SAMMENDRAG AV HOVEDPROSJEKTET

Tittel:	Card Management System Enterprise Java Beans	**Nr.** : **Dato** : 23.05.2001
Deltaker(e):	Guy Steffen Brun Oddgeir Kaspersen Øyvind Steinbekken Lars Arne Høeg	
Veileder(e):	Frode Haug	
Oppdragsgiver:	ErgoGroup	
Kontaktperson:	Bjørn Olav Berg	
Stikkord	Database, Enterprise Java Beans, Jboss, Smartkorthåndtering.	
Antall sider: 141	**Antall bilag: A - E**	**Tilgjengelighet :** Åpen

Kort beskrivelse av hovedprosjektet:

Det skal utvikles en testapplikasjon for å håndtere klargjøringen av smartkort. Det fungerende systemet skal effektiviseres ved at løsningen utvikles i Enterprise Java Beans. Applikasjonen skal kommunisere mot en database der alle data om smartkortene, kortinnehaverne og applikasjonene på kortene ligger. Det skal være mulig å legge inn data, hente ut data og endre eller slette fra databasen. Løsningen skal utvikles i ulike typer Beans for å teste ut effektiviteten og ytelsen til de ulike komponentene, samt drøfte hvorvidt det er nødvendig å ta i bruk denne teknologien. Senere utvidelser vil kunne gjøre at applikasjonen blir en del av et komplett system for administrering av smartkort.

FORORD

Smartkort har siden det ble oppfunnet på 70-tallet vært i stor utvikling. Det er først i den senere tid at teknologien rundt smartkortet har blitt god nok til effektiv og sikker kommersiell bruk. Det er ventet at smartkortet i fremtiden vil ta over for de tradisjonelle magnetstripekortene som fremdeles er mest brukt. Lagringskapasiteten og sikkerheten til magnetstripekortet er svært begrenset i forhold til smartkortet. Med smartkortet kan man tilby flere tjenester på samme kort. Det er derfor naturlig å anta at smartkortet vil bli svært vanlig i fremtiden. For å administrere smartkortet med tilhørende kortinnehavere og applikasjoner må man ha et underliggende korthåndteringssystem. Til slike systemet stilles det høye krav til sikkerhet og effektivitet.

Vår oppdragsgiver ErgoGroup ønsket at vi startet et prosjekt der vi skulle utvikle et korthåndteringssystem basert på den nye teknologien Enterprise Java Beans. Vår oppdragsgiver har allerede et korthåndteringssystem utviklet i eldre teknologi. Det er ønskelig at det nye systemets ytelse skal testes opp mot det gamle systemet. korthåndteringssystemet som vi skal utvikle er i første omgang kun tenkt til internt bruk hos ErgoGroup.

Vi vil rette en takk til vår prosjektveileder Frode Haug, som har vært til stor hjelp under prosjektarbeidet. Vi vil også rette en takk til Bjørn Olav Berg som har vært vår kontaktperson. Sist men ikke minst vil vi takke ErgoGroup for lån av grupperom og utstyr.

Gjøvik 23.05.2001

Guy Steffen Brun Øyvind Steinbekken

Lars Høeg Oddgeir Kaspersen

INNHOLDSFORTEGNELSE

1.0 INNLEDNING

1.1 BEGREPER

1.1.1 Definisjoner

ALC Application Load Certificate - Sertifikat som er nødvendig for å laste applikasjoner.

ALU Application Load Unit - Enheten som laster applikasjonen.

BMP Bean Managed Persistence - En avart av Entity Beans. Du håndterer de persistente operasjonene selv, inkludert lagring, lasting og datasøking inni entity beanen. Persistent vil si at komponentenes tilstand blir lagret til en annen lagringsplass (secondary) som for eksempel en relasjonsdatabase.

CMP Container Managed Persistence - Containeren/serveren utfører alle funksjoner for komponentenes data tilgangslag for deg, inkludert lagring, lasting og søking på komponentenes data.

CMS Card Management System - System som står for klargjøringen av et smartkort helt frem til det når sluttbrukeren.

EJB Enterprise Java Beans - en serverside komponent arkitektur som tillater og forenkler bygging av enterprise-klasse distribuerte objekt applikasjoner i Java uten at brukeren trenger å lage et eget komplekst distribuert objekt rammeverk.

Entity Bean En Entity Bean er en komponent som representerer persistent data.

Java-klient Java-utviklet klient som kommuniserer mot en server.

Jboss Applikasjonsserveren vi benytter under prosjektet.

Jbuilder Javabuilder – Program for utvikling av javakomponenter. I prosjektet benyttes dette til å utvikle det grafiske brukergrensesnittet for modulene.

JNDI Java Naming Directory Interface.

Modul Komponent som implementeres i applikasjonen for å få all funksjonalitet på plass. En liten del av den totale løsningen.

Multos MultiapplikasjonOS – Standard smartkortteknologi som gjør at det er mulig å ha flere uavhengige applikasjoner på samme kort. Dette er opprinnelig utviklet av Mondex International.

Oppetid Et mål på hvor lenge et system er oppe og går i løpet av et døgn.

Plattform Hva slags operativsystem det opereres på (for eksempel Linux, Windows, MAC)

Responstid Et mål hvor lang tid systemet bruker på å svare på en forespørsel fra klienten.

RMI Remote Method Invocation – Grensesnittet mellom EJB-objekter.

RUP Rational Unified Process - Systemutviklingsmodell

SFSB Stateful Session Bean - Et objekt som kun kan utføre prosedyrer uten å ta vare på data.

1.1.2 Kort innføring i Enterprise Java Beans

1.1.2.1 Introduksjon

Enterprise Java Beans brukes for å utvikle komponenter på serversiden. Man kan ved hjelp av disse komponentene bygge distribuerte EJB-objekter. Ved å bruke EJB kan man skrive sikre og pålitelige applikasjoner uten å lage et eget komplekst, distribuert rammeverk. EJB handler om rask applikasjonsutvikling for serversiden - man kan raskt og enkelt konstruere serverside komponenter i Java. EJB er dessuten designet for å støtte applikasjoners flyttbarhet og gjenbruk.

1.1.2.2 EJB server og EJB container

En applikasjonsserver forsyner EJB-objektene (applikasjonen) med tjenester slik som transaksjonstjenester og sikkerhetstjenester. Disse tjenestene er nødvendige for at EJB-objektene skal være robuste og sikker for mange brukere samtidig. For EJB deles dette i to deler:

EJB container er stedet hvor EJB-objekter arbeider og kjøres. Det kan være mange objekter som er aktive av gangen i containeren. EJB containeren er ansvarlig for å håndtere disse objektene. Kommunikasjonen skjer ved kall av ulike metoder som må være implementert i hver enkelt objekt.

EJB serveren kan inneholde en eller flere containere. Serveren håndterer lavnivå system ressurser og allokerer ressurser til containere når det trengs.

I praksis finnes det ikke noe helt klart skille mellom EJB container og EJB server. Disse to blir ofte omtalt som en og samme ting.

1.1.2.3 Bean Managed Persistent Entity Bean (BMP)

En BMP er et EJB objekt som kan håndtere og ta vare på data. Disse objektene arbeider ofte mot en database. Utvikleren som lager objektet må selv sørge for å kode alle container definerte metoder selv, slik som ejbLoad, ejbStore, ejbFindByPrimaryKey, for å nevne noen. Disse metodene (med flere) kalles av serveren for eksempel for å hente data fra databasen og legge data inn i et EJB-objekt. Man må altså håndtere de persistente operasjonene selv. Dette innebærer også å opprette forbindelse til databasen samt kode SQL-spørringer mot databasen.

1.1.2.4 Container Managed Persistent Entity Bean (CMP)

En CMP er et EJB objekt som på samme måte som BMP kan håndtere og ta vare på data. Utvikleren som lager slike objekter trenger ikke å kode container definerte metoder og de persistente operasjonene selv. Dette inkluderer lasting av data fra database til et EJB objekt, lagring, søk og oppretting av kommunikasjon mot databasen. Det eneste man må gjøre er å beskrive hvilke felter i databasen som skal mappes/relateres opp mot hvilke felter i EJB objektet. Serveren vil da bruke den forhåndsdefinerte lagringsenheten (databasen) den har til rådighet. Dette fører til en teoretisk databaseuavhengighet og tillater deg å bytte mellom lagringsenheter siden man ikke trenger å skrive noen kode som går direkte mot databasen. Til konklusjon kan man si at man som utvikler sparer mye kodetid.

1.2 PROSJEKTETS BAKGRUNN

Ergo Group (tidligere Posten SDS) er en ledende leverandør av IT-tjenester og produkter til offentlig og privat sektor. Ergo Group har hovedkvarter i Oslo og regionkontorer i Gjøvik, Trondheim og Mo i Rana. Konsernet har ca 1200 ansatte på landsbasis.

Selskapets sentrale satsningsområder er datadrift og overvåking, sikker elektronisk informasjonsformidling, smartkort, lønns- og økonomitjenester og IT og logistikk.

Vår oppdragsgiver har regionkontorer på Gjøvik og har spesialisert seg innenfor smartkort. Smartkort er en rimelig ny teknologi i rask utvikling. Smartkortet ble tatt i bruk tidlig på 70-tallet, men først i den senere tid er teknologien blitt standardisert. Satsingen på dette feltet er nå økende.

I motsetning til tradisjonelle magnetstripekort (kredittkort) kan smartkortet inneholde flere applikasjoner. Ergo Group har tatt i bruk Multos sin smartkortteknologi. Dette gjør smartkortet til en sikker teknologiløsning. Man kan dermed lagre personlige data på kortet.

Ergo Group har tidligere hatt flere prosjekter knyttet opp mot næringslivet og Høgskolen i Gjøvik. Dette har blant annet ført til et pilotprosjekt for Norsk Tipping, Adgangskort på HiG og kundekort hos statens lånekasse for utdanning. Ergo Group ønsker nå å ta i bruk Enterprise Java Beans for videre utvikling og effektivisering av systemene som støtter opp under smartkort. I første omgang ønsker de å skrive om et eksisterende system for å sammenlikne effektivitet og ytelse.

Vår oppgave blir å utvikle en løsning innenfor dette feltet.

1.3 PROSJEKTMÅL

- Effektivisere det nåværende systemet.
- Utvikle en applikasjon som håndterer data om smartkort, kortinnehavere, applikasjoner og brukere av systemet.
- Klargjøre applikasjonen slik at den er klar for integrering med modulen som demonstrerer lastingen av applikasjoner over Internett.

1.4 OMFANG

Oppgaven går ut på å lage et Card Management System (heretter bare referert til som CMS) for multiapplikasjons smartkort. Systemet skal administrere kortene, personene kortene er utstedt til og applikasjonene som er lastet på kortene. Dette skal implementeres ved hjelp av Enterprise Java Beans teknologien (EJB).

EJB er en komponentmodell/rammeverk for tjenersiden.
Denne modellen er basert på en flerlags, distribuert objektarkitektur, og er en utvidelse av JavaBeans. En komponentmodell definerer en omgivelse for å støtte gjenbrukbare komponenter.

Vi skal også særlig konsentrere oss om bruken av Entity Beans, om dette fungerer som tenkt og om det er mer effektivt med den mer tradisjonelle håndteringen av data (ytelse). Vi skal i tillegg se nærmere på hvordan Container Managed Persistence og Bean Managed Persistence påvirker ytelsen på det ferdige produktet.

Utgangspunktet er et nåværende CMS utviklet av Ergo Group. Vi skal lage en egen versjon av dette systemet ved å benytte Enterprise Java Beans.

I oppgaven skal vi ta utgangspunkt i en kravspesifikasjon som Ergo Group har utarbeidet for et større CMS. Utfra dette skal vi lage en kravspesifikasjon for systemet vi skal implementere.
Vi skal også utvikle en designbeskrivelse av systemet. I dette arbeidet skal det legges spesiell vekt på å designe systemet for Enterprise Java Beans arkitektur.

1.5 FULLSTENDIG OPPGAVEDEFINISJON

Innholdet i de forskjellige smartkortsystemene varierer fra system til system. Generelt består et system av en kortleser som er koblet opp mot en pc og et sentralt register der sertifikatene verifiseres og godkjennes for bruk. CMS står for klargjøring av kortet slik at dette blir personalisert. Systemet som implementeres skal kunne gjøre følgende:

Man skal ha mulighet til å registrere, vedlikeholde og slette data om smartkortet, data om personene kortet er utstedt til, brukerne av systemet og applikasjonene. Moduler som må implementeres:

Bruker
Det skal implementeres en påloggingsmodul i systemet. Systemoperatøren skal ha mulighet til å legge inn nye brukere og endre eller slette eksisterende brukere.

Kortinnehaver
Systemet skal registrere nye kortinnehavere. Dette gjøres vanligvis fra fil, men systemet skal også kunne håndtere manuell registrering. Det skal være mulig å hente ut data om, endre eller slette informasjon om eksisterende kortinnehavere.

Kort
Systemet skal registrere nye kort. Dette kan utføres både manuelt og fra fil. Det skal også være mulig å hente ut data om, endre eller slette informasjon om eksisterende kort. CMS må holde oversikt over hvert kort gjennom hele kortets livssyklus. Man må derfor kunne endre status på et kort.

Applikasjon
Nye applikasjoner skal importeres fra fil. Det er ikke mulig å endre på eksisterende applikasjoner, men det skal derimot være mulig å hente ut data om og slette eksisterende applikasjoner. Systemet skal kunne administrere sertifikater som trengs for å kunne laste og slette applikasjoner som finnes på smartkortet

Klargjøre for lasting av applikasjoner
Systemet skal gjøres klart slik at det er i mulig å bygge det sammen med modulen for lasting av applikasjoner over Internett. Data må hentes ut fra databasen for å laste disse applikasjonene.

1.6 MÅLGRUPPE

Målgruppe for vårt prosjekt vil i første omgang være vår oppdragsgiver Ergo Group. Deres hensikt med oppdraget er å teste ut en ny teknologi og vurdere om det er behov for å anskaffe den som erstatter for det fungerende CMS de nå benytter.

1.7 STUDENTENES FAGLIGE BAKGRUNN

Alle fire gruppemedlemmene har bakgrunn fra allmennfagutdanning på videregående skoler og Høgskolen i Gjøvik. Her på høgskolen går alle tredje og siste året på datalinjen. Prosjektarbeid har vært en viktig del av undervisningen for å løse ulike problemstillinger. Vi har hatt to større prosjektarbeider gjennom studiene her.

Vi har også bakgrunn fra systemutviklingsfagene og programmering i flere språk deriblant Java.

Ingen av oss har noe erfaring fra smartkortteknologien og Enterprise Java Beans. Dette er noe vi må sette oss inn i for å kunne løse oppgaven.

1.8 ARBEIDSFORM

Vi har for det meste valgt å dele opp gruppa slik at to og to har arbeidet sammen. Dette for å effektivisere arbeidstida. Vi har også satt oss inn i nødvendig litteratur for å kunne løse oppgaven. Dette har vært litteratur gjennom Ergo Group og informasjon på Internett. Når det gjelder implementeringen av modulene har vi benyttet oss av det tildelte grupperommet hos Ergo Group. Der har vi arbeidet tett opp mot oppdragsgiver, som har vært behjelpelig med spørsmål som meldte seg underveis. Gruppa har også levert mye av arbeidet til veileder for tilbakemeldinger under prosjektets gang.

1.9 ORGANISERING AV RAPPORTEN

Rapporten er organisert på følgende måte:

Kapittel 1: Innledning
Viser hvordan hele rapporten er bygd opp, samt sammenhengen mellom hvert kapittel. Innledningen inneholder ordforklaringer brukt i rapporten samt en kort innføring i Enterprise Java Beans teknologien. Videre følger en fullstendig oppgavebeskrivelse med detaljer om hva den går ut på samt arbeidsformen til gruppa.

Kapittel 2: Kravspesifikasjon
Analyse av hvilke krav som stilles til systemet. Kravspesifikasjonsdokumentet definerer de funksjonelle og operasjonelle kravene til systemet.

Kapittel 3: Design
Hovedtrekkene fra design av systemet beskrives. Bygger på kravspesifikasjonen. Viser hvordan gruppa har valgt å løse oppgaven kravspesifikæsjonen beskriver.

Kapittel 4: Implementasjon
Denne delen viser hvordan vi har realisert systemet ut fra kravspesifikasjon og designdokument.

Kapittel 5: Testing
Beskriver hvordan systemet er testet og hvor robust det er.

Kapittel 6: Konklusjon
Inneholder en drøfting av resultatet og konklusjonen for prosjektet.

Kapittel 7: Litteraturliste
Litteratur og kilder som er brukt under arbeidet med prosjektet.

Kapittel 8: Vedlegg
Her ligger alle vedlegg for prosjektet.

2.0 KRAVSPESIFIKASJON

2.1 BRUKERBESKRIVELSE

2.1.1 Omgivelser

Systemet består av en Java klient, en databaseserver og en applikasjonsserver. Databasen er en Oracle database.
Klienten og serveren er koblet sammen via lokalnettverket.

EJB er plattform uavhengig derfor vil softwaren i prosjektet også være det. Primært skal systemet utvikles for Windows NT. Systemet er en intern testversjon for Ergo Group, og skal derfor fungere i lokalene til Ergo Group. Det vil si romtemperatur og norsk strømnett.

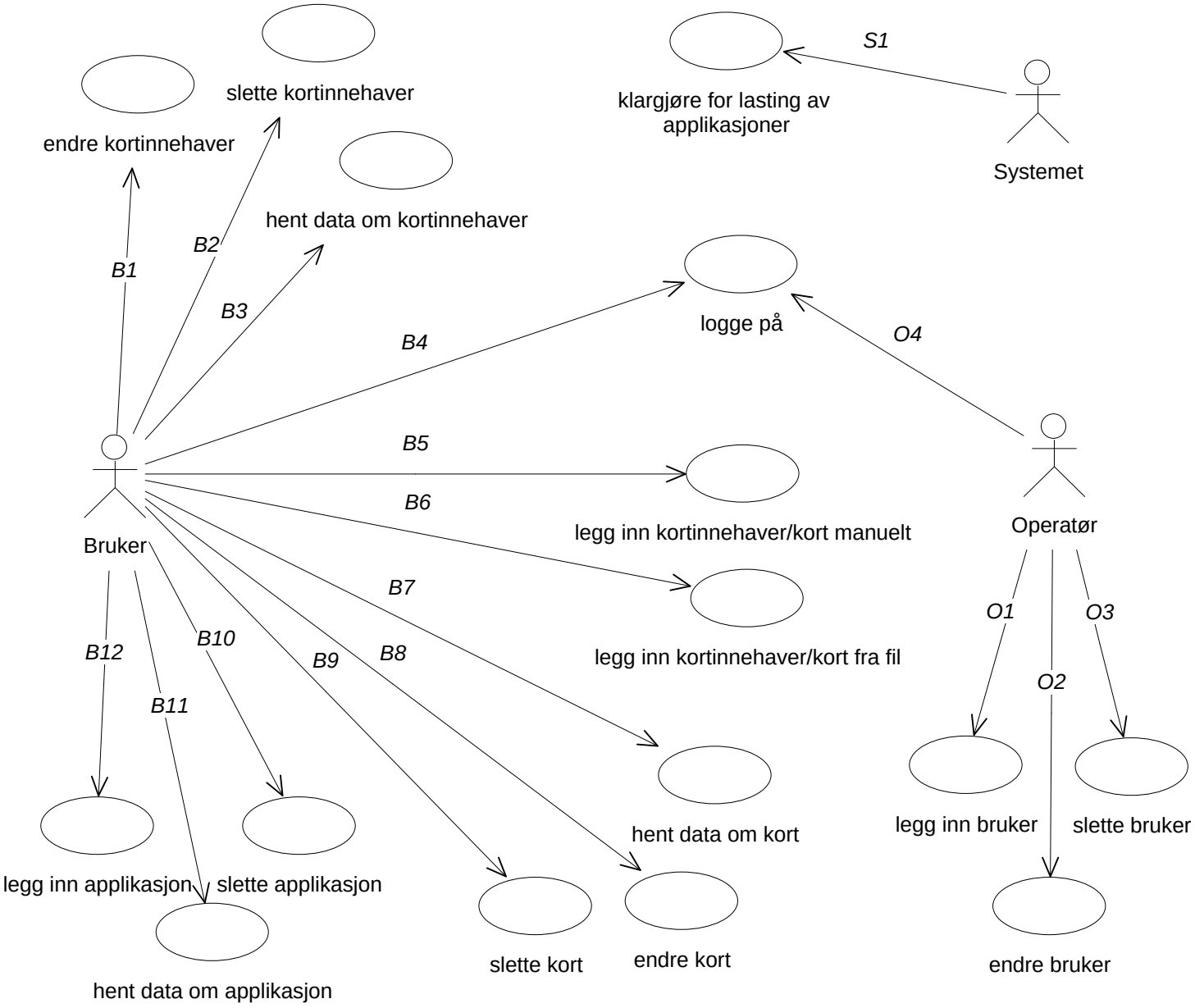
Kommuniseringen med kortet vil foregå ved hjelp av Multos sin teknologi. Kommunikasjonsstandarden som skal benyttes er EJB.

2.1.2 Systemets brukere

Siden systemet er for internt bruk forventes det at sluttbruker har god kunnskap om data. Det er derfor sannsynlig at sluttbruker og systemoperatør er samme person. På grunn av høy kompetanse hos bruker vil et minimum av opplæring være påkrevd.

2.1.3 Funksjon

Use case diagram for å beskrive handlingen en aktør utfører på systemet. Skal gi en enkel oversikt uten å gå inn på funksjonelle spesifikasjoner og krav.



Figur 2.1

Overordnet Use case:
Det overordnede use caset skal gi en forståelse av de grunnleggende prosessene.

Use case: **B1 - Endre kortinnehaver**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne hente ut data om eksisterende kortinnehavere. Det må være mulighet for å endre disse dataene.

Use case: **B2 - Slette kortinnehaver**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet skal kunne hente ut en eksisterende kortinnehaver. Denne kortinnehaveren må kunne slettes fra databasen.

Use case: **B3 - Hent data om kortinnehaver**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne hente ut informasjon om aktuell kortinnehaver ut fra databasen og presentere dette på skjerm.

Use case: **B4,O4 - Logge på**
Aktør: Bruker, operatør
Type: Primært
Beskrivelse: Systemet skal inneholde en påloggingsmodul for å kunne autorisere brukere på systemet. Dette gjøres ved brukernavn- og passordkontroll. Deretter vil brukeren få satt sine rettigheter uavhengig om han er systemoperatør eller vanlig bruker.

Use case: **B5 - Legg inn kortinnehaver/ kortinformasjon manuelt**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne legge inn kortinnehaver/ kortinformasjon, samt hva slags type kort(utviklingskort eller vanlig brukerkort). Dette skal kunne utføres manuelt via tastatur. Informasjonen lagres i databasen..

Use case: **B6 - Legg inn kortinnehaver/ kortinformasjon fra fil**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet mottar en fil som inneholder informasjon om kortinnehaver/ kort. Filen må kunne leses på klienten og importere data inn i de riktige tabellene i databasen.

Use case: **B7 - Hent data om kort**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne hente ut informasjon om aktuelt kort ut fra databasen og presentere dette på skjerm.

Use case: **B8 - Endre kort**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må håndtere eksisterende kort som skal oppdateres. Det må være mulig å endre disse dataene. Status på kort må også kunne endres.

Use case: **B9 - Slette kort**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne hente ut eksisterende kort som skal slettes (foreldet eller utrangert). Kortet må deretter slettes fra databasen.

Use case: **B10 - Slette applikasjon**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må hente ut en eksisterende applikasjon som skal slettes. Applikasjonen må deretter slettes fra databasen.

Use case: **B11 - Hent data om applikasjon**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne hente data om applikasjon fra databasen og presenteres på skjerm for bruker.

Use case: **B12 - Legg inn applikasjon**
Aktør: Bruker
Type: Primært
Beskrivelse: Systemet må kunne legge inn data om ny applikasjon. Dette skal importeres fra fil.

Use case: **O1 - Legg inn bruker**
Aktør: Operatør
Type: Primært
Beskrivelse: Systemet skal kunne opprette nye brukere på systemet. Disse brukerne får tildelt passord og brukernavn.

Use case: **O2 - Endre bruker**
Aktør: Operatør
Type: Primært
Beskrivelse: Systemet må kunne hente ut data om eksisterende bruker. Det må være mulighet for å endre disse dataene.

Use case: **O3 - Slette bruker**
Aktør: Operatør
Type: Primært
Beskrivelse: Systemet må kunne hente ut en eksisterende bruker. Brukeren må deretter kunne slettes fra databasen.

Use case: **S1 - Klargjøre for lasting av applikasjoner**
Aktør: Systemet
Type: Primært
Beskrivelse: Systemet skal gjøres klart slik at det er mulig å bygge det sammen med modulen for applikasjoner over Internett. For at det skal kunne være mulig å laste en applikasjon over Internett, trengs det en del i tillegg, ALU (Application Load Unit) og ALC (Application Load Certificate). Disse dataene må det være mulig at CMS tar vare på i en eller annen form.

2.1.4 Operasjon

Siden systemet er et forholdsvis lite testsystem forventes det høy oppetid. Ved eventuelle feilsituasjoner må systemoperatør finne feilkilden og utbedre feilen utfra dokumentasjonen til systemet.

Systemet skal ha en påloggingsmodul for autorisering av brukere. Denne kommuniserer mot brukertabellen i databasen. Ved brudd startes klient på nytt. Prosedyrer for tap av data under overføring fra database vil ikke implementeres i vår løsning.

2.1.5 Aspekter omkring livssyklus

Ergo Group vurderer å bygge videre på testprosjektet ved å legge på flere moduler for å få et komplett CMS. EJB gjør det enkelt å bygge på moduler.

Siden programmet som utvikles er plattformuavhengig og forholdsvis lite vil systemet være enkelt å flytte og skalere. Det må gjøres justeringer ettersom hvilken applikasjonsserver man velger å bruke.

All kode skal dokumenteres og alle ferdige moduler skal ha versjonsnummer som skal være synlig for bruker av programmet. Dette gjør det enklere å integrere nye moduler og å oppgradere det eksisterende systemet.

2.1.6 Ytelse

Systemet skal ikke erstatte et eksisterende system. Det nye systemet skal fungere parallelt med det eksisterende. Forskjellige moduler i systemet skal utvikles ved hjelp av ulike typer Java beans. Vi skal bruke Container managed beans og bean managed. Disse skal testes opp mot hverandre med hensyn på ytelse. Vi regner med liten responstid siden vi opererer på et lokalnett.

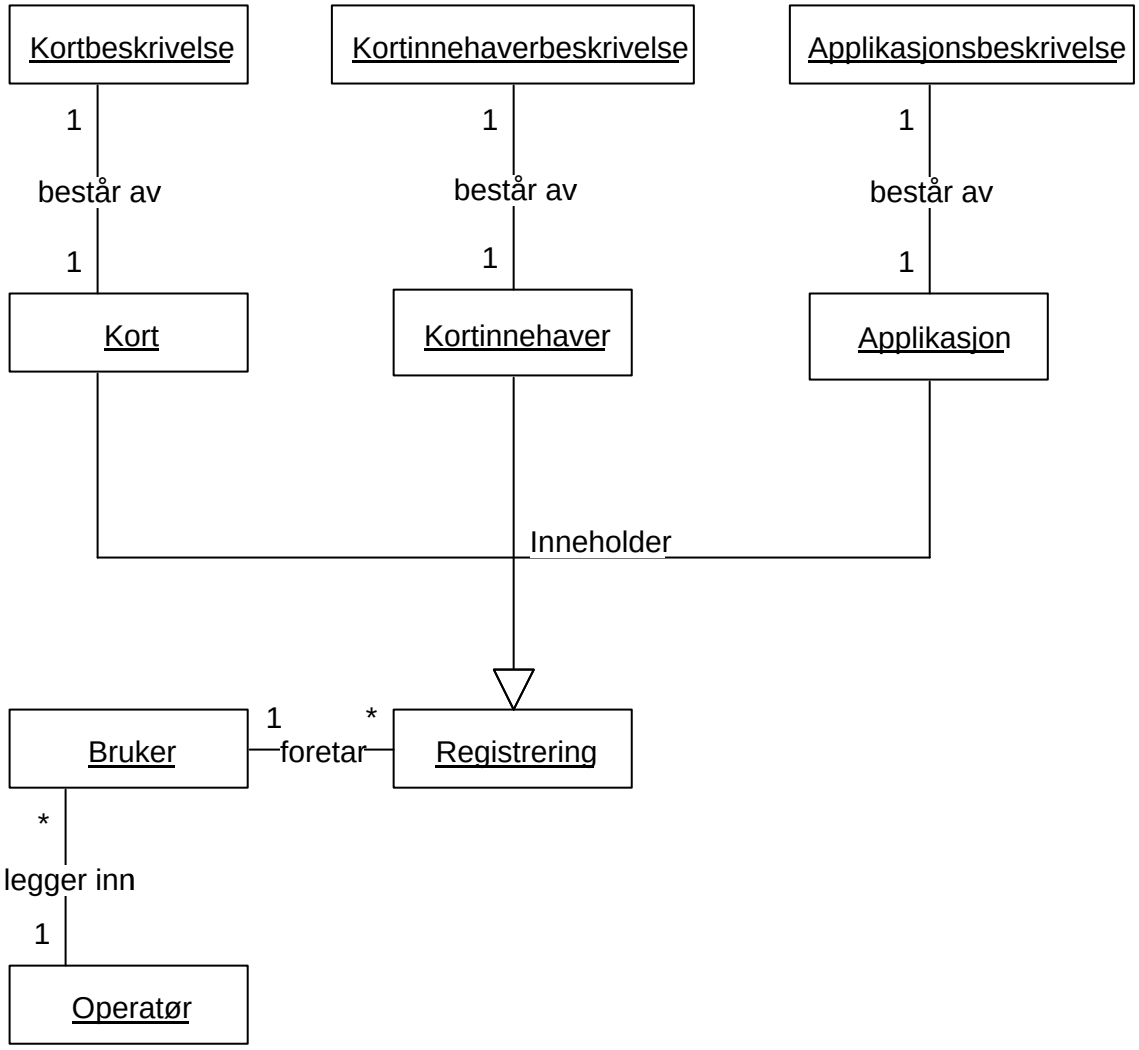
2.1.7 Begrensninger

Systemet skal i utgangspunktet kjøres på Windows NT eller Windows 98. Det grafiske brukergrensesnittet er utviklet i Java som er plattformuavhengig. EJB ligger integrert på Jboss sin applikasjonsserver som også er plattformuavhengig.

2.2 FUNKSJONELL SPESIFIKASJON

2.2.1 Konseptuelt klassediagram

En viktig del av analysefasen er å identifisere ulike konsepter og overføre disse til et klassediagram. En konseptuell modell er en beskrivelse av hvordan den virkelige verden ser ut. Modellen illustrerer viktige konsepter rundt oppgaven. Dette er det viktigste produktet å lage i løpet av den objekt orienterte analysen. Konseptet kan være en ide, et objekt eller en ting. Den gir ingen beskrivelse av software designet.



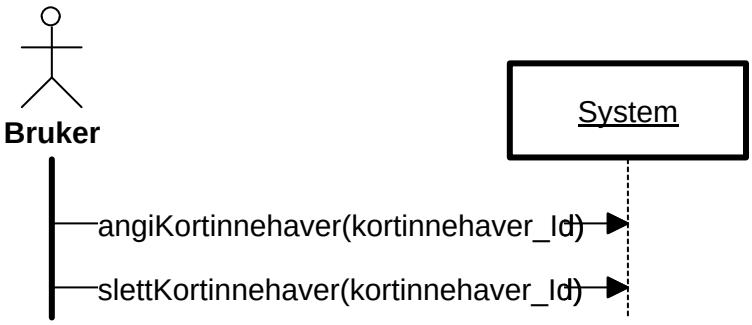
Figur 2.2

2.2.2 System sekvens diagram

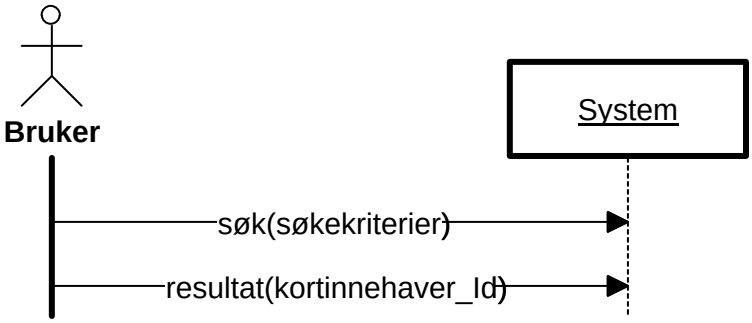
Et system sekvens diagram gir et bildet av hver enkel komponent i use caset, de eksterne hendelser som brukeren forårsaker, rekkefølgen på disse og systemhendelser.

Use case: B1 - Endre kortinnehaver

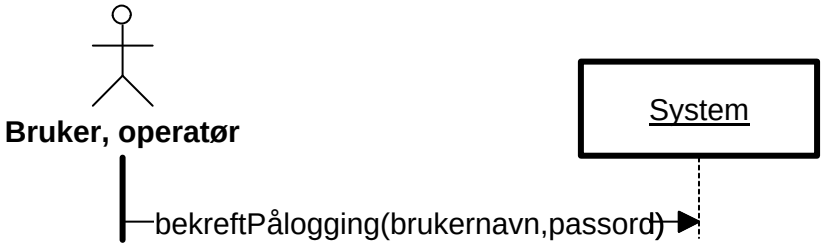
Use case: B2 - Slette kortinnehaver



Use case: B3 - Hent data om kortinnehaver

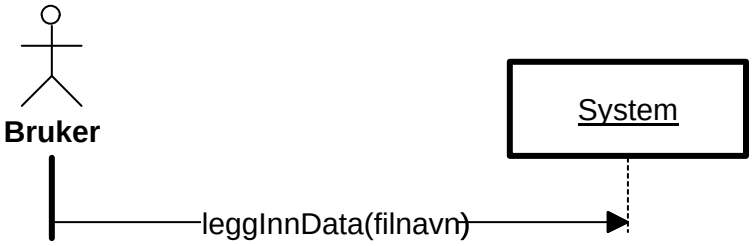


Use case: B4, O4 - logge på

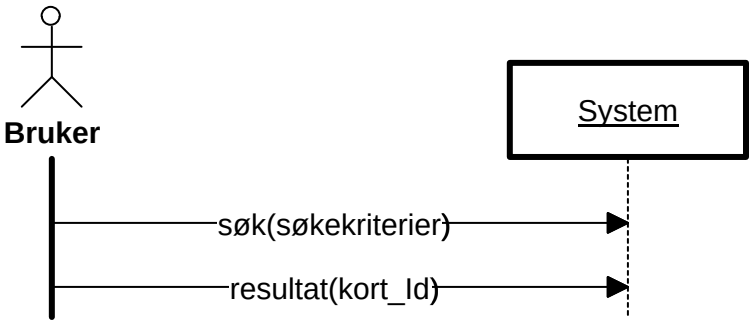


Use case: B5 - Legg inn kortinnehaver/ kortinformasjon manuelt

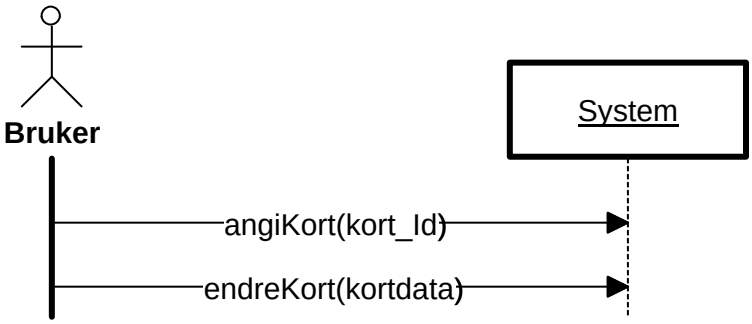
Use case: B6 - Legg inn kortinnehaver/ kortinformasjon fra fil



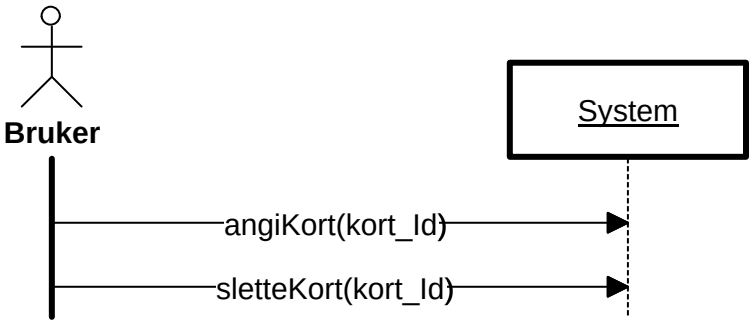
Use case: B7 - Hent data om kort



Use case: B8 - Endre kort

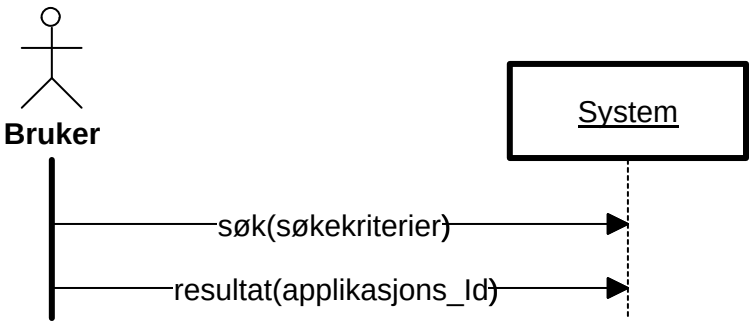


Use case: B9 - Slette kort

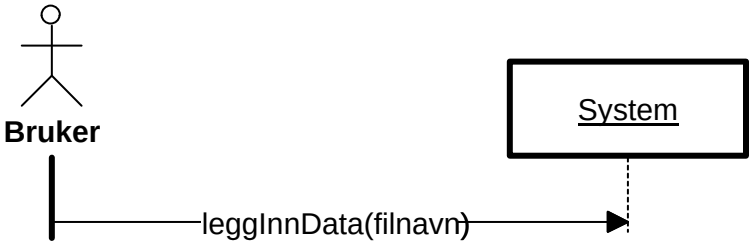


Use case: B10 - Slett applikasjon

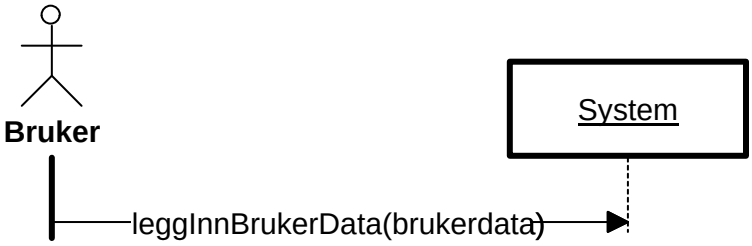
Use case: B11 - Hent data om applikasjon



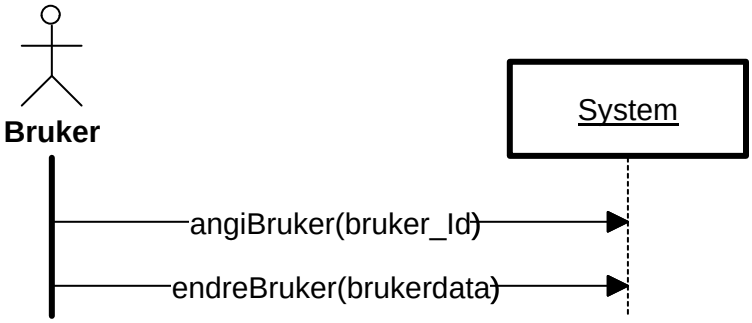
Use case: B12 - Legg inn applikasjon



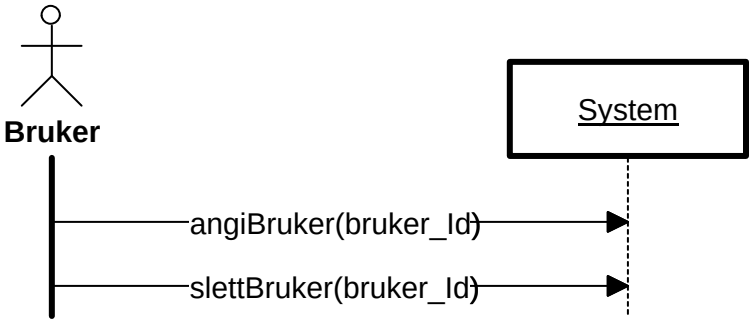
Use case: O1 - Legg inn bruker



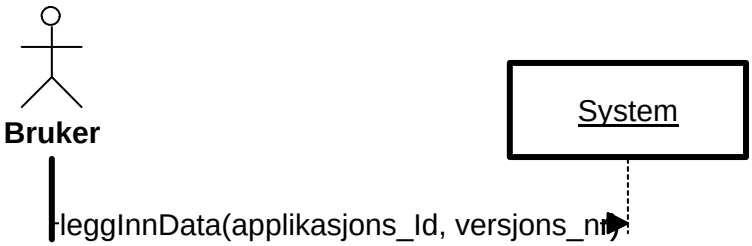
Use case: O2 - Endre bruker



Use case: O3 - Slette bruker



Use case: S1 - Klargjør for lasting av applikasjoner



2.2.3 Kontrakter

En kontrakt er et dokument som beskriver hva en operasjon skal gjøre. Kontrakten detaljbeskriver funksjonene angitt i systemsekvensdiagrammet.

Use case: B1 - Endre kortinnehaver.

Navn:	**angiKortinnehaver(kortinnehaver_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot kortinnehaver_id for kortet som skal endres. Søke etter angitt kortinnehaver mot databasen.
Type:	System
Kryssreferanser:	Use case: Endre kortinnehaver.
Merknad:	
Unntak:	Hvis ikke kortinnehaveren eksisterer, må systemet gi en feilmelding om dette.
Utdata:	
Prebetingelser:	Databasen eksistere.
Postbetingelser:	

Navn:	**endreKortinnehaver(kortinnehaverdata)**
Ansvar:	Funksjonen skal sjekke at alle felter har gyldige verdier. Den skal oppdatere riktige tabeller i databasen.
Type:	System.
Kryssreferanser:	Use case: Endre kortinnehaver.
Merknad:	
Unntak:	Systemet må gi en feilmelding ved ugyldig inntastet data og hvis feil på overføring til databasen.
Utdata:	Data sendes til databasen.
Prebetingelser:	Kortinnehaveren må eksistere i databasen.
Postbetingelser:	Databasen er modifisert med nye data.

Use case: B2 - Slette kortinnehaver.

Navn:	**angiKortinnehaver(kortinnehaver_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot kortinnehaver_id for kortinnehaveren som skal slettes.
Type:	System
Kryssreferanser:	Use case: Slette kortinnehaver.
Merknad:	
Unntak:	Hvis ikke kortinnehaveren eksisterer, må systemet gi en feilmelding om dette.
Utdata:	
Prebetingelser:	Systemet er klart til å ta imot data.
Postbetingelser:	

Navn:	**slettKortinnehaver(kortinnehaver_Id)**
Ansvar:	Funksjonen skal sørge for å slette angitt kortinnehaver fra databasen.
Type:	System
Kryssreferanser:	Use case: Slette kortinnehaver.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under slettingen.
Utdata:	
Prebetingelser:	Kortinnehaveren som skal slettes må finnes i databasen.
Postbetingelser:	Kortinnehaveren blir slettet fra databasen.

Use case: B3 - Hent data om kortinnehaver.

Navn:	**søk(søkekriterier)**
Ansvar:	Funksjonen skal sørge for at et søk mot databasen med hensyn på inntastede søkekriterier blir utført.
Type:	System
Kryssreferanser:	Use case: Hente data om kortinnehaver.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under søket mot databasen.
Utdata:	
Prebetingelser:	Søkekriteriene må være tastet inn.
Postbetingelser:	

Navn:	**resultat(kortinnehaver_Id)**
Ansvar:	Funksjonen skal sørge for å presentere søkeresultater på skjerm ut fra søk.
Type:	System
Kryssreferanser:	Use case: Hente data om kortinnehaver.
Merknad:	
Unntak:	Dersom det har oppstått en feil ved søket.
Utdata:	
Prebetingelser:	Databasesøket må være utført.
Postbetingelser:	

Use case: B4, O4- Logge på

Navn:	**bekreftPålogging(brukernavn, passord)**
Ansvar:	Funksjonen skal sørge for å sjekke brukernavn og passord, gi riktige rettigheter avhengig av om det er en bruker eller operatør som har logget på.
Type:	System
Kryssreferanser:	Use case: Logge på
Merknad:	
Unntak:	Systemet må kunne gi en feilmelding dersom det oppstår feil på brukernavn eller passord.
Utdata:	
Prebetingelser:	Funksjonen må ha mottatt brukernavn og passord. Systemet må ha tilgang til databasen som inneholder brukernavn og passord.
Postbetingelser:	Systemet logger aktuell bruker på systemet.

Use case: B5 - Legg inn kortinnehaver/kortinformasjon manuelt.

Navn:	**leggInnData(kortinnehaver/kort informasjon)**
Ansvar:	Funksjonen skal sørge for å sjekke at inntastet data er gyldig og å legge data inn i riktige tabeller i databasen.
Type:	System
Kryssreferanser:	Use case: Legg inn kortinnehaver/kortinformasjon manuelt.
Merknad:	
Unntak:	Hvis feil på overføring til databasen, må systemet gi en Feilmelding. Systemet må også gi feil ved ugyldig inntastet data.
Utdata:	Data sendes til databasen.
Prebetingelser:	Nødvendig informasjon må være tastet inn.
Postbetingelser:	Databasen er modifisert med nye data.

Use case: B6 - Legg inn kortinnehaver/kortinformasjon fra fil.

Navn:	**leggInnData(filnavn)**
Ansvar:	Funksjonen skal sørge for at valgt fil fra dialogboksen blir åpnet. Funksjonen skal også sørge for å legge data inn i de riktige tabellene i databasen.
Type:	System
Kryssreferanser:	Use case: Legg inn kortinnehaver/kortinformasjon fra fil.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis filen er på feil format eller ikke lesbar av andre grunner. Hvis feil på overføring til databasen, må systemet gi en feilmelding.
Utdata:	Data sendes til databasen.
Prebetingelser:	Filen som skal lastes og databasen må eksistere.
Postbetingelser:	Databasen er modifisert med nye data.

Use case: B7 - Hent data om kort.

Navn:	**søk(søkekriterier)**
Ansvar:	Funksjonen skal sørge for at et søk mot databasen med hensyn på inntastede søkekriterier blir utført.
Type:	System
Kryssreferanser:	Use case: Hente data om kort
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under søket mot databasen.
Utdata:	
Prebetingelser:	Søkekriteriene må være tastet inn.
Postbetingelser:	

Navn:	**resultat(kort_Id)**
Ansvar:	Funksjonen skal sørge for å presentere søkeresultater på skjerm ut fra forespørsel.
Type:	System
Kryssreferanser:	Use case: Hente data om kort.
Merknad:	
Unntak:	Dersom det har oppstått en feil ved søket.
Utdata:	
Prebetingelser:	Databasesøket må være utført.
Postbetingelser:	

Use case: B8 - Endre kort.

Navn:	**angiKort(kort_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot kort_id for kortet som skal endres.
Type:	System
Kryssreferanser:	Use case: Endre kort.
Merknad:	
Unntak:	Hvis ikke kortet eksisterer, må systemet gi en feilmelding på dette.
Utdata:	
Prebetingelser:	Systemet må være klart for å ta imot data.
Postbetingelser:	

Navn:	**endreKort(kortdata)**
Ansvar:	Funksjonen skal sjekke at alle felter har gyldige verdier. Den skal oppdatere riktige tabeller i databasen
Type:	System
Kryssreferanser:	Use case: Endre kort.
Merknad:	
Unntak:	Systemet må gi en feilmelding ved ugyldig inntastet data og hvis feil på overføring til databasen
Utdata:	Data sendes til databasen.
Prebetingelser:	Kort og database må eksistere.
Postbetingelser:	Databasen er modifisert med nye data.

Use case: B9 - Slette kort.

Navn:	**angiKort(kort_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot kort_id for kortet som skal slettes.
Type:	System
Kryssreferanser:	Use case: Slette kort.
Merknad:	
Unntak:	Hvis ikke kortet eksisterer, må systemet gi en feilmelding om dette.
Utdata:	
Prebetingelser:	Systemet er klart til å ta imot data.
Postbetingelser:	

Navn:	**slettKort(kort_Id)**
Ansvar:	Funksjonen skal sørge for å slette angitt kort fra databasen.
Type:	System
Kryssreferanser:	Use case: Slette kort.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under slettingen.
Utdata:	Data sendes til databasen.
Prebetingelser:	Kortet som skal slettes må finnes i databasen.
Postbetingelser:	Kortet blir slettet fra databasen.

Use case: B10 - Slett applikasjon

Navn:	**angiApplikasjon(applikasjons_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot applikasjons_Id til applikasjonen som skal slettes fra databasen.
Type:	System
Kryssreferanser:	Use case: Slett applikasjon.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis applikasjonen ikke finnes i databasen.
Utdata:	
Prebetingelser:	Systemet må være klart til å ta imot data.
Postbetingelser:	

Navn:	**slettApplikasjon(applikasjons_Id)**
Ansvar:	Funksjonen skal sørge for å slette angitt applikasjon fra databasen.
Type:	System
Kryssreferanser:	Use case: Slett applikasjon.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under sletting av applikasjonen.
Utdata:	Data sendes til databasen.
Prebetingelser:	Applikasjonen som skal slettes må finnes i databasen.
Postbetingelser:	Applikasjonen blir slettet fra databasen

Use case: B11 - Hent data om applikasjon

Navn:	**søk(søkekriterier)**
Ansvar:	Funksjonen skal sørge for at et søk mot databasen med hensyn på inntastede søkekriterier blir utført.
Type:	System
Kryssreferanser:	Use case: Hent data om applikasjon.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår en feil under søket mot databasen.
Utdata:	
Prebetingelser:	Søkekriterier må være tastet inn.
Postbetingelser:	

Navn:	**resultat(applikasjons_Id)**
Ansvar:	Funksjonen skal sørge for at data om applikasjonen blir presentert på skjerm.
Type:	System
Kryssreferanser:	Use case: Hent data om applikasjon.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår en feil under søket.
Utdata:	
Prebetingelser:	Databasesøket må være utført.
Postbetingelser:	

Use case: B12 - Legg inn applikasjon

Navn:	**leggInnData(filnavn)**
Ansvar:	Funksjonen skal sørge for å velge og åpne fil fra dialogboksen. Funksjonen skal også sørge for å legge informasjon om applikasjonen inn i riktige tabeller i databasen. Det skal også lages en link fra databasen til ALU(Application Load Unit) og ALC (Application Load Certificate), som ligger lagret på fil.
Type:	System
Kryssreferanser:	Use case: Legg inn applikasjon.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under overføring til databasen eller hvis filen er på feil format eller ikke lesbar av andre grunner.
Utdata:	Data sendes til databasen.
Prebetingelser:	Fila og databasen må eksistere.
Postbetingelser:	Database modifisert med nye data.

Use case: O1 - Legg inn bruker.

Navn:	**leggInnBrukerdata(brukerdata)**
Ansvar:	Funksjonen skal kunne håndtere registrering av nye brukere på systemet. Funksjonen skal også kunne gå gjennom alle feltene og sjekke at alle inntastede verdier er gyldige. Funksjonen skal også legge data inn i riktige tabeller i databasen.
Type:	System
Kryssreferanser:	Use case: Legg inn bruker.
Merknad:	
Unntak:	Systemet skal gi en feilmelding dersom inntastede verdier. er ugyldige.
Utdata:	
Prebetingelser:	Data må være tastet inn av operatøren.
Postbetingelser:	En ny bruker er lagt til i databasen.

Use case: O2 - Endre bruker.

Navn:	**angiBruker(bruker_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot bruker_id for bruker som skal endres.
Type:	System
Kryssreferanser:	Use case: Endre bruker.
Merknad:	
Unntak:	Hvis ikke brukeren eksisterer, må systemet gi en feilmelding om dette.
Utdata:	
Prebetingelser:	Systemet må være klart for å ta imot data.
Postbetingelser:	

Navn:	**endreBruker(brukerdata)**
Ansvar:	Funksjonen skal sjekke at alle felter har gyldige verdier. Den skal oppdatere riktige tabeller i databasen
Type:	System
Kryssreferanser:	Use case: Endre bruker.
Merknad:	
Unntak:	Systemet må gi en feilmelding ved ugyldig inntastet data og hvis feil på overføring til databasen.
Utdata:	Data sendes til databasen.
Prebetingelser:	Brukeren og databasen må eksistere.
Postbetingelser:	Databasen er modifisert med nye data.

Use case: O3 - Slette bruker.

Navn:	**angiBruker(bruker_Id)**
Ansvar:	Funksjonen skal sørge for å ta imot bruker_id for bruker som skal slettes.
Type:	System
Kryssreferanser:	Use case: Slette bruker.
Merknad:	
Unntak:	Hvis ikke bruker eksisterer, må systemet gi en feilmelding om dette.
Utdata:	
Prebetingelser:	Systemet er klart til å ta imot data.
Postbetingelser:	

Navn:	**slettBruker(bruker_Id)**
Ansvar:	Funksjonen skal sørge for å slette angitt bruker fra databasen.
Type:	System
Kryssreferanser:	Use case: Slette bruker.
Merknad:	
Unntak:	Systemet skal gi en feilmelding hvis det oppstår feil under slettingen.
Utdata:	Data sendes til databasen.
Prebetingelser:	Bruker som skal slettes må finnes i databasen.
Postbetingelser:	Bruker blir slettet fra databasen.

Use case: S1 - Klargjør for lasting av applikasjoner

Navn:	**hentDataomApplikasjon(applikasjons_Id, versjon)**
Ansvar:	Funksjonen skal sørge for å ta imot applikasjonsnavn eller ID og versjon som innparameter. Funksjonen skal også sørge for å hente lagrede data fra databasen, og returnere dette i form av et EJB-objekt.
Type:	System
Kryssreferanser:	Use case: Klargjør for lasting av applikasjon.
Merknad:	
Unntak:	Dersom det har oppstått feil ved kommunikasjon mot databasen.
Utdata:	
Prebetingelser:	Funksjonen må vite applikasjons_Id og versjon.
Postbetingelser:	

2.2.3 Overordnede operasjonelle systemkrav

2.2.3.1 Normal operasjon

2.2.3.1.1 Ytelse
Systemet må kunne håndtere mange kort og brukere. Databaseserveren har kapasitet til denne datahåndteringen.

2.2.3.1.2 Sikkerhet
Feilhåndtering: Systemet skal kjøre kontroller på at verdiene som blir behandlet er lovlige og gi lettforståelige feilmeldinger dersom så ikke er tilfelle

Uvedkomne: Systemet skal føre logg på alle transaksjoner og hendelser i systemet. For å logge seg på systemet må man oppgi gyldig brukernavn og passord.

Backup vil kjøres en gang per uke siden dataene ikke forandrer seg vesentlig.

2.2.3.1.3 Oppstart
Systemet skal være ferdigtestet og klar til bruk 16. mai. Det er et testsystem beregnet kun til intern bruk og er derfor ikke en del av det aktive systemet. En eventuell overlapping ved bytting av system er ikke aktuell for vår del.

2.2.3.1.4 Innebygde tester
Systemet skal ved feil- og eller kritiske situasjoner gi tilbakemelding til bruker av systemet. Disse feilmeldingene vil vises ved hjelp av dialogbokser i applikasjonen.

2.2.3.2 Operasjon i feilsituasjoner
I alle feilsituasjoner skal det føres logg. Dette vil gjelde applikasjonsserversiden, databaseserveren og brukerfeil.

2.3 BEGRENSNINGER

2.3.1 Software design begrensninger

2.3.1.1 Software standarder og språk

Implementasjonsspråket som skal benyttes er Java. Modulene skal utvikles som Java-applikasjoner.

Ovenfor alle funksjoner og klasser skal det kommenteres hva denne inneholder. Dokumentasjon og kommentering skal være på norsk.

2.3.1.2 Software grensesnitt

Vi bruker EJB for å utvikle grensesnittet mellom de forskjellige modulene. For å utvikle det grafiske brukergrensesnittet skal standard Java benyttes. Samme grafiske brukergrensesnitt skal brukes på alle moduler. Det grafiske skal være enkelt å sette seg inn i.

2.3.1.3 Software pakker/verktøy

Databasen utvikles i Oracle og modulene som kommuniserer med den utvikles i Java. For å utarbeide modulenes grafiske grensesnitt benyttes Jbuilder. Applikasjonsserveren er JBoss.

2.3.1.4 Software kommunikasjonsstandarder og grensesnitt

Utgangspunkt til internt bruk hos Ergo Group. Løsningen skal utvikles som en applikasjon med grafisk brukergrensesnitt. RMI er grensesnittet mellom EJB objekter.

2.3.1.5 Database

Ergo Group har per i dag en eksisterende database. Vi skal ta utgangspunkt i den eksisterende databasen og lage en egen versjon av denne. For applikasjonsdelen eksisterer det ikke noen database i dag, denne må vi utvikle selv.

2.3.1.6 Operativsystem

Systemet skal i utgangspunktet gå på Windows NT, men er plattformuavhengig. Ingen krav til at systemet skal gå på et spesielt operativsystem.

2.3.1.7 Toleranse, marginer og muligheter/tilfeller

Eventuelle begrensninger vil være kapasiteten til databasen. Systemet skal kunne håndtere mange kort og kortinnehavere. For vår testversjon av systemet vil det ikke dreie seg om så store mengder data at databasen ikke vil kunne ta hånd om dette.

2.3.2 Hardware design begrensninger

Systemet vil bestå av datamaskiner med Windows NT plattform som er koblet opp i et nettverk mot en server. Problemer med dataoverføring kan oppstå dersom mange klienter forsøker å kommunisere med databasen samtidig. For vår del vil det dreie seg om få klienter så overføringshastigheten vil ikke fremstå som noe stort problem. Det vil ellers ikke være noen hardware begrensninger for vårt system.

2.3.3 Brukerdesign begrensninger

Brukerne av systemet har høy IT kompetanse. Det anses likevel som hensiktsmessig at brukerprogrammene har en enkel og selvforklarende layout og at de inneholder enkle hjelpefunksjoner.

2.4 ASPEKTER OMKRING LIVSSYKLUS

2.4.1 Dokumentasjon

Det må integreres egne hjelpefunksjoner for systemet. Dokumentasjon av kode må foreligge for videre vedlikehold. Denne dokumentasjonen skal foreligge i Word format.

2.4.2 Modul og integrasjonstesting

Hver modul skal testes hver for seg, deretter må modulene integreres og testes sammen. Systemet må testes for både riktig og feil input. Testverdier inn og forventet verdier ut samt forventede verdier på søk. Til slutt vil det kjøres en systemtest. De ulike bean teknologiene skal testes mot hverandre. Dette gjøres ved å teste de ulike modulene som er utviklet i de forskjellige teknologiene.

2.4.3 Konfigurasjons- og versjons styring

Endringer, oppdateringer og nye versjoner av kode og dokumenter skal påføres dato, navn og versjonsnummer i filnavnet. Følgende format skal brukes: Filnavn_DDMMAA_Initialer_versjon.

Det skal foreligge utskrift av modulene når de er ferdige. Sikkerhetskopier skal ligge under prosjektområdet på skolen, samt lokalt på egne maskiner.

2.4.4 Krav til support, service og vedlikehold

Små krav til service og vedlikehold. Ergo Group vil selv ta seg av service og vedlikehold av systemet hvis systemet skal utvikles videre.

2.4.5 Krav til utvidelser

Det må legges til rette for at systemet kan utvides. Siden EJB benyttes vil det være enkelt å utvide systemet. Moduler det vil være aktuelt å utvide systemet med vil være personaliseringsmoduler.

2.5 ASPEKTER OMKRING INSTALLASJON

Systemet skal fungere som et separat testsystem. Det vil ikke være noen krav til omlegging. Opplæringsbehovet vil være lite siden brukeme er godt kjent med data.

2.6 UTGIVELSER UNDERVEIS

Første versjon av ferdige moduler skal leveres til oppdragsgiver for vurdering og tilbakemeldinger.

2.7 AKSEPTANSE KRAV

Systemet skal fungere som beskrevet i kravspesifikasjonen. Det vil si at det skal være mulig blant annet å registrere kortinnehavere (personer), kort og applikasjoner, og hente ut opplysninger om disse. Det nye systemet skal være utviklet med bruk av EJB.

3.0 DESIGNDOKUMENT

3.1 INTRODUKSJON

Formålet med designdokumentet er å skal gi en oversikt over arkitekturen og brukergrensesnittet til systemet. Designdokumentet tar utgangspunkt i krav, konsepter og relasjoner beskrevet i kravspesifikasjonen. Dokumentet skal også beskrive alle data til og fra systemet. Vi vil ta utgangspunkt i objektorientert design under utviklingen av designdokumentet.

3.1.1 Mål for systemet

Systemet skal være brukervennlig og ha kort responstid. Det er viktig at systemet er lett å utvide med tanke på videreutvikling. Det er også ønskelig at det skal være lett å drive vedlikehold og gjenbruk på systemet.

3.2 DATABASE DESIGN

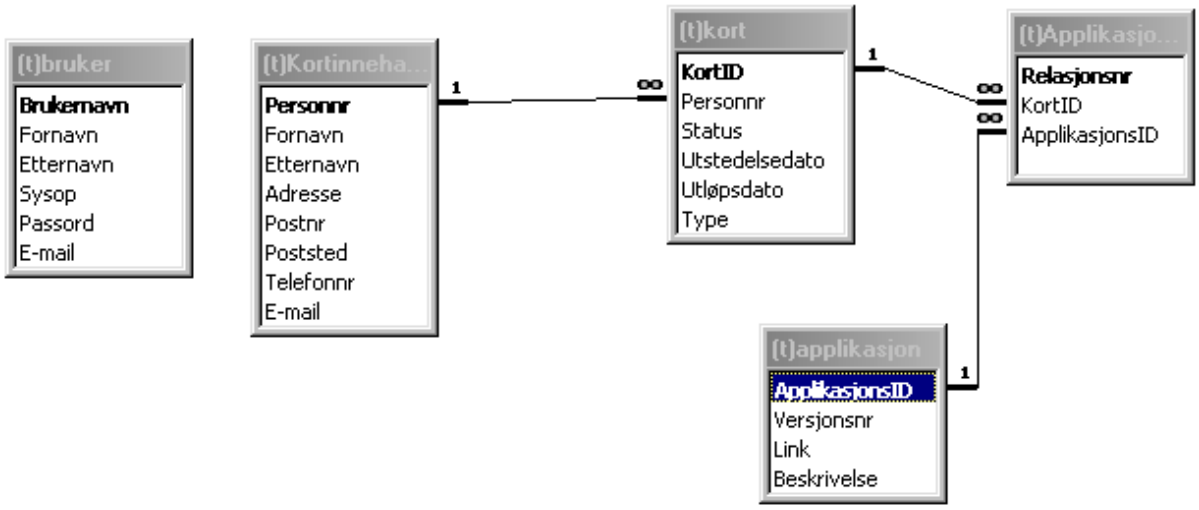
3.2.1. Termer og definisjoner

Term	Standard	Definisjon
		Dette symbolet beskriver en tabell i databasen. I dette eksemplet er (t)applikasjon navnet på tabellen, mens Versjonsnr, Link, osv er de forskjellige feltene tabellen består av. Det uthevede feltet utgjør primærnøkkelen i tabellen.
		Dette symbolet brukes i denne sammenhengen for å beskrive relasjoner mellom tabeller i databasen. Eksemplet her viser en en-til-mange relasjon mellom to tabeller

3.2.2. Overordnet modell

Databasen SCMS innholder informasjon om kort, kortholdere, applikasjoner, brukere av systemet og kortholdere og informasjon knyttet til disse fem begrepene som trengs for å kunne personalisere kort. med tilhørende tabeller og hvordan de hører sammen.

3.2.2.1 Databasen SCMS



Figur 3.1

3.2.3. Detaljert beskrivelse

Figuren (figur 3.1) over viser hvordan de forskjellige tabellene i databasen henger sammen. I tillegg viser den hvilke felter som finnes i de forskjellige tabellene. Under følger en mer utfyllende beskrivelse av hver enkelt tabell og hvert enkelt felt.

3.2.3.1 (T)Bruker

Tabellen (T)Bruker innholder informasjon om brukerne av systemet. Disse blir lagt inn av en systemoperatør som kontrollerer systemet.

Feltnavn	Datatype	Beskrivelse
Brukernavn	Varchar(10)	Brukerens påloggingsnavn i systemet. Primærnøkkel i tabellen
Fornavn	Varchar(25)	Brukers fornavn
Etternavn	Varchar(25)	Brukers etternavn
Sysop	Varchar(3)	Et ja/nei felt som angir om bruker er systemoperatør
Passord	Varchar(11)	Brukerens passord i systemet
Email	Varchar(40)	Brukerens email-adresse

3.2.3.2 (T)Kortinnehaver:

Tabellen (T)Kortinnehaver innholder informasjon om kortholderne. Dette blir enten tastet inn manuelt av en bruker av systemet eller importert fra fil.

Feltnavn	Datatype	Beskrivelse
Personnr	Char(11)	Kortholderens personnummer. Primærnøkkel i tabellen
Fornavn	Varchar(25)	Kortholderens fornavn
Etternavn	Varchar(25)	Kortholderens etternavn
Adresse	Varchar(30)	Kortholderens adresse
Postnr	Char(4)	Kortholderens postnr
Poststed	Varchar(30)	Kortholders poststed
Telefonnr	Char(8)	Kortholderens telefonnummer
Email	Varchar(40)	Kortholderens email-adresse

3.2.3.3 (T)Kort:

Tabellen (T)Kort innholder informasjon om kortene som sirkulerer blant kortinnehaverne. Dette blir enten tastet inn manuelt av en bruker av systemet eller importert fra fil.

Feltnavn	Datatype	Beskrivelse
KortID	Char(11)	Kortets ID i systemet. Primærnøkkel i tabellen
Personnr	Char(11)	Personnummeret til tilhørende kortinnehaver
Status	Varchar(8)	Statusfelt som angir om kortet er gyldig eller ikke
Utstedelsedato	Varchar(8)	Datofelt som angir når kortet ble utstedt
Utløpsdato	Varchar(8)	Datofelt som angir når kortet utløper
Type	Char(4)	Felt som angir hva slags type kort det er.

3.2.3.4 (T)Applikasjon:

Tabellen (T)Applikasjon innholder informasjon om applikasjonene som er tilgjengelig for å lastes på kortene. Dette blir lastet inn fra fil inn i databasen.

Feltnavn	Datatype	Beskrivelse
AppliksjonsID	Char(5)	Applikasjonens ID i systemet. Primærnøkkel i tabellen
Versjonsnr	Varchar(5)	Applikasjonens versjonsnr
Link	Varchar(20)	Beskrivelse av hvor applikasjonen ligger i systemet
Beskrivelse	Varchar(64)	Beskrivelse av applikasjonen

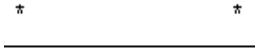
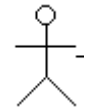
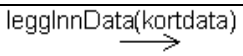
3.2.3.5 (T)Applikasjonslinje:

Tabellen (T)Applikasjonslinje innholder informasjon om hvilke applikasjoner som ligger på de ulike kortene. Et kort kan inneholde mange applikasjoner og en applikasjon kan ligge på mange kort.

Feltnavn	Datatype	Beskrivelse
KortID	Char(11)	Kortets ID, en av to primærnøkler i tabellen samt fremmednøkkel som er koblet mot tabellen Kort.
ApplikasjonsID	Char(5)	Applikasjonens ID, den andre primærnøkkelen i tabellen samt fremmednøkkel som er koblet mot tabellen Applikasjon.

3.3 ARKITEKTUR OG DESIGN

Avsnittet skal gi en inngående beskrivelse av de forskjellige modulene og funksjonene i systemet. Først vil systemet som en helhet bli beskrevet og senere vil vi gå nærmere inn på hver enkelt modul.

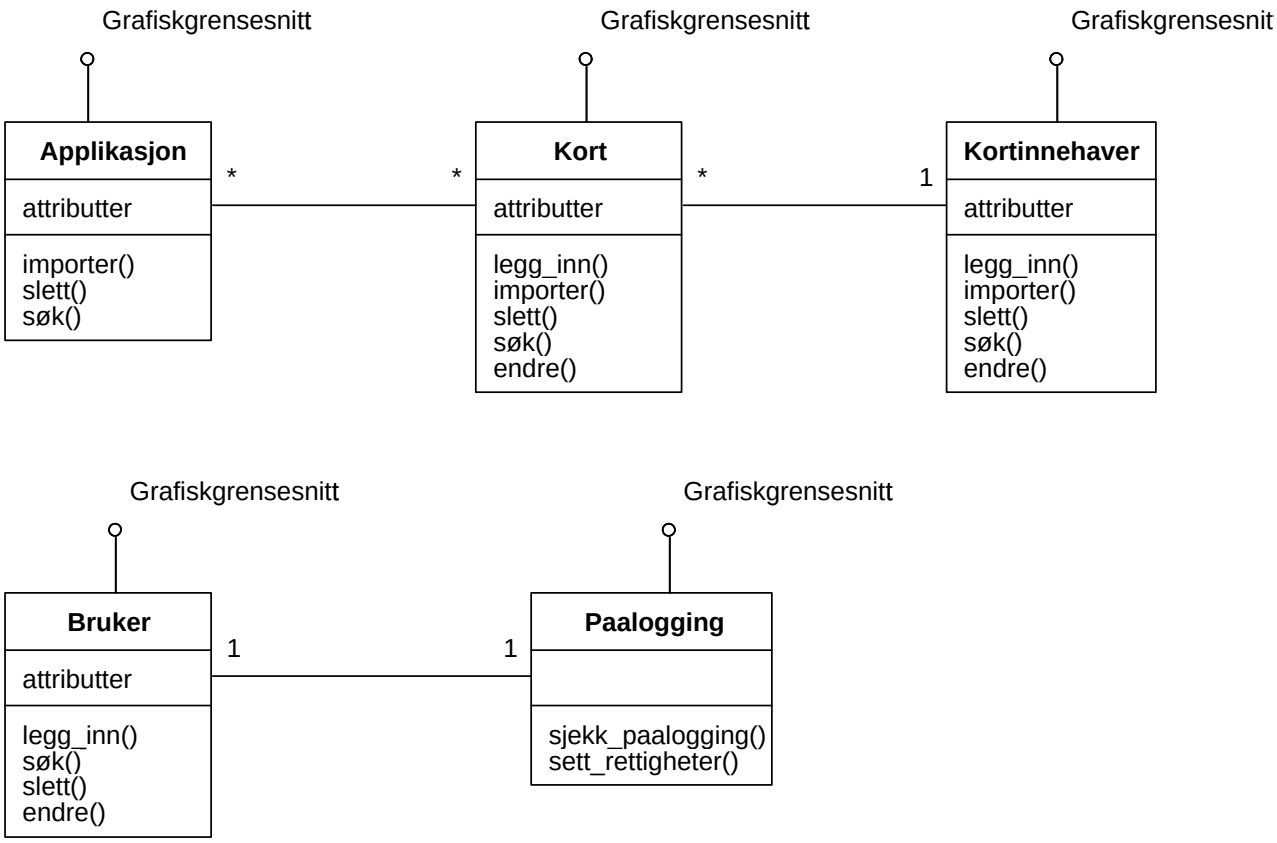
Term	Standard	Definisjon
		Dette symbolet representerer en klasse med attributter og tilhørende funksjoner i klassediagrammet. Øverst er navnet på klassen, midterste feltet inneholder attributtene og nederste feltet presenterer funksjonene i klassen.
		Dette symbolet brukes i klassediagrammet for å beskrive relasjoner mellom klasser. Eksemplet her viser en mange-til-mange relasjon mellom to klasser.
		Dette symbolet brukes i klassediagrammet for å illustrere at klassen har et tilhørende grafisk brukergrensesnitt.
		Dette symbolet brukes i kollaborasjonsdiagrammene og illustrerer at en systemoperasjon trigges.
		Dette symbolet brukes i kollaborasjonsdiagrammene og illustrerer en funksjon og hvilken retning den går. I dette tilfellet leggInnData med parameterne kortdata.
		Dette symbolet brukes i kollaborasjonsdiagrammene og representerer en klasse.

3.3.1 Program struktur

Systemet skal utvikles ved hjelp av Enterprise Java Beans (EJB). Det finnes primært to typer EJB. Den ene typen er Entity Beans og den andre typen er Session Beans. Vårt system skal utvikles i både Session Beans og Entity Beans. Session Beans er delt inn i to typer: Stateless og Stateful. Entity Beans er også delt inn i to typer: Bean Managed Persistence (BMP) og Container Managed Persistence (CMP). En del av oppgaven vår er å teste de to teknologienes ytelse mot hverandre. Derfor har vi blitt pålagt å lage en del av systemet i BMP og den andre delen i CMP. Vi skal også kode en modul ved bruk av en Stateful Session Bean. Dette for å dekke et større område av EJB teknologien.

3.3.1.1 Klassediagram for systemet

Klassediagrammet skal gi en oversikt over de forskjellige klassene og funksjonene i systemet. Det vil også bli inkludert relasjoner mellom de forskjellige klassene.



Figur 3.2

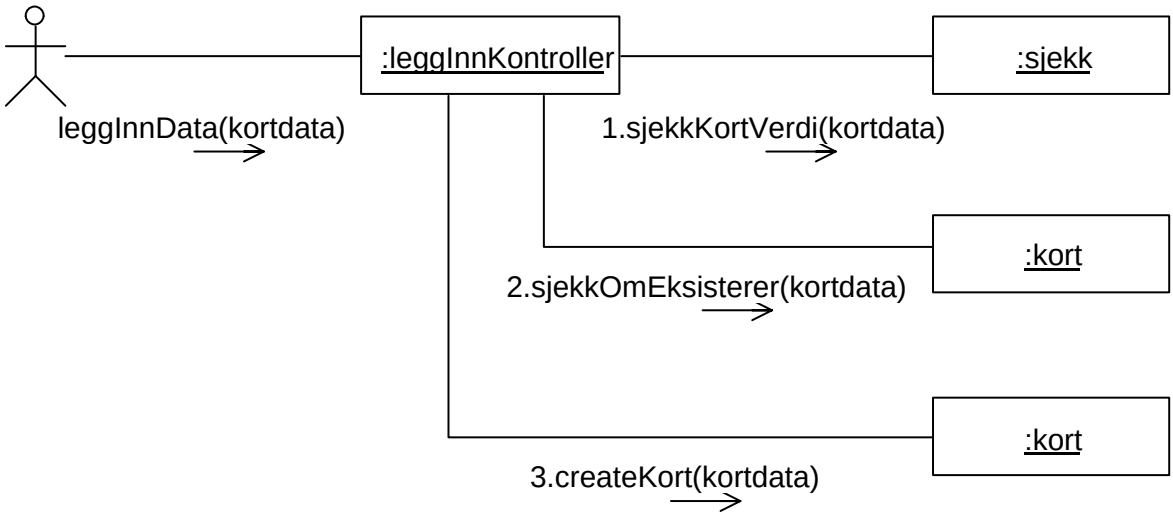
3.3.2 Beskrivelse

Kollaborasjonsdiagrammene viser hvilke klasser som er definert og flyten av informasjon mellom disse. Vi har valgt å benytte oss av noen patterns for å fordele ansvaret på flere klasser og å holde kompleksiteten på et forholdsvis lavt nivå. Denne typen diagrammer er viktig i fasen der man skal illustrere hvilke oppgaver de ulike klasser i systemet har. Kollaborasjonsdiagrammene skal gi en oversikt over informasjonsflyt, klasser og ansvar i systemet.

Det lages et kollaborasjonsdiagram for hver enkelt operasjon som systemsekvensdiagrammene beskriver. For å lage dette benyttes kontraktene fra kapittel 2.2.3 sammen med tilhørende use case. Vi vil ta for oss modul for modul derfor kommer ikke use casene i kronologisk rekkefølge. Modulene kommer i tilfeldig rekkefølge.

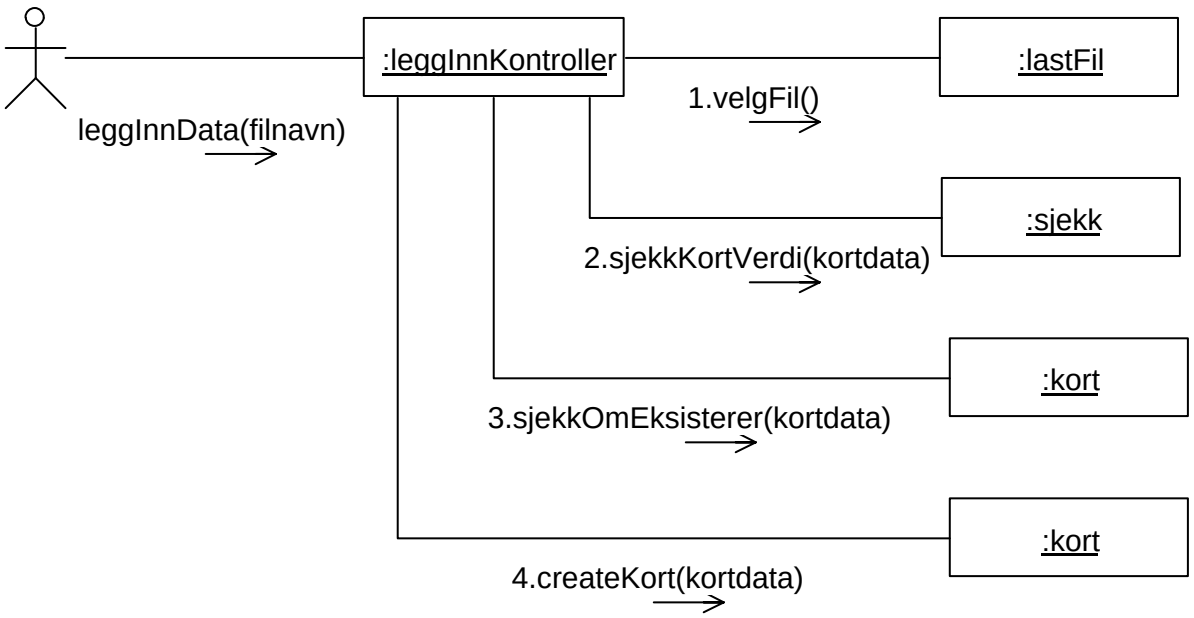
3.3.2.1 Kort

Use case: B5 - Legg inn kortinformasjon manuelt



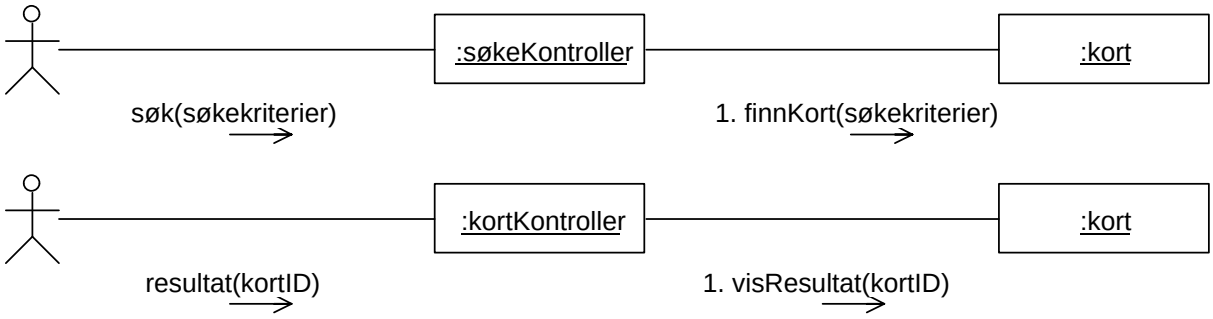
Kortinformasjon skal kunne legges inn manuelt. Kortdataene må sjekkes slik at gyldig data blir lagt inn i databasen. Før det registreres nytt kort må det gjøres en sjekk om kortet eksisterer fra før. Deretter kan det nye kortet registreres med kortdata i databasen.

Use case: B6 - Legg inn kortinformasjon fra fil



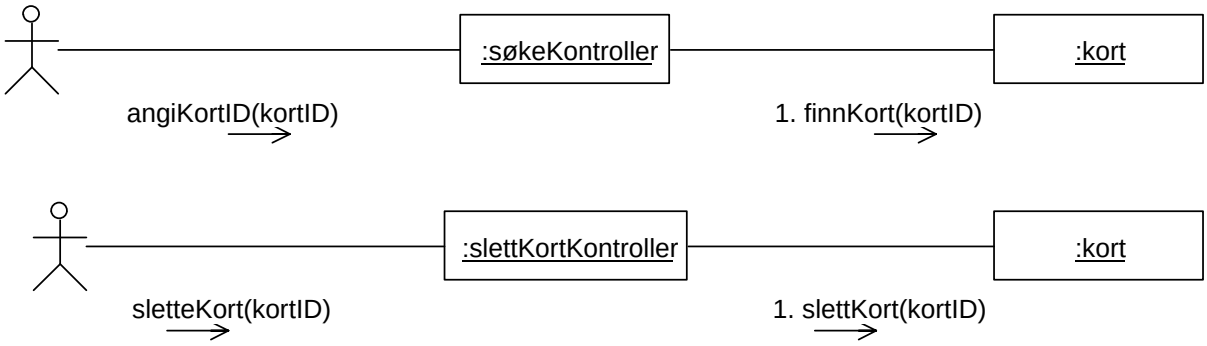
Kort skal kunne legges inn fra fil. Kortdata importeres fra fil og kortdataene må sjekkes slik at disse er gyldige. Det må også sjekkes at kortet ikke eksisterer fra før. Deretter kan et nytt kort registreres med kortdata i databasen.

Use case: B7 - Hent data om kort



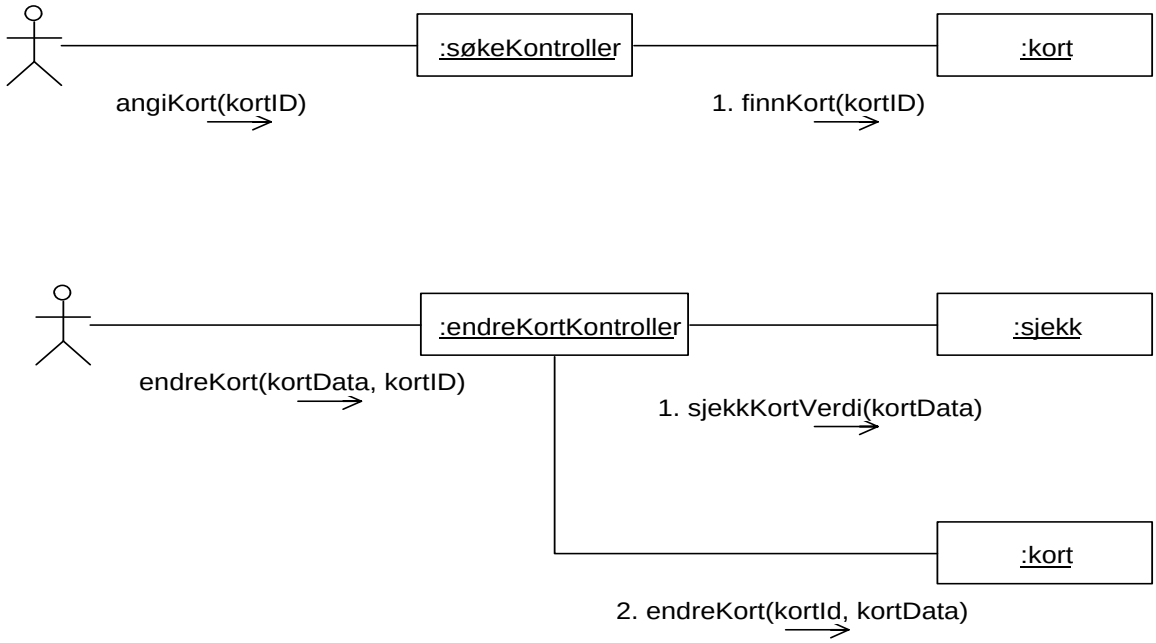
Data skal kunne hentes ut om kort ved hjelp av en forespørsel mot databasen med inntastede søkekriterier. Resultatet må presenteres til brukeren.

Use case: B9 - Slette kort



Man skal kunne slette et kort fra databasen ved å oppgi kortets ID, dersom kortet finnes vil dataene presenteres for brukeren før det eventuelt blir slettet.

Use case: B8 - Endre kort



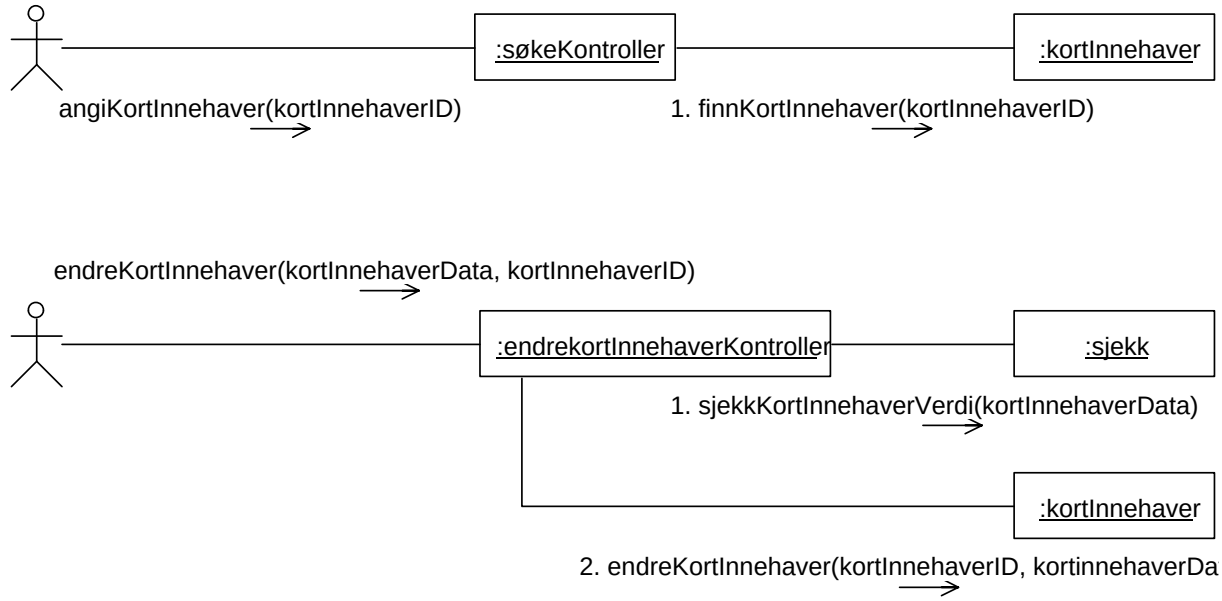
Man skal kunne endre et kort fra databasen ved å oppgi kortets ID, dersom kortet finnes vil dataene presenteres for brukeren og han har mulighet til å endre på felter. Det må testes om de nye verdiene er gyldige før de kan legges inn i databasen.

3.3.2.1.1 Algoritmisk modell

Denne modulen skal utvikles ved å benytte CMP.

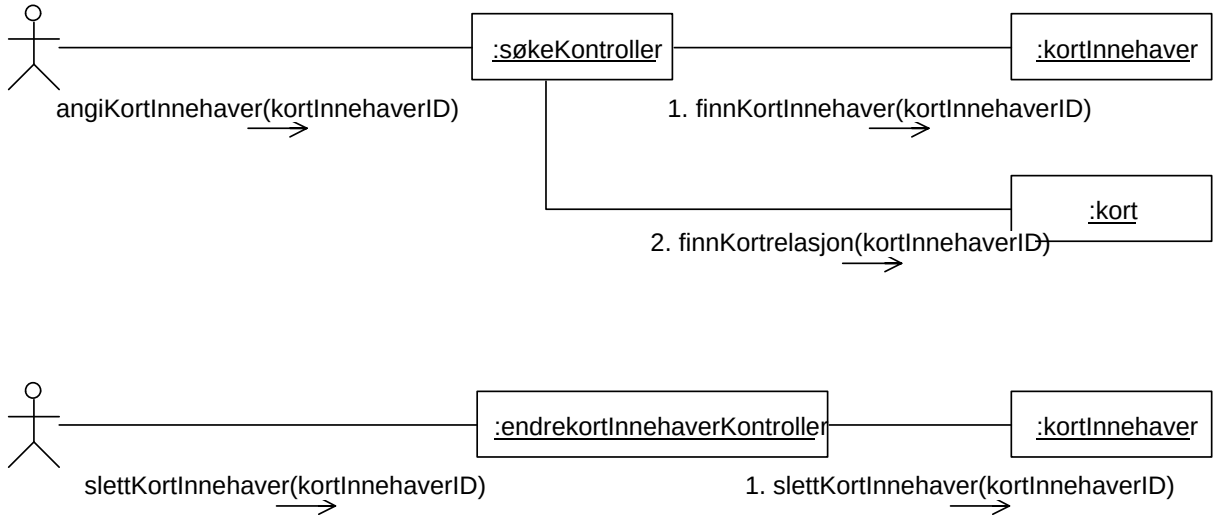
3.3.2.2 Kortinnehaver

Use case: B1 - Endre kortinnehaver



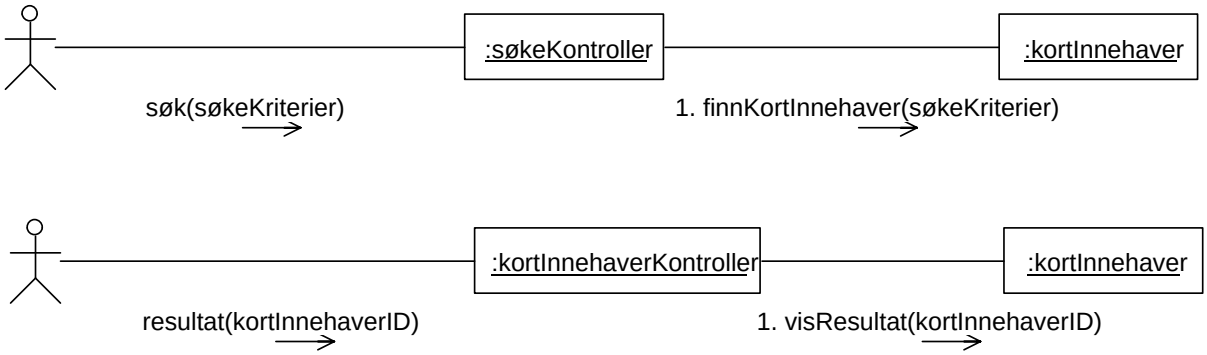
Man skal kunne endre en kortinnehaver fra databasen ved å oppgi kortinnehaverens personnummer, dersom kortinnehaveren finnes vil dataene presenteres for brukeren og han har mulighet til å endre på felter. Det må testes om de nye verdiene er gyldige før de kan legges inn i databasen.

Use case: B2 - Slette kortinnehaver



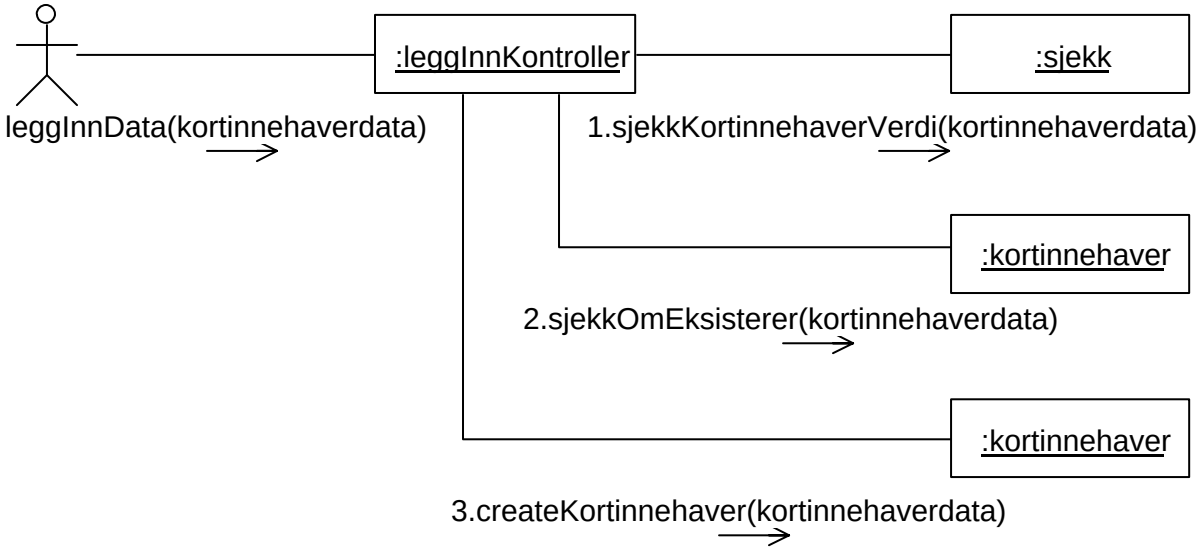
Man skal kunne slette en kortinnehaver fra databasen ved å oppgi kortinnehaverens personnummer, dersom kortinnehaveren finnes vil det sjekkes mot databasen at det ikke er registrert noen aktive kort på kortinnehaveren. Deretter vil dataene om kortinnehaveren presenteres for brukeren før de eventuelt blir slettet.

Use case: B3 - Hent data om kortinnehaver



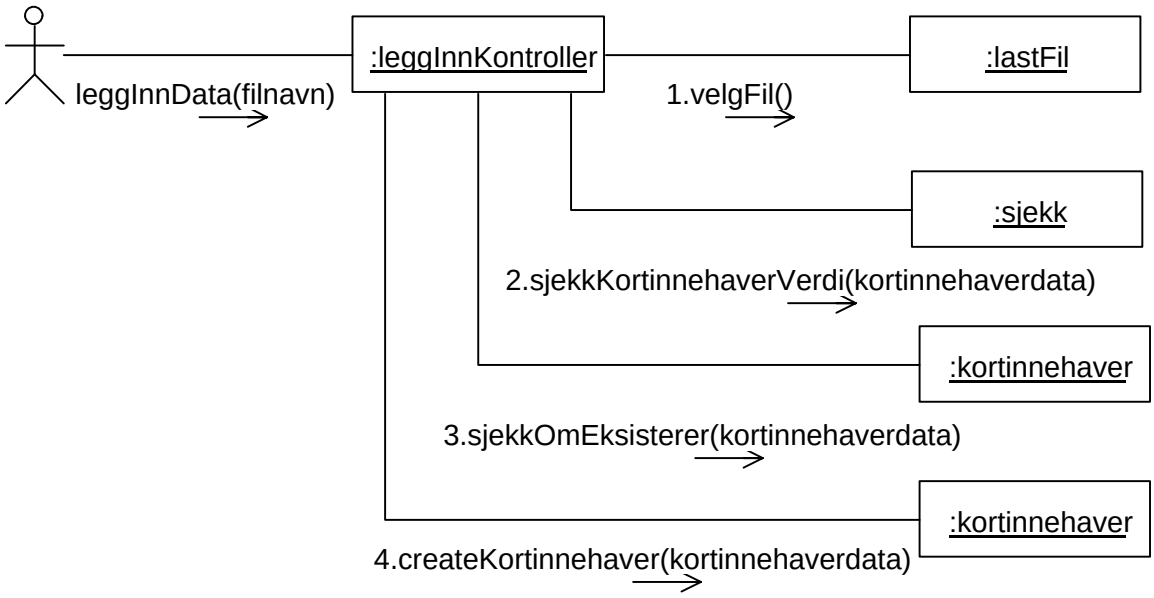
Data skal kunne hentes ut om en kortinnehaver ved hjelp av en forespørsel mot databasen med inntastede søkekriterier. Resultatet må presenteres til brukeren.

Use case: B5 - Legg inn kortinnehaver manuelt



Kortinnehaverinformasjon skal kunne legges inn manuelt. Dataene må sjekkes slik at kun gyldige verdier blir lagt inn i databasen. Før det registreres en ny kortinnehaver må det gjøres en sjekk på om han finnes fra før. Deretter kan den nye kortinnehaveren registreres med data i databasen.

Use case: B6 - Legg inn kortinnehaver fra fil

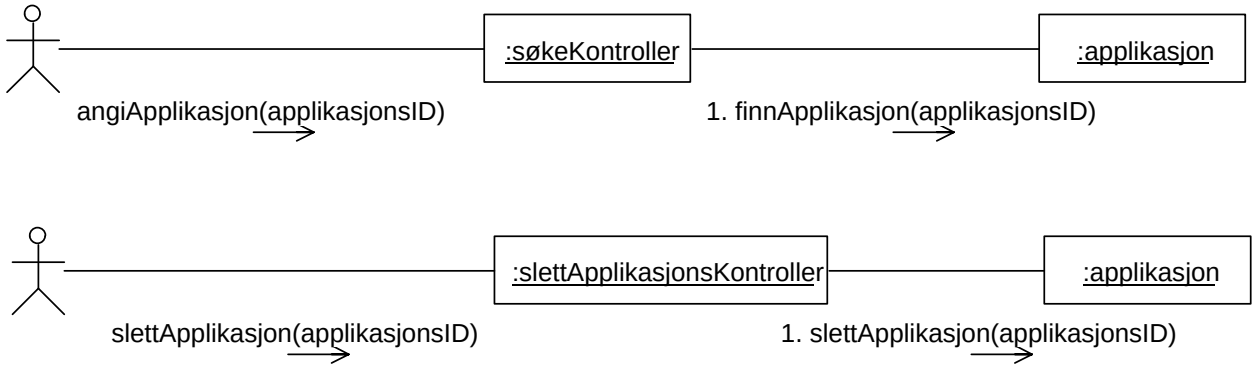


Kortinnehaver skal kunne legges inn fra fil. Kortinnehaverdata importeres fra fil og det må sjekkes at disse er gyldige. Det må også sjekkes at kortinnehaver ikke eksisterer fra før. Deretter kan en ny kortinnehaver registreres med data i databasen.

3.3.2.2.1 Algoritmisk modell
Denne modulen skal utvikles ved å benytte BMP.

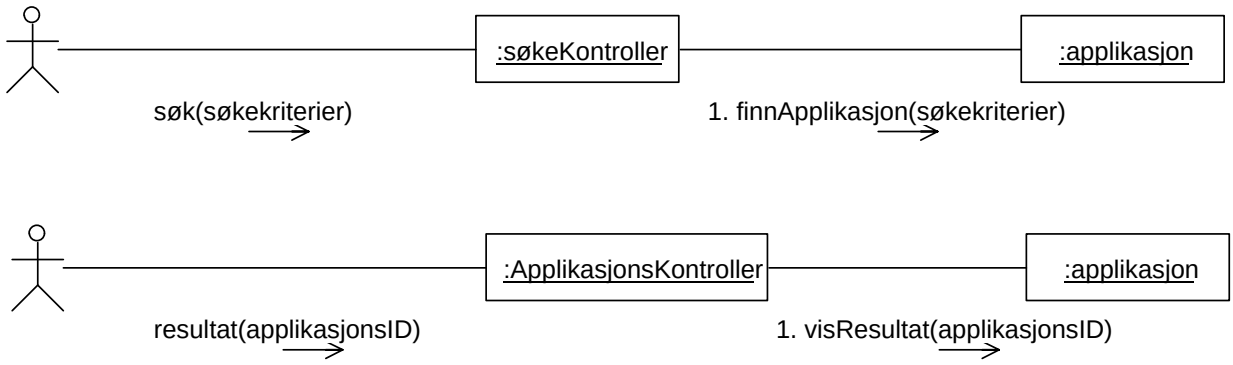
3.3.2.3 Applikasjon

Use case: B10 - Slette applikasjon



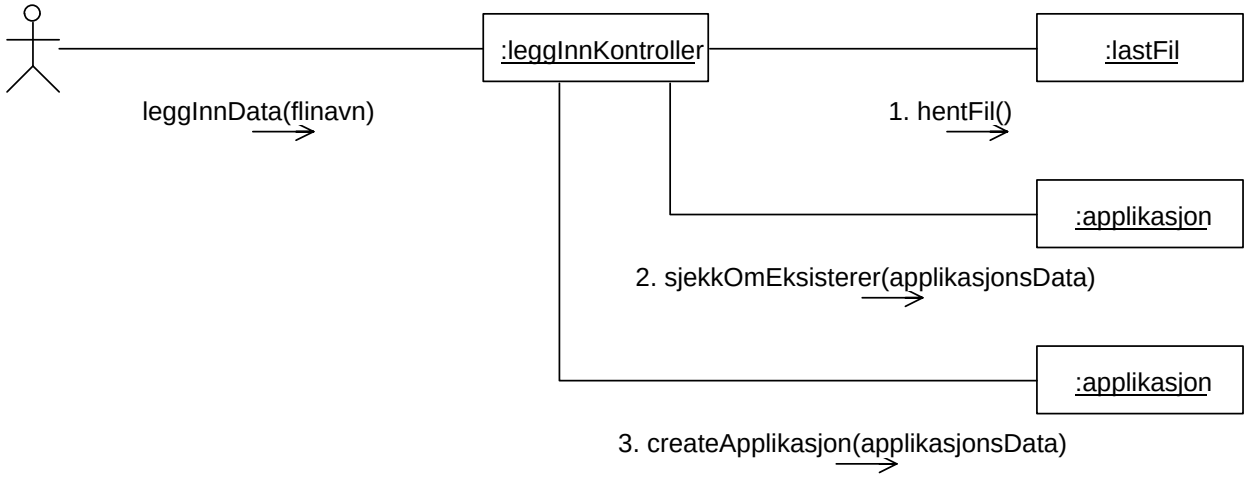
Man skal kunne slette en applikasjon fra databasen ved å oppgi applikasjonens ID, dersom applikasjonen finnes vil dataene om applikasjonen presenteres for brukeren før de eventuelt blir slettet.

Use case: B11 - Hent data om applikasjon



Data skal kunne hentes ut om en applikasjon ved hjelp av en forespørsel mot databasen med inntastede søkekriterier. Resultatet må presenteres til brukeren.

Use case: B12 - Legg inn applikasjon fra fil



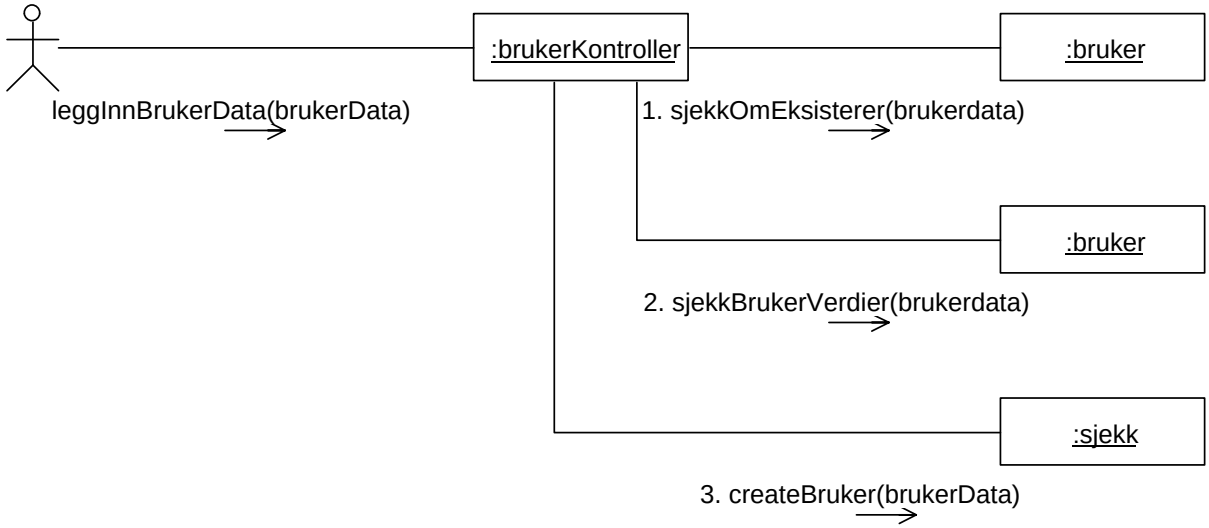
Applikasjon skal kunne legges inn fra fil. Applikasjonsdata importeres fra fil og det sjekkes at disse er gyldige. Det må også sjekkes at samme versjon av applikasjonen ikke eksisterer fra før. Deretter kan en ny applikasjon registreres med data i databasen.

3.3.2.3.1 Algoritmisk modell

Denne modulen skal utvikles ved å benytte CMP.

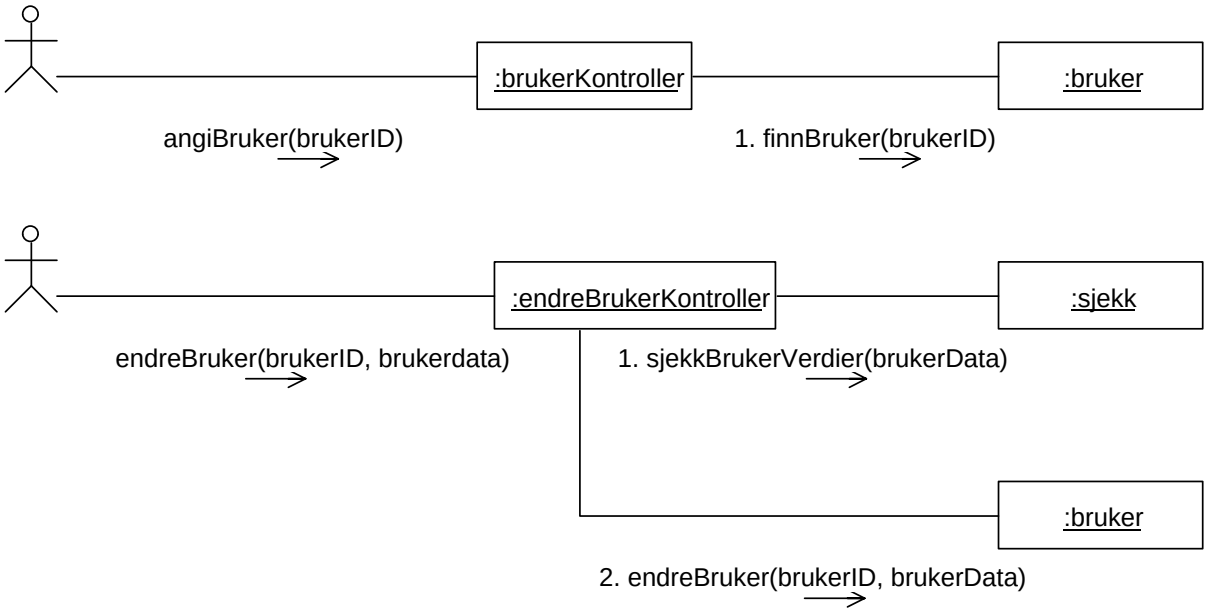
3.3.2.4 Bruker

Use case: O1 - Legg inn bruker



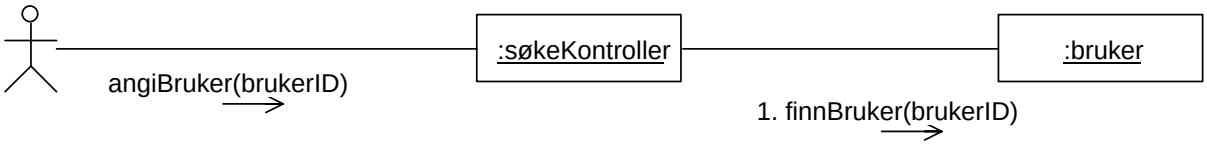
Brukerinformasjon skal legges inn manuelt. Dataene må sjekkes slik at det kun er gyldige verdier som blir lagt inn i databasen. Før det registreres en ny bruker må det gjøres en sjekk på om han finnes fra før. Deretter kan den nye brukeren registreres med brukerdata i databasen.

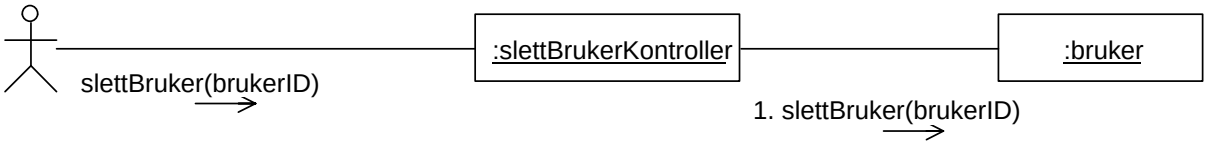
Use case: O2 - Endre bruker



Man skal kunne endre en bruker fra databasen ved å oppgi brukerens ID, dersom brukeren finnes vil dataene presenteres for systemoperatøren og han har mulighet til å endre på felter. Det må testes om de nye verdiene er gyldige før de kan legges inn i databasen.

Use case: O3 - Slette bruker





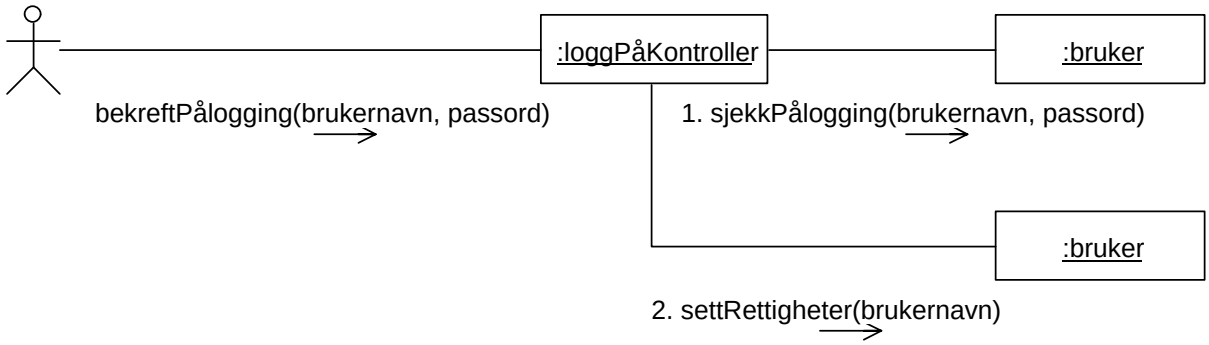
Man skal kunne slette en bruker fra databasen ved å oppgi brukerens ID, dersom brukeren finnes vil dataene om brukeren presenteres for systemoperatøren før de eventuelt blir slettet.

3.3.2.4.1 Algoritmisk modell

Denne modulen skal utvikles ved å benytte BMP.

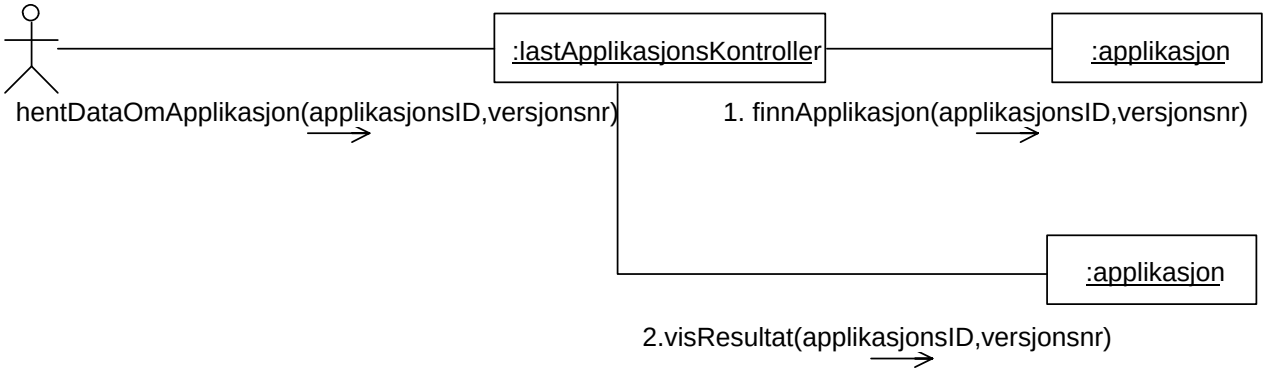
3.3.2.5 System operasjoner

Use case: B4, O4 - Logge på



Bruker av systemet taster inn brukernavn og passord. Det sjekkes mot databasen hva slags bruker dette er (vanlig bruker eller systemoperatør) og om brukernavn/passord stemmer. Utfra dette blir tilgangsrettighetene satt. Bruker blir deretter logget på systemet.

Use case: S1 - Klargjøre for lasting av applikasjoner



Ved klargjøring for lasting av applikasjoner må det hentes ut data om applikasjonen og disse må presenteres.

3.3.2.5.1 Algoritmisk modell

Denne modulen skal utvikles ved å benytte SFSB.

3.3.2.6 Restriksjoner/Begrensninger

I CMP er det containeren som tar seg av kommunikasjonen med databasen. Den genererer også automatisk SQL-uttrykkene som er nødvendige i denne forbindelse. Dette setter klart begrensinger fordi man ikke har mulighet til å skrive mer komplekse SQL-uttrykk. Dersom det er nødvendig å aksessere flere tabeller i databasen kan dette skje ved at en bean kaller en annen bean.

I BMP må brukeren kode SQL-uttrykkene. Man har dermed en større mulighet for å utføre avanserte søk mot databasen.

3.3.3 Filformat

Systemet skal kunne håndtere input fra fil. Dette gjelder for kort, kortinnehaver og applikasjon. Disse filene er semikolonseparerte med en record per linje. Antall felter vil variere utfra hvilken modul filen gjelder for. Det antas at filene er på riktig format som angitt nedenfor.

3.3.3.1 Kort

Filformatet er på følgende form:
KortID;Personnr;Status;Ustedelsedato;Utløpsdato;Type;

3.3.3.2 Kortinnehaver

Filformatet er på følgende form:
Personnr;Fornavn;Etternavn;Adresse;Postnr;Poststed;Telefonnr;Email;

3.3.3.3 Applikasjon

Filformatet er på følgende form:
ApplikasjonsID;Versjonsnr;Link;Beskrivelse;

3.4 BRUKERGRENSESNITT

Alle modulenes grafiske brukergrensesnitt er utarbeidet i JBuilder 4. Vi har lagt vekt på at fargene i skjermbildene skal være behagelige å arbeide med og at skriften skal være tydelig. Alle menyvalgene skal være enkle å følge og brukeren skal lett kunne navigere seg rundt.

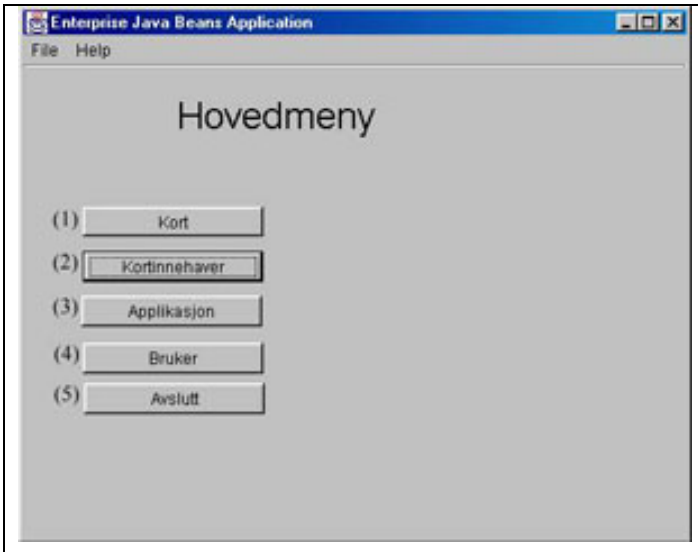
3.4.1 Real use case

Real use cases viser hvordan use casene fra kravspesifikasjonen vil bli realisert. Denne metoden er nyttig for å få en god og fullstendig oversikt over brukergrensesnittet til systemet.

I dette kapittelet ligger alle modulene med de tilhørende skjermbildene beskrevet etter hverandre. Derfor er ikke skjermbildene sortert etter use case nummereringen.

3.4.1.1 Hovedmeny

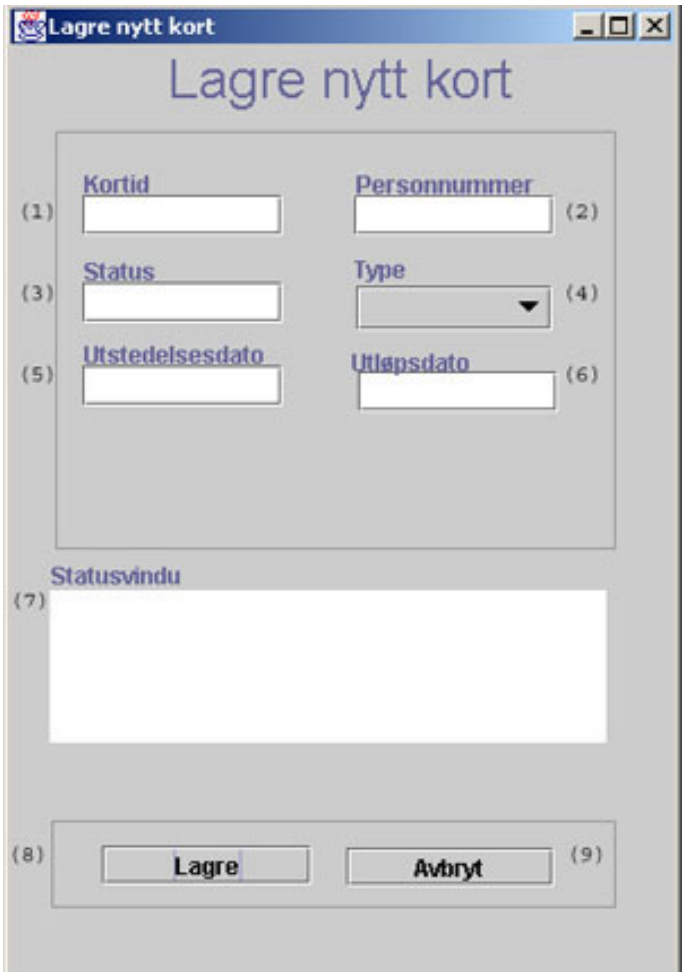
Hovedmeny: Valg for å gå til ønsket modul.



(1) Administrere kort (detaljer i 3.4.1.2 Kort).
(2) Administrere kortinnehaver (detaljer i 3.4.1.3 Kortinnehaver).
(3) Administrere applikasjon (detaljer i 3.4.1.5 Applikasjon).
(4) Administrere bruker (detaljer i 3.4.1.4 Bruker)
(5) Logge av systemet og avslutte sesjonen.

3.4.1.2 Kort

Lagre nytt kort: Use case B5 Legg inn kortinnehaver/kort manuelt



(1) Angir kortets ID

(2) Angir hvilken kortinnehaver kortet tilhører.

(3) Angir hvilken tilstand kortet er i (i bruk, sperret osv)

(4) Angi korttype (testkort eller aktivt kort)

(5) Dato da kortet ble utstedt.

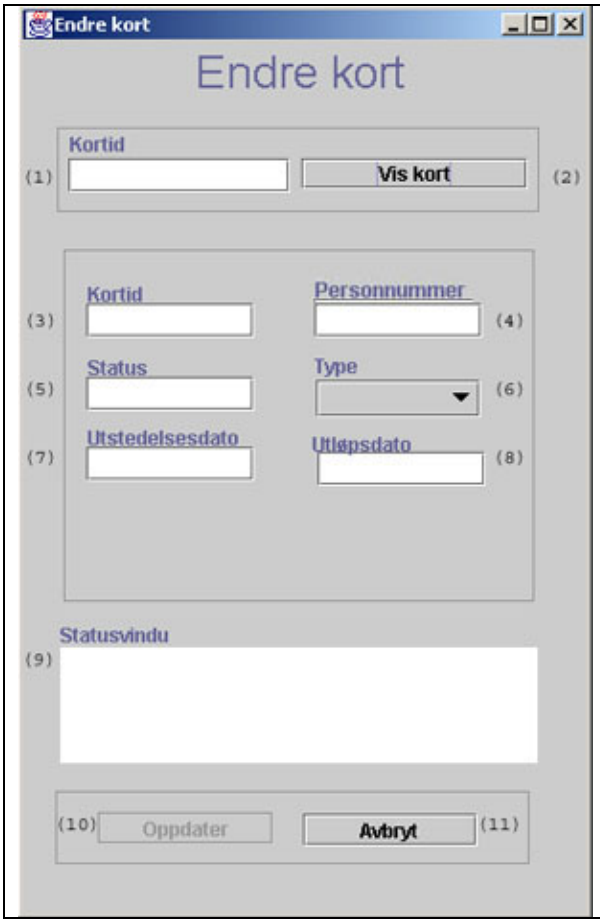
(6) Datoen kortet utløper og må fornyes.

(7) Meldinger til bruker om ulike hendelser.

(8) Lagrer kortet i databasen.

(9) Lukker vinduet.

Endre kort: Use case B8 Endre kort



(1) Angir kortets ID

(2) Henter kortet fra databasen.

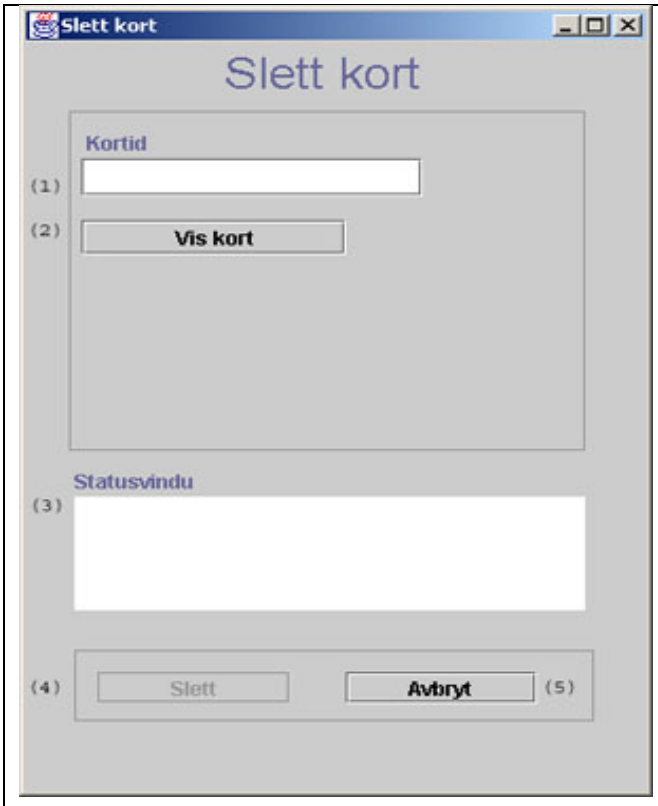
(3) – (8) Felt som kan endres på. Samme felter som i lagre kort.

(9) Informerer bruker om hendelser.

(10) Lagrer endringer i databasen.

(11) Lukker vinduet.

Slett kort: Use case B9 Slette kort



(1) Angir kortets id

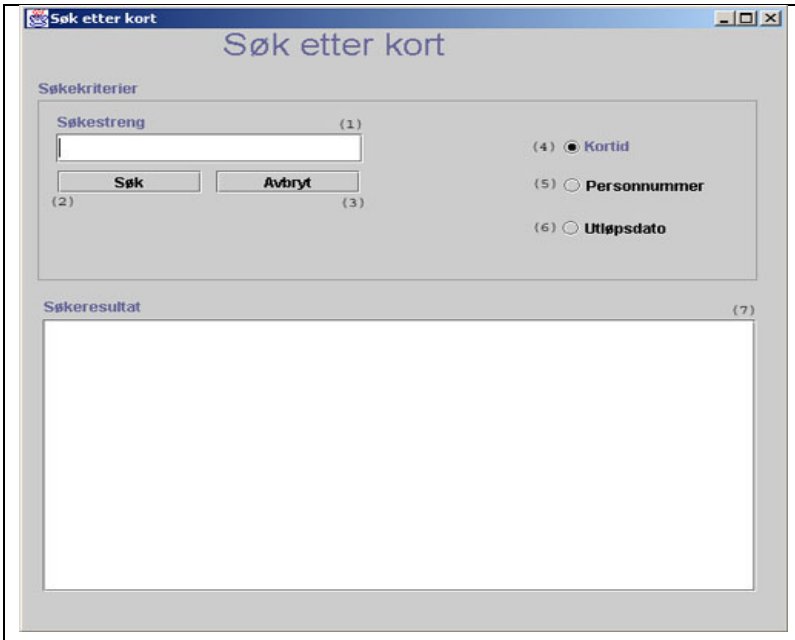
(2) Henter angitt kort fra databasen.

(3) Informerer bruker om hendelser.

(4) Sletter det valgte kortet.

(5) Lukker vinduet.

Søk etter kort: Use case B7 Hente data om kort



(1) Angir søketekst

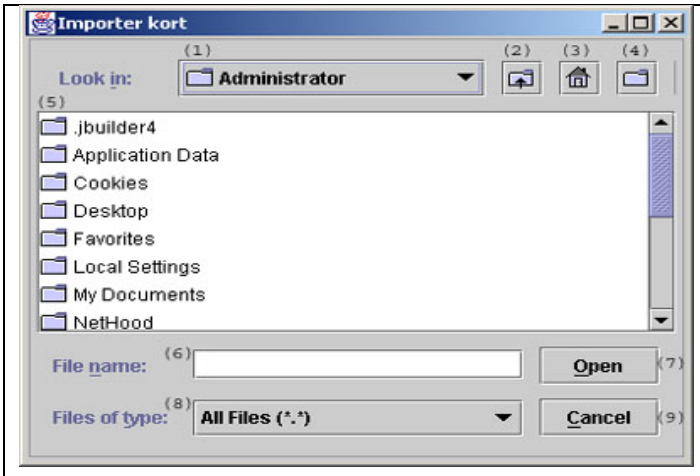
(2) Søker gjennom databasen.

(3) Lukker vinduet.

(4) - (6) Spesifiserer søket

(7) Viser søkeresultat.

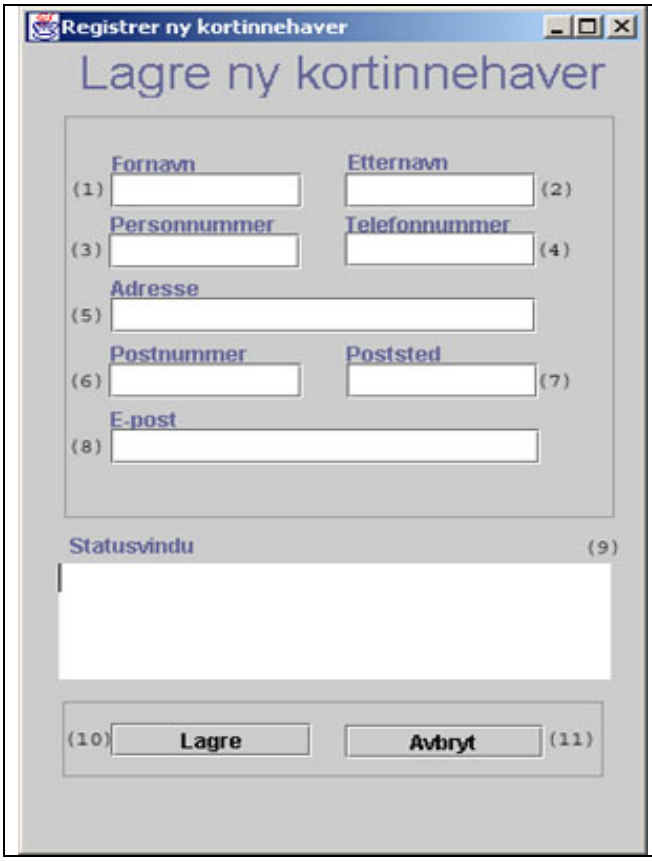
Importere kort: Use case B6 Legg inn kortinnehaver/kort fra fil



(1) Viser filstruktur.
(2) Går en katalog opp i filstrukturen.
(3) Går til hjemmekatalogen.
(4) Lager en ny katalog.
(5) Viser kataloginnhold.
(6) Viser valgt fil.
(7) Åpner valgt fil.
(8) Filtrerer filtyper.
(9) Lukker dialogboksen.

3.4.1.3 Kortinnehaver

Lagre ny kortinnehaver: Use case B5 Legg inn kortinnehaver/kort manuelt



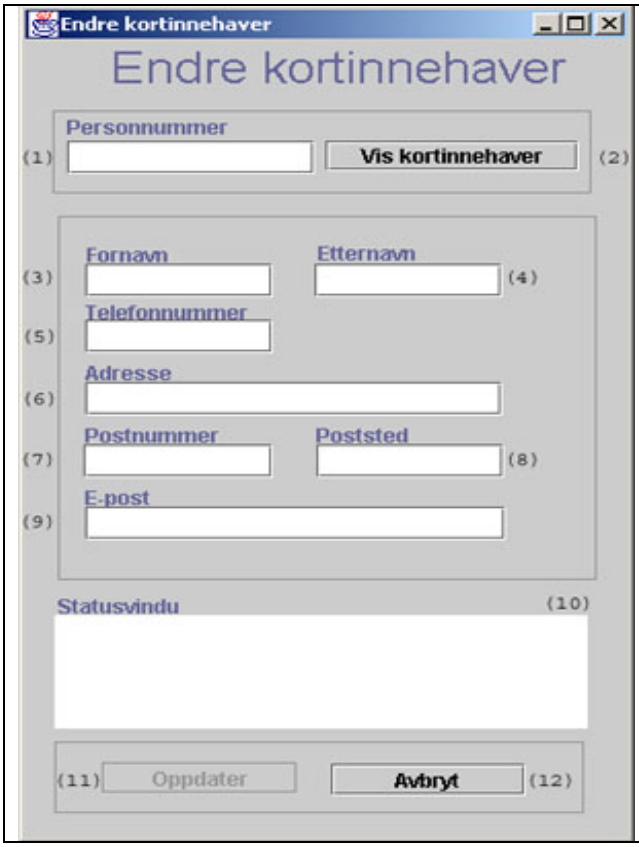
(1) - (8) Kortinnehaverinformasjon

(9) Informerer bruker om hendelser.

(10) Lagrer ny kortinnehaver i databasen.

(11) Lukker vinduet.

Endre kortinnehaver: Use case B1 Endre kortinnehaver



(1) Angi kortinnehavers personnummer som skal endres.

(2) Henter angitt kortinnehaver fra databasen og viser denne i felt (3) - (9).

(3) - (9) Kortinnehaverinformasjon.

(10) Informerer bruker om hendelser.

(11) Oppdaterer kortinnehaver i databasen.

(12) Lukker vinduet.

Slett kortinnehaver: Use case B2 Slette kortinnehaver

(1) Angi personnummeret til kortinnehaveren som skal slettes.

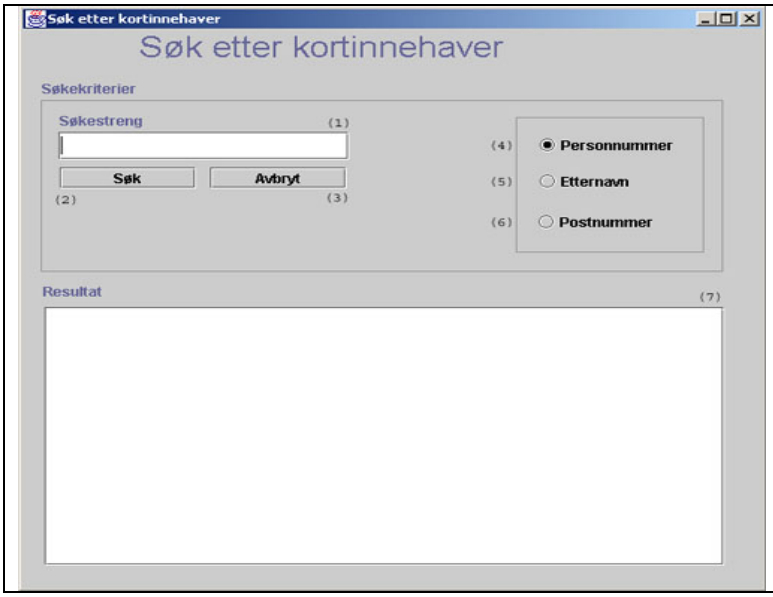
(2) Henter kortinnehaver fra databasen og displayer denne i (3)

(3) Viser kortinnehaver som skal slettes.

(4) Sletter kortinnehaver fra databasen.

(5) Lukker vinduet.

Søk etter kortinnehaver: Use case B3 Hent data om kortinnehaver



(1) Angir søketekst

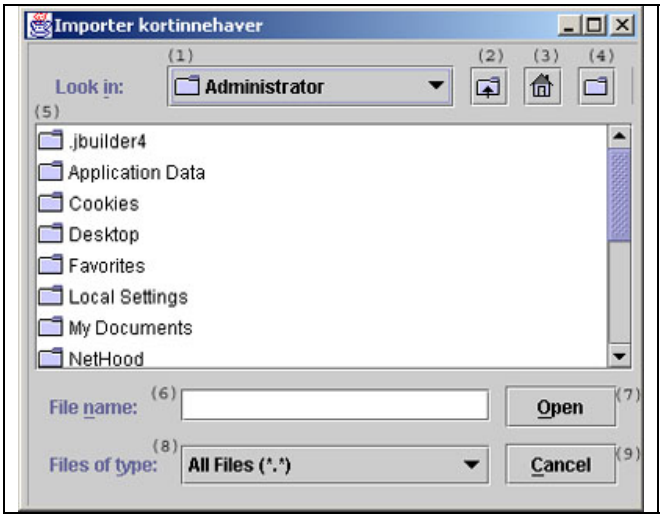
(2) Søker gjennom databasen.

(3) Lukker vinduet.

(4) - (6) Spesifiserer søket

(7) Viser søkeresultat.

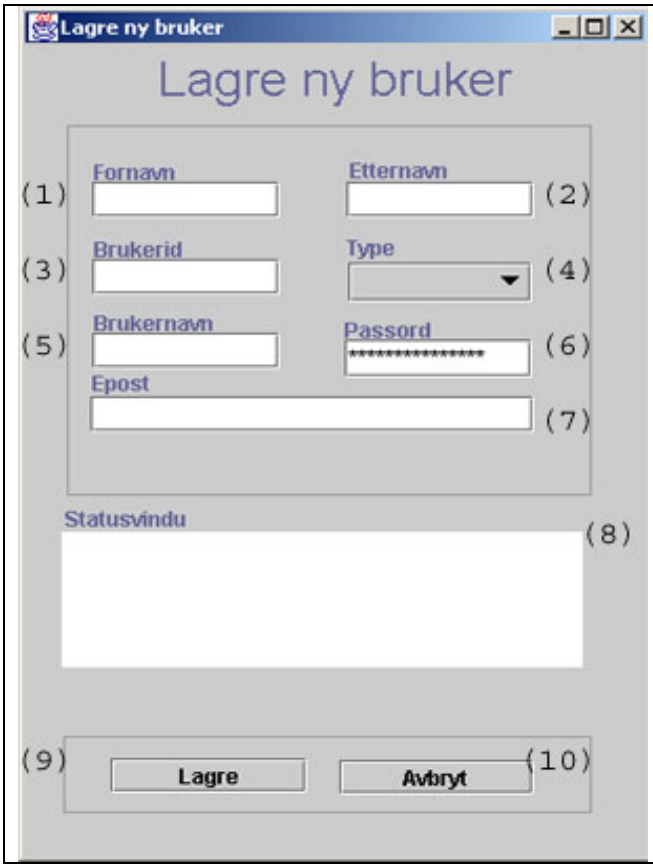
Importere kortinnehaver: Use case B6 Legg inn kortinnehaver/kort fra fil



(1) Dropdownbox som viser filstruktur
(2) Går en katalog opp i filstrukturen.
(3) Går til hjemmekatalogen.
(4) Lager en ny katalog.
(5) Viser kataloginnhold.
(6) Viser valgt fil.
(7) Åpner valgt fil.
(8) Dropdownbox som filtrerer filtyper
(9) Lukker dialogboksen.

3.4.1.4 Bruker

Lagre ny bruker: Use case O1 Legg inn ny bruker



(1) - (3) Brukerinformasjon

(4) Dropdownbox som angir om bruker er operatør eller vanlig bruker.

(5) Angir brukernavnet til den nye brukeren.

(6) Passordet til ny bruker.

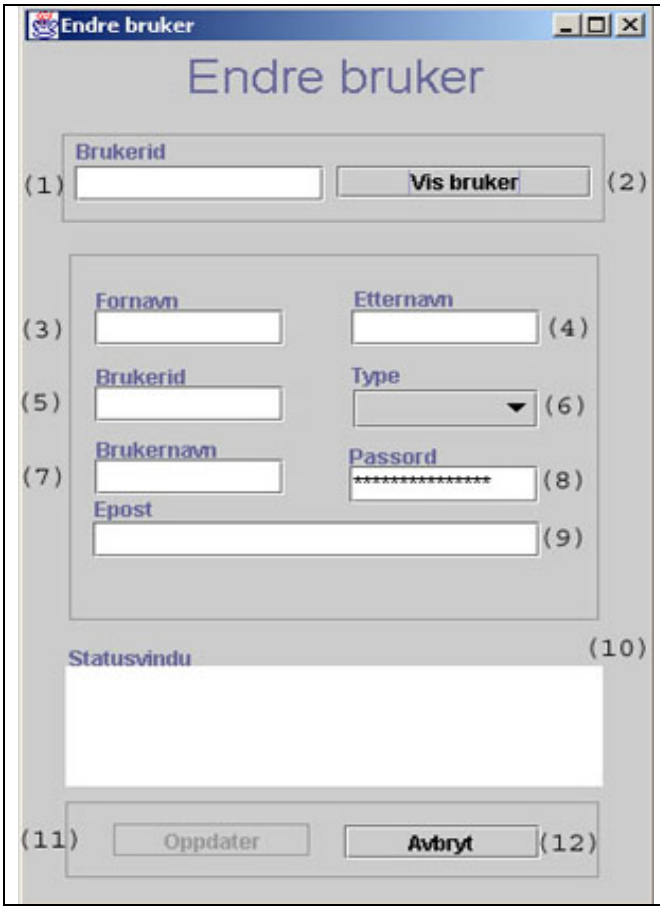
(7) Epost adresse til ny bruker.

(8) Informerer systemoperatør om hendelser.

(9) Lagrer ny kortinnehaver i databasen.

(10) Lukker vinduet.

Endre bruker: Use case O2 Endre bruker



(1) Felt som angir brukers ID

(2) Henter bruker fra databasen.

(3) – (9) Felter som kan endres på. Samme felter som i lagre bruker.

(10) Informerer bruker om hendelser.

(11) Lagrer endringer i databasen.

(12) Lukker vinduet.

Slett bruker: Use case O3 Slette bruker

(1) Felt som angir brukerid til bruker som skal slettes.

(2) Henter bruker fra databasen og displayer denne i (3).

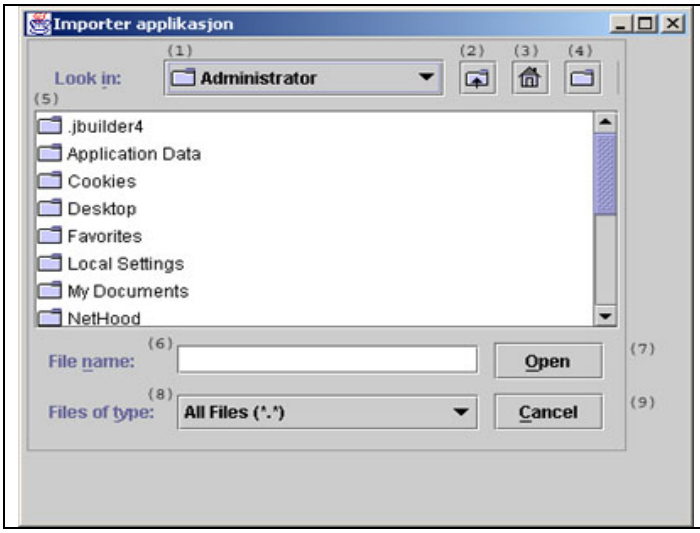
(3) Viser bruker som skal slettes.

(4) Sletter bruker fra databasen.

(5) Lukker vinduet.

3.4.1.5 Applikasjon

Importere applikasjon: Use case B12 Legg inn applikasjon



(1) Dropdownbox som viser filstruktur.
(2) Går en katalog opp i filstrukturen.
(3) Går til hjemmekatalogen.
(4) Lager en ny katalog.
(5) Vindu som viser kataloginnhold.
(6) Viser valgt fil.
(7) Åpner valgt fil.
(8) Dropdownbox som filtrerer filtyper.
(9) Lukker dialogboksen.

Slett applikasjon: Use case B10 Slette applikasjon

(1) Felt som angir applikasjon som skal slettes.

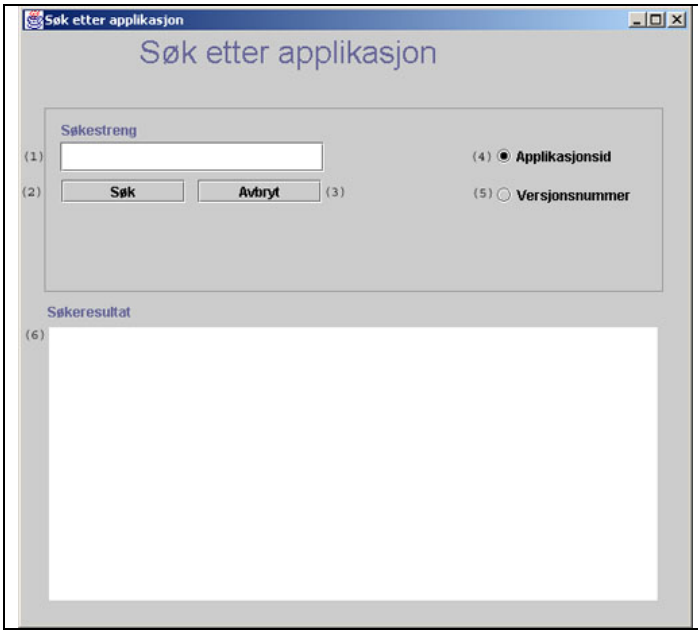
(2) Henter applikasjon fra databasen og displayer denne i (3).

(3) Viser bruker som skal slettes.

(4) Sletter bruker fra databasen.

(5) Lukker vinduet.

Søk etter applikasjon: Use case B11 Hent data om applikasjon



(1) Felt som angir søketekst

(2) Søker gjennom databasen.

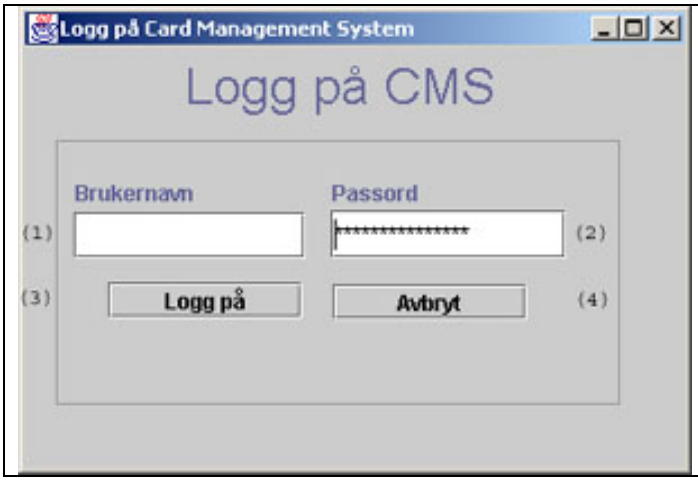
(3) Lukker vinduet.

(4) - (5) Spesifiserer søket

(6) Viser søkeresultat.

3.4.1.5 Påloggingsmodul

Logg på CMS: Use case B4.O4 Logge på



(1) Felt som angir brukernavnet til vedkommende som ønsker å logge på CMS.
(2) Passordfelt som bruker må angi for å logge på CMS.
(3) Sjekker inntastet brukernavn og passord mot databasen.
(4) Lukker vinduet.

3.4.2 Standarder

Alle vinduene i applikasjonen CMS skal ha lik størrelse på knapper, tekstbokser, passordbokser, etiketter og overskrifter så langt det lar seg gjøre. Skriften på knappene skal være "Dialog" 12 pkt, svart farge. Overskrifter og etiketter skal ha lik størrelse, farge og type. For overskrifter skal vi benytte "Dialog" 28 pkt, lilla skriftfarge. For etiketter skal vi benytte "Dialog", 12 pkt, lilla skriftfarge. Panel skal ha rammestil "titled". Tekstbokser og passordbokser skal ha lik høyde. Statuslinje med forklarende hjelpetekst skal finnes nederst i alle vinduer.

4.0 IMPLEMENTERING

4.1 INNLEDNING

Dokumentet skal gi en beskrivelse av utviklingsmiljøet og ulike verktøy som er anvendt. Det beskrives også hvilke prinsipper og standarder som er brukt i arbeidet med kodingen samt hvordan systemet er implementert. Eksempler fra kildekode ligger vedlagt (se kapittel 4.9).

4.2 VERKTØY

Under arbeidet med utviklingen av det grafiske brukergrensesnittet for de ulike modulene, har vi benyttet oss av Jbuilder. Dette er svært gunstig med tanke på at Jbuilder automatisk generer kode for de komponentene som blir opprettet og tatt i bruk. Man bruker ganske enkelt dra og slipp prinsippet og HDSHDF (Hva Du Ser er Hva Du Får) prinsippet. Under arbeidet med EJB objektene/klassene har vi brukt Ultraedit fordi dette programmet gjenkjenner (highlighter) Javakode. Programmet er dessuten brukervennlig når man har oppe mange moduler/filer samtidig. Ellers er dette programmet raskt og enkelt å bruke i forhold til Jbuilder som krever mye av maskinen det kjører på. Databasen som applikasjonen er utviklet i er en Oracle database.

4.3 UTVIKLINGSMILJØ

I arbeidet med implementeringen har vi benyttet oss av grupperommet vi fikk tildelt etter avtale med ErgoGroup. Der har vi hatt fra en til to datamaskiner til rådighet. Disse maskinene har hatt det meste av den nødvendige software vi trengte.

4.4 STANDARDER

Under utviklingen av de ulike modulene har vi i den grad det er mulig innarbeidet standardiseringsrutiner for koden. Dette er først og fremst for å gjøre koden lettere å lese med tanke på at det er andre som skal sette seg inn i koden og for å hindre feil. For å gjøre koden så intuitiv som mulig har vi blant annet benyttet oss av selvforklarende navn på funksjoner og variable. Innrykk er benyttet for å markere hvilke funksjoner og løkker som hører sammen.

4.5 EJB OG JBOSS

For å kunne benytte seg av EJB trenger man en applikasjonsserver som har mulighet til å kjøre EJB objektene. Jboss er en av få open source EJB applikasjonsservere som kan lastes ned gratis fra http://jboss.org. Jboss krever lite maskinressurser og tar liten plass på harddisken. Andre slike servere som finnes på markedet er tunge å kjøre og svært dyre å kjøpe.

Ulempen med Jboss er at det ikke har vært så mye dokumentasjon å finne, men dette ser nå ut til å bli bedre.

Et EJB objekt består av mange filer. Minimum er tre klasser (remote interface, homeinterface og bean implementasjonen) og en deployment descriptor. For å gjøre det enklere blir alle disse filene pakket til en fil på formatet jar. Denne filen blir så lagt inn under en spesiell katalog under Jboss kalt deploy. For å gjøre et EJB objekt aktivt på serveren må denne jar filen plasseres under deploy katalogen. Dette kalles å "deploye" et EJB objekt. Jboss har en funksjon som kalles autodetect. Med det menes at serveren oppdager når et nytt EJB objekt blir plassert inn under deploy katalogen.

Remote interface klassen definerer metoder som er tilgjengelige utenfra. Dette vil som regel være egendefinerte metoder. Denne klassen heter alltid det samme som navnet på EJB-objektet med .Java som endelse.

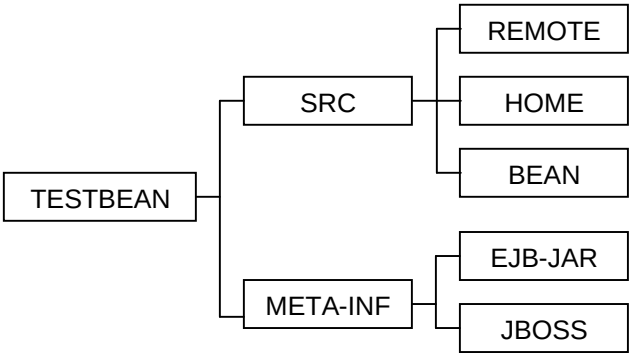
Home interface klassen spesifiserer på hvilken måte man oppretter et nytt EJB objekt som implementerer remote interface. Her definerer man create og finder metoder. Denne klassen har samme navn som EJB objektet med Home.java som endelse.

Bean klassen implementerer metodene som er definert i remote og home interface. Disse metodene brukes av containeren. Denne klassen har samme navn som EJB objektet med Bean.java som endelse.

Deployment descriptoren forteller hvilke klasser som står for bean implementasjonen, home interface og remote interface. Man kan også her angi hvilke felter i EJB objektet som skal mappes til hvilke felter i databasen ved CMP.

Deployment descriptoren må alltid kalles ejb-jar.xml. Det er også mulig å overstyre databaseinnstillinger i en fil som må hete jboss.xml. Man kan her også angi Java Naming Directory Interface. Begge disse filene må ligge i en katalog med navnet META-INF.

Eksempel på katalogstruktur for å opprette et EJB objekt:



Figur 4.1

Etter at EJB objektet er deployed på serveren kan det opprettes en klient som kan benytte objektets metoder.

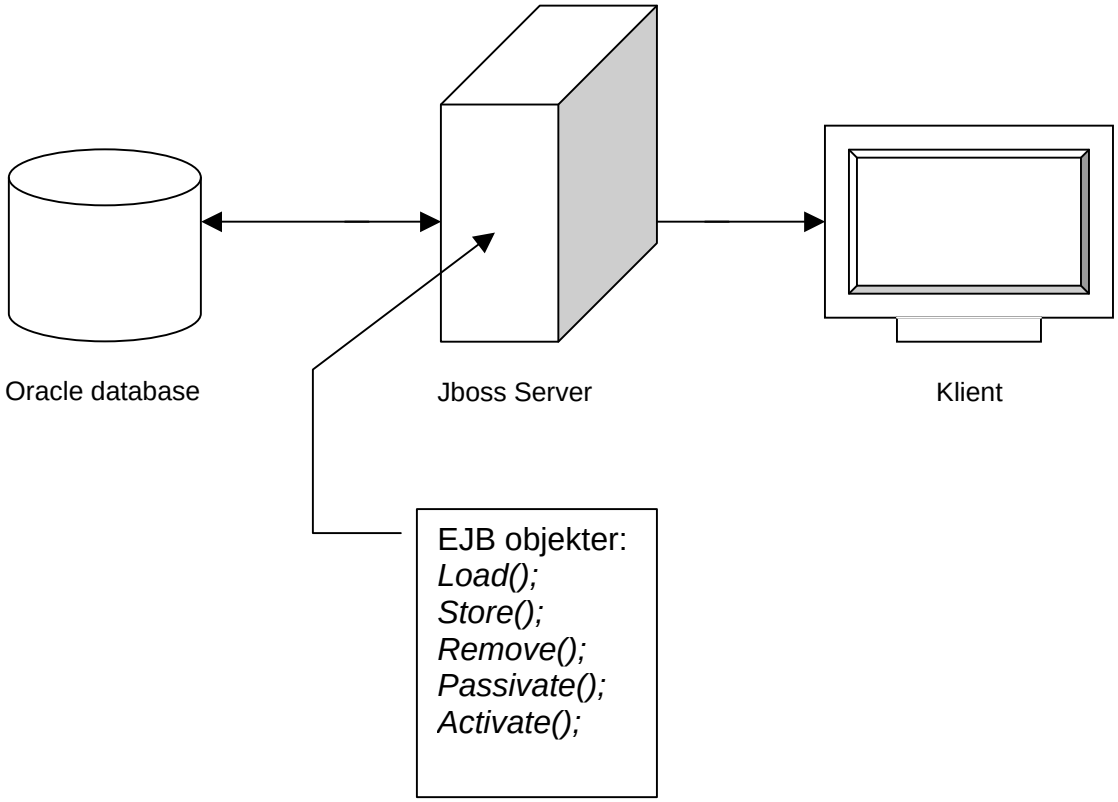
Jboss og klient på forskjellige maskiner
Hvis klienten skal kjøres på en egen maskin der Jboss serveren går på en annen maskin, må **localhost** endres til IP adressen til serveren.

```
System.setProperty("java.naming.factory.initial",
                   "org.jnp.interfaces.NamingContextFactory");
System.setProperty("java.naming.provider.url", "localhost:1099");
```

I tillegg til dette trenger klienten noen filer som ligger under Jboss katalogen. Disse filene er lib\ext\ejb.jar og client\jboss-client.jar.

4.6 SYSTEMSTRUKTUR

Skissen under viser hvordan systemet med server og database er bygd opp.



Figur 4.2

4.7 KODESTRUKTUR

Her beskrives det hvordan systemet er bygd opp. Hvilke filer som inngår, funksjonaliteten til disse samt katalogstruktur. Alle katalogene ligger under rotkatalogen EJB.

4.7.1 Kort

Kort modulen inneholder følgende filer:

Katalogen Kort

KortLagre.java er filen som skal sørge for å lagre nødvendig informasjon om et kort. Filen har et grafisk brukergrensesnitt der den tar imot inntastet informasjon fra bruker av systemet. Denne filen kaller *create*() i KortBean (EJB objekt), som siden sørger for å lagre dette i databasen ved å kalle *ejbStore*(). KortLagre.java representerer use case B5.

KortEndre.java er filen som skal sørge for å endre et kort som allerede er registrert. Filen har et grafisk brukergrensesnitt der den tar imot en kortID. Denne kortID'en blir så hentet fra databasen sammen med all informasjon om kortet. For å hente inn et spesifikt kort kalles *findByPrimaryKey*(kortid) av denne filen. For å laste informasjonen inn i et EJB objekt kalles *ejbLoad*() prosedyren til objektet. Informasjonen som kommer opp kan så endres. Ved endring oppdateres alle felter i EJB objektet og objektet kaller *ejbStore*() som oppdaterer databasen. KortEndre.java representerer use case B8.

KortSlett.java er filen som sletter et angitt kort fra databasen. Filen har et grafisk brukergrensesnitt hvor den tar imot en kortID fra bruker av systemet. Denne filen kaller *findByPrimaryKey*(kortID) for å finne riktig objekt (kort). EJB objektet kaller så *ejbRemove*(), Kortet er da slettet. KortSlett.java representerer use case B9.

KortSoek.java er filen som skal søke etter kort. Man kan søke på ulike kriterier. Ved søk kaller denne filen ulike find-prosedyrer i EJB objektet. KortSoek.java representerer use case B7

KortApplikasjon.java er filen som gjør det mulig å registrere en gitt applikasjon på et gitt kort. Filen har et grafisk brukergrensesnitt. Den søker opp alle tilgjengelige kortID'er og viser disse i en dropdownbox. Så søkes alle tilgjengelige applikasjoner opp ved å kalle *findAll*() prosedyren for Applikasjon (Applikasjon EJB objekt). Applikasjonene blir også representert ved hjelp av en dropdownbox. Ved registrering av en applikasjon på et kort blir *create*() prosedyren for Applin (Applin EJB objekt) kjørt. *Create*() prosedyren for Applin sørger for å registrere kortID og applikasjonID i en egen tabell (applikasjonslinjetabell) som kobler korttabellen sammen med applikasjontabellen. KortApplikasjon.java representerer ingen use case siden dette er en utvidelse av oppgaven.

KortSlettApplin.java er filen som gjør det mulig å slette en gitt applikasjon på et gitt kort. Filen har et grafisk brukergrensesnitt. Den søker opp alle tilgjengelige kortID'er

og viser disse i en dropdownbox. Alle applikasjoner på tilhørende kort søkes opp og vises i en også dropdownbox. Applikasjon på kort slettes ved at applin.*remove*() kalles. KortSlettApplin.java representerer ingen use case siden dette er en utvidelse av oppgaven.

KortImporter.java er filen som skal gjøre det mulig å registrere kort fortløpende fra en fil. KortImporter.java har et grafisk brukergrensesnitt i form av en åpne-fil-dialogboks. Filen er representerer ved use case B6

Katalogen Kort/SRC

Kort.java er remote interface til Kort EJB objektet. Her finnes rutinene *objectContent*() som returnerer innholdet i EJB objektet ved hjelp av en string array, en *update*(String personnr, String status, String utstedelsedato, String utlopsdato, String type) som oppdaterer EJB objektets innhold og til slutt en boolean *child*(String kortid) som sjekker om kortet har registrerte applikasjoner og returnerer en boolean (false eller true). Denne filen har ikke noen grafisk brukergrensesnitt.

KortHome.java er home interface som definerer alle finder og create metodene. Følgende metoder er definert. *create*(String kortID, String personnr, String status, String utstedelsedato, String utlopsdato, String type), *findByPrimaryKey* (String kortID), *findByPersonnr*(String personnr), *findByUtlopsdato*(String utlopsdato), *findByStatus*(String status), *findByUtstedelsedato*(String utstedelsedato), *findAll()*. Den første oppretter objektet og returnerer dette. Den andre finner et spesifikt objekt og returnerer dette. De fem siste returnerer en enumeration som er en samling av alle objektene den fant. Denne filen har ikke noen grafisk brukergrensesnitt.

KortBean.java er hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se vedlegg D. Denne filen har ikke noen grafisk brukergrensesnitt.

Katalogen Kort/META-INF

Ejb-jar.xml er filen som forteller at Kort.java er remote interface, KortHome.java er homeinterface og KortBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.2 Kortinnehaver

Kortinnehavermodulen inneholder følgende filer:

Katalogen Kortinnehaver

KortinnehaverLagre.java er filen som skal sørge for å lagre nødvendig informasjon om en kortinnehaver. Filen har et grafisk brukergrensesnitt der den tar imot inntastet informasjon fra bruker av systemet. Denne filen kaller *create*() i KortBean (EJB objekt), som siden sørger for å lagre dette i databasen ved å kalle *ejbStore*(). KortinnehaverLagre.java representerer use case B5.

KortinnnehaverEndre.java er filen som skal sørge for å endre en allerede registrert kortinnehaver. Filen har et grafisk brukergrensesnitt der den tar imot et personnummer. Dette personnummeret (kortinnehaveren) blir så hentet fra databasen sammen med resten av informasjonen om kortinnehaveren. For å hente inn en spesifikk kortinnehaver kalles *findByPrimaryKey*(personnummer) av denne filen. For å laste informasjonen inn i et EJB objekt kalles *ejbLoad*() prosedyren til objektet. Informasjonen som kommer opp kan så endres. Ved endring oppdateres alle felter i EJB objektet og objektet kaller *ejbStore*() som oppdaterer databasen. KortinnehaverEndre.java representerer use case B1.

KortinnehaverSlett.java er filen som sletter en gitt kortinnehaver fra databasen. Filen har et grafisk brukergrensesnitt hvor den tar imot et personnummer fra bruker av systemet. Denne filen kaller så *findByPrimaryKey*(personnummer) for å finne riktig objekt (kortinnehaver). EJB objektet kaller så *ejbRemove*(), Kortinnehaveren er da slettet. KortinnehaverSlett.java representerer use case B2.

KortinnehaverSoek.java er filen som skal søke etter kortinnehavere. Man kan søke på ulike kriterier. Ved søk kaller denne filen ulike find-prosedyrer i EJB objektet. KortinnehaverSoek.java representerer use case B3

KortinnehaverImporter.java er filen som skal gjøre det mulig å registrere kortinnehavere fortløpende fra en fil. KortinnehaverImporter.java har et grafisk brukergrensesnitt i form av en åpne-fil-dialogboks. Filen er representerer ved use case B6

Katalogen Kortinnehaver/SRC

Kortinnehaver.java er remote interface til Kortinnehaver EJB objektet. Her finnes rutinene *objectContent*() som returnerer innholdet i EJB objektet ved hjelp av en string array og en *update*(String fornavn, String etternavn, String postnr, String poststed, String adresse, String tlf, String epost) som oppdaterer EJB objektets innhold. Denne filen har ikke noen grafisk brukergrensesnitt.

KortinnehaverHome.java er home interface som definerer alle finder og create metodene. Følgende metoder er definert. *create*(String personnr, String fornavn, String etternavn, String adresse, String postnr, String poststed, String telefonnr, String email), *findByPrimaryKey*(String pk), *findByEtternavn*(String etternavn),

findByPostnr(String postnr). Den første oppretter objektet og returnerer dette. Den andre finner et spesifikt objekt og returnerer dette. De to siste returnerer en enumeration som er en samling av alle objektene den finner. Denne filen har ikke noen grafisk brukergrensesnitt.

KortinnehaverBean.java hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se Vedlegg D. Denne filen har ikke noen grafisk brukergrensesnitt.

Katalogen Kortinnehaver/META-INF

Ejb-jar.xml er filen som forteller at Kortinnehaver.java er remote interface, KortinnehaverHome.java er homeinterface og KortinnehaverBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.3 Bruker

Brukermodulen inneholder følgende filer:

Katalogen Bruker

BrukerLagre.java er filen som skal sørge for å lagre nødvendig informasjon om en bruker av systemet. Filen har et grafisk brukergrensesnitt der den tar imot inntastet informasjon fra en administrator av systemet. Denne filen kaller *create*() i BrukerBean (EJB objekt) som siden sørger for å lagre dette i databasen ved å kalle *ejbStore*(). BrukerLagre.java representerer use case O1.

BrukerEndre.java er filen som skal sørge for å endre en allerede registrert bruker. Filen har et grafisk brukergrensesnitt der den tar imot et brukernavn. Dette brukernavnet (brukeren) blir så hentet fra databasen sammen med resten av informasjonen om brukeren. For å hente inn en spesifikk bruker kalles *findByPrimaryKey*(brukernavn) av denne filen. For å laste informasjonen inn i et EJB objekt kalles *ejbLoad*() prosedyren til objektet. Informasjonen som kommer opp kan så endres. Ved endring oppdateres alle felter i EJB objektet og objektet kaller *ejbStore*() som oppdaterer databasen. BrukerEndre.java representerer use case O2.

BrukerSlett.java er filen som sletter en gitt bruker fra databasen. Filen har et grafisk brukergrensesnitt hvor den tar imot et brukernavn fra administratoren av systemet. Denne filen kaller så *findByPrimaryKey*(brukernavn) for å finne riktig objekt (bruker). EJB objektet kaller så *ejbRemove*(), Brukeren er da slettet. BrukerSlett.java representerer use case O3.

BrukerSoek.java er filen som skal søke etter brukere. Man kan søke på ulike kriterier. Ved søk kaller denne filen ulike find-prosedyrer i EJB objektet. BrukerSoek.java er ikke representert ved noe use case siden denne delmodulen er en utvidelse av oppgaven.

Katalogen Bruker/SRC

Bruker.java er remote interface til Bruker EJB objektet. Her finnes rutinene *objectContent*() som returnerer innholdet i EJB objektet ved hjelp av en string array og en *update*(String fornavn, String etternavn, String sysop, String passord, String epost) som oppdaterer EJB objektets innhold. Denne filen har ikke noen grafisk brukergrensesnitt.

BrukerHome.java er home interface som definerer alle finder og create metodene. Følgende metoder er definert *create*(String brukernavn, String fornavn, String etternavn, String sysop, String passord, String email), *findByPrimaryKey*(String pk), *findByEtternavn*(String etternavn), *findByFornavn*(String fornavn). Den første oppretter objektet og returnerer dette. Den andre finner et spesifikt objekt og returnerer dette. De to siste returnerer en enumeration som er en samling av alle objektene den fant. Denne filen har ikke noen grafisk brukergrensesnitt.

BrukerBean,java er hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se Vedlegg D. Denne filen har ikke noen grafisk brukergrensesnitt.

Katalogen Bruker/META-INF
Ejb-jar.xml er filen som forteller at Bruker.java er remote interface, BrukerHome.java er homeinterface og BrukerBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.4 Applikasjon

Applikasjonmodulen inneholder følgende filer:

Katalogen Applikasjon

ApplikasjonSlett.java er filen som sletter en gitt applikasjon fra databasen. Filen har et grafisk brukergrensesnitt hvor den tar imot en applikasjonid fra brukeren av systemet. Denne filen kaller så *findByPrimaryKey*(applikasjonid) for å finne riktig objekt (applikasjon). EJB objektet kaller så *ejbRemove*(), Applikasjonen er da slettet. ApplikasjonSlett.java representerer use case B10.

ApplikasjonSoek.java er filen som skal søke etter applikasjoner. Man kan ikke søke på ulike kriterier. Ved eksekvering av denne filen vil *findAll*() prosedyren i EJB objektet liste opp alle applikasjonene som finnes på systemet. ApplikasjonSoek.java er representert ved use case B11.

ApplikasjonImporter.java er filen som skal gjøre det mulig å registrere applikasjoner fortløpende fra en fil. ApplikasjonImporter.java har et grafisk brukergrensesnitt i form av en åpne-fil-dialogboks. Filen er representerer ved use case B12.

Katalogen Applikasjon /SRC

Applikasjon.java er remote interface til Applikasjon EJB objektet. Her finnes rutinene *objectContent*() som returnerer innholdet i EJB objektet ved hjelp av en string array og *child*(String applikasjonID) som sjekker om applikasjonen er registrert på noen kort. Den sistnevnte funksjonen returnerer en boolean (false eller true). Denne filen har ikke noen grafisk brukergrensesnitt.

ApplikasjonHome.java er home interface som definerer alle finder og create metodene. Følgende metoder er definert: *create*(String applikasjonID, String versjonnr, String link, String beskrivelse), *findByPrimaryKey*(String applikasjonID), f*indAll*(). Den første oppretter objektet og returnerer dette. Den andre finner et spesifikt objekt og returnerer dette. Den siste returnerer en enumeration som er en samling av alle objektene den fant. Denne filen har ikke noen grafisk brukergrensesnitt.

ApplikasjonBean.java er hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se Vedlegg D. Denne filen har ikke noen grafisk brukergrensesnitt.

Katalogen Applikasjon /META-INF

Ejb-jar.xml er filen som forteller at Applikasjon.java er remote interface, ApplikasjonHome.java er homeinterface og ApplikasjonBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.5 Applin

Applinmodulen inneholder følgende filer:

Katalogen Applin

Katalogen Applin inneholder ingen filer med grafisk brukergrensesnitt.

Katalogen Applin /SRC

Applin.java er remote interface til Applin EJB objektet (Applikasjonslinjetabellen, se kapittel 3.2). Her finnes rutinen *objectContent*() som returnerer innholdet i EJB

objektet ved hjelp av en string array. Denne filen har ikke noen grafisk brukergrensesnitt.

ApplinHome.java er home interface som definerer alle finder og create metodene. Følgende metoder er definert: *create*(String relasjonnr, String applikasjonid, String kortid), *findByPrimaryKey*(String relasjonnr), *findByApplikasjonid*(String applikasjonid), *findByKortid*(String kortid), *findAll*(). Den første oppretter objektet og returnerer dette. Den andre finner et spesifikt objekt og returnerer dette. De tre siste ligner på metode to med unntak av søkekriteriet. Denne filen har ikke noen grafisk brukergrensesnitt.

ApplinBean.java er hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se Vedlegg D. Denne filen har ikke noen grafisk brukergrensesnitt.

Katalogen Applin /META-INF

Ejb-jar.xml er filen som forteller at Applin.java er remote interface, ApplinHome.java er homeinterface og ApplinBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.6 Paalogging

Paaloggingmodulen inneholder følgende filer:

Katalogen Paalogging

Paaloggingsmodul.java er filen som skal sørge for å hente påloggingsinformasjon fra vedkommende som skal logge på systemet. Filen har et grafisk brukergrensesnitt der den tar imot brukernavn og passord. Denne filen kaller *create*() i PaaloggingBean (EJB objekt) som siden sørger for å kalle metoden *sjekkPaalogging*(brukernavn, passord) som sjekker brukernavn og passord mot databasen. Paaloggingsmodul.java representerer use case B4, O4.

Katalogen Paalogging /SRC

Paalogging.java er remote interface til Paalogging EJB objektet. Her finnes rutinen *sjekkPaalogging*(String brukernavn, String passord) som returnerer rettighetene til vedkommende som logger på (bruker eller systemoperatør). Denne filen har ikke noen grafisk brukergrensesnitt.

PaaloggingHome.java er home interface som definerer alle finder og create metodene. Følgende metode er definert: *create*(). Denne oppretter objektet og returnerer dette. Denne filen har ikke noe grafisk brukergrensesnitt.

PaaloggingBean.java er hovedklassen til EJB objektet. Det er her alle prosedyrene som er definert i home og remote interface blir implementert. For nærmere informasjon se Vedlegg D. Denne filen har ikke noe grafisk brukergrensesnitt.

Katalogen Paalogging /META-INF

Ejb-jar.xml er filen som forteller at Paalogging.java er remote interface, PaaloggingHome.java er homeinterface og PaaloggingBean.java er EJB objektets hovedklasse. Denne filen har ikke noen grafisk brukergrensesnitt.

Jboss.xml er filen som brukes for å overstyre default konfigurasjon. Her blir JNDI satt.

4.7.7 Meny

Kortinnehavermodulen inneholder følgende filer:

Katalogen Meny

Meny.java er filen som har ansvaret for hovedmenyen. Brukere av systemet kan fra denne menyen aksessere alle delmoduler av systemet.

AboutBox.java er filen som viser hvem som har laget applikasjonen.

4.8 KOMPILERING

For å kompilere alle filene benyttes standard javakompilering. I tillegg trenger applikasjonen noen biblioteker (blant annet ejb.jar og jboss-client.jar) som følger med Jboss. For å gjøre kompileringen av de ulike modulene og alle EJB objektene enklere har vi laget bat filer. Man slipper da å skrive lange kommandoer på kommandolinja hver gang noen skal kompileres. Disse filene ligger under hver modulkatalog, og kompilerer hver sin del av hver modul(eksempel KortSoek.java). En slik bat fil (compl.bat) kan se slik ut for å kompilere KortLagre.java:

javac -classpath ..\..\lib\ext\ejb.jar;..\..\client\jboss-client.jar;. KortLagre.java

Tilsvarende for andre deler av modulen og for de resterende modulene, med unntak av filnavnet til slutt.

For å kompilere selve EJB objektet (eksempelvis Kort), det vil si KortHome.java, Kort.java og Kortbean.java kan man gjøre dette på følgende måte:

javac -classpath ..\..\..\lib\ext\ejb.jar;..\..\..\lib\jdbc2_0-stdext.jar *.java

Disse kompilerte filene legges så I en egen .jar fil (eksempelvis Kort.jar) sammen med descriptorfilene som ligger under META-INF katalogen. En bat fil for å lage en jar fil kan se slik ut:

jar cvf Kort.jar src\Kort.class src\ Kort Home.class src\ Kort Bean.class META-INF\ejb-jar.xml META-INF\jboss.xml

Etter at jar filen er laget, kopierer man denne filen (eksempelvis Kort.jar) til deploy katalogen under Jboss. Jar filen blir da deployet på serveren og er klar til bruk.

For at alle modulene skal fungere må classpath settes lik hver enkelt modul som inngår i systemet. Det vil si kort, kortinnehaver, bruker, applin og applikasjon.

4.9 KODEEKSEMPEL

Vi har valgt å legge ved deler av koden til modulen bruker (CMP) og applikasjon (BMP). Descriptor filen tilhører brukermodulen.

4.9.1 Eksempel på en CMP Entity Bean

```
package src;
import javax.ejb.*;
import javax.naming.*;
import javax.sql.*;
import java.rmi.RemoteException;
import java.sql.*;
import java.util.*;

/*
Denne klassen inneholder implementeringen av metodene spesifisert
i home og remote interface for 'Bruker' EJB
*/

public class BrukerBean implements EntityBean   {

      transient private EntityContext ctx;

      public String fornavn;
      public String etternavn;
      public String sysop;
      public String brukernavn;
      public String passord;
      public String email;

      /*
      Constructor
      */

      public BrukerBean()     {System.out.println("[Nytt brukerObjekt
                               opprettet på serveren]");}

      /*
      Oppretter en instans av Bruker.Returnere null fordi den virkelige
      opprettelsen skjer i EJB containeren.
      */

      public String ejbCreate(String brukernavn, String fornavn, String
                              etternavn, String sysop, String passord,
                              String email) throws CreateException,
                              RemoteException   {

            this.fornavn            = fornavn;
            this.etternavn          = etternavn;
            this.sysop              = sysop;
            this.brukernavn         = brukernavn;
            this.passord            = passord;
            this.email              = email;

            return null;
      }

      /*
```

```
      Oppdaterer objektets variable. ejbStore() kjøres av containeren.
      */

      public void Update(String fornavn, String etternavn, String sysop,
                         String passord, String epost) {

            this.fornavn            = fornavn;
            this.etternavn          = etternavn;
            this.sysop              = sysop;
            this.passord            = passord;
            this.email              = epost;
      }

      /*
      Returnerer en array med objektets variable
      */

      public String[] objectContent()     {

            String objectCont[];
            objectCont    = new String[6];

            objectCont[0] = brukernavn;
            objectCont[1] = fornavn;
            objectCont[2] = etternavn;
            objectCont[3] = sysop;
            objectCont[4] = passord;
            objectCont[5] = email;

            return objectCont;
      }

      /*
      Kalles når objektet har blitt initialisert
      */

      public void ejbPostCreate(String brukernavn, String fornavn, String
                                etternavn, String sysop, String passord,
                                String email) throws RemoteException { }
      /*
      Metoder som benyttes av EJB containeren
      */

      public void setEntityContext(EntityContext ctx) throws
                                     RemoteException   { this.ctx = ctx; }

      public void unsetEntityContext() throws RemoteException {
                  this.ctx = null; }

      public void ejbActivate() throws RemoteException { }

      public void ejbPassivate() throws RemoteException {
                  brukernavn  = null; }

      public void ejbLoad() throws RemoteException { }

      public void ejbStore() throws RemoteException { }

      public void ejbRemove() throws RemoteException { }
}
```

4.9.2 Eksempel på en BMP Entity Bean

```
package src;
import javax.ejb.*;
import javax.naming.*;
import javax.sql.*;
import java.rmi.RemoteException;
import java.sql.*;
import java.util.*;

/*
Denne klassen inneholder implementeringen av metodene spesifisert
i home og remote interface for 'Applikasjon' EJB
*/

public class ApplikasjonBean implements EntityBean    {

      Connection con                = null;
      DataSource ds                 = null;
      InitialContext ic             = null;

      protected EntityContext ctx   = null;
      private String dbName         = "java:OracleDB";

      private String applikasjonID;
      private String versjonnr;
      private String link;
      private String beskrivelse;

      /*
      Constructor
      */

      public ApplikasjonBean()      {System.out.println("[nytt
                        ApplikasjonsObjekt opprettet på serveren]");}

      /*
      Oppretter en instans av Applikasjon. Returnerer objektet som er
      opprettet.
      */

      public String ejbCreate(String applikasjonID, String versjonnr,
                              String link, String beskrivelse) throws
                              CreateException, RemoteException    {
            try   {
                  makeConnection();

                  String insertStatement = "INSERT INTO APPLIKASJON VALUES
                                           ( ? , ?, ?, ?)";
                  PreparedStatement ps =
                                    con.prepareStatement(insertStatement);

                  ps.setString(1, applikasjonID);
                  ps.setString(2, versjonnr);

                  ps.setString(3, link);
                  ps.setString(4, beskrivelse);


                  ps.executeUpdate();
```

```
            ps.close();
            con.close();

      }catch (Exception ex)   {
            throw new EJBException("ejbCreate: " + ex.getMessage());

      }
      this.applikasjonID      = applikasjonID;
      this.versjonnr          = versjonnr;
      this.link               = link;
      this.beskrivelse        = beskrivelse;

      return applikasjonID;
}

/*
Sjekker om applikasjonen er registrert på kort. Hvis ikke returneres
true.
*/

public boolean child(String applikasjonID) throws RemoteException {

      boolean resultat = false;

      try   {
            makeConnection();
            String selectStatement  = "SELECT * FROM
                        APPLIKASJONSLINJE WHERE APPLIKASJONID= ? ";
            PreparedStatement ps    =
                              con.prepareStatement(selectStatement);
            ps.setString(1,applikasjonID);
            ResultSet rs            = ps.executeQuery();
            resultat                = rs.next();
            ps.close();
            con.close();
      }catch      (Exception ex)    {
            System.out.println("child(): " + ex.getMessage());

      }

      if (resultat) return false;
      else  {
            return true;

      }
}

/*
Returnerer en array med objektets variable
*/
public String[] objectContent()     {

      String objectCont[];
      objectCont    = new String[4];

      objectCont[0] = applikasjonID;
      objectCont[1] = versjonnr;
      objectCont[2] = link;
      objectCont[3] = beskrivelse;

      return objectCont;
```

```
}

      /*
      findermetode som søker etter alle applikasjoner i
      applikasjontabellen.
      Returnerer alle treff som en enumeration (samling).
      */

      public Enumeration ejbFindAll() throws FinderException,
                                             RemoteException   {
            String pk;
            Vector v = new Vector();
            System.out.println("[ejbFindAll()]");

            try   {
                  makeConnection();
                  String selectStatement  = "SELECT * FROM APPLIKASJON";
                  PreparedStatement ps    =
                                    con.prepareStatement(selectStatement);
                  ResultSet rs            = ps.executeQuery();
                  while(rs.next())  {
                        pk = rs.getString("applikasjonID");
                        v.addElement(pk);
                  }
                  ps.close();
                  con.close();
            }catch      (Exception ex)    {
                  throw new EJBException("ejbFindAll(): " +
                                             ex.getMessage());
            }

            return v.elements();
      }

      /*
      findermetode som søker etter en primærnøkkel i applikasjontabellen i
      databasen. Returnerer
      funnet primærnøkkel.
      */

      public String ejbFindByPrimaryKey(String pk) throws FinderException,
                                                   RemoteException   {
            boolean resultat;

            try   {
                  makeConnection();
                  String selectStatement  = "SELECT APPLIKASJONID FROM
                                    APPLIKASJON WHERE APPLIKASJONID= ? ";
                  PreparedStatement ps    =
                                    con.prepareStatement(selectStatement);
                  ps.setString(1,pk);
                  ResultSet rs            = ps.executeQuery();
                  resultat = rs.next();
                  ps.close();
                  con.close();
            }catch      (Exception ex)    {
                  throw new EJBException("ejbFindByPrimaryKey: " +
                                             ex.getMessage());
            }

            if (resultat) return pk;
```

```
      else  {
            throw new ObjectNotFoundException("Row: "+pk+" not
                                                found");

      }
}

/*
Kalles når objektet har blitt initialisert.
*/

public void ejbPostCreate(String applikasjonID, String versjonnr,
                          String link, String beskrivelse) throws
                          RemoteException {

}

/***resten av metodene benyttes av containeren***/

/*
setter context
*/

public void setEntityContext(EntityContext ctx) throws
                                          RemoteException   {
      this.ctx = ctx;
      try {
                  System.out.println("setEntityContext()");

      } catch (Exception ex) {
            throw new EJBException("Unable to connect to database. "
                                                +ex.getMessage());
      }
}

/*
fjerner context.
*/

public void unsetEntityContext() throws RemoteException {
      System.out.println("unsetEntityContext()");
      try   {
            this.ctx = null;
      }catch (Exception ex)   {
            throw new EJBException("unsetEntityContext: " +
                                                ex.getMessage());
      }
}

/*
aktiverer et passivert ejb objekt.
*/

public void ejbActivate() throws RemoteException {
      System.out.println("ejbActivate()");
      applikasjonID     = (String) ctx.getPrimaryKey();
}

/*
passiverer et ejb objekt.
*/
```

```
public void ejbPassivate() throws RemoteException {
      System.out.println("ejbPassivate()");
      applikasjonID     = null;
}

/*
laster et ejbobjekt på serveren fra databasen.
*/

public void ejbLoad() throws RemoteException {
      System.out.println("ejbLoad()");

      try   {
            makeConnection();
            String selectStatement  = "SELECT * FROM APPLIKASJON
                                        WHERE APPLIKASJONID = ? ";

            PreparedStatement ps    =
                              con.prepareStatement(selectStatement);
            ps.setString(1, this.applikasjonID);
            ResultSet rs            = ps.executeQuery();

            if (rs.next())    {
                  this.versjonnr    = rs.getString(2);
                  this.link         = rs.getString(3);
                  this.beskrivelse  = rs.getString(4);

                  ps.close();
                  con.close();
            }
            else  {
                  ps.close();
                  con.close();
                  throw new NoSuchEntityException("Row for id " +
                        applikasjonID+ " not found in database. ");
            }
      }catch (Exception ex)   {
            throw new EJBException("ejbLoad() " +ex.getMessage());
      }
}

/*
 lagrer et ejb objekt i databasen.
 */

public void ejbStore() throws RemoteException {
      System.out.println("ejbStore()");
      try   {
            makeConnection();
            String updateStatement  = "UPDATE APPLIKASJON SET
                                       VERSJONNR=?,"+
                                      "LINK=?,BESKRIVELSE=?"+

                                      "WHERE APPLIKASJONID=?";

            PreparedStatement ps =
                              con.prepareStatement(updateStatement);
            ps.setString(1, versjonnr);
            ps.setString(2, link);
            ps.setString(3, beskrivelse);
```

```
                ps.setString(4, applikasjonID);
                int rowCount      = ps.executeUpdate();
                ps.close();
                con.close();

                if(rowCount == 0) {
                      throw new EJBException("STORING ROW FAILED");
                }
          }catch (Exception ex)   {
                throw new EJBException("ejbStore() " +ex.getMessage());
          }
    }

    /*
    fjerner en instans av kort fra databasen.
    */

    public void ejbRemove() throws RemoteException {
          System.out.println("ejbRemove()");
          try   {
                makeConnection();

                String deleteStatement  = "DELETE FROM APPLIKASJON WHERE
                                           APPLIKASJONID = ? ";

                PreparedStatement ps    =
                                  con.prepareStatement(deleteStatement);

                ps.setString(1, applikasjonID);
                ps.executeUpdate();
                ps.close();
                con.close();

          }catch (Exception ex)   {
                throw new EJBException("ejbRemove() " +ex.getMessage());
          }

    }

    /*
    kobler opp mot databasen.
    */

    private void makeConnection()throws NamingException, SQLException {
          ic                = new InitialContext();
          ds                = (DataSource) ic.lookup(dbName);
          con               = ds.getConnection();
    }
}
```

4.9.3 Eksempel på en descriptorfil

```
<?xml version="1.0" encoding="Cp1252"?>
  <ejb-jar>
     <description>Bruker ver1</description>
     <display-name>Bruker</display-name>
     <enterprise-beans>
       <entity>
         <description>Bruker</description>
         <ejb-name>Bruker</ejb-name>
         <home>src.BrukerHome</home>
         <remote>src.Bruker</remote>
         <ejb-class>src.BrukerBean</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>False</reentrant>
         <cmp-field><field-name>fornavn</field-name></cmp-field>
         <cmp-field><field-name>etternavn</field-name></cmp-field>
         <cmp-field><field-name>sysop</field-name></cmp-field>
         <cmp-field><field-name>brukernavn</field-name></cmp-field>
         <cmp-field><field-name>email</field-name></cmp-field>
         <cmp-field><field-name>passord</field-name></cmp-field>
         <primkey-field>brukernavn</primkey-field>
         <transaction-type>Container</transaction-type>
         <resource-ref>
            <res-ref-name>OracleDB</res-ref-name>
            <res-type>java.sql.DataSource</res-type>
            <res-auth>Container</res-auth>
         </resource-ref>
       </entity>
    </enterprise-beans>
  </ejb-jar>
```

5.0 TESTING OG KVALITETSSIKRING

5.1 INNLEDNING

Dette kapittelet viser hvordan testingen og kvalitetssikringen av applikasjonen er ivaretatt. Det beskrives hvordan testingen og kvalitetssikringen er utført og resultatet av dette. Vi valgt å ta utgangspunkt i V-modellen. Modellen går ut på å teste hver enhet for seg selv. Deretter en integrasjonstest der modulene testes sammen. Til slutt foretar vi en systemtest der vi ser på systemet som en helhet og sammenlikner opp mot kravspesifikasjonen. Denne formen for testing gjør det enklere å oppdage, feilsøke og utbedre feil.

5.2 ENHETSTEST AV MODULER

5.2.1 Pålogging

Her vil bruker eller systemoperatør taste inn brukernavn og passord. Det har blitt testet på at både brukernavn og passord sjekkes opp mot databasen. Det ble også sjekket input av feil brukernavn og/eller feil passord. Vi har i tillegg testet at bruker og systemoperatør får riktige rettigheter ved pålogging.

5.2.2 Bruker

5.2.2.1 Lagre

Det har blitt testet at brukernavn, fornavn, etternavn, passord og e-mail er fylt inn. Deretter har det blitt testet at lengden av disse feltene ikke sprenger feltene i databasen. Dette ble testet ved å legge inn for store verdier i feltene og la et felt stå tomt. Det har også blitt testet at samme bruker ikke kan legges inn dobbelt ved å angi samme primærnøkkel.

5.2.2.2 Endre

Samme testprosedyre som på lagre. Her skal imidlertid ikke primærnøkkelen kunne endres.

5.2.2.3 Søk

Det har blitt søkt på alle kriterier opp mot data som finnes i databasen. Det ble også søkt på ugyldig data for å teste at dette ikke gir resultater.

5.2.2.4 Slett

Det ble først sjekket at brukeren som skulle slettes eksisterte i databasen.
Deretter testet vi på at en bruker som ble slettet, fysisk ble slettet fra databasen.

5.2.3 Kortinnehaver

5.2.3.1 Lagre
Det har blitt testet at personnummer, fornavn, etternavn, adresse, postnummer, poststed, telefon og e-post er fylt inn. I tillegg sjekket vi på at primærnøkkelen (personnummer) var numerisk og bestod av elleve siffer. Dette ble testet med en verdi høyere og en verdi lavere samt riktig verdi. Det har også blitt testet at lengden av feltene ikke sprengte feltene i databasen ved å taste inn for store verdier i feltene. I tillegg er det testet at samme kortinnehaver ikke kan legges inn dobbelt ved å angi samme primærnøkkel.

5.2.3.2 Importer fra fil
Vi testet at riktige verdier ble lagt inn riktig i databasen. Filen antas å være på riktig format. Det ble også testet at samme kortinnehaver ikke kunne legges inn dobbelt ved å angi samme primærnøkkel.

5.2.3.3 Endre
Samme testprosedyre ble utført som på lagre. Her kan imidlertid ikke primærnøkkelen endres.

5.2.3.4 Søk
Det ble søkt på alle kriterier opp mot data som eksisterte i databasen. Det har også blitt søkt på ugyldig data for å teste at dette ikke gir resultater.

5.2.3.5 Slett
Sjekket at kortinnehaver fantes og ble slettet fysisk fra databasen.

5.2.4 Applikasjon

5.2.4.1 Importer fra fil
Testet at riktige verdier ble lagt riktig inn i databasen. Filen antas å være på riktig format. Det har også blitt testet at samme applikasjon ikke kunne legges inn dobbelt ved å angi samme primærnøkkel.

5.2.4.2 Søk
Testet at alle applikasjoner ble hentet ut fra databasen.

5.2.4.3 Slett
Sjekket at applikasjonen fantes og ble slettet fra databasen.

5.2.5 Kort

5.2.5.1 Lagre
Det ble testet at kortid, personnummer, utstedelsesdato og utløpsdato. I tillegg ble det sjekket at primærnøkkelen (kortid) var numerisk og bestod av elleve siffer. Dette ble testet med en verdi høyere og en verdi lavere samt riktig verdi. Det ble også sjekket at lengden av feltene ikke sprengte feltene i databasen ved at det ble tastet inn for store verdier i feltene. I tillegg ble det testet at samme kort ikke kunne legges inn dobbelt ved å angi samme primærnøkkel.

5.2.5.2 Importer fra fil
Det ble testet at riktige verdier ble lagt riktig inn i databasen. Filen antas å være på riktig format. Vi har også sjekket at samme kort ikke kunne legges inn dobbelt ved å angi samme primærnøkkel.

5.2.5.3 Endre
Samme testprosedyre som på lagre. Her kan imidlertid ikke primærnøkkelen kunne endres.

5.2.5.4 Søk
Søkte på alle kriterier opp mot data som eksisterte i databasen. Det ble også søkt på ugyldig data for å teste at dette ikke ga resultater.

5.2.5.5 Slett
Sjekket at kort fantes og ble slettet fysisk fra databasen.

5.2.5.6 Legg inn applikasjon på kort
Sjekket at alle kort ble hentet ut fra databasen. Sjekket at applikasjoner ble registrert på riktige kort i databasen. Testet også at det ikke var mulig å legge inn samme applikasjon to ganger på samme kort.

5.2.5.7 Slett applikasjon fra kort

Sjekket at alle kort ble hentet ut fra databasen med tilhørende applikasjoner. Det ble også testet at tilhørende applikasjoner fysisk ble slettet i databasen. Sjekket også at det ikke var mulig å slette applikasjoner som ikke fantes på kortet.

5.3 INTEGRASJONSTEST

Her testet vi grensesnittet mellom moduler som er avhengig av hverandre.

5.3.1 Kortinnehaver

Slett kortinnehaver er avhengig av kort. Det ble testet at vi ikke klarte å slette en kortinnehaver som hadde kort registrert på seg.

5.3.2 Applikasjon

Slett applikasjon er avhengig av kort. Det ble testet at det ikke var mulig å slette en applikasjon som var registrert på et eller flere kort.

5.3.3 Kort

Det ble testet at det ikke var mulig å legge inn et nytt kort med en kortinnehaver som ikke eksisterte.

Det var ikke mulig å få registrert applikasjon på kort hvis det ikke fantes noen applikasjoner eller kort i databasen.

Det ble sjekket at det ikke var mulig å slette kort som var registrert med applikasjoner. Alle applikasjoner måtte slettes fra kortet først.

5.4 SYSTEMTEST

Hele systemet ble testet som en helhet. Kravspesifikasjonen var grunnlaget for denne testingen. Sjekket at de krav som var fastsatt i kravspesifikasjonen var implementert i løsningen. Eventuelle avvik kommenteres.

5.5 CMP vs BMP

En viktig del av testingen var å undersøke ytelsen på de forskjellige bean typene. CMP ble testet opp mot BMP. Vi hadde ingen verktøy til rådighet for å teste disse bean typene, dermed ble metoden vi brukte enkel og primitiv. Vi tok for oss en modul utviklet i CMP og en modul utviklet i BMP. Siden importer fra fil gjør det enkelt å legge inn flere rekorder valgte vi å kjøre testene i *KortinnehaverImporter*() og *KortImporte*r(). Begge disse metodene har en create metode der en ny kortinnehaver eller et nytt kort blir opprettet i databasen. Rett før og etter dette systemkallet printet vi ut tidspunktet gitt i millisekunder. Dette ga oss en differanse i millisekunder som gjorde at vi kunne sammenlikne CMP og BMP.

Det er vanskelig å benytte denne metoden til å gjøre nøyaktige og pålitelige målinger. Mange feilkilder kan virke inn på resultatet.

Alle målinger er i millisekunder.

Målinger utført i CMP:

Før	Etter	Differanse
644	704	60
74	154	80
295	345	50
445	485	40
535	565	30
605	635	30
785	845	60
916	976	60
36	66	30
156	196	40
276	356	80
456	486	30
586	617	31
687	717	30
787	887	100
957	1007	50
107	137	30
207	237	30
318	368	50
438	458	20
548	588	40
648	688	40
768	818	50
878	908	30
988	1009	21

129	189	60

Middelverdi: **45,08**

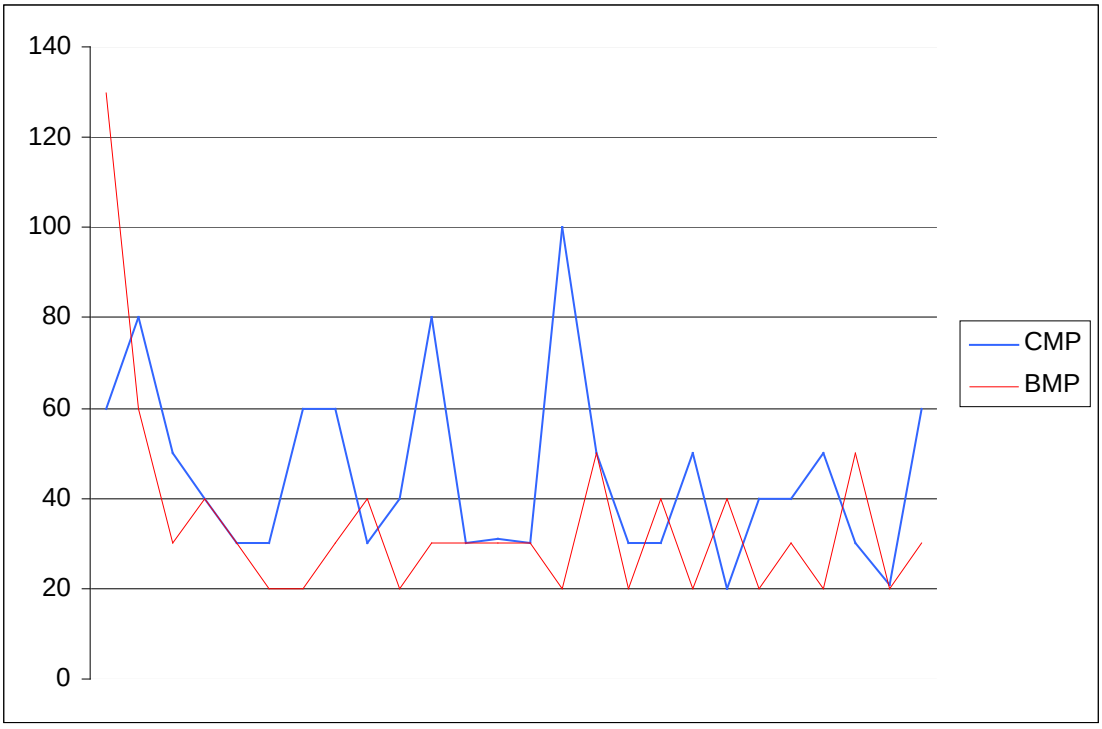
Målinger utført i BMP:

Før	Etter	Differanse
28	158	130
709	769	60
829	859	30
899	939	40
79	109	30
160	180	20
230	250	20
340	370	30
430	470	40
510	530	20
620	650	30
690	720	30
780	810	30
881	911	30
971	991	20
31	81	50
131	151	20
201	241	40
321	341	20
391	431	40
491	511	20
632	662	30
712	732	20
802	852	50
942	962	20
12	42	30

Middelverdi: **34,62**

Utfra målingene ser det ut til at BMP har bedre ytelse enn CMP. Vi mener likevel at vi ikke kan fastslå at BMP har bedre ytelse enn CMP. Dette på grunn av at målemetoden er for usikker og unøyaktig. Siden systemet vårt bare utfører enkle spørringer mot databasen, vil ikke måleverdiene utgjøre store differanser mellom CMP og BMP

Grafene på figur 5.1 viser ytelsen på CMP mot BMP.



Figur 5.1

5.6 KONKLUSJON

Vi startet med å teste alle delmoduler hver for seg først. Disse ble testet med fremgangsmåte beskrevet i kapittel 5.2. Alle delmodulene passerte alle tester. Etter dette utførte vi en integrasjonstest hvor vi testet enkeltmoduler mot hverandre som beskrevet i kapittel 5.3. Dette fungerte slik som det skulle uten noen feilmeldinger. Til slutt testet vi systemet i helhet. Dette fungerte som forventet uten noen form for feilmeldinger. Vi anser oss ferdige med all testing og forventer at systemet skal være stabilt og robust.

6.0 AVSLUTNING

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

6.1 DRØFTINGER / DISKUSJONER

Her tar vi først for oss modul for modul og ser på endringer i disse i forhold til kravspesifikasjonen. Så tar vi til slutt for oss systemet som en helhet. Vi tar ikke for oss mindre endringer som anses som redaksjonelle. Slike endringer er for et eksempel å endre navn på et felt i databasen. Det er kun forholdsvis store endringer som har innvirkning på systemets virkemåte eller teknologi som vi ser nærmere på her.

6.1.1 Pålogging

Påloggingsmodulen er utviklet i henhold til hvordan den er beskrevet i kravspesifikasjonen. De tilhørende funksjonene til modulen Pålogging er utviklet slik det er beskrevet i 6.2.2 Bruker.

Denne modellen kunne vært løst i enten CMP eller BMP. Årsaken til at vi valgte å utvikle denne i SFSB var at vi ønsket å få en så bred innsikt som mulig i EJB.

6.2.2 Bruker

Modulen Bruker skulle vært utviklet ved hjelp av BMP, men vi valgte i stedet å gjøre dette i CMP. Årsaken til dette er beskrevet i 6.2.3 Kort. I tillegg måtte vi utvide modulen litt. Det viste seg at vi ikke hadde tatt hensyn til at man måtte kunne søke opp registrerte brukere i systemet. Dette anser vi som mer brukervennlig og relevant å ha med i løsningen.

Da vi gikk i gang med utviklingen av modulen inneholdt den tilhørende tabellen i databasen et eget tallfelt som fungerte som primærnøkkel. Dette var tenkt å fungere som brukernes personlige ID i systemet. Etterhvert kom vi fram til at dette var unødvendig og at påloggingsmodulen ikke ville fungere som tiltenkt. Årsaken til dette er at systemet da tillater flere brukere å ha samme brukernavn og passord. Ettersom vi valgte å gjøre om brukernavnet til primærnøkkel i denne tabellen unngikk vi dette problemet.

I designdokumentet er det angitt at funksjonene *finnBruker*(brukerID), *sjekkBrukerVerdier*(brukerData*), endreBruker*(brukerID, brukerData), *slettBruker(*brukerID), *sjekkOmEksisterer*(brukerData) og *createBruker*(brukerData) skal ligge i egne klasser som tar seg av disse operasjonene. Da vi arbeidet med designdokumentet hadde vi imidlertid lite kunnskap om hvordan EJB arkitekturen fungerte. Etterhvert som vi kodet viste det seg at det var nødvendig å løse det slik at alle de omtalte funksjonene ligger i EJB objektet og blir kalt av klienten i stedet.

Som en følge av dette er også de tilsvarende funksjonene for de andre modulene løst på samme måte.

6.2.3 Kort

Modulen Kort skulle i utgangspunktet vært utviklet ved hjelp CMP. Dette lot seg ikke gjøre på grunn av at CMP genererer ferdige SQL-uttrykk og tar seg av databasekommunikasjon. Vi valgte derfor å løse modulen i BMP. Dette gir mulighet for å skrive mer kompliserte SQL-uttrykk. Årsaken til dette ligger i tabellen (t)Applikasjonslinje som angir hvilke applikasjoner som ligger på de ulike kortene. Denne tabellen består av to primærnøkler som også er fremmednøkler til tabellene (t)Kort og (t)Applikasjon. EJB har en del innebygde funksjoner som må implementeres. Dette gjelder i dette tilfellet spesielt *ejbFindByPrimaryKey*() som tar imot en primærnøkkel. Ettersom vi benytter to primærnøkler i denne tabellen var vi nødt til å skrive om denne funksjonen, noe som ikke lar seg gjøre i CMP.

Det viste seg imidlertid at det ikke var mulig å skrive om ejbFindByPrimaryKey funksjonen slik vi hadde tenkt ved hjelp av BMP. Det var heller ikke mulig å implementere slett applikasjon på kort med det tabelloppsettet vi hadde. Dette på grunn av at det bare er mulig å sende med en primærnøkkel ved sletting av en applikasjonslinje. Problemet var at ved sletting av et kort vet vi bare kortets id og ikke id'en(e) til tilhørende applikasjon(er). Løsningen på dette var å sette inn et tredje felt i tabellen som består av de to feltene som lå der fra før. Dette ble da primærnøkkelen i tabellen og de to andre feltene ble gjort om til å være fremmednøkler

I ettertid viser det seg at dette allikevel kunne vært utviklet i CMP fordi vi da unngår hele problemet med to primærnøkler i en tabell. Som en naturlig følge av at modulen Kort ble utviklet i BMP så endret vi de andre modulenes utviklingsteknologi slik at det fremdeles ble utviklet to moduler i CMP og to i BMP.

I designdokumentet er det også angitt at en rekke funksjoner som opererer mot modulen kort skal ligge i egne klasser. Dette har vi også valgt å løse på måten som er beskrevet i 6.2.2 Bruker.

6.2.4 Kortinnehaver

Denne modulen ble opprinnelig utviklet ved hjelp av BMP men dette ble endret til CMP. Årsaken til dette er beskrevet i 6.2.3 Kort.

Modulen er stor grad utviklet i henhold til beskrivelsen i kravspesifikasjonen. I designdokumentet er det angitt at en rekke funksjoner som opererer mot modulen kortinnehaver skal ligge i egne klasser. Dette har vi valgt å løse på samme måten som er beskrevet i 6.2.2 Bruker.

En viktig tilleggsfunksjon vi la til var at det ikke skulle være mulig å slette en kortinnehaver fra databasen dersom vedkommende hadde registrerte kort. I praksis fungerer det slik at de registrerte kortene først må slettes fra databasen. Dette blir brukeren gjort oppmerksom på.

6.2.5 Applikasjon

Denne modulen ble opprinnelig utviklet ved hjelp av BMP men dette ble endret til CMP. Årsaken til dette er beskrevet i 6.2.3 Kort.

Modulen er i stor grad utviklet slik den er beskrevet i kravspesifikasjonen. I designdokumentet er det angitt at en rekke funksjoner som opererer mot modulen applikasjon skal ligge i egne klasser. Dette har vi valgt å løse på måten som er beskrevet i 6.2.2 Bruker.

I samråd med oppdragsgiver kom vi til at det var nødvendig å kunne legge inn og slette applikasjoner på kortene (se 6.2.3). Vi fant det derfor naturlig å utvide funksjonen *slettApplikasjon*() fordi en applikasjon i databasen sannsynligvis ligger på mange av kortene som sirkulerer blant kortinnehaverne. Dette utføres det en kontroll på ved forsøk på å slette en applikasjon. Årsaken til dette er at i praksis vil det innebære at alle kortene må kalles inn og applikasjonen må slettes fra disse eller eventuelt oppdateres før den slettes fra databasen.

Under utvikling av funksjonen *applikasjonSoek*() vurderte vi en løsning der man hadde mulighet til å søke på flere kriterier. Ettersom applikasjonene ikke har noen attributter som det er spesielt hensiktsmessig å søke på valgte vi i stedet å liste ut alle applikasjonene som ligger i databasen.

6.2.6 Systemet

Systemet som en helhet tilfredstiller i stor grad kravspesifikasjonen. De endringer som er gjort på systemet er tilleggsfunksjoner som har kommet som en naturlig utvidelse av oppgaven underveis i prosjektet.

Modulen klargjøre for lasting av applikasjoner over Internett viste det seg at vi ikke hadde mulighet til få implementert i løsningen. Dette skyldes at vi var avhengig av gruppa som arbeidet med lasting av applikasjoner over Internett. For at denne modulen skulle blitt realisert måtte de vært ferdig med sin del tidligere. Denne modulen var derfor ikke høyt prioritert fra starten av.

6.2.7 Utviklingsmodell

Utviklingsmodell ble valgt i samråd med oppdragsgiver. Vi følte at det var naturlig å bruke den samme modellen som de benytter i egne prosjekter. Dermed falt valget på RUP. I ettertid har vi sett at dette muligens ikke var den beste løsningen. RUP er en meget tung og krevende modell som egner seg for større prosjekter med personer som har erfaring fra tidligere prosjekter. Gruppa kunne lite om denne modellen og hadde gjennom systemutviklingsfaget sett på andre modeller som det hadde vært mer fordelaktig å ta i bruk.

6.2.8 Hvorfor EJB?

Hvorfor skal man så benytte seg av EJB? Ved bruk av EJB servere blir kompleksiteten redusert betraktelig under komponentutvikling. Dette på grunn av at EJB automatisk støtter tjenester som dataoverføring, sikkerhet og databasetransaksjoner.

EJB serveren administrerer de underliggende transaksjonene, slik at utvikleren kan konsentrere seg om hva selve applikasjonen skal gjøre. Det vil med andre ord si business metoder. Siden EJB teknologien er basert på Java kan komponentene brukes på en hvilken som helst plattform og under et hvilket som helst operativsystem.

6.2 KRITIKK AV OPPGAVEN

Under arbeidet med prosjektoppgaven har vi gjort oss en del erfaringer, positive såvel som negative. Det er spesielt utarbeidingen av funksjonaliteten til systemet vi vil rette den største kritikken mot, men det er også andre ting som burde/kunne vært utført eller tilrettelagt annerledes.

Når det gjelder funksjonaliteten til systemet føler vi at vi under oppstartsfasen hadde liten kontroll av hva som skulle være med i de ulike modulene. Selv om vi arbeidet hardt for å få en så dekkende kravspesifikasjon som mulig viste det seg etterhvert at vi måtte legge til en del funksjonalitet og gjøre om noe av det planlagte underveis. Det er særlig i kodefasen vi har oppdaget manglene i systemet og utbedret modulene. Årsaken til dette ligger dels i at vi ikke hadde god nok oversikt over hva systemet skulle inneholde og dels i at oppdragsgiver ikke var klar nok i beskrivelsen av systemet de ønsket.

Valget av teknologi for systemet var heller ikke så heldig. Vi kunne særs lite om de ulike teknologiene vi har brukt så mye tid gikk med til å sette oss inn i disse. Spesielt Jboss bød på mye problemer fordi dokumentasjonen av denne var veldig mangelfull da vi gikk i gang med kodefasen. Dette har heldigvis bedret seg etterhvert med mange eksempler og god dekning av ulike bruksområder. På bakgrunn av dette vil vi anslå at prosjektet gikk i gang to til tre måneder for tidlig.

Databasen skulle opprinnelig opprettes i MsSQL. Etter mye prøving og feiling med den fikk vi opplyst fra Jboss at MsSQL ikke er kompatibel med systemet deres. Derfor valgte vi å opprette en Oracle-database i stedet og fikk enda en ny teknologi å sette oss inn i. Arbeidet med denne gikk bra og fikk i gang kommunikasjonen med databasen med suksess. I ettertid viser det seg at MsSQL er kompatibel med Jboss og at det igjen var dokumentasjonen av Jboss som var mangelfull.

I samråd med oppdragsgiver ble vi enige om å bruke RUP som utviklingsmodell da det er modellen de bruker under utvikling av systemer. Ulempen var at vi ikke kunne noe særlig om denne utviklingsmodellen samt at vi fikk opplæring i den på skolen samtidig som vi skulle arbeide med prosjektet. Det var derfor stor usikkerhet om

hvordan vi skulle legge opp arbeidet med prosjektet. Vi ser også at denne modellen egner seg bedre for utvikling av større systemer.

En modul oppdragsgiver satte fokus på senere i prosjektet var modulen som skulle utvikles sammen med studentprosjektet om lasting av applikasjoner over Internett. For at dette skulle ha blitt en ferdigutviklet modul måtte den andre prosjektgruppen vært ferdig med sitt prosjekt lenge før innleveringsfristen for prosjektene.

6.3 VIDERE ARBEID

Videre utvikling av prosjektet vil i første rekke dreie seg om modulen for lasting av applikasjoner over Internett. Dette vil innebære at databasen blant annet må utvides med felter for ALU og ALC, det må opprettes egne EJB objekter som håndterer disse transaksjonene og applikasjonen må utvides med flere funksjoner.

6.4 GRUPPAS ARBEID

Ettersom prosjektgruppen består av fire personer fant vi det naturlig å arbeide i to par. Dermed får vi utrettet enda mer enn når hele gruppen arbeider samlet samt at gruppen kan arbeide med ulike emner samtidig. Årsaken til at vi ikke delte gruppen i fire var at de ulike gruppemedlemmene satte seg inn i ulike teknologier og kunne derfor drive aktiv opplæring på hverandre under arbeidet. Denne typen arbeidsmetode sørger for at det oppstår færre feil ved at den enkelte passer på den andre. For at denne arbeidsformen skal være effektiv er det viktig med god kommunikasjon og hyppige oppdateringer innad i gruppen samt å ha klar oppgavefordeling. Dette har fungert veldig bra både mellom gruppemedlemmene og til oppdragsgiver.

Videre har vi lagt sterkt vekt på å ha en jevn fordeling av arbeidsmengde gjennom hele prosjektperioden. Dette føler vi at vi har lykkes med i stor grad. Eneste faktor som har spilt negativt inn her er undervisning i skolefag med eksamener før og etter påske med tilhørende skoleoppgaver som har krevd mye tid. Dette har vi kompensert med høyere arbeidsmengde på prosjektet i enkelte perioder.

Gruppemedlemmene har hatt anledning til å disponere sin egen tid, men med ansvar for at tildelte oppgaver blir utført til angitt tid. Høy selvdisiplin og godt samarbeid mellom medlemmene har sørget for at denne ordningen har fungert utmerket.

Ved valg av prosjektleder kom vi til at det beste for hvert enkelt medlem ville være at alle fikk prøve seg som leder. Dette er ikke nødvendigvis den beste ordningen for prosjektet, men vi anså at erfaringen for den enkelte ville bli desto større. Det har også fungert slik at av- og påtroppende prosjektleder har hatt ett uformelt møte for å diskutere prosjektets status ved overlappingen. Alle har fungert som leder i omtrent like lange perioder.

Under store deler av prosjektperioden har det vært arrangert møter med veileder og oppdragsgiver for å sørge for at arbeidet gikk etter planen. Etterhvert som vi ble ferdige med kravspesifikasjonen og designdokumentet følte vi ikke behov for å ha

møter med veileder, men heller disponere tiden til arbeid med koding. Under denne tiden har vi henvendt oss til veileder for hjelp til denne delen. Ved stagnering under arbeid med fagmessige problemer i systemutviklingen har vi henvendt oss til faglærer.

Under arbeidet med prosjektet hadde vi en rekke milepæler som fremgår av fremdriftsplanen (se vedlegg A1). Forprosjektet gikk etter planen og ble levert til gitt tidsfrist. Etter dette fikk vi erfare at det var satt av knapt med tid til kravspesifikasjonen og designdokumentet. I tillegg tok det lengre tid enn planlagt å sette seg inn i de ulike teknologiene som skulle benyttes under kodingen. Derfor kom kodefasen i gang senere enn planlagt. I framdriftsplanen fremgår det at hver modul skal kodes hver for seg etter hverandre, men på grunn av forskyvningen og behovet for opplæring av hverandre valgte vi å arbeide parallelt med modulene. Dette er en hovedårsak til at kodefasen likevel ble ferdig i tide. Testing av produktet og sluttstilling av rapporten gikk etter planen.

Arbeidet med kodingen har som regel foregått på dagtid i ukedagene fordi åpningstidene hos oppdragsgiver ikke gir oss anledning til å være der på kveldstid eller helligdager. For å utnytte tiden vi har disponibelt hos oppdragsgiver har vi derfor lagt mye av rapportarbeidet til kveldstid.

På slutten av prosjektet ble gruppen invitert til et møte i forbindelse med lansering av lånekort på HiG. Deltagerne på dette møtet var representanter fra oppdragsgiver, veileder, en fra gruppen samt en journalist fra ComputerWorld. Grunnen til at gruppen vår var representert var fordi vi skulle fortelle om prosjektet vårt og kontakten mellom HiG og arbeidslivet.

6.5 KONKLUSJON

Gjennom arbeidet med hovedprosjektet har vi fått verdifull erfaring i å arbeide med en større oppgave gjennom et lengre tidsrom tett på de samme personene. Vi har opplevd hvordan det er å arbeide i både medgang og motgang under perioden. Det har ikke oppstått noen større interne konflikter blant gruppemedlemmene selv om det til tider kan være tungt å arbeide i de samme omgivelsene over større perioder. Dette er ikke nødvendigvis det beste for prosjektet da økt engasjement kanskje kan føre til høyere kvalitet på produktet.

Vi har hele tiden arbeidet mot en ekstern oppdragsgiver. Gjennom dette har vi fått en viss opplevelse av hvordan det fungerer i næringslivet. Samarbeidet med oppdragsgiver har fungert bra og det har vært lett å få veiledning selv om kontaktpersonen ikke nødvendigvis har vært tilstede. I store deler av tiden har gruppen fungert helt selvstendig der også de enkelte har fått tildelt egne oppgaver. Dette vil vi forhåpentligvis få nytte av senere i arbeidslivet.

Prosjektoppgaven resulterte i et produkt som forhåpentligvis oppdragsgiver og skolen vil dra nytte av senere. For vår egen del er vi godt fornøyd med hva vi har prestert og kommet fram til. Vi føler også at modifiseringene vi har gjort underveis har bidratt til å gjøre det ferdige produktet langt bedre enn hva kravspesifikasjonen tilsier.

Gjennom arbeidet med testapplikasjonen har vi fått satt oss grundig inn i grunnprinsippene ved EJB. Denne teknologien er det stadig flere som velger å ta i bruk samtidig som mange ikke har så mye kunnskap om den. På grunnlag av dette ser vi det som en klar fordel at vi er i stand til å utvikle et system basert på denne teknologien.

7.0 LITTERATURLISTE

Faglitteratur:
Mastering Enterprise Java Beans, Wiley, Ed Roman
Applying UML and Patterns, Prentice Hall, Craig Larman
Thinking in JAVA, Prentice Hall, Bruce Eckel

Kilder fra Internett:
http://jboss.org
http://theserverside.com
http://java.sun.com
http://hig.no

Mailinglister fra Jboss.org og Theserverside.com