kode skjer under selve kontrolleren.

Modeller listen inneholder alle modeller i databasen, og derav også alle tabeller i databasen. Det som derimot ikke kan leses ut fra figuren, er innholdet i disse tabellene.

3.1.6 Innlogging og rettigheter

For å forklare hvordan rettighetssystemet er bygget opp, har vi laget en figur som inneholder kun essensen av det viktigste fra databasen.



Figur 6

Rettighetene i dette prosjektet er basert på ”Access control list”, som er metoden bruk i CakePHP for å holde styr på hvem som skal ha tilgang til hva i systemet.

ACL i CakePHP består hovedsakelig av tre tabeller, aros, acos, og aros_acos. Acos innholder koblingene mellom kontrollere og views i systemet. Aros inneholder referanser mot grupper og bruker, og koblingene mellom disse. Den siste tabellen er aros_acos hvor man kan lage koblinger mellom de to tabellene, og sette rettigheter mellom disse.

Selve innloggingsinformasjonen lagres i brukertabellen, der passordene er kryptert med SHA256 kryptering. For rettigheter bruker vi CakePHP sin ACL komponent, og for innlogging bruker vi Auth komponenten.

3.2 *Detaljert Design*

3.2.1 *LDAP* | *Uthenting av studentinformasjon* |

Om LDAP:
LDAP er forkortelse for Lightweight Directory Access Protocol, og er et språk som brukes til oppslag i en katalogtjeneste mot en server. Uninett[2] har en egen LDAP katalog som blir brukt av de fleste skolene tilknyttet Uninett. Denne LDAP katalogen inneholder diverse opplysninger om studenter og ansatte på de enkelte skolene. Det ligger selvsagt ikke ute personopplysninger eller annen kritisk data, og alle med tilgang til nettet kan søke opp personer ved Uninett sin søkemotor; http://www.katalog.uninett.no/ldap/finn/. Under ser vi et eksempel av uthenting av data fra denne:

Navn	Fornavn Etternavn
Organisasjon	Avdeling for informatikk og medieteknikk
E-post	fornavn.etternavn@hig.no
Hjemmeside	http://www.stud.hig.no/~080808/
Tittel	Student

Figur 7 – LDAP uthenting fra Uninett.

Bruk i prosjektet:
I prosjektet har dette blitt løsningen for uthenting av studentinformasjon til databasen våres. Denne løsningen gir selvsagt ikke ut all data vi skulle ønske vi kunne fått om studentene og ansatte hos HiG. Og vi vil heller ikke kunne hente informasjon om tidligere studenter, selve Alumnene, ettersom Uninett sin LDAP katalog inneholder kun nåværende studenter.

For uthenting av studentinformasjon har vi utviklet et eget skript[3] for å gjennomføre denne oppgaven. Vi endte da med over 3500 "test" brukere for Alumni systemet vårt, disse brukerne besto av alle studenter og ansatte ved HiG lagret på Uninett sin LDAP katalog.
Å bruke en LDAP løsning som dette for et eventuelt sluttprodukt burde for all del unngås. For det første vil det være mye data om studenter eller ansatte som ikke ble registrert. Man får heller ingen oversikt over hvilke studenter som faktisk fullfører studie og blir en Alumni ved Høgskolen i Gjøvik. Grunnen til at vi endte med denne løsningen for prosjektoppgaven våres er at som en bacheloroppgave kunne det vært uheldig å gi oss tilgang til personopplysninger eller andre sensitiv data om studenter og ansatte. Vi ser selvfølgelig at dette da ikke kunne gjennomføres, men vi har skrevet om vår tenkte løsning og hvordan denne burde bli utført.

[2] Uninett - De som driver nett og nettjenester for høgskoler/universitet
[3] Skript-filene ligger som vedlegg på CD

3.2.2 *Feide og Uninett* *| Som Innlogging og Autorisering |*

Nåværende løsning:
For at man skal få noe ut av å bruke Alumni system må det inneholde en del data i databasen. Disse dataene vil blant annet være brukerne av systemet; som Alumner, nåværende studenter og ansatte ved skolen. Hvor hver av disse gruppene har sine data som er viktig for den enkelte gruppen. Når systemet skal legge til en ny bruker er det ønskelig at brukeren/administrasjonen trenger å legge til minst mulig data manuelt. Dette for å unngå blant annet unødvendig mye administrerende arbeid; spesielt ved starten og slutten av semestrene hvor det er en naturlig økning av aktivitet.

Løsningen vi har så langt i prosjektet for registrering, er å hente ut data fra LDAP[4]-katalogen til Uninett. Ved hjelp av et skript henter vi ut alle studenter og ansatte som er registrert på Høgskolen i Gjøvik, dette står det mer under LDAP kapitelet. Hvorfor vi ønsket å se på en annen løsning for uthenting av data, har med følgende problemer:

- Studentene sine data inneholder kun; Navn, Avdeling, E-post (HiG-Mail), Studentnummer med adresse til hjemmeområde, Tittel (Student).
- Ansatte har i tillegg til data som studentene, også; Telefon- og Mobilnummer, Rom nummer ansatte holder til og et bilde. Det vil da være en del data om studentene og ansatte som hadde vært ønsket.
- Tidligere studenter blir ikke lagret i LDAP-katalogen, disse blir slettet 2-3 måneder etter fullført utdannelse (Vårsemesteret).
- Studenter eller ansatte som slutter i løpet av et skoleår, blir ikke fjernet før på samme oppdateringstidspunkt som punktet over.

Med disse problemene vil vi fort få en rekke flere problemer når vi skal ta hånd om registrering/innlogging i vårt system. Skal vi la brukerne registrere resterende nødvendig data selv, må vi ha en form for sjekk av data eller at administrasjonen manuelt godkjenner brukere og deres data. Sammen med dette blir det også mye manuelt vedlikehold av brukerne i systemet.

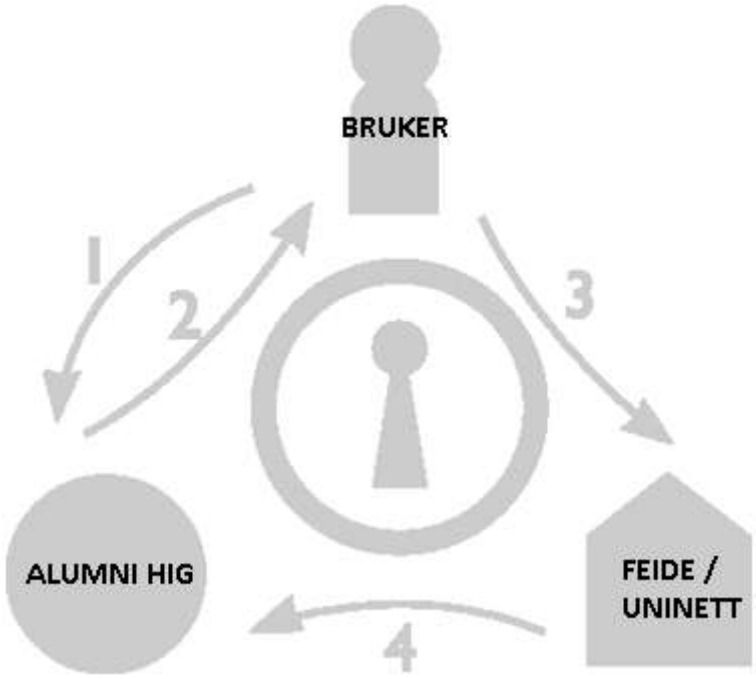
Tenkt løsning:
Men det er her Feide sammen med Uninett vil være en veldig god løsning. Feide er teknologi og plattform uavhengig, og tilbyr alle innen utdanningssektoren en standard identitetsadministrering. Høgskolen i Gjøvik har allerede en avtale med Feide og Uninett, som blant annet gjelder bruken av Studentweb[5]. Studweb er noe alle studenter og de fleste ansatte ved HiG allerede kjenner godt til, og er en nettside som blir godt brukt. Her holder også studentene oftest oppdatert kontaktinformasjon[6] ved hele studieperioden.

Vår tenkte løsning er da å få kjørt samme innloggingssystem for Alumni HiG, slik at det meste av nødvendig data blir automatisk overført til vårt system og vi får også en autentisering av brukeren via Feide/Uninett.

[4] Lightweight Directory Access Protocol
[5] Studweb - Studentenes side for utdanningsplan/resultater o.l.
[6] Figur 11

Når/hvis en student ønsker å være deltager i Alumni HiG, trenger studenten kun å registrere/logge seg inn med sitt studentnummer og passord på alumni siden. Hvilke data studenten ønsker å meddele med andre studenter eller skolen er opp til studenten selv å bestemme.

Under ser vi vår tenkte autentisering mot Feide og Uninett.



Figur 8 – Innloggingsprosess – hentet fra Feide Integration Guide[7]

Når en bruker logger seg inn i vårt Alumni system via Feide skjer følgende prosess:

1. Brukeren går inn på Alumni HiG sin innloggingsside via en nettleser.
2. Tjenesten sender brukeren til Feide sin innloggingstjeneste.
3. Her skriver brukeren inn sitt brukernavn og passord til feidesystemet, i vårt tilfelle studentnummer og selvvalgte passord.
4. Brukernavnet og passordet blir verifisert av Uninett, og hvis det godkjennes sendes brukerens data til Alumni HiG via feide. Her blir **kun** data godkjent mellom Feide og Alumni HiG sendt.

På neste side ser vi vår tenkte løsning visualisert, denne er basert på Universitetet i Bergen sin løsning for alumni registrering og innlogging.

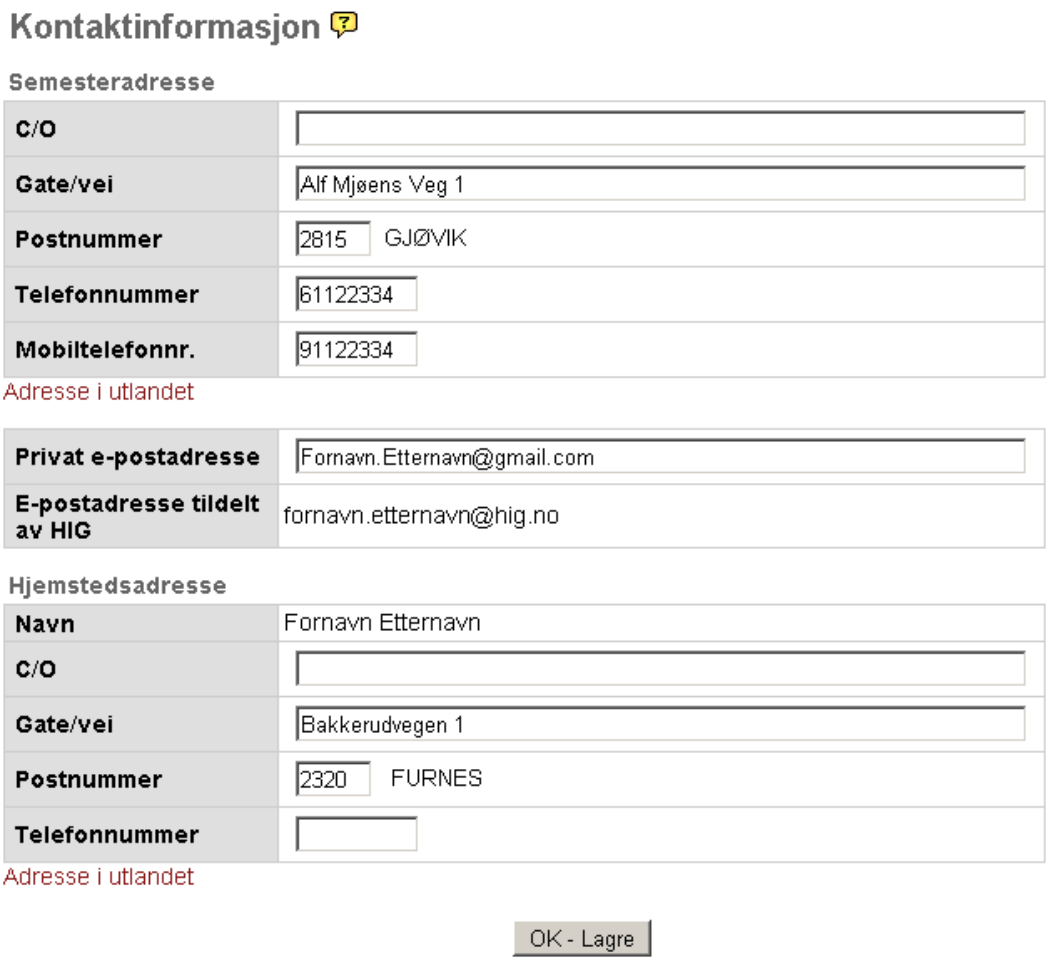
[7] http://www.feide.no/sites/feide.no/files/documents/Feide_integration_guide.pdf (mai 2010)

Tenkt visuell innloggingsprosses:

Figur 9 – Innloggingsside[8] med alternative innloggingsmetoder.

Figur 10 – Ved valg 1 over får vi følgende påloggingsvindu[9] opp

[8] Universitetet i Bergen sin løsning https://alumni.uib.no/pages/open/ (mai 2010)
[9] Standard påloggingsside for feide, eksempel https://www.studweb.no/ (mai 2010)



Figur 11 – Førstegangsinnlogging vindu, hvilke opplysninger ønskes registrert

3.2.3 Brønnøysundregistrene

Som oppslagsverk mot bedrifter:
Det å få koblet oss opp mot Brønnøysundregistrene var en løsning vi tenkte mye på å gjennomføre i prosjektet. Vår tenkte løsning var at når en bedrift for første gang ble registrert av en Alumni under sin jobbinformasjon, ville vårt system automatisk slå opp navnet på bedriften hos BRREG[10]. Og dermed kunne vi hente ut informasjon om bedriften deriblant besøksadresse, internettside, daglig leder, telefonnummer eller andre ønskede opplysninger. Hvor denne informasjonen dermed blir lagret i vår database over bedrifter. Med denne løsningen ville vi fått en mer automatisert registrering av bedrifter i vårt system, samt også at data om bedriften blir bekreftet av BRREG. Dette ville spare administrasjonen for jobb ved at man slipper det og manuelt måtte verifisere innholdet en Alumni har lagt inn om en bedrift.

Når en Alumni registrerer å ha jobbet i en bedrift som allerede eksisterer i databasen våres (fig. 1), trenger dermed ikke denne brukeren å registrere noe ekstra opplysninger en stilling og en stillingsbeskrivelse Alumnen har hos denne bedriften. Vi trenger dermed heller ikke å gjøre noe oppslag mot BRREG. Om brukeren finner noen av opplysningene feilaktive, kan brukeren forandre disse og sende en forespørsel til administrasjonen. Administrasjonen velger da om de vil godkjenne eller ikke godkjenne disse opplysningene.

Figur 12

Noe som er viktig å nevne er at tjenesten BRREG tilbyr koster penger hvis man ønsker tilgang til enhetsregisteret og foretaksregisteret for automatisk uthenting av data. Prisen har en årlig fast pris, samt en mengdepris på antall oppslag man gjør mot BRREG:[11]

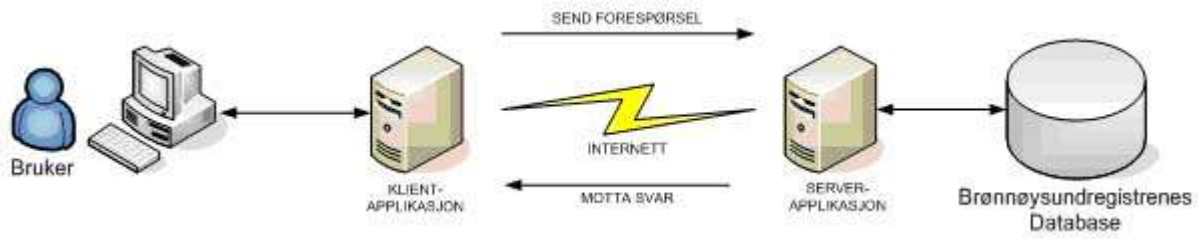
- Årlig fast pris : kr *5 000,-*
- Per oppslag : kr *0.50,-*
- Pakkepris antall oppslag
  - 20 000 : kr *8 000,-* *(40 øre per)*
  - 50 000 : kr *15 000,-* *(30 øre per)*
  - 200 000 : kr *50 000,-* *(25 øre per)*
  - 1 000 000 : kr *200 000,-* *(20 øre per)*

I forbindelse med prosjektet var vi i kontakt med en konsulent i Brønnøysundregistrene[12]. Og vi fikk i den sammenheng skaffet en avtale som ga oss fri tilgang til deres enhetsregister under hele prosjektperioden. For å hente ut data fra dette var vi avhengig av å bruke Web Services.

[10] Brønnøysundregisteret
[11] Priser hentet fra http://www.brreg.no/automatiske/webservices/private.html (mai 2010)
[12] Se vedlegg 2 – Mail transaksjon BRREG

Web Services:

Web Services [13] er teknologien BRREG bruker for å distribuerte data til sine kunder. Kort forklart baserer Web Services seg på applikasjon til applikasjon kommunikasjon. Dette vil si at brukerens programvare kan sende en forespørsel til programvare ved Enhetsregisteret, hvor denne igjen henter ut forespurt data og returnerer dette til brukerens programvare. Under ser vi en illustrasjon av prosessen for uthenting av data:



Figur 13 - Kommunikasjonsprosess[14]

For å koble seg opp mot Web Services hos BRREG trenger man et brukernavn og passord, dette for å verifisere tilgangen til Enhetsregisteret og for å identifisere kunden av tjenesten for blant annet loggføre antall oppslag som har blitt gjort. Avtalen vi fikk med BRREG ga oss en slik tilgang med fri bruk av antall oppslag.

Programvaren på vår side (klient-applikasjon) må utvikles selv, BRREG bidrar ikke med teknisk bistand på dette område. Men de er behjelpelige med å gi faglig rådgivning innen bruken av grunndata fra Enhetsregisteret.

For utvikling av programvaren må man ha kjennskap til XML-schema[15]. Det brukes ikke direkte XML, men noe som er basert på dette og kalles Web Services Description Language eller forkortet WSDL. Ved opprettelse av et slikt WSDL-dokument kan vi dermed hente ut data ved å gjøre en oppkobling mot WSDL-filen om ligger tilgjengelig på BRREG sitt webområde.

Dette var en løsning vi fikk sett litt på under prosjektet, men ikke fikk laget noe direkte programvare som ble brukt i det nåværende Alumni hjemmeområdet. Et problem som også dukket opp er at spørringer mot Enhetsregisteret **må** gjøres med organisasjonsnummeret til bedriften man skal ha informasjon om. Dette stiller da krav til at når en Alumni registrer jobbinformasjon, må brukeren kunne organisasjonsnummeret, noe som er svært uheldig. Dette er jo noe fåtallet av tidligere/nåværende ansatte i en bedrift som kan i hodet, og man er dermed avhengig av å gjøre et søk hos eksempelvis BRREG sin søkeside[16].

Med kravet om organisasjonsnummer satte vi utviklingen av vår tenkte løsning på vent. Noe som endte med at det dermed ikke ble gjort noe mer på denne delen under prosjektet. Dette oppslagsverket håper vi da kan bli tatt opp igjen ved en eventuell videreutvikling av vårt prosjekt.

---

[13] http://en.wikipedia.org/wiki/Web_service (mai 2010)
[14] Hentet fra http://www.brreg.no/automatiske/webservices/ (mai 2010)
[15] Extensible Markup Language
[16] http://w2.brreg.no/enhet/sok/ (mai 2010)

4 Programvare

4.1 *WampServer*

4.1.1 Hvorfor WampServer

WampServer er en pakke for **W**indows med **A**pache, **M**ySQL og **P**HP. WampServer er enkel å installere og sette opp, da man kun trenger å gjøre en helt vanlig Windows installasjon for å få en standard webserver. WampServer er lisensiert under GPL lisensen, og uten noen form for lisenskostnader.

Apache er en webserver for å vise webinnhold lokalt, eller ut mot internett. MySQL er en database som ofte brukes for webtjenester.
PHP er et skriptspråk for å kode applikasjoner og vise dette ut som webinnhold. Det er også med PHP vi i dette prosjektet kobler seg opp mot databasen, henter ut informasjonen og behandler denne for å vises ut i webserveren.
Både Apache, MySQL og PHP er basert på åpne og frie lisenser, og koster ingen ting i lisensavgifter å bruke.

Siden WampServer bruker lite resurser, samt er enkelt å starte og stoppe om ønskelig, fungerer denne veldig godt også på arbeidsstasjoner.
WampServer er også som nevnt tidlige basert på åpne og frie lisenser, dette gjør ikke bare at man slipper lisenskostnaden, men også at man slipper problematikken dersom man skulle ønske å gå over til et annet operativsystem enn Windows. Faktisk er både Apache, MySQL og PHP er fult mulig å bruke på de aller fleste systemer i verden.

WampServer leveres også standard med programvaren phpMyAdmin for administrering av MySQL databasen. I dette prosjektet har vi derimot valgt å bruke MySQL Workbench for dette formålet, da denne pakken er mye kraftigere, enklere og bedre å bruke.

4.1.2 Endringer før man kan begynne å bruke WampServer

Endringer i Apache:
For å kunne bruke Apache sammen med CakePHP, er man nødt til å skru på apachemodulen ”mod_rewrite”. Dette gjøres ved å endre linjen:

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

til

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Endringer i MySQL:
Som standard leveres WampServer uten noe passord på MySQL databasen, dette er ganske skummelt da dette betyr at alle kan få tilgang. For å endre dette går man inn på WampServer menyen, og inn i MySQL Console.
Fra MySQL Console skriver man inn blankt passord når man skal logge seg inn, og så skriver man så:

```
UPDATE mysql.user SET Password=PASSWORD('nytt_passord') WHERE User='root';
FLUSH PRIVILEGES;
```

Endringer i PHP:

Dersom man ønsker å koble seg opp mot skolens LDAP server, må man aktivere PHP modulen ” php_ldap.dll”, dette gjøres ved å gå inn i alle konfigurasjonsfilene ”php.ini” til PHP under WampServer, og endre:

```
;extension=php_ldap
```

til

```
extension=php_ldap
```

4.2 *jQuery*

4.2.1 Hvorfor jQuery

Når vi så etter et rammeverk for JavaScript så vi først og fremst etter et godt og omfattende rammeverk med mye funksjonalitet, samt et rammeverk med god støtte for nettlesere. Vi fant fort ut at dette kunne jQuery gjøre på en god og elegant måte. Siden jQuery er basert på de åpne lisensene MIT og LGPL, er rammeverket også fritt for lisenskostnader og kan brukes akkurat slik man måtte ønske å bruke det.

jQuery har også en tankegang som gjør at koden man skriver blir veldig logisk, og samtidig veldig minimalistisk. jQuery biblioteket er som nevnt ovenfor veldig omfattende, og allikevel er hele biblioteket på snaue 70kb.

Siden jQuery er et rammeverk som hele tiden er under utvikling, og har store firmaer bak seg som Google, Mozilla, Digg, og overraskende mange andre internettsider. Dette er også kanskje grunnen til at jQuery stadig vokser og stjeler markedsandeler.
I Norge brukes jQuery pr. 07.05.2010 blant annet på nettsider som Feide, VG, Finn, og Stortinget.

4.2.2 jQuery i prosjektet

I dette prosjektet har vi brukt jQuery i sammenheng med å lage dynamisk grafiske funksjoner, som skjuling og visning av informasjon. Vi har også brukt jQuery for Ajax[17] behandling. Med å bruke jQuery har vi kunnet korte ned en del kode, samt skrevet veldig pen, ren og logisk kode.

Siden jQuery er basert på MIT og LGPL lisensene, har også de fleste tredjeparts tillegg basert på jQuery også disse lisensene. Dette kommer veldig godt med, og har resultert til at vi i dette prosjektet ikke har måttet ”finne opp kruttet på nytt”, og heller kunne brukt disse tilleggene. Disse tilleggene er da selvfølgelig uten noen form for lisenskostnader.
Tilleggene vi har brukt er AutoComplete, jsTree og TinyMCE. For å finne ut mer om disse, se medlagte vedlegg til rapporten.

4.2.3 jQuery og CakePHP

Vi bestemte oss allerede tidlig i prosjektet at vi skulle bruke jQuery, noe som var ganske lurt da vi etter hvert bestemte oss for å bruke CakePHP som rammeverk for PHP. Dette er da fordi CakePHP fra versjon 1.2.x til versjon 1.3.x byttet over til jQuery som standard bibliotek for JavaScript.

Etter at CakePHP bestemte seg for å bruke jQuery som standard bibliotek, gikk de over til å bruke noe som heter en ”JS hjelper” for å hjelpe brukeren med å kode. Vi har som regel skrevet koden manuelt, da JS har veldig dårlig støtte for bruk av tillegg til jQuery.

I tidlige versjoner av CakePHP var ”Prototype” det rammeverket for javascript som ble brukt. På denne tiden brukte man hjelpere av typen ”javascript” og ”ajax”. Disse vil fortsatt fungere

[17] Ajax er en teknologi for å kunne hentet ut informasjon fra et PHP skript og putte dette i en nettside som allerede er ferdig lastet uten å måtte laste siden på nytt. Dette resulter i et veldig dynamisk innhold på siden.

i versjon 1.3.x. Utviklingen av disse modulene er derimot stoppet, og disse hjelperne vil bli fjernet ved en senere versjon av CakePHP.

JS hjelperen som brukes i versjon 1.3.x og senere, har også mulighet for å bruke både ”Prototype” og ”MooTools” rammeverket. Dersom man bruker selve hjelperen for å kode javascript vil det være enklere i ettertid å skifte mellom disse rammeverkene.

Vi har sett litt på ytelsen til de tre rammeverkene ved å kjøre testen ”slickspeed”[18], som finnes på MooTools sine hjemmesider. Man kan her fort tolke at Prototype rammeverket er en del tregere enn jQuery og MooTools.

Da jQuery både har flere nedlastbare tillegg, og vi har fått et generelt inntrykk av at jQuery er bedre likt, at CakePHP har favorisert jQuery, og at det er flere og flere som går over til jQuery. Ser vi ingen grunn til å velge noe annet rammeverk for javascript koden våres.

[18] SlickSpeed(pr. 03.05.10): http://mootools.net/slickspeed/

4.3 *CakePHP*

4.3.1 Hvorfor CakePHP:

Vi begynte å se etter PHP rammeverk først for lett å kunne håndtere mulighet for å betjene felles grensesnitt for alle undersider, og enkelt kunne bytte til et annet grensesnitt om ønskelig. Vi så også på muligheten for språk-, passord- og brukerhåndtering i prosjektet, samt åpenheten i rammeverkenes lisenser.

Vi fant fort at det var ganske mange grensesnitt ute på markedet som kunne brukes til dette, det som derfor ble veldig avgjørende var to faktorer. Den første var å fjerne alle som ikke var oppdatert på over ett år. Den andre faktoren vi så på var at vi enkelt og greit så etter hva som ble brukt av norske firmaer, og hva som var aktuelt på det norske markedet.

Etter dette satt vi med kun to aktuelle rammeverk, Zend og CakePHP. Vi gikk så for Cake siden vi så på Cake som et mer omfattende rammeverk for større prosjekter, mens Zend ville vært bedre mot små og enkle applikasjoner. Grunnen til at vi så på Cake som det riktige system for dette prosjektet, var hvor dynamisk Cake faktisk er. Og med tanke på at Cake er modulbasert, kan enkelt prosjektet våres bli tatt over av en annen gruppe senere.

En annen avgjørende faktor var at dersom man starter med Cake, kan man når som helst også ta i bruk Zend under Cake, men dersom man starter med Zend, er det vanskelig og tungvint å ta i bruk Cake.

Valget ble derfor til CakePHP, noe vi fortsatt mener har vært et meget godt valg.

4.3.2 Hvorfor ikke CakePHP

CakePHP har noen få problemer, og noe som kunne vært gjort bedre. Største problemet med Cake, er at Cake er ganske unikt, og derav en god del man må sette seg inn i før man kan starte kodingen. På den lyse siden er derimot Cake ganske enkelt og raskt å utvikle mot når man først har satt seg inn i rammeverket.

Et annet problem med CakePHP, er at dersom man ikke er oppmerksom på hva man gjør, vil applikasjonen fort kunne bli veldig treg. Dette er derimot noe som ofte er fullt mulig å kompensere for, og derav få ytelsen opp igjen.

Når man utvikler med CakePHP for første gang, vil man oppleve at ordet magi veldig ofte går igjen i dokumentasjonen. Når ordet magi blir brukt i dokumentasjonen betyr det at ting skjer i systemet som ikke logisk skulle skje i den sammenhengen, men allikevel er riktig. Dette er noe som fort blir ganske så frustrerende for utvikleren. I disse situasjoner har vi ofte vært nødt for å analysere kildekoden til CakePHP for og faktisk forstå hvorfor utfallet blir som det blir.

Ellers er det en del dårlig dokumentasjon, dette har hovedsakelig vært på grunn av at vi har brukt en uferdig utgave av CakePHP, og vil bli rettet før det blir aktuelt å jobbe videre på prosjektet.

Vi har også prøvd å spørre etter hjelp på CakePHP sin offisielle informasjonskanal. Denne er da gjennom IRC[19] teknologien. Dette er noe som har vist seg vanskelig, da flere på CakePHP kanalen er lite hjelpsomme, og derav ønsker å krangle fremfor å hjelpe.

Et lite problem med CakePHP er også navnet som veldig ofte går igjen. Siden rammeverket heter CakePHP, får man veldig lyst på kake. Dette er noe som også har resultert til at vi har brukt en liten formue på kaker til gruppen.

4.3.3 Forklaring på CakePHP

CakePHP er et rammeverk som har som formål å hjelpe utviklere til og lettere kunne lage mest mulig dynamiske systemer. CakePHP fungerer som en oversetter, slik at dersom en utvikler lager et system opp mot en bestemt database, vil dette systemet uten problemer også kunne brukes mot andre databaser.

CakePHP er basert på de åpne MIT og GPL lisensene, disse er gratis å bruke og det koster oss dermed ingenting i lisenskostnader. Lisensene er også veldig åpen for at utvikleren skal kunne gjør endringer om ønskelig. Det er derimot ikke anbefalt å gjøre endringer i kjernen av selve rammeverket, da disse vil være veldig utsatt ved oppdatering.

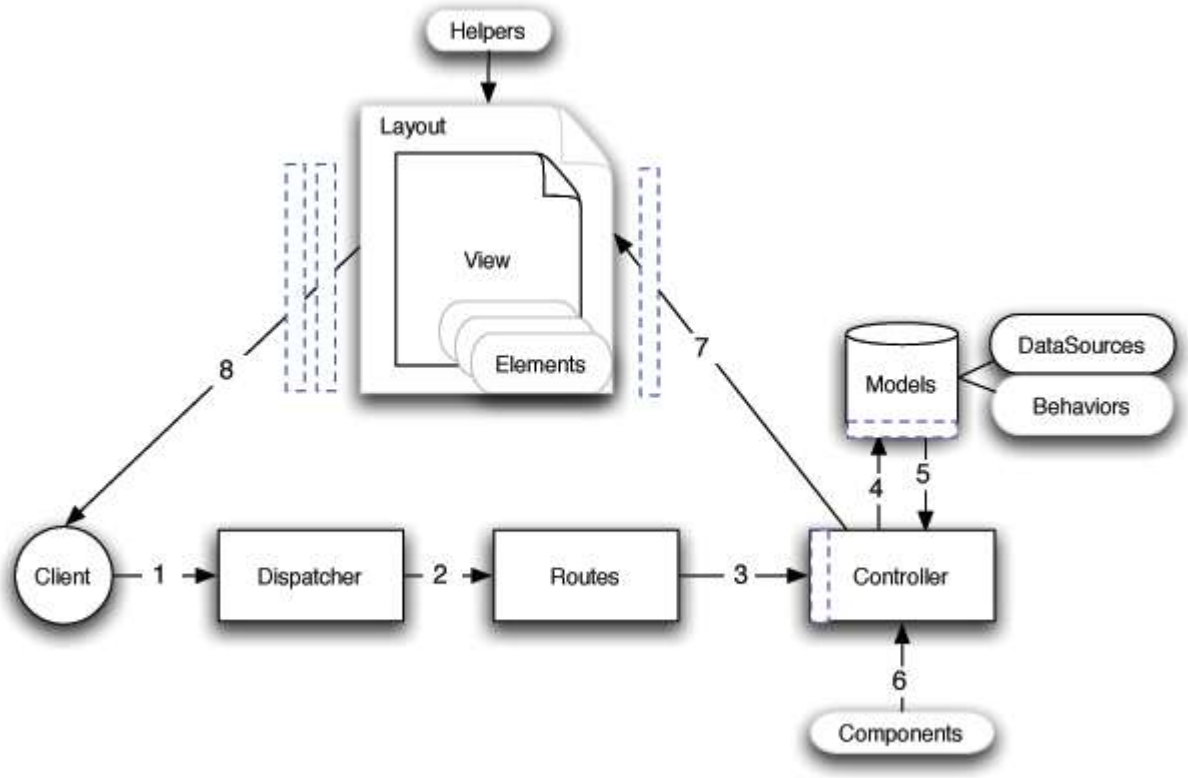
CakePHP er veldig greit å jobbe med, men skal man bruke Cake for første gang, er man nødt til å måtte sette av en del tid for å lære seg hvordan systemet fungerer. Cake har under dette prosjektet dessverre hatt veldig dårlig dokumentasjon. Dette er fordi vi startet å bruke Cake i overgangen til versjon 1.3.x, noe som resulterte til at manualen ikke var skikkelig skrevet, og at de aller fleste eksempler på internett og i bøker ikke fungerte som de skulle.

På grunn av manglende dokumentasjon, har vi måttet finne ut det meste rundt systemet på egenhånd, ofte ved hjelp av prøve og feilemetoden, samt i noen tilfeller også måtte lese og analysere kildekoden til selve Cake kjernen. Heldigvis har dokumentasjonen blitt en del bedre mens vi har jobbet med prosjektet, noe som gjør at dokumentasjonen fra Cake sin side begynner å bli veldig god dersom noen skulle ta opp igjen prosjektet senere.

Når man lager en applikasjon i Cake, kan man enten lage denne i ”app” mappen under rammeverket eller en mappe med valgfritt navn som ikke allerede er i bruk. Dersom man bruker ”app”, vil denne blir satt til hovedapplikasjonen, og om man fra en nettleser går rett inn i hovedmappen til Cake, blir man automatisk sendt videre hit. Ellers må man skrive inn applikasjonsnavnet manuelt bak adressen til Cake.

[19] IRC - Internet Relay Chat. En teknologi for å kommunisere med andre over internett.

Flyten i systemet under en slik forespørsel kan illustreres som figuren under:



Bilde hentet fra CakePHP v1.3 manualen, punkt 2.2 – Sist sett 05.05.2010.
Figur 14

Denne illustrasjonens elementer kan deles inn i tre hoveddeler:

1-3 Spørringen:
Plukker fra hverandre spørringen fra klienten, og gir denne videre til kontrolleren og generer en side til klienten istedenfor å gå rett inn på den logiske adressen. Filer relatert til spørringer og deling av disse, finnes under applikasjonens ”config” mappe, i filen ”routes.php”.

”Dispatcher” filtrer ut spørringen dersom det er en feil i adressen, og sender brukeren til annen ønskelig adresse. Eksempelvis sender brukeren til innloggingssiden dersom man prøver å gå inn på en side som krever innlogging og man ikke er logget inn.

”Routes” henter ut elementer i adressen og bruker disse for å generere og/eller konfigurere siden brukeren skal få opp. Dette kan for eksempel være at brukeren besøker ”www.sidens_url/eng/alumnis/minside/12/34/56/”. Her ville ”routes” ha gjort følgende:
”eng” sendes inn i hovedkontrolleren som setter språkkonfigurasjonen til engelsk.
”alumnis” forteller systemet at kontrolleren alumnis skal brukes.
”minside” forteller systemet at funksjonen(og viewet) i kontrolleren minside skal brukes.
12, 34, 45 blir sendt inn som parametere og det er opp til utvikleren selv om disse skal brukes til noe. Eksempelvis kunne disse fylt inn data om alumni.

3-7 Applikasjonen:
Er her selve utviklingen kommer inn i bildet, og det er her spørringer mot databasen og behandling av data foregår. Filer relatert til selve applikasjonen finnes under applikasjonens ”controllers” og ”models” mappene. Standard Components er derimot en del av Cake biblioteket og finnes derfor som en del av selve kjernen til Cake.

Controller er den delen av Cake som knytter hele applikasjonen sammen, og er den delen av systemet hvor mesteparten av koden skrives. Kontrolleren kan behandle parametere som blir sendt inn av brukeren for så å sende en spørring mot databasen gjennom modellen basert på disse. Dataen som kommer tilbake kan kontrolleren så behandle lokalt, før den sendes videre inn i ”viewet” og genererer en side for brukeren.

Models definerer koblingen mellom informasjonen som behandles av Cake, og den faktiske informasjonen i databasen. Alt som defineres i modell filen, er koblinger og restriksjoner som gjøres internt i Cake. Dette betyr at dersom man for eksempel definerer en fremmednøkkel i modellen, behøver man ikke å ha en database som støtter fremmednøkler da Cake i dette tilfellet tar over denne rollen. I modellen defineres også restriksjoner på hvordan informasjonen skal se ut før den puttes inn i databasens tabeller. Dette gjør at man her kan legge inn prosedyrer som sjekker at alt er riktig formatert. Eksempler på dette er at en e-postadresse må inneholde ”@” og ”.” og på riktig plassering. At et passord må være på minimum x tegn, og maks y tegn.

Components er ferdige komponenter i Cake som enkelt kan brukes til å utvide systemet slik man måtte ønske. Komponentene vi har brukt i dette systemet er for det meste for at applikasjonen skal kunne ha flere brukere og håndtering rundt dette.

7-8 Utseende:
Generer utseende for siden brukeren skal se, dette har kun med det grafiske. Filer relatert til prosjektets utseende finnes i applikasjonens ”views” mappen, med unntak av standard ”helpers” som tilhører Cake biblioteket.

Layout er den delen som er mest felles for alle brukere, og er det grafiske ”skallet” som brukes på alle sider. Eksempel på dette er at man kan ha et layout for Høgskolen i Gjøvik, hvor alle deres studenter vil se et design, mens studentene ved Høgskolen i Lillehammer vil se et helt annen design, selv om de i utgangspunktet er på samme side, og informasjonen som vises er den samme.

Elements er felles kode som er ønskelig å kunne hente ut dersom man trenger det, eksempel på dette er menyer, informasjonsbokser og lignende.

View er den informasjonen som er unik for hver side, det er her det vises brukerspesifikk informasjon basert på hva som hentes ut fra databasen gjennom kontrolleren.

Helpers er kun for å hjelpe til med å rydde opp koden for utviklerne, og brukes for å generere strukturert og riktig data, slik at utvikleren skal kunne konsentrere seg mer mot funksjonaliteten fremfor å måte bruke tid og resurser på å lete etter strukturfeil i koden, samt skrive typisk strukturkode om og om igjen i systemet.

4.3.4 Konfigurasjon av Cake:

CakePHP henter ut alle konfigurasjoner fra mappen ”config” under applikasjonen. I denne mappen kan man finne ”database.php”, som inneholder informasjon som trengs for å koble opp mot databasen.
For å konfigurere selve kjernen av applikasjonen gjøres dette i ”Core.php”. Denne konfigurasjonsfilen inneholder blant annet diverse sikkerhets -, tegnsett -, og feilmeldingsnivå innstillinger.

4.3.5 Websiden:

Alt som hører til selve websiden som bilder, JavaScript, og stilsett filer, vil man kunne finne under en mappe som heter webroot i applikasjonen.
På grunn av at Cake tar fra hverandre alle adresser som blir sendt inn i applikasjonen av brukeren, vil også stiene til websidens filer kunne virke ulogisk. Her er det derfor anbefalt å bruke CakePHP sine ”helpers” filer. Ved å bruke ”helpers”, og ikke definere stiene til disse filene manuelt, vil man hele tiden kunne være trygg på at man får hentet ut disse.

4.3.6 CakeConsole:

CakeConsole er et kommandobasert program for manuelt kunne endre informasjonen i databasen knyttet mot Cake.
Grunnen til at man har dette programmet og ikke bruker en vanlig database editor, er Cake sin måte å bygge opp og knytte informasjonen i databasens tabeller sammen. Dette fordi Cake kan ha fremmednøkler i databasen, selv om databasen i utgangspunktet ikke støtter dette. En annen grunn er at dersom man begynner å jobbe med rettigheter i Cake, vil denne informasjonen bli lagt inn i databasen som traverserte[20] trestrukturer. Denne informasjonen vil bli vanskelig å tolke av mennesket, og vanskelig om ikke tilnærmet umulig å endre manuelt dersom man har mange oppføringer i databasen.

[20] Å traversere treet betyr å vise treets struktur som en tekststreng som man så kan bygge treet opp igjen fra.

4.4 *Annen programvare*

NetBeans IDE
For selve utviklingen brukte vi hovedsakelig gratisverktøyet NetBeans IDE 6.9M1. Denne versjonen har noe støtte for feilrettingsmekanismer rundt CakePHP. NetBeans IDE er ellers veldig oversiktelig og godt å jobbe med.

Notepad++
For raske enkle endringer har vi brukt Notepad++. Dette programmet er ikke i nærheten av like kraftig som NetBeans IDE, men derimot en del raskere å starte opp.

MySQL Workbench
For databaseadministrasjon har vi brukt MySQL Workbench. Dette er et gratisprogram som gir mye funksjonalitet, og gjør det enkelt å administrere databaser.

LDAP Browser
Vi har brukt LDAP Browser litt i systemet for å koble oss opp mot skolens LDAP server for å få et innblikk i dets oppbygging og struktur.

Cobian Backup
På serveren har vi brukt Cobian Backup for å lage backup prosedyrer på koden og dokumentasjonen. Cobian Backup er et enkelt og gratis program for å håndtere dette.

DropBox
For en ekstra sikkerhet, har vi brukt DropBox som et tillegg til prosjektet for å oppdatere hovedfilene på serveren. DropBox er gratis opp for de første 2 gigabytes lagret, og koster penger utover dette. De tilbyr også en viss form for versjonskontroll, slik at tidligere versjoner av filene kan hentes ut igjen i ettertid.

5 Implementering og realisering

5.1 *Database*

5.1.1 Utvilkling av databasen:

I databasefasen har vi vært gjennom mange utkast på hvordan databasen burde se ut. Vi har under hele utviklingen vært ofte i kontakt med både prosjekt- og tekniske veileder. Det er spesielt Simon McCallum og Jayson David Mackie vi har fått mye hjelp og tips av underveis i prosjektet, de har vært en stor ressurs vi var heldige å få bruke. De tok også opp problemstillinger vi ikke hadde tenkt på, noe som gjorde at sluttresultatet ble med solid og robust for fremtidlige utfordringer. Se vedlegget[21] for hvordan databasen var tenkt etter databasefasen.

Selv om vi hadde et veldig godt utgangspunkt etter å ha jobbet oss ferdig med databasefasen. Var det naturligvis noen forandringer som måtte gjøres ved introdusering av cakePHP. Under vil vi gå gjennom forandringene som ble gjort underveis i webfrontfasen.

5.1.2 Databaseforandring ved overgang til cakePHP:

Ved overgang til rammeverket cakePHP medførte det at vi måtte gjøre enkelte forandringer i databasen. Det var også forandringer vi valgte å gjøre underveis for også å effektivisere databasen. Vi vil videre se på de forskjellige problemstillingene og endringer vi gjorde underveis og som til slutt ga oss databasen vi sitter igjen med nå ved prosjektslutt, se vedlegg[22].

Databasemotoren:
Dette var et tema som dukket opp underveis i kodingen av webfrontenden. Vi visste jo allerede at det fantes en rekke alternativer innen valg av databasemotorer, og ved tidligere vurderinger falt valget på InnoDB. Hovedgrunnen for dette valget var jo at vi ønsket støtte for fremmednøkler.

Etter å ha fått mer kjennskap til cakePHP ble vi oppmerksom på at vi ikke lenger var avhengig av fremmednøkler støtte i selve databasemotoren. Dette har seg med at selve rammeverket cakePHP tar seg av denne henvisningen til andre data når dette er ønskelig. Og vi kunne dermed prioritere andre krav til databasen enn hva vi hadde initielt.

Valget falt til slutt på MyISAM som også er satt som standard databasemotor i MySQL. MyISAM bruker, i forhold til InnoDB, mye mindre harddiskplass, samt at den også bruker mindre minne under bruk. Minnebruken kommer av at InnoDB gjør mye mer mellomlagring ved data- og indeksbehandling, men det er også dette som gjør at InnoDB er mer robust ved store datamengder. Ved vurdering av et Alumni system kom vi fram til at det vil ikke være veldig store datamengder eller transaksjoner.
MyISAM er og raskere fordi denne databasemotoren foretar komprimering av data, som igjen fører til raskere transaksjoner for brukerne. Den har også bedre søking enn InnoDB.

[21] Se vedlegg 3 - Database
[22] Se vedlegg 4 – CakePHP Database

Navnendring:
CakePHP krevde litt forandringer ved navnsetting av tabeller og kolonner. Hver tabell er blant annet avhengig av å ha en egen ID kolonne, som også må hete ID for å gjenkjennes i rammeverket. Ved refereringen fra en kolonne til en annen tabell måtte også navnsetting følge visse krav for at utviklingen av systemet skulle gå raskere. Kolonnen må arve navnet etter hva tabellen heter og avsluttes med "_id". Eksempelvis vil E-post tabellen ha en kolonne som heter "user_id", som gjør at en e-post tilhører User med iden til "user_id".

Noe som også er anbefalt ved CakePHP er at tabeller skrives i flertall, hvor eksempelvis "Events" er hva tabellen heter, mens "Event" er en spesifikk event vi henter ut ved hjelp av en id. Denne navngivingen ble ikke fulgt helt under prosjektperioden, så enkelte av tabellene ble ikke skrevet med flertallsnavngivingen.

Tabeller:
Databasen vi sitter igjen med etter prosjektperioden inneholder ikke alle tabellene som ble laget under databasefasen. Dette på grunn av at CakePHP og krever at man setter opp modeller, views og kontrollere for hver enkelt tabell, har vi valgt å legge til tabeller etter som vi trengte dem. Ved implementering av ny funksjonalitet i videre utvikling kan nye tabeller enkelt leges til grunnet at CakePHP er modulbasert.

Tabellene aros, acos og aros_acos er tre tabeller vi ble nødvendig når vi skulle ha opp autoriseringssystemet med ACL. Nærmere forklaring av disse er tatt opp i autoriserings/innloggings delen av prosjektrapporten.

"Users" tabellen har også tatt mer over for "Alumni", samt blitt sammensveiset med "Login" tabellen. Dette grunnet det var mer naturlig å bruke "User" tabellen på denne måten i CakePHP.

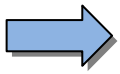
Ved "Messages" tabellen har vi også utforsket litt bruken av ENUM som datatype. Denne kan fylles med forskjellige tekst som "lest", "ikke lest" og lignende, dermed kan man sette at en beskjed er lest ved disse verdiene og ikke med "kryptiske" verdier som at 1 = lest, 0 = ikke lest.

5.2 *Kode metoder*

5.2.1 HTML, JavaScript og PHP

Når vi har kodet Html og JavaScript i dette prosjektet, har vi ofte utnyttet situasjonen med å putte selve koden under "echo" funksjonen til PHP.

```
$().ready(function() {
    //Skjuler element:
    $('#id').hide();
});
```



```
echo
"$().ready(function() {".
    //Skjuler element:
    "$('#id').hide();".
"});";
```

Tabell 1

Fordelene med å gjøre det på denne måten, er at brukeren slipper å hente måte laste inn unødvendig informasjon som kommentarer, mellomromstegn, tabulatortegn, og linjeskift. Dette korter ned betydelig hva brukeren er nødt til å laste inn, og siden dette kun leses av nettleseren, vil ikke brukerne merke noe til dette.

Resultatet for brukeren vil derfor i dette tilfelle bli: ” $().ready(function() { $('#id').hide() }); ” Dette er mye raskere å laste, men ikke så veldig brukervennlig. Om koden er brukervennlig eller ikke å lese for sluttbrukeren, har derimot ingen verdens ting å si så lenge selve utvikleren har pen og ryddig kode.

5.2.2 SQL til CakePHP

Når man bruker CakePHP vil en SQL[23] spørring bli noe annerledes enn det man er vant med. I eksemplet under skal vi sammenligne to like spørringer, der den ene er skrevet i SQL, og den andre i CakePHP:

SQL:

```
SELECT Alumni.id, User.id
FROM Users AS User, Alumnis AS Alumni
WHERE
( Alumni.fnavn = 'navn' OR Alumni.enavn = 'navnesen')
AND User.skole_id = 1  AND User.aktiv = true AND User.id = Alumnis.user_id;
```

CakePHP:

```
$this->User->find( 'all', array(
        'fields' => array( 'Alumni.id', 'User.id' ),
        'conditions' => array(
                'OR' => array(
                        'Alumni.fnavn =' => 'navn',
                        'Alumni.enavn =' => 'navnesen',
                ),
                'User.skole_id =' => 1,
                'User.aktiv =' => true,
)));
```

Tabell 2

Som man kan se er CakePHP spørringen noe mer komplisert og innviklet enn den skrevet i SQL. Fordelen med å skrive spørringene i Cake fremfor å gjøre det i SQL ser man først når man får større og mer komplekse spørringer. Dette er fordi Cake er mer dynamisk mot endringer i koden, og med dette kan man enklere knytte programkode direkte mot spørringen.

Man kan også se ut ifra disse to eksemplene at er at i CakePHP sitt tilfelle behøver man ikke definere noen form for fremmednøkkel. Dette er fordi CakePHP allerede har definert denne koblingen i modellen, og vet at disse to er koblet sammen av typen en til en. Det som er noe merkelig, er at man i Cake sitt tilfelle bare kan be om å få Alumni relatert informasjon direkte uten og definer hvilke tabeller man skal hente informasjonen fra.

[23] SQL - Structured Query Language. Språk for å kommunisere mot databaser.

Dette er lov å gjøre i Cake så lenge man har definert fremmednøkkelen til Alumni i modellfilen til User.
CakePHP er nemlig smart nok til å forstå at man da også er ute etter informasjon fra Alumni, og kompensere for dette.

Det man også kan se ut ifra dette eksemplet, er at ”Users AS User” og ”Alumnis AS Alumni” ikke er med i Cake spørringen. Dette kommer av at modellfilen er definert i entallsform, og dersom man ikke legger til noe ekstra informasjon, vil ikke flertallsformen fungere i Cake spørringen.

Behandling av resultatet
Når man gjør en spørring med CakePHP, vil svaret fra denne bli satt inn i en matrise som er logisk oppbygd etter data. I tilfellet over vil man derfor få en matrise som er bygd opp på følgende måte:

```
array(
        'User' => array( 'id' ),
        'Alumni' => array( 'id' )
);
```

Tabell 3

Dersom man da kaller matrisen for eksempelvis “$x”, vil man kunne hente informasjonen ved å bruke: $x['User']['id'] og $x['Alumni']['id']

Å hente ut informasjonen fra matriser på denne måten, og gjøre behandling på denne, er mye enklere og mer oversiktelig enn dersom man skulle ha brukt vanlige SQL spørringer.

Om man henter ut mer informasjon enn bare id, vil denne bli laget på samme måte som over.

5.3 *CakePHP*

5.3.1 Før man starter med applikasjon:

Før man starter opp en applikasjon laget i Cake, er man nødt til å ta en titt inn i ”core.php” under konfigurasjonsmappen til applikasjonen.

Det man skal se etter her er at feilsøkingsmodus er skrudd av. Grunnen til at man skal skru av denne, er for at brukeren ikke skal bli forvirret av advarsler og feilmeldinger som er ment for utviklerne, samt at ytelsen vil bli en del bedre.
Feilsøkingsmodus skrus av ved å sette ”Debug level” til 0.

Av sikkerhetsgrunner bør man endre ”Security salt” og ”cipherSeed” også. Disse fungerer som en krypteringsnøkkel, og bør derfor være noe annet enn standard nøklene.
Merk at dersom man endrer disse, vil ikke lengre passordene på de gamle brukerne fungere, og derfor bør man gjøre dette før systemet tas i bruk.

5.3.2 Components

5.3.2.1 Auth

Auth er en komponent som tar seg av alt som har med å passordbeskytte deler av applikasjonen, slik at kun de som faktisk har en bruker, skal kunne se denne data.
Auth komponenten inneholder rutiner for innlogging, utlogging, og passordkryptering.

Det kan defineres fritak fra hvilke sider som er avhengig av en innlogging ved å putte denne koden inn under "beforeFilter()" i kontrolleren:

```
$this->Auth->allow( ... );
```

Dersom brukeren prøver å gå inn på en side hvor det trengs innlogging for å gå inn, vil denne brukeren automatisk bli sendt til innloggingssiden istedenfor. Når brukeren så logger seg inn her, vil brukeren så bli sendt tilbake til den siden som ble prøvd besøkt først.

Auth komponenten jobber som standard ifra "Users" kontrolleren, og forventer at det her er definert en funksjon som heter login og en som heter logout. Disse funksjonene vil CakePHP legge til en del skjult funksjonalitet, slik at utvikleren ikke behøver å bry seg om dette.
Dersom man skal logge inn fra et annet sted, vil ikke Cake se på dette logisk, og man er nødt til å bruke funksjonen " `$this->Auth->login( ... );` " for å logge seg inn.

Passord
I CakePHP har man tre alternativer til passordkryptering, MD5, SHA1, og SHA256. Vi valgte SHA256, da dette er den sikreste av de tre alternativene. Det man må passe på når man bruker SHA256, er at siden SHA256 lager en lengre hash[24] av passordet. Dette resulterer til at man må bruke et større felt for å lagre passordet, enn det som er oppgitt i manualen. I dette prosjektet har vi brukt et felt på 80 tegn, og dette er da langt nok for SHA256.

[24] Passordhash er resultatet etter at passordet er kryptert.

Vær imidlertid oppmerksom på at man ikke får noen feilmelding dersom man har for lite lagringsfelt i databasen, noe som fort kan resultere til at man begynner å feilsøke kode. Dersom man har for lite lagringsfelt i databasen, vil man bare de deler av hashen lagres, og man vil aldri kunne logge seg inn i ettertid.

Alle disse tre krypteringsalgoritmene er av typen enveiskrypteringer, det vil si at det er umulig å dekryptere passordet tilbake til originalen. Dette vil si at når en bruker logger seg inn, så må passordet brukeren har plottet inn krypteres, for så å sammenlignes med det krypterte original passordet.

Når man bruker enveiskrypteringer har man en liten ulempe med at flere passord kan ha samme hash, og derav kan flere andre passord også fungere for å logge seg inn for en bruker. Å treffe et av disse andre passordene er derimot så lite sannsynlig at dette ikke kan regnes som en sikkerhetsrisiko. Fordelen er derimot at dersom noen skulle klare å komme seg inn i systemet med metoder som "brute force hacking"[25], vil man da høyst sannsynligvis få noe helt annet opp på skjermen enn originalpassordet, og kan da ikke prøve dette passordet på andre tjenester brukeren har konto.

Innlogging
Innloggingen kan enten skje ved at brukeren fyller ut et html skjema under login, eller at systemet finner en cookie[26] lagret på brukerens maskin, med riktig innloggingsinformasjon.

En cookie opprettes ved at brukeren trykker på "husk meg" feltet ved innlogging. Denne cookien må logges inn ved hjelp av auth sin login funksjon som nevnt tidligere. Prosedyren rundt dette har vi lagt i App kontrolleren, slik at en sjekk etter cookies alltid blir kjørt før en side vises.
Siden vi har valgt å gjøre det på denne måten, vil man aldri kunne merke at man automatisk blir logget ut ved inaktivitet dersom man har huket av for "husk meg" ved vanlig innlogging.

Dersom man logger seg inn på vanlig måte gjennom html skjema, vil Cake her automatisk lage en passord hash av skjemafeltet "password" og bytte ut passordet med denne hashen. Dette er det siste som skjer etter man trykker på knappen "log inn", og dette kan by på noen problemer.

Problemene som kan oppstå med at passordet hashes, er at dersom man ikke tømmer passordfeltet ved feil passord, vil det inntastede passordet byttes ut med hashen til det passordet som er feil i skjemaet.
Dette kan ordnes på to måter, enten ved å gjøre som nevnt over å tømme skjemaet med å sette at feltet alltid skal starte med å vise tomt felt, eller å kalle feltet for noe annet enn "password", og kopiere innholdet over til et skjult felt kalt "password", eller hashe passordet manuelt.

[25] Brute Force Hacking, er at en datamaskin prøver alle kombinasjoner en etter en til man finner en som passer.
[26] En informasjonskapsel som lagres lokalt hos bruker og kan hentes ut igjen av applikasjonen senere.

Registrering
Når man registrerer en bruker må man passe på at all informasjon i html formen stemmer overens med navnene på database og modell feltene. Om man gjør dette er det veldig rett fram under registrering.

Siden det i registreringsprosessen er ønskelig at brukeren må gjenta passordet sitt for å se at det stemmer, blir det derimot noe problemer. Det første man bør gjøre her, er å faktisk putte feltet man skal sammenligne mot først, og la brukeren få en følelse for at dette er primærfeltet, og at det andre er sekunderfeltet.
Grunnen til dette er for at man skal kunne sjekke passordet mor gitte kriterier, og gi feilmelding tilbake på dette. Dersom man gjør dette på det faktisk passordfeltet, vil Cake sjekke passordkriterier mot hashen, og ikke mot selve passordet.

Dette vil da si at alle sjekker for selve passordet bør gjøres mot feltet der man har gjentatt passordet, og ikke mot selve passordet.

Det man også må ta hensyn til, er at passordet og gjentagelsen av passordet må sjekkes opp mot hverandre. Vi kom fram til at den beste måten å gjøre dette på, var å faktisk hashe passord gjentagelsesfeltet inn i ett nytt skjult felt under kontrolleren, sende og sende denne inn i modellen.
Når dette er gjort kan man skrive selve sjekken. Den blir litt spesiell, da man må sjekke gjentagelsen av passordet mot en funksjon, som returnerer en sann eller usann verdi om passord hash1 og passord hash2 er like.

Dersom disse er like, godkjennes passordgjentagelsen.

Utlogging
Utlogging skjer ved å i kontrolleren kjøre utloggingsfunksjonen i auth komponenten, dette kan gjøres med følgende kode:
“ $this->redirect($this->Auth->logout()); “

Vær oppmerksom på at dersom man har cookies eller sesjoner[27] knyttet mot brukeren, må prosedyrer på å fjerne disse manuelt gjøres før man logger brukeren ut.

5.3.2.2 Rettigheter

"Access control list" er metoden CakePHP styrer rettigheter på. Dette må ikke forveksles med Auth. Auth tar hånd om innlogging, mens Acl tar hånd om rettighetene til den innloggede brukeren.

ACL komponenten er avhengig av 3 ekstra tabeller i databasen, aros, acos og aros_acos. Sammensetningen av disse tre tabellene er da nok til å kunne gi god rettighetshåndtering på bruker og gruppenivå.

[27] En sesjon er den informasjonen som følger brukeren hele tiden til man forlater nettsiden.

Når man oppretter en ny Acl, vil ikke brukeren klare å gå inn på noen sider som krever innlogging i hele systemet. Det vil da faktisk være så ille at brukeren ikke klarer å logge seg ut engang. Får å fikse dette kan man enten lage et skript i Cake, hvor man setter Auth modulen til å godkjenne for alle, for så selvfølgelig å fjerne denne senere, eller bruke CakeBake.

Aros - access request objects
Er en tabell som inneholder informasjon om en bruker eller en gruppe. Denne tabellen er avhengig av 6 felter for å fungere.

- ID - Oppføringens id.
- Parent_id - Oppføringens far/hvem som er over noden i hierarkiet.
- Model - Hvilken modell som er knyttet mot oppføringen(User/Group).
- Foriegn_key - En id til den faktiske brukeren eller gruppen.
- Lft & Rght - Disse to er kun for at CakePHP skal bygge trestrukturer av informasjonen.
- *Alias - Valgfri kallenavn for oppføringen.*

Noe som er viktig å få med seg er at dersom man oppdaterer aro feil, slik at man i aro er medlem av en gruppe, men i gruppefeltet i databasen er medlem av en annen. Så er det aro som gjelder, selv om på det grafiske viser at man er medlem av den andre gruppen.

Acos - access control objects
Er en tabell som inneholder alle funksjoner i applikasjonen, slik at man finne disse funksjoners riktige plass i hierarkiet.
Siden CakePHP tvinger utvikleren til å måtte lage en funksjon for hvert view, og derav en funksjon for hver side. Vil en rettighet mot en funksjon bestemme om brukeren skal kunne gå inn på siden eller ikke.

Eksempel er "controllers->Admins->group_add". Her er "controllers" roten i treet, og viser så til at vi har en kontroller som heter "Admins", og at under denne kontrolleren har vi en funksjon som heter "group_add".

Selve oppbyggingen av tabellen er lik som Aro, men med Aco trenger man ikke "model" feltet, og foreign_key feltet. Dette fordi Aco refererer til skrevede funksjoner, og ikke annen informasjon i databasen.

For å generere og sette alle Acos inn i databasen, kan man bruke skriptet som finnes under "10.2.6 An Automated tool for creating ACOs "[28] i CakePHP manualen. Bruker man dette skriptet må man da i ettertid fjerne rettighetene for funksjonene i selve skriptet.

Det finnes ikke noe slikt skript for Aros og Aros_acos. Dette er noe utviklerne selv må bestemme hvordan skal bli satt opp.

Aros_Acos
Denne tabellen knytter Aro og Aco sammen, og setter rettigheter mellom disse. Man har 4 rettigheter som må settes: les, skriv, endre, og slett.

[28] http://book.cakephp.org/view/1549/An-Automated-tool-for-creating-ACOs (mai 2010)

Disse rettighetene kan settes med 3 verdier: -1, 0, 1. Der "-1" betyr ingen tilgang, "0" bety arver hva enn som må være satt videre opp i hierarkiet, og "1" som betyr tillat.
Ellers inneholder tabellen også en unik id til oppføringen for å kunne identifisere den, samt to fremmednøkler, en til aro, og en til aco.

Når dette er på plass, kan man sette at Aro(bruker eller gruppe), skal få en gitt rettighet for valgt Aco(Funksjon)

CakeConsole
Kommandoene med CakeConsole vil da være:

"cake.exe acl create aro bruker/gruppe_navn/id foreldre_navn/id"

Med denne kommandoen kan man bygge opp Aro hierarkiet som det måtte passe for utvikleren.
Alternativer til create: delete.

"cake.exe acl create aco funksjon_navn/id foreldre_navn/id"

Denne kommandoen lar utvikleren legge inn nye funksjoner. Merk. her finnes det et automatisk skript i Cake manualen(10.2.6)[28] for å automatisere denne jobben.
Alternativer til create: delete.

"cake.exe acl grant aro_navn/id aco_navn/id all"

Denne kommandoen lar brukeren sette rettigheter mellom bruker eller gruppe og funksjoner som skal kunne besøkes.
Alternativer til grant: Deny og inherit.

Dersom man skal sette rettigheter på navn, må man ha feltet alias med i tabellen.

5.3.2.3 Sesjoner

En sesjon i CakePHP er en kapsel med informasjon som, følger brukeren mellom sidene. Denne informasjonen vil være med brukeren, helt til brukeren logger seg ut. Informasjonen som er aktuell å bruke fra denne kapselen, er brukernavn, og id. Det vil også ligge med de resterende brukerdata fra brukertabellen i databasen, men dette trenger man sjeldent å ta hensyn til som utvikler, og er der først og fremst for med som en sikkerhetsmekanisme for å identifisere at brukeren er den man utgir seg for å være.

Den mest vanlige måten å bruke sesjoner på er for å identifisere selve brukeren, slik at man kan hente ut informasjon om brukeren fra selve databasen. For å hentet ut brukerens id bruker man koden: “ $this->Session->read('Auth.User.id') ”

Det som er noe uvanlig, er at CakePHP bruker sesjoner også for å skrive og sende med feilmeldinger videre i systemet. For å skrive en feilmelding, kan man gjøre dette ved hjelp av: “ $this->Session->setFlash('...') “ og for å vise meldingen: ” $this->Session->flash() ”.

Når man jobber med sesjoner, kan man også legge med egen informasjon inn i sesjonen. I dette prosjektet har vi lagt inn en identifikasjon av rollen til brukeren, dette gjelder kun for å få det grafiske i systemet til å se riktig ut, og har ingen ting med hva brukeren har rettigheter til å gjøre.

For å skrive egen informasjon inn i en sesjon, bruker man funksjonen:
” $this->Session->write(…) ”
for å lese og hente ut informasjon igjen bruker man funksjonen:
” $this->Session->read(…) ”
og dersom man ønsker å slette informasjonen, kan man gjøre dette med:
” $this->Session->delete(…) ”

5.3.2.4 Cookies

Dersom man ønsker å kunne lagre data lokalt hos brukeren, slik at systemet skal kunne hente ut dette senere, gjøres dette ved hjelp av cookies.
Cookies er en liten informasjon kapsel som inneholder gitte innstillinger.

I applikasjonen våres kan man huke av for ”husk meg” ved innlogging, og en cookie med brukernavn, og passordhashen vil bli lagret lokalt på maskinen.
Når brukeren så går inn i systemet igjen, vil systemet se etter om denne cookien finnes som det første den gjør. Dersom cookie finnes, og informasjonen stemmer, vil brukeren bli logget inn.

Problemet med cookies er at det oppstår et lite sikkerhetsspørsmål. Det første man må ta hensyn til, er at dersom man er på en offentlig maskin, vil andre bli automatisk logget inn som brukeren. Dette er da en uheldig situasjon, og er derfor man alltid bør logge seg ut når man forlater en datamaskin.

En cookie lagret på en datamaskin, kan bli hentet ut og brukes av andre brukere senere. For å forhindre dette har Cake noen sikkerhetsalternativer.
En cookie blir automatisk ugyldig etter 5 dager, dersom ikke noe annet er definert. Det er også mulig å legge inn en krypteringsnøkkel, slik at det blir noe vanskeligere å hente ut informasjonen. Det er også mulig å få CakePHP til å kreve at all trafikk må gå gjennom en sikker https tilkobling. Begge de sistnevnte er skrudd av som standard, og må derfor legges inn manuelt i koden.

Det man bør ha i bakhodet når man bruker cookies, er at dersom man har flere nettlesere, må man ta hensyn til at hver av disse vil lagre sine egne cookies, og at man da må logge seg ut fra alle nettleserne.

I dette prosjektet bruker vi også cookies for å lagre mer enn bare innloggingsinformasjon. Vi har nemlig lagt inn en mekanisme for å lagre språkinnstillinger i cookies. Dette gjør derfor at hvis man velger at applikasjonen skal vise sidene på engelsk, vil denne innstillingen også være der neste gang man logger seg inn. Denne cookien er derimot kun lagret i 20 dager, så dersom man bruker lengre tid enn dette mellom hver innlogging, er man nødt til å ” trykke på engelskflagget” på nytt.

5.3.3 Helpers

5.3.3.1 Skjema

Skjemahjelperen brukes i CakePHP for enkelt å kunne bygge opp skjemaer for utfylling og endring av informasjon. Før man starter å lage et skjema må man sette ett hode for skjemaet ved ” <?php echo $form->create('…'); ?> ”, og man bør også avslutte skjemaet med ” <?php echo $form ->end();?> ”.

Et skjema vil da også lage en ”Send inn” knapp, dersom dette ikke er definert. Vær oppmerksom på at dersom man ikke definerer dette selv, og ikke avslutter skjemaet på riktig måte, vil man kunne ende opp med noen enkle grafiske feil.

Automagiske elementer
Med automagiske skjema elementer mener CakePHP et element som i stor grad selv finner ut hva det skal brukes til, og konfigurerer seg deretter. Dette fungerer i en viss grad, og gjør at det alltid blir morsomt å se hva førsteutkastet til koden blir.

Automagien baserer seg på å tolke navn og typer i systemet, og bygger opp skjemaene etter dette. Dette vil si at har man et felt i en tabell som heter ”pasword”, vil dette feltet i skjemaet automatisk bli et inntastningsfelt, og at det man skriver skjules bak ”*” tegn. Et annet eksempel blir dersom man har en dato, vil denne automatisk bli en liste hvor man kan velge dag i måned, måned og år.

Det vil alltid være mulig å overskrive disse automatiske innstillingene om man mener de burde være annerledes.
Automagiske elementer er ellers veldig logisk oppbygd, og er de elementene som er enklest å gjøre endringer mot.

Manuelle elementer
Manuelle elementer er noe enklere oppbygd, og gjør kun det de er ment for å gjøre. Dette gjør da at man har liten frihet til endringer, og at man må lese nøye på dokumentasjonen for hvert element. En komplett liste over disse elementene, og hvordan de brukes kan man finne i CakePHP sin dokumentasjon under ” 7.3.5 Form Element-Specific Methods ”[29].

5.3.3.2 HTML

Tabeller
I CakePHP kan man om ønskelig bruke en hjelper for å lage enklere tabeller. Denne er veldig fin og veldig ryddig og bruke.
Det finnes to typer tabellhjelpere, en som brukes for å generere cellenavn, og for å generere selve innholdet i tabellen. Dersom man bruker hjelperen til å generere innholdet i tabellen må man derimot være oppmerksom på en ting. Dersom man skal sette egne innstillinger på enkeltceller i radene, bør man unngå å bruke denne hjelperen.
Det lar seg gjøre å sette disse innstillingene i selve hjelperen, men man vil da ende opp med en mer komplisert og rotete kode enn om man hadde skrevet koden på vanlig måte.

[29] http://book.cakephp.org/view/1413/Form-Element-Specific-Methods (mai 2010)

I dette prosjektet har vi derfor valgt i noen tilfeller å bruke hjelperen, mens i andre har vi valgt å gjøre det på den vanlige måte.

Vi hadde også tidligere i prosjektet der vi fant ut at tabellhjelperen var veldig grei å bruke for å få finere tabeller. Dette fordi man veldig enkelt kunne sette egne stilsett mot par og odde rader av tabellen. Problemet med dette er imidlertid at siden dette genereres på PHP nivå, er også dette veldig sårbart mot endringer gjort av javascript.
Derfor burde man i de tilfelle hvor man fjerner, legger til, skjuler eller viser rader dynamisk, også gjøre stilendringene ved hjelp av javascript. Dersom dette blir tilfellet, vil ikke lengre denne hjelperen ha noe å si på par og odde stiller.

Lenker
Hjelper for generering av lenker er den html hjelperen vi har brukt mest i systemet. Denne brukes for å generere lenker til sider som ligger under applikasjonen i Cake. Grunnen til at man trenger en egen metode for å generere disse lenkene, er at disse sidene ikke nødvendigvis ligger på den logiske plasseringen under websiden, men under virtuelle plasseringer generert av Cake.

En stor fordel med også å bruke hjelpere, er dersom man gjør noen endringer i standardinnstillingene i Cake. I dette tilfellet kan man risikere at alle adresser under applikasjonen blir endret. Dette er da noe denne hjelperen tar hensyn til, og derav kompenserer videre for.

I CakePHP har man også hjelpere som er bygget opp på så å si samme måte som lenke-hjelperen, men med noe annet formål. Eksempler på dette er image-hjelperen for å hente ut bilder, css-hjelper for å hente ut stilsett for applikasjon, og script-hjelper for å hente ut javascript filer.

Alle disse hjelperne er til for å hente ut filer lokalt fra serveren, uten at utvikleren skal behøve å måtte bry seg om de faktiske plasseringene til filene.

Paginator helper
Paginator hjelperen må ikke forveksles med Paginator komponenten i CakePHP. Hjelperen er til som et tillegg til komponenten, og er kun for å få det grafiske til å se riktig ut.
Ved å bruke hjelperen til paginator, kan man vise informasjon som ”side x av y” og ”treff fra X til Y”.
Paginator hjelperen vil også hjelpe utvikleren å generere passende knapper, slik at man enkelt kan hoppe mellom de forskjellige siden. Eksempel på disse knappene er ” 1|2|3|4 | …”, og typiske ”første”, ”forrige”, ”neste”, og ”siste” knapper.

5.4 *Problemer med CakePHP*

5.4.1 Problemer med paginator:

5.4.1.1 Ytelse

Det største problemet med paginator er at dersom man har flere tabeller som knyttes sammen, vil Cake automatisk gjøre dette ved hjelp av SQL spørringen Join[30]. Problemet med at Cake bruker Join er at ytelsen blir kraftig redusert.

Vi testet i dette prosjektet med ca 3500 testbrukere. Ved dette tilfellet brukte Cake rundt 14 sekunder for å vise kun 25 brukere i våres tilfelle. Dette vil også da bli tilfellet for hver side som blir vist, da treffene ikke blir mellomlagret. Dette er langt tregere enn hva som er akseptabelt i våres tilfelle, så her måtte vi gjøre en del endringer.

Vi så først på hvor vi kunne spare inn på tiden her, vi så at det ble gjort 2 spørringer, en først for å telle antallet brukere, og en for å hente informasjonen om disse.

Vi gikk derfor inn og manuelt inn og skrev om hele prosedyren for telling ved å tvinge Cake til å bruke manuelle SQL spørringer istedenfor. Ved å gjøre dette fikk vi telleprosedyren ned i noen få millisekunder. Problemet var derimot at vi informasjonsspørringen fortsatt tok over 7 sekunder.

Siden en spørring på 7 sekunder fortsatt var alt for høyt, og at denne tiden økte tilnærmet et eksponentielt mønster etter jo flere brukere vi la inn, var heller ikke dette en løsning vi var fornøyd med.
Et alternativ vi kunne bruke da var også å skrive om denne spørringen. Problemet med dette var at vi da hadde havnet med veldig mye kode, og at hele paginator komponenten ville mistet sin funksjon, da vi likeså godt kunne skrevet dette manuelt fra starten av.

Den endelige løsningen ble derfor å tvinge paginatoren til å ikke utføre SQL Join, og heller la Cake bygge opp SQL spørringene etter hvert som de trengtes. Med å gjøre dette kunne vi fortsatt bruke paginator-helperen for å generere dynamisk sideinnhold.

[30] Join knytter sammen tabeller som har tilhørlighet med hverandre.

5.4.1.2 Fjerning av databasesammenkoblinger

For å fjerne sammensetning av tabeller fra CakePHP sin spørring, kan man bruke to forskjellige metoder. Den ene metoden er å endre innstillingene for rekursjon i Cake. Her har man mulighet for å fjerne alle form for sammensetning, eller eventuelt skru på automatisk sammensetning av kun de nærmeste koblingene mellom tabeller.

Vi brukte derimot metode nummer to, som gikk ut på å sette koblingen manuelt. Dette gjorde vi for å få mer frihet senere til å kunne bruke opp igjen deler av koden i ettertid om ønskelig.
Siden vi her ikke skulle ha noen koblinger i det hele tatt, ville resultatet av de to metodene bli akkurat det samme.

For å endre koblinger manuelt, må man først legge til en såkalt "Behaviors" før paginato funksjonaliteten. En "Behaviors" er en metode i Cake for å overskrive de standard måtene Cake oppfører seg på. Under "Behaviors" må man så legge til "Containable" for å si til Cake at man manuelt ønsker å bestemme koblingene i spørringen.
I våres tilfelle ble da den komplette koden for dette:
"$this->User->Behaviors->attach('Containable');"
Eventuelt alternativ 1: *"$this->User->recursive = -1;"*

Når "Containable" er satt, er det bare å legge til "'contain' => false," i paginatoren for å si at det her ikke skal være noen koblinger. Her kan man da selvfølgelig definere egne koblinger om dette heller er ønskelig.

5.4.1.3 Få funksjonaliteten tilbake

Når vi hadde gjort dette, var ytelsen nede på 1-2 millisekunder, dette var mer enn godt nok for dette prosjektet. Problemet var i midlertidig det at siden vi nå hadde fjernet alle koblinger, så hadde vi også fjernet all mulighet for å korte ned listen på informasjon som lå i andre tabeller. Vi kunne foreksempel nå ikke søke på fornavn og etternavn i "alumnis" tabellen, da Cake var avhengig av å gå inn i "alumnis" gjennom en sammenkobling basert på fremmednøkkelen i "users" tabellen.

Dette er derimot ganske enkelt å ordne, da man enkelt kan gå rett inn i "alumnis" tabellen dersom den er definert som en del av matrisen " $uses" som finnes øverst i kontrolleren. Dersom man skal legge til egne tabeller i kontrolleren må man derimot være nøye på å også definere at det også skal tas med standardtabellen til kontrolleren i matrisen.

Når alt dette er på plass kunne vi så gjøre søk på "alumnis" og "admins" tabellene, hvor vi hentet ut en liste med fremmednøklene til "users" tabellene. Denne listen la vi så til som en parameter på selve paginator søket, der paginatoren kun skulle hente ut brukere som lå i denne listen.

Ytelsen var nå opprettholdt og resultatet var riktig. Problemet var da imidlertid at resultatet kun inneholdt informasjon fra tabellen "users", da det var denne som ble knyttet mot i pagintatoren.
Ved å utnytte måten CakePHP henter ut informasjonen sin og viser denne, kunne vi nå bare bruke resultatet fra paginatoren som da kun inneholder noen få brukere til å hente ut

resterende informasjon fra "admins" og "alumnis" tabellene. Dette kunne da gjøres ved å hente ut brukere fra disse tabellen som hadde fremmednøkkel lik primærnøkkelen i resultatet fra paginatornen.
Denne informasjonen vi så fikk ut, kunne vi så bare legge til i matrisen på til paginator resultatet, for å så sende alt videre inn i "viewet".

5.4.1.4 Resultatet

Ved å gjøre alt dette ble resultatet nøyaktig det samme som før, mens ytelsen ble kortet ned fra flere sekunder til noen få millisekunder.
Grunnen til at ytelsen ble såpass forbedret, var at databasen koblet sammen alle de flere tusen brukerne vi hadde, for så å gjøre alle opperasjoner videre på disse, for så å kun vise noen få av brukerne.

Det vi derimot gjorde, var å korte ned antallet med brukere til de riktige, for så å bygge opp igjen informasjonen med falske koblinger på disse. Dette kunne gjøres en del enklere enn den metoden vi brukte, men vi ville da ikke hatt mulighet for å holde det dynamiske i Cake rammeverket på plass, slik at de fordelene Cake har å tilby kunne brukes på en riktig og god måte i ettertid.

Dersom vi derimot ikke hadde gjort det på denne måten, ville man fått en del problemer dersom man skulle lagt til ekstra funksjonalitet i ettertid, da man i det tilfellet ikke kunne bruke CakePHP sine metoder, men måtte laget sine egne.

En alternativ løsning på problemet, men som er uakseptabelt i dette prosjektet, er å istedenfor å bruke flere små tabeller, bruke en stor som inneholder all informasjon. Med å gjøre dette vil ikke databasen behøve å gjøre SQL-Joins, og resultatet blir økt ytelse, og enklere å kode mot.

Problemet med denne løsning blir at databasen fort blir oversiktelig, og en del vanskeligere å jobbe opp mot. Det vil også bli noe mer å tenke på rundt sikkerheten, da det vil være knyttet tomme*(forhåpentligvis)* administrator relaterte felter også under alle ikke-administratorer.

5.4.2 Tegnsett problemer

5.4.2.1 Tegnsett

Når vi startet dette prosjektet, så vi på to aktuelle tegnsett[31]. UTF-8 og ANSI(Windows-1252). Vi brukte ANSI, både da denne er mer Windows vennelig, og fordi man enklere kan gå fra ANSI til UTF-8 i ettertid, enn UTF-8 til ANSI.
Dette er fordi UTF-8 er bedre støttet, og ANSI i trenger noe mer spesialtilpassing for å fungere optimalt.

5.4.2.2 Konfigurasjon

Uansett hvilket tegnsett man måtte velge, er det viktig å prøve å holde seg til ett å samme tegnsett hele veien. Dersom man henter ut informasjon fra eksterne sider, er det best å sjekke at denne data har samme tegnsett som applikasjonen.

[31] En bestemt koding over hvordan et tegn skal være oppbygd, samt hvile tegns som så er støttet.

Når man har valgt et tegnsett, er det derfor viktig å sjekke følgende som kan ha innvirkning på hva sluttresultatet blir:

1. Er databasen satt til å bruke dette tegnsettet, og er dette tegnsettet definert i CakePHP sin database konfigurasjon.
2. Har man definert riktig tegnsett i html hodet til layout filen.
3. Er selve koden i riktig tegnsett(om tekstfilene er formatert riktig).
4. Har man satt riktig tegnsett i CakePHP applikasjonens konfigurasjon.

Dersom man passer på å holde kravene over, vil man sjeldent få problemer. Noe vil derimot kunne oppstå, og er dette vi har satt fokus på videre under problemer.

5.4.2.3 Problemer

Ascii tabellen

Når vi lagte liste og søkefunksjonalitet basert på fornavn og etternavn, kom vi over et problem når det gjaldt spesialtegnene æ, ø, og å. Disse tegnene er vanlige i det norske språk, men har dessverre alt for dårlig støtte.
Når vi la inn funksjonalitet for å ta hånd om dette, måtte lage sammenligninger og oppslag mot ASCII[32] tabellen.

Å ta hensyn til spesialtegn er i og seg selv ikke så vanskelig. Problemet kom imidlertid når vi skulle gjøre database søk, da vi brukte en metode kalt REGEXP[33]. Måten vi brukte denne metoden på var å gjøre et søk som inneholdt strengen "REGEXP '^[x-y]'". Denne sier da at vi skal hente ut alle av en valgt verdi, som starter på bokstaven x, og fram til og med bokstaven y. Denne metoden er da helt fin og bruke fram til man kommer til spesialtegnene, og man må bruke noen ekstra linjer kode for å behandle dette.

Det første man ved spesialtegn må ta hensyn til, er at ASCII alfabetet ikke har æ, ø, og å rett etter z, dermed må man ta et hopp over en del andre tegn før man kommer hit. Et annet problem som man fort finner, er at spesialtegnene heller ikke kommer i samme rekkefølge som vanlig. Den riktige rekkefølgen er "å, æ, og ø", der "å" har lavest ASCII verdi, og "ø" har høyest. Man må også her merke seg at disse tegnene har andre spesialtegn mellom seg, og derfor ikke bør brukes med en fra og til "-" metode i SQL spørringen.

Riktig spørring for første bokstav filtrering på hele alfabetet vil da bli:
" etternavn REGEXP '^[a-z]' OR etternavn REGEXP '^[åæø]' ".
'^' indikerer at det kun skal filtreres på første bokstav. "[" og "]" definerer hva som skal søkes på, og dersom man setter ett "-" tegn i mellom, får man en fra og til verdi.

Omkoding av tegnsett

Når vi i prosjektet har jobbet med parametere som blir sendt mellom sider, har vi kommet over det problemet at dersom en parameter inneholder spesialtegn, vil teksten i denne parameteren være kodet under UTF-8.
PHP har her innebygde funksjoner for å kode mellom tegnsettene, "utf8_decode()", og "utf8_encode()".

[32] En tabell over alle tegn i tegnsettet på datamaskinen.
[33] REGEXP(Regular expression), en funksjon for å definere regler på hva svaret må inneholde i en SQL spørring.

Vi brukte her dekodings funksjonen "utf8_decode()" for å få ANSI kode ut av UTF-8. Det vi fant ut fort, var at denne funksjonen manglet funksjonalitet for å faktisk sjekke at innholdet var kodet i UTF-8, noe som endte med at enkelte ting som allerede var kodet til ANSI fra før, faktisk ble kodet på nytt. Problemet med dette var da at denne teksten ble korrupt og derav ubrukelig videre.

Løsningen på dette ble at vi måtte legge inn en egen sjekk for å passe på at teksten kun ble kodet om dersom den faktisk var UTF-8.

Sjekken vi da endte opp med var:

```
foreach ($this->passedArgs as $key => $passed){
    if(mb_detect_encoding($passed) == 'UTF-8')
        $this->passedArgs[$key] = utf8_decode($passed);
}
```

Figur 15

Denne sjekken går da gjennom alle parametere tilsendt, sjekker om disse er kodet i UTF-8, og koder disse om til ANSI dersom dette er tilfellet.

Ajax og tegnsett

Dersom man henter ut informasjon ved hjelp av Ajax, så vil man ikke ha noe mulighet for å definere kodingen i noen form for HTML hode. Man vil også ha begrenset mulighet for å kode om tegnsettet som forklart før, da JavaScriptet også vil få en innvirkning på resultatet.

Vi prøvde her mange forskjellige metoder for å få dette til å fungere, og den som virka mest lovende var å sette tegnsett innstillinger i jQuery sin konfigurasjon. Vi fulgte her jQuery sin offisielle dokumentasjon, men det så ikke ut til å ha noen innvirkning.

Metoden vi til slutt valgte å brukte, var å lage et CakePHP element som vi inkluderte inn i starten av alle Ajax skriptene. Dette elementet inneholdt kun PHP koden: "header('Content-Type: text/html; charset=windows-1252');"

Dette ble da den eneste metoden som faktisk fungerte, og denne har vi da ikke hatt noen problemer med siden.

5.4.3 Problemer rundt Rettighetsbehandling

5.4.3.1 Hvorfor problemer

Mange av problemene kommer fordi vi har valgt å bruke en metode i CakePHP som heter "ActAs" som knyttes mot rettighetsmodulen. Denne metoden gjør det enklere for utvikleren å jobbe mot applikasjonen, da alt av rettigheter skal gå automatisk. Fordelen blir mye mindre kode, mens problemet blir at man må lage spesialrutiner ved oppretting, oppdatering, og sletting av brukere.

Problemet ved å bruke "ActAs", er at dersom man oppretter eller oppdaterer, vil man få tomme felter i databasen der man før skulle ha nøkler for arv og fremmednøkler. Løsningen

på dette er å bruke en kodesnutt som man kan finne under *"10.2.4 Acts As a Requester"*[34] i CakePHP manualen. Denne koden bruker man da som den er, og det den gjør er kun å sende informasjonen videre inn i rettighetsbehandlingen.

Denne koden vil også sannsynligvis bli integrert rett inn i CakePHP rammeverket ved et senere tidspunkt, da det er veldig sjeldent at man gjør noen endringer på denne. Vi har imidlertid gjort en liten endring, men det er kun snakk om en duplisering av koden med minimale endringer for å kunne sende inn den gamle informasjonen inn i rettighetsbehandleren, dersom ingen ny informasjon er satt.

5.4.3.2 Navnsetting

Siden funksjonen over fra Cake sin dokumentasjon, kun oppdaterer det aller mest nødvendige som arv, og fremmednøkler, vil ikke feltet "alias" bli oppdatert. Dette feltet inneholder da et kallenavn for brukeren eller gruppen, og har ingen betydning annet enn for å få bedre oversikt over hva som er knyttet til hva.

Vi ønsket å få oppdatert dette feltet også ved endring. Dette kunne derimot ikke gjøres som en del av funksjonen over(*10.2.4*)[34], men måtte gjøres manuelt etter for hver gang.
For å gjøre dette måtte vi da først hente ut en datakapsel basert på riktig rettighetsoppføring av brukeren eller gruppen. Dette gjorde vi med å søke etter fremmednøkler som passet med brukerens eller gruppens id.

Koden for å gjøre dette(der x er enten "User" eller "Group"):

```
$data = $this->Acl->Aro->findByForeignKey($this->x->id);
```

Her brukte vi først en kode som hentet ut brukeren/gruppen basert på navnet, dette funger greit på brukere, men siden man skal kunne ha flere grupper i systemet som heter det samme, vil man her få et problem.

I neste omgang måtte vi oppdatere kallenavnet i denne datakapselen:

```
$data['Aro']['alias'] = 'nytt_navn';
```

Og i siste omgang, må man sette denne datakapselen tilbake som en rettighetsoppføring.

```
$this->Acl->Aro->save($data);
```

Med disse tre linjene kan vil vi kunne holde kallenavnet oppdatert. Dette er ikke nødvendig, men en stor fordel for utviklere som vil gå inn i databasen manuelt i ettertid.

5.4.3.3 Sletting

Dersom man skal slette kun en bruker, vil brukeren og alle rettigheter til denne brukeren slettes samtidig. Dersom man derimot skal slette flere brukere i CakePHP, må dette gjøres ved hjelp av "DeleteAll" funksjonen til Cake. Denne funksjonen har i motsetning til vanlig "Delete", ikke satt på mulighet for å slette rettighetene til brukeren som standard. Dette resulterer til at ytelsen ved sletting blir mye høyre, men man får mange døde rettigheter i databasen som bare står og gjør systemet tregere i selve bruk.

[34] http://book.cakephp.org/view/1547/Acts-As-a-Requester (mai 2010)

En standard "DeleteAll" funksjon er bygd opp på følgende måte:
" deleteAll(mixed $conditions, $cascade = true, $callbacks = false) "

Her vil $conditions være en matrise som definerer reglene ved slettingen, $cascade definer om oppføringer i tabeller knyttet sammen med brukeren gjennom fremmednøkler skal slettes, og siste $callbacks må da settes til "true" for at også rettheter skal slettes.

Riktig funksjon blir derfor:
" deleteAll(array('condition' => array(...)), true, true) "

Dette er som nevnt tidligere veldig tregt, og sletting av noen tusen brukere tar noen minutter, i motsetning til sekunder som den gjør uten "callbacks". Det som derimot må sies, er at dette er en funksjon man bruker veldig sjeldent, og at det derfor er bedre å ha lavere ytelse her, og holde databasen ren for "død" informasjon.
Siden denne funksjonen kan ta noen minutter er det også viktig å legge med en ekstra liten konfigurasjon med denne. Problemet er nemlig det at PHP har automatisk stopper skriptet om det bruker for lang tid. Dette er en konfigurasjon mot PHP, og ikke mot CakePHP.

Konfigurasjonen gjør med linjen:

set_time_limit(0);

5.4.4 Overgang fra CakePHP 1.3 RC til Stable.

Siden den stabile versjonen av CakePHP ikke kom ut før mot slutten av koden, måtte vi bruke "release candidates" nummer 1 og 2 av rammeverket under utviklingen. Vi har derimot testet med den stabile versjonen av programvaren, og fant ut at det var noe problemer rundt rettighetene. Problemet var at man aldri fikk tilgang til å gå inn på funksjoner som lå under andre funksjoner som underfunksjoner.

Dette kunne enkelt løses ved å legge alle funksjonene rett under "Admins", "Alumnis", "Brukere", etc.
Dette er da en midlertidig løsning, men den fungerer helt fint. Problemet er i midlertidig at det vil være noe mer jobb å administrere, og at det vil se noe mer rotete ut. Dette er også mest sannsynligvis noe som er noe enklere å skrive om i ettertid. Disse problemene var så vidt nevnt i CakePHP sin dokumentasjon, og det var lagt opp i denne dokumentasjonen for at mer skal komme i senere tid.

Funksjoner berørt av dette er:

- select_user_ajax_*
- select_company_ajax_*
- endre_arbeid
- ajax_user_create_company

(indikerer at det finnes flere funksjoner med samme start på navnet, og regler gjelder for dem alle)*

Disse funksjonene mener vi burde vært satt som underfunksjoner av andre funksjoner, slik at man enkelt skulle kunne sette samme rettigheter på dem alle i en og samme opperasjon.

5.5 *jQuery*

5.5.1 Hvordan bruke jQuery

Det første man må gjøre før man kan bruke jQuery, er å inkludere en av javascript filene som ligger i jQuery pakken. Det finnes her to filer, en ”Production”, og en ”Development”. Development versjonen lar deg få innblikk i hvordan javascriptene fungerer. Produksjonsversjonen er derimot pakket ned til kun å inneholde det datamaskinen trenger for å tolke pakken, samt en hodedel som inneholder navn, versjon og lisens informasjon. For vårt prosjekt var det best å bruke produksjonsfilen da denne versjonen er raskere og vi ikke har behov for innblikk i hvordan jQuery kjernen fungerer.

For å bruke jQuery, bygger man alt opp som jQuery funksjoner inne i en ”document ready” funksjon: ” $().ready(function() { … }); ”

Der hvor det her er ”…” vil man kunne skrive jQuery koden. Dersom man skal lage rene JavaScript funksjoner, kan også disse inneholde jQuery kode, men disse funksjonene må skrives utenfor ”document ready” funksjonen.

En typisk jQuery kode kan derfor være:
”
```
$().ready(function() {          //Dokumentet er ferdig lastet, kode kjøres.
        $('#id').hide();        //Skjuler elementet med id #id.
        $('#id2').show();       //Viser det skjulte elementet #id2.
});
```
”
Som man kan se på eksemplet over, er jQuery ganske så enkelt å forstå med noe øving. Selv om jQuery rammeverket er såpass kompakt som der er, har det funksjonalitet for nesten alle tenkelige javascript funksjonalitet.

5.5.2 jQuery og CakePHP

5.5.2.1 Kode med manuelle skript

Siden det i CakePHP er mest aktuelt å skrive koden i selve viewet, og dette er et av de siste leddene i prosessen med å generere siden for selve brukeren, er man nødt til å legge til noen ekstra linjer med kode for å be Cake om å ta hensyn til dette.

Den enkleste måten å gjøre dette på, er å skrive koden mellom to linjer med kode:
” $html->scriptStart(array('inline' => false)); ” og “ $html->scriptEnd(); ”

Siden vi setter “inline” til å være falskt, vil koden havne på sin riktige plass i hodet på den genererte html siden.

5.5.2.2 Kode ved hjelp av JS hjelperen

For å bruke JS hjelperen må, man først koble seg mot et element ved hjelp av koden:
” $js->get('#id'); ”

Etter man har koblet seg opp mot et element, kan man for eksempel legge til en hendelse mot det valgte elementet.
Et eksempel på dette for å skrive ut teksten ”hello world!” på skjermen vil være:
” $js->event('click', "alert('hello world!');"); ”

For at alt skal fungere, er man nødt til å “kompilere” koden inn i hodet. For å få dette til må man ha en avsluttende linje med:
” $js->writeBuffer(array('inline' => false)); ”

5.6 *Integrering mot Facebook og LinkedIn*

I designet av systemet var det lagt opp til implementering av en modul som koblet systemet opp mot Facebook som importerte kontaktdata fra Facebook. En plugin til et eksternt sosial nettverk fungerer slik at brukeren kan gjennom alumnisystemet logge seg inn på sin bruker i det eksterne sosiale nettverket. Alumnisystemet vil da kunne hente ut informasjon fra brukerens profil, samt kunne publisere informasjon på brukerens profil.

Når dette skulle implementeres kom vi dog fram til at en modul mot Facebook ikke ville være hensiktsmessig grunnet kvaliteten på informasjonen som ligger der. Brukere vil ofte legge ut uriktig informasjon på Facebook som spøker mellom venner eller for og ikke litt bli gjenkjent av personer de ikke ønsker å bli gjenkjent av. Mange brukere oppdaterer og kontaktinformasjonen sin på Facebook meget sjeldent slik at den blir utdatert.

Ved videre utvikling av prosjektet ser vi mulighet for å lage tillegg mot LinkedIn. Ettersom LinkedIn er et sosialt nettverk med veldig profesjonelt og seriøst preg har kontakt- og jobbinformasjonen her høy kvalitet.

6 Testing

Testing av systemet har vært en integrert del av utviklingsprosjektet. Hvor ny funksjonalitet har blitt kontinuerlig enhetstestet, og hvor vi har prøvd og fått i gang brukertester der det var ønskelig.

Enhetstester

Ved utvikling av PHP har vi brukt NetBeans IDE. Dette programmet har fra versjon 6.9M1 og utover også støtte for CakePHP. Ved å bruke NetBeans vil vi få opp feilmelding direkte på de fleste syntaktiske feil, men ikke på de logiske.
Innebygget i CakePHP er det derimot en egen feilsøkingsmekanisme, som har 3 nivåer. Ved nivå 0, er all form for feilmelding skrudd av. Ved nivå 1, gis det advarsler og feilmeldinger ved feil. Det siste nivået nummer 2, viser også de faktiske spørringene mot databasen som blir gjort bak kulissene. Denne informasjonen viser også noe begrenset feilrapportering av databasespørringene, tiden spørringen tok, og eventuelt hvor mange oppføringer i databasen som ble besøkt under spørringen.
Vi har også brukt CakePHP sin loggings funksjon. Loggfilen ”error.log” kan man finne under applikasjonens ”./tmp/logs/” mapper. Loggingsfunksjonen har først og fremst blitt brukt ved problemer rundt rettigheter. I loggfilen står det da brukerinformasjon, samt hvilken del av applikasjonen som er forsøkt besøkt uten hell.
Ved feilsøking av JavaScript, har vi laget våres egne enkle prosedyrer som skriver ut informasjonen på skjermen. Vi har også lagt inn en melding som skrives ut i starten av scriptet ved utvikling, og på denne måten vil meldingen ikke vises ved syntaktisk feil.

Ekstern testing

I utviklingen av systemet har vi fått bistand fra Rachel Gibb. Hun har mangeårig erfaring fra å jobbe med alumnisystemer ved Universitetet i Otago New Zealand. Hun har fungert som en konsulent for oss som har utført tester av administratorsiden av systemet og gitt råd fra ståstedet til en erfaren bruker.

Intern testing

Personell ved IMT-avdelingen til HIG har blitt gitt tilgang til gjeldende versjon av systemet for testing av den vanlige brukersiden. Noen av disse testerne er alumni ved skoler som allerede har alumnisystemer, dette gjør at vi har testere med og uten erfaring i slike systemer fra før. Disse testerne er også undervisere ved tekniske informatikk studier og regnet som kompetente brukere. De egner seg derfor for en tidlig testfase der det kan være mange tekniske problemer, men ved senere testing bør også testere med mindre teknologisk ekspertise brukes.

Videre testing

Ved videre utvikling av prosjektet vil testing med en større brukermasse være nødvendig. Løsningen vi vil foreslå for dette er å sette opp et prøveprosjekt ved IMT på HIG der studenter ved IMT oppfordres til å bruke systemet. Dette vil gi en stresstesting med mange bruker og gi et vidt spekter av testere som har ulike forutsetninger, kunnskaper og forventninger.
Videre testing kan så utvides til hele HIG for enda større belastning på systemet, samt en brukermasse som er har mindre forventet teknisk kompetanse en IMT-studentene.

7 Presentasjon av prosjektet for Høgskolestyret ved HIG

Det ble holdt en presentasjon av prosjektet for Høgskolestyret ved HIG den 22. april 2010, se veldegg for foiler fra denne presentasjonen. Presentasjonen tok for seg hva et ferdig alumnisystem kunne tilby skolen, nåværende status på Alumni Communication System og hva som krevde videre utvikling, samt alternativer skolen hadde for videre utvikling og ferdigstilling av systemet etter dette prosjektet.

Høgskolestyret var positiv til prosjektet og det ble fra mange i styret uttrykt at dette var et system som høgskolen lenge hadde hatt et behov for. De var og positive til løsningene som systemet presenterte. Styret var dog bekymret for videre utvikling og drifting av systemet grunnet tidligere erfaringer med programvare og systemer utviklet av studenter. Slike systemer blir ofte ikke vedlikeholdt eller videreutviklet etter studentene som utviklet det er ferdig med sin prosjektoppgave.

8 Avslutning

Diskusjon av resultater

Prosjektets mål var å utvikle en tidlig versjon av et alumnisystem som skulle fungere både som en demonstrator for systemets potensial og en plattform for å bygge videre på for utvikling av et ferdig system.

Systemet vi har bygd opp er laget fullt modulært, veldokumentert og robust. I tilegg har vi lagt mye arbeid i beskrive å forklare programvaren som er brukt for å gjøre det enklere for de som skulle videreutvikle systemet. Som sådan mener vi at vi har lagd et system som egner seg veldig godt for videre utvikling.

Ganske mange av funksjonene vi hadde tenkt å implementere har vi ikke fått implementert. Dette kommer av omfanget til programmet. Vi opplevde det som litt frustrerende å jobbe på et prosjekt som var så omfatende at vi ikke kunne bli ferdig, men kun skulle lage en del av systemet som det kan utvikles videre på.

Noe som gjorde at vi fikk mindre tid til implementeringen i forhold til hva vi hadde planlagt var at det gikk veldig mye tid i å sette seg inn i verktøy og program, da særlig CakePHP. CakePHP hadde en del mangler hva gjelder dokumentasjon grunnet at programmet var i en større oppdateringsprosess og all dokumentasjon var skrevet for forrige versjon. Vi måte dermed prøve oss mye fram og studere kildekoden til Cake ved bruk av den nye versjonen.

I lys av dette kunne vi ved valg av en eldre versjon av CakePHP eller ved å bruke enklere programvare ha spart mye tid på læringsprosessen og fått mer tid til implementering av funksjoner. Dette hadde dog gått på bekostning av modulariteten og robustheten som vi søkte og ville oppnå, samtidig som den eldre versjonen av CakePHP skulle fases ut og ville miste funksjonalitet. Vi mener derfor at vårt valg av nyeste versjon av CakePHP var riktig.

Videreutvikling

Prosjektet er bygd opp fra bunnen av for at andre skal kunne videreutvikle på det i ettertid, da dette var et krav fra skolen. Det vil derfor være ideelt å lage en ny bacheloroppgave ut i fra dette prosjektet. De som skal utvikle prosjektet videre står veldig fritt i hvilke funksjoner de vil implementere på grunn at prosjektet er modulbasert og benytter et MVC basert rammeverk.

Evaluering av gruppens arbeid

Petter og Oddbjørn kjente hverandre godt fra før, mens Lasse kjente hverken Petter eller Oddbjørn. Denne gruppen har allikevel fungert veldig godt, og vi har alltid klart å komme til en enighet uten behov for avstemninger eller annen overstyring.

Vi har jobbet veldig mye sammen på grupperommet, og derav hatt en god dialog hele veien underveis. Arbeidsoppgaver har blitt godt fordelt, og alle i gruppen har hjulpet gruppen videre med sin kompetanse på en god og effektiv måte.

Når det gjelder prosjekt som arbeidsform, har vi derfor vært veldig heldig, og har ikke noe negativt å si om det.

Konklusjon
Det har vært interessant og endelig få jobbe med en reell prosjektoppgave. Dette har også vært en høy motivasjonsfaktor for alle og har gjort at vi alle har tatt i et ekstra tak. Hver enkelt på gruppen har også lært mye i løpet av prosjektperioden, da vi har jobbet med et MVC basert rammeverk, og derav måtte sette oss inn i en tankegang som har vært helt ny for oss alle.

Det har vært lærerikt og havne i den situasjonen hvor vi har måttet endre hele tankegangen grunnet rammeverket, og at vi har måttet kompensere for dette på en god måte. Det har også vært meget interessant og kunne jobbe tett med faglig kompetent personell som Simon McCallum og Jayson David Mackie. Det har også vært artig å få sjansen til å utfordre engelskkunnskapene, da all kommunikasjon med Simon og Jayson var på engelsk, og Simon i tillegg har en ganske så interessant dialekt på engelsken sin.

Det har også vært interessant å jobbe med flere reelle bruker, og også kunder i form av Høgskolestyret ved Høhskolen i Gjøvik.

Alt i alt er vi ganske så fornøyd, og håper noen ønsker å jobbe videre med prosjektet senere og utvikler systemet som vi mener har mye potensiale. Vi mener og HIG vil ha stor nytte av er alumnisystem.

9 Kilder

Alle kilder er dobbeltsjekket den 19.05.2010.

CakePHP

http://book.cakephp.org/
http://www.php.net/
http://teknoid.wordpress.com/2008/11/28/cakephp-url-based-language-switching-for-i18n-and-l10n-internationalization-and-localization/
http://bakery.cakephp.org/articles/view/autologin-component-an-auth-remember-me-feature
http://stackoverflow.com/questions/2130167/cakephp-fills-in-the-password-field-on-create-new-user-failure

MySQL

http://www.mysqltutorial.org/
http://www.w3schools.com/sql/default.asp
http://www.java2s.com/Code/SQL/CatalogSQL.htm
http://www.mysql.com/

LDAP

http://www.devshed.com/c/a/PHP/Using-PHP-With-LDAP-part-1/

JavaScript

http://docs.jquery.com/Main_Page
http://www.jstree.com/
http://blog.jeremymartin.name/2008/02/easy-multi-select-transfer-with-jquery.html
http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/
http://tinymce.moxiecode.com/

HTML, CSS

http://www.w3schools.com/

Bøker:

CakePHP Application Development by Ahsanul Bari and Anupom Syam.
ISBN 978-1-847193-89-6.