HOVEDPROSJEKT:

TITTEL

DynaDesk – Nyheter på trådløse enheter

DynaDesk – News on wireless devices

FORFATTER(E): Elin Synnøve Kirkevoll Berger
Håvard Fjær
Johannes Leiknes Nag
Stina Hagen
Aimée Skevik

Dato: 23. mai 2002

Sammendrag av hovedprosjektet

Nr. : 1
Dato: 23. mai 02

Tittel: DynaDesk – Nyheter på trådløse enheter
DynaDesk – News on wireless clients

Deltakere: Elin Synnøve Kirkevoll Berger
Håvard Fjær
Stina Hagen
Johannes Leiknes Nag
Aimée Skevik

Veileder: Høgskolelektor Rune Lossius

Oppdragsgiver: GAN Media AS

Kontaktperson: Svein Mathisen

Stikkord (4 stk): Filtrering av nyheter, nyhetsmedarbeidere, enhetstilpasning, IP-soner.

Antal sider: 150 **Antall bilag:** 9 **Tilgjengelighet:** Åpen

Kort beskrivelse av hovedprosjektet:

DynaDesk systemet tilbyr informasjonsmedarbeidere, nyhetsbyråer og aviser et mobilt, trådløst filter som gir brukerne kontinuerlig oppdatert informasjon over bærbare enheter. Profilen sørger for selektiv, personifisert informasjon til de trådløse enhetene. I dette ligger en dynamikk som tillater brukere å endre sin brukerprofil via den trådløse enheten. Dette innebærer at enheten blir matet med personifisert informasjon så lenge man er på nett. Profilen kan endres, avhengig av hvilke tema kunden er opptatt av. Brukere kan utføre redaksjonell bearbeiding av saker supplert på basis av nyheter som er hentet ut fra DynaDesk.

Muligheten for å benytte IP-sone teknologi for å hente lokasjonsbaserte nyheter har vært undersøkt, men hovedprosjektgruppen anbefaler GAN Media å utsette implementasjonen til IP-soner er mer utbredt og standardisert.

gan media

DynaDesk

– nyheter på trådløse enheter

Forord

Denne rapporten er utarbeidet som en del av hovedprosjektet DynaDesk – nyheter på trådløse enheter. Siste semesteret av de to-årige og tre-årige utdannelsene ved Høgskolen i Gjøvik utfører studentene et hovedprosjekt. Prosjektet er obligatorisk og teller 6 vekttall av de totale 60 som inngår i ingeniørstudiet. Dette prosjektet skal ta utgangspunkt i en realistisk og faglig relevant problemstilling, og legges opp slik at kunnskap og ferdigheter fra flere fagområder i studiet benyttes. Rapporten inneholder dokumentasjon fra arbeidet med å utvikle en prototype ved hjelp av Java-programmering.

GAN Media inngikk høsten 2001 samarbeidsavtale med Høgskolen i Gjøvik. I den forbindelse ønsket de at studenter ved skolen skulle utføre et prosjekt for dem. Høgskolelektor Sven Erik Skarsbø formidlet kontakt mellom GAN Media og oss, noe som førte til at DynaDesk-prosjektet ble satt igang.

Oppdragsgiver GAN Media fortjener en stor takk for all hjelp og ikke minst lån av Pocket PC. Våre kontaktpersoner Svein Mathisen og Pepijn Oomen har alltid tatt seg tid til å svare på spørsmål, og har generelt vært veldig positive til gruppens arbeid.

Vi ønsker også å takke vår veileder fra HiG, Rune Lossius. Han har med sin kompetanse innen både teknologi og prosjektarbeid vært til stor hjelp for prosjektgruppen. Gjennom hele prosjektet har han peilet oss inn på riktig spor ved å komme med gode råd og vink om hvordan ting kan løses.

Gjøvik, mai 2002

Elin Synnøve Kirkevoll Berger

Håvard Fjær

Johannes Leiknes Nag

Stina Hagen

Aimée Skevik

Innhold

Design

Implementering

Testing og kvalitetssikring

Diskusjon av resultat

1 Introduksjon

Innledning

Dette kapitlet er innledningen til rapporten. Her finner du en beskrivelse av hvordan rapporten er organisert, og sammenhengen mellom de ulike kapitlene. Videre følger en fullstendig definisjon av oppgaven. Vi vil utdype hva oppgaven går ut på – med formål, problemområde, omfang og avgrensning. Det henvises videre til DynaNews som oppgaven bygger på, [se vedlegg B]. Målgruppen for prosjektrapporten blir presentert i mål- og mottakeranalysen.

1.1 Rapportens organisering

Rapporten er i sin helhet lagt opp etter malen Høgskolen i Gjøvik har pålagt oss. Rapporten er delt inn i ti kapitler. Innholdet i de forskjellige kapitlene er som følger:

Kapittel 1 – Introduksjon. Her forklares det kort om oppgavedefinisjonen, rammer, målgrupper, terminologiliste, prosjektgruppens bakgrunn og dens arbeidsform.

Kapittel 2 – Kravspesifikasjonen. Her defineres hva vi skal løse – de funksjonelle og operasjonelle kravene til systemet.

Kapittel 3 – Analyse av hva systemet skal dekke i forhold til kravspesifikasjonen og noen vurderinger i forhold til dette. Omfatter også en brukerveiledning for DynaDesk.

Kapittel 4 – Design. Hovedtrekkene fra designet av systemet beskrives. Vi ser hvordan gruppen har valgt å løse oppgaven kravspesifikasjonen beskriver.

Kapittel 5 – Implementering. Omhandler valg av programmeringsspråk, metoder, verktøy og detaljert pratktisk beskrivelse av hvordan oppgaven er løst.

Kapittel 6 – Testing og kvalitetssikring. Viser hvordan gruppen har testet systemet underveis i utviklingen og hvordan kvalitetsikringen av produktet og arbeidsformer har foregått.

Kapittel 7 – Diskusjon. Her diskuteres resultatetene av produktet og rapporten i forhold til kravspesifikasjonen. Vi ser også på svakheter og muligheter i systemet.

Kapittel 8 – Konklusjon på oppgaven. Inneholder hovedkonklusjonen, faglige resultater, subjektive oppfatninger av oppgaven og drøfting av relevante temaer. Det presenteres hva som kunne vært gjort bedre og hvordan. Vurdering av gruppens arbeidsinnsats, prosjektresultatet, samarbeidet med veileder og oppdragsgiver, og om vi holdt oss innefor tidsrammene.

Kapittel 9 – Litteraturliste.

Kapittel 10 – Vedlegg. Definisjoner, statusrapporter, forprosjekt, møtereferater, aktivitetslogg, Ganttskjema, rapport om IP-sone, konfidensialitetsavtale.

1.2 Oppgavedefinisjon

1.2.1 Formål og problemområde

Redaksjonsmedarbeidere som er på farten vil ha nytte av selektiv, personifisert informasjon til trådløse enheter. Informasjonen skal velges ut etter personlig profil, og evt. etter lokasjon. Brukeren vil ha behov for å endre denne profilen fra sin trådløse ehnet. Det er også ønskelig å kunne bearbeide informasjonen redaksjonelt, basert på stoffet som hentes inn.

1.2.2 DynaNews – vårt utgangspunkt

Mange selskaper, organisasjoner og den offentlige sektor arbeider med informasjonsinnhenting. Nyhetsformidling, internettportaler, konkurrentovervåking eller temabasert informasjonsinnhenting er noen eksempler. GAN Media tilbyr idag tjenesten DynaNews, som automatiserer innhenting av nyheter. [Se vedlegg B] for nærmere beskrivelse fra GAN Media. De ønsker dette systemet tilpasset trådløse enheter.

1.2.3 Resultatmål

Prosjektet skal føre fram til prototypene DynaDesk 1.0, DynaDesk 1.1 og sluttproduktet DynaDesk 2.0. I sluttproduktet skal det utarbeides en rapport hvor vi tar stilling til om IP-sone teknologi kan benyttes som en del av DynaDesk, for å tilby brukerene lokasjonsbaserte nyheter.

1.2.4 Effektmål

Effektmålet for gruppen er å tilegne seg gode fagkunnskaper innen JSP, JavaServlets, Java, XML, trådløs kommunikasjon, databaser og ikke minst prosjektarbeid. Gjennomføringen av prosjektet vil føre til at gruppens medlemmer får erfaring med å jobbe i et langvarig prosjekt hvor man fordyper seg innenfor et fagområde. Dette innebærer erverving av kunnskap og erfaringer innen planlegging av oppgaver, strukturering av arbeidet, oppfølging av prosjektets framdrift, vurdering av eventuelle tiltak som må igangsettes, samarbeid mellom prosjektdeltakerne, oppdragsgiver og veileder, dokumentering av arbeidet samt rapportskriving.

1.2.5 Omfang og avgrensing

Oppgaven omfatter å hente selektiv, personifisert informasjon til trådløse enheter. Dette innebærer at enheten blir matet med personifisert informasjon så lenge man er på nett. Profilen kan endres, avhengig av hvilke saksområder man er opptatt av.
På grunnlag av studier vil vi også vurdere mulighetene og eventuelt utarbeide:

- Kobling mot desksystemer for mobil, redaksjonell bearbeiding av saker supplert med/på basis av informasjon av DynaNews.
- Lokasjonsbaserte tilpasninger av profilene, som – eksempelvis ved bruk av IP-sone teknologi – kan foreta geografiske og innholdsmessige tilpasninger av kildene.

1.2.6 Tidsramme

Produktet skal utvikles i løpet av fem måneder våren 2002. Sluttdato for prosjektet er 23. mai 2002.

1.2.7 Oppdragsgivere

GAN Media, v/Jon Urdal og Svein Mathisen.

1.2.8 Oppdragstakere

Elin Synnøve Berger
Håvard Fjær
Stina Hagen
Johannes Nag
Aimée Skevik (Prosjektleder)

1.2.9 Kontrakt

På grunn av prosjektets konfidensialitetskrav og GAN Medias rettigheter er det tegnet en konfidensialitetsavtale mellom prosjektgruppen og GAN Media. Hensikten med avtalen er å besørge nødvendig konfidensialitet knyttet til informasjon som er virksomhetskritisk for GAN Media og som følgelig ikke under noen omstendighet skal tilflyte tredjeparter eller benyttes av prosjektdeltageren utenfor rammene av dette prosjektet. [Se vedlegg C]

1.2.10 HiGs rolle

Rollen til Høgskolen i Gjøvik er å veilede gruppen under prosjektet. Dette gjøres av veileder, høyskolelektor Rune Lossius.

1.3 Ressurser og rammer

Hos HiG har vi fått tildelt et grupperom som vi kan benytte til prosjektformål. Dette grupperommet skal fordeles på to grupper.

Av utstyr har vi en server (hig.gan.no) som GAN Media har stillt til vår disposisjon med nødvendig programvare og kode. Vi har fått låne en PC fra HiG og vil også bruke våre egne. Disse maskinene er tilkoblet Internett. GAN Media skaffer en PDA m/ nettilkobling til bruk under utviklingen.

Av litteratur har biblioteket faglitteratur tilgjengelig, og her er det også mulig å bestille bøker fra andre biblioteker i Norge. I tillegg finnes det gode manualer og hjelp på Internett. Veiledere ved HiG og GAN Media står også til disposisjon.

1.4 Målgruppe for rapporten

Denne rapporten er i sin helhet skrevet med tanke på en enkel og god dokumentasjon overfor skolens sensorer for hovedprosjektet, veileder og til GAN Media som skal utvikle prosjektet videre.

Gjennom hele denne rapporten, inklusive vedlegg, har vi tatt utgangspunkt i at språket som brukes skal kunne forstås av personer med generell allmennkunnskaper. Det må sies at kapittel 4 og 5 hovedsakelig er rettet mot lesere med en viss programmeringsbakgrunn. Vi henviser til terminologi listen i pkt. 1.7, som vil forklare ord og uttrykk som ikke med enkelhet kan forstås ut fra teksten. Definisjoner av data-tekniske ord og uttrykk finnes i [vedlegg I].

1.5 Målgruppe for produktet

Målgruppen for DynaDesk er redaksjonsarbeidere som jobber trådløst og som forflytter seg geografisk.

1.6 Studentenes faglige bakgrunn

1.6.1 Studiebakgrunn

Fire av gruppens medlemmer går den treårige grafiske ingeniørutdanningen, mens gruppens femte medlem går den to-årige data- og multimedialinjen skolen tilbyr. De som går grafisk har gjort bevisste valg av fagkombinasjonen siden begynnelsen av studiet ved å ta flere fag som tilbys dataingeniørlinjen. Gruppen har tilsammen et vidt spekter av kompetanse.

Nedenfor følger en liste over noen av de fagområdene vi har kompetanse på, relatert til prosjektoppgaven:

- HTML, XML, CSS
- Brukervennlighet
- Databaser
- C++
- Algoritmiske metoder
- Java
- Klient- og serversideprogrammering
- SQL

1.6.2 Relevant jobberfaring

Håvard er ansatt i firmaet NETTOPP AS på Gjøvik som utvikler nettsteder for små og mellomstore firmaer. Elin Synnøve er engasjert av Nettskolen.no som gjennom sammensmelting av fag, pedagogikk og teknologi skal bidra til lønnsomme kvalitetsløsninger innen fleksibel læring. Johannes har startet foretaket GRAFIKER.NO som bl.a. drifter nettstedet sveio.com. Stina var engasjert av BYTE AS innen klient-serversideprosjekter sommeren 2001.

1.6.3 Organisering

Arbeidsformen til gruppen går under problembasert læring (PBL). Gjennom ukemøter og daglige morgenmøter har arbeidsoppgavene blitt fordelt på midlertidige team. For å utnytte størrelsen på gruppen har møtevirksomheten vært viktig for utveksling av kompetanse og erfaringer.

1.6.4 Kompetanseheving

Nødvendig teknisk informasjon skaffes gjennom fortløpende parallelle studier. Disse er basert på følgende kilder:

- Lånt faglitteratur fra bl.a. høgskolens bibliotek.
- Elektroniske og skriftlige manualer som medfølger programmer eller hentes fra Internett.
- GAN Media, gjennom vår kontaktperson Svein Mathisen, og vedlikeholder av DynaNews, Pepijn Oomen.

– Høgskolen i Gjøvik, da i første rekke gruppemedlemmene, med utveksling av kompetanse og erfaring oss i mellom – men også konsultering hos veileder Rune Lossius og andre ansatte.

1.7 Kommunikasjon

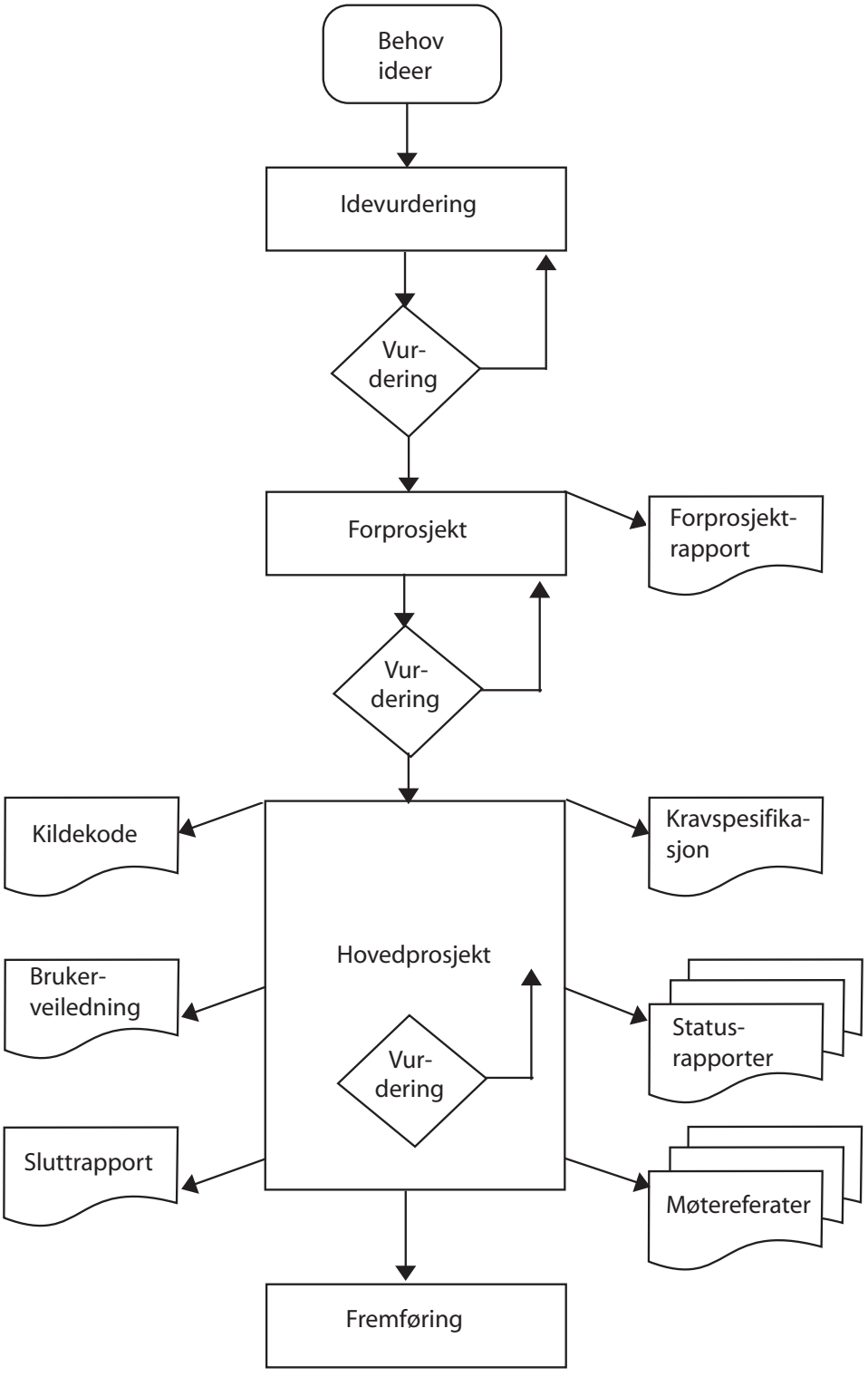
Kommunikasjonen mot oppdragsgiver skjer gjennom møter annenhver fredag og ellers gjennom telefon, e-post og andre elektroniske medier. Prosjektgruppen og oppdragsgiver opprettet e-postadressen hig@gan.no for «massekommunikasjon». Veileder ved HiG vil også være tilgjengelig på møter gjennom e-post eller på sitt kontor. Møtene med veileder var satt opp hver mandag, og i etterkant av dette hadde gruppen ukemøte. Innad i gruppen vil vi jobbe i mindre grupper og ha gruppemøter hver morgen.

1.7.1 Verktøy

Et viktig verktøy for kommunikasjon i prosjektarbeidet skal være prosjektnettstedet, http://hig.gan.no/hig/, som er utviklet av GAN Media og skal oppdateres av oss i prosjektgruppen. Her skal det føres prosjektdagbok for alle gruppemedlemmene, lastes opp referater og andre viktige dokumenter. Det skal også legges ut linker til nettsteder med relevant informasjon, og annen informasjon som angår prosjektet. Gjennom dette nettstedet vil de som er involvert i prosjektet ha mulighet til å følge med på fremgangen.

1.8 Gangen i prosjektet

Prosjektet ble under planleggingen delt inn i fire faser, dette er illustrert i figur 1:
Forprosjekt – hvor vi vurderte oppgaven og utarbeidet planer for hovedprosjektet. Se forprosjektsrapporten [vedlegg H] for nærmere dokumentasjon.
DynaDesk 1.0 – Første prototypen hvor vi har klar arkitektur og informasjonsutveksling fra klient til database og vice versa.
DynaDesk 1.1 – Gir mulighet for å bearbeide nyheter og sende dem videre til eksempelvis en redaksjon.
DynaDesk 2.0 – Lokasjonsbasert informasjon, basert på IP-sone teknologi.



FIGUR 1. Illustrasjonene viser hvordan gangen i hovedprosjektet har foregått.

1.9 Terminologiliste

Oppdragsgiver:	GAN Media
HiG:	Høgskolen i Gjøvik
Veileder:	Rune Lossius, høgskolelektor ved HiG.
Hun, han, brukeren:	En bruker av DynaDesk
PC:	En arbeidsstasjon med f.eks. Mac OS, Windows, Linux.

Følgende ord brukes brukes om hverandre for å variere språket:

- server, webserver og internettjener
- trådløs enhet, håndholdt enhet, PDA, Palm og Pocket PC
- nettside og webside
- JavaBeans, bean og beans
- systemet, produktet, løsningen, prototypen og applikasjonen

2 Kravspesifikasjon

Innledning

I dette kapittelet defineres de funksjonelle og operasjonelle kravene til systemet. Hva vi skal løse blir forklart i detalj.

2.1 Brukerbeskrivelse

2.1.1 Omgivelser

Systemet skal kjøres på en standard internettjener og være tilgjengelig for brukere via Internett. Internettjeneren skal kjøre Apache webserver med Tomcat, og en DB2 database. Vi optimaliserer løsningen for Linux operativsystem på serversiden.

Brukerene skal kunne bruke sine nettlesere for å aksessere systemet. Systemet skal optimaliseres for PDA og bærbare PC-er, med nyere nettlesere.

2.1.2 Systemets brukere

Driftspersonell: GAN Media vil stå for driften av DynaDesk. Disse har kjennskap fra DynaNews-systemet og drifter i dag dette. De har ansvar for å implementere DynaDesk i DynaNews – eventuelt erstatte DynaNews. Vi vil ha kontinuerlig kontakt med disse under utviklingen av prosjektet. Gjennom møtevirksomheten og rapporten fra prosjektet forventer vi at de lærer og forstår systemet.
Brukere: Primært redaksjonsarbeidere i nyhetsbyråer og aviser som forflytter seg geografisk i arbeidet sitt. Disse ønsker kontinuerlig oppdatert informasjon over bærbare enheter. Vi forutsetter at disse har kjennskap til bruk av Internett og generell kunnskap om bruk av nettleser og PC/PDA. Systemet skal være selvforklarende for denne gruppen.

2.1.3 Operasjon

Systemet bør være tilgjengelig så mye som mulig. Dette er avhengig av oppetiden for GAN Medias internettjenere. Ved feil i systemet må dette rettes av driftspersonellet og de som er ansvarlig for vedlikehold av internettjenester hos GAN Media. Systemet skal selv takle feil i input og klare å gjenopprette seg selv etter en driftsstans i systemene.

2.1.4 Ytelse

Ytelsen til systemet vil være begrenset av internettjeneren, klientenheten, nettleseren og internettforbindelsen.

2.1.5 Forutsetninger

Vi forutsetter at alle brukerene har tilgang til Internett på PDA eller PC. På PDA vil vi teste på de mest utbredte nettleserene, mens på PC er kravet at sidene fungerer på 4.0-nettlesere eller bedre.

2.2 Funksjonell kravspesifikasjon

2.2.1 Funksjon

Systemet skal kjenne igjen hvilken type enhet brukeren aksesserer fra og ut ifra dette velge et tilpasset brukergrensesnitt. Systemet skal videre validere innlogging av bruker. Er innlogging godkjent skal brukeren komme til sine nyheter. Her finnes det en menylinje hvor man kan velge å søke i nyheter, endre profilen eller skrive

en ny artikkel. Artikkelskjema og eventuell eksport av disse dataene skal kun være tilgjengelig dersom man entrer siden med en PC.

2.2.2 Funksjonell struktur

Selve systemet utvikles i løpet tre faser; DynaDesk 1.0, DynaDesk 1.1 og DynaDesk 2.0.
I DynaDesk 1.0 skal funksjonaliteten for gjenkjenning av enhet og mulighet for å hente og lese nyheter implementeres.
I Dynadesk 1.1 skal endring av profil og artikkelskjema implementeres. Kun brukere med PC vil ha tilgang til artikkelskjemaet, da dette vil være vanskelig og tungvint å bruke for bærbare enheter med lav skjermoppløsning. I artikkelskjemaet skal bruker kunne klippe ut elementer fra en nyhet og lime inn i et tomt tekstfelt, i kombinasjon med at hun skriver egen tekst.
I DynaDesk 2.0 skal det skrives en rapport som avgjør om IP-sone teknologi er noe Dynadesk kan benytte seg av for å oppnå lokasjonsbasert innhenting av nyheter. Dette skal så eventuelt implementeres i DynaDesk. All funksjonalitet fra tidligere prototyper skal videreutvikles og ferdiggjøres, eventuelle feil rettes opp. Ekstra funksjonalitet skal implementeres i denne prototypen om vi får tid. Selve systemet skal være testet ferdig med alle funksjonene implementert.

2.2.3 Dataspesifikasjon

2.2.3.1 Input

Brukere av systemet legger inn sine data via et grafisk brukergrensesnitt i nettleseren. For å få tilgang til sidene må de logge seg inn med gyldig brukernavn og passord.
Endre profil: Når brukeren har logget seg inn skal hun ha mulighet til å endre profilen sin. Det vil si at hun kan endre eller legge til stikkord, kategorier og/eller kilder – altså nettsteder.
Artikkelskjema: Brukeren skal her kunne komponere sin egen artikkel basert på en nyhet som er enten hentet fra sin egen profil eller fra en annen kilde i systemet. Skjemaet vil bestå av et tekstfelt og et vindu med en aktuell nyhet. I tekstfeltet kan bruker skrive inn egen tekst, samt kopiere inn elementer fra nyheten. Bruker skal ha mulighet til å lagre sin nykomponerte artikkel.

2.2.3.2 Output

Bruker vil få feilmelding dersom felter er feil utfylt. Hun vil da få sjansen til å fylle ut feltene igjen. Når operasjoner er utført vil bruker få tilbakemelding på at dette er gjort, hvis endringene ikke vises direkte i skjermbildet.

2.2.3.3 Datarammeverk

En bruker kan være en bedrift med gyldig brukernavn og passord. Denne brukeren innehar en profil i DynaDesk systemet som brukes for å hente inn nyheter som er av interesse for dem. Profilen består av to kategorisystem; et for stikkord, og et for kilder. Nyhetslisten genereres ut fra stikkord sortert etter kildekategori. Maks antall kilder og stikkord avgjøres av abonnementstype hos GAN Media.

2.2.4. Overordnede operasjonelle systemkrav

2.2.4.1 Normal operasjon

Modus og kontroll: Når en bruker kommer inn på systemet vil hun være oppkoblet så lenge hun ikke er inaktiv lengre enn en bestemt tidsperiode. Er hun det må hun koble seg opp på nytt.
Ytelse: Ytelsen til systemet vil være begrenset av internettjeneren, klientenheten, nettleseren og internettforbindelsen.
Systemet DynaNews bestemmer hvor ofte en kilde skal skannes og hvor ofte nyheter blir hentet. Så snart DynaNews har funnet nyheter som passer til brukerens profil blir profilnyhetene oppdatert.
Sikkerhet: Systemet sjekker input fra bruker, og gir feilmelding dersom denne ikke er riktig. Er feltene tomme får bruker feilmelding om dette, er det noe annet som er feil, får de beskjed om å prøve igjen. Sikkerhet rundt konfigurering av brukerrettigheter ivaretas av selve systemet.
Oppstart og nedtagning: Systemet er nytt og skal installeres på samme internettjener som DynaNews. Systemet tas ned når internettjeneren blir tatt ned.
Tilgjengelighet: Internettjeneren avgjør tilgjengeligheten. Systemet vil ha samme oppetid som internettjeneren.

2.2.4.2 Operasjon i feilsituasjoner

Feilrapportering: Brukeren vil få en forklarende feilmelding dersom de gir feil input.
Sikkerhet: Feil skal ikke påvirke sikkerheten. Dersom det oppstår en ukontrollert driftsstans i internettjeneren, skal det ikke føre til tap av data som allerede er lagret. Går systemet ned, må alle brukere logge seg inn på nytt.

2.3 Begrensinger

2.3.1 Begrensninger fra oppdragsgiver

Systemet skal ikke føre til noen nye investeringer i programvare eller maskinvare hos GAN Media.

Produktet vil i første omgang kun bli tilpasset PC og PDA, og disse skal ikke trenge annet programvare enn nettleser. Tilpassing til mobiltelefon er ikke et krav fra GAN Media.

2.3.2 Software design begrensinger

2.3.2.1 Software standarder og språk

Systemet skal utvikles i Java, JSP, HTML, CSS og JavaScript. Det skal benyttes DB2 database.

2.3.2.2 Software pakker/verktøy

For at DynaDesk systemet skal fungere er det avhengig av at DynaNews også fungerer. Går DynaNews ned, vil DynaDesk ikke få nye nyheter å presentere for brukeren. For å benytte systemet må bruker ha installert en nettleser av typen 4.0 eller nyere. Det er ønskelig at nettlesere støtter CSS og JavaScript. For PDA vil vi optimalisere systemet for de nettleserene som er mest utbredt.

2.3.2.3 Software kommunikasjonsstandarder og grensesnitt

Kommunikasjon mellom bruker og applikasjon skal skje via Internett. Brukere skal kunne gå inn på sidene ved å bruke en vanlig internettforbindelse eller via en trådløst forbindelse.

2.3.2.4 Database

I løsningen benytter vi en kopi av DB2-databasen som oppdragsgiver bruker i DynaNews. Det er et ønske fra GAN Media å lage systemet slik at det er enkelt å skifte til andre databasetyper.

2.3.2.5 Operativsystem

Systemet vil når det er ferdig utviklet gå på en JVM (Java Virtual Machine), noe som i praksis vil si at systemet kan kjøres på de fleste operativsystemer, som Windows og Linux.

2.3.3 Hardware-designbegrensinger

Kapasiteten på DynaDesk-tjenesten er først og fremst avhengig av internettjeneren, og dens kapasitet, samt kapasiteten på internettforbindelsen.

2.4 Aspekter rundt livssyklus

Informasjonen i systemet vil innhentes gjennom DynaNews. Andre data i systemet skal vedlikeholdes av driftspersonellet. Det er de som avgjør hvilke brukere som har tilgang og hvilke klienter det er støtte for. Den tekniske driften og ansvaret for at systemet er tilgjengelig, har de som fra før driver internettjenester ved GAN Media. Siden DynaDesk er front-end til DynaNews vil DynaDesk gå ut av drift om dersom DynaNews legges ned.

2.2.1 Krav

Brukerstøtte: GAN Media står for brukerstøtte når systemet er satt i bruk.
Vedlikehold: Systemet skal stille små eller ingen krav til service og vedlikehold.
Utvidelser: Ved eventuell videreutvikling av systemet er det GAN Media som har ansvaret for dette.
Konfidensialitet: GAN Media krever at kildekoden skal være konfidensiell.

2.4.2 Krav til utvidelser

Systemet skal være enkelt å utvide med ny funksjonalitet og design. Det legges derfor stor vekt på å skille design og kode, samt å holde orden i klasser og funksjoner i koden.

2.5 Aspekter omkring installasjon

2.5.1 Innstallering

GAN Media skal selv stå for implementeringen av DynaDesk på deres internettjener.

2.5.2 Overgang og omlegging

Systemet skal kunne benyttes som frontend-delen på oppdragsgivers allerede eksisterende løsning, DynaNews. Det vil si det er DynaDesk-brukeren skal operere mot. For at DynaDesk skal kunne erstatte hele DynaNews må det i tillegg utvikles et administratorgrensesnitt, men dette ligger ikke innenfor prosjektoppgavens rammer.

2.5.3 Opplæring

Vi skal ikke gi noen opplæring på bruk av systemet. Systemet skal være selvforklarende og gi gode feilmeldinger dersom kundene gjør feil.

2.6 Utgivelser underveis

2.6.1 Statusrapporter

Vi skal levere tre statusrapporter underveis til veileder og oppdragsgiver. Disse skal leveres ved milepælene 8. mars, 12. og 30. april. De skal inneholde gruppens vurderinger vedrørende vårt ståsted i forhold til tidsskjemaet og vår fremdriftsplan, hva vi har gjort ferdig og hva vi holder på med av arbeidsoppgaver. Disse rapportene vil også gi en beskrivelse av samarbeid og motivasjon, og hvilke problemer og muligheter vi ser vedrørende hovedprosjektet fremover i tid. Statusrapportene ligger som [vedlegg F].

2.6.2 IP-sonerapport

En rapport om IP-soner vil bli skrevet som en del av DynaDesk 2.0. Her skal vi undersøke mulighetene for å benytte IP-sone teknologi som et verktøy for å hente inn lokasjonsbaserte nyheter. Intensjonen til GAN Media er at bruker kan få nyheter ut fra sin geografiske plassering uten å fortelle systemet om lokasjonen.

2.7 Rutiner for organisering av kvalitetssikring

2.7.1 Dokumentasjon

Kildekoden er i sin helhet konfidensiell, etter krav fra oppdragsgiver. Vi kommer i stedet til å lage Javadocs som viser oversikt over de klasser og metoder som lages i løsningen. Rapporten vil dokumentere prosjektgangen som er en etterlevelse av prosjekt planleggingen. Prosjektweben vil dokumentere møter og aktiviteter. Vi har utarbeidet vår egen mal som vi skal bruke til møtereferater.

2.7.2 Verktøy

Å velge riktig verktøy er en del av kvalitetsikringen av et prosjekt. I vårt tilfelle er dette utviklingsprogramvare og grafisk programvare.

JBuilder – Benyttes til programering.
Visio – Lage Gantt skjema.
Microsoft Word og Adobe InDesign – Benyttes til å skrive rapport.
Adobe Photoshop – Lage figurer, grafikk til nettsidene, logo.
Adobe Illustrator – Lage illustrasjoner som skal følge med rapporten.
Second Copy 2000 – Sikkerhetskopiering.
Java Forte – dokumentasjon av Javaklasser, Javadocs.

2.7.3 Backup

IT-tjenesten prioriterer backup av hovedprosjektene. I tillegg vil vi ukentlig ta backup mellom GAN Media-serveren og HiG-serveren og mellom våre egne datamaskiner. Versjonskontroll utføres for å ta vare på de riktige filene.

2.7.4 Problemrapportering og tiltak

Ved prosjektstart utformet vi et sett med retningslinjer for hva som skal gjøres dersom problemer oppstår, innen visse områder.

Ved fravær:
Ved fraværs skal dette meldes fra på forhånd.
Rapportering: møtereferat.
Tiltak: andre gruppemedlemmer tar over oppgaver til fraværende.

Ved ikke oppnådd kontakt med veileder/oppdragsgiver:
Rapportering: møtelogg.
Tiltak: purring via e-post eller telefon.

Ikke overholdt tidsfrist i forhold til milepæl:
Rapportering: avvik/status-rapport til veileder og oppdragsgiver.
Tiltak: Jobbe mer så en tar igjen fremdriftsplan, hvis dette ikke er mulig må oppgavens omfang revurderes.

2.7.5 Modul- og integrasjonstesting

Hver modul vil når den er ferdig utviklet bli testet opp mot den foreløpige løsningen. Alle funksjonene i DynaDesk kan utvikles parallelt for senere å integreres inn systemet. Dette gjør det lett for GAN Media å videreutvikle systemet.

2.7.6 Konfigurasjons- og versjonsstyring

Vi utvikler systemet i løpet av tre milepæler, og for hver milepæl skal vi komme frem til en ny prototype: DynaDesk 1.0, DynaDesk 1.1, mens den endelige versjonen kalles DynaDesk 2.0.

2.8 Feature Creep

Under utviklingen av produktet ble det avdekket behov og ønsker fra prosjektgruppen og oppdragsgiver om ekstra funksjonalitet utover de krav som var satt i utgangspunktet. Disse funksjonene blir beskrevet nedenfor, og har i større eller mindre grad blitt implementert som en del av løsningen.

2.8.1 Brukernivåtilganger

Det skal dersom ekstra tid utvikles tre ulike brukernivåer i DynaDesk. Dette blir da superbruker, bruker og administrator hos GAN Media.
Bruker – en vanlig bruker hos kunden. Denne kan logge seg inn og se på de nyhetene bedriften abonnerer på. Siden disse ikke nødvendigvis innehar noen IT-erfaring vil systemet være selvforklarende.
Superbruker – en bruker hos kunden som kan ha varierende kjennskap til systemet og til IT generelt. Denne har tilgang til å endre bedriftens profil i DynaDesk. Systemet skal være selvforklarende. Alle ansatte på GAN Media har også superbrukertilgang.
Administrator – et utvalg av ansatte hos GAN Media som har god kjennskap til systemets oppbygging og virkemåte. De har tilgang til hele DynaDesk systemet og ansvaret for å vedlikeholde det. De kan blant annet endre, slette og legge til nye brukere og agenter.

2.8.2 Flerspråkmulighet

Det skal utvikles en funksjonalitet som gjør at når en bruker går inn i DynaDesk vil all statsisk tekst på nettsiden vises i det språket som brukeren har definert i sin nettleser. Dette ser vi for oss må skje ved at systemet først må kjenne igjen språket til nettleseren, og deretter at all statisk tekst settes inn på dette språket.

2.8.3 Søkefunksjon

Brukeren skal ha mulighet til å søke på et valgfritt ord. Når søket utføres skal systemet lete gjennom alle kildene, og presentere de nyhetene som inneholder dette søkeordet.

2.8.4 Hjelpefunksjon

Dersom det blir ekstra tid på slutten vil gruppen lage en enkel hjelpefunksjon som skal finnes i hele systemet. Denne skal gi forklaringer til alle valg og funksjoner som man kan utføre i systemet. Den vil basere seg på analysekapittelet (kap. 3) av rapporten.

2.9 Akseptansekrav

Systemet skal tilfredsstille de kravene som er gitt i denne kravspesifikasjonen. I tillegg kan oppdragsgiver komme med tilføyingen og rettinger kontinuerlig. Feil som oppdages etter 16.05.02 vil ikke bli rettet opp.

3 Analyse

Innledning

Her fremlegges en analyse av hva systemet skal dekke i forhold til kravspesifikasjonen, og noen vurderinger i forhold til dette. Kapittelet skal gi leseren forståelse og oversikt over brukergrensesnittet til systemet. Det legges vekt på at lesere uten programeringsbakgrunn skal forstå hva som forklares og menes.

3.1 Hva systemet skal inneholde

Kravene til systemets funksjonalitet ble første gang definert gjennom kravspesifikasjonen. I løpet av prosjektperioden har disse blitt ytterligere spesifisert etter dialog med oppdragsgiver og veileder. I tillegg er flere funksjoner blitt lagt til.

Systemet skal ha en layoutmessig utforming som tilfredsstiller oppdragsgivers ønsker for et slikt system. Brukere må logge seg inn med validerte brukernavn og passord, og de skal ha muligheter for å gjøre ulike endringer i profilen sin. Når de logger seg inn skal de ha mulighet til å gjøre dette fra ulike typer klienter, henholdsvis PDA og PC. Uansett hva slags enhet de kobler seg på med skal de ha mulighet til å hente opp de nyhetene de abonnerer på, samt å gjøre endringer i profilen sin.

3.2 Eksisterende løsninger

Systemet bygger på DynaNews; et system utviklet av GAN Media for innhenting av nyheter. Se vedlegg for nærmere beskrivelse av DynaNews.

Et annet liknende system er Nyheter.no, som innhenter nyheter fra norske nettsider hvert femte minutt. De leverer også til publiseringssystemer og lagrer ingressene sine lengre enn DynaNews – DynaNews lagrer ingresser i to døgn. For å få søketilgang må man være abonnent hos Nordiske Nyheter AS, men for å se de siste nyhetene i databasen trenger man ikke være abonnent. Vi har ikke funnet informasjon om at sidene er tilpasset andre enheter enn PC. Altså er dette en tjeneste som likner mer på vårt utgangspunkt DynaNews enn på produktet vi skal utvikle.

3.2.1 Hva løste tidligere system?

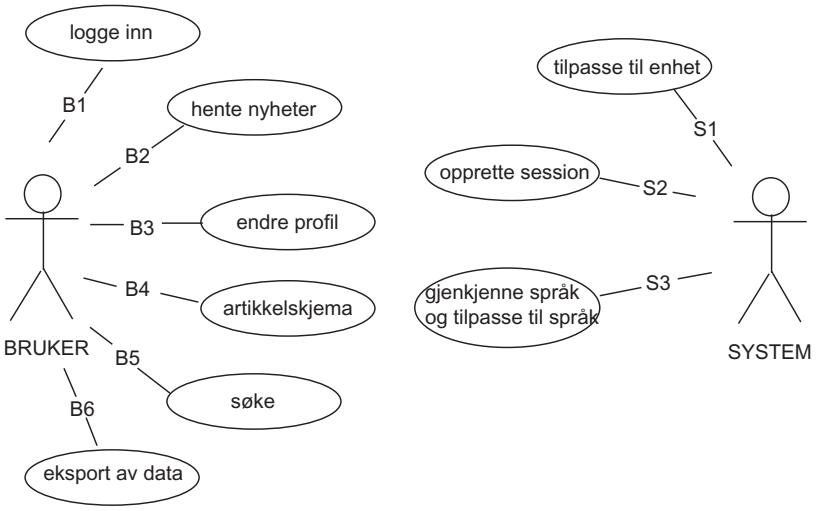
DynaNews2000 automatiserer innhentingen av informasjon som selvstendig applikasjon og modul i Gloria – som er publiseringssystem fra GAN Media. DynaNews foretar automatisert lenkeovervåking, med valgfri pollefrekvens om hvor ofte kilden skal skannes, og leverer nyheter i ulike format til kunden. Kunden kan selv legge til kilder og kriterier til søket gjennom et administrativt grensesnitt, samt å kategorisere etter kilder eller nøkkelord.DynaNews ivaretar en rekke ulike statistiske data – og melder fra til administrator via SMS eller e-post, om en kilde får status «inaktiv». For nærmere informasjon, se vedlegg eller kontakt GAN Media.

3.2.2 Hva løste ikke tidligere system?

- Hverken nyhetene eller grensesnittet er tilpasset ulike enheter.
- Kunden kan ikke søke etter informasjon.
- Det finnes ikke noe system for å lage egne artikler basert på kildene.
- Informasjonen er ikke lokasjonsbasert.
- Ikke tilpasset språk, som norsk og engelsk.

3.2.3 Hva skal dette systemet gjøre?

DynaDesk 1.0 tilpasser alle sider til ulike enheter og lar brukeren søke etter nyheter.
DynaDesk 1.1 har et system for å lage egne artikler basert på innhentet stoff, med mulighet for å sende det til en redaksjon.
DynaDesk 2.0 skal se på muligheten for å benytte IP-sone teknologi for å hente inn lokasjonstilpassede nyheter, forbedre de andre prototypene og løse mest mulig feature creep. Dette er illustrert i figur 2.



FIGUR 2. Figuren viser en oversikt over de funksjoner som den endelige versonene av DynaDesk skal ha ved hjelp av et overordnet use-case.

Use case: B1 – Logge inn
Aktør: Bruker
Type: Primært
Beskrivelse: Brukeren må være registrert for å kunne logge seg inn. Brukervariablene må tas vare på så lenge brukeren er pålogget.

Use case: B2 – Hente nyheter
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet skal kunne hente ut spesifiserte nyheter. Dette er spesifisert i brukerens profil.

Use case: B3 – Endre profil
Aktør: Bruker
Type: Primært
Beskrivelse: Bruker skal kunne endre nøkkelord, kilder og tilhørende kategorier.

Use case: B4 – Artikkelskjema
Aktør: Bruker
Type: Primært
Beskrivelse: Bruker kan lage sine egne artikler ut i fra nyheter han/hun abonnerer på.

Use case: B5 – Søk
Aktør: Bruker
Type: Primært
Beskrivelse: Bruker skriver inn søkeord, og systemet lister opp nyhetene som inneholder dette ordet.

Use case: B6 – Eksport av data
Aktør: Bruker
Type: Sekundært
Beskrivelse: Bruker velger eksporttype og systemet eksporterer data til valgt kilde og på valgt format.

Use case: S1 – Tilpasse til enhet
Aktør: Systemet
Type: Primært
Beskrivelse: Et sett filer med tilpasset design velges ut fra klient/enhets-type.

Use case: S2 – Opprette session
Aktør: Systemet
Type: Primært
Beskrivelse: Et sessionobjekt må opprettes for å ta vare på brukervariablene og informasjon om klienttype.

Use case: S3 – Gjenkjenne og tilpasse til språk
Aktør: Systemet
Type: Sekundær
Beskrivelse: Systemet skal kjenne igjen nettleserens språkinnstilling, og velge språk basert på det.

3.3 GUI

3.3.1 Generelt

Det er viktig at sidene fungerer optimalt i forhold til funksjonalitet, navigasjon, interaktivitet og innhold. Vi har derfor valgt en formell stil som fokuserer på stoffet som presenteres på sidene. Kravene våre er at systemet skal være enkelt å sette seg inn i for nye brukere samtidig som det skal være effektivt å bruke for avanserte brukere.

3.3.2 Skjermoppløsning

- For PC er sidene tilpasset oppløsning på 800x600 og bedre.
- Sidene for PDA fungerer på 160x160, som er den minste skjermoppløsningen på PDA-markedet.
- For alle sidene gjelder det at de er skalerbare, helst uten horisontalt scrolling. Grafikk og bakgrunn utvides over den tilgjengelige plassen.

3.3.3 Nettsidenes utforming

Det er lagt stor vekt på utforming av sidene med hensyn på valg av farger, grafikk, tekst og brukergrensesnitt. Løsningene vi har kommet frem til er avhengig av hvilken type enhet siden lages for.

3.3.3.1 Hvor befinner brukeren seg på nettstedet?

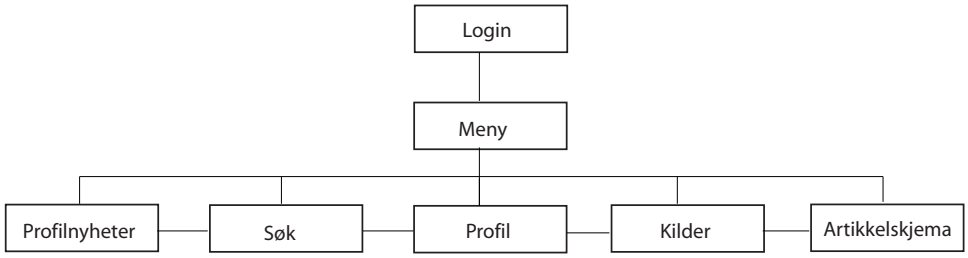
Det er tatt hensyn til at brukeren hele tiden skal vite hvor hun befinner seg både globalt – hvilket nettsted, og lokalt – hvor i nettstedet. DynaDesk-logoen som alltid vises øverst til høyre, som vil fortelle brukeren at hun befinner seg på DynaDesk

nettstedet. Posisjonen vises i URL, sidetittel, korttittel øverst til venstre og sidetittel under menylinjen. Når brukeren har logget seg på vil status vises øverst i boksen på venstre side. Statusboks og tittel under menylinjen er droppet på PDA på grunn av plassmangel.

3.3.3.2 Hvor skal brukeren gå?

Hovednavigasjonen gjøres i menylinjen på toppen av siden, ellers navigeres det med linker/knapper. Eksterne linker er markert med blå farge og har understreking når du holder pekeren over.

3.3.3.3 Flytskjema for nettstedet



FIGUR 3. Her vises gangen i nettstedet, slik det oppleves for brukeren. Artikkelskjemaet vil ikke være tilgjengelig for PDA.

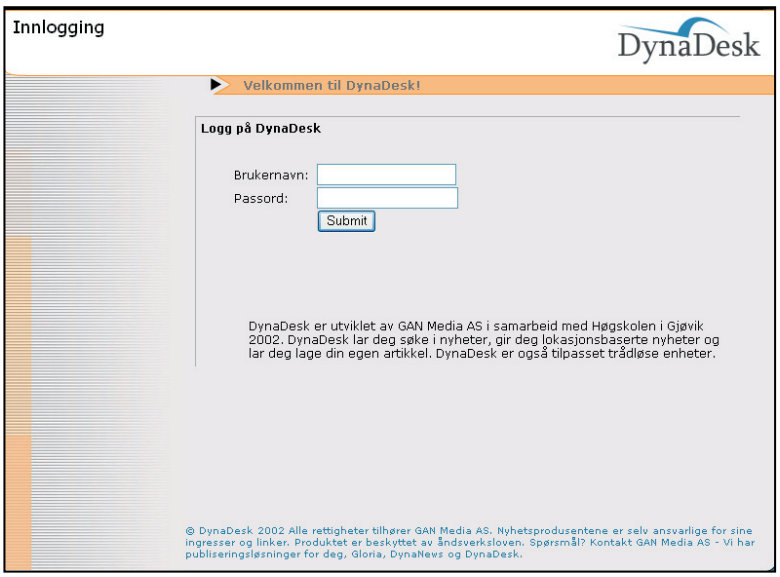
3.3.4 Systemets funksjoner

Før brukeren entrer DynaDesk sjekker systemet hvilken type enhet brukeren sitter på, nettsiden som åpnes er tilpasset denne. Vi har laget to ulike grafiske brukergrensesnitt; en til PDA og en til PC. Det kan lett utvidets til flere, som for eksempel til WAP. Engelsk er defaultspråket for begge enhetene.

3.3.4.1 For PC

Login

Brukeren entrer loginsiden til DynaDesk hvor det gis kort informasjon om hva DynaDesk er, samt hvem som har utviklet det. Det finnes en logo på siden for å informere brukeren hvilket nettsted hun har kommet på, i tillegg til en



FIGUR 4. Innlogging på PC

informasjonstekst som sier at brukeren er på innloggingssiden på nettstedet. Brukeren fyller inn de tomme feltene med brukernavn og passord og dette valideres. Ble det godkjent blir brukeren sendt videre til nyheter-siden.

Nyheter

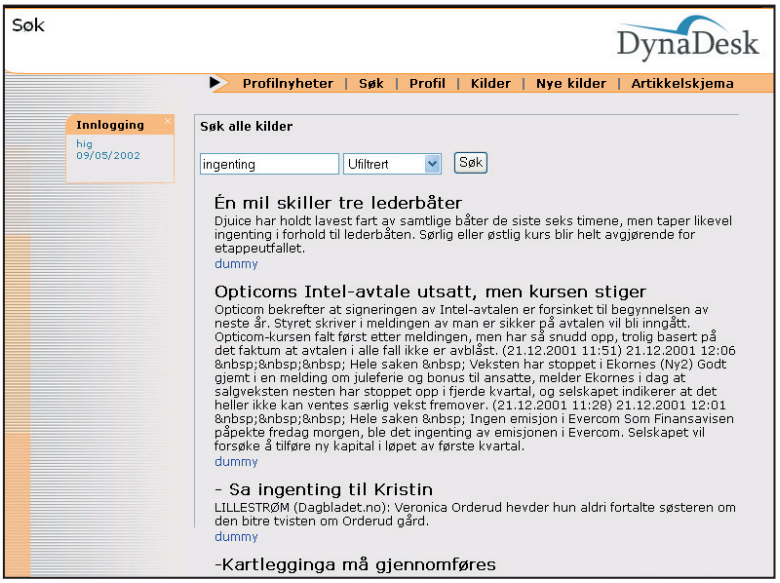
Alle nyhetene som genereres ut ifra brukerens profil blir listet opp her. Alle nyhetene inneholder tittel, ingress og en link til nettsiden artikkelen ligger på. En boks som heter «Innlogging» dukker opp til venstre for nyhetene, og denne innholder informasjon om hvilken bruker som er innlogget, dagens dato, samt en loggut-link. Trykker brukeren på loggut-linken blir hun sendt tilbake til Innloggingssiden. Denne boksen blir værende på samme sted hele tiden, uansett hvor på nettstedet brukeren befinner seg.

En meny blir tilgjengelig på toppen av siden. Denne består av nyheter, som er siden brukeren nå befinner seg på, Søk, Stikkord, Kilder, Nye kilder og Artikkelskjema.

FIGUR 5.

Søk

Brukeren kan her søke på stikkord enten gjennom alle kildene som finnes DynaDesk, eller kun gjennom de som finnes i brukerens profil. Brukeren taster inn søkeordet i det tomme tekstfeltet, velger om søket skal gå gjennom alle kildene eller kun de i profilen, og trykker på søk-knappen. Siden blir da oppdatert med treffene listet opp under søkefeltet. Nyhetene som passet kriteriene blir satt opp på lik måte som i Profilnyhet-siden, med tittel, ingress og link. Tekstfeltet for søkeordet er nå blitt tømt, klar for et nytt søk. Se figur 6.

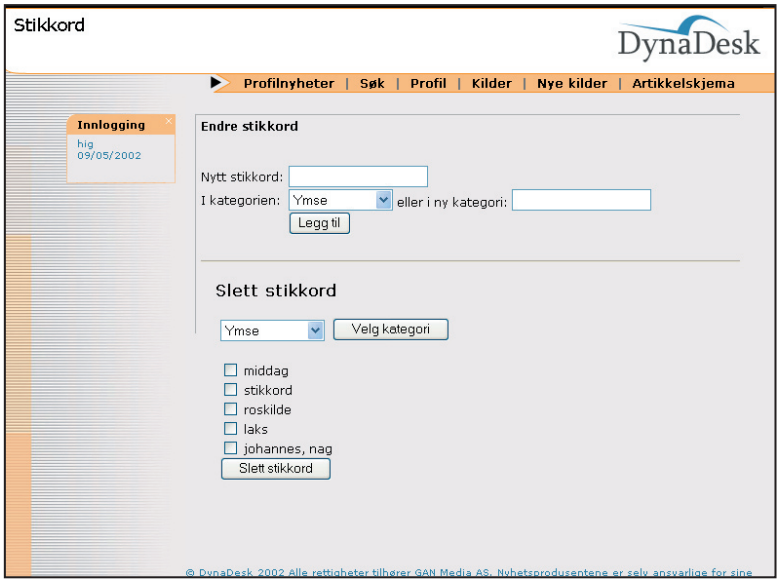


FIGUR 6.

Stikkord

Stikkordsiden har to hovedfunksjoner; lag nytt stikkord og slett stikkord. Øverst kan brukeren skrive inn et nytt stikkord i det tomme tekstfeltet. Stikkordet må ligge i en kategori, og denne velges enten fra en dropdownliste, eller det opprettes en ny. For å utføre endringene trykker brukeren på knappen «Legg til».

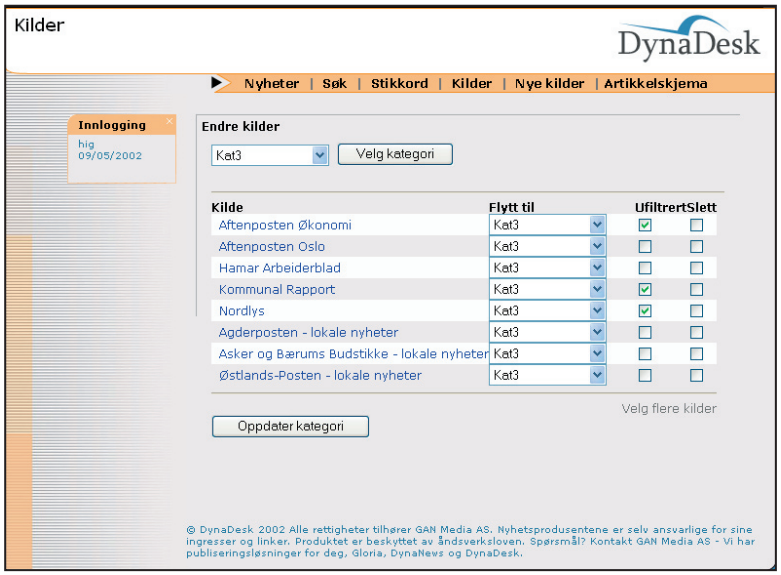
Slette stikkord gjøres på den nederste delen av siden. Stikkordene listes opp ut ifra hvilken kategori som er valgt. Denne kan endres fra en dropdownliste. Stikkordene vil da oppdatere seg når brukeren velger en annen kategori. Ved siden av hvert stikkord er det en avhukingsboks som brukeren huker av hvis hun vil slette det aktuelle stikkordet. For å slette stikkord må hun trykke på submit-knappen merket «Slett stikkord».



FIGUR 7.

Kilder

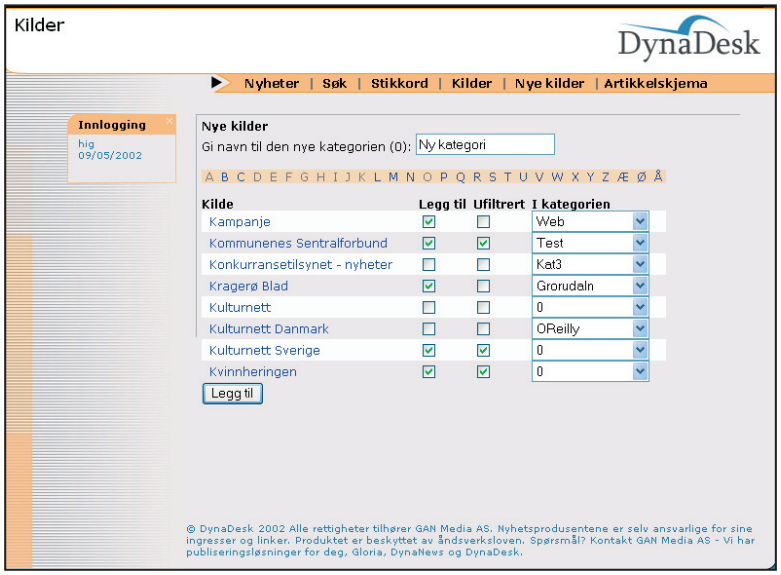
På denne siden kan brukere velge om kilden skal legges i en ny kategori, om den skal filtreres eller om den skal slettes. Først velger brukeren hvilken kategori kilden ligger lagret i. Siden blir da oppdatert og alle kildene i denne kategorien listes opp. Brukeren kan velge om disse kildene skal flyttes til en annen kategori, om de ikke skal filtreres eller om de skal slettes. For å utføre endringene må hun trykke på «Oppdater kategori»-knappen.



FIGUR 8.

Nye kilder

Brukeren kommer inn på en side hvor hun kan legge til nye kilder. Ønsker brukeren å lage en ny kategori å legge kilden i kan dette gjøres ved å fylle inn navnet på den nye kategorien i det tomme tekstfeltet. Ved å trykke på en av bokstavene i alfabetet listes de kildene som begynner på denne bokstaven opp. En kilde som listes opp er også en link til nettstedet slik at brukeren kan ta en titt før det bestemmes om den skal legges til i profilen. Til hver kilde kan brukeren ved hjelp av avhukingsbokser og dropdownlister velge om den skal legges til, om den skal være ufiltrert, og om den skal legges i en eksisterende eller ny kategori. Når endringene er gjort trykker brukeren på



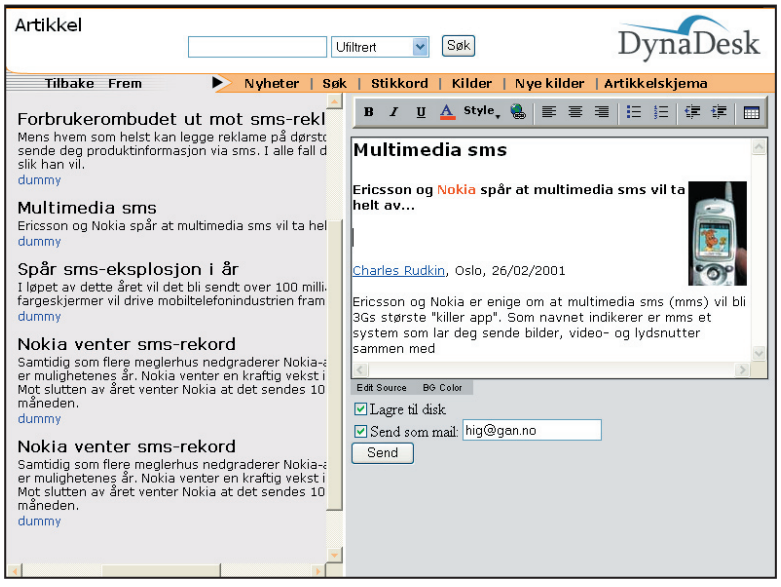
FIGUR 9.

«Legg til»-knappen. Når databasen er opptadert blir siden generert på nytt, og kildene som er lagt til i profilen fjernes fra listen.

Artikkelskjema

Å ha tilgang til denne siden forutsetter at brukeren sitter ved en PC. Hun får da opp en side som har to deler, en til å åpne artikler i, og en til å komponere sin egen i. Her har brukeren noen vanlige tekst editor muligheter, som for eksempel å endre font, størrelse og farge. Brukeren har mulighet til å markere tekst fra den andre artikkelen og dra den over i sin egen. Dette kan også gjøres med bilder. Dette gir reportere og journalister en ny og bedre mulighet til å lage artikler i felten, samt å samarbeide med kollegaer som er stasjonert på ulike geografisk områder.
Når brukeren er ferdig med artikkelen sin kan den lagres på ulike medier. Velger hun «Send som mail» sendes artikkelen som HTML-kode til den oppgitte e-postadressen. Brukeren kan også velge å lagre artikkelen på harddisk. Det åpnes da et «Lagre som»-vindu hvor hun kan velge hvor på maskinen artikkelen skal lagres.

FIGUR 10. Dette skjermbildet viser hvordan artikelskjemaet vil se ut for brukere som har Microsoft Internet Explorer 5.5 eller bedre.



FIGUR 11. Dette skjermbildet viser hvordan artikkelskjemaet vil se ut for brukere som ikke har Microsoft Internet Explorer 5.5 eller bedre.

3.3.4.2 For PDA

I dette avsnittet vil vi se på forskjellen i grensesnittet mellom PC og PDA. Like elementer vil ikke kommenteres på nytt.

Login

Brukeren kommer til en side som ved hjelp av en logo forteller at hun har kommet til DynaDesk. Her finnes det ingen informasjon om hva DynaDesk er, fordi det rett og slett ikke er plass. Er det noen som ønsker mer informasjon, finnes det kontaktinformasjon på bunn av siden. Det vesentlige på denne siden er et tekstfelt for brukernavn og passord hvor en eksisterende kunde skal skrive inn brukernavn og passord. Når dette er gjort og brukeren trykker på submit, valideres kunden og sendes videre til nyhetsiden dersom valideringen ble godkjent.

FIGUR 12.

Nyheter

Denne siden er bygget opp på samme måte som for PC. For å se alle nyhetene må denne siden skrolles vertikalt. Menyen på siden består av Nyheter, som er siden brukeren er på nå, Søk, Stikkord, Kilder og Nye kilder. For at brukeren skal vite hvor hun befinner seg lokalt på nettstedet blir dette opplyst øverst til venstre på siden.

FIGUR 13.

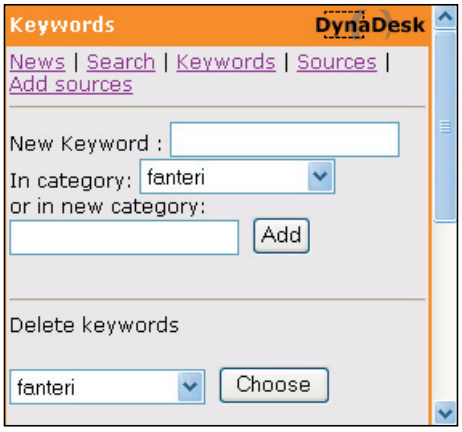
Søk

Denne siden er bygget nøyaktig som søksiden for PC, men med forenklet grafikk.

FIGUR 14.

Stikkord

Denne siden er bygget nøyaktig som stikkordsiden for PC, men med forenklet grafikk.



FIGUR15.

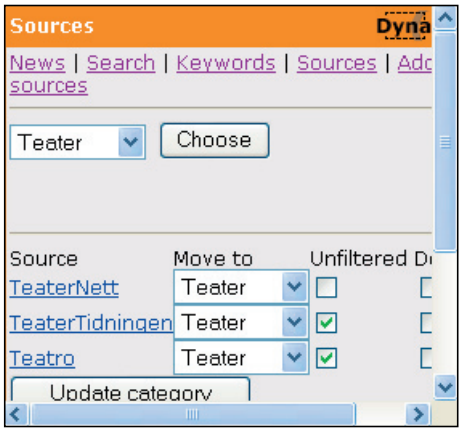
Kilder

Denne siden er bygget nøyaktig som kildersiden for PC, men med forenklet grafikk.

FIGUR 16.

Nye kilder

denne siden er bgget nøyaktig som stikkordsiden for PC, men med forenklet grafikk.



FIGUR 17.

4 Design

Innledning

Hovedtrekkene fra design av systemet beskrives. Kapittelet gir et overordnet bilde av hvordan gruppen har valgt å løse oppgaven kravspesifikasjonen beskriver. Det skal gi leseren en forståelse og oversikt over arkitekturen til systemet, dets oppbygging og funksjoner, og de vurderingene som ligger til grunn for det. Prinsipper som ligger til grunn for det grafiske brukergrensesnittet vil også bli beskrevet. Det legges vekt på at systemutviklere skal forstå hva som forklares og menes.

4.1 Valg av programmeringsspråk

Både blant gruppens medlemmer og fra oppdragsgiver var det ønske om å skille utseende og logikk mest mulig. Oppdragsgiver benytter Linux server, og ville i tillegg at teknologien vi skulle benytte i utviklingen var «open source». Dermed utelukket dette tidlig enkelte serversidespråk som f.eks. ASP og ASP.NET. Før vi valgte språk gjorde vi en undersøkelse hvor vi sammenlignet PHP, JSP og JavaServlets mot hverandre. Valget falt i første omgang på en kombinasjon av JSP, JavaServlets og Java fordi dette gir god mulighet til å skille logikk fra presentasjon. Senere i prosessen omstrukturete vi systemet og tok i bruk Struts og tag biblioteker. Bakgrunnen for omlegging til Struts var at Strutsplattformen inneholder tagger som medfører mer ryddige og oversiktlige JSP-filer, som igjen gjør det lettere å implementere designet.

4.2 Systemarkitektur

Dette avsnittet vil gi en beskrivelse av arkitekturen i system-arkitekturen i DynaDesk, og hvordan denne arkitekturen henger sammen med Struts´ rammeverk. Vi har lagt mye arbeid i å tenke og tegne opp alternative arkitekturer for systemet. I samråd med GAN Media ble vi enige om hvilken vi skulle satse på. Vi har også tatt stilling til om DynaDesk skal utvikles uavhengig eller avhengig av GAN Medias DynaNews. Vi forutsetter at leseren har lest vedlegget med GAN Medias definisjon og forklaring av DynaNews.

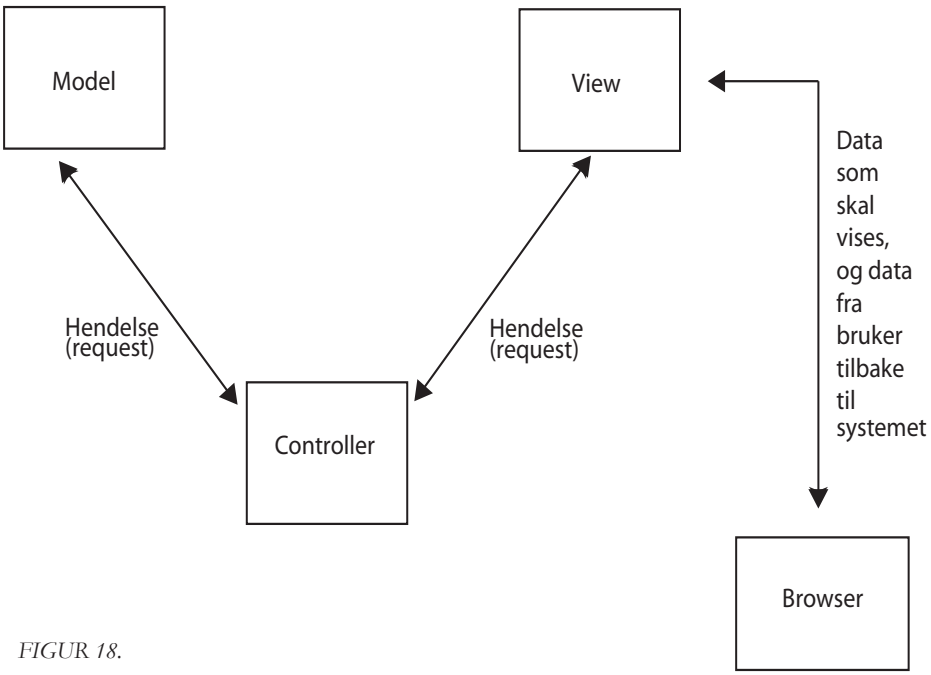
4.2.1 Generelt

Systemarkitekturen har vært omarbeidet i etapper hvor vi for hver etappe har utvidet eller omarbeidet systemets virkemåte ut fra erfaring fra forrige løsning, eller fordi vi har tatt i bruk nye teknologier. Dette fordi vi har jobbet mot en mest mulig dynamisk løsning slik at man i fremtiden lett kan utvide systemet. Med dette som grunnlag har vi lagt stor vekt på å skille designdelen fra logikk-delen, slik at disse delene kunne utvikles separat for siden å kobles sammen. Systemets arkitektur er bygget opp etter MVC-modellen og Struts-rammeverket.

4.2.2 MVC-modellen og Struts-rammeverket

Struts er et «open-source» rammeverk som brukes for å bygge web applikasjoner med Java Servlets og JavaServer Pages. Struts-applikasjoner bygges over MVC-modellen. MVC-modellen er et mønster for oppbygning og strukturering av en applikasjon i tre lag; Model, View og Controller. Denne modellen ble utformet som en del av Smalltalk-80 versjonen av Smalltalk programmeringsspråket – et objektorientert programmeringsspråk – for å separere logikk fra GUI, i følge Rune Lossius, Høgskolelektor ved HiG. Kort beskrevet representerer Model-laget business-logikken i applikasjonen, View-laget gir GUI til brukeren og representerer det logiske presentasjonslaget av applikasjonen, og Contolleren tar imot hendelser fra brukeren, validerer brukerens input og hvilke elementer som skal sendes videre til Model-laget.

Det som er karakteristisk for denne modellen er at Model, View og Controller håndteres som separate enheter, og at endringer gjort i Model automatisk vil reflekteres i de forskjellige Views. Denne trelagsmodellen gjør det enkelt å utvikle fleksible applikasjoner som er enkle å vedlikeholde, og vi får en klar separasjon mellom de ulike komponentene i applikasjonen. Sammenhengen mellom Model, View og Controller illustreres i figur 18.



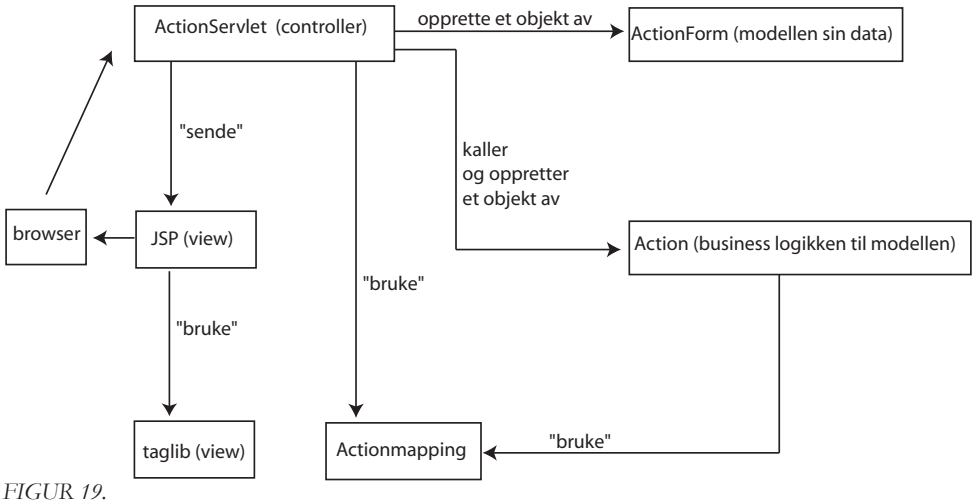
FIGUR 18.

Som figuren viser er det Controller-laget som styrer applikasjonen. Controlleren tar imot reqeuester fra browseren via View-laget og sender disse videre til Model-laget. Model-laget sender data til til View-laget via Controller-laget, og View-laget formaterer og filterer de data som skal vises i browseren.

Struts er bygd opp med fem komponenter som utgjør MVC-modellen. Her har vi trukket linjer mellom de ulike Struts-komponentene og hvor de hører til i MVC-modellen.

De fem komponenter som utgjør en Struts-basert applikasjon består av både applikasjonsspesifikke- og Strutsspesifikke-komponenter. Disse er:

1. HTML-former, generert av JSP-er ved å bruke Struts´ HTML-taglibrary. Disse ligger i View-laget.
2. Struts´ ActionServlet-controller som mottar data fra form. ActionServlet er konfigurert av en XML-fil, struts-config.xml, denne ligger under Controller-laget.
3. Applikasjons-spesifikke underklasser av Struts´ Form-klasse, som er JavaBeans med egenskaper som korresponderer med feltene i den aktuelle formen i en JSP-fil. ActionServleten oppretter automatisk en Form-bean, populerer dens egenskaper med data fra formen og lagrer den i den spesifiserte scopen; vanligvis session eller response. ActionServleten vil da, hvis dette er spesifisert, validere sitt innhold for å sjekke at den har fått gyldig input fra brukeren. Hvis dette ikke blir funnet gyldig sendes brukeren tilbake til JSP-formen. Dette er en del av Controller-laget.
4. Applikasjonsspesifikke underklasser av Struts´ Action-klasse inneholder og kaller applikasjonens business-logikk. Her kan nødvendig data lagres i request eller session objektet som JSP-en trenger for å produsere responsen. Dette er en del av Model-laget.
5. ActionMappings er en Struts-definert klasse som representerer mappingen mellom URL-mønsteret som brukes i en klient-forespørsel og ActionForms og Action-klasser som skal brukes i forespørselen. ActionMappings defineres i struts-config.xml. Dette er en del av Controller-laget.



FIGUR 19.

ActionServleten styrer applikasjonen, og er Controlleren i applikasjonen. Den oppretter Form- objekter og kaller Action-klassene, og bruker ActionMappingen definert i struts-config.xml. Denne filen inneholder en oversikt over hvilke Form-objekter som skal opprettes, og hvilke Actions som skal brukes. ActionServletens liv starter med en forespørsel fra browseren, enten via URL-en som spesifiserer hvilken ActionMapping som skal brukes, eller fra en form.

ActionServleten sender data fra og til Action-objektet via Form-objektet. JSP bruker taglibraries for å vise og motta data fra brukeren i browseren. DynaDesk er bygget opp som en Struts-basert applikasjon som følger MVC-modellen.

Vi ser nå på en mer detaljert sammenheng mellom MVC-modellen og komponentene i DynaDesk.

Model-laget i DynaDesk består av Action-klasser og beans. Action-klassene utfører endringene og kommuniserer med View-laget ved hjelp av beans som respons til hendelser fra brukeren. Beans inneholder data hentet fra databasen via Action-klassene. Vi har samlet alle funksjoner som kommuniserer med databasen i en egen servlet. Denne klassen er ikke en Action-klasse, men representerer den viktigste business-logikken i systemet – all kommunikasjonen mot databasen. Dette har vært nødvendig for å unngå for mange tilkoplinger til databasen. Denne initialiserer tilkoplingen én gang i `init()`-metoden. Det er også logisk å samle funksjoner som utfører noen av de samme operasjonene i samme klasse.

Det skal være et mål i utformingen av systemet å la Action-klassene bare håndeter det som er absolutt nødvendig for å utføre ønskede operasjoner. Vi har lagt stor vekt på å skille business-logikk fra applikasjons-logikk, hvor Action-klassene sammen med database-klassen representerer business laget i DynaDesk, og Form-klassene sammen med Struts´ ActionServlet representerer Applikasjons-laget.

View-laget i DynaDesk består av JSP-filer, taglibraries og resourcefiler. View-laget gir GUI til brukeren og representerer det logiske presentasjonslaget av applikasjonen. View-laget representerer den nåværende tilstanden i Model-laget utifra Model-lagets beans. Beans brukes i JSP-filer ved hjelp av Struts-spesifikke tagger og elementer fra Controller-laget. View-laget skal også presentere de ulike elementene brukeren kan jobbe med for å utføre en operasjon; f.eks former og linker.

Taglibrariene som brukes i View-laget er satt opp og definert i Struts-rammeverket. Struts 1.0 inneholder fire tag-libaraies; HTML, bean, logic og template. Template-librariet er ikke brukt i DynaDesk. Vi vil derfor ikke komme nærmere inn på dette her.

HTML-librariet inneholder tagger for å generere HTML, skrive ut verdier fra en ActionForm i HTML input-felter, og legge ved session-iden i URL-er, slik at ikke cookies er påkrevd i nettleseren.
Logic-biblioteket brukes til flytkontroll innenfor en JSP. Det er f.eks mulig å loope gjennom over en collection (f.eks. en vektor), eller sende brukeren videre til en annen side.
Bean-biblioteket brukes for å manipulere JavaBeans innenfor JSP-en. Dette bliblioteket inneholder tagger som definerer scripting variable basert på egenskaper i eksisterende beans, og inneholder en tag som kjenner igjen hvilket språk brukeren har definert i nettleserens innstillinger. Message-taggen viser statisk tekst som den henter fra Resource-filer.

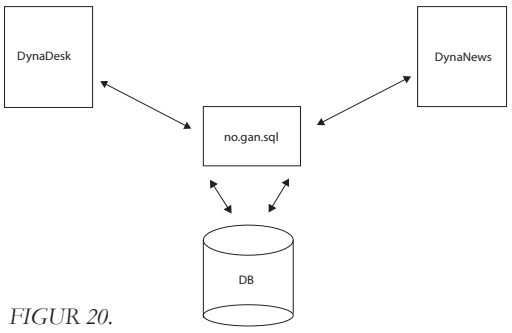
Språk
Som nevnt gir Struts rammeverket mulighet til å tilpasse statisk tekst til brukerens valgte språk i Internett-instillingene. For å få dette til å fungere lagres all statisk tekst i Resource-filer – én Resource-fil for hvert språk. Hver Resource-fil navgis med Resource_xx.properties hvor xx er det nasjonalte toppdomenet for språket.

Controller
Controlleren er bindeleddet mellom View-laget og Model-laget i MVC-arkitekturen. Controlleren i DynaDesk bestar av Form-klasser og ActionServleten som genereres ut fra struts-config.xml. Form-klassene tar imot input fra brukeren via JSP-ene i View-laget og fra databasen via Model-laget. Form-klassene utfører operasjoner for å bestemme hva som skal sendes til Model-laget og til View-laget. Controll-laget representerer applikasjons-logikken i vårt system.

For hver Form-klasse har vi en tilhørende Action-klasse. Disse kan sammen ha flere forskjellige JSP-filer, noe som har vært en stor fordel i utviklingen av DynaDesk hvor vi har et sett med JSP filer for hver enhet.

4.2.3 Sammenhengen mellom DynaDesk og DynaNews

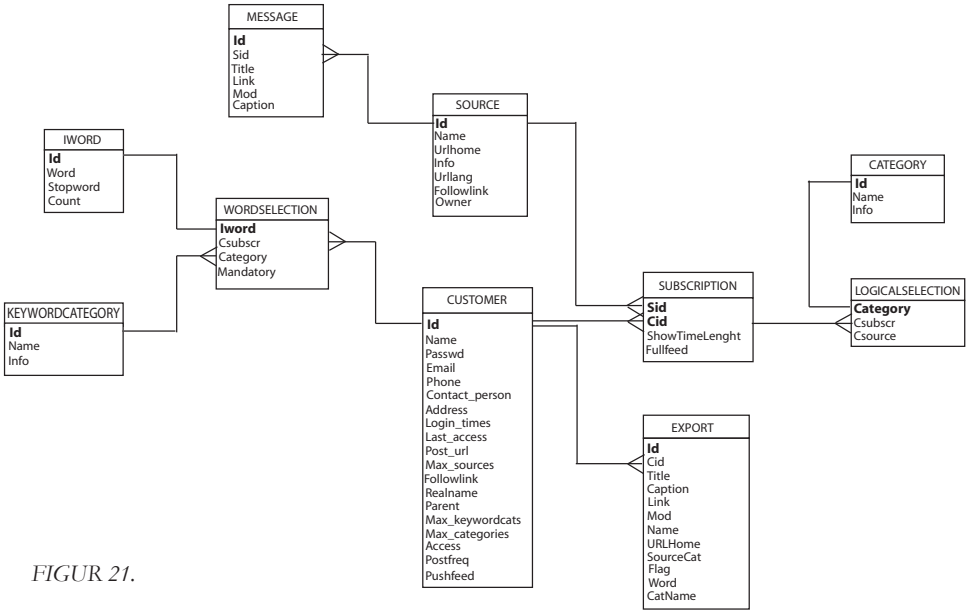
DynaDesk er utviklet uavhengig av klasser fra DynaNews. Koplingen mellom DynaDesk og DynaNews ligger i databasen. DynaDesk bruker en Java-pakke utviklet av GAN Media, no.gan.sql , for kommunikasjon mot databasen. Denne pakken er uavhengig av andre klasser i GAN Medias system, men brukes av både DynaDesk og DynaNews for database kommunikasjon. DyanNews henter inn informasjon som brukes av DynaDesk. DynaDesk kan sees på som «front-end» delen av DynaNews. Vi har jobbet mye med å finne den mest hensiktsmessige oppbyggingen av DynaDesk i forhold til DynaNews, og vi vurderte en periode å implementere DynaDesk inn i DynaNews-pakken. Dette har allikevel ikke blitt gjort fordi funksjonaliteten er så forskjellig fra funksjonaliteten i DynaDesk. DynaDesk er derfor funksjonelt sett uavhengig av DynaNews, men er avhengig av de data DynaNews henter inn og legger i databasen. Uten disse dataene vil DynaDesk ha liten verdi.



FIGUR 20.

4.2.4 Databasen

Databasen vi benytter er en DB2 database. Denne databasen er utformet av GAN Media, og vi har ikke hatt mulighet til å endre tabeller eller relasjoner fordi databasen også brukes av DynaNews. Fordi GAN Media ikke har et fullstendig databasekart har vi måttet gå igjennom de ulike tabellene selv via kommandolinje grensesnittet til DB2. Databasen er bygd opp av 28 tabeller, og uten et oversiktlig databasekart var det en stor utfordring å få oversikten over hvilke tabeller vi skal bruke, primærnøkler, sekundærnøkler og relasjonene mellom disse. Av de 28 tabellene som finnes i databasen bruker vi ti av disse i vår løsning. Disse er illustrert i ER-diagramet.



FIGUR 21.

Her følger en beskrivelse av de tabellene og de feltene vi benytter oss av i databasen. Vi har ikke beskrevet felter eller tabeller vi ikke benytter oss av i DynaDesk.

Customer

Inneholder informasjon om kunden. Denne informajsonen legges inn av administrartor fra GAN Media i deres DynaNews-system.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Brukerens ID i systemet. Primærnøkkelen i tabellen.
Passwd	Varchar(32)	Brukerens passord for pålogging.
Email	Varchar(32)	Kontakt epost adresse til kunden.
Max_sources	Integer(4)	Maksimalt antall kilder kunden kan legge til.
Max_ketwordcats	Integer(4)	Maksimalt antall stikkordskategorier en kunde kan legge inn.
Max_categories	Integer(4)	Maksimalt antall kildekategorier en kunde kan legge inn.

Iword

Inneholder stikkordene i systemet. Disse legges inn av bruker via DynaDesk-systemet. Primærnøkkel autonummereres ved spørringer mot databasen.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Stikkordets ID i systemet. Primærnøkkel i tabellen.
Word	Varchar(32)	Stikkordet.

Keywordcategory

Inneholder stikkordskategoriene i systemet. Legges inn av bruker via DynaDesk.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Categoriens ID i systemet. Primærnøkkel i tabellen.
Name	Varchar(32)	Categorinavnet.

Wordselection

Inneholder hvilke stikkord en kunde abonnerer på og i hvilken kategori disse ligger. Lages utfra keywordcategory og Iword i DynaDesk.

Feltnavn	Datatype	Beskrivelse
Iword	Integer(4)	Id-en på stikkordet. Primærnøkkel i tabellen.
Csubscr	Integer(4)	Id på kunden som abonnerer på stikkordet.
Category	Integer(4)	Id på kategorien stikkordet ligger i.

Source

Inneholder informasjon om kildene i systemet. Legges inn av administrator i DynaNews-systemet.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Id-en på stikkordet. Primærnøkkelen i tabellen
Name	Varchar(32)	Navnet på kilden, eks. Aftenposten.
Urlhome	Varchar(32)	Link til kilden

Subscription

Inneholder hvilke kunder som abonnerer på hvilke kilder. Legges inn av bruker via DynaDesk systemet.

Feltnavn	Datatype	Beskrivelse
Sid	Integer(4)	Id på kilden. Delt primærnøkkel.
Cid	Integer(4)	Id på kunden som abonnere på kilden. Delt primærnøkkel.

Category

Inneholder kildekategoriene i systemet. Legges inn av bruker via Dynadesk.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Id på kategorien. Primærnøkkel i tabellen. Autonummereres.
Name	Varchar(32)	Navnet på kategorien.

Logicalselection

Inneholder informasjon om hvilke kategorier en kildene ligger i for hver enkelt bruker. Legges inn på grunnlag av Category og Subscription tabellene av DynaDesk-systemet.

Feltnavn	Datatype	Beskrivelse
Category	Integer(4)	Id på kategorien kilden ligger i. Primærnøkkel i tabellen.
Csubscr	Integer(4)	Id på kunden som abonnerer på kilden.
Csource	Integer(4)	Id på kilden.

Message

Inneholder alle nyheter hentet inn fra alle kilder i databasen. Legges inn via DynaNews-systemet.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Id på nyheten. Primærnøkkel i tabellen.
Sid	Integer(4)	Id på kilden nyheten er hentet fra.
Title	Varchar(254)	Tittelen på nyheten.
Link	Varchar(254)	Ekstern link til nyheten.
Mod	Timestamp(26)	Tidspunktet nyheten er hentet inn.
Caption	Varchar(1024)	Ingress til nyheten.

Export

Inneholder nyheter som har gitt treff i en kundes stikkord. Legges inn via DynaNews-systemet.

Feltnavn	Datatype	Beskrivelse
Id	Integer(4)	Id på nyheten. Primærnøkkel i tabellen.
Cid	Integer(24)	Id på kunden som har fått treff på nyheten.
Title	Varchar(254)	Tittelen på nyheten.
Caption	Varchar(1024)	Ingress til nyheten.
Link	Varchar(254)	Ekstern link til nyheten.
Mod	Timestamp(26)	Tidspunktet nyheten er hentet inn.
Name	Varchar(254)	Navnet på kilden nyheten er hentet fra.
URLhome	Varchar(254)	Ekstern link til nyheten.
Sourcecat	Integer(4)	Id på kategorien kilden det har vært treff på ligger i.

4.3 Grafisk brukergrensesnitt

Nettsiden er utformet etter prinsipper rundt brukervennlighet og design. Vi vil her prøve å gi en innsikt i hva slags kriterier som er lagt til grunn for utformingen av brukergrensesnittet til DynaDesk. Vi vil gi en kort innføring i hvilke grunnprinsipper vi følger, og fortelle bakgrunnen for de valg vi har tatt med hensyn på sidenes innhold og de ulike klientenhetene.

Vi har hele tiden forsøkt å lage et grensesnitt som er så brukervennlig som mulig. Brukervennlighet er en subjektiv opplevelse, og vil derfor oppleves forskjellig ut fra brukerens erfaring og kompetanse. Vi antar at vår målgruppe [se avsnitt 1.5] er godt vant til å benytte Internett og andre elektroniske medier. Med dette som utgangspunkt etterstreber vi å lage et enkelt og stilfullt design ut fra de prinsipper og råd som denne rapporten tar opp.

Det er viktig at sidene fungerer optimalt i forhold til funksjonalitet, navigasjon, interaktivitet og innhold. Vi har derfor valgt en formell stil som fokuserer på stoffet som presenteres på sidene. Sidene er tilpasset oppløsning på 800x600 på PC og 160x160 på PDA. Ved denne skjermoppløsningen, eller bedre, vil derfor DynaDesk sidene se bra ut.Våre krav er at DynaDesk skal være enkelt å sette seg inn i for nye brukere, samtidig som det skal være effektivt for avanserte brukere.

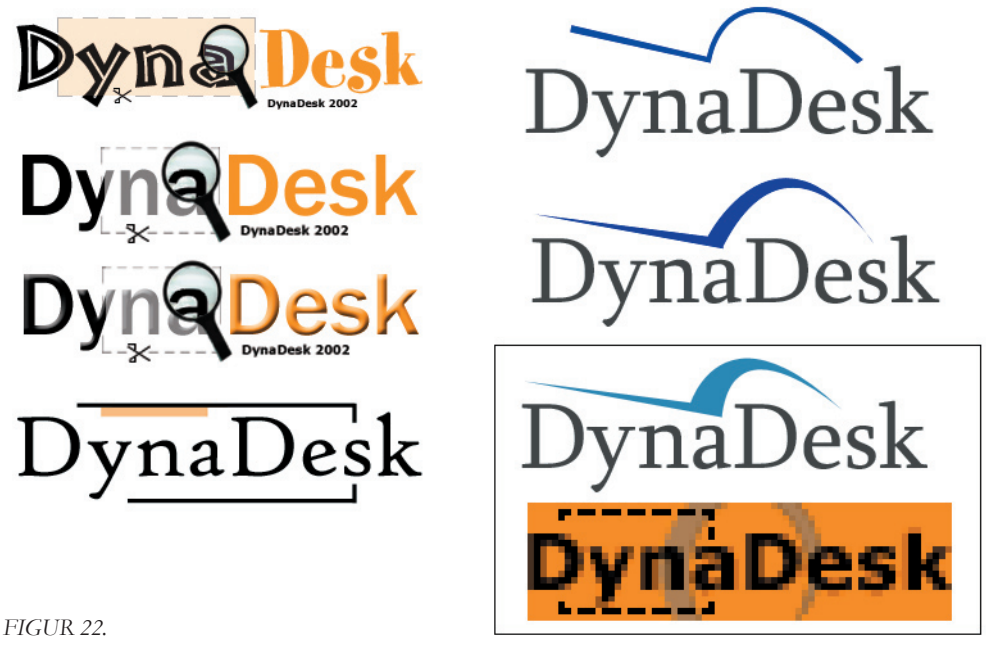
4.3.1 Logo

Logoen skal introdusere kunden til produktet vårt, og det er da fint at den er knyttet opp mot produktets mål og muligheter. Valg av bokstaver og form kan være avgjørende for at kunden skal finne produktet verdig. Vi vil skape et behagelig bilde av produktet ved å forme en logo som er optimalisert mot det vi leverer. Ved utvikling av logo må det vektlegges at den ikke skal være dominerende eller prangende. Fargevalg er også viktig, fargene skal helst ikke være så synlige at de stikker ut øynene av potensielle kunder, men ha en behagelig og imøtekommende effekt. En god logo er stilren, tidløs, lett å huske, passer i sammenhengen, og kunne brukes på flere medier.

En logo kan bestå av:

- en enkel skrifttype
- skrift kombinert med symbol
- kun et symbol.

Logoen må lages som vektorbasert grafikk. Dette gjør at den lett kan tilpasses



FIGUR 22.

forskjellige medier og størrelser uten å miste kvalitet. Vi må også ha en pixelbasert versjon for web, da i GIF-format som komprimerer uten tap av kvalitet.
Det første logoutkastet vi lagde gjenspeilte konseptet veldig direkte og brukte GAN Medias fargekode. En firkant med saks kombinert med et forstørrelsesglass skulle lede hen på redigering og søk av nyheter. Denne logoen ble forkastet på prototypestadiet fordi den ble for komplisert. En forenklet PDA-utgave er implementert for håndholdte enheter som en prøveordning.

Vårt endelige logoforslag består av skrift og et bølgesymbol. Dette symbolet skal lede hen på at nyheter samles inn, bearbeides og sendes videre. Logoen er behagelig for øyet og fargen står i fin kontrast til de oransje fargene som mye blir brukt i GAN Media-sammenheng. Det blir opp til GAN Media å vurdere hvilken logo som skal brukes. Vi foreslår å bruke tilnærmet like logoer for alle enheter og medier.

4.3.2 Innholdsdesign

I forbindelse med utforming av ett brukergrensesnitt for et nettsted er det flere prinsipper man bør følge. Første bud er at innholdet på siden må reflektere meningen med nettstedet. I vårt prosjekt er det nyheter og systemet DynaDesk som skal gjenspeiles på nettsidene. Derfor har vi ikke tatt med noe særlig informasjon om selve prosjektarbeidet eller GAN Media på sidene. Denne type informasjon finnes kun i form av fotnoter og copyrighttekster. I tillegg må man tilpasse innholdet til en målgruppe og deres forventninger.

DynaDesk sin hovedmålgruppe er nyhetsmedarbeidere, og det vil derfor være naturlig at det er nyheter som står mest i fokus. Dette gir seg utslag i at sidene kun består av lett grafikk, og at tekstene og menyene er enkle og lette å forstå. Et annet punkt man bør ha i bakhodet i sammenheng med innholdsdesign er at all tekst må sjekkes for stavefeil og gramatiske feil. Dette er et særlig viktig punkt når nettstedet er laget for kommersielt bruk slik DynaDesk er. Dårlig språk og stavefeil gir ikke akkurat inntrykk av profesjonalitet.

Et annet prinsipp er at nettsider som inneholder mye informasjon bør være søkbare. Nyhetene i DynaDesk hentes inn fra database, og vi har laget en egen side for å kunne søke i databasen etter nyheter ut i fra et søkeord. En annen ting som er viktig med hensyn på innholdsdesign er at man strukturerer informasjonen slik at når en bruker skumleser siden finner han relevant informasjon uten å måtte lese all tekst. På nettsiden til DynaDesk er dette gjort ved å dele opp informasjonen i blokker med forklarende overskrifter.

4.3.3 Tekst på Internett

En designer må i tillegg til innholdet også ta tekstformatering i betraktning når hun skal utvikle et nettsted. Ett aspekt er at fonter vises forskjellig på ulike operativsystemer. For å unngå forskjell på PC og Mac oppgir vi derfor fontstørrelser i pixler i stedet for punkt (pt). Ett annet problem er at ikke alle fonter eksisterer på alle maskiner. For å sikre en viss kontroll over fontbruken benytter vi Verdana og Arial fordi de er standard på Internett, og så godt som alle brukerene har disse fontene tilgjengelig. Samtidig definerer vi alternative fonter dersom disse fontene ikke finnes hos brukeren.

4.3.3.1 Noen generelle regler for tekst på web

- ikke fontstørrelser under 10 pixler. Brødtekst mellom 10-12 pixler.
- bruk fonter som er installert på alle maskiner.
- ha blokker med tekst som er lette å lese.
- ha kontraster mellom tekst og bakgrunn.
- bruk et begrenset antall med fonter for å skape et rent, profesjonelt design.

– ha aldri lange tekstblokker i STORE BOKSTAVER eller kursiv.
– bruk fonter som er spesielt laget for web/skjermvisning.
– bruk understreking på linker i brødtekst (når markøren glir over), slik at brukeren forstår at det er klikkbart.

4.3.4 Fargebruk

Mange webdesignere går bort fra å bruke websikre farger, ettersom mulighetene begrenser seg til 216 farger. PC-parken i Norge har blitt bedre de seneste årene, og flere og flere PC-er har god oppløsning og millioner av farger. På grunn av dette får man større frihet med tanke på valg av farger. Vi går ut i fra at brukergruppen vår sitter ved gode skjermer.

4.3.4.1 Farge på tekstelementer

– Vi har valgt svart skriftfarge, dette gir bra kontrast og leselighet.
– Feilmeldinger er røde. Rødt er noe man forbinder med varsel og fare som trekker oppmerksomhet.
– Eksterne lenker er blå. Blå er standardfargen på lenker og gir et klart signal.

4.3.4.2 Kontraster mellom tekst og bakgrunnsfarge

Vi har sørget for dempet, men god kontrast mellom tekst og bakgrunnsfarge. Teksten er primært svart på offwhite bakgrunn. For å unngå at sidene våre blir veldig «grå» har vi en header med hvit bakgrunn – dette gir kontrast til siden og letter den grå «tåken».

4.3.5 Fargevalg og harmoni

Harmoni er behagelig for øyet. Hvis vi klarer å oppnå en fargeharmoni, vil det skape visuell orden, interesse og balanse for brukeren. Hvis det ikke er harmoni, vil det enten være kaotisk eller kjedelig. Vi har bestemt oss for primært tre farger som bakgrunn:  (gammelblå, ferskengul og offwhite). Disse er valgt ut i fra hvilken kontrast de gir mot teksten og grafikken rundt. Gammelblå og ferskengul er tilnærmet komplementærfarger. Disse fargene vil skape maksimal kontrast og stabilitet. Grå er en akromatisk farge. Det betyr bl.a. at den med fordel kan anvendes sammen med andre farger fordi den ikke forstyrrer. Vi bruker grå for å få plass til og avbalansere farger – vi prøver å passe på at den ikke blir for dominerende og ser at de andre fargene påvirker oppfattelsen av den grå.

4.3.5.1 Fargesymbolisme

Blått er assosiert med eleganse, og virker beroligende. Blått gir gode, avslappende følelser. Mørke blåfarger assosieres med rikdom, styrke og respekt, og er forretningsfargen fremfor noen. Blått er populært i tider med usikker økonomi, den kalles ikke «The Colour of Trust» for ingenting.

Oransje har mye til felles med rødt; den er også en farge som roper på oppmerksomhet. Oransje er fargen for glede, energi og aktivitet. Intense oransje toner stimulerer appetitten på samme måte som rødt, men oransje er en farge det er svært vanskelig å dempe. Gjør du den mørkere får den et uklart, gjørmete uttrykk. Oransje passer godt som kontrastfarger til blått og grønt fordi fargene har samme temperatur.

Grått finnes i utallige nyanser, men kan være vanskelig å velge – den tar farge fra ting rundt seg, og kan ofte stikke mot blått, hvitt, gult eller grønt, dvs den har et skjær av en av disse fargene – en simultankontrast. Grått er rolig og nøytralt, men alene blir grått kjedelig. Grått er den store bakgrunnsfargen du nesten ikke ser, men som kan fremheve noe annet; navigasjonen eller en logo.

4.3.6 Navigasjon

Navigasjon bør gjenkjennes av bruker som navigasjon. Dette vil si at det ikke er lurt å eksperimentere for mye med for eksempel plasseringen av menylinjen i forhold til det som er allment akseptert og tradisjonelt på nettet. Vestlige brukere leser fra venstre til høyre og fra øverst til nederst. Vi har valgt å plassere menylinjen på toppen av skjermen for å fange brukerens blikk ved hjelp av en trekant og luft til venstre. Rekkefølgen på linkene i menyen er også logiske i den forstand at vi regner med at brukeren vil være ute etter å søke, for deretter å abonnere på stikkordene til sin webløsning/profil. Som en ekstrafunksjonalitet ligger artikkelskjemaet. Navigasjonen er plassert på samme sted gjennom alle websidene, og ved hjelp av gode titler, URL og markering i menylinjen skal brukeren skjønne hvilken side han er på. Vi har forsøkt å gjøre alle linker så forklarende som mulig slik at brukeren skjønner hvor han havner ved å klikke på disse. Dersom linken går til an annen side (for eksempel en avis), bruker vi navnet til denne siden som linkens navn – og blå farge. Går linken til en DynaDesk-side brukes navnet/temaet på undersiden. Brukeren skal komme til den informasjonen han er ute etter på under tre klikk.

4.3.7 Design for håndholdte enheter

Vi bruker de samme prinsippene for håndholdte enheter, men det er noen flere forhold man må ta hensyn til. Når man designer for håndholdte enheter er det en utfordring å tilpasse innholdet. Det må ikke bli for lite, slik at det blir for overfladisk, enkelt og ubrukelig – men samtidig må det heller ikke bli for mye, slik at det sprenger kapasiteten på skjermstørrelse og lastingskapasitet/tyngde. Vi må basere oss på standard og enkel HTML uten bruk av CSS, JavaScript og lignende.

4.3.7.1 Hver pixel teller

Skjermstørrelsen er meget begrenset på en håndholdt enhet. Hvis man ser bort fra mobiltelefoner, har PDA den minste skjermen, med 160x160 pixler. Når browseren har plassert tittellinje og scroller er det bare 150x140 pixler igjen! Disse må brukes respektfullt – minimalt med plass må gå til grafikk, menyer og marger.

4.3.7.2 Hver byte koster

En håndholdt enhet har begrensede muligheter for grafikk og kode. Kun enkle HTML-tagger og tabeller kan brukes, kvaliteten på kontrast og farger varierer stort og enkelte enheter har ikke engang fargeskjerm. En annen begrensinger er lastingskapasitet, siden bør ikke være mer enn 50-60kb, men allikevel stor nok til å vise så mye informasjon en trenger for at det i det hele tatt skal være nyttig.

5 Implementering

Innledning

Dette kapittelet omhandler valg og anvendelsen av programmeringsspråk, metoder og verktøy. Prinsipper og standarder som er valgt å legge seg på i arbeidet med kodingen vil bli beskrevet, og det vil trekkes frem eksempler av kildekoden. Det legges vekt på at utviklere og programmerere skal forstå hva som forklares og menes.

5.1 Presentasjon

5.1.1 Logo

Vårt endelige logoforslag er laget i Photoshop og eksportert til formatene eps (vektorbasert) og gif (bitmapbasert). Logoen er lagt inn i HTML-koden slik at den gjenspeiles på alle websider. For PDA har vi implementert en annen type logo som er en forenklet versjon av det første logoutkastet vi lagde.

5.1.2 JSP-sider

Hver JSP-side er bygget opp av mange komponenter og filer for å oppnå størst mulig fleksibilitet og skalerbarhet. Skal det lages en ekstra side kan designet lett bygges opp og tilpasses. Stiler, fonter, marger og lignende er bestemt i en ekstern CSS-fil. For PDA har vi brukt ren HTML-kode pga. PDA-ens begrensninger.

HTML-fil:

```
<LINK REL="stylesheet" href="../gfx/stiler.css" type="text/css">
```

CSS-fil:

```
A.toplink:active {
  font-size: 12px;
  font-family: verdana;
  font-weight: 600;
  text-decoration: none;
  color: #777777;
}
```

Her er et eksempel fra CSS-filen som er lenket eksternt fra HTML-koden. CSS-koden forteller hvordan linkene i menylinjen skal se ut når de er aktive.

HTML-en er delt opp i flere filer som inkluderes i JSP-filene. Dette er gjort for at alt skal kunne endres på et sted, slik at vi unngår duplisering av kode. Skal en eksempelvis bytte ut bakgrunnsfargen, gjøres dette i en fil, og fargen vil så endres i alle filer som inkluderer denne filen.

JSP-fil:

```
1  <%@ include file="top1.html" %>
2  width="80%"
3  <%@ include file="top1b.html" %>
4  <bean:message key="message.top" />
5  <%@ include file="top2.html" %>
```

Vi ser at HTML-filer er inkludert i linje 1, 3 og 5. Disse filene inneholder standard HTML-kode som brukes i alle JSP-filene. I linje to finner vi et eksempel på HTML-kode som er spesifikt for hver JSP-fil. Linje 4 henter statisk tekst fra Resource-filen for det gjeldende språket, som flettes inn i HTML-en.

5.2 DynaDesk på Strutsplattform

5.2.1 Struts

I dette avsnittet vil vi gå igjennom de mest sentrale stegene i hvordan DynaDesk fungerer sammen med Struts-plattformen. Det vil være en fordel å samtidig følge med på flytkartet for DynaDesk. (Fig. 22)

Det første som skjer når en bruker kobler seg til DynaDesk er en redirect fra index.jsp til UnitAction-klassen. UnitAction-klassen finner ut hvilken type enhet brukeren logger på med, og velger JSP-filsett basert på denne informasjonen.

index.jsp

```
<logic:redirect page="/unit.do" />
```

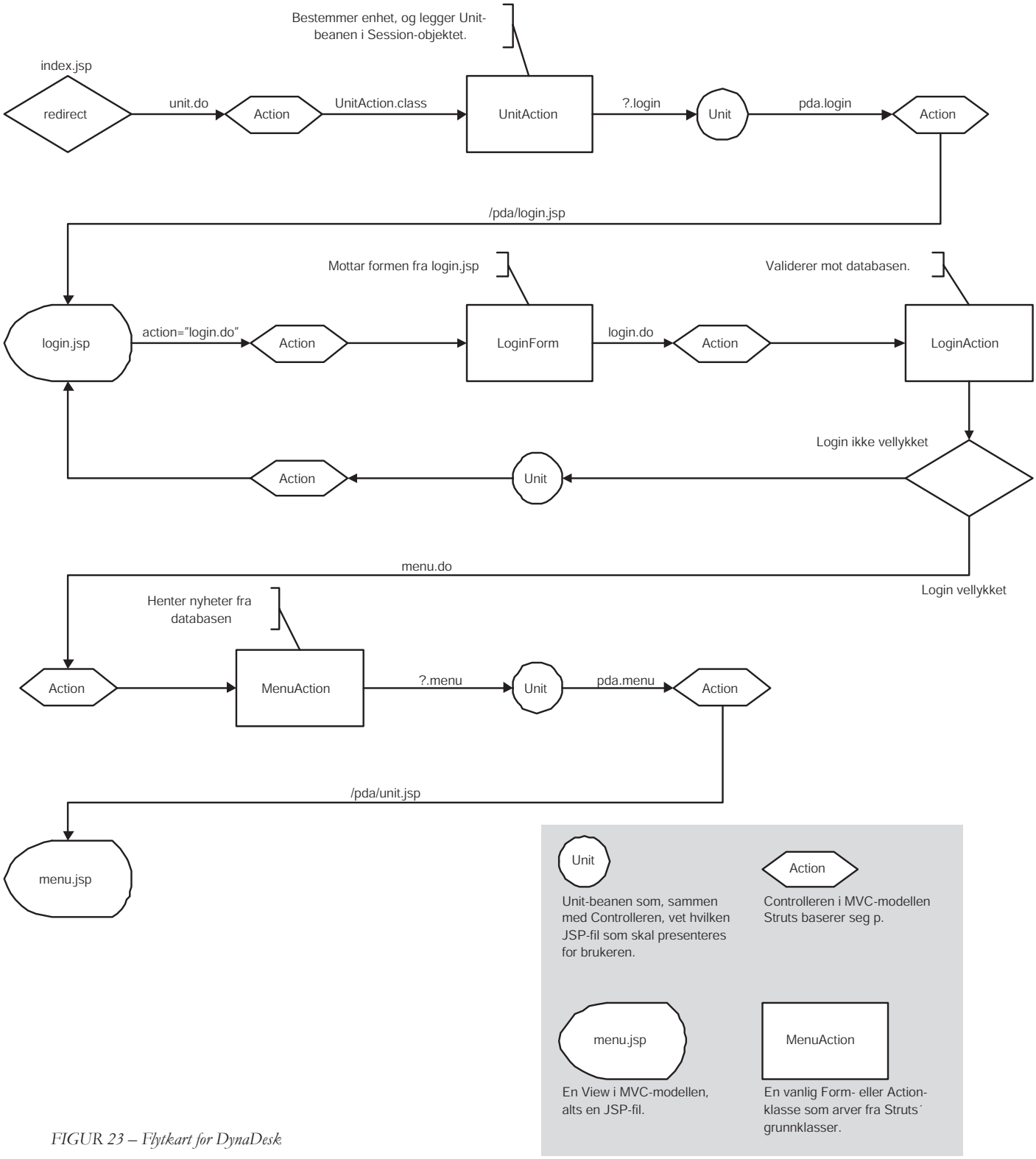
Som vist i koden over aktiviseres UnitAction-klassen med URL-en `/unit.do`. Denne URL-en kan like gjerne skrives direkte i browseren, som i en redirect i en JSP-fil. For at Struts skal kunne finne koblingen mellom URL-en og Action-klassen som skal kjøres må dette være konfigurert i struts-config.xml, som har en viktig rolle i Controller-delen av MVC-modellen Struts baserer seg på.

struts-config.xml

```
1  </struts-config>
     ...
2    <action-mappings>
3      <action path="/unit" type="no.gan.dynadesk.UnitAction" />
       ...
4    <action-mappings>
5  </struts-config>
```

struts-config.xml overtar mange av de oppgavene som en vanlig Java-servlet ville hatt i applikasjoner som ikke benytter Struts. Struts lager faktisk en servlet basert på det som står i struts-config.xml. Dette konfigureres i web.xml (linje 9-12):

```
1  <web-app>
2   <servlet>
      ...
3     <servlet-name>action</servlet-name>
4     <servlet-class>org.apache.struts.action.ActionServlet
        </servlet-class>
5     <init-param>
6       <param-name>application</param-name>
7       <param-value>no.gan.dynadesk.DynadeskResources</param-value>
8     </init-param>
9     <init-param>
10      <param-name>config</param-name>
11      <param-value>/WEB-INF/struts-config.xml</param-value>
12    </init-param>
      ...
13  </servlet>
14  <servlet-mapping>
15    <servlet-name>action</servlet-name>
16    <url-pattern>*.do</url-pattern>
17  </servlet-mapping>
    ...
18 </web-app>
```



FIGUR 23 – Flytkart for DynaDesk

Et par andre ting som skjer i web.xml er også verdt å ta med seg:
- I linje 7 får Struts vite hvor den finner propertyfilene som inneholder all statisk tekst som skal inn på websidene. Disse kan inneholde tekst i ulike språk.
- I linje 16 kommer .do-endelsen på Action-klassene, som f.eks. `/unit.do`, på plass.

Når UnitAction-klassen har avgjort hvilken type enhet brukeren logger inn med, sendes brukeren videre til login-siden som presenterer en HTML-form hvor brukeren kan logge seg inn. Fordi DynaDesk er tilpasset ulike enheter, oppsto det et problem i forbindelse med hvordan Struts-Controlleren arbeider. Normalt ville Struts ut i fra hva som er konfigurert i struts-config.xml på egen hånd greid å finne den riktige JSP-filen å presentere for brukeren, men ikke i vårt tilfelle. For å gå rundt dette, måtte vi i de ulike Action-klassene benytte Unit-beanen vår til å avgjøre hvilke JSP-filer som skal presenteres. Forenklet i UnitAction-klassen ser dette slik ut:

```
1  public ActionForward perform(ActionMapping mapping,
       ActionForm form ... ) {
2    Unit unit;
3    String userAgent = request.getHeader("User-Agent");
4    unit = new Unit(findUnit(userAgent));
5    HttpSession session=request.getSession();
6    session.setAttribute(Constants.UNIT, unit);
7    return(mapping.findForward(unit.forwardName(Constants.LOGIN)));
8  }
```

I klasser som arver fra Action-klassen – som klassen i eksemplet over gjør – vil metoden `perform()` kjøres automatisk på samme måte som `main()` i en vanlig Java-applikasjon. Når `perform()` kjøres, opprettes først en Unit-bean som vil gi Struts den informasjonen den trenger til å vise de riktige JSP-filene. Basert på `User-Agent`-stringen vi finner i request-objektet greier UnitAction-klassen å finne hvilken enhet brukeren som logger seg på benytter. Informasjonen legges i Unit-beanen som videre legges i Session-objektet. Fra Session-objektet vil også andre Action-klasser kunne benytte seg av Unit-beanen senere. I den siste linjen i eksemplet returnerer vi et ActionForward-objekt tilbake til Struts, som vil gi informasjon om hvilken JSP-fil Struts skal presentere for brukeren. Legg merke til at vi benytter Unit-beanen vi opprettet litt lengre oppe som parameter. Om det er en PC som benyttes, vil tekststrengen `forwardName` returnerer se slik ut: `pc.login`. Basert på denne verdien og en tilsvarende verdi i struts-config.xml, vil Struts finne riktig JSP-fil å sende brukeren til:

struts-config.xml

```
<struts-config>
  ...
  <global-forwards>
    ...
    <forward name="pc.login" path="/pc/login.jsp" />
    ...
  <global-forwards>
  ...
<struts-config>
```

Vi har nå greid å sende brukeren videre til `/pc/login.jsp`. Når brukeren har fylt inn brukernavn og passord og trykket submit, tar Struts imot denne formen ved hjelp av en Form-bean. LoginForm, som Form-beanen heter i dette tilfellet, har get- og set-metoder som tilsvarer feltene i HTML-formen. Disse tar imot verdiene brukeren har skrevet inn. Fordelen med Form-beans er at de på et tidlig stadie abstraherer POST og GET-metodene HTTP-protokollen benytter, slik at resten av applikasjonen vår kun forholder seg til Java-datatyper.

Igjen begrenser det – at View-laget i MVC-modellen til Struts må forholde seg til flere enheter – mulighetene våre til å utnytte funksjonalitet i Struts. I en mer vanlig sammenheng ville vi kunne bruke `validate()`-metoden i Form-beans, som kan gjøre enkle konverteringer og validerer de verdiene som er kommet inn fra formen. I vårt tilfelle med login-formen ville vi kunne sjekke om brukernavn- eller passord-feltet var tomt. Var et av feltene tomt, ville `validate()`-metoden returnert `false`, og sendt brukeren tilbake til login-formen med en feilmelding. Siden Struts krever at vi i struts-config.xml oppgir hvilken JSP-fil vi vil sende brukeren tilbake til om `validate()`-metoden returnerer `false`, vil ikke denne funksjonen være tilgjengelig for oss. Dette er fordi vi er avhengige av å bruke Unit-beanen, som vi henter fra session-objektet, for å finne hvilken JSP-fil som tilhører den påloggede enheten. På grunn av dette problemet må vi derfor utføre valideringen av de ulike formene i de tilhørende Action-klassene.

Et eksempel på hvordan det egentlig skulle sett ut i struts-config.xml ser man i linje 6, hvor `input="login.jsp"` er tatt med. I DynaDesk er denne attributten utelatt, siden vi også måtte ha tatt med `/pc/`, eller `/pocketPc/` i filnavnet – hvilket ikke vil virke, siden vi på forhånd ikke kan vite noe om hvilken type enhet som benyttes.

struts-config.xml

```
1  <struts-config>
2    <form-beans>
3      <form-bean name="loginForm"
         type="no.gan.dynadesk.LoginForm" />
       ...
4    </form-beans>
     ...
5    <action-mappings>
6      <action path="/login" type="no.gan.dynadesk.LoginAction"
        name="loginForm" validate="false" input="login.jsp" />
      ...
7    </action-mappings>
     ...
8  <struts-config>
```

I linje 3 i eksemplet over er mappingen mellom action-atributten i JSP-formen og LoginForm-beanen definert. I linje 6 ser man hvordan denne Form-beanen kobles til den relaterte Action-klassen ved hjelp av `name="loginForm"`, slik at Struts vet hvor den skal sende resultatet av det som blir tatt opp av formen. I linjen under ser man hvordan denne mappingen brukes i en form i en JSP-fil.

login.jsp

```
<html:form action="/login" focus="username">
```

Resultatet brukeren får tilbake til browseren sin som HTML ser slik ut:

login.jsp – HTML

```
1  <form name="loginForm" method="POST"
     action="/login.do;jsessionid=nbld5u2jv1">
     ...
2  </form>
3  <script language="JavaScript" type="text/javascript">
4    <!--
5      document.forms ["loginForm"].elements["username"].focus()
6    // -->
7  </script>
```

Her er det flere ting å legge merke til:

- `action="/login"` i JSP-formen er blitt oversatt til `action="/login.do"` i den resulterende HTML-formen, i henhold til det som står i `<servlet-mapping>` i web.xml. `/login.do` tilsvarer LoginAction-klassen i Model-laget som skal behandle dataene, og utføre valideringen av brukeren mot databasen.
- `name="loginForm"` tilsvarer den mappingen som er beskrevet i `<form-beans>`-taggen i struts-config.xml, slik at Struts vet hvilken Form-bean den skal bruke for å ta imot dataene; LoginForm.
- `method="POST"` er automatisk lagt til.
- `;jsessionid=nbld5u2jv1` blir lagt til bakerst på URL-en i HTML-formens action-attributt. Struts vil gjøre dette for å kunne beholde session-objektet i de tilfellene brukeren benytter en browser som ikke støtter cookies, eller har cookies slått av.
- Siden vi skrev `focus="username"` i formen i login.jsp, har Struts automatisk, i linje 3 til 7, lagt til et JavaScript som gir username-boksen i HTML-formen fokus.

Når brukeren er validert mot databasen, sendes han til neste side ved å returnere et nytt ActionForward-objekt til Struts-rammeverket:

LoginAction.java

```
return (mapping.findForward(unit.forwardName(Constants.MENU)));
```

MenuAction.java

MenuAction-klassen kobler seg til Database-servleten og henter ut en vektor med Message-beans som den sender videre til menu.jsp:

```
1  public ActionForward perform(...) {
2    Unit unit = (Unit)request.getSession().
                 getAttribute(Constants.UNIT);
3    if ( unit == null || request.getSession().isNew() )
4      return (mapping.findForward(Constants.INDEX));
5    Database database = (Database) servlet.getServletContext().
                 getAttribute("database");
6    Customer customer = (Customer)request.getSession().
                 getAttribute(Constants.CUSTOMER);
7    request.setAttribute(Constants.MESSAGES, database.
                 getMessages(customer));
8    return  mapping.findForward(
                 unit.forwardName(Constants.MENU)));
9  }
```

For å finne de riktige Message-beanene for brukeren, sender vi med customer-objektet til `database.getMessages()` (Linje 7)

Den vakre koden under er mye av grunnen til at vi valgte Struts som plattform. JSP-en som tar imot vektoren med Message-beans, og presenterer nyhetene som er inni, ser slik ut:

menu.jsp

```
1  <logic:notPresent name="unit" scope="session">
2    <logic:redirect page="/unit.do" />
3  </logic:notPresent>

4  <logic:notPresent name="messages" scope="request">
5    <logic:redirect page="/menu.do" />
6  </logic:notPresent>

7  <html:html locale="true">
8    <head>
9      <title><bean:message key="message.title" /></title>
10   </head>
```

```
11   <logic:iterate id="message" name="messages">
12     <bean:write name="message" property="title" />
13     <bean:write name="message" property="caption" />
14     <a title="
         <bean:message key="message.fullStory" />" href="
         <bean:write name="message" property="link" />
       ">
15     <bean:write name="message" property="source" />
16   </logic:iterate>

17 </html:html>
```

I linje 1 til 3 sjekker vi om brukeren har fått opprettet unit-objektet sitt enda, eller om han kom rett inn på siden f.eks via en bookmark. Brukeren sendes til UnitAction-klassen om unit-objektet mangler.

I neste blokk (linje 4-6) sjekker vi om om det eksisterer en vektor med Message-beans. Vi skal bruke denne litt lengre nede i koden, så vi må være sikre på at den eksisterer. Gjør den ikkje det, sender vi brukeren tilbake til MenuAction-klassen, så den kan hente en ny vektor fra databasen.

```
<html:html locale="true">
```

Gir beskjed til Struts om at Struts skal bruke property-filer til statisk tekst i ulike språk, som angitt i web.xml:

```
<param-value>no.gan.dynadesk.DynadeskResources</param-value>
```

I linje 9 benyttes en av propertyene i DynadeskResources som tittel på den resulterende HTML-siden med `<bean:message key="message.title" />`-taggen. På en engelsk side ville resultatet blitt:

```
<title>Latest News</title>
```

Linje 11 til 16, hvor nyhetene blir skrevet ut, er selve «rosinen i pølsa». Her ville vi normalt ha måttet gå inn med script-kode for å loope gjennom vektoren og skrevet ut beanene. I stedet har Struts en egen `<logic:iterate>`-tag som gjør dette på en mye mer elegant måte. Den henter messages-vektoren fra request-objektet, gir den en id – som vi vil se benyttes av `<bean:write>`-taggene – og looper igjennom.

`<bean:write>` peker på message-beanen som er aktiv i iterate-loopen med key-attributten, og henter ut propertyen den vil ha med property-attributten. Resultatet av `<bean:write name="message" property="title" />` kunne f.eks være «Mener solhatt ga promille»; altså tittelen i en nyhet-bean.

Dette er en liten snutt fra DynadeskResources som også har lurt seg inn i iterate-loopen: `<bean:message key="message.fullStory" />`

Kun som en illustrasjon på at absolutt all statisk tekst skal være lagt på ett sted, og på flere språk. Frem til nå har vi sett på de mest grunnleggende, men også mest sentrale delene av Struts. Vi har sett på hvordan vi bruker Struts til å kontrollere flyten i en applikasjon, hvordan vi sender kontrollen over til en ny klasse eller JSP-fil, og hvordan vi sender data mellom klasser og JSP-filer. Til slutt så vi hvordan disse dataene ble presentert for brukeren ved hjelp av en JSP-fil.

I det neste avsnittet vil vi sette mer fokus på selve DynaDesk, og trekker kun frem Struts der Struts´ funksjonalitet er sentral.

5.2.2 DynaDesk

Generelt for alle klassesettene i DynaDesk er at de har en Action-klasse, en Form-bean, og en «vanlig» bean. Det finnes unntak hvor det f.eks ikke finnes beans, det er flere ulike beans, eller det ikke finnes en Form-bean. Vi har to klasser som ikke er vanlige Struts-klasser; databaseklassen som tar seg av kommunikasjon med databasen, og en Constants-klasse som inneholder alle konstantene vi bruker i DynaDesk. Eksempler på konstanter kan være verdier som tilsvarer action-mappingene i struts-config.xml eller SQL-spørringer.

5.2.2.1 Unit-klassesettet

Dette klassesettet har vi allerede sett litt på i introduksjonen av Struts. Siden det ikke skal sendes noen form til UnitAction-klassen, har ikke Unit-klassesettet noen Form-bean. UnitAction er den klassen som alltid skal kjøres først når en bruker kommer inn på DynaDesk. Den avgjør om brukeren logger inn med en PC, PocketPC, Palm, WAP-telefon, eller andre enheter som er støttet av DynaDesk.

For å gjøre dette analyserer UnitAction-klassen `User-Agent`-stringen som response-objektet i HTTP-protokollen gir fra seg. Ut i fra hvor mange treff UnitAction får på stikkord vi har oppgitt for en type enhet, greier klassen å avgjøre hvilken enhet det er brukeren logger inn med. Under er eksempler på noen slike filtre:

```
pc.filter            = MSIE, NT
mac.filter           = Netscape, Macintosh, MSIE, Mac_PowerPC
explorer55.filter    = MSIE, NT, 5.5, 6.0
explorer55Mac.filter = MSIE, Mac_PowerPC, 5.5, 6.0
pocketPc.filter      = PPC, 3.02, CE, compatible, Mozilla/2.0
palmOs.filter        = PalmOS 3.0, EudoraWeb, Blazer, UPG1
```

Dette er hva hvert filter får som poengsum ut i fra User-Agent-stringer under:

```
Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0; T312461)
```

pc:	2 poeng
mac:	1 poeng
explorer55:	3 poeng
explorer55Mac:	2 poeng
pocketPc:	0 poeng
palmOs:	0 poeng

Som vi ser «vinner» explorer55 med 3 poeng, og dermed velges JSP-filsettet for denne enheten. Denne koden i UnitAction teller opp poengene for hvert filter:

```
1  for(int k = 0; k < filter.length; k++){
2    if(userAgent.indexOf((String)filter[k])!=-1)
3      counter[i]++;
4  }
```

`filter[]` er en array med de ulike stikkordene for hver enhet. Vi har fått tak i denne arrayen ved å tokenize de forskjellige filter-stringene. F.eks `explorer55.filter`. For hvert av disse stikkordene vi finner i userAgent, øker vi `counter[]` med ett poeng. Når vi har gitt hvert filter de poengene de skal ha, sammenligner vi de og returnerer det «seirende» JSP-filsettet:

```
1  public String getFileSet(int counter[]) throws IOException{
2    int chosenSet = 0;
3    for(int i = 0; i < CountJspSets; i++){
4      if(counter[chosenSet]<counter[i])
5        chosenSet = i;
6    }
7    return unitNames[chosenSet];
8  }
```

Til slutt opprettes Unit-beanen som lagres i Session så det kan benyttes av de andre Action-klassene:

```
1  unit = new Unit(findUnit(userAgent));
2  session.setAttribute(Constants.UNIT, unit);
```

5.2.2.2 Login-klassesettet

Dette klassesettet så vi også på i innledningen for Struts. Her vil vi legge til litt om hvordan feilmeldinger blir behandlet i Struts, og om hvordan vi validerer brukeren mot databasen.

Feilmeldinger

Struts har en egen tag til bruk i JSP-ene for å gi feilmeldinger til brukeren; `<html:errors />` Vi kan jobbe mot denne taggen både fra Form- og Action-klassene, men som nevnt tidligere har vi i DynaDesk ikke mulighet til å benytte denne funksjonen fra Form-klassene, så vi må helt ned i Action-klassene for å fylle errors-objektet med de feilmeldingene brukeren skal ha. Her er et forenklet eksempel på hvordan vi gjør dette i LoginAction-klassen:

```
1  ActionErrors errors = new ActionErrors();
2  if ((password == null) || (password.length() < 1))
3    errors.add("password",
       new ActionError("error.password.required"));
4  if ((username == null) || (username.length() < 1))
5    errors.add("username",
       new ActionError("error.username.required"));
6  if (!errors.empty()) {
7    saveErrors(request,   errors);
8    return (mapping.findForward(
       unit.forwardName(Constants.LOGIN)));
9  }
```

I koden over sjekker vi om password- eller username-feltet er tomt. I tilfelle ett eller begge er tomme, legges feilmeldinger til i errors-objektet. Legg merke til saveErrors-metoden i linje 7, rett før LoginAction returnerer tilbake til login.jsp. saveErrors lager errors-objektet i request, så det til være tilgjengelig for `<html:errors>`-taggen i JSP-en.

En annen viktig ting å legge merke til er, som i JSP-ene, at vi ikke bruker statisk tekst i feilmeldingen våre, men feilmeldinger fra DynadeskResources-propertyfilen:

```
new ActionError("error.username.required")
```

Validering mot databasen

Når begge feltene er fylt inn er neste steg å validere mot databasen. Valideringen foregår i Database-servleten, som returnerer et customer-objekt med informasjon om kunden om valideringen er vellykket. En den ikke vellykket returneres `null`.

```
1  Customer customer       = database.getCustomer(username, password);
2  if (customer == null) {
3    errors.add(ActionErrors.GLOBAL_ERROR,
       new ActionError("error.password.mismatch"));
4    saveErrors(request,   errors);
5    return (mapping.findForward(
       unit.forwardName(Constants.LOGIN)));
6  }
7  request.getSession().setAttribute(
     Constants.CUSTOMER, customer);
8  return (mapping.findForward(unit.forwardName(Constants.MENU)));
```

Vi sjekker om customer-objektet er null i linje 2, legger til en feil i errors-objektet og returnerer tilbake til login.jsp om det er tilfellet. Customer-objektet lagres i session-objektet for bruk senere (linje 7), og vi returnerer til Menu-siden; den første siden brukeren vil se når han har greid å logge inn.

5.2.2.3 Search-klassesettet

Når vi nå skal se på Search-klassesettet vil vi samtidig benytte muligheten til å se på en Form-bean. Form-beans er det Struts benytter for å abstrahere de dataene som kommer fra en HTML-form til Java-datatyper, og gi en god objektorientert måte å kommunisere dataene på. I tillegg er de et vanlig sted å legge application-logic, som å f.eks validere en form. Form-beans arver fra ActionForm. Her er de første og siste linjene i en vanlig Form-bean:

```
1  public final class SearchForm extends ActionForm {
2    private String searchString;
3    private String searchDomain;
4    private String action;
5    private String forwardName;

6    public SearchForm() {
7      super();
8    }

9    public String getSearchString(){
10     return searchString;
11   }

12   public void setSearchString(String searchString) {
13     this.searchString = searchString;
15   }
...
16   public ActionErrors validate(...) {
17     return null; // Alltid valid
18   }
19 }
```

Som man ser er dette kun deler av SearchForm-beanen. Alle beans skal ha en tom constructor, og alle Form-beans skal arve fra ActionForm. To metoder utgjør en property i en bean; en get- og en set-metode. De to metodene må henholdsvis returnere og motta samme datatype for at de skal fungere i en JSP. De kan altså ikke motta en streng, for å returne en vektor i den andre enden. `validate()`-metoden i linjene 16 til 18 er det som returneres til Struts, slik at Struts kan avgjøre om formen skal sendes videre til den tillhørende SearchAction-klassen, eller om den skal sendes tilbake til formen i search.jsp hvis dataene som er fylt inn ikke er riktige. Igjen, siden vi må forholde oss til flere typer enheter, kan vi ikke benytte oss av denne funksjonaliteten, men i stedet alltid returnere `null` som innebærer at Struts gir kontrollen videre til SearchAction-klassen. En eventuell validering vil skjer der i stedet. Under er formen i search.jsp som behandles av SearchForm-beanen:

```
1  <html:form action="search.do">
2    <html:text property="searchString" />
3    <html:select property="searchDomain" size="1">
4      <html:option value="all">
5        <bean:message key="sources.all" />
6      </html:option>
7      <html:option value="profile">
8        <bean:message key="common.menuItemMessage" />
9      </html:option>
10   </html:select>
11   <html:hidden property="action" value="search" />
12   <html:hidden property="forwardName" value="search" />
13   <html:submit><bean:message key="search.top" /></html:submit>
14 </html:form>
```

Siden `action` sender formen til `search.do` i linje 1, kan det se ut som om formen sendes direkte til SearchAction-klassen, men Struts sender den altså først til SearchForm-beanen. Der tar Form-beanen imot `<html:text>`- og `<html:select>`-taggene med de tilsvarende set-metodene med samme navn.

Når vi etterpå skal få tak på verdiene ifra SearchAction-klassen trenger vi bare å tilkalle get-metoden til den verdien vi er ute etter:

```
1  public ActionForward perform( ActionMapping mapping,
                                 ActionForm form, ... ) {
     ...
2    searchString = ((SearchForm)form).getSearchString();
3    if (searchString != null)
4      searchString = "%" + searchString + "%";

5    searchDomain = ((SearchForm)form).getSearchDomain();
6    if (searchDomain == null)
7      searchDomain = Constants.ALL;

8    forwardName = ((SearchForm)form).getForwardName();
9    if (forwardName == null)
10     forwardName = Constants.SEARCH;

11   request.setAttribute(Constants.MESSAGES,
       database.getMessages(searchString, searchDomain));
12   return (mapping.findForward(unit.forwardName(forwardName)));
13 }
```

`perform()`-metoden i Action-klassen gjør to sentrale objeter tilgjengelig for oss; ActionMapping mapping og ActionForm form. I tillegg er HttpServletRequest request og HttpServletResponse response tilgjengelig. Vi skal se på form-objektet først.

I linje 2 er form-objektet godt gjemt innimellom casting og metode-kall. Form-beanen som kommer inn via form-objektet er SearchForm-beanen, men for å faktisk kunne bruke beanen må vi caste det om til den riktige klassen. Når vi har gjort det kan vi hente verdien vi er ute etter gjennom `getSearchString()`-metoden. Vi validerer og bygger litt på den, og går videre til `getSearchDomain()`-metoden. Samme opplegget her med å caste om og validere.

`getForwardName()` vet hvilken JSP-side resultatet av søket skal sendes til. Vi må ha med denne verdien siden vi bruker søke-formen både på søkesiden og i artikkelskjemaet. Vanligvis ville vi bare brukt `Constants.SEARCH` for å avgjøre hvilken side vi skulle sende kontrollen videre til, men nå er den kun satt som standardverdi i tilfelle det skulle være sendt noe feil fra `<hidden>`-taggen i formen.

Til slutt, i linje 11, henter vi ut en vektor med Message-beans fra databasen med `searchString` og `searchDomain` som parametere, og legger vektoren i request-objektet. Verdien i `searchDomain` avgjør forøvrig om vi skal søke i alle kilder, eller kun de kildene brukeren har valgt i profilen sin.

Det var gjennomgangen av Search-klassesettet. Vi så ekstra nøye på Form-beans og hvordan de binder JSP-former og Action-klassene sammen. Vi vil snart vise litt mer av den viktige Database-servleten hvor all kommunikasjon med databasen foregår.

5.2.2.4 Profile-klassesettet

I Profile administrerer brukeren nøkkelordene DynaNews bruker for å filtrere nyheter. Man kan legge til og slette nøkkelord, og gruppere nøkkelordene i kategorier. Når vi nå dykker inn i koden, vil vi først se hvordan vi henter Profile-beanen fra databasen. Etterpå skal vi se hvordan vi bruker Profile-beanen i profile.jsp. I ProfileAction-klassen tilkaller vi `getProfile()`-metoden i Database-servleten:

```
profile = (Profile)database.getProfile(customer);
```

Her kommer hele koden for `getProfile()`-metoden. Den er litt lang, men den er et godt eksempel på hvordan vi kommuniserer med databasen, og benytter vi beans til å transportere dataene videre:

```
1  public Profile getProfile(Customer customer)
2  throws ServletException {

3    Vector keywords   = new Vector();
4    Vector categories = new Vector();
5    ResultSet resultSet;
6    PreparedStatement statement;
7    RdbmsHelper rdbms;

8    try {
9      rdbms = RdbmsHelper.connect(database, null);

10     statement = rdbms.getPreparedStatement(
11       Constants.SQL_SELECT_KEYWORDS);
12     statement.setObject(1, customer.getId());
13     resultSet = statement.executeQuery();
14     while (resultSet.next()) {
15       keywords.add(
16         new Keyword(
17           resultSet.getString("word"),
18           new Integer(resultSet.getInt("iword")),
19           new Integer(resultSet.getInt("category"))
20         )
21       );
22     }

23    statement = rdbms.getPreparedStatement(
24      Constants.SQL_SELECT_CATEGORIES);
25    statement.setObject(1, customer.getId());
26    resultSet = statement.executeQuery();
27    while (resultSet.next())
28      categories.add(
29         new Category(
30           resultSet.getString("name"),
31           new Integer(resultSet.getInt("id"))
32         )
33       );

34     statement.close();
35     return new Profile(keywords, categories);
36   }

37   catch (SQLException e) {
38     throw new ServletException(e.getMessage());
39   }
40   finally {
41    if (rdbms != null) {
42       try {
43         rdbms.close();
44       } catch (GANSQLException e) {
45         throw new ServletException(e.getMessage());
46       }
47     }
48   }

45 }
```

De første linjene i metoden er greie. Vi oppretter objektene vi skal jobbe med og kobler til databasen. Siden vi bruker SQL-spørringer med parametere, må vi bruke en PreparedStatement til å kjøre spørringen i stedet for Statement, som vi kunne brukt i om vi bygde opp SQL-setningen i koden. I linje 12 og 13 setter vi parameteren som sier hvilken kundes profil vi skal hente, og kjører spørringen mot databasen. I linje 14 til 22 går vi gjennom resultatet vi fikk fra databasen, og legger Keyword-beans i keyword-vektoren. Vi gjør det samme for kategoriene i linje 23 til 33. Til slutt lukker vi statementen og legger keywords- og categories-vektorene i en ny Profile-bean. Profile-beanen returnerer vi til profile-objektet i ProfileAction-klassen, og da er sirkelen sluttet. Helt til slutt fanger vi opp eventuelle feil, og lukker koblingen mot databasen.

I ProfileAction-klassen legger vi profile-beanen i request-objektet og sender kontrollen videre til profile.jsp. I profile.jsp finner man tre former. En til å legge inn nye stikkord i en eksisterende eller ny kategori, en til å velge hvilken kategori man skal slette stikkord fra, og en siste til å slette stikkord. Vi skal gå i mer detalj på den første av disse formene:

```
1  <html:form action="profile.do">
2    <html:text property="newKeyword" />
3    <logic:present name="profile" property="categories"
4     scope="request">
5      <bean:define id="keywordCategories" name="profile"
6       property="categories"/>
7      <html:select property="categoryId" size="1">
8        <html:options collection="keywordCategories"
9         property="categoryId" labelProperty="category" />
10     </html:select>
12     <bean:message key="profile.newCategory" />
13   </logic:present>
14   <html:text property="newCategory" />
15   <html:hidden property="action" value="newKeyword" />
16   <html:submit>
17     <bean:message key="profile.add" />
17   </html:submit>
18 </html:form>
```

`<html:form>`-taggen har vi sett på før, så den går vi ikke nærmere inn på. `<html:text>`-taggen i linje 2 lager en vanlig `<input type="text">`-HTML-tag.

I linje 3 og 12 ligger et `<logic:present>`-element. Dette sjekker om categories-vektoren med category-beans ligger i profile-beanen. Profile-beanen finner vi der vi la den; i request-objektet. Om den finner categories-vektoren kjører den koden som ligger i elementet.

Det er tilfeller hvor vi ikke kan bruke et objekt direkte fordi det ligger inni et annet; da må vi bruke `<bean:define>`-taggen for å gjøre dette tilgjengelig. I linje 4 henter vi categories-vektoren fra profile-beanen og legger dette i et nytt objekt; keywordCategories. `<html:select>`-elementet starter i linje 7. Resultatet av denne linjen i HTML-formen ser slik ut: `<select name="categoryId" size="1">`. Det eneste som er verdt å merke seg her, er at propery i JSP-formen tilsvarer name i HTML-formen. Mer spennende blir det når vi går over til `<html:options>`-taggen i linje 7. Her looper taggen på egenhånd gjennom keywordCategories-vektoren, og henter ut categoryId og category-navnet fra Category-beanene som ligger inni. Resultatet i HTML-formen blir slik:

```
<option value="3">Innenriks</option>
<option value="4">Sport</option>
<option value="6">Teknologi</option>
```

Til slutt kommer en ny tekstboks, en `<hidden>`-tag og submit-knappen. `<hidden>`-taggen benytter vi for å avgjøre hvilken av de tre formene på siden som er sendt.

Vi beskriver ikke Form-beanen igjen, men går i stedet rett til hvordan vi behandler dataene fra Form-beanen i ProfileAction-klassen.

Først avgjør vi hvilken form som er sendt (success-variablen er ikke i bruk):

```
1  if ( ((ProfileForm)form).getAction().equals(
2         Constants.PROFILE_INSERT_KEYWORDS) )
3    succsess = addKeyword();
4  if ( ((ProfileForm)form).getAction().equals(
5         Constants.PROFILE_SELECT_CATEGORY) )
6    succsess = chooseCategory();
7  if ( ((ProfileForm)form).getAction().equals(
8         Constants.PROFILE_DELETE_KEYWORDS) )
9    succsess = deleteKeyword();
```

Vi ser av `<hidden>`-taggen at vi har sendt newKeyword-formen, og tilkaller derfor `addKeyword()`-metoden:

```
1  private boolean addKeyword() throws ServletException {
2    Vector newKeywords =
3      (((ProfileForm)form).getKeywordsToInsert());
4    Integer categoryId = (((ProfileForm)form).getCategoryId());
5    if (newKeywords.isEmpty())
6      return false;
7    String newCategory =(((ProfileForm)form).getNewCategory());
8    if( (newCategory == null) || (newCategory.length() < 1) ){
9      try{
10       this.categoryId = categoryId;
11       database.insertKeywords(newKeywords, categoryId,
                                 customer);
12     }
13     catch (Exception e) {
14     errors.add(ActionErrors.GLOBAL_ERROR,
15       new ActionError("error.generic", e.getMessage()));
16     return false;
17     }
18   }
19   else {
20     try{
21       categoryId = database.insertKeywords(
22         newKeywords, newCategory, customer);
23     }
24     catch (Exception e) {
25       errors.add(ActionErrors.GLOBAL_ERROR,
26         new ActionError("error.generic", e.getMessage()));
27       return false;
28     }
29   }
30   profile = (Profile)database.getProfile(customer);
31   profile.setCategoryId(categoryId);
32   ((ProfileForm)form).setCategoryId(categoryId);
33  return true;
34 }
```

Vi henter først en vektor med de nøkkelordene som skal legges inn i databasen, og id-en til kategorien de evt. skal legges inn i, fra ProfileForm-beanen. Vi henter også et evt- nytt kategorinavn. Om vi har fått de verdiene vi trenger, sender vi vektoren med Keyword-beanene, categoryId-en og customer-objektet som parameter til `insertKeyword()`-metoden i Database-servleten. Metoden i servleten looper igjennom vektoren, og bruker en vanlig `INSERT INTO`-setning for å legge verdiene inn i databasen. Det er to overloads for `insertKeywords()`-metoden. En for nøkkelord som skal legges inn i en eksisterende kategori, og en for de som skal legges inn i en ny.

Det var en rask titt på Profile-klassesettet. Vi så hvordan data ble hentet fra databasen, og hvordan vi fylte komponenter i JSP-en med disse dataene via

Profile-beanen. Vi så også hvordan vi behandlet dataene som ble sendt tilbake til ProfileAction-klassen. Funksjonaliteten for å slette nøkkelord og velge kategori som skal vises er veldig lik det vi gikk gjennom, det blir derfor ikke beskrevet nærmere.

5.2.2.5 Sources-klassesettet

Dette er klassene som administrerer hvilke kilder kunden har. Kunden kan velge om alle nyhetene i kilden skal tas med eller om de skal filtreres på nøkkelordene kunden har oppgitt i profilen sin. Kunden kan også fjerne kilder, eller flytte kildene til en annen kategori. Funksjonaliteten i disse klassene har en del likhetstrekk med Profile-klassesettet, så vi vil kun trekke frem det som er spesielt for Sources-klassesettet.

Etter at vi har fått de kildene brukeren abonnerer på fra databasen, bruker vi koden under til å finne ut hvilke kilder kunden allerede har valgt ikke skal filtreres på stikkord.

```
1  sourceList = sources.getSources();
2  Vector selectedFullFeeds = new Vector();
3  for (int i = 0; i < sourceList.size(); i++){
4    if ( ((Source)sourceList.get(i)).getFullFeed()
       .equals(new Integer(1)) ){

5      selectedFullFeeds.add(((Source)sourceList.
         get(i)).getSourceId().toString());
6    }
7  }
8  ((SourcesForm)form).setSelectedFullFeeds(
     selectedFullFeeds.toArray());
```

I linje 3 til 6 går vi igjennom samtlige kilder kunden abonnerer på og undersøker hvilke som har FullFeed lik 1. Kildene som har FullFeed skal ikke filtreres på nøkkelord. Verdiene vi får legges rett i Form-beanen (linje 8), slik at disse vises som allerede valgte når vi presenterer JSP-filen for kunden:

```
1  <html:form action="sources.do">
2    <html:multibox property="selectedFullFeeds">
3      <bean:write name="sources" property="sourceId" />
4    </html:multibox>
5    <html:hidden property="selectedFullFeeds" value="true" />
6  </html:form>
```

Vi kjenner igjen selectedFullFeeds fra SourcesForm i linje 2 i JSP-en. Denne henter altså hvilke checkbokser som skal være valgt fra getSelectedFullFeed i SourcesForm når kunden kommer inn på Sources-siden. I linje 3 henter vi samtlige Sources kunden abonnerer på – også de som er filtrert på nøkkelord.

Struts har en liten bug som slår til når kunde vil ha alle nyhetene filtrert på nøkkelord, og dermed fjerner avhukingen på samtlige checkbokser. Struts sender tilbake den samme arrayen med `selectedFullFeeds` tilbake til Form-beanen, i stedet for en tom array som en skulle forvente. For å løse dette har vi lagt til en `<hidden>`-tag som lurer Struts til å sende med en dummy-verdi i arrayen, slik at denne aldri er helt tom. Denne verdien blir ignorert av rutinene vi har i Form-beanen:

```
1  public Vector getSourcesToUpdate(){
2    ...
3    for(int i=0; i<selectedFullFeeds.length; i++){
4      selectedFullFeedsHashtable.put(
         (String)selectedFullFeeds[i], new Integer(1));
5    }
     ...
6    for(int i=0; i<sourcesOld.size(); i++){
7      newCatgoriesHashtable.put(
         (String)((Source)sourcesOld.get(i)).getSourceId().
           toString(), new Integer(categoryId[i])
```

```
        );
8     }

9     for(int i=0; i<sourcesOld.size(); i++){
10      fullFeedInteger = (Integer)selectedFullFeedsHashtable.get(
          ((Source)sourcesOld.get(i)).getSourceId().toString()
        );
11      if(fullFeedInteger.equals(null))
12        fullFeedInteger = new Integer(0);
13      newCategories = (Integer)newCatgoriesHashtable.get(
          ((Source)sourcesOld.get(i)).getSourceId().toString()
        );
14      try{
15        sourcesNew.add(new Source(
            ((Source)sourcesOld.get(i)).getName(),
            ((Source)sourcesOld.get(i)).getSourceId(),
            newCategories,
            fullFeedInteger,
            ((Source)sourcesOld.get(i)).getUrl()
          ));
16      }catch (Exception e){}
17    }
      ...
18    for(int i = 0; i < sourcesOld.size(); i++){
19      oldSources = (Source)sourcesOld.get(i);
20      newSources = (Source)sourcesNew.get(i);

21      if(oldSources.getSourceId().equals(
          newSources.getSourceId())&&
        !( oldSources.getFullFeed().equals(
          newSources.getFullFeed())
        &&(oldSources.getSourceCategoryId().equals(
          newSources.getSourceCategoryId())))
        )
22        sourcesToUpdate.add((Source)sourcesNew.get(i));
23    }
24    return sourcesToUpdate;
25  }
```

I linje 3 til 5 oppretter vi en hashtabell med `sourceId` for de kildene som brukeren har valgt skal ha full feed. Det samme gjør vi for `categoryId` i linje 6 til 8. Videre i linje 15 legger vi nye Source-beans i `sourcesNew`-vektoren med de nye verdiene brukeren har sendt med formen. Vi undersøker hvilke sources som er endret etter at formen er sendt, (Linje 21), og returnerer disse til SourcesAction-klassen, som sender vektoren videre til Database-servleten for oppdatering mot databasen.

5.2.2.6 AddSources-klassesettet

Kunden kan legge til nye kilder profilen sin, bestemme hvilken kategori denne skal ligge i – eventuelt opprette en ny. Kunden velger også om kilden skal filtrerers på nøkkelordene i profilen eller ikke. Nedenfor ligger koden i AddSourcesForm-beanen som finner ut hvilke sources som skal legges til:

```
1  public Vector getSourcesToAdd(){
2    for (int i=0; i<selectedSources.length; i++)
3      selectedSourcesHashtable.put(
         new String (selectedSources[i]), new Integer(1));

4    for(int i=0; i<selectedFullFeeds.length; i++)
5      sourcesFullFeed.put(
         new String (selectedFullFeeds[i]), new Integer(1));

6    for(int i=0; i<sources.size(); i++){
7      if(sourcesFullFeed.get(((Source)sources.get(i)).
           getSourceId().toString())!=null){
8        isFullFeed = new Integer(1);
9      }
10     else
11       isFullFeed = new Integer(0);
```

```
12      if(selectedSourcesHashtable.get((
          ((Source)sources.get(i)).getSourceId()).
          toString())!=null){
13          sourcesNew.add(new Source( ((Source)sources.get(i))
              .getSourceId(), new Integer(categoryIds[i]),
              isFullFeed)
            );
14      }
15    }
16    selectedSourcesHashtable.clear();
17    sourcesFullFeed.clear();
18    return sourcesNew;
19  }
```

I linje 2 og 3 finner vi hvilke kilder som skal legges til i kundens profil. I linje 4 og 5 finner vi hvilke av disse kildene som ikke skal filtreres på nøkkelordene i profilen. Til slutt lager vi Source-beanene som inneholder informasjon om hvilken source som skal legges inn, i hvilken kategori den skal ligge i, og om den skal filtreres eller ikke. Disse beanene legges i en vektor og returneres AddSourcesAction-klassen, som sender den videre til databasen. Om noen av kategoriene har fått tallet 0 fra formen, vil Database-servleten opprette en ny kategori med det navn som oppgitt i formen. Slik gjøres dette i Database-servleten:

```
for(int i=0; i<sources.size(); i++){
  sourceToAdd = (Source)sources.get(i);
  if( sourceToAdd.getSourceCategoryId().
      compareTo(new Integer(0)) == 0)
    newCategory = true;
}
```

5.2.2.7 Article-klassesettet

Det meste av fuksjonaliteten i Article skjer på klientsiden. Klassesettet henter verdiene som blir sendt med formen og sender innholdet videre til de eksport-alternativene som er valgt; lagre til disk eller send som e-post.

ArticleAction har kun en funksjon: Hente framesettet som utgjør artikkel-skjemaet. I dette framesettet ligger ArticleEdit-klassesettet som tar imot formen som blir sendt på vanlig måte.

Vi kommuniserer med MSHTML-komponenten med JavaScript. Alle kommandoer utføres på markert tekst. Grensesnittet er bygget opp i CSS, med lag som kan skjules og vises etter behov. Et eksempel på hvordan JavaScriptet kommuniserer med komponenten:

```
document.write('<img src="right.gif" class="btn"
  onmouseup="document.execCommand("JustifyRight");">');
```

Denne linjen sier at den markerte teksten i editoren skal bli høyrejustert når man trykker på ikonet for høyrejustering. ArticleEdit avgjør hvilke eksport-alternativer som er valgt, og aktiviserer disse.

Lagre til disk

Når vi lagrer til disk legges innholdet i request-objektet, og vi sender kontrollen videre til saveToDisk.jsp. ArticleEditAction-klassen sender videre:

```
1  request.setAttribute(Constants.ARTICLE,
     new Article( ((ArticleEditForm)form).getArticleText() ) );
2  return (mapping.findForward(unit.forwardName(
     Constants.ARTICLE_SAVE_TO_DISK)));
```

I saveToDisk.jsp har vi satt mime-typen til å være ukjent, slik at browseren gir brukeren en «Save As»-dialogboks. Grunnen til at vi bruker scripting i stedet for en tag i denne sammenhengen er at Struts automatisk konverterer markup-tegn til HTML-

entiteter, noe som ikke er ønskelig i denne sammenhengen siden vi vil ha source-koden til artikkenen vi har skrevet.

```
1  <%@ page language="java" contentType="none" %>
2  <%=((no.gan.dynadesk.Article)request.getAttribute("article")).
     getArticleText()%>
```

Sende e-post

`sendMail()` er en metode som ligger i ArticleEditAction-klassen, og tar blant annet artikkenen som parameter for den mailen som skal sendes. Koden sier vel det meste:

```
1  private void sendMail(String subject, String body,
                         String toAddress) {
2    try {
3      Properties props = System.getProperties();
4      props.put("mail.smtp.host", "127.0.0.1");
5      Session session = Session.getDefaultInstance(props, null);
6      MimeMessage message = new MimeMessage(session);
7      message.setFrom(new InternetAddress(
         "Gan Media - DynaDesk <dynadesk@gan.no>"));
8      message.addRecipient(message.RecipientType.TO,
         new InternetAddress(toAddress) );
9      message.setSubject(subject);
10     message.setText(body);
11     Transport.send(message);
12   } catch (MessagingException e) {
13     addError("Feil ved sending av e-post");
14   }
15 }
```

I `props` angir vi de parameterene vi trenger for å finne SMTP-serveren vi skal bruke for å sende mailen. Videre oppretter vi et message-objekt som inneholder selve artikkelen, subject, to, from, osv. `Transport.send(message);` sender mailen til e-postadressen som er angitt som `RecipientType.TO`

Det var det siste, og kanskje det enkleste klassesettet i DynaDesk. Vi har lagret artikkelen til disk, og vi har sendt den som en e-post.

5.3 Bakgrunn for verktøyvalg

GAN Media bruker Linux-serverere med Apache internettjener i sin eksisterende løsning, og det var derfor naturlig og etter ønske fra GAN Media at vi benyttet dette i løsningen vår. I prosjektperioden satt oppdragsgiver opp en egen server som bare prosjektgruppen benyttet. Dette ble gjort for å unngå at vi underveis i utviklingen av løsningen skulle ødelegge for DynaNews. Serveren ble driftet av en kontaktperson ved GAN Media.

Selve programmeringen har foregått i UltraEdit, og ikke i JBuilder som opprinnelig planlagt. Det tok en god stund før vi fikk tilgang på JBuilder fra skolen, og hele gruppen var da godt i gang med programmeringen i UltraEdit. Siden alle i gruppen var godt kjent med dette programmet valgte vi å fortsette med det istedetfor å bruke tid på å bytte.

Til utvikling av nettsidene ble det også brukt UltraEdit. Alle gruppemedlemmene har god erfaring med HTML, så ingen hadde behov for å benytte oss av HTML-editorer. Adobe Photoshop og Illustrator ble brukt til bildebehandling.

Som prosjektstyringsverktøy har vi brukt Visio. Dette har hovedsakelig begrenset seg til utforming av Gantt-skjema og ansvarsfordeling.

5.4. Prinsipper vi har fulgt

Arbeidsoppgavene innen hver fase ble fordelt mellom gruppemedlemmene. For å få ny tankegang på utviklingen ble det til tider også omrokkert på sammensetningen av gruppene gjennom av fasene.

Hver funksjon ble utarbeidet ut fra kravene spesifisert i kravspesifikasjon og sydd sammen. Videre ble de vurdert av veileder og oppdragsgiver, og forslag til forbedringer ble tatt imot, deretter ble endringer og oppdateringer utført.

Ved starten av prosjektet bestemte vi oss for et sett med regler for navngiving i programkoden. Disse er de samme som Sun anbefaler for Java-språket:

- Alle navnene skal være så beskrivende som mulig.
- Alle konstanter skal kun bestå av store bokstaver.
- I metode- og variabelnavn skal det være små bokstaver i begynnelsen av navnet, og alle de påfølgende ordene i navnet skal ha stor forbokstav.
- Klassenavn har stor forbokstav.
- Pakker har kun små bokstaver.

5.5 Beskrivelse av forløp

GAN Media opprettet en egen server med kopi av DynaNews-databasen, og på denne la de DynaNews-løsningen slik at vi kunne studere funksjonaliteten. Etter en periode med studie av klassene i DynaNews, samt en oppfriskning av Java-kunnskapene satte vi igang med to funksjoner; gjenkjenning og listing av profilnyheter. Da disse var gjort begynte utvikling av design, rapport og artikkelskjema parallelt. Utviklingsmessig fortsatte prototypingen med endring av profil. På dette punktet avgjorde vi at vi skulle benytte Struts. Det som da allerede var utviklet måtte bygges om til å følge Struts-rammeverket. Med Struts kom tilpassning til språk, og vi la fortløpende inn støtte for bokmål, nynorsk og engelsk. Vi fortsatte med administrasjon av kilder og stikkord samt utvikling av søkefunksjoner.

6 Testing og kvalitetssikring

Innledning

Kapittelet viser hvordan gruppen har testet systemet underveis i utviklingen og hvordan kvalitetsikringen av applikasjonen og prosjektgjennomføringen har blitt ivaretatt.

6.1 Organisering av kvalitetssikring

I dette avsnittet vil vi se på kvalitetsarbeidet med utgangspunkt i de punkter som ble beskrevet i kravspesifikasjonen, spesielt under punkt 2.7 – rutniner for kvalitetssikring. Vi vil gjøre en vurdering av arbeidet som har blitt gjort for å sikre kvaliteten på produktet, arbeidsmetoder og fremdrift i prosjektet.

6.1.1 Dokumentasjon og versjonsstyring

Alle Java-klasser ble dokumentert underveis i form av kommentarer til koden både for vår egen del, og som hjelp for de som senere eventuelt skal videreutvikle systemet. Disse kommentarene brukes for å generere Javadocs (se vedlagte CD). Å benytte Javadocs for å dokumentere kodedelen av prosjektet har vært et viktig kvalitetssikrings-verktøy for selve produktet. Å se våre egene kommentarer generert som Javadocs, som vanligvis er skrevet på en oversiktlig og profesjonell måte, har bevisstgjort oss på å lage forståelige og gode kommentarer. Videre har Javadocs-ene hjulpet oss til å avdekket feil, mangler og unødvendige elementer i koden fordi de har gitt oss en helt annen oversikt over funksjonene i de enkelte klassene. Hvis GAN Media skal videreutvikle applikasjonen vil det være en fordel å ha denne oversiktlige dokumentasjonen av koden. Det er også en god dokumentasjon av klassene vi har laget for vår egen del og for rapporten sin del. Vanligvis blir Javadocs kommentert på engelsk. Vi har valgt å skrive disse på norsk fordi vi har en norsk målgruppe for disse dokumentene.

Prosjektets nettside med prosjektdagbok har fungert som et viktig dokumentasjons- og kvalitetssikringsverktøy. Vår prosjektweb tok utgangspunkt i et system som er utviklet av GAN Media. Dette nettstedet har hjulpet oss til å ha kontroll over hva som har blitt gjort når av hvem, og tidsforbruket som har blitt brukt på de ulike oppgavene. Dette har også vært nyttig for å bevisstgjøre gruppen på hvor lang tid de forskjellige oppgavene faktisk har tatt. Dette har vi dratt nytte av videre i prosjektet og har øket vår evne til å vurdere hvor lang tid en oppgave vil ta. Vi har hele tiden fullt opp prosjektdagboken i forhold til det som ble planlagt og illustrert i Gantt-skjema [se vedlegg A].

Gruppen ble i begynnelsen av prosjektet tildelt et fellesområde til lagring av høgskolen. Dette området har fungert som et backup- og rapportlagrings-område for gruppen. Området har blitt strukturert slik at det har vært enkelt å finne frem til dokumentene. Vi delte inn i følgende hovedmapper: backup, brev, dokumentasjon, kode, programvare, rapporter og referater. Deretter ble disse inndelt igjen i undermapper etter behov.

Navngiving av rapporter, referater og kode har fulgt forhåndsbestemte regler for å skille de ulike versjonene fra hverandre. De har ligget i bestemte mapper med beskrivende navn, og filen har vært navngitt med år, måned og dag. Eksempelvis Y02M10D31, som betyr 31. oktober 2002.

6.1.2 Verktøy

Til koding og utvikling av nettsidene benyttet vi UltraEdit og ikke JBuilder som planlagt i kravspesifikasjonen. (jfr. punkt 2.6.4 og 5.2). UltraEdit ble også benyttet til utvikling av nettsidene. UltraEdit er en multispråk-editor som gir god oversikt over kode, med nummerering av linjer, fargevalg og har mulighet til å ha flere filer åpne samtidig. Spesielt det å kunne lagre direkte til FTP har vært nyttig, siden vi hele

tiden har jobbet mot serveren hos GAN Media. UltraEdit har ikke noen innebygd debuggings-funksjon, så vi utviklet egne debuggings-funksjoner direkte i koden. Gruppen syns dette har fungert på en god måte.

For å unngå pixelering er alle illustrasjoner og figurer i rapporten laget i vektorgrafikk, laget med Adobe Illustrator. All grafikk på nettsidene er laget i Adobe Phototshop, et verktøy som gir god kvalitet for web. I backup-rutinene ble Second Copy 2000 og FTP NetDrive benyttet. Til generering av Javadocs brukte vi Forte for Java CE.

6.1.3 Backup

I den mest hektiske perioden i prosjektet ble det automatisk tatt backup av koden og rapporten hver sjette time. På en PC hjemme hos en av gruppens medlemmer ble FTP NetDrive koblet til serveren vi hadde hos GAN Media, slik at Second Copy 2000 kunne kopiere både disse filene og filene vi har på hovedprosjekt-serveren til HiG. I tillegg til å ta backup av alle filene, har Second Copy 2000 mulighet til å ta vare på endringer som er gjort siden forrige backup, slik at vi kunne gå tilbake om vi gjorde noen alvorlige feil vi ikke kunne rette opp selv. I tillegg til at Second Copy 2000 gjorde disse versjons-backupene, gjorde vi det samme manuelt hver gang vi følte vi hadde nådd et mål eller skulle gjøre omfattende endringer i koden eller rapportene våre.

6.1.4 Problemrapportering og tiltak

De retningslinjer som ble definert i punkt 2.7.4 om tiltak og rapportering i forbindelse med visse feilsituasjoner har blitt fulgt opp. Under utviklingen av produktet ble det lagt spesiell vekt på å skrive utfyllende statusrapporter til hver milepæl [se vedlegg F]. Disse statusrapportene gjorde at det ble enklere å sette sammen hovedrapporten mot slutten av prosjektet. Disse har fungert som en jevnlig og god dokumentsajon på tilstanden i prosjektgruppen og på produktet, samt som en rapportering til veileder og oppdragsgiver.

Alle problemer prosjektgruppen har hatt i løpet av prosjektet er dokumentert i disse, og i perioden etter at statusrapportene har vært fremlagt har vi fulgt opp dette og lett etter årsaker til disse problemene. Problemer og avvik som ikke har hatt utslag på produktet eller på gruppens arbeid har kun blitt dokumentert. Ved mer kritiske problemer og avvik har vi lagt vekt på å rette opp disse ved avviksbehandling og korrigerende tiltak. Eksempler på avviksbehandling i prosjektet har vært å utvide tidsfrister. For eksempel ble tidsfristen for milepæl endret to ganger i løpet av prosjektet. Dette ble rapportert til veileder og oppdragsgiver i statusrapportene. Eksempel på korrigerende tiltak har vært å legge om systemet til Struts for å hindre JSP-scripting i koden, og for å gi flerspråk-støtte.

6.2 Kvalitetssikring av produktet og testing

Vi har valgt å ta utgangspunkt i V-modellen for kvalitetssikring av produktet. Denne modellen går ut på å teste hver modul for seg selv; deretter en integrasjonstest der modulene testes sammen. Til slutt foretok vi en systemtest der vi så på systemet som en helhet og sammenliknet den opp mot kravspesifikasjonen. Denne formen for testing gjør det enklere å oppdage og utbedre feil. Utførsel av testingen er i hovedsak foretatt av gruppens medlemmer etterhvert som de ulike funksjoner og moduler er blitt ferdigutviklet. I tillegg har både oppdragsgiver og veileder testet produktet flere

ganger under utviklingsprosessen, og kommet med mange gode tilbakemeldinger som igjen har medført endringer/utbedringer av produktet. Til slutt ble systemet testet av utenforstående brukere med noe internetterfaring, og ut fra deres synspunkter ble produktet evaluert enda en gang. De krav og valg som avvek fra kravspesifikasjonen ble diskutert i gruppen og med oppdragsgiver før de eventuelt ble implementert i løsningen.

Testingen har hovedsaklig blitt utført på PC. GAN Media lot oss låne en Pocket PC, og denne har blitt benyttet i alle testfasene; fra enkeltfunksjoner til det integrerte systemet. Hovedforskjellen mellom testingen på PC og på PDA har vært brukergrensesnittet. Siden vi visste om de tekniske begrensningene til PDA i forhold til PC på forhånd, har testene hovedsakelig dreid seg om design og grafikk. Så lenge funksjonaliteten fungerete på PC ville den fungere på andre enheter. Vi har også testet kontinuerlig på Mac, og på en Palm-emulator.

6.3 Kvalitetssikring av prosjektprosessen

Kontakt med oppdragsgiver ble opprettholdt ved møter annenhver uke i tillegg til kommunikasjon via e-post og andre elektroniske medier. Alle avtalte møter ble avholdt, og statusrapporter ble levert i henhold til revidert tidsplan. Frem til første milepæl var nådd hadde vi ukentlige møter med veileder. Vi valgte å ha møtene så ofte for å bli kjent med veileder, og for få hjelp til å peile gruppen inn på rett spor. Det var også et behov for tett oppfølging i denne perioden. Senere i prosessen ble det holdt møter etter innlevert statusrapport og ved behov, i snitt ble det holdt møter annenhver uke. I statusrapportene ble det gitt en grundig beskrivelse av hvilke funksjoner som skulle utvikles i løpet av milepælen, hva som var utført og hvordan det var utført. Rapportene ga også en god indikasjon på hvordan gruppen lå an i forhold til de tidsrammer som ble satt opp i Gantt-skjemaet. All aktivitet ble loggført i prosjektdagboka på prosjektweben.

7 Diskusjon av resultat

Innledning

Dette kapittelet omhandler de resultater vi har kommet frem til gjennom arbeidet med hovedprosjektet.

7.1 Produktet

7.1.1 Hvordan innfrir løsningen kravspesifikasjonen

Løsningen innfrir kravene fra den opprinnelige kravspesifikajsonen på en god måte. Alle funksjoner som ble spesifisert i den første kravspesifikasjonen har blitt implementert i den endelige løsningen, bortsett fra innhentig av lokasjonsbaserte nyheter. Dette var spesifisert som et eventuelt punkt i kravspesifiskajonen. For de enhetstypene vi har hatt tilgjengelig for testing, PC, Mac, Pocket PC, og Palm-emulator fungerer systemet som ønsket, dvs. grensesnittet og funksjonalitet blir tilpasset de ulike typer enheter. Under utviklingen av produktet ble det avdekket behov og ønsker om ekstra funksjonalitet og andre teknologier utover de krav som var satt i utgangspunktet. Disse utvidelser er beskrevet i punkt 2.8 som feature creep. Deriblant forslag både fra oppdragsgiver og gruppen om omlegging til Struts. Se forøvrig punkt 7.1.5 om hvorfor vi valgte å legge om til Struts.

7.1.2 Avviket fra kravspesifikasjonen diskuteres

Funksjonene som er implementert i produktet avviker ikke fra de krav som ble beskrevet i kravspesifikasjonen. Det betyr ikke at disse funksjonene er feilfrie eller at de ikke kan løses på en bedre måte, men disse manglene er så små og detaljerte at de ikke har blitt definert som et krav i kravspesifikasjonen. Se avsnitt 7.1.3.1 for en utgreining om kjente svakheter med DynaDesk.

Det var et ønske fra oppdragsgiver å hente lokasjonsbaserte nyheter ved bruk av IP-sone teknologi. Etter en grundig undersøkelse av dagens IP-sone situasjon, dokumentert i en egen rapport [se vedlegg G], kom gruppen frem til en rekke faktorer som gjør at lokasjonsbasert henting av nyheter ikke er implementert i DynaDesk. Konklusjonen fra denne rapporten er at slik IP-sonene er bygd opp og fungerer i dag vil det vanskelig la seg gjøre å bruke denne teknologien til å hente lokasjonsbaserte nyheter.

7.1.3 Hva bør gjøres bedre

Gruppen har sett under utviklingen av DynaDesk at kravspesifikajonen burde ha vært mer utfyllende, spesifisert og detaljert. Dette skyldes at ingen av gruppens medlemmer har lært hvordan et slikt dokument skal skrives. Det hadde vært ønskelig at skolen arrangerte et intensivkurs om hvordan en kravspesifikasjon skal skrives. Et slikt kurs kunne gjerne vært arrangert like etter prosjektstart og vært rettet mot de som ikke har gått datalinjen, men som har et prosjekt som faller inn under systemutvikling. Selve utarbeidelsen av kravspesifikasjonsdokumentet burde blitt utviklet på et tidligere tidspunkt og vært diskutert mer med oppdragsgiver slik at denne var fullstendig klar før utviklingen av selve produktet startet.

Som nevnt i kravspesifikasjonen ble det underveis i prosessen fremsatt ønsker om ny funksjonalitet både fra gruppen selv og fra oppdragsgiver. Noe er blitt implementert, men ønsket om flere brukernivåer og en hjelpefunksjon har ikke blitt implementert i den endelige løsningen, fordi vi ikke hadde tilstrekkelig med tid. Dette var heller ikke funksjonalitet som ble planlagt under planleggingsprosessen av prosjektet, så vi ser på dette som en mulighet for DynDesk; ikke en mangel.

Feil som oppstår i DynaDesk blir fanget opp ved hjelp av exceptions. Disse feilmeldingene burde ha blit logget i en egen log-fil. Dette er ikke implementert i DynaDesk.

Vi anbefaler GAN Media å gjøre visse endringer i databasestrukturen – spesielt navngivingen på felter i en del av tabellene. Det er blant annet ikke mulig å opprette

kildekategorier og stikkordkategorier uten å legge henholdsvis kilder eller stikkord i disse. Videre er navngivningen i flere av tabellene meget inkonsekvent. F.eks kunde id har tre forskjellige navn; cid, id og csubscr. Dette fører til unødvendig forvirring.

7.1.3.1. Kjente svakheter ved DynaDesk

Etter hvert som vi ville bygge mer funksjonalitet i DynaDesk, har vi angret på at vi ikke har lagt med mer informasjon i de ulike beanene vi benytter for å transportere data. Denne ekstra informasjonen kunne ha blitt brukt til å gi kunden tilbakemeldinger.

Både feilmeldinger og annen tilbakemelding til brukeren er i rød tekst. Strengt tatt burde kun feilmeldingene være rød, men pga. begrensninger for `<html:errors>`-taggen i Struts kan vi ikke variere fargen på denne avhengig av om tilbakemeldingen er en feil eller ikke. Alternativt kunne vi laget en egen metode for tilbakemeldinger.

Databasen er bygget opp på en slik måte at vi det kan oppstå samtidighets-problemer ved innlegging av nye stikkord, stikkordskategorier og kildekategorier. Primærnøkkelen i disse tabellene atuonummereres via SQL-spørringer mot databasen. På grunn av dette kan det oppstå problemer når vi skal legge inn disse data i de ulike kundes profil. Muligheten for at en kunde får et feil stikkord eller kategori lagt til i sin profil er til stede. Vi har ikke funnet noen måte å komme rundt dette på, men sjansene for at to brukere legger inn et nytt stikkord, stikkordskategori eller kildekategori nøyaktig samtidig er svært små. Ikke engang da er det garantert at det vil oppstå feil. Dette karakteriseres derfor av gruppen som et kjent problem ved DynaDesk. Løsninger på problemet kan være å omstrukturere databasen, eller utføre operasjonen mot databasen i en transaksjon.

I profilen har kunden mulighet til å velge hvilke kilder hun skal abonnere på ufiltrert (alle nyheter fra kilden) og hvilke kilder som skal filtreres mot stikkord. Hvilke kilder som er valgt ufiltrert hentes opp fra databasen og markeres med en avhukning i avhukningsboksen som tilhører kilden. Kunden kan velge å ufiltrere kildene ved å fjerne avhukningen i avhukningsboksene. Problemet oppstår dersom kunden ønsker å fjerne alle avhukningsboksene fra samtlige kilder. Struts´ måte å håndeter avhukningsbokser på gjør at de tidligere valgte verdiene i avhukningsboksene vil «henge igjen».

Vi har funnet en midlertidig løsning som gjør at vi kommer rundt dette problemet, men som gruppen ikke syns er en tilfredstillende løsning. Vi lurer Struts-rammverket ved å ta ibruk en dummy-verdi. Vi håper at Struts´ neste versjon vil håndtere avhukningsbokser på en mer tilfredstillende måte.

Struts har en velbygd feilhåndtering, og denne går mye ut på å bruke Form-beans til å validere input fra brukeren. På grunn av at vi bruker forskjellige filsett for de forskjellige enhetene kan vi ikke bruke denne gode valideringsfunkjsonen. Dette gjør at vi må gjøre endel av valideringen i Action-klassene. Dette gir oss en minder grad av skille mellom applikasjons-logikk og business-logikk. Se avsnitt 5.2 for forklaring om hvorfor vi ikke kan bruke denne funksjonen.

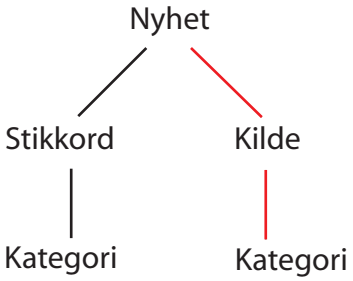
7.1.4 Muligheter videre

7.1.4.1 Publisering

DynaNews brukes opp mot Gloria for publisering av nyheter mot portalløsninger. DynaDesk vil fungere som en frontend for DynaNews. Vi ser da at vi kan lette brukerens administrering av publiseringen ved å utvide DynaDesk til å omfatte publisering. Vår løsning med artikkelskjemaet kan også knyttes opp mot dette. En bruker vil ha behov for å tilpasse hva som sendes til en portalløsning. Grunnlaget

ligger i profilen, med stikkord og kilder. Det finnes adskilte kategorisystem for kilder og stikkord (se figur 24), dette fordi stikkord gjelder alle kilder. Vi ser for oss at kunden kan ha nytte av å få nyhetene kategorisert for ulike seksjoner i portalen sin. Dette krever at man bygger et administrasjonssystem til dette formålet. Noen kunder har også ønske om å godkjenne nyheter før de slippes på portalen, for å slippe irrelevante nyheter. De kan da få mulighet til dette i DynaDesk, evt. med valg om å slippe alle nyheter gjennom. Når brukeren lager en egen nyhet i artikkelskjemaet, vil det også være greit å ha mulighet for å sende denne direkte til publisering, gjerne i en egen kategori. Vi ser for oss at dette kan samkjøres med de metodene som brukes for publisering av nyheter i DynaNews i dag.

7.1.4.2 Administrering av brukere



FIGUR 24.

DynaNews inneholder en administrator-del for å legge inn nye brukere og nye kilder. Denne delen kan være hensiktsmessig å implementere i DynaDesk, slik at brukerne og administrator forholder seg til samme system.

7.1.4.3 Brukernivå

Hvis funksjonaliteten beskrevet ovenfor skal implementeres i DynaDesk, avdekkes det straks et behov for forskjellige brukernivåer. Det kan være ønskelig flere brukerer kan dele et abonnement med forskjellige nivåer. Dette gjelder spesielt firmaer som har et abonnement på DynaDesk. Det kan da være ønskelig å gi bare en bruker tilgang til å publisere nyheter direkte på deres nettsted.

I tillegg vil det være behov for en bruker fra GAN Media med et administrator-brukernivå. Jf. feature creep i avsnitt 2.8.

7.1.4.4 Hjelpefunksjon

DynaDesk burde hatt et godt utbygd hjelpesystem. I analysekapittelet (kap. 3) av denne rapporten dokumenterer vi systemet vårt med tanke på brukeren. Mye av dette stoffet burde implementeres i en hjelpefunksjon.

7.1.5 Hvorfor omlegging til Struts

I utgangspunktet utviklet vi systemet ut fra en trelagsmodell hvor man har et lag som tar seg av business-logic, et lag som kontrollerer dataflyten og et lag som synliggjør dataene for bruker. I DynaDesk 1.0 fulgte vi denne oppbygningen. All business-logic utført i Java-klasser som jobber mot databasen via SQL-pakken fra DynaNews. En sentral JavaServlet (Dynadesk.java) fungerte som oppgavefordeler og sendte data som Java-beans videre til JSP-filer. JSP-filene tok seg av presentasjonen til klienten/enheten. Bakgrunnen for å benytte denne strukturen var ønsket om i størst mulig grad skille brukergrensesnittet fra resten av koden. Til tross for at strukturen ga gode muligheter for å skille kode og design, ble det i enkelte JSP-filer mye scripting for å loope gjennom data. I tillegg ble controller-servleten stor og uoversiktelig. I samråd med oppdragsgiver ble vi enige om å ta i bruk Struts og dens tag-biblioteker.

I Design-kapittelet beskrives Struts mer detaljert, men hovedgrunnen til at vi valgte Struts er skillet av business-logic, controller og presentasjons-laget. Dette gjør at hver av disse delene kan jobbes med uten at det medfører endringer for de andre delene. Det er fullt mulig å få en slik separasjon uten å bruke Struts-rammeverket, men Struts reduserer arbeidet med å lage en web-applikasjoner siden mye allerede er laget ferdig og debugget. Det kan også være lettere å få til et godt skille mellom presentasjon og logikk fordi Struts spesifiserer hvilken funksjonalitet som skal ligge i hvilke klasser. Måten Struts er bygd opp medfører også renere kode i JSP-filene. Istedet for å bruke JSP-scripting for å loope gjennom data i JSP-filene finnes det egne tagger som gjør dette i logic-tagbiblioteket til Struts. Noen ganger kan det alikevel være nødvendig med noe scripting, men vi har klart å unngå dette i våre JSP-filer.

GAN Media ønsket også å tilpasse systemet til det språket brukeren har definert i sin nettleser. Dette er innebygd funksjonalitet i Struts; det eneste man trenger å gjøre er å lage en resourcefil som eksempelvis har endelsene «_no» for norsk språk, «_se» for svensk språk osv. Disse resourcefilene er propertyfiler som inneholder statisk tekst på de respektive språkene. Ut fra dette velger Struts automatisk riktig resourcefil slik at bruker får all statisk tekst presentert i samme språk som det språk han har stillt inn i nettleseren sin. Dette var også en avgjørende faktor for at gruppen valgte å legge om til Struts. Vi har jobbet med flere filsett, og alle disse filsettende inneholder statisk tekst. Struts gir oss mulighet til å lagre all statisk tekst i egne filer, og det vil bli en mye enklere jobb å endre eller legge til statisk tekst i disse filene. Alternativet hadde vært å lete seg frem til teksten i de forskjellige filene for de forskjellige språkene og endre den der. Omleggingen til Struts har vært noe problematisk siden rammeverket ikke er særlig godt dokumentert. Dette vil mest sannsynlig forbedre seg etterhvert som Struts blir mer utbredt.

7.2 Rapporten

Rapporten har blitt utviklet parallellt med selve produktet, men mer intensivt i sluttfasen. I begynnelsen av prosjektet fokuserte gruppen på å skrive utfyllende statusrapporter slik at disse kunne benyttes som grunnlag for selve hovedrapporten. På slutten av prosjektet har vi delt inn gruppen slik at noen jobbet med rapporten og andre med produktet. Alle gruppens medlemmer har deltatt i rapportskrivings-prosessen, og dette har fungert bra. Inndelingen av rapporten følger skolens mal, og tidligere hovedprosjekt har også blitt studert.

7.3 DynaDesk – rettigheter og kopijournalistikk

DynaNews er basert på videreformidling av nyheter fra eksterne kilder. Dette gjøres ved å vise tittel, ingress og lenke til kilden. Det råder en allmenn oppfatning om at dette er greit. I DynaDesk går vi et skritt videre og lar brukerne i et artikkelskjema bruke «dra og slipp»-metoden til å komponere deres egen artikkel basert på søk. Det er opp til brukerne å oppgi kilder og forholde seg til opphavsrettlige regler. Ved bruk av lengre sitater bør tillatelse innhentes. En artikkel som lages i DynaDesk må forholde seg til Åndsverksloven. Vanligvis oppsummerer man med at det er greit å kopiere ingresser og legge ved referansepekere. Vi har valgt å trekke fram noen momenter som gjelder digital publisering og reproduksjon.

7.3.1 Er verk som er tilgjengelige på Internett offentliggjort?

Hvorvidt et verk er offentliggjort eller ikke er relevant i forhold til en avgrensingsregler for opphavsretten, f.eks. sitatrett

- Ja – hvis det er et åpent forum
- Nei – hvis det er tilgjengelig kun for en kontrollert/lukket brukergruppe

7.3.2 Bruk av lenker

Referensepeker: En kobling som brukeren kan velge å aktivere
Bruk av referansepekere til offentliggjort materiale er ikke det samme som publisering. Selv om bruk av referansepeker innebærer at materialet blir mer tilgjengelig, vil dette falle utenfor opphavsmannens enerett til å gjøre materialet tilgjengelig for allmennheten.
Hente-peker: En peker som automatisk inkluderer f.eks. et bilde i en webside
Bruk av hentepeker kan derimot innebære «republisering», noe som krever opphavsmannens samtykke.

7.3.3 Kopijournalistikk

På Internett driver en rekke portaler det man kan kalle kopijournalistikk. Nyheter flyter og eksistensen av kildehenvisninger er varierende. Noen grunner til dette kan være:

- Fristelsen er stor til å fremstille andres arbeid som eget, og fremstille eget medium som raskere og bedre enn andres.
- Fremdeles hersker det en del ureflekterte holdninger om informasjonens eiendomsløshet på Internett. Når informasjonen er sluppet løs, er den åpent tilgjengelig og kan benyttes av alle.
- Det er enkelt og tidsbesparende å klippe ut hele eller deler av en artikkel hvor som helst på nettet og lime det inn i egen avis, for deretter å gjøre kosmetiske endringer eller ingen endringer i det hele tatt. Kopiering er både raskere og enklere enn å skrive av, og kanskje derfor mindre moralsk belastende.
- Kopijournalistikken befinner seg i en journalistisk gråsone. Det er som kjent akseptabelt å referere andre medier, dersom man refererer kilden. Men ett sted går grensen mellom referering og kopiering – men det er ingen eksakt grense for hvor den går.

Kopijournalistikk er et presseetisk grenseland. Som kjent er det akseptabelt å referere til andre mediers saker, men det forutsetter at man i det minste gir saken en egen vinkling og bygger videre på den. Går man til det juridiske regelverket blir ikke grensene særlig mye klarere. Juridisk sett er både markedsføringsloven og åndsverkloven aktuelle. Dersom en artikkel er å anse som et åndsverk, gjelder åndsverkloven. Men hvorvidt en artikkel har såkalt verkshøyde, er et vurderingsspørsmål. Dersom en artikkel er et åndsverk gjelder opphavsretten som gir journalisten – men oftest mediet – rett til å rå over den. Om artikkelen brukes av andre skal opphavsmannens navn oppgis. Men hva om man endrer artikkelen minimalt slik at det teoretisk sett blir en annen artikkel? Når formen endres er artikkelen blitt til et annet verk. Dessuten avgrenser sitatretten opphavsretten. Med loven kommer man altså bare et stykke på vei. Det handler ikke så mye om jus som om den journalistiske sansen for fair play og for hva som tjener journalistikken på lengre sikt.

Konklusjonen må da være at med DynaDesk går GAN Media enda et skritt videre enn med DynaNews når det gjelder nyhetsformidling. Samtidig som det gir en portal eller en bruker enda større muligheter til å skreddersy stoffet, vil det også gi brukeren et større moralsk og presseetisk ansvar. Muligens vil det være på sin plass å informere om god presseskikk i DynaDesk?

7.3.4 Filtrering av websider

Vi har vurdert å strippe sider for grafikk og kode for å gjøre det lettere å browse med bærbare enheter. Teknisk sett er dette et enkelt grep å gjøre; websiden leses gjennom et filter som bare slipper ren tekst, tabeller og lenker gjennom.

Det minst problematiske vil være å kjøre et JavaScript på klientsiden. Dermed er det hver enkelt klient som sitter med ansvaret for filtreringen. Dessverre kan enkelte bærbare enheter (eks. Palm) ikke kjøre JavaScript. Alternativet er å gjøre dette serverside. Det vil føre til samme resultat, men alle sidene vil da gå gjennom vår server. Dette kan gi et visst ansvar og i verste fall føre til tvister og rettsaker. Problemet ligger i at man på denne måten vil filtrere bort det meste av reklamen, som tross alt finansierer mange websteder.

De fleste håndholdte enheter har mulighet til å slå av grafikk i browseren. På denne måten tar hver enkelt bruker ansvar for hvilke websider han vil se på hvilken måte.

8 Konklusjon

Innledning

Konklusjonen inneholder hovedkonklusjonen, faglige resultater, subjektive oppfatninger av oppgaven og drøfting av relevante temaer. Vurdering av gruppens arbeidsinnsats, prosjektresultater, veileder og om vi holdt oss innefor tidsrammene er tatt med. Det er lagt vekt på at lesere uten programmeringsbakgrunn skal forstå hva som forklares og menes.

8.1 Hovedkonklusjon

Vi mener selv at oppgaven er løst på en god måte ut fra utgangspunktet vårt. Ingen av gruppemedlemmene hadde tidligere vært med på et så stort prosjekt, og i tillegg valgte vi å benytte teknologi som ingen var særlig godt kjent med fra før. Dette var noe vi tok hensyn til i utformingen av oppgavens omfang og kravspesifikasjon. Store deler av prosjektet har blitt brukt til å sette seg inn i hvordan oppdragsgivers eksisterende system fungerer, og å lære oss nye teknologier. For oss var det viktig å lage et solid og godt strukturert produkt ut i fra kravspesifikasjonen – i stedet for å ende opp med en ufullstendig løsning.

Vi har kommet frem til en løsning som kan tas i bruk slik den foreligger idag. Det er lagt opp til at det skal være enkelt for oppdragsgiver å videreutvikle løsningen, slik at DynaDesk kan bli integrert som frontend i deres eksisterende løsning; DynaNews. Det som mangler for at løsningen skal kunne benyttes til et slikt formål er et administrator system. Gruppen er fornøyd med resultatet vi har oppnådd, spesielt med tanke på effektmålene som ble definert i punkt 1.2.4. Gruppen har hatt en god utviklingslinje i prosjektperioden. I kapittel 7 har vi tatt opp ting som kunne gjøres bedre eller annerledes.

8.2 Faglige resultater

Kompetansemessig har alle i gruppen utviklet seg mye siden prosjektstart. Alle i gruppen har tatt del i de ulike delene av prosjeket, slik at alle har hatt en faglig utvikling. Selv om alle hadde endel programmerigserfaring var nok kunnskapsnivået noe spredt blant gruppens medlemmer, og det ble derfor naturlig at enkelte jobbet mer med programmeringen enn andre. Ingen av oss hadde jobbet noe særlig i et Linux-miljø før prosjektet startet, så dette ble en ny erfaring å ta med seg. Vi har lært mye av å stadig legge om systemet, og vi har hele veien vurdert hvordan løsningen kan gjøres bedre. Ettersom vi har arbeidet i en stor gruppe har gruppemedlemmene tilegnet seg gode kunnskaper i prosjektadministrasjon, prosjektplanlegging og kvalitetssikring av prosjektarbeid.

Arbeidet med oppgaven har gitt oss en verdifull plattform å bygge videre på, spesielt når det gjelder Java-programmering og prosjektarbeid.

8.3 Evaluering av prosjektet

8.3.1 Prosjektgjennomføringen

Prosjektet er blitt gjennomført som beskrevet i fig. 1. som viser flyten i prosjektet. Gruppen startet med å analysere oppgaven med tanke på mengde, type arbeid, ferier, eksamener og tid. På denne måten kunne vi sette opp en forventet fremdriftsplan, som gjorde det mulig for oss å komme igjennom oppgaven og dens gjøremål på en slik måte at vi skulle kunne nå våre mål. I etterkant har vi sett at enkelte deler av prosessen har tatt lengre tid enn forventet, og har derfor måtte endre på tidsplanen. Ny funksjonalitet og omlegging til Struts har også vært en medvirkende årsak til at vi måtte utvide tidsrammene. Endringene vil fremgå av de vedlagte Gantt-skjemaene.

Hvert gruppemedlem ble i fremdriftsplanen også tildelt sine spesifikke ansvarsområder, som man skulle være ansvarlig for at ble påbegynt, utført og fullført. Dette ble kun delvis gjennomført fordi vi i planleggingsfasen ikke var helt sikre på hvordan oppgaven skulle løses. Dermed ble inndelingen av oppgaver muligens litt for detaljert i forhold til hvordan prosjektet senere har utviklet seg. Eksempelvis ble utviklingsdelen DynaDesk 1.1 inndelt i fire deler: grensesnitt, artikkelskjema, eksportering av data og mottak av data. De tre siste punktene har alle falt under kategorien «programmere prototype» i det reviderte Gantt-skjemaet. Dette fordi ingen av disse punktene har vært så store enkeltvis at det var naturlig å ha de som enkeltstående punkter.

8.3.2 Prosjektweb

Gjennom hele prosjektet har vi hatt tilgang til en prosjektweb, med publiseringssystem utviklet av GAN Media. Vi har også hatt tilgang til fellesområder på skolen.

8.3.2.1 Referater

Alle referatene fra møtene ved GAN Media ble lagt ut på prosjektweben. Dette for å gjøre dem tilgjengelige for våre kontaktpersoner ved GAN Media. Referater fra andre møter ble lagt på fellesområde på skolen fordi det da var lettere å laste opp og ta fram igjen. Det er viktig å angi versjon og dato i dokumentnavnet, slik at man kan se tilbake på ting.

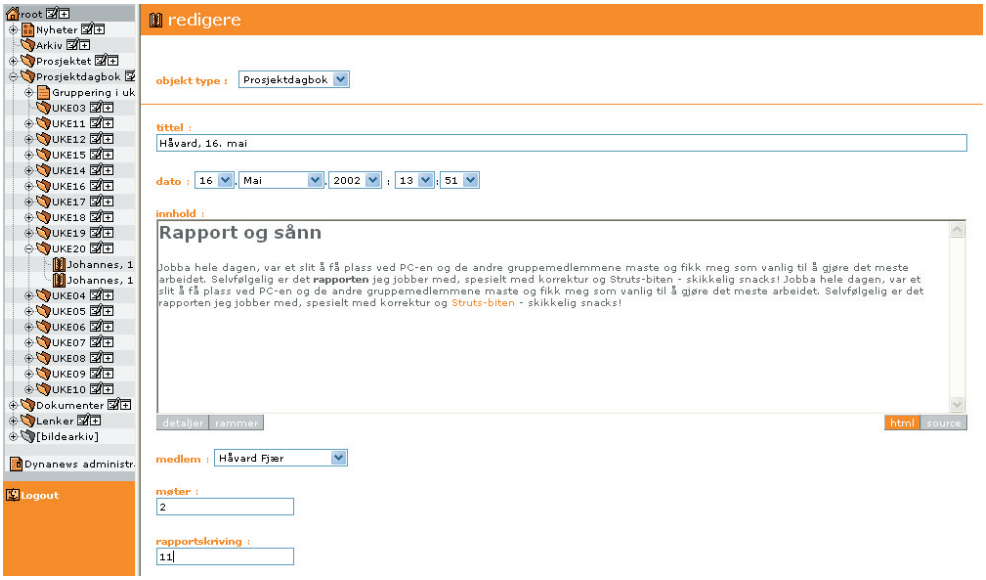
8.3.2.2 Presentasjon

På prosjektweben kunne hvem som helst lese om det vi drev med, se fig. 25. All kontaktinformasjon var lett tilgjengelig. Vi hadde mulighet til å publisere nyheter, men så ikke behovet og forholdt oss til skolens krav.

FIGUR 25.

8.3.2.3 Prosektdagbok

Denne delen av prosjektweben var spesialtilpasset for oss. Her kunne vi føre opp timetallene på hver aktivitet, se fig. 26. Dette danner grunnlaget for å se på tidsdisponeringen gjennom prosjektperioden, [se vedlegg D]. Alle gruppemedlemmene har hver dag skrevet i dagboken – dette gir oss en reel oversikt.



FIGUR 26.

8.3.2.4 Lenkesamling

På prosjektweben begynte vi å legge ut lenker til internettressurser mens vi drev research. Dette avviklet vi da vi ikke fant det tjenlig – mange av lenkene ble aldri besøkt igjen, dessuten var det lettere å notere det på annet vis, eksempelvis i referanselisten til rapporten.

8.3.2.5 Konklusjon

En prosjektweb er et viktig verktøy i prosjektarbeidet. Her kan det kommuniseres mellom oppdragsgiver, prosjektgruppe og kunde. Noen deler kan offentliggjøres, mens andre er passordbeskyttet. I et prosjekt er det viktig med et felles lagringssted, hvor man har oversikt over dokumenter og gangen i prosjektet. Hvis medlemmene jobber geografisk spredt vil en prosjektweb nesten være en nødvendighet. Det som lastes opp på prosjektweben bør ikke endres – videreutvikling og forbedring må komme i nye dokumenter.

8.3.3 Evaluering av prosjektet

8.3.3.1 Gruppens vurdering av oppbygning og samarbeid

DynaDesk har vært et spennende og utfordrende prosjekt å jobbe med. Gruppen har stort sett arbeidet sammen to og to, eller alene, alt etter oppgavens egenart. Samarbeidet mellom gruppemedlemmene har fungert bra, og vi har ikke hatt de store disputtene i løpet av prosjektgangen. Der det har vært uenighet om løsning på problemer, videre fremdrift eller løsing av oppgaver, har vi kunnet diskutere oss frem til enighet. Gruppen har i løpet av de fem månedene vi har hatt til rådighet arbeidet jevnt [se vedlegg D], og har hatt gruppemøte hver morgen. På disse morgenmøtene ble arbeidsoppgaver fordelt og status på påbegynte oppgaver gått gjennom. Gruppen

har i prosjektperioden hatt et grupperom til disposisjon sammen med en annen gruppe. Det meste av arbeidet i prosjektet har foregått her. Det har vært en stor fordel for gruppen å sitte samlet og jobbe slik at vi har vært tilgjengelig for hverandre til enhver tid.

Vi har lært mye gjennom det å jobbe i gruppe, og har fått en klarere oppfatning om hvordan vi oppfører oss som enkeltpersoner, og hvordan man blir oppfattet av de andre i gruppen. Å tørre å spørre de andre for å dra nytte av hverandres kunnskaper og lære at «alle kan ikke gjør alt» har vært noen av de viktigste erfaringene å ta med seg.

8.3.3.2 Gruppens vurdering om samarbeidet med oppdragsgiver

Samarbeidet med oppdragsiver GAN Media har fungert meget bra. Vi har hatt statusmøter annenhver fredag hos oppdragsgiver hvor vanligvis begge kontakt-personene – Pepijn Oomen og Svein Mathisen – har vært tilstede. I forkant av stausmøtene har gruppen samlet sammen spørsmål og sendt disse via e-post slik at oppdragsgiver har hatt mulighet til å forberede seg. Utenom disse møtene har vi benyttet e-post, MSN Messenger, telefon og lignende, dersom vi har hatt behov for komme i kontakt, og som regel har vi fått rask tilbakemelding. GAN Media har vist velvilje til gruppen og stilt informasjon, materiell og tid til rådighet. De har vist interesse for oppgaven gjennom hele prosjektet, og kommet med tilbakemeldinger til gruppens arbeid.

8.3.3.3 Gruppens vurdering om samarbeidet med veileder

Gruppens veileder ved skolen, Rune Lossius, har vært meget behjelplig og kommet med konstruktive råd underveis. Han er dyktig innenfor de teknologiene vi har benyttet i prosjektet, og har hele tiden kommet med forslag til hvordan ting kan og bør gjøres. Vi har hatt stor nytte av hans erfaringer med oppbygging av systemarkitektur og rapportskriving. I begynnelsen av prosjektet hadde vi møter ukentlig på veileders kontor, noe som var meget nyttig i starten av prosjektet for å peile prosjektgruppen inn på riktig spor. Etter hvert som prosjektet skred frem følte verken gruppen eller veileder behov for ukentlige møter, og det ble i stedet avholdt møter etter ved milepæler eller når det var behov for det.

9 Litteraturliste

Innledning

I dette kapittelet henviser vi til de kilder vi benyttet oss av i løpet av prosjektperioden. Kilder som er benyttet under skriving av IP-sone sapporten, ligger som et vedlegg til denne.

9.1. Bøker og manualer

«Professional JSP – 2nd edition»
Wrox Press Ltd.
Brown, Burdick, Falkner, Galbraith, Johnson, Kim, Kochmer, Kristmundsson, Li, Malks, Nelson, Palmer, Sullivan, Taylor, Timney, Tyagy, Van Damme, Wilkinson
ISBN 1-861004-95-8
United States of America, 2001

«Java Gently»(second edition)
Addison-Wesley Longman Limited
Judy Bishop
England, 1998
ISBN 0-201-34297-9

«Java Examples in a Nutshell» (second edition)
O'Reilly & Associates, Inc.
David Flanagan
United States of America, 2000
ISBN 0-596-00039-1

«Web Development with JavaServer Pages»
Manning Publications Co.
Duane K. Fields, Mark A. Kolb
England, 2000
ISBN 1-884777-99-6

«Developing Java™ Servlets»
Sams Publishing
James Goodwill
United States of America, 1999
ISBN 0-672-31600-5

«Java™ How To Program» (third edition)
Prentice-Hall, Inc.
H.M. Deitel, P.J. Deitel
United States of America, 1999
ISBN 0-13-012507-5

«Thinking in Javs»
Prentice Hall PTR
Bruce Eckel
United States of America, 1998
ISBN 0-13-659723-8

«Designing Web Usability: The Practice of Simplicity»
New Riders Publishing
Jacob Nielsen
Indianapolis, 2000
ISBN 1-56205-810-X

«Information Appliances and Beyond»
Morgan Kaufmann Publishers
Eric Bergman
United States of America, 2000
ISBN 1-55860-600-9

9.2 Internettressurser

«Java™ 2 Platform, Standard Edition, v 1.4.0 API Specification» (2002, 30. januar), nedlastet 04.02.2002 fra http://java.sun.com/j2se/1.4/docs/api/index.html.

«Servlet and JavaServer Pages API Documentation» (2002, 8. februar), nedlastet 12.02.2002 fra http://jakarta.apache.org/tomcat/tomcat-4.0-doc/servletapi/index.html.

«Apache Struts Framework (Version 1.1-b1)» (2002, 19. mars), nedlastet 02.04.2002 fra http://jakarta.apache.org/struts/api/index.html.

«The Struts User's Guide», nedlastet 02.04.2002 fra http://jakarta.apache.org/struts/userGuide/.

Forelesningsnotater fra faget «Elektronisk publisering» (2001, 05. oktober), nedlastet 16.04.2002 fra http://spray.nettavisen.no/elpub/forelesning051001.doc.