Boligformidling

Sluttrapport

2014

Alexander Gård, s198585, 1. år IT

Arlen Syver Wasserman, s193956, 1. år IT

Petter Soland Wold Gjerstad, s181110, 3. år Dataingeniør



Innhold

[Om oppgaven 1](#_Toc387928105)

[Rammeverk for applikasjon 2](#_Toc387928106)

[Om programmet 2](#_Toc387928107)

[Ramme 2](#_Toc387928108)

[Programstruktur 3](#_Toc387928109)

[Produktdokumentasjon 3](#_Toc387928110)

[Avvik ifra oppdragsgivers ønsker 3](#_Toc387928111)

[Datastruktur 3](#_Toc387928112)

[Systemkrav 4](#_Toc387928113)

[Java-versjon 5](#_Toc387928114)

[Prosessdokumentasjon 5](#_Toc387928115)

[Arbeidsfordeling 5](#_Toc387928116)

[Alex: 6](#_Toc387928117)

[Arlen: 6](#_Toc387928118)

[Petter: 7](#_Toc387928119)

[Utviklingsverktøy 7](#_Toc387928120)

[Prioritering 7](#_Toc387928121)

[Testrapport 8](#_Toc387928122)

# Om oppgaven[[1]](#footnote-1)

*“Dere får nå i oppdrag å lage et datasystem for et firma som driver med boligformidling for utleiemarkedet. Firmaets kunder vil være utleiere og boligsøkende. Ved hjelp av applikasjonen, der utleieboliger og boligsøkende er registrert, skal kundebehandlere formidle kontakt mellom disse to aktørene i markedet.*

*Utleiere vil få sine boligtilbud presentert og får derved hjelp til å nå potensielle leietakere. Boligsøkende registrerer på sin side sine ønsker og krav til en eventuell bolig, samt personlige opplysninger som kan ha betydning for utleieren, og får derved hjelp til å finne en egnet bolig.*

*Ved å matche ledige boliger mot de boligsøkendes beskrivelser skal firmaet informere sine kunder om aktuelle leietakere og boliger. Hvordan denne informasjonen skal gis, har ikke oppdragsgiver noen klar formening om, men håper dere kan finne en passende løsning.*

*Firmaet skal registrere alle leieforhold som opprettes, slik at boligen og leietakeren da ikke lenger vil være registrert som henholdsvis ledig og boligsøkende.*

*For at firmaet skal kunne videreutvikle sin forretningsidé og strategi, ønsker de at programmet skal lagre all historikk nyttet til leiekontrakter som har blitt inngått i firmaets regi.”*

# Produktdokumentasjon

//En **produktdokumentasjon** som beskriver hvordan systemet (programmet) er bygget opp og hvordan ting virker sammen. Dette "dokumentet" er til for den som vil sette seg inn i systemet, f.eks. for å kunne vedlikeholde det, videreutvikle det, endre det, eller evaluere det ved sensur!

//Database stuff burde være med i produktdokumentasjonen

// Dere bør derfor ha med figurer som illustrerer hvordan eventuelle klassehierarkier er bygget opp.

// Hvis dere ikke har rukket å innfri alle ønskene oppdragsgiveren har til programmet, må dere her skrive noe om hvilken funksjonalitet dere har prioritert og hvorfor, og hva som bør gjøres videre med programmet slik at andre programmerere kan fortsette der dere slapp.

// Har dere derimot lagt inn funksjonalitet utover oppdragsgiverens ønsker, bør dere også skrive noe om det.

## Om programmet

Applikasjonen vil fungere som en portal for både kunder og kundebehandlere hvor kunder som ønsker å leie ut sine leiligheter kan registrere disse, og kunder som ønsker å leie leiligheter kan klikke og søke etter disse registrerte leilighetene. Igjennom en kundebehandler kan man opprette leiekontrakter mellom utleier og boligsøker.

Programmet kan eksempelvis installeres på datamaskiner i firmaets lokaler hvor besøkende kunder kan klikke igjennom katalogen, legge ut en ny annonse, melde seg opp til varsler. En kundebehandler har også denne applikasjonen installert og kan opprette kontrakter ut fra forespørsler ifra kunder. Det er slik at man må igjennom en kundebehandler for å opprette leiekontrakter.

Da kravene i oppgaven “Boligformidling” henviser til at *“... ansatte i firmaet skal formidle kontakt mellom disse to aktørene i markedet*”, er systemet kun begrenset proporsjonalt i forhold til hvor mange kundebehandlere firmaet har.

## Ramme

Siden applikasjonen skal utvikles for et firma vil vi legge vekt på at applikasjonen skal kunne benyttes av flere personer. Utviklingen vil også skje med tanke på at applikasjonen vil bli kjørt fra flere operativsystemer. Vi har gått ut ifra at det vil bli lagret data for X antall kunder, og applikasjonen har derfor blitt utviklet med tanke på skalerbarhet slik at hvis lagring av data skal bli utført lokalt på hver pc som kjører programmer vil dette medføre at hvert program har tilgang til ulik informasjon.

Vi har valgt å løse dette problemet ved å lagre dataen til applikasjonen i en SQL database. Selv om PC-en som applikasjonen kjører på vil være avhengig av internett tilgang, mener vi at det fortsatt er mer hensiktsmessig å løse problemet ved hjelp av en database.

Applikasjonen vil bli utviklet med tanke på skalerbarhet. Vi har også gått ut fra at applikasjonen vil bli kjørt fra flere operativsystemer, og applikasjonen vil bli utviklet med tanke på dette.

## Programstruktur

Vennligst se den vedlagte Javadoc-filen for en detaljert beskrivelse av strukturen til applikasjonen.

## Stuff

Programmet bruker en MySQL database til å kunne samkjøre dataen mellom maskiner som kjører programmet, slik at eventuelle datamaskiner som kjører programmet inneholder samme informasjon, som for eksempel i et kundebehandler senter blir dette veldig praktisk.

For å gjøre program koden mest oversiktlig og justerbar har vi valgt å dele programmet i flere klasse typer. Database klassene og dens arvinger Data\_XXX inneholder SQL kommandoene som blir kjørt mot en MySQL server, og returnerer bearbeidet informasjon ifra databasen.

Hva måtte vi droppe av funksjonalitet?

Vi fikk tatt med alle de prioriterte funksjonalitetene, men det kan fortsatt bli lagt til flere funksjonaliteter. Se liste under….

Hva som bør gjøres videre med programmet, hvilke funksjonaliteter bør bli lagt til:

* Bedre GUI skalering
* Sende email hvis kunde har glemt passord
* Sende boliginformasjon til kunder på email(bør skrives om)
* Visning av kontrakter knyttet til en bolig, til utleiere.
* Threading på mulige trege internettforbindelser

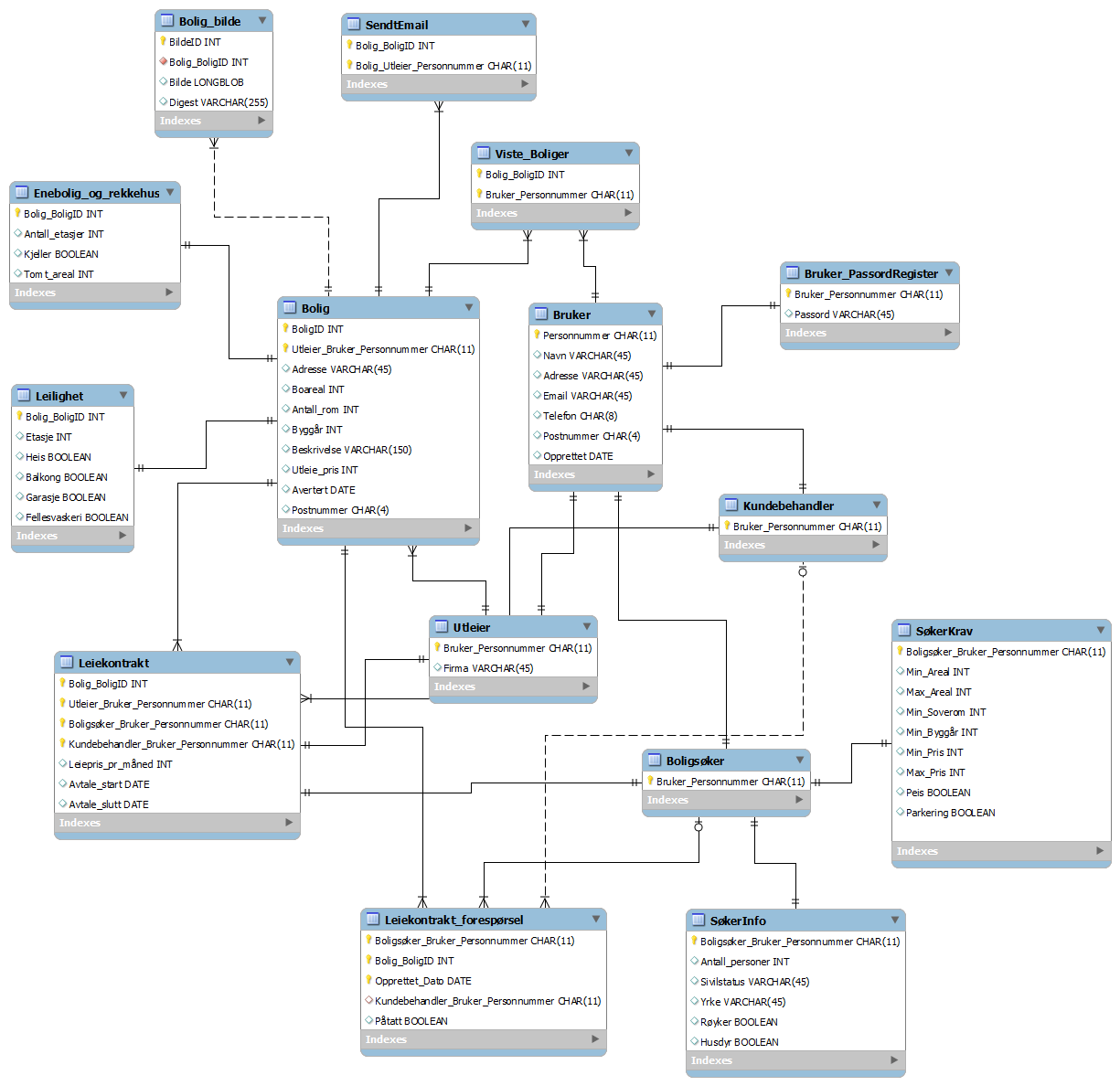
## Avvik ifra oppdragsgivers ønsker

Hva har vi gjort som ikke var påkrevd av oppdragsgiver:

* Database
* Utelatt noe info om kunder osv.

### Datastruktur

Oppdragsgiver ønsket at applikasjonen skulle lagre en intern datastruktur til fil. Vi har valgt å kommunisere med en SQL database. Dette for at applikasjonen skal kunne kjøre på flere av oppdragsgivers PCer og kunne samkjøre de. Lagringen vil skje ved at en kundebehandler kan ta sikkerhetskopi av databasen ved utskrivning av en SQL fil, samt importere databasen fra en slik fil.



Figuren ovenfor viser et EER (Enhanced Entity Relationship)-diagram over databasen hvor dataen vil bli lagret. Tabellen SendEmail er ikke tatt i bruk av applikasjonen, men hvis det skulle tenkes at applikasjonen skal utvikles videre, vil denne tabellen være nødvendig for funksjonalitet som vi ikke fikk implementert.

## Systemkrav

|  |  |  |
| --- | --- | --- |
| **Type** | **Krav** | **Anbefaling** |
| Lagringsplass | 20 kB tilgjengelig på harddisk | Kun for kundebehandler:  Lagringsplass for sikkerhetskopi vil variere etter antall kunder/boliger som er registrert. |
| RAM | 256 MB | 256 MB eller mer |
| Prosessor | Pentium 2 266 MHz prosessor | Pentium 2 266 MHz prosessor eller raskere |
| Skjermoppløsning | 1366 x 768 piksler | 1366 x 768 piksler eller høyere |
| Operativsystem | Må oppfylle kravene for installasjon av Java 7, se link for mer informasjon.  <https://www.java.com/en/download/help/sysreq.xml> | Ingen anbefalinger |
| Internett | Dial-Up | Bredbånd/Fiberoptisk kabel |

## Java-versjon

Vi har valgt å utvikle applikasjonen vår i Java 7. Sammenlignet med Java 6 og eldre versjoner, har Java 7 generelt bedre stabilitet, hastighet og sikkerhet som alle er viktige egenskaper for applikasjonen vår. En mulighet kunne også vært og bruke Java 8, men ettersom denne versjonen fortsatt er i utviklingsfasen har vi valgt å se bort fra dette alternativet. Ettersom det vil bli brukt funksjonalitet som er spesifikt for Java 7 og høyere vil ikke applikasjonen la seg kjøre på en maskin som kun har Java 6 installert.

# Prosessdokumentasjon

## Arbeidsfordeling

//burde teksten under være i produktdokumentasjon?

// Det skal gå fram av dokumentasjonen **hvem som har gjort hva** av prosjektets forskjellige deler.

Ved planleggingen og skrivingen av kravspesifikasjonen, fikk man inntrykk av at man skulle lage et mest mulig profesjonell og reel brukbar applikasjon og ikke et pseudo program basert på at man skulle vise hva man hadde lært av Java. Dette medførte at vi valgte å bruke SQL for å samkjøre kundene som skulle bruke programmet. Men selv om dette er saken, fins det mye Collection bruk.

Allerede i kravspesifikasjonen valgte vi å dele SQL databehandlingen mest mulig ifra GUI klassene. Av flere grunner:

1. Videreutvikling: En framtidig programmerer trenger ikke direkte å forholde seg til databasen og dens struktur, men heller bruker metoder og objekter til å manipulere databasen.
2. Samkjøring mellom utviklerne: For å minske dobbelt koding av SQL kommandoer er det bedre å samle like SQL kommandoer slik at et gruppemedlem ikke trenger å sette seg inn i alle database kommandoene, men heller fokusere på sine oppgaver.
3. Oversikt: Ved å se igjennom Data\_XXX klassene får man en god oversikt over hvordan databasen fungerer, og hvordan man selv skal kunne lage sine egne SQL kommandoer.

Nedenfor ligger oversikten over hva som ble programmert gjennom prosjektoppgaven.

### Alex:

Databasestruktur

Data\_Bruker

Data\_Kontrakter

Data\_Login

Data\_Statistikk

Database

GraphPanel

Leiekontrakt

LeiekontraktForespørsel

Main

MainPanel\_Kunde

MainPanel\_Kundebehandler

MainPanel\_Login

MainPanel\_NotConnected

SubPanel\_AdminTools

SubPanel\_KontraktForespørsler

SubPanel\_LogIn

SubPanel\_LogOut

SubPanel\_RegistrerBehandler

SubPanel\_RegistrerKontrakt

SubPanel\_RegistrerKunde

SubPanel\_VisKontrakter

SubPanel\_Statistikk

Table

Vindu\_Main

### Arlen:

Programskjelletstruktur

Bolig

Enebolig

Leilighet

Bruker

Data\_Boliger

MainPanel

SubPanel

SubPanel\_Boligeier

SubPanel\_Boliger

SubPanel\_RegistrerBoliger

### Petter:

Databasestruktur(Mindre endringer på tabeller)

SubPanel\_KundeProfil (sammensatt gui og logikk klasse(deriblant DBHS))

SubPanel\_Kunder (sammensatt gui og logikk klasse(deriblant DBHS))

//hvilke valg som er gjort og hvorfor det ble valgt nettopp slik, dvs prioritering

## Prioritering

Vi fikk tatt med alle de prioriterte funksjonalitetene, men det kan fortsatt bli lagt til flere funksjonaliteter. Se liste under….

Hva som bør gjøres videre med programmet, hvilke funksjonaliteter bør bli lagt til:

* Bedre GUI skalering
* Sende email hvis kunde har glemt passord
* Sende boliginformasjon til kunder på email(bør skrives om)
* Visning av kontrakter knyttet til en bolig, til utleiere.
* Threading på mulige treige internetforbindelser

# Testrapport

//En **testrapport** som viser hva slags kvalitetssikring som er gjennomført: hvilke tester som er utført, hvilke feil det er testet med hensyn på, etc.

//hvilke: testet panelene systematisk hver for seg

//hensyn: la stor hensyn på koblingen til databasen, store muligheter for feil

For å forsikre oss om at applikasjonen var mest mulig “bug fri” har vi testet vert av panelene nøye.

Det viser seg at bruk av SQL gjør det lettere å fange og forhindre feil. Forutsatt at man catch’er SQLException’er som blir kastet, og har riktige data constraints i databasen.

Under er det en liste over bugs i applikasjonen som fortsatt ikke har blitt fjernet.

Liste over bugs:

* GUI skalering
* Utvidelse av brukertype til enten boligsøker eller utleier oppdateres ikke automatisk i KundeProfils gui. Man må manuelt bruke et musetrykk i vinduet for å få oppdateringen til å skje.
* Bruken av Long fremfor int skaper trøbbel i SubPanel\_KundeProfil.visBoligVarsler kan skape trøbbel da longs ikke vil trigge NumberFormatException. Helst skal samtlige metoder i KundeProfil operere med ints. Rare lagringer vil oppstå hvis brukeren angir en desimal verdi.

1. Sitatet er hentet ut fra oppgaveteksten [↑](#footnote-ref-1)