UNIVERSITETET I AGDER

Butterfly Friends, fadderorganisasjon håndteringssystem

Bachelorrapport - DAT304, våren 2017

av
Eirik Baug

Veileder: Sigurd M. Assev

Fakultet for teknologi og realfag
Universitetet i Agder

Grimstad, mai 2017

Status: Endelig

Nøkkelord: Fadderorganisasjon, Brukervennlighet, Bildebehandling, Nettsideløsning, Administrasjon

Resymé: Oppgaven prøver å lage en nettsideløsning for en mindre fadderorganisasjon som kan håndtere de fleste aspekter av driften og håndtere kommunikasjonen med organisasjonens medlemmer. Løsningen har som mål å være brukervennlig og effektivisere arbeidsprosessen til organisasjonen, spesielt når det gjelder behandling av organisasjonens medlemmer og fadderbarn, samt bildeopplastning, behandling og fremvisning.

Løsningen har blitt utviklet ved å se på andre suksessfulle løsninger som løser problemene på en god måte. I tillegg har det blitt undersøkt en rekke relevante designprinsipper for nettsider for å gjøre nettsiden så behagelig og brukervennlig som mulig.

Løsningen viser at effektivitet og tilbakemelding på handlinger er viktig for et system som er behagelig og effektivt å bruke, samtidig bilder og nyheter står sentralt når man skal lage et engasjerende og inkluderende system for faddere i organisasjonen.

Obligatorisk egenerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	☑
2.	Jeg/vi erklærer videre at denne besvarelsen: • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☑
3.	Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	☑
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	☑
5.	Jeg/vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	☑
6.	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	☑

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).

Oppgaver som er unntatt offentlighet eller tausehetsbelagt/konfidensiell vil ikke bli publisert.

Jeg/vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	☑ Ja ☐ Nei
Er oppgaven båndlagt (konfidensiell)? (Båndleggingsavtale må fylles ut)	☐ Ja ☑ Nei
Er oppgaven unntatt offentlighet? (inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)	☐ Ja ☑ Nei

Forord

Bakgrunnen for denne rapporten er et bachelorprosjekt utført av student Eirik Baug ved Universitetet i Grimstad (UiA) våren 2017. Prosjektet er forespurt av fadderorganisasjonen Butterfly Friends i håp om å få et system som enkelt og på en effektiv måte kan håndtere en del av de daglige arbeidsoppgavene som organisasjonen foretar seg.

Butterfly Friends er liten fadderorganisasjon stasjonert i Vikersund, Buskerud, som håndterer 2500 fadderbarn og 230 lønnede ansatte i Gambia der de bygger skoler og tilbyr fri skolegang til trengende barn. [44]

Prosjektet startet 20.02.17 og ble avsluttet 21.05.17

Jeg vil gjerne takke min veileder, førstelektor Sigurd Munro Assev, for veiledning gjennom prosjektperioden.

Eirik Baug,
Mai 2017,
Grimstad

Innholdsfortegnelse

1 Innledning

Prosjektet har som formål ha utvikle et system for håndtering av en mindre fadderorganisasjon. Nettsiden er ment å håndtere både faddere og ansatte i organisasjonen og gi fadderne en nettside der de kan følge med på arbeidet til til organisasjonen or deres fadderbarn, samt gi de ansatte en plattform for å håndtere en database av tilhørende bestående av brukere, barn, bilder og artikler som tilhører organisasjonen. De ansatte vil, fra administrasjon delen av nettsiden, kunne administrere disse ressursene til organisasjonen.

1.1 Bakgrunn

Vi lever i et nettbasert samfunn i dag hvor det meste av informasjon kan nås fra datamaskinen din. Dette gjør også internettet et ekstremt kraftig middel for å nå ut til folk, noe som er viktig for alle seriøse bedrifter i dagens samfunn. Vil du være anerkjent og relevant som en bedrift eller organisasjon må du være istand til, i hvert fall i en viss grad, å

kommunisere og administrere organisasjonen over nettet, og dette gjelder ekstra mye for bedrifter som er avhengig av frivillige donasjoner og medlemmer, slik som nettopp en fadderorganisasjon.

En fadderorganisasjon bør vise hva de driver med, slik at de kan engasjere frivillige til å støtte saken og sende midler, for den beste reklamen er nettopp det arbeidet de utfører til vanlig. Arbeidet som må gjøres, for en fadderorganisasjon som ikke allerede har gjort dette, er å bygge et system som formidler deres arbeid til verden, og som engasjerer og griper dem som måtte følge med på dette. Store fadderorganisasjoner som Redd Barna er flinke på og reklamere for sitt arbeid og kan skilte med en pent bygget side med både blogginnlegg, videoer og bilder som viser besøkende hvem de er og hva de gjør, samt enkle løsninger for å hjelpe folk til å hjelpe dem.

Problemet som skal løses blir altså nettopp dette. Hvordan kan man skape en engasjerende side som lar folk følge med på arbeidet, støtte organisasjonen og gir faddere mulighet til å bli følelsesmessig investert i deres fadderbarn ved å få et innblikk i hva støtten de sender faktisk utretter. Det hele vil koke ned til et robust administrasjonssystem i bakgrunnen for å håndtere formidle nettopp dette. Effektivitet og brukervennlighet vil også stå i fokus under utviklingen av prosjektet ettersom systemet må være enkelt å bruke når det skal håndtere et såpass stor database og administrere mange funksjonaliteter. Systemet må derfor være intuitivt og innbydende å bruke for de mange ansatte som skal bruke det.

1.2 Problemdefinisjon

Det ultimate målet i dette prosjektet er å lage et system som på en engasjerende, oversiktlig og brukervennlig måte kan håndtere en mindre fadderorganisasjon og også gi en nettside der faddere kan følge med i prosessen. Noen av målene og funksjonene er etter forespørsel fra oppdragsgiver, mens andre mål er i henhold til hva jeg selv har ansett som verdifulle funksjoner for denne typen nettside. Det ble spesifikt etterspurt av oppdragsgiver et system som er betydelig mer brukervennlig enn det som er i dag. Dagens nettside er svært statisk og tilbyr lite til faddere og ansatte i organisasjonen, slik at å lage noe som er bedre enn det vil trolig ikke bli et stort problem. Derimot er jeg ikke personlig interessert i å levere et system som bare akkurat når opp til det som blir etterspurt. En del av problemet blir derfor å hvordan systemet kan bli mest mulig brukervennlig og enkelt å bruke, mens det fortsatt beholder all planlagt funksjonalitet.

Mål

Prosjektet deles opp i mindre funksjoner som alle er delmål som alle bidrar til å lage et system som oppnår målet satt for prosjektet ved å løse sine egne problemer.

Må-mål

Mål som må være oppfylt for at prosjektet skal kunne anses som en suksess

Bilde-tagging- og lagringssystem.

En funksjon som ble spesifikt etterspurt at oppdragsgiver er et bilde-tagging og lagringssystem. Målet med denne funksjoner er å gi faddere et innblikk i livene til deres respektive fadderbarn. En viktig del av en fadderorganisasjon er det å gi fadderne, som støtter organisasjonen, følelsen av at det de bidrar med faktisk nytter, at det gjør en forskjell. Intensjonen er dermed å oppnå dette ved å kunne gi jevnlige oppdateringer med bilder og blogger gjennom nettsiden.I dag skriver fadderorganisasjonen individuelle brev til deres respektive medlemmer, noe som er veldig koselig for fadderne, men forferdelig ineffektivt for

fadderorganisasjonen. Denne ineffektiviteten gjør også at oppdateringene som sendes ut til fadderne naturlig nok blir færre tiden mellom hver oppdatering kan bli lang.

Spørsmålet som må stilles ved dette delmålet er “hvordan lager man et funksjonelt og fleksibelt bilde lagrings og fremvisning system som fremhever organisasjonens arbeid og lar faddere enkelt og effektivt aksessere bilder og ta del i utviklingen til deres respektive fadderbarn.” Systemet må derfor fungere på en slik måte at det blir enkelt å tilknytte bilder til faddere og fadderbarn på en oversiktlig og brukervennlig måte, mens systemet samtidig ivaretar personens rettigheter i henhold til personvernloven.

Administrasjonspanel av organisasjonens faddere, ansatte og fadderbarn.

Et annet mål som ble spesifikt etterspurt av oppdragsgiver og som naturlig nok er helt vesentlig å ha når man skal skal lage et system for en fadderorganisasjon er et administrasjonspanel for alle ansatte, faddere og fadderbarn. Denne funksjonen forutsetter, på lik linje med de fleste andre av nettsidens funksjoner, at det ligger en godt utformet database i bunn som inneholder alle brukere, fadderbarn og relasjonene mellom dem. Derfor blir en del av dette målet å finne ut av hvordan man utformer databasen på en mest mulig hensiktsmessig måte og deretter utfører dette.

For å drive en fadderorganisasjon må man kunne både se og redigere eksisterende entiteter i databasen. En database der ingenting kan slettes eller endres blir veldig vanskelig og slitsom og bruke etterhvert som den fylles opp med rader som ikke lenger er i bruk eller viser feil informasjon. Det er dermed et av de viktigste målene å lage et system som enkelt håndterer dette. Enkelt er altså nøkkelordet her, store database systemer kan være tunge å jobbe med så en del av målet vil være å ha én enkelt side som gir deg mulighet til å se og endre informasjon, samt søke, filtrere og slette rader i databasen. Dette høres ut som noe som kan bli meget rotete ettersom man presser inn såpass mye mye funksjonalitet. Et annet mål blir dermed å organisere dette på en oversiktlig og brukervennlig måte slik at man kan for eksempel enkelt søke opp et fadderbarn for så å enkelt søke opp dets tilhørende fadder. “Hvordan gjøres dette effektivt med et enkelt design som utfører all funksjonalitet uten å måtte spre det utover et stort antall sider som krever mange sideinnlastninger” er spørsmålet jeg stiller meg for dette delmålet.

Bør-mål

Mål man bør ha oppfylt for å levere et godt produkt

System for blogginlegg og mail

Oppdragsgiver nevnte også at de per i dag sender brev til hver enkelt fadder, og at det hadde vært fint med en funksjon som gjør det enklere å nå ut til faddere via email. Målet med denne funksjonen er altså øke organisasjonens mulighet for å administrere “public relations” eller PR. Som nevnt over er det viktig å holde kontakt med fadderne for å vise dem at deres støtte har en innvirkning. Det er dermed planlagt både email- og blogg-funksjon for dette. Organisasjonen burde kunne sende ut nyhetsbrev til deres medlemmer og ansatte. Dette vil kreve et hensiktsmessig mail API og mailserver. Siden åpenhet om fadderorganisasjonens arbeid og engasjering av faddere er en viktig del av siden må jeg finne ut av hvordan man på enklest og på best mulig måte når ut til og administrere kommunikasjonen med nevnte faddere. Det å lage en tekst editor som lar deg lagre tekst som så kan vises på forsiden blir enkelt, men dette skaper ikke en særlig engasjerende leseropplevelse for nettsidens medlemmer og heller ikke særlig innbydende for bloggens forfattere. Blogg og nyhetsfunksjonen må derfor løse problemet på en pen og oversiktlig måte som tilbyr et

skriveverktøy er innbydende for ansatte å skrive i, samtidig som innleggene er behagelige å lese. Her må det også inkluderes en kommentarfunksjon slik at faddere kan engasjere seg direkte i innleggene og gi tilbakemelding på arbeidet.

Funksjonalitet for at faddere kan logge inn og se informasjon og bilder om sine fadderbarn.

Dette relaterer i stor grad til målet om å lage et bildetagging system, så i tillegg til at bilder skal kunne lagres i databasen med de rette relasjonene så må faddere også ha brukere med mulighet for å logge seg inn. Dette vil kreve et ganske omfattende innloggingssystem for å forsikre seg at informasjon blir lagret og sendt på en forsvarlig måte. I tillegg er målet å ha forskjellige brukerroller slik at forskjellige nivåer av brukere kan aksessere og administrere forskjellige deler av siden.

System for innmelding og godkjenning av nye brukere.

Et enkel og effektiv måte å få medlemmer på effektiviserer arbeidet til organisasjonen. Målet er at brukere skal kunne legges inn manuelt i databasen av administrator, men og gi brukere en mulighet til å manuelt be om medlemskap. Denne funksjonaliteten skal automatisk legge til brukere i databasen dersom forespørselen blir godkjent av administrator. I og med at dette i utgangspunktet ikke er en nettside som alle bare kan bli medlem av trengs det en prosess der forespørsler om medlemskap sendes inn og deretter enten blir akseptert eller avvist. Målet er altså og effektivisere denne prosessen slik at siden kan enkelt få medlemmer ved hjelp av noen få tastetrykk.

Kan-mål

Mål som kan være greit å ha, men som ikke er viktige for suksessen til prosjektet

Håndtering av donasjoner

Den siste store funksjonaliteten som bør nevnes er håndtering av donasjoner til organisasjonen. Dette blir ikke ansett som den viktigste funksjonen siden det håndteres relativ greit manuelt i dag ved at folk melder seg opp til månedlige fradrag. Derimot hadde det vært ideelt om prosessen kunne effektiviseres gjennom nettet, og det er sannsynlig at organisasjonen vil motta flere donasjoner når prosessen ligger enkelt tilgjengelig på nett og ikke krever at man direkte kontakter organisasjonen slik som det fungerer i dag. Donasjonene må håndteres av et hensiktsmessig API og være både lett synlig og tilgjengelig for både medlemmer og andre som ønsker å støtte organisasjonen. Nøkkelen her er å gjøre ting enkelt og digitalisere prosessen uten at de ansatte i organisasjonen trenger å løfte en finger. Systemet må på en måte som er veldig enkel å bruke for sidens besøkende kunne ta imot både engangs donasjoner og månedlige fradrag samtidig som det tilbys et verktøy for oversikt og administrasjon av donasjon systemet.

1.3 Forutsetninger og begrensninger

Prosjektet burde være innenfor rimelighetens grenser med mine nåværende kunnskaper innenfor webutvikling, selv om det faktum at jeg er alene om prosjektet potensielt kan begrense funksjonaliteten til det ferdige produktet ettersom jeg kun har meg selv å regne med. Det kan gjøre at for eksempel de mindre viktige målene ikke blir ferdig i tide til leveringen av prosjektet. Jeg forutsetter at all funksjonalitet, unntatt kanskje donasjonfunksjonen som jeg regner som den minst viktige, blir ferdig og funksjonalitetsmessig lever opp til en god brukervennlig standard. Designet skal være enkelt, oversiktlig, responsivt og, så klart, være pent, men jeg er ikke noen webdesigner så det og gjøre nettsiden virkelig flott å se på blir ikke min prioritet. Det er mye ferdigheter i det å lage

en veldig pen og innbydende nettside, men det krever mye tid og undersøkelser samtidig som det ikke bidrar til den generelle funksjonaliteten til nettsiden som er den jeg ønsker og få frem. Det er likevel viktig at nettsiden ser bra ut for å oppnå ønsket om brukervennlighet og effektivitet.

Brukervennlighet er altså viktig, og for brukervennlige sider er det ofte viktig å være mobilvennlig [1]. Dette blir derimot ikke en prioritet for meg, siden nettsiden i utgangspunktet ikke er laget for å brukes på mobil. Dersom jeg ønsket å også lage en mobilvennlige side måtte designet og funksjonene i stor grad legges om for å legge til rette for dette, noe som jeg ser på som unødvendig bruk av tid og ressurser samtidig som det kan skade effektiviteten og funksjonaliteten til siden. Siden vil kunne brukes på mobil, men det vil ikke bli tatt hensyn til gjennom utviklingsperioden. Siden skal likevel lages så responsivt som mulig for å kunne brukes på forskjellige skjermer og nettleservindu størrelser, det er bare på de aller minste skjermene jeg ikke prioriterer.

Som nevnt tidligere er dette systemet laget for en mindre fadderorganisasjon. Det blir altså ikke tatt høyde for det man ofte må tenke på når man lager et system for en større organisasjon som krever funksjonalitet for undersjefer, avdelinger av ansatte med spesifikke oppgaver og kanskje også funksjon for håndtering av regionene organisasjonen er spredt utover.

Selv om det er et håndteringssystem for en fadderorganisasjon som utvikles er det dessverre ikke tid nok for en person å dekke alle aspekter av en slik organisasjon på en god måte. Det er derfor viktig å velge ut de viktigste funksjonene og heller implementere dem på en så bra som mulig. Funksjonalitet som håndtering av lønn for ansatte har for eksempel ikke blitt tatt i betraktning siden i tillegg til å ikke være spesielt nødvendig i og med at det allerede er systemer på plass for å håndtere lønn, så er det en svært tidkrevende funksjonalitet som ikke bringer vitale funksjoner til nettsiden.

1.4 Problemløsning

For å løse ethvert stort problem i prosjektsammenheng kreves det planlegging, hvilke oppgaver skal gjennomføres, hva skal resultatet bli, hvilke problem skal løses og sist, men ikke minst, hvordan. Første steg vil være og definere problemet som i dette prosjektet vil være å finne ut hvordan man skal lage et engasjerende og effektivt nettside system for en mindre fadderorganisasjon. Deretter deler jeg dette problemet opp i mindre deler som hver løser sin del av det store hovedproblemet, slik som beskrevet i seksjon 1.2 under mål. Disse delproblemene løses deretter ved at de deles opp i mindre arbeidsoppgaver ut fra kravene til løsningen, som så får tildelt et estimert antall timer og en dato der de forventes å være ferdig. Arbeidet vil så bli utført i sprinter med tidsfrister for å nå milepælene i prosjektet som indikerer slutten på en sprint.

Etterhvert som siden lages vil jeg ta i bruk nyttige kilder for best mulig lage siden på en optimalisert og hensiktsmessig måte. Dette kan for eksempel være kilder på nettside design prinsipper, optimalisering av databasen, lignende løsninger og generelt andre kilder som gjør det mulig å nå målet mitt på en hensiktsmessig måte. Enkelte delproblemer er mer komplekse, for eksempel er det min intensjon å lage et bildebehandling og tagging system der spørsmålet blir å finne ut av hvordan man lager et responsivt, effektivt, oversiktlig og fleksibelt system for nettopp dette. Her er det ingen enkel løsning og strategien består i hovedsak av å sette seg inn i hvordan en bruker vil finne det enklest å håndtere disse bildene og deretter se for seg handlingsforløpet. For denne oppgaven og andre oppgaver vil jeg ta i bruk flytskjemaer for å illustrere for meg og andre hvordan, og i hvilken rekkefølge, oppgaver skal bli utført. Jeg har også intensjoner om å hente mye inspirasjon og ideer fra folk som har løst lignende problemer før. I dette eksempelet om bildetagging systemet vil det

være hensiktsmessig å se på Facebook som trolig har et svært mye brukt og velutviklet system for akkurat dette.

Det er få krav til løsningen fremsatt av oppdragsgiver. Oppdragsgiver vil ha en nettside for deres fadderorganisasjon som er betydelig mer fleksibel og effektiv enn det den er i dag, og som tilbyr funksjonalitet for å enkelt engasjere og holde kontakt med faddere. Funksjonelle krav fra deres side er de to delmålene som listes opp under Må-mål i delkapittel 1.2, bildetagging system og fadder, ansatt og fadderbarn administrasjonssystem. Siden disse to tingene er de eneste konkrete funksjonene som ble etterspurt har de blitt gitt en høy prioritet, samtidig som de er ganske viktige for å løse hovedproblemet. Utover dette har oppdragsgiver vært relativt stille og vanskelig å få kontakt med, så mine egne tanker om hvordan problemene i prosjektet skal løses og hvilke funksjoner som er vesentlige å ha med blir i stor grad opp til meg å bestemme og jeg utformer derfor i stor grad kravlisten på egenhånd basert på det jeg mener, med underbygning fra liknende suksessfulle løsninger, er hensiktsmessige krav til et slikt prosjekt.

Den nåværende nettsiden til fadderorganisasjonen, Butterfly Friends, er i dag ganske langt under standarden for denne typen organisasjoner. Dette gjør det vanskelig for nåværende og potensielle faddere å følge med og engasjere seg i arbeidet organisasjonen, noe som potensielt kan hindre fremgang fordi det minsker mengden donasjoner og medlemmer organisasjonen får når informasjon om arbeidet ikke er tilgjengelig. I tillegg til å levere en velfungerende side av skole- og hobbymessige interesser ser jeg på det som min jobb å hjelpe organisasjonen så den kan kommunisere enklere med sine medlemmer og jobbe mer effektivt, og derav forhåpentligvis indirekte hjelper nettsiden trengende barn ved å gjøre organisasjonens administrative arbeid enklere og mer åpent for omverdenen dersom de ønsker dette.

Under prosjektperioden vil det bli relevant å bruke hensiktsmessige verktøy for å gjøre koden bedre og strukturere prosjektet. Det vil bli ført timelister og ukentlige møter med veileder gir svar på spørsmål og hjelper å planlegge neste steg. Her vil det også bli tatt referater fra alle møter. Kode vil håndteres av versjonskontroll systemet Git der kode vil bli backet up på nett med tilhørende, beskrivende commit-meldinger. Det vil også være viktig å kommentere og dokumentere koden i prosjektet for fremtidig arbeid etter prosjektets slutt. Verktøyet ReShaper vil passe på at kode når standardene for kommersielt skrevet kode.

1.5 Rapportstrukturen

I de påfølgende kapitlene går jeg nærmere og nærmere inn på hvordan jeg har tenkt å løse problemene og oppnå målene for prosjektet, før jeg diskuterer litt rundt løsninger og veien videre og til slutt avslutter rapporten med en konklusjon.

Kappitel 2

Dette kapittelet omhandler hvordan Butterfly Friends har løst nettside problemet sitt i dag, og hva jeg kan hente fra det, samt forskjellige anerkjente løsninger på flere av de andre problemene i prosjektet med noen overfladiske kommentarer fra min side på hvordan jeg legger opp min egen løsning. Videre diskuterer jeg her designprinsipper jeg ønsker og forholde meg til for å oppnå målene jeg har for design før jeg avslutter med en liste med forklaringer av relevante ord og verktøy.

Kappitel 3

Kapittel 3 er løsningskappitetelet. Her presenterer jeg først kravene til løsningen, funksjonalitetsmessig, samt et kapittel for krav for design og hvorfor jeg mener det er hensiktsmessig å legge opp mitt design på denne måten. Videre følger et stort kapittel som i

detalj går inn på hvordan løsningene er implementert. Kapittel 3 avsluttes med testing av krav og resultater av testingen.

Kappitel 4

Kapittelet går ut på å drøfte løsnigene og implementeringene og argumentere for hvorvidt de faktisk løser problemene de var ment å løse og hvorvidt dette kunne være gjort på en annen like god eller bedre måte. Kapittelet tar også i kortfattethet for seg hva veien videre potensielt kan være dersom prosjektet skal ytterligere utvides.

Kappittel 5

Dette er det siste kapittelet i rapporten og er også prosjektets konklusjon. Her forsøker jeg å nøste opp i løse tråder, oppsummere og drive rapporten til en konklusjon av resultatet på arbeidet.

2 Teknisk bakgrunn

2.1 Dagens løsninger

Jeg er ikke den første som lager en nettside for en fadder- eller veldedighetsorganisasjon. De aller fleste organisasjoner i dag har tilgang på systemer for å motta donasjoner og holde faddere oppdatert, så det er naturlig å trekke inspirasjon fra disse når jeg skal lage min egen side. Disse nettsidene tilhører ofte store fadderorganisasjoner og det er ikke realistisk å nå fullstendig opp til dette nivået under denne prosjektperioden, spesielt med tanke på at jeg bare er én person som utvikler, men det er likevel nyttig å se på disse sidene med et kritisk blikk og hente ut aspekter fra design og funksjonalitet som jeg anser som nyttige. Det blir forøvrig også et ganske overfladisk blikk på løsningene ettersom størsteparten av funksjoner og problemløsninger som skal inn i prosjektet er kun tilgjengelig som medlemmer eller ansatte i organisasjonen, noe de også vil være i profesjonelle fadderorganisasjoner. Inspirasjonen som hentes fra disse sidene vil derfor i hovedsak være de offentlige funksjonene til nettstedene.

Det finnes ikke konkrete standarder på hvordan man løser de fleste problemene som inngår i dette prosjektet, selv om det er sterke antydninger fra allerede suksessfulle nettsider på hvordan et slikt problem kan løses på en god måte. Webdesignprinsipper vil bli tatt i bruk og selv om det finnes forskjellige meninger om hva som er de beste prinsippene å følge fins det noen universelle trekk som er ganske klare i en godt designet nettside. Det blir her veldig viktig å gjøre mine egne meninger samtidig som jeg ser på hva andre har gjort og mener når jeg lager det jeg ønsker at skal bli et effektivt og brukervennlig system.

2.1.1 Dagens løsning

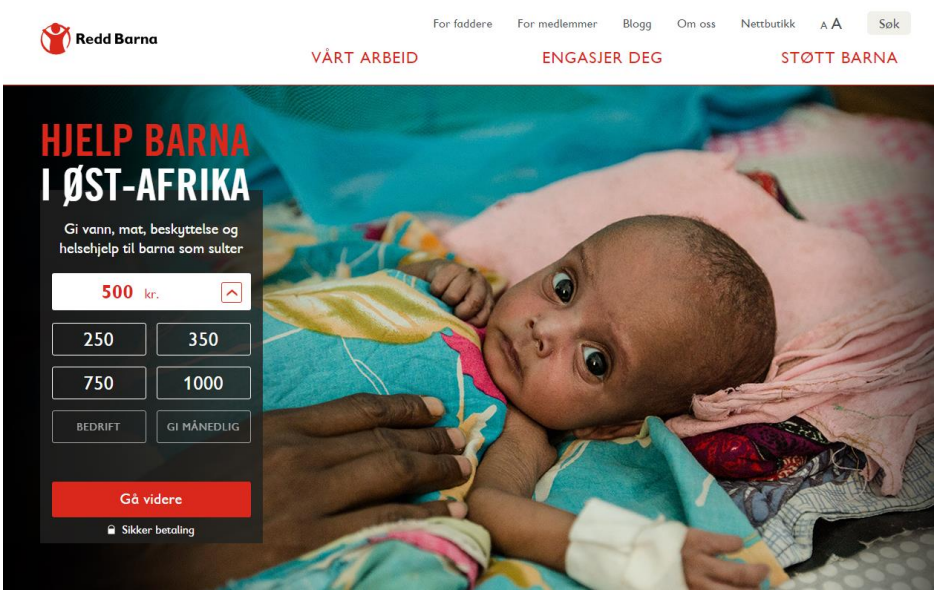
Figur 1: Figuren viser et bilde tatt fra den nåværende nettsiden til fadderorganisasjonen Butterfly Friends.

Den nåværende løsningen til fadderorganisasjonen Butterfly Friends består idag av en veldig statisk nettside. Dette vil si at siden har veldig mye innhold som ikke endrer seg etter behov. Nettsiden tilbyr mye informasjon om organisasjonen, men ikke organisert på en spesielt oversiktlig måte, med linker og informasjon litt spredt ut overalt på nettstedet. Siden tilbyr mangfoldige linker som enten er døde eller bare fører til en tom side, og nyhetene som er presentert på høyre side består av en årsrapport som ikke viser noe annet enn en tom side og en nyhetsartikkel fra 2014. Siden er altså statisk, informasjon er ikke oversiktlig og enkelt organisert og funksjonaliteten er manglende og i beste fall svært simpel. Siden tilbyr også en svært enkel medlemsinnmeldingsfunksjon, der informasjon skrives inn i et skjema og sendes til organisasjonen på mail.

Det er ikke mye å hente fra denne siden ettersom siden ikke holder opp til standarden for moderne nettsider i noen, eller i så fall svært liten, grad, og spesielt ikke siden det er en nettside for en fadderorganisasjon hvor det er såpass viktig å dele informasjon og ha mulighet til å bruke nettsiden til å skape en bro mellom faddere, fadderbarn og organisasjonsarbeidet. Min jobb blir heller å gjøre alt som denne siden ikke får til, ved å lage et mye mer dynamisk, brukervennlig, oversiktlig og innbydende system. Selv om jeg ikke kan få så mange tips på hvordan jeg bør legge opp min egen løsning fra denne siden så kan jeg likevel bruke den som et eksempel på hva som ikke bør gjøres. Døde linker eller linker som bare fører til tomme sider er et veldig dårlig designvalg og dette må hindres ved at man kan fjerne informasjon som nyheter som ikke er i bruk lenger og oppdatere og supplere med ny informasjon ettersom det er behov for det. Funksjonalitet som ikke fungerer eller bare gir feilmelding uansett hvilke hvordan det brukes må også unngås. Dersom siden inkluderes

tredjepartsfunksjonalitet som APIer bør disse funksjonene ha mulighet for å kunne skrus av eller fornyes ettersom API ikke nødvendigvis varer evig.

2.1.2 Redd barna



Figur 2: Bilde fra forsiden til fadderorganisasjonen Redd Barna

Redd barna er en av de større fadderorganisasjonene i norge med ca. 7500 medlemmer og et stort antall uregistrerte frivillige (tall fra 2013) [3]. Deres nettside, etter å ha undersøkt og utprøvd funksjonaliteten, har flere funksjoner og trolig bedre design enn det jeg vil være i stand til å oppnå i løpet av prosjektperioden, og derfor er det noen viktige design punkter å notere seg her: Når man først går inn på nettstedet møter man siden som vises over i figur 2. Presentasjonen, og spesielt førsteinntrykket, er viktig når du skal lage denne typen side som er avhengig av donasjoner og engasjement fra frivillige. Følelser engasjerer folk og får dem til å bidra med penger. Derfor er det strategist smart med en løsning som denne der det første som møter deg er et bilde av et “stakkarslig” barn som trenger hjelp, plassert rett ved siden av en mulighet for å velge donasjonsmengde og gi støtte. Dette spiller på sympatien folk har for trengende barn, noe som i sammenheng med veldedighetsorganisasjoner burde være moralsk akseptabelt selv om man bruker folks følelser til å få dem til å betale.

Under det første bildet som møter deg er har Redd Barna også en blogg- og nyhetsfunksjon som holder folk oppdatert på arbeidet og hjelper organisasjonen å vise hvor relevant nettopp deres arbeid er. Disse to funksjonene kontribuerer begge til engasjementet fra folk som besøker siden. Følelser, åpenhet og arbeid og informasjon er, som nevnt tidligere, veldig viktig for denne typen organisasjon og det blir relevant å bruke inspirasjon fra andres løsning, og da spesielt Redd Barna sin løsning som jeg anser som en god løsning, for å løse deler av dette prosjektets problem.

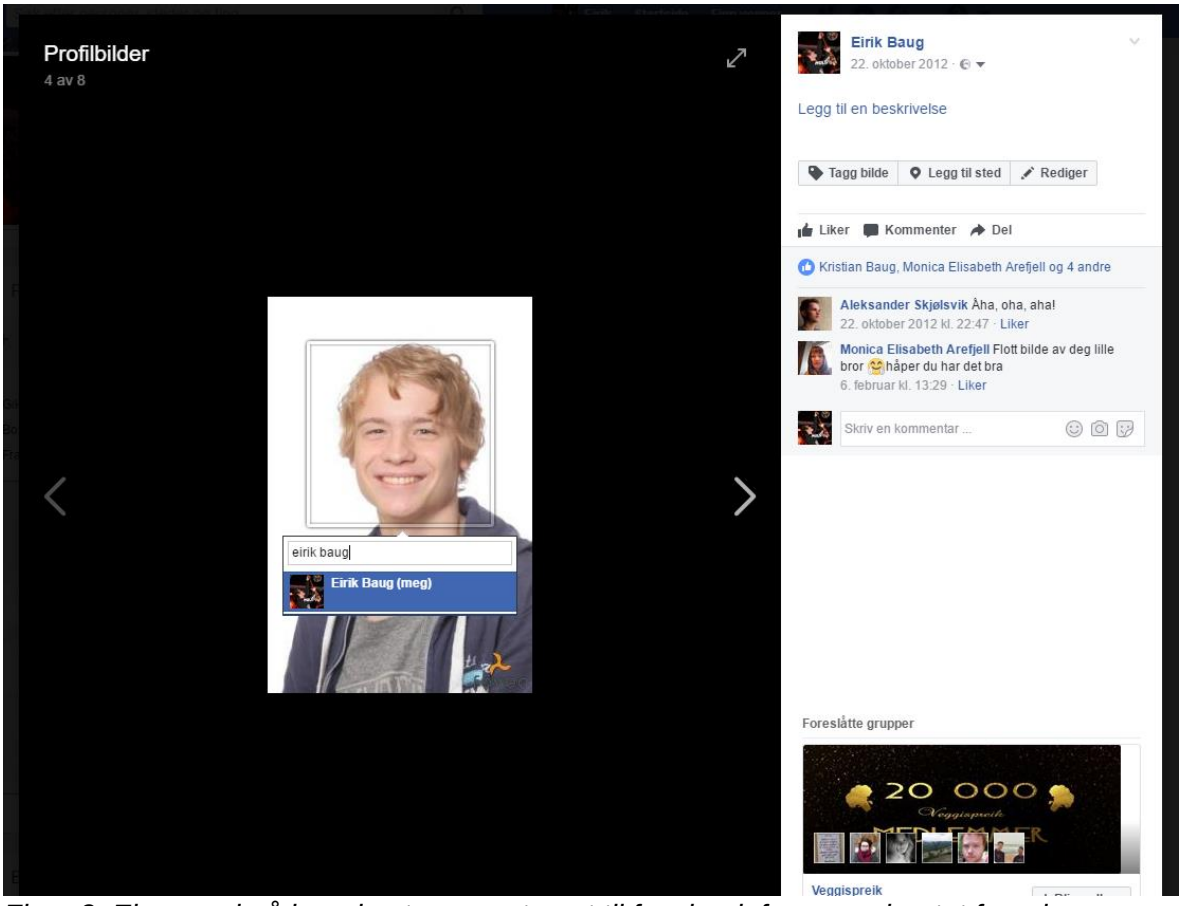
Løsningen jeg utvikler vil ta for seg, og forhåpentligvis, inspirere følelser og engasjement for en god sak. Forsiden er altså viktig i denne prosessen. Bilder, donasjoner og blogginnlegg blir sentralt og i fokus, samtidig som er også viktig å ikke overvelde brukeren med informasjon eller spille for mye på følelsene deres siden det fort kan bli irriterende og heller slå feil vei når folk føler seg manipulert. For min egen løsning av designet tror jeg det blir relevant å se på hvordan jeg kan gi forsiden et ganske simpelt design, samtidig som jeg legger en del vekt på bilder og blogginnlegg for å holde faddere oppdatert og engasjere nye

medlemmer, samt mulighet donasjoner og medlemskap. Når man først har klart å engasjere noen er det vesentlig at ikke donasjonsfunksjonen er for langt unna.

2.1.3 Facebook sitt bildevisning- og taggingsystem

Bildetaggingsystemet som skal implementeres på siden er noe ikke mange andre organisasjoner som har, ihvertfall ingen av fadderorganisasjon nettstedene jeg har undersøkt i dette prosjektet. Et slikt system mener jeg gir en liten fordel overfor andre organisasjoner ettersom det personaliserer nettsiden for faddere på et dypere nivå og lar fadderne knytte sterkere følelsesmessige bånd med sine fadderbarn når siden gir mulighet til å følge med på utviklingen og livene til barna du støtter. Det er en viktig del av helheten til nettsiden så derfor må de beste gode metoder for å implementere dette undersøkes.

Det finnes derimot ikke mange sider som tilbyr slike bilde tagging systemer til sine brukere. Store sosiale medier som Twitter og Google Plus har ikke noe slik funksjonalitet, men derimot har sosiale medier giganten Facebook akkurat dette. Facebook, som er det ledende sosiale mediet i dag med rapportert 1,94 milliarder aktive brukere månedlig i 2017 [2] tilbyr trolig det som man nærmest kan kalle en standard på et slikt system, ettersom det er utviklet over flere år med erfaringer og endringer. Systemet daterer ihvertfall helt tilbake til 2010 da jeg selv ble medlem av Facebook. Dette systemet vil bli viktig for min egen utvikling av et lignende system som skal tilby mye av den samme funksjonaliteten med enkelte endringer etter funksjonalitetsbehov for dette prosjektet og kritiske vurderinger av Facebook sitt system.



Figur 3: Eksempel på hvordan taggesystemet til facebook fungerer, hentet fra min personlige Facebook profil.

Figur 3, over, gir et innblikk inn i hvordan et slikt system fungerer på en så populær nettside som Facebook. Det man ikke ser så tydelig her er at bildet vises i en såkalt modalboks, som vil si en mindre boks som overdekker nettsidens innhold slik at den etterspurte informasjonen kan vises fokus på toppen av resten av innholdet. Flere andre store sider hvor bildevisning er sentralt som Twitter og Instagram viser bilder på samme måte, og dette er nok i dag den mest brukte måten å vise bilder på i og med at bilde kan forstørres og vises frem uten at en ny side må lastes inn. Bildet viser også hvordan taggingen fungerer ved at en boks omslutter vedkommendes hode og en søkeboks gjør det mulig å legge til brukere fra databasen som har fjeset sitt inne i boksen. Funksjonen tilbyr også at du kan legge til både sted og beskrivelse, samt et kommentarfelt og "liker-knapp".

Alle som har brukt dette systemet vet at det er både enkelt og effektivt, noe som gjør det verdt å trekke inspirasjon fra. Det må derimot nevnes at mitt eget system ikke er tilknyttet et sosialt medie slik som Facebook sitt system i høyeste grad er. Systemet jeg utvikler vil bli ganske lukket og bildene vil kun være tilgjengelige til innloggede brukere som også er tagget i dem eller har fadderbarn som er. Derfor anser jeg det ikke som nødvendig å legge til slike funksjoner som deling og liker-knapper, og heller bare implementere det som er simpelt, effektivt og nyttig for denne løsningen. Taggeboksen med søkefeltet er derimot høyst relevant i denne sammenheng, og det jeg tror er den viktigste delen av systemet til Facebook å ta med meg videre. Her tilbys det en enkel måte å tagge venner ved hjelp av en boks som viser akkurat hvilken person som er tagget og hvor i bildet personen befinner seg, sammen med et raskt og effektivt søk etter brukere i databasen. Systemet som jeg skal implementere kan potensielt bestå av et bilde hvor man legger til mennesker som er med i en boks under bildet, men dette blir noe mangelfullt i forhold til ønsket funksjonalitet. Det vil fungere best dersom faddere kan se nøyaktig hvem som er deres fadderbarn på bildet ettersom de ofte ikke har møtt sine respektive fadderbarn i virkeligheten og det kan være vanskelig å vite hvem som er hvem i et bilde med mange barn.

Facebook tilbyr også automatisk ansiktsgjenkjenning som effektiviserer taggeprosessen ganske mye og den er såpass sofisitikert at du ikke kan legge til taggebokser manuelt fordi deres algoritme finner i de aller fleste tilfeller det som er å finne av fjes. Jeg har ikke tilgang til en så sofistikert algoritme så det er nok gunstig at jeg tillater å manuelt legge til boksene, selv om det å ha en ansiktsgjenkjenningsalgoritme til hjelp i tillegg ville vært ideelt.

2.1.4 Blogginnlegg og Medium

Som jeg allerede har nevnt mange ganger i denne rapporten er relasjonene med givere og medlemmer ekstremt viktig å holde ved like. Det er viktig å holde kontakten med medlemmer, vise at arbeidet gjør nytte og inspirere engasjement gjennom tydelig og jevnlig kommunikasjon med nevnte brukere. Det er flere måter å nå ut til folk på, men den enkleste måten er å legge ut nyheter og oppdateringer på organisasjonens egne nettsider. Av alle nettsidene til de meste kjente veldedighetsorganisasjonene har jeg ikke funnet en som ikke implementerer en form blogg- eller nyhetsfunksjon. Jeg skal altså også implementere en slik funksjon, og her er det viktig å se på hvordan suksessfulle nettsider har løst dette før meg for å best mulig lage en funksjon som ser bra ut og er enkel og behagelig å bruke for den ansatte som håndterer denne funksjonen. Det er ganske åpenbart at funksjonen må se bra ut og kunne representere organisasjonen på en god måte, men det er nesten like viktig at funksjonen er behagelig og innbydende å bruke for de ansatte slik at de enkelt kan skrive og administrere artikler på en enkel og motiverende måte.

Figur 4: Eksempel på blogginleggskriving i Medium

Medium er en av de ledende bloggsidene på markedet i dag med mellom 51 og 200 ansatte [4]. De tilbyr en veldig enkel og veldig effektiv måte å dele dine ideer og erfaringer på. Skjermbilde over er tatt direkte fra deres "rich text editor" (online teksteditor) som enda så enkel den ser ut tilbyr en veldig kraftfull og behagelig skriveopplevelse. Det er noe slikt jeg vil oppnå for mitt system ettersom dette systemet tilbyr enkel bilde- og video- opplasting og visning, samt et enkelt konsept der tittelen blir det første avsnittet du skriver og etterhvert som du skriver legger til avsnitt og bilder i artikkelen følger det en veldig simpel, men effektiv toolbar med deg nedover på siden (toolbaren aktiveres ved å trykke på "+" tegnet). Mer skal ikke til for å lage en blogg, for det viktigste er ikke at det er masse funksjoner, men heller at de funksjonene som faktisk er der er en fornøyelse å bruke. Overflødig mange funksjoner kan i tillegg virke mot sin hensikt ved at de forvirrer brukeren i stedet for å hjelpe. Målet her er å lage pene, leselige artikler og for dette kreves ikke veldig innviklet redigeringsfunksjonalitet slik som for eksempel Word tilbyr, men det er nok heller mer hensiktsmessig å lage artiklene på ganske like, men fleksible formater slik at ikke hver artikkel på siden blir seende vidt forskjellige ut.

I tillegg til funksjonaliteten gjør Momentum det enkelt ved å bygge bloggen sin under et WYSIWYG prinsipp. WYSIWYG står for "what you see is what you get" er et prinsipp som handler om å lage webapplikasjoner der brukeren ikke må kunne html eller et lignende markup språk for å legge til innhold på en nettside [5]. I denne sammenheng betyr det altså at slik bloggen ser ut når du redigerer den under skriveprosessen er slik den blir seende ut når innlegget faktisk blir publisert og lagt ut for offentligheten på nettsiden. Dette er en praksis jeg gjerne vil implementere i dette prosjektet ettersom det er usannsynlig at de ansatte som skal administrere denne siden har kunnskaper til html og koding. Jeg vil altså ta en del av prinsippene til Momentum når jeg implementerer systemet mitt, både på hvordan

det ser ut og hvordan fungerer. I tillegg må artiklene kunne lagres og publiseres, samt et større system for å filtrere og editere allerede opprettede artikler.

2.2 Designprinsipper

Designprinisipper omhandler hvordan man bør legge opp designet og funksjonaliteten til nettsiden din slik at nettsiden ser bra ut og er enkel å bruke. Siden brukervennlighet og effektivitet er såpass relevant for min oppgave er det også viktig at jeg undersøker gode praksiser for dette og, så sant det er mulig holder meg til disse under utviklingsprosessen.

Feedback

Feedback, eller tilbakemelding, er den informasjonen nettsiden gir til brukeren hver gang brukeren gjør en handling. Dette kan for eksempel være at en knapp blir mørkere når man trykker på den slik at brukeren faktisk får opplevelsen av å trykke på en knapp selv om det bare er noen piksler på en skjerm som endrer seg. Feedback kan også være elementer som viser at applikasjonen arbeider, som fremdriftslinjer og spinnende hjul (lastesymbol). Dette er ekstremt viktig for brukervennligheten til siden. Selv om prosessen ikke tar noe lengre tid om du ikke har en fremdriftslinje som holder styr på fremdriften så er det veldig frustrerende å ikke vite hvorvidt prosessen er igang eller ikke, og gjør at siden vil virke tung og slitsom å bruke. Brukeren trenger altså visuell bekreftelse på at deres handling faktisk ble utført. [21]

Navigation

Navigasjonen på siden er viktig. Den bør være enkel og intuitiv. Et rotete system er ikke innbydende å arbeide med, så det er viktig at sidens funksjoner deles opp i logiske deler og plasserer på hensiktsmessige plasser på nettsiden. Det bør være tydelig hva som er hvor og hvordan funksjonaliteten skal nås, samtidig som navigasjonen ikke må være for innviklet. navigasjonen bør altså kunne lede deg dit du skal uten å ta for mye plass og være forvirrende. Den bør også deles opp i et logisk hierarki der funksjoner som følger etter hverandre på en hensiktsmessig måte i hierarkiet. Et godt prinsipp og følge er “Tre klikks regelen” som sier at brukeren bør kunne finne det de leter i tre klikk eller mindre fra et hvilket som helst punkt på nettsiden. [21]

Farger og fonter

Noe av det første brukere legger merke til når de laster inn siden er fargene. Fargene bør være vennlige og lyse, i hvert fall hovedfargen, og man bør velge farger som komplementerer hverandre, altså at de er forskjellige farger, men samtidig nære i nok i spekteret til at det blir en behagelig flyt mellom dem. Sider som Facebook, Twitter og Youtube bruker helt hvit for å vise innholdet og gjerne en Komplementær farge, som en form for lys grå som bakgrunnsfarge som ligger rundt hovedinnholdet. En tredje farge bør også velges, og det er gjerne denne som blir den fargen folk assosierer med en spesifikk nettside. Denne fargen representerer siden. Facebook har for eksempel en blåfarge og Youtube en rødfarge. En annen viktig visuell funksjon for at siden skal bli god å bruke er fonter. Riktig font, fontstørrelse og og linjehøyde må velges slik at siden passer til sitt formål og er behagelig å lese fra, spesielt for en side som baserer seg i så stor grad på å dele informasjon som i dette prosjektet.

Layout

Layouten eller oppsettet bør være enkel og ryddig. Ting må plasseres logisk sammen, gjerne i en grid layout, hvor siden deles opp i seksjoner med relaterte elementer. Det er viktig å adskille innhold som er av forskjellige typer for å lage et mest mulig ryddig oppsett. Brukere

leser ikke gjennom ting, de skummer. Derfor bør innhold organiseres i et såkalt f-mønster. [21]

Figur 5: Bildet viser hvordan brukere leser informasjon på en nettside [22]

Som figur 5 over forsøker å illustrere så leser brukere informasjon i et slags F-mønster, som vil si at de starter oppe til venstre og leser mot høyre, mens fokuset avtar etterhvert som blikket beveger seg nedover på siden. Dette er viktig å ta høyde for, slik at de viktigste funksjonene kommer i fokus. [21] En annen viktig ting å inkludere i layouten er rett og slett tomrom. Hvis du tar en titt på store, suksessfulle, sider ser du at det er mye tomrom på siden som ikke er fylt med annet enn enn en enslig farge. Dette gjør innhold adskilt og strukturerer siden ved å adskille innhold, slik at en god mengde tomrom er viktig for brukervennligheten. Populære sider som Twitter og Youtube tar hyppig i bruk tomrom for å lede brukeren til det som faktisk er viktig å vise. [21]

Lastetid

Brukere liker ikke å vente og blir fort utålmodige. Tar ting for lang tid er sjansene store for at siden bare blir lukke før prosessen er ferdig stor. Derfor er det viktig, så sant det lar seg gjøre, å optimalisere lastetider, databasesøk og bildestørrelser slik at brukeren ikke blir ventende for lenge.

Responsivt design

Et responsivt design er et design som fungerer på mange skjermstørrelser. Når skjermen skaleres ned følger også resten av innholdet med og plasserer seg på plasser som gir mening. Jo bedre nettsiden fungerer på forskjellige skjermstørrelser jo mer responsiv er den, og det er viktig at nettsiden kan ta høyde for at nettleservinduet forandrer seg etter brukerens preferanser. De fleste moderne sider bruker rammeverk som Bootstrap for å oppnå mest mulig responsivitet, og det blir også det som blir brukt i stor grad i denne løsningen.

Presentasjon av informasjon

Når man lager sider som inneholder veldig mye informasjon som for eksempel en nettside for en fadderorganisasjon slik som i dette prosjektet så er det viktig å bare vise den informasjonen man trenger, når man trenger det. For eksempel når man lister ut informasjon fra databasen er det ikke ønskelig å vise all informasjon om entiteten på en gang. Kun når brukeren spør om å redigere et bilde eller se informasjon om en ansatt bør denne informasjonen vises. Inntil brukeren ber om tilhørende informasjon bør denne informasjonen gjemmes, for så å vises på en responsiv og rask måte når bruker ber om dette.

2.3 Ordforklaringer og verktøy

Under gir jeg en rask innføring i diverse verktøy og begreper som er sentrale i prosjektet.

Git

Git er et versjonskontrollsystem for håndtering av kode i gruppe eller individuelt. Git lar deg gjennom kommandolinja laste opp og hente kode som er lagret på en server, og lar deg lage uavhengige grener av koden som gjør at forskjellige personer i gruppeprosjektet kan jobbe på hver sin "kodegren" uten å komme i konflikt med hverandre. [7]

C#

Objektorientert programmeringsspråk som er utviklet av Microsoft og baserer seg på C++ og Java. Språket er utviklet i forbindelse med .NET plattformen til Microsoft og er også kodespråket som kjører på serversiden av nettsiden i dette prosjektet. [8]

.NET

C# programmer kjører på .NET rammeverket som er en integrert del av alle windowsystemer som kjører med det virtuelle utførelsessystemet Common Language Runtime eller CLR. .NET rammeverket inneholder mange klassebiblioteker som brukes ofte i C# koding og gjør språket til det det er i dag. ASP.NET MVC, som bruker i dette prosjektet, er et webrammeverk basert på .NET. [8]

Google ReCaptcha

Google ReCaptcha er et API utviklet av google for å bekjempe "bots", som er servere og datamaskiner som på programmatisk vis prøver og aksessere, hente informasjon fra og lage brukere på sider. Disse bottene har sjelden ærlige intensjoner og Google ReCaptcha APIet beskytter nettsiden din mot angrep fra disse ved å tvinge bottene til å verifisere seg som ekte brukere. [9]

SendGrid

SendGrid er en mailtjeneste med tilhørende API. SendGrid lar deg enkelt sende mail over nettet ved hjelp av simple mail transfer protocol (smtp). APIet har også støtte for vedlegg og lar deg enkelt spore og se statistikk over den mailen som er sendt ut. [10]

Areas

Areas er en måle og dele opp og strukturere kode i ASP.NET MVC applikasjoner. Dette er nyttig for applikasjoner som deles opp i veldig forskjellige deler som for eksempel en nettside med et kundepanel og et admin panel. Hvert area på siden får hver sin mappe med tilhørende undermapper som inneholder egne kontrollere, modeller og views. Dette gjør koden mer strukturer, oversiktlig og enkel å jobbe med. [11]

MVC

MVC er arkitekturmønster for utvikling av software og er en måte å strukturere kode på. MVC baserer seg på at koden deles opp i tre logiske komponenter; modeller, views og kontrollere (derav forkortelsen MVC) som alle har sin oppgave i software systemet og håndterer og sender informasjon mellom entitetene. Hver entitet har sin dedikerte oppgaver, der kontrollerende håndterer den avanserte logikken og beregningene, som så fyller modellen med verdier som igjen sendes for å vises til bruker i viewet. [12]

HTML

HTML (Hypertext Markup Language) er et markup-språk for utvikling av nettsider og lar deg enkelt utforme og designe views ved hjelp av CSS som er et hjelpemiddel for å endre og designe HTML elementer etter behov. [15]

Bootstrap

Bootstrap er et populært Javascript, HTML og CSS rammeverk som gjør frontend design av nettsider betydelig enklere. Med Bootstrap kan siden din veldig raskt bli pen og strukturert uten at du må skrive særlig mye css på egenhånd siden du bruker de ferdiglagde CSS klassene innebygd i rammeverket. [16]

JSON

JSON står for Javascript Object Notation og er en standard for å serialisere og utveksle data på en enkel tekstbasert notasjonsform mellom to systemer eller innenfor samme system. JSON er basert på Javascript sin objekts notasjon for å vise å lagre informasjon, noe som gjør biblioteket enkelt og intuitivt å jobbe med. [6]

AJAX

AJAX står for Asynchronous Javascript And XML. XML er litt misledende i navnet for selv om XML var standarden når AJAX først ble introdusert er det JSON som er mest brukt i dag for overføring av data med AJAX. AJAX er et Javascript kodebibliotek som tar i bruk noen teknikker for å asynkront kommunisere med serveren uten at man krever at siden må lastes inn på nytt. Veldig nyttig når man trenger å hente spesifikk informasjon fra serveren og kun ønsker og oppdatere en liten del av den innlastede siden med denne informasjonen. [17]

jQuery

jQuery er et populært Javascript Bibliotek som er laget for å forenkle Javascript utvikling i nettleseren. Javascript baserer seg på å binde events og funksjoner til DOM-elementer i nettleseren og jQuery kutter ned koden som kreves for å få tak i disse elementene. jQuery hjelper også å utjevne forskjeller mellom forskjellige nettlesere og gjør at du slipper å tenke på hvordan din kode vil kjøre i nettlesere som potensielt tolker ting forskjellig. Biblioteket tilbyr også mange nye Javascript-extension metoder, metoder som ikke finnes i vanlig Javascript. [18]

jQueryUI

jQueryUI er en brukergrensesnittplugin til jQuery og tilbyr en rekke elementer for å gjøre brukeropplevelsen på siden rikere og mer tilfredstillende for brukeren. Funksjonaliteter i pluginen er for eksempel fremdriftslinjer, autocompleting inputs og popupbokser. [20]

Vue.js

Vue er et frontend rammeverk for utvikling av nettsider av samme type som de kjente rammeverkene Angular.js og React.js. Vue er fokusert på og godt egnet for å programmere applikasjoner med behov for “two-way-binding” som vil se at når en variabel endrer seg, som for eksempel teksten i et inputfelt, så oppdateres automatisk også kodevariablene knyttet til inputfeltet deler av siden oppdateres der det er nødvendig for å legge til rette for dette. Når man initialiserer Vue.js lager det et dataobjekt som inneholder data og metoder for siden, dette bindes så til DOM-elementet og Vue håndterer så alle variabler og metoder på siden. [19]

Modalbokser

Modalbokser er en teknikk som lar deg vise informasjon på nettsiden ved å legge en boks eller modalboks over innholdet for å vise den informasjonen som bruker er ute etter akkurat i dette øyeblikket mens all annen informasjon blir liggende i bakgrunnen. Dette er altså bokser som fyller i større eller mindre grad nettleservinduet med informasjon og som kan raskt krysses vekk for igjen å vise innholdet på nettsiden bak.

Stripe

Stripe er en betalingstjeneste utviklet av Stripe.com som aksepterer betalinger for produkter og tjenester med VISA eller mastercard via sitt egetutviklede API. APIet er utviklet med tanke på token-basert betaling der nettsiden i seg selv ikke håndterer kortinformasjon, men heller leverer informasjonen til APIet som returnerer en token som kan brukes til å belaste kortet til bruker. [36]

ID

ID står for identifikasjon og i HTML sammenheng referer det til en eksakt HTML element på siden aksesserbart med en unik ID.

Div-elementer

Div elementer er sentrale HTML elementer som definerer en seksjon på siden. Disse elementene har ingen annen funksjonalitet enn å holde grupper av relaterte elementer og dermed gjøre det mulig å aksessere og strukturere disse gjennom det overliggende div-elementet.[47]

Paging

Paging er en teknikk for å dele opp store lister og dermed effektivisere innlasting ved å kun vise de en del av hele listen av gangen. Ved hjelp av navigasjonsknapper kan vi bevege oss til større og mindre indekser av listen. [49]

Nuget

Nuget er en pakkehåndteringsprogram for .NET applikasjoner som gjør det mulig å hente inn og bruke eksterne pakker som er en del av nugest open-sorurce kildegalleri. Disse pakkene gjør det mulighet å introdusere, modifisere og bruke kodebiblioteker for .NET laget av profesjonelle bedrifter og privatpersoner [50].

3 Løsning

3.1 Krav

3.1.1 Krav til Bildeadministrasjonsystemet

Må ha

- 1.1 Taggebokser
    - Bildene må kunne tagges med visuelle taggebokser som viser hvilken person som befinner seg inne i boksen. Disse boksene må kunne flyttes rundt på bildet og størrelsen må kunne endres
- 1.2 Autofullfør databasesøk

    - Personer fra databasen bør kunne legges til i taggeboksene. Her trengs en autofullfør funksjon som henter og foreslår personer fra databasen når brukeren skriver i taggeboksen.
- 1.3 Databasekoblinger
    - Bildet må kobles sammen med brukere og fadderbarn fra databasen som befinner seg i bildet basert på taggeboksene. Deretter kan disse bildene hentes opp igjen, slik at en fadder kan se bilder av seg selv og sine fadderbarn
- 1.4 Bildefremvisning
    - Bildene må vises frem for faddere på en oversiktlig og behagelig måte, med taggebokser intakt. Det med derfor være en bildefremvisningsfunksjon. BIldene kan heller ikke vises frem alle på en gang, ettersom den kan være store mengder av dem i databasen. De må dermed vises frem på en hensiktsmessig måte og hentes etter behov.
    -

Bør ha

- 1.5 Bildeadministrasjonspanel
    - Bildene må kunne administreres fra adminpanelet etter at de er lastet opp. Her må man kunne endre på informasjonen til bildet som beskrivelsen og taggeboksene. Man skal også kunne slette bilder eller deaktivere dem så de ikke er tilgjengelige for brukere.
- 1.6 Beskrivelse og tidspunkt
    - Bildene må kunne vise en beskrivelse og tidspunkt for når de ble lagt inn i databasen
- 1.7 Opplastning med AJAX
    - Bildeopplastningen bør kunne skje med AJAX for å gjøre systemet mest mulig effektivt og brukervennlig slik at nettsiden ikke trengs å lastes inn på nytt for hver gang et bilde blir lastet opp. AJAX opplastingen bør også kunne følges via en fremdriftslinje.
- 1.8 Optimalisert databasesøk
    - Mye av funksjonaliteten på siden baserer seg på å søke i database, da er det viktig at henting av informasjon fra denne databasen er effektiv og rask. Dette gjelder spesielt for autofullfør funksjonen til taggeboksene.

Kan ha

- 1.9 Automatisk ansiktsgjenkjennelse
    - Bildene bør ha mulighet til å bruke automatisk aniktsgjenkjennelse. Dette må gjøres gjennom et hensiktsmessig API eller bibliotek.

3.1.2 Krav til databaseadministrasjonsystemet

Må ha

- 2.1 Søk og filtrering
    - Bruker må kunne søke og filtrere i databasen basert på søkekriterier. Disse søkekriteriene bør være relativt avanserte som for eksempel bør man kunne vise alle barn som er født 19.02.2005 og som også heter Terje.
- 2.2 Paging
    - Hele databasen kan ikke listes ut på en side. Paging må dermed brukes for å dele elementene opp i lister med et visst antall elementer per side.
- 2.3 Informasjonsredigering

    - Informasjonen til brukere og fadderbarn må kunne endres. Dette inkluderer de fleste databasevariabler, som navn og hvorvidt brukeren er utestengt fra siden eller ikke og profilbilde.

Bør ha

- 2.4 Informasjon i modalbokser
    - Visning av brukerinformasjon og redigering bør skje med modalbokser slik at bruker slipper å laste inn en helt ny side for hver bruker det trengs informasjon om.
- 2.5 Rollehierarki
    - Systemet bør ha et hierarki av roller. Det vil si at brukere som er lavere eller på samme sted i hierarkiet ikke kan endre informasjonen til hverandre. På toppen skal det være en bruker, eieren av siden, som kan endre informasjonen til alle andre
- 2.6 Effektiv jobbing
    - Arbeidsprosessen skal effektiviseres og systemet skal være enkelt å jobbe med. Det betyr at informasjon bør kunne hentes og endres uten at det krever fullstendige sideinnlastninger hver gang. Dette må gjøres med AJAX kall mot serveren. Dette gjelder også søk og filtrering.

3.1.3 Krav til blogg/nyhetsfunksjon

Må ha

- 3.1 BIldeopplastning og redigering
    - Bloggen må kunne ha mulighet for opplastning og enkel redigering av bilder. Dette må være en enkel prosess og bildene må kunne lastes opp uten at siden må lastes inn på nytt. Bildene må også kunne flyttes rundt i artikkelen etter brukers ønsker.
- 3.2 Publisering- og lagringsfunksjon
    - Artikler bør kunne lagres enkelt under skriveprosessen, helst via skrivebrodsnarveien ctrl+s. Det bør også være funksjonalitet som hindrer at en ansatt med uhell legger ned nettleseren og endringer blir tapt. Artikler kan heller ikke bli publisert med en gang de lagres. Det må være en separat publiseringsfunksjon som gjør at artikkelen faktisk blir lagt ut på forsiden.
- 3.3 Søk- og filtreringsfunksjon, samt redigering
    - Etter et en artikkel er skrevet og publisert, eller bare under skriveprosessen etter at artikkelen er lagret så må innlegget kunne hentes opp igjen. Altså må bruker kunne lagre en artikkel som er under arbeid for så å hente den opp igjen og fortsette der de stoppet. Dermed må en søk og filtreringsfunksjon være implementert. Bruker må kunne søke gjennom og finne sine artikler og filtrere på databasevariabler som innhold og dato. Dette må også fungere med paging og fungere uten page refresh. Artiklene må også kunne slettes når de hentes opp igjen, dersom dette er ønskelig

Bør ha

- 3.4 WYSIWYG type skriveverktøy
    - Blogg/nyhetsfunksjonen må være av type “what you see is what you get” for å sikre at ansatte ikke trenger å tenke på hvordan artikkelen de skriver kommer til å se ut på forsiden. Slik den ser ut når den skrives er altså slik den skal se ut når den er publisert.
- 3.5 Fonter og stiler
    - Skriveverktøyet må tilby forskjellige fonter og stiler/verktøy for å gjøre leseopplevelsen bedre. Dette inkluderer funksjonalitet som tabeller, Youtube videoer og sitater.

- 3.6 Hjelp og tips
    - Brukeren må ha tilgang til en funksjon som kan fortelle litt om hvordan man bruker editoren

Kan ha

- 3.7 Kommentarsystem
    - Artikler må ha et kommentarsystem sånn at brukere kan engasjere seg i diskusjonen om innlegget når innlegget er publisert.

3.1.4 Krav til emailfunksjon

Må ha

- 4.1 Teksteditor
    - Det kreves en hensiktsmessig HTML teksteditor for å skrive og utforme mails. Her må bilder kunne legges til skriftstørrelse velges.
- 4.2 Mottakere
    - Bruker må kunne velge hvem som skal motta emails. Det burde være funksjonalitet for å velge og sende til alle faddere, alle ansatte eller begge deler. I tillegg, dersom ikke alle skal få mailen så kan man søke og legge til personer fra databasen. emailer som ikke ligger i databasen bør også kunne legges til.
- 4.3 Sendingsbekreftelse og emne
    - Når mailen skal sendes må man bekrefte at den faktisk skal sendes ved å legge til et emne for mailen. Emailen lastes så opp og sendes uten at nettsiden oppdateres slik at bruker kan lese over og se at det de sendte var riktig.
- 4.4 Mailserver og API
    - Applikasjonen må ha tilgang til en mailserver som sender mailen der den skal og tilhørende API for å kommunisere med serveren.

Bør ha

- 4.5 Vedlegg
    - Mailen må også kunne støtte opplastning av vedlegg, som bilder, dokumenter og videoer.

3.1.5 Krav til medlemskapsforespørelsesfunksjon

Må ha

- 5.1 Innsendingsskjema
    - Det må være et innsendingsskjema der bruker skriver personlig informasjon som adresse, email og navn. Her må bruker også skrive litt om seg selv og hvorfor de vil bli medlem av organisasjonen.
- 5.2 Akseptering og sletting av forespørseler
    - Når en forespørsel blir sendt inn blir den lagt til i databasen og kan deretter bli hentet opp i administrasjonspanelet. Her trengs det en hensiktsmessig og enkel måte og vise forespørsler på. Dette inkluderer søkefunksjon og paging. Bruker må kunne akseptere eller slette disse forespørslene. Når en forespørsel blir behandlet skal det sendes en email til forespørselens oppgitte email som forteller om utfallet. Her kan det også legges til en personlig melding fra personen som behandler forespørselen.
- 5.3 Passordsetting

    - Når en bruker har blitt akseptert til siden, blir det automatisk laget en bruker i databasen med oppgitt informasjon. Derimot blir det ikke satt noe passord, for dette må bruker sette selv. Når bruker deretter får mailen får de en link til nettsiden der passord kan settes.

Bør ha

- 5.4 ReCAPTCHA bott-beskyttelse
    - Innmeldingsskjema, som er åpen for offentligheten må ha en verifikasjon via et ReCAPTCHA API som kan sjekke at brukeren faktisk er menneskelig. Skjema kan ikke sendes inn med mindre dette er tilfellet.
- 5.5 Brukervilkår
    - Brukeren må, før innsending av skjema, godt brukervilkår presentert i en pdf. Skjema kan ikke sendes inn før disse er godtatt. Brukervilkårene kan lastes opp i administrasjonspanelet.

3.1.6 Generelle krav

Må ha

- 6.1 Kodestruktur og dokumentasjon
    - Koden må tilstrekkelig kommenteres og møte kommersielle standarder for kode slik at senere utvikling av koden blir enklere.

Bør ha

- 6.2 Sosiale medier
    - Organisasjonen må kunne legge til sine egne sosiale medier på nettsiden. Disse vises så på hensiktsmessige steder.
- 6.3 Forsidebilder
    - Bilder som skal vises må kunne legges til på forsiden. Disse må kunne legges til fra administrasjonspanelet.
- 6.4 Adresse og generell informasjon om organisasjonen
    - Adresse og generell informasjon om organisasjonen må kunne legges til i administrasjonspanelet. Dette vises så på hensiktsmessige steder.
- 6.5 Administrering av API nøkler
    - Organisasjonen må kunne administrere og håndtere sine egne API nøkler og passord for å håndtere dette.
- 6.6 Passordresetting
    - Dersom en bruker har glemt sitt passord kan kan skrive inn sin email og få tilsendt en link der passord kan resettes. Passord må også kunne resettes dersom man er innlogget.
- 6.7 Profilbildeopplastning og endring av personlig informasjon
    - Bruker må kunne sette sitt eget profilbilde og endre informasjon om seg selv som adresse og telefonnummer.

3.1.7 Krav til donasjonsystem

Må ha

- 7.1 Donasjoner
    - Donasjoner må kunne gjøre til organisasjonen, enten anonymt eller ikke. Dersom brukeren er logget inn kan donasjonen linkes til denne brukeren. Bruker må også kunne sette seg opp på en månedlig donasjon plan.
- 7.2 Kvittering
    - Kvittering på donasjon vises i nettleseren og sendes på mail. Bruker kan deretter dele donasjonen på sosiale medier om de ønsker.
- 7.3 Abonnementer

- Donasjonsabonnementer må kunne legges til i administrasjonspanelet. Dette vil si hvor mye penger som skal trekkes hver måned dersom en fadder velger abonnemangsfunksjonaliteten. Abonnementer må også kunne skrus av og fjernes fra databasen etter ønske fra administrator.

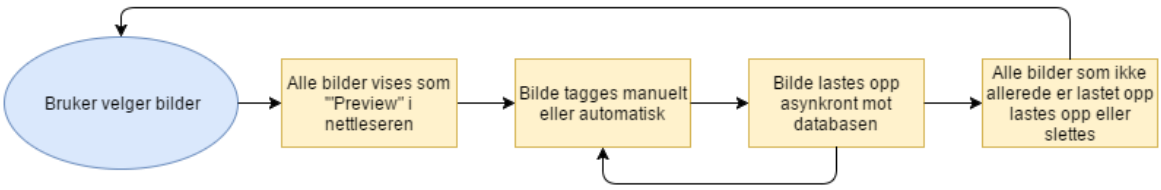
Bør ha

- 7.4 Oversikt over donasjoner
  - Det bør være mulighet for å se oversikt over donasjoner i administrasjonspanelet. Her kan man se hvem som donerte hva, når og hvor mye. Søk og filtrering for å finne frem til donasjoner.

3.1.8 Krav til design

Det er visse krav som må oppfylles dersom de forskjellige systemene skal best mulig levere den funksjonaliteten som er ønsket. Overordnede krav er krav som at systemet skal være effektivt, oversiktlig, brukervennlig, gi nok tilbakemelding og tilby et enkelt, men pent design slik som nevnt i tidligere kapitler. Videre følger mer konkrete krav til navigasjon og design av diverse funksjoner på nettsiden. Disse kravene er reflekterer sterkt, som teksten vil vise, resonnementene og prinsippene til design som begrunnet i kapittel 2.1 og 2.2.

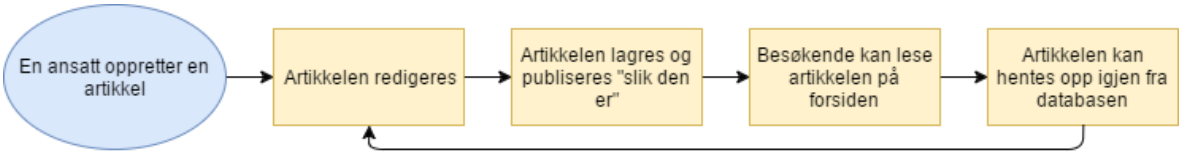
Bildetaggingsystemet



Figur 6: Diagrammet viser hvordan bildetaggingsystemet ideelt sett skal fungere

Over vises diagrammet på hvordan flyten i bildetaggingen skal gå. Intensjonen er at mange bilder skal håndteres samtidig, og ikke bare en og en av gangen slik som de gjerne gjøres på Facebook. Med det mener jeg at administratorer på nettsiden ofte vil laste opp større bulker av bilder samtidig, så det å her vise dem i modalbokser hver for seg vil trolig skape mer frustrasjon enn gjør systemet brukervennlig. Jeg velger derfor å gjøre dette ved at brukeren velger bilder og disse bildene vises så i nettleseren uten å lastes opp til serveren, noe som Facebook velger å gjøre. Det er ikke hensiktsmessig å laste opp disse bildene før de er ferdig behandlet så inntil videre hentes de bare direkte fra der de ligger på datamaskinen din og vises deretter i nettleseren som en forhåndsvisning. Bildene blir deretter plassert etter hverandre horisontalt nedover, som gjør at bruker enkelt kan håndtere ett og ett bilde, for så og bla litt ned og ta neste bilde etter at forrige er ferdig behandlet. Brukeren kan her legge til taggebokser automatisk via et ansiktsgjennkjenningsbibliotek eller legge boksene manuelt til bildet med musepekeren. Disse boksene må kunne fjernes, flyttes og størrelsen må kunne reguleres for å passe de forskjellige ansiktene i bildet, og de vil alle inneholde en søkeboks der man kan legge til brukere som ligger i databasen. Deretter, når man er ferdig, kan man laste opp bildene asynkront mot databasen og holde styring på opplastningsprosessen via fremdriftslinjer for de forskjellige bildene. Når dette er gjort kan nye bilder velges som erstatter de gamle, opplastede bildene.

Artikkelsystemet

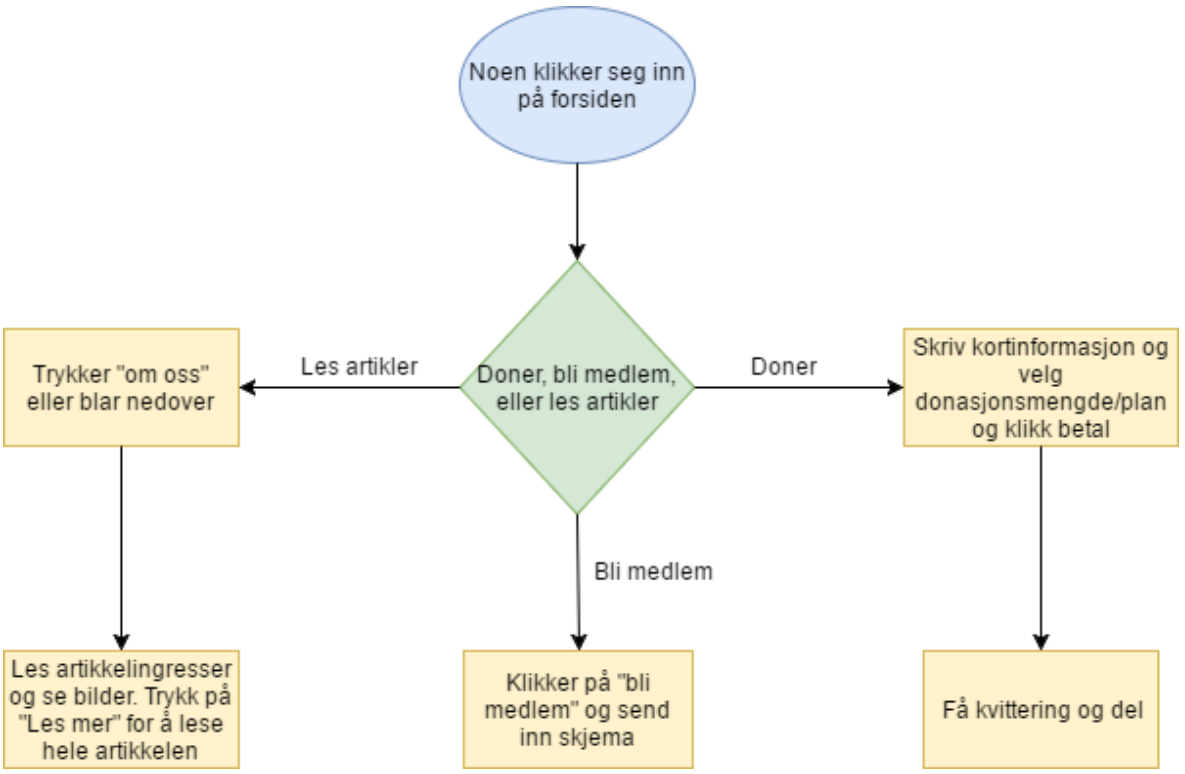


Figur 7: Handlingsforløp for håndtering av innlegg på nettsiden

Over vises et enkelt handlingsforløp på hvordan jeg tenker å legge opp min løsning for håndtering av artikler og blogger. Først opprettes et dokument som redigeres og utformes av brukeren, her gjerne med forskjellige stiler, bilder og videoer. Deretter lagres og publiseres denne artikkelen. Artikkelen kan naturligvis også lagres uten at artikkelen også blir publisert slik at den kan gjøres ferdig på et senere tidspunkt. Når artikkelen er publisert legges den ut på nettsidens forside slik som den vises når den redigeres i teksteditoren, uten at forfatteren må tenkte på formatering av tekst og form. Innlegget kan så leses av alle som måtte finne på å besøke siden. Dersom forfatteren, eller enhver annen med tilstrekkelig tillatelse ønsker å behandle artikkelen vil behandle artikkelen etter at den er lagret og lukket vil den kunne hentes opp igjen fra databasen ved hjelp av et søk- og filtreringsystem. Her kan artikkelen redigeres, lagres, slettes eller publiseringen stoppes. Prosessen kan gjentas etter dette.

Forsiden

Nettsidens forside er det første en ny person ser når han for første gang klikker seg inn på siden. Denne siden danner altså førsteinntrykket til brukeren, og det er viktig å gjøre et godt førsteinntrykk, ikke overvelde brukeren med informasjon, men samtidig være oversiktlig og vise hvor brukerne skal gå for å komme dit de vil. Det vil her være viktig å prøve og vekke engasjement og følelser i brukeren, slik som uttrykt i kapittel 2.1.2. Brukere vil dermed, når de først laster inn forsiden bli møtt med bilder og mulighet for å donere penger. Dersom du ikke er logget inn vil du også bli spurt om hvorvidt du ønsker å bli medlem av siden. Alt må være pent organisert uten for mye informasjon på en gang. Dersom brukere blar nedover på siden vil de komme til diverse artikler pent presentert med bilder, overskrifter og litt om hva de handler om som de kan trykke på for å lese mer om nettopp dette. Informasjon må være lett tilgjengelig og lett å tilegne seg; altså lesbart og presentert på en oversiktlig måte. Du vil også fra forsiden ha enkel tilgang til en “om oss” type side som tar for seg informasjon, adresse og kontaktinformasjonen til organisasjonen.



Figur 8: Figuren viser hvordan handlingsforløpet for en ny besøkende til siden vil utarte seg.

Dersom du allerede er en eksisterende bruker på nettsiden gjelder ikke dette i samme grad siden du da også kan velge å logge inn, gå til profilen din eller se bildene dine. Du vil heller ikke bli spurt om du ønsker å bli medlem når du allerede er innlogget som medlem.

3.2 Designspesifikasjoner

Javascript og AJAX

For klientsideprogrammering er det hensiktsmessig å bruke scriptingspråket Javascript. For nettopp klientsiden, altså den delen av siden som er tilgjengelig på din datamaskin, er det få alternativer til Javascript. Javascript er i praksis det eneste gode alternativet dersom du ønsker å bygge en moderne nettside. Alternativer som ActionScript og Java er i dag lite eller ikke brukt i det hele tatt ettersom Javascript utvikling stadig blir bedre med tilføring av nye teknologier og plugins introdusert til språket som gjør det mulig å legge til mye funksjonalitet på siden med bare noen få linjer kode. I tillegg har Javascript AJAX, et bibliotek for asynkron kommunikasjon med serveren. Siden brukervennlighet er en veldig stor del av dette prosjektet er AJAX svært viktig siden det gjør det mulig å oppdatere mindre deler av siden og databasen uten at du trenger å oppdatere hele siden og på den måten får systemet til bli betydelig mer behagelig og effektivt å bruke. [17]

Webrammeverk

Som rammeverk for dette prosjektet har jeg valgt ASP.NET MVC 5 med, som nevnt, Javascript på klientsiden og C# på serversiden. Det skal ikke legges skjul på at dette valget har med at dette er et rammeverk jeg har god kunnskap til fra før av og jeg vet godt hva dette rammeverket er i stand til. Dette har gjort at jeg har måttet tilbringe mindre tid på å lære og sette meg inn i rammeverket noe som dermed også effektivisert arbeidet ganske mye og

kutter ned læringsperioden før implementering av nye funksjoner. Selv om det hjelper å ha kjennskap til rammeverket er ikke dette den eneste grunnen til at det er dette jeg har valgt.

Identity Framework

En del av målet i min prosjektoppgave er å ha et fullstendig innloggingssystem med egne brukere og brukerroller. Identity framework er nettopp dette, et rammeverk for håndtering av brukere som kommer komplett med innloggingssystem, registreringssystem, passord setting og brukerklasse (ferdig klasse for brukere som kan utvides med ekstra felter og funksjoner etter behov). I tillegg har rammeverket støtte for roller som vil si at hver bruker får tildelt en rolle som tilsier deres rettigheter på siden [14], som dermed også hjelper å oppfylle et av kravene i kapittel 3.1; brukerhierarki.

Det og skrive og implementere et funksjonelt innloggingsystem som ivaretar og tar i bruk gode sikkerhetsrutiner er en ganske stor jobb og krever mye testing. Derfor har det aldri vært planen og implementere noe slikt fra bunnen av i dette prosjektet når det er allerede velfungerende alternativer å velge mellom.

Areas

Areas er en måte å strukturere kode på ved å dele forskjellige seksjoner av siden inn i sine separate mapper med egne kontrollere, modeller og views.[11] Dette er nyttig for sider som har klare forskjeller på utseende og funksjonalitet i de forskjellige regionene av siden, som for eksempel i dette prosjektet som er delt opp i et administrasjonspanel og et brukerpanel for vanlige brukere og folk som ikke er innlogget. Dette er ikke nødvendig for funksjonaliteten til siden, men det gjør koden betydelig mer oversiktlig og enklere å jobbe med store systemer.

Partial views

Partial views er ikke et konsept som er unikt for ASP.NET, men det er såpass relevant for arbeidet i dette prosjektet at jeg blir nødt til å nevne det. Partial views er, som navnet tilsier, deler av views. Dersom du ønsker å kun oppdatere en mindre del av siden kan jeg hente et partial view, fylle det med de variablene fra serveren som jeg har behov for, og fylle det inn i hoved-viewet brukeren befinner deg på. Dette er ekstremt relevant for jobbing med AJAX og for en effektiv brukeropplevelse når vi henter og oppdaterer segmenter av sider uten å oppdatere hele siden.

Entity Framework

Entity Framework er et rammeverk for å kommunisere med og håndtere data i en .NET applikasjon. Entity gjør oppsett og databasehåndtering enkelt. Når du lager et .NET prosjekt og velger at du vil ha med database kommer databasen ferdig oppsatt sammen med resten av prosjektet. Entity tilbyr en relasjonsdatabase som passer utmerket for dette prosjektet og har også en innebygd database initializer som gjør det enkelt å legge inn testverdier for å teste applikasjonen.

Entity Framework baserer seg på en egen type SQL kalt Entity SQL [23] og er gratis å bruke under lisens fra Microsoft [24]. Entity abstraktifiserer databasearbeid ved å hente database entiteter og la deg håndtere dem som vanlige objekter i programkoden. Databasen og dens tabeller og rader aksesseres gjennom såkalte LINQ queries som gjør at du kan gjøre databasespørringer enkelt og uten å måtte bruke et spesialisert SQL språk [25]. Databasetabeller kan også skrives som vanlige klasser i C# som under kompileringsprosessen omformes til Entity SQL tabeller.

Entity Framework er altså svært nyttig for arbeid mot relasjonsdatabaser siden du sparer en del tid på å la Entity, som et mellomlag mellom applikasjonen og databasen, ta seg av oversettingen mellom C# LINQ queries. [25]

Rammeverk Versjon

Versjonen av rammeverket jeg har valgt å bruke, ASP.NET MVC 5, er ikke nyeste versjonen av rammeverket. Den aller nyeste versjonen er ASP.NET MVC .Core som er et relativt nytt rammeverk som differensierer seg fra MVC 5 på flere måter.

Grunnen til at jeg velger å holde meg til MVC 5 og ikke gå over til .Core er delvis fordi MVC 5 er noe jeg er kjent med og gjør at jeg raskt kan få til det jeg vil som er veldig nyttig i en prosjektsituasjon som denne med en fastsatt deadline.

Etter å ha gjort en del undersøkelser kommer jeg frem til at det ikke er noe som rammeverket MVC .Core kan gjøre som ikke MVC 5 også kan gjøre. Derimot er det visse optimaliseringer og forenklinger som gjør MVC .Core til et bra alternativ. .Core er raskere til å kompilere og oppdatere kode under utviklingsprosessen siden det kjører på et filsystem og det har støtte for Grunt som enkelt gjør det mulig å søke etter og legge Javascript pakker til i prosjektet ditt [26]. MVC .Core er også plattformuavhengig, og rammeverket er ekstra optimalisert for å kjøre på skybaserte webplatformer [27]. Det er en del flere forskjeller enn disse også, men det virker ikke hensiktsmessig å ta for seg alle her. Disse forskjellene er ikke ting som er veldig vesentlige når optimaliseringene og endringene er såpass små, selv om rent ytelsesmessig er nok .Core et noe bedre rammeverk [27]. Derimot er det ingenting MVC .Core tilbyr som jeg må ha; installering av Javascript pakker kan jeg enkelt gjøre manuelt, cross-platform utvikling har jeg ikke bruk for når jeg jobber alene på min Windows maskin, og optimalisering av skykomputasjon vil ikke ha stor mye å si for applikasjoner som ikke kjører veldig tunge prosesser og kalkulasjoner som nettsiden i dette prosjektet.

Problemet med .Core er derimot at at strukturen på prosjektet er endret seg og en del kode-syntaks har fått fra mindre til større endringer som gjør at det krever ekstra tid å sette seg inn i det nye systemet, tid jeg ikke føler jeg kan påkoste meg i en prosjektsammenheng når gevinsten er såpass liten. Rent funksjonalitetsmessig er MVC 5 også et sikrere valg ettersom det er et rammeverk som har vært ute på markedet i årevis og hjelpemidler, tutorials og svar på kodespørsmål finns det store mengder av på internett. I tillegg, på grunn av prosjektstruktur og at hvordan prosjektet kjører på et fundamentalt nivå, er endret i MVC .Core er det diverse pakker som ikke lenger støttes i den nye versjonen av rammeverket. For eksempel støttes ikke pakker som PagedList.MVC, som er den paging pakken jeg bruker i prosjektet.

API valg

Enkelte funksjoner i nettsiden styres av ekstern kode som må håndteres med riktige APIer.

Email API

Kravene tilsier at siden skal ha funksjonalitet for å sende email med vedlegg og derfor er det viktig å velge et API som gir mening. Det første APIet jeg vurderte var å bruke Gmail sitt API, eller en annen stor emailtjeneste som Hotmail eller Yahoo, sine SMTP(Simple Mail Transfer Protocol) servere til å sende email, i og med at disse vanligvis er gratis. Problemet med disse er at dersom du har en gratis konto kan du kun sende email til 500 personer av gangen før du blir utestengt i 24 timer. [28]. Dette blir for lite for en fadderorganisasjon med over 500 medlemmer. Gmail tilbyr også at du kan øke grensen til 2000 interne og 500 eksterne (ikke Gmail brukere) mottakere for 4 euro i måneden. [30] Dette er også noe i laveste laget.

Sendpulse er en annen email tjeneste som i hovedsak baserer seg på abonnementsbaserte emails, men som også har mulighet for sending av unike email. De tilbyr 2500 abonnenter og 15000 emails i måneden helt gratis. Over det koster det fra 9.85 dollar og oppover. [30]

Til sist er det emailtjenesten SendGrid som tilbyr opptil 40000 email i måneden for 9.95 dollar hver måned. [31] Dersom du eier en Azure konto og lager din SendGrid konto gjennom denne får du også 25000 email i måneden helt gratis. [32]
Av alternativene er det SendGrid som blir valgt, spesielt fordi de tilbyr 25000 gratis email med en Azure konto og nettsiden vil høyst sannsynlig bli hosted på Azure ettersom dette er svært enkelt med en ASP.NET applikasjoner og Azure tilbyr en pay-as-you-go tilnærming der du betaler kun for det du bruker. 25000 email burde holde en måned for en organisasjon på størrelse med denne.

ReCAPTCHA API

Det trengs også et API som håndterer bruker validering og sjekker at siden blir aksessert av ekte mennesker og ikke eksterne programmer, i og med at det blir lagt ut et innmeldingsskjema ubeskyttet og i all offentlighet på nettsiden. Etter å ha søkt litt rundt ser jeg fort at det ikke egentlig er andre alternativer enn Google sitt ReCaptcha API. Det finnes andre alternativer, men etter personlig testing virker det ikke som det er noen som ser ut til å komme i nærheten av design og funksjonalitet som Google leverer på dette området.

Donasjon API

Til slutt må jeg ta for meg hvilket API for donasjoner jeg mener er mest verdifullt å bruke. De tre alternativene som virker rimeligst er Stripe, Paypal og Chargify.

Priser:

- Chargify: Laveste pristrinn er 149 dollar i måneden eller 1.2% av inntektene for opptil 12000 dollar. i måneden. Er inntekter mer enn dette må du opp på neste pristrinn som er 299 dollar og 1.2% av inntektene. Tjenesten har støtte for periodiske betalinger, men her koster det 1 dollar per betaling (ca. 8.56 kr) . I tillegg må det betales “gateway fees” som vanligvis ligger på rundt 2.9% + 30 cent. [33]
- Paypal: Paypal tilbyr betalingsløsning for 3.4% + 2.80 kr for salg innad i Norge og 3.4% + et fast gebyr for betalinger fra utlandet. Utenlandsbetalinger er generelt dyrere. Tjenesten tilbyr støtte for periodiske betalinger. [34]
- Stripe: Stripe tar 2.4% + 2 kr for norske kort og 2.9% + 2 kr for utenlandske. Denne løsningen har også støtte for periodiske betalinger.

Paypal har også tilbud for veldedige organisasjoner med en avgift på 1.9 % + 2.80 nok. Dette gjelder derimot kun dersom organisasjonen har en inntekt på mellom 80000 og 400000 kroner i måneden, noe jeg tviler på er tilfellet for en fadderorganisasjon med ca. 600 medlemmer.

Paypal er nok definitivt det dyreste alternativet her og APIet er i tillegg kjent for å være vanskelig å jobbe med så det går ut. Stripe ser generelt ut til å være det billigste alternativet samtidig som API er laget for å være lett å bruke siden det er laget med visjon om enkel implementering av betalingsløsninger. [36]

Klientsiderammeverk

Klientsiden trenger strengt tatt ikke et rammeverk, men det kan være nyttig og effektivt å ha, ihvertfall for deler av applikasjonen. Populære klientsiderammeverk inkluderer Angular.js, React.js og Vue.js. Vue er det nyeste av de tre og også rammeverket som er raskest voksende i popularitet. Sammenlignet med Angular er kompleksiteten til Vue lavere og optimaliseringen av kode er bedre ettersom Angular har en tendens til å blir tregere dersom

du har for mange bindinger i dokumentet. Vue tilbyr også en mye løsere struktur som ikke krever at du holder deg til like strenge krav for kodestruktur slik som Angular krever. [37] React og Vue er relativt like ytelsesmessig med Vue liggende litt foran. Jeg er derimot mer kjent med Vue enn de andre rammeverkene så det er det jeg har valgt for dette prosjektet når det likevel kommer ut på toppen funksjonalitetsmessig.

3.3 Implementering

Implementering av prosjektet deles inn i logiske deler etter hvilke problemer som er løst og funksjoner som er implementert. Dette inkluderer konkrete løsninger og også overordnede designvalg.

Som et utgangspunkt for implementeringen av prosjektet ligger relasjonsdatabasen. Her refererer jeg til vedlegg 1 som består av et bilde av databasemodellen. Her stadfestes veldig viktige relasjoner i databasen som at en fadder kan ha ett eller flere fadderbarn og og både faddere og fadderbarn kan ha ett eller flere bilder tilknyttet seg. Det er også forsøk på å unngå redundans i størst mulig grad ved å normalisere databasen ned til 3. normalform der dette lar seg gjøre. [45] For eksempel er det lagt til en adresse tabell slik at brukere med samme adresse kan tilknyttes samme rad i adressetabellen, og når en ny bruker legges til sjekkes det om adressen allerede finnes i databasen. Dersom dette er tilfellet legges bruker til den allerede eksisterende adressen. I enkelte tabeller som donasjoner er ikke adressene nødvendigvis fullstendig oppgitt ettersom det er opp til bruker å dele så mye de ønsker slik at disse er ikke koblet til de vanlige adressene.

Forskjellen på vanlige brukere og ansatte er at de ansatte er også tilknyttet en eget ansatt tabell med relevant informasjon om ansatte. Jeg har også lagt til egne tabeller for API nøkler og henter dermed nøklene fra databasen når det er bruk for dem. Dette er for å unngå dårlige API nøkkel praksiser som å legge dem direkte i kildekoden eller i prosjektfiler. [38]

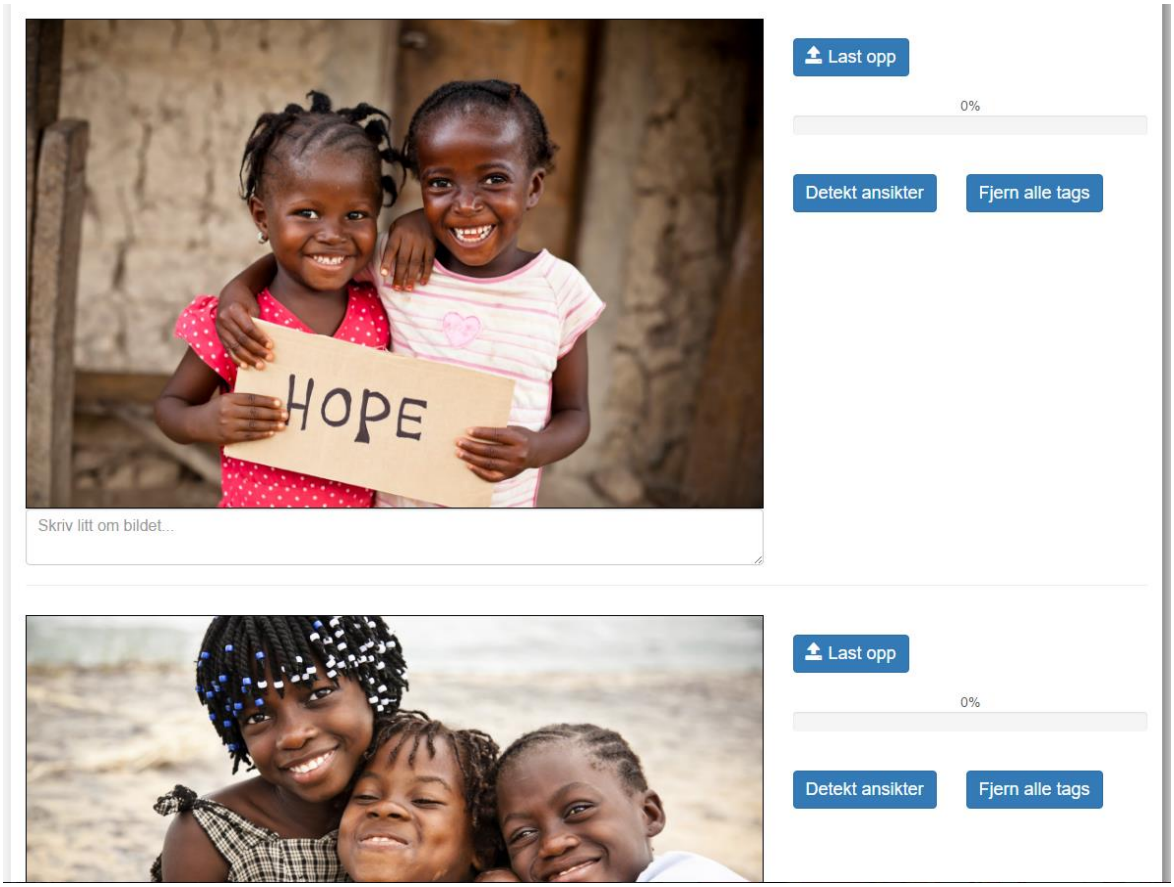
Databasen inneholder også en egen filtabell som håndterer alle typene filer i prosjektet som bilder, PDFer og videoer. Disse differensieres med en FileType enum som sier hvilken type fil det er.

Generelt sett har koden blitt kommentert og dokumentert med C# sine innebygde dokumentasjonsfunksjoner. Dette vil si å dokumentere funksjonalitet i koden ved å skrive funksjonenes bruksområde og parameter over funksjonen og også kommentere inne i funksjonene der det er nødvendig. Koden er også skrevet med kommersielle standarder ved å bruke verktøyet ReShaper sine koderengjøring og formateringsfunksjoner som automatisk jobber gjennom koden og formaterer stil og variabelnavngiving slik at det følger samme kodestandard. [46] Klientside kode har også blitt dokumentert der det føles nødvendig, men ikke i filene som faktisk sendes til klientsiden. Det er ikke god praksis å gi vekk sitt åndsverk ved å kommentere klientsidekode som deretter lastes inn av bruker som kan lese all koden fra sin datamaskin. I stedet blir de originale kodefilene kommentert mens kodefilene som lastes inn på klientsiden er komprimerte versjoner av disse filene uten kommentarer slik at både åndsverket er beskyttet og innlastingen blir optimalisert når kodefilene tar mindre plass.

3.3.1 Bildeadministrasjon- og taggingsystem

Et av prosjektets største problem er å sette opp et bildetaggingsystem for bildevisning til medlemmer. Jeg har tidligere i rapporten nevnt Facebook sitt eget system som etter min egen erfaring med dette kan si at er svært robust og effektivt. Løsningen min er dermed løst basert på dette systemet med unntak at det ikke er meningen at bildene skal legges ut til offentligheten.

Først og fremst har bildetagningen fått sitt eget panel som er lagt til Adminpanelet. Denne delen av administrasjonspanelet kan nås dersom du er en ansatt, eier eller administrator i organisasjonen. Her kan du velge hvilke bilder du ønsker å laste opp ved hjelp av HTML sin fileinput knapp. Du kan her også velge ett eller flere bilder for opplastning. Jeg legger en eventlistener på DOM objektet til filinput elementet og følger med på endringer i hvilke filer som blir valgt og tilknyttet dette elementet.



Figur 9: Skjermbilde fra oppsettet til systemet med 2 bilder valgt

Det som skjer når du da har valgt alle filene du skal ha fra datamaskinen din og klikker "åpne" er at det kjøres en loop som looper over disse objektene. Det første som skjer er at det sjekkes at filen faktisk er et bilde, for dersom dette ikke er tilfellet er jeg ikke interessert i å bruke filen videre. Dette stopper likevel ikke applikasjonen fra og behandle de andre bildene som er valgt slik som forventet. Jeg lager deretter et Javascript FileReader objekt som kan lese diverse fildata fra bildet. Videre bruker jeg Filereader sitt onload event som kjører hver gang filereader er ferdig med å lese en fil og binder funksjonalitet til den. Denne funksjonaliteten går ut på å vise bildet i nettleseren sammen med tilhørende knapper og fremdriftslinjer tilknyttet bildet. Bildet får her tildelt en unik ID som tilsvarer iterasjonen i den nåværende for-loopen og jeg legger et HTML span element som inneholder alle elementer tilknyttet bildet. Knappene som tilhører hvert bilde er knapper for å laste opp, detektere ansikter og fjerne alle tags.

Filereader objektet kan også lese posisjonen til bilde på datamaskinen din og det er denne posisjonen jeg bruker som kildeURL i image-tagen som bildet legges inn i. Bildet er altså ikke lastet opp ennå, det bare vises i nettleseren med URLen tilsvarende der den er funnet på din datamaskin. Dette er viktig for å oppfylle kravet om at bilder skal kunne tagges før de lastes opp. Alle elementer tilknyttet bildet får sine egne unike IDer basert på IDen til bildet. Dersom

brukerne er misfornøyd og ikke vil bruke de valgte bildene kan de bare velge noen nye og deretter iterere over listen og fjerne alle bilder som allerede er der før igjen jeg legger til de nye. Bildet legges deretter til først i et output element som viser alle bildene på rekke horisontalt nedover som vist på figur 9. Dette er gjort med tanke på enkel håndtering av arbeid. Jeg vurderte her også å behandle hvert bilde enkeltvis ved å vise dem i hver sin modalboks og trykke neste når du var ferdig med taggingen, men jeg landet på at dette trolig er den enklere og mer oversiktlige måten siden du kan bla nedovere eller oppover etterhvert som du jobber med bildene og dermed får et bedre overblikk.

Det siste som gjøres med et bilde når det velges er å legge til en håndteringsfunksjon. Denne funksjonen tar inn IDen til bildet og legger til en eventlistener på div elementet som omkranser bildet som lytter etter dobbeltklikk. Når man dobbeltklikker på bildet legges det til en boks rundt musepekeren din som i utgangspunktet er 150 piksler bred og 150 piksler vid. Denne boksen kan endres i størrelse og flyttes etter behov.

Figur 10: Eksempel på taggebokser lagt til et bilde.

Figuren over viser eksempel på et bilde som har to bokser assosiert med seg. Dene ene boksen er klarere enn den andre fordi, som bilde ikke viser siden skjermbilder ikke inkluderer musepekere, er musepekeren min over denne boksen og den blir dermed fremhevet. Dette gjøres med CSS3 sine skaleringsattributter.

Siden boksene plasseres inne i div elementet som omkranser bildet må koordinatene til boksen kalkuleres ut fra bildet som sitt eget koordinatsystem der oppe til venstre er posisjon (0,0). Når jeg da henter posisjonen til musepekeren for å plassere boksen der musepekeren er må jeg ta hensyn til at musepekerkoordinatene som ikke er de samme som koordinatene til boksen siden disse kalkuleres ut fra det relative koordinatsystemet boksen befinner seg i, nemlig det omkransende div-elementet. jeg bruker dermed Javascript sin offset funksjon som finner ut hvor langt div elementet til bildet er fra kanten av skjermen. Denne verdien trekkes

så fra musepekerposisjonen som deretter gir differansen og dermed også posisjonen innad i bildets koordinatsystem som vist under.

```
var parentOffset = $(this).parent().offset();
var x = e.pageX - parentOffset.left;
var y = e.pageY - parentOffset.top;
```

I tillegg må det tas høyde for at boksene kan ende utenfor bildets rammer. Bokser utenfor bildet blir ikke seende bra ut ettersom det er ingen poeng i å tagge utsiden av bildet. Dette vil skje dersom du dobbeltklikker ved kanten av bildet og boksen som omkranser musepekeren vil da delvis ende opp utenfor bildet. Dette er enkelt å løse dersom dette ender med at x eller y er mindre enn 0 for da er det bare og sette dem til nettopp 0 som flytter boksen akkurat innenfor på venstresiden. Dersom dette ikke er tilfellet må jeg igjen finne differansen mellom der boksen slutter og der bildet slutter og så trekke fra denne verdien.

```
if (x + width > $("#" + id).width()) {
  var overflowX = x + width - ($("#" + id).width());
  x -= overflowX;
}
```

Som vist i kodesnutten over der jeg behandler for x sjekker jeg om lengden av boksen pluss posisjonen til musepekeren blir lengre enn bredden til bildet og dersom dette er tilfellet trekker jeg fra denne “overflow”-differansen som flytter boksen akkurat innenfor grensene til bildet.

Koordinatverdiene blir så brukt til å lage en taggeboks. Alle tagger består av en div med elementer som inputfelt for navn og en knapp for å fjerne taggen. Alle disse elementene får sine egne unike verdier basert på IDen til bildet. I tillegg blir det her lagt til funksjonalitet for å flytte og endre på størrelsen til taggen. jQueryUI pluginen til jQuery gjør dette veldig enkelt ved å bruke .draggable() og .resizable() som gjør div-elementet som boksen består av både flyttbar og skalerbar. I tillegg lar disse funksjonene deg definere et område boksen må holde deg innenfor, noe som gjør at du ikke kan flytte taggeboksene utenfor rammen til bildet.

Lukke-knappen til boksen fjerner elementet fra siden dersom den blir trykket, og input feltet er ment for å skrive inn hvem som befinner seg i boksen. Dette feltet er det som kobler bildet til en bruker i databasen og dersom bildet lastes opp med et tom inputfelt blir ikke taggeboksen lagret i databasen. En av kravene systemet er at brukere skal kunne se bilder av seg selv og sine fadderbarn. Dermed er det ikke nok å bare skrive inn et navn, det må legges til faktiske entiteter fra databasen. Dermed brukes AJAX til å kommunisere med serveren hver gang det skrives inn noe i dette feltet. Når noe her skrives, dersom feltet ikke er tomt, sendes et AJAX kall til serveren som returnerer en liste med brukere og fadderbarn som inneholder bokstavene i søket. Disse lastes inn som en liste under søkefeltet med jQueryUI Autocomplete sin _renderItem() funksjon som lar meg utforme en liste med elementer slik jeg vil og legge den til under søkefeltet. Denne listen viser brukerens navn, deres email og et profilbilde dersom det finnes. Dersom ingen bilde er lagt til vises et standard bildet i stedet.

Figur 11: Bildet viser søk med søkeord "ma"

Som på bildet over vises brukere og deres profilbilder for å enklest mulig kunne velge riktig person som befinner seg i boksen. Dersom det er et fadderbarn er det ikke lagt til en email.

Til disse elementene er det også tilknyttet en ID som er deres identifikasjon i databasen og hva slags type entiteten er (bruker eller fadderbarn). Når et element velges legges databaseIDen, navn og type til som attributter på boksen.

Søket må gå raskt, fordi brukere er ikke villige til å vente på lange søk i dette tilfellet så det har søket har, så godt det lar seg gjøre, blitt optimalisert på serversiden. Entity tilbyr en rekke metoder for å optimalisere innhenting av databaseinformasjon. Dette inkluderer for eksempel å definere alle relasjoner mellom tabeller i databasen ved å bruke virtual properties. Når relasjoner er virtual properties bruker rammeverket noe som heter lazy loading. Dette vil si at relasjonene til database elementet kun hentes dersom det blir brukt i viewet, og ikke før dette skjer, noe som medfører at søk etter elementer i databasen går raskere når ikke de relaterte elementene også må hentes. I tillegg trenger vi ikke å hente alle radene til et element dersom disse ikke brukes slik at dersom vi vet akkurat hvilke rader vi vil trenge kan vi be om å velge kun disse slik som vist nedenfor. [48]

```
List<UserObj> users= _context.Users
                .Where(p => p.Fname+" "+p.Lname.contains(name))
                .Select(x =>  new  UserObj
                    {
                        Name= x.Fname+" "+x.Lname,
                        Id = x.Id,
                         Type = "user"
                    }).ToList();
```

Slike optimaliseringer er brukt flere steder i prosjektet der det er hensiktsmessig, selv om ofte ønskes det å hente alle radene til et element.

Jeg må også nevne ansiktsdeteksjonsfunksjonen som brukes til å automatisk legge bokser rundt ansiktene i bildet. Denne funksjonen styres av en jQuery plugin for nettopp dette som heter Facedetection. Denne fungerer ved at du sender inn bildet og pluginen gir deg tilbake koordinatene til ansiktene ut fra den originale størrelsen til bildet. Her må koordinatene skaleres opp eller ned avhengig av om størrelsen på bildet som vises er større eller mindre enn den originale størrelsen. Funksjonen returnerer en x og y ratio mellom bildet som er vist

og orginalbildet, og denne verdien ganges så med koordinatene for å finne størrelsen på boksen som skal vises. Det er en del feildeteksjoner med denne funksjonen så jeg la også til en knapp som fjerner alle tags i bildet dersom ansiktsdeteksjonen legger til en mange feile bokser.

Bildet er nå tagget og det er klart til å lastes opp. Vi legger en eventlistener til opplastingen slik at fremdriften kan følges prosentvis med en fremdriftslinje. Dette gjøres ved å binde et event til AJAX sin xhr attribut som rapporterer fremdriften til AJAX requesten og denne verdien legges oppdaterer fremdriftslinjen som er animert med Bootstrapklasser. Mens bildet gjøres klat til å lastes opp sjekkes det først om Javascript sin FormData er støttet. Er det ikke det er nettleseren din for gammel og du får ikke lov til å laste opp før du har oppdatert til en nyere versjon (Dette gjelder typisk versjoner av Internett Explorer laveren enn versjon 7). Hver gang et bilde lastes opp legges IDen til bildet i en liste som vi sjekker før bildet sendes slik at bildet ikke sendes flere enn én gang. Bildet legges så til FormData objektet som skal sendes til serveren.

Beskrivelsesteksten må også legges til FormData objektet for koden går løs på alle taggeboksene. Først må den originale høyden og bredden til bildet finnes, for det er den som lagres i databasen. Bilde og boksene skal kunne vises i hvilken som helst størrelse uten at boksene flytter seg eller blir feil størrelse. Det eneste som skjer er at dersom bildet ikke skal vises i sin originale størrelse så må boksene skaleres ned eller opp for å kompensere for dette. Dette gjøres ved å finne skaleringskoeffesientene til bildene som er, for y-koordinatene, den viste høyden til bildet i nettleseren delt på orginalhøyden til bildet. Før en boks legges til FormData objektet deles da boksens verdier på skaleringskoeffesientene for å finne størrelsen på boksen tilsvarende originalstørrelsen på bildet. Boksen legges deretter i FormData objektet med disse verdiene, brukerIDen, navnet på brukeren og typen bruker før dette legges til AJAX- kallet og sendes til serveren. En tilbakemelding vises på skjermen basert på om opplastingen var en suksess eller ikke.

På serveren hentes filen fra request objektet og legges til i databasen. Tagboksene som nå er representert som et JSON objekt blir deserialisert og lagt omformet til databaseobjekter. Alle taggebokser legges til i sin egen databasetabell (se vedlegg 1) og tilknyttes bildet i databasen. Alle personene som er i tagboksene blir også direkte tilknyttet til bildet. Files blir lest inn gjennom en BinaryReader og blir lagret i databasen som en byte array.

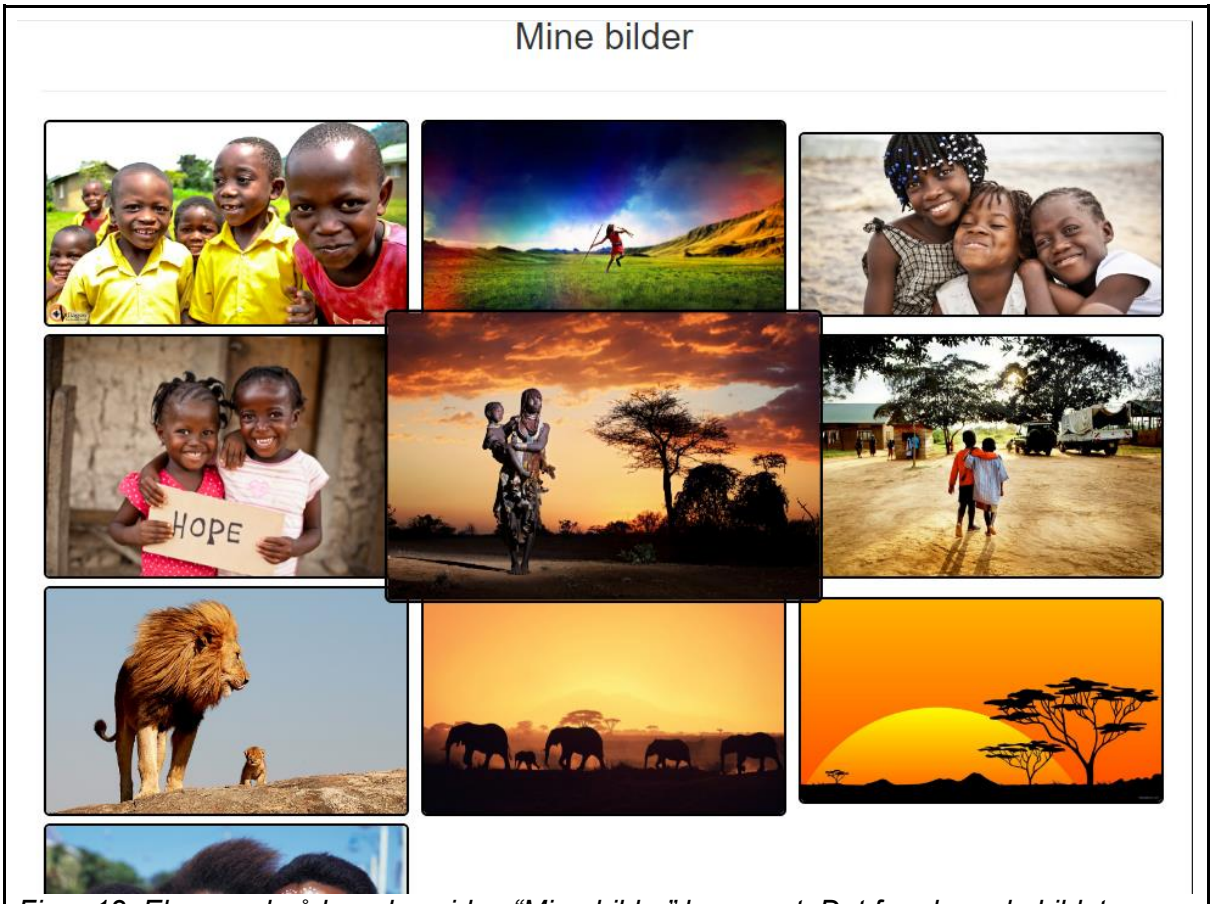
Dersom alt går etter planen returneres en melding i grønt som forteller brukeren om suksessen og dersom en exception skjer under lagring av bildet og bokser returneres exception-meldingen i rødt. Applikasjonen tillater at du laster opp et og et bilde dersom du ønsker det, etterhvert som du er ferdig å behandle hvert bilde, eller så har du også mulighet til å laste opp alle bildene samtidig når du er ferdig å behandle dem alle.

Jeg har også tatt høyde for skalering av nettleservinduet. Det oppstod opprinnelig et problem når brukeren endret på nettleservinduets størrelse og bokser ble flyttet rundt på bildet uten at dette var intensjonen. Dette løses ved å ikke bruke relative, men absolutte pikselverdier på koordinater for boksene og jeg hindrer også bilde i og skalere ned sammen med nettleservinduet. I tillegg har jeg valgt at bildene skal alle vises med en bredde på 700 piksler. Dersom bildet er mindre enn dette vil det skaleres opp på bekostning av resolusjonen. Dette er for å gjøre det enklere å behandle små bilder. Dette er ikke en funksjon for å vise frem bilder, men heller for å behandle dem slik at de kan vises senere.

Opphenting av bilder

Når et bilde er suksessfullt lagret i databasen må det også hentes opp igjen for visning. Dette skal altså bare skje for de som er tilknyttet dette bildet. Når en bruker logger inn vil en “Mine bilder” link dukke opp som man kan trykke på. Dersom bruker følger linken vil man komme til

en side som henter inn alle bildene fra databasen som brukeren er tilknyttet. Ikke medlemmer av siden vil ikke kunne se disse bildene, selv om de fikk en direkte link til den ettersom kontrollermetoden som henter bildene krever at du faktisk har tillatelse til å se bildet før bildet faktisk hentes.

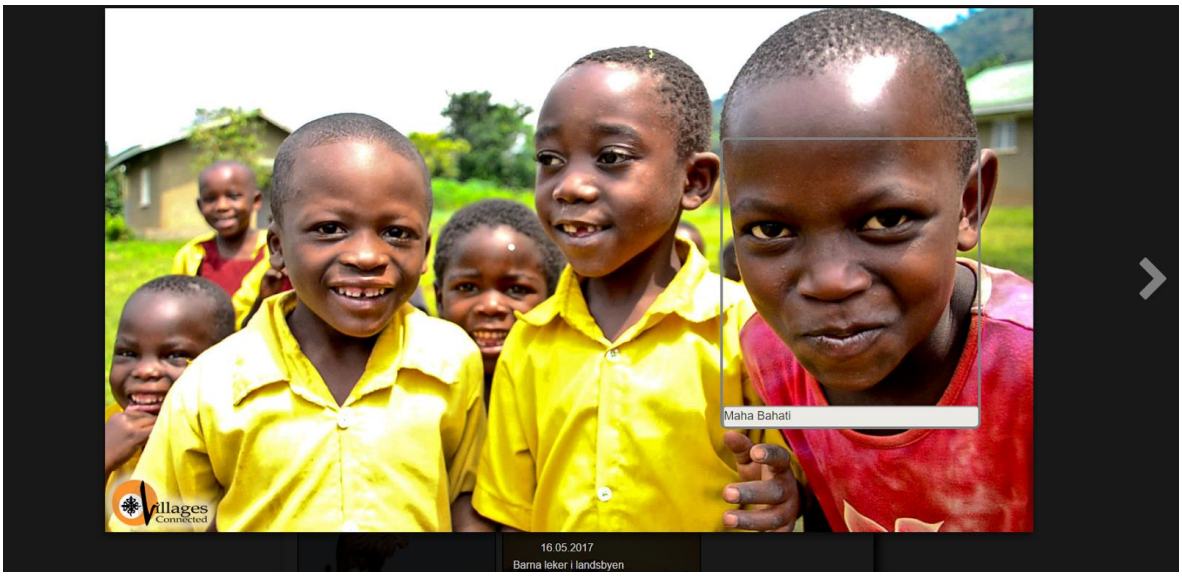


Figur 12: Eksempel på hvordan siden "Mine bilder" kan se ut. Det fremhevede bildet er som musepekeren er over.

Som vist på figur 12 hentes bildene fra databasen. Disse lastes inn etter tur ordnet fra de nyeste bildene til de eldste. Når brukeren går inn på siden returneres alle IDene til bildene som denne brukeren er assosiert med. BIldene i seg selv lastes først inn etter at resten av siden i sin helhet er lastet inn. Dette gjør at man slipper å vente dersom det er mange store bilder som skal lastes inn, og bildene vil dermed dukke opp etterhvert som de blir lastet inn etter tur. Samtidig så laster jeg ikke inn alle bildene til brukeren med en gang dersom det er flere enn fem av dem. Dette er også for at brukeren ikke skal bli overveldet med lange lastetider og serveren skal slippe å laste inn mange bilder som det ikke er behov for.

Lastingen skjer ved hjelp av AJAX og en jQuery plugin som heter WayPoint. Waypoint lar deg binde eventlistenere til brukerens posisjon på siden. Dersom brukere, slik som i dette tilfellet, blar ned til bunnen av siden vil et event bli avfyrt som forteller oss at brukeren har nådd bunnen av siden. Det som skjer da er at det gjøres et AJAX kall til serveren som henter 5 nye bilder, returnerer dem og legger dem til under bildene som allerede ligger der. slik fortsetter prosessen med å hele tiden hente eldre og eldre bilder etterhvert som brukeren blar nedover på siden helt til det ikke lenger kan hentes 5 nye bilder. Da hentes heller så mange som er igjen, eller ingen i det hele tatt dersom bruker er tom for bilder. Når bilder er i prosessen av å hentes vises også en CSS3 animasjon av et hjul som spinner rundt og indikerer at det arbeides. Dette er samme metode som nettsider som Instagram og Twitter bruker og det effektiviserer bilde-innlastning samtidig som det ser bra ut, og siden føles

raskere og mer responsiv av på grunn av den stegvise innlastingen. Her gjorde jeg et bevisst valg ved å ikke bruke paging, som er å dele bildene inn i sidetall, slik som jeg gjør de fleste andre steder i applikasjonen. Når oppgaven er å kun vise bildene til brukeren uten at de skal ha administrative muligheter som å redigere er dette en mer behagelig og brukervennlig måte å vise bildene på siden de lastes dynamisk inn etter etterspørsel, og alle bildene kan sees på en gang dersom man blar ned og dermed slipper man å klikke seg til neste side hver gang man vil se en ny seksjon med bilder.

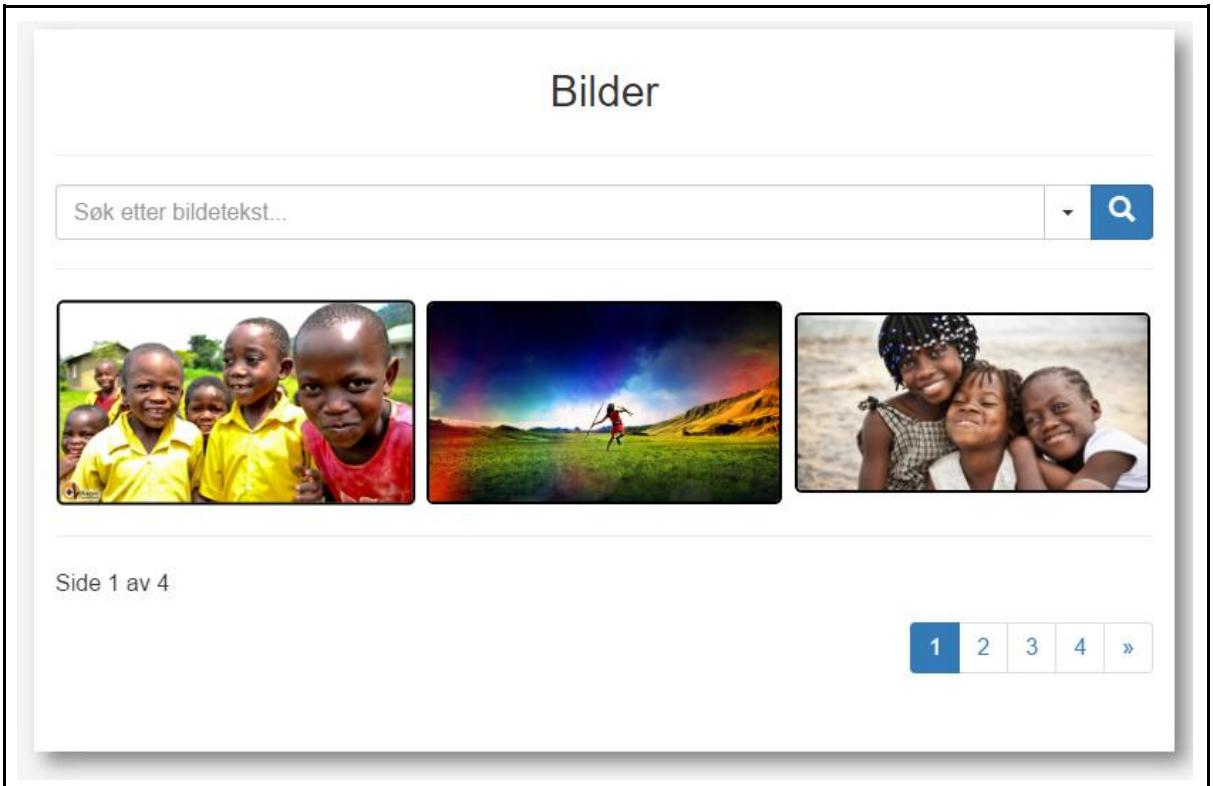


Figur 13: Eksempel på en modalboks for bildevisning

Dersom brukeren trykker på et av bildene popper det opp en modalboks med bildet i full størrelse, med mindre bildet er større enn 1400 piksler, som er maksverdien for visning av bilder ettersom over dette blir de plagsomt store ved at de tar en for stor del av skjermen. Som vist i figuren nedenfor vises her datoen bildet ble lagt inn i databasen sammen med en bildebeskrivelse dersom administrator har lagt til dette. Bildet vises så sammen med taggeboksene som kun vises dersom dersom du beveger musepekeren over ansiktene til personene i bildet. På den måten må du ikke se boksene som er i veien for resten av bildet, men dersom du vil kan du bruke musen se boksene og finne ut av hvem som er i bildet. Dette gjøres med CSS sin opacity attribute som settes til 1 slik at boksen vises når musen er over boksen, og 0 når den forlater området. Piler på siden indikerer at du kan gå til neste bildet om du ønsker. Dette kan man også bruke piltastene til og escape eller det å trykke utenfor boksen lukker boksen slik at siden igjen viser bildesamlingen som ligger under. Boksen åpnes også med en CSS3 zoom-in animasjon som gjør at bildene virker mer responsive og brukeren får tilbakemelding på sine handlinger. Alt i alt er systemet implementert for å være effektivt og brukervennlig med animasjoner, modalbokser og asynkron innlastning etter behov.

Administrering av eksisterende bilder

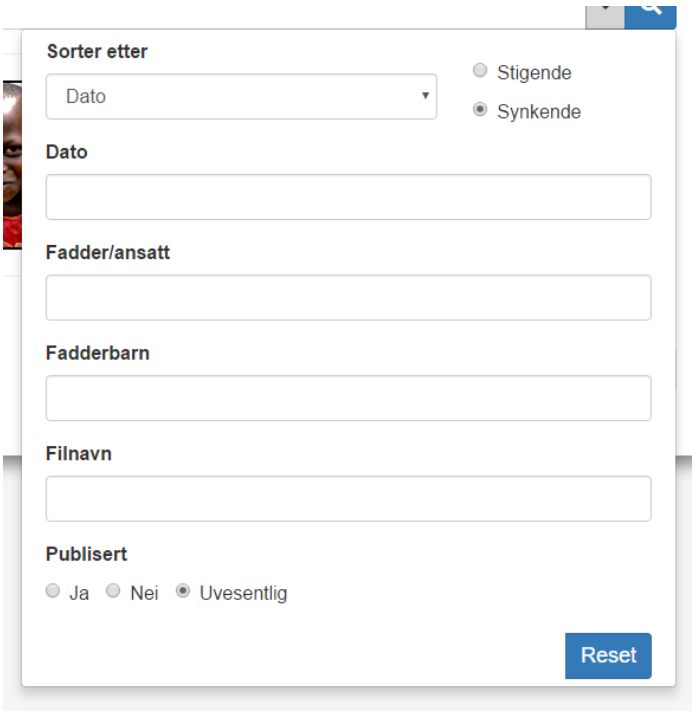
Systemet ville ikke vært spesielt brukervennlig og dynamisk dersom data ikke kunne endres eller slettes fra databasen slik at når et bilde ble lastet opp blir det for alltid liggende i databasen.



Figur 14: Administrasjonspanelet for allerede opplastede bilder

Det har derfor blitt laget et eget panel for nettopp dette. Her kan allerede eksisterende tags på bildene endres, fjernes og nye tags kan legges til, på lik linje med når de først ble lagt til, bare at nå vises bildene i en modalboks slik som på “mine bilder” siden. Modalboksen har nå også knapper for å oppdatere etter at du er ferdig med redigeringen. Det er også lagt til et tekstarea med den nåværende bildeteksten som kan endres om dette er ønskelig. I tillegg til oppdateringsknappen er det to andre knapper for å slette og gjemme bildet. Dersom du velger å slette bildet vil en en annen modal-boks vises som spør deg om du er sikker på dette. Trykker du ja sendes en AJAX request til serveren som sletter bildet og dets tilhørende tagebokser. Alt skjer med AJAX og bilde-seksjonen oppdateres automatisk når operasjonen er ferdig og viser bildene uten bildet som ble slettet. Den tredje knappen er en knapp som bestemmer hvorvidt bildet skal vises for brukere eller ikke. Denne styrer en database variabel som sier nettopp dette; om bildet er “aktivt” eller ikke. Dette er lagt til for å ha en måte å ikke vise bildet på nettsiden samtidig som man ikke skal måtte slette det helt fra databasen for å fjerne det fra brukersiden. Kanskje brukere har meldt fra om at de vil ha bildet fjernet fra siden og det kan dermed fjernes fra brukersidene intill en avgjørelse blir tatt.

I tillegg er det lagt til en søk og filtreringsmetode for å finne akkurat de bildene som bruker er ute etter ved å kunne legge til flere avanserte søkekriterier som alle tas med i databasesøket når bildene hentes fra serveren. Dersom pilen ved siden av søk knappen trykkes vises det en dropdown meny med avanserte søk og filtrering alternativer som vist i Figur 15.



Figur 15: Filtreringsalternativer for bilder

Her kan man finne bilder som inneholder spesifikke personer som faddere, ansatte og fadderbarn. Du kan også finne bilder som alle inneholder en bestemt fadder og fadderbarn ved å fylle ut både fadder- og fadderbarnfeltene. Søk kan også vises etter bildetekst, dato og andre felter, og kan også vises stigende eller synkende rekkefølge. Søket skjer, som omtrent alt annet i applikasjonen, med AJAX hvor alle filtreringsverdier postes til serveren og det er en filtreringsmetode på serveren som bruker disse verdiene fra filteringsskjemaet til databasesøket. En “pagedlist” med bilder basert på disse variablene som brukes i SQL forespørselen blir så hentet og returnert for så å oppdatere bilde seksjonen av siden med disse bildene. En pagedlist er et utdrag av hele listen basert på søket, dersom et søk er gjort. En paged list tar inn hvilket sidetall av listen som skal returneres og hvor mange elementer som inngår i hver side. Sidetall defineres derfor som en global variabel i applikasjonen som brukes til alle pagingmetodene. Listen kan deretter returneres og viser listen som tilsvarer den nåværende siden vi befinner oss på i pagingsystemet. Det er nuget pakken PageList.Mvc som introduerer pagedlist klassen og lar applikasjonen bruke denne til å dele opp lister i sidetall. Filtrering- og pagingmetoden som brukes her er også den som er brukt i resten av applikasjonen.

All innlasting og prosessering av data indikeres med et laste-symbol for å gi tilstrekkelig med tilbakemelding til brukeren. Skjer det noe galt under noen av database operasjonene returneres også unntaksmelingen og vises på siden uten at systemet krasjer. Det hele lager en veldig enkel, effektiv og intuitiv måte og jobbe med og administrere bildene med korte innlastningstider, avanserte søk og tilbakemeldinger på handlinger. Animasjoner brukes også på bildene for å fremheve og vise valgte bilder.

3.3.2 Medlemskap

Ett av kravene er at potensielle faddere skal kunne be om medlemskap. De skal ikke kunne registrere seg som en vanlig nettside, men de skal sende inn et skjema som blir håndtert og akseptert eller avvist av en administrator. Søknadsskjemaet består av en del personlig informasjon som adresse og annen kontaktinformasjon som skal lagres sammen med

brukeren i databasen og brukeren tvinges til å skrive litt om seg selv og hvorfor han vil bli medlem av organisasjonen Butterfly Friends. I tillegg sender bruker inn personnummer, dersom bruker vil oppgi dette, slik at organisasjonen kan gi skattefradrag dersom det bidras med 500 kroner eller mer. [39] I tillegg trengs samtykke fra personen angående bildeutleggelse, for ifølge datatilsynet må man i henhold til personvernloven alltid ha samtykke før man legger ut bilder på nett. [40] Dette løser jeg ved å introdusere bruker for brukervilkår som må samtykkes for at forespørselen skal gå gjennom. Det er ikke spesielt hensiktsmessig at jeg skriver brukervilkår for en side jeg ikke skal ha noe særlig med, så den smartere løsningen er her å lage en funksjon der sideadministrator selv kan laste opp en PDF med brukervilkår. Dette gjøres i admin-seksjonen av siden under et panel jeg kaller diverse. Dette panelet inneholder litt forskjellige funksjoner som ikke er store nok til å få sitt eget panel og da heller har blitt samlet sammen i ett som API nøkler, sosiale medier og også brukervilkår. Brukervilkårene, som må være av filtypen PDF, lastes her opp og lagres i databasen med PDF som filtype.Denne filen knyttes til en egen tabell som inneholder en relasjon til filen i filtabellen og en boolsk variabel som sier om funksjonaliteten er på eller ikke. Dersom brukervilkår ikke finnes fra før av i databasen lages en ny brukervilkår-rad, og dersom den finns overskrives fila i den eksisterende raden. Det vil aldri være mer enn ett brukervilkårdokument. Brukervilkårfunksjonen kan slås av eller på ettersom sideadministrator ønsker det. Kanskje ikke sideeier er interessert i å bruke bildefunksjonalitetene til siden, og da er det vel heller strengt tatt ikke nødvendig med brukervilkår. Uansett så er det en nødvendig funksjon å tilby, og sideadministrator kan tilføye seksjoner etter behov, og utforme og oppdatere brukervilkår når dette er ønskelig.

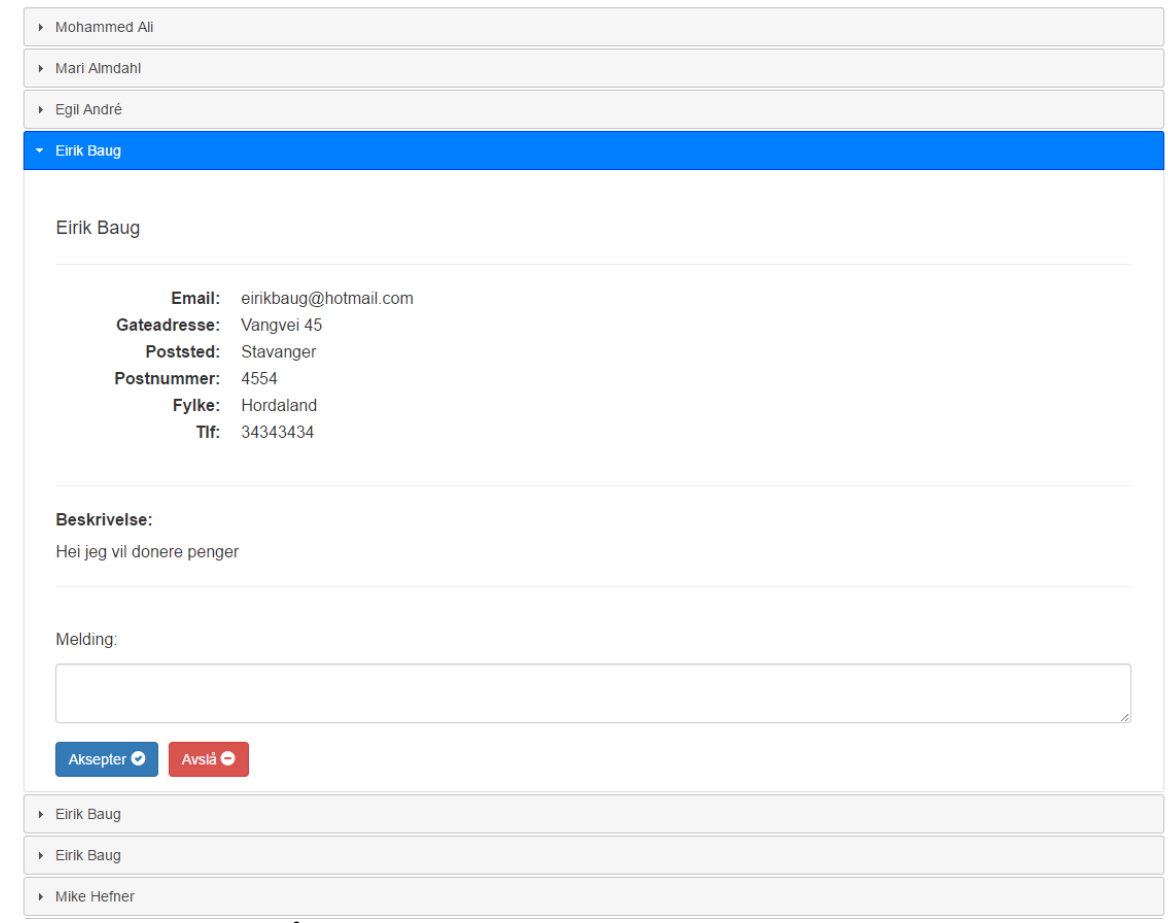
Bruker kan lese brukervilkårene direkte i nettleseren ved å trykke på linken som blir presentert ved siden av en sjekkboks der bruker erklærer at det samtykkes til brukervilkårene. Linken som fører til brukervilkårene returnerer klassen FileContentResult sine metode File. Denne metoden bruker filtypen og innholdet i filen til å vise filen direkte i nettleseren, uten at jeg må skrive noe mer kode for å vise frem dette. Samme metoden kan brukes for andre filer som filmer og videoer. PDFen åpnes altså i nettleserens innebygde PDF-leser, noe alle moderne nettlesere har i dag, der bruker enkelt kan bla gjennom og også skrive ut siden dersom det er nødvendig.

I tillegg til å godkjenne brukervilkårene må også bruker passere en test for å sjekke at det faktisk er et menneske som forespør og ikke en “bot” med dårlige intensjoner. Her kommer Google ReCaptcha APIet inn. Dette APIet verifiserer bruker ved å gi tester som, ideelt sett, kun mennesker kan bestå. Dette innebærer i utgangspunktet å huke av en checkbox, og google forsøker basert på denne handlingen å verifisere om dette ble utført av en bot eller ikke. [9] Etter at bruker har klikket om sjekken er ferdig blir en ny variabel kalt “g-Recaptcha-Response” lagt til i insendingsskjemaet. Denne variabelen er en token som på serversiden kan brukes til å verifisere bruker ved å gjøre et kall til reCapthca APIet med tokenen og den hemmelige nøkkelen til APIet, og dermed få svar fra google sine servere om brukeren bestod testen eller ikke. En egen ReCapthca klasse har blitt laget for å håndtere disse kallene og den går ut på at jeg bruker C# sin WebClient klasse til å laste ned en string fra en URL med token og hemmelig API nøkkel som parametere. Denne verdien brukes til å bestemme troverdigheten til brukeren.

Dersom denne testen passeres, bruker legger inn etterspurt informasjon og godtar brukervilkårene, blir skjemaet send inn med AJAX som omtrent alle andre skjemaer i applikasjonen, og deretter lagt til som en entitet i MembershipRequest tabellen i databasen. Dersom database innleggingen går etter planen og form-validasjonen går igjennom vil brukeren bli fortalt at han vil bli kontaktet så snart som mulig.

Nettsidens administratorer har så mulighet til å akseptere disse forespørslene i et eget panel i den administrative delen av nettsiden. Denne delen er lagt opp med paging på lik linje med

alle andre lister i administrasjonspanelene, og dette er vist på et trekkspillformat. Med dette mener jeg at siden viser en liste over medlemskapsforespørseler som i utgangspunktet viser navnene til de som sender forespørselen, men når elementet blir trykket utvides elementet og viser all informasjon som brukeren sendte inn. Jeg har valgt en hensiktsmessig 10 elementer per liste her for at innlastningen skal gå hensiktsmessig raskt og samtidig fylle opp mesteparten av skjermen med navn.



Figur 16: Eksempel på hvordan listen over forespørsler er strukturert

I tillegg til denne listen kan bruker søke etter spesielle forespørsler dersom det er nødvendig å finne en spesifikk forespørsel. Dette søket, i motsetning med alle andre søkefunksjoner i applikasjonen tilbyr ikke en filtreringsmetode. Filtreringsmetoder er relativt tidkrevende funksjonalitet å legge til på en god måte på grunn av alle variablene som må håndteres og sendes frem og tilbake under paging-prosessen slik at det virket ikke hensiktsmessig å legge det til her med begrenset tid til disposisjon. Panelet blir trolig å altfor stort ettersom forespørsler slettes etter behandling. Man kan likevel enkelt søke gjennom listen ved å gi et hvilket som helst søkeord i et enkelt søkefelt.

Administrator vil så velge å avslå eller akseptere denne forespørselen. Her vil han kunne skrive en melding i meldingsfeltet som vist på Figur 16. Når en søknad blir behandlet vil en mail sendes angående resultatet av denne løsningen som inneholder en standardmelding sammen med en ekstra melding dersom det er ønskelig å si noe utover at forespørselen er avvist eller godtatt. Til denne funksjonaliteten bruker jeg SendGrid, emailtjenesten brukt i dette prosjektet, til å sende mail til brukeren. Her brukes C# sin SmtpClien klasse til å koble til SendGrid sin smtp server, port 587. Meldingen er et C# MailMessage objekt der jeg setter emne, tekst; både HTML tekst og vanlig, samt mottakere. Denne mailen kan spores fra SendGrid sine nettsider der klienten vil bli nødt til å lage en bruker dersom de ønsker denne

tjenesten. Her kan man sjekke at mailen er suksessfullt levert og åpnet. På nettsiden varsles brukeren om at mailen er suksessfullt sendt over til SendGrid, og dersom sendingen mislykkes blir brukeren informert om dette også.

Når en bruker blir akseptert lages en bruker med denne brukerens personlige informasjon i databasen. Derimot blir ingen passord satt. Det virker ikke hensiktsmessig å la brukeren sende inn passord med forespørselen siden dette fort kan glemmes. Mailen vil derfor inneholde en link der brukeren kan sette sitt passord og på denne måten blir også Mailen godkjent som brukerens faktiske E-mail siden det krever at brukeren logger inn og følger linken. Identity Framework sin UserManager klasse som håndterer brukere i rammeverket har en UserTokenProvider metode som kan brukes til å lage passordresetting-tokens for den spesifikke brukeren. Disse tokenene kan lages selv om brukeren ikke har satt noe passord fra før. Kort sagt så må denne passordresettings-tokenen leveres til rammeverket sammen med IDen til brukeren for å få lov til å sette et passord. Derfor sendes en link sammen med mailen om at brukeren har blitt akseptert, som deretter fører til sidens passordsettingside. Parameterene i denne URLen er brukerens ID og passordresettingtokenen. Bruker IDen er en tilfeldig generert string på rundt 30 karakterer, etter å ha telt selv, mens jeg klarte ikke finne noen offisiell dokumentasjon på hvor lang passord-resettingstokenen er, men etter å ha tatt en titt på den ser den ut til å være vel over 200 karakterer. Disse to må matche når Identity sjekker dem opp mot hverandre under passordresettingen. Dersom brukeren har linken er det enkelt å sette sitt passord, mens det skal noe til for å få tak i denne linken av folk med dårlige intensjoner.

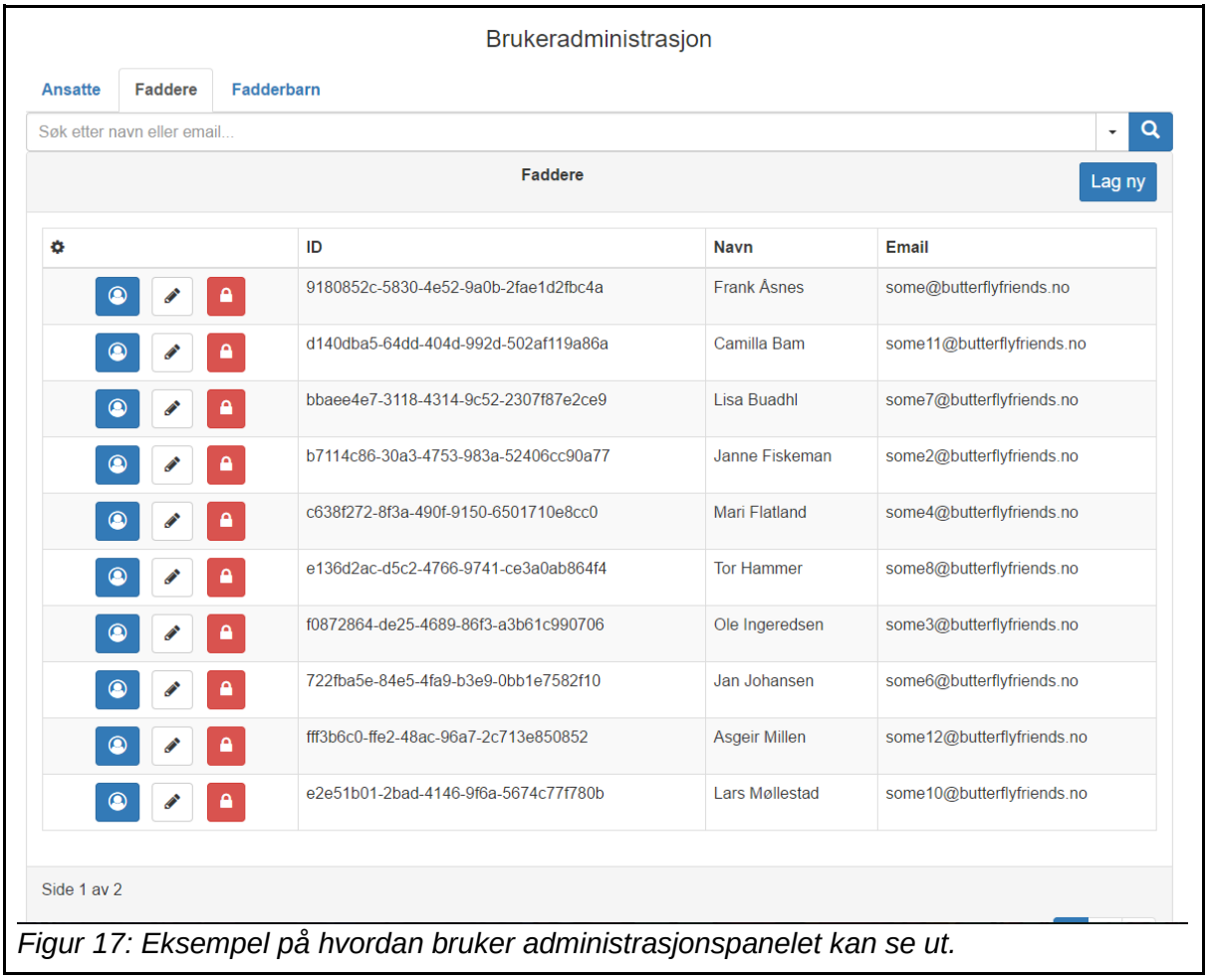
Linken ligger ikke engang direkte i mailen ettersom den blir kortet ned av SendGrip APIet når linken går innom deres servere før den redirekteres til passordresettingen. Dersom noen likevel skulle få tak i en link må de også vite hvilken email som er tilhørende linken for å logge inn siden dette ikke gjøres av seg selv.

Passordresettingstokens i Identity lever i 24 timer, og etter dette kan de ikke brukes lengre. Det vil si at linken ikke fungerer etter 24 timer. [51] Klarer derimot ikke bruker å sette sitt passord innen denne tiden kan de bruke sidens “Glemt passord?” funksjon som det ligger link til på innloggingssiden. Siden det allerede eksisterer en bruker i databasen fungerer denne funksjonen også uten å først måtte ha et fungerende passord. En lignende email sendes da til bruker slik at passordet har nye 24 timer til å settes.

3.3.3 Bruker- og fadderbarnadministrasjonsystem

Dette var altså en funksjon spesifikt etterspurt av prosjekteier og er et ganske omfattende panel for håndtering og filtrering av database-entiteter. Her var også brukervennlighet i stort fokus og det er implementert på en måte som gjør det enkelt å jobbe med entitetene. Dette panelet er først og fremst delt opp i 3 seksjoner, en for hver type entitet som skal håndteres. Her bruker jeg Bootstrap sine faner til å dele siden opp siden i 3 faner for håndtering av ansatte, faddere og fadderbarn. Dette er også et relativt sensitivt panel som noen kan gjøre mye skade på dersom noen begynner å slette alt. Derfor er panelet stengt av til kun eieren av siden og sideadministratorer. Vanlige ansatte er begrenset til blogg og bilder.

Nødvendigheten til dette panelet er klart. Man må ha en overordnet metode for å administrere medlemmer i en fadderorganisasjon på over 600 medlemmer. All informasjon om brukere må kunne endres og brukere må både kunne slettes og stenges ute.



Figur 17: Eksempel på hvordan bruker administrasjonspanelet kan se ut.

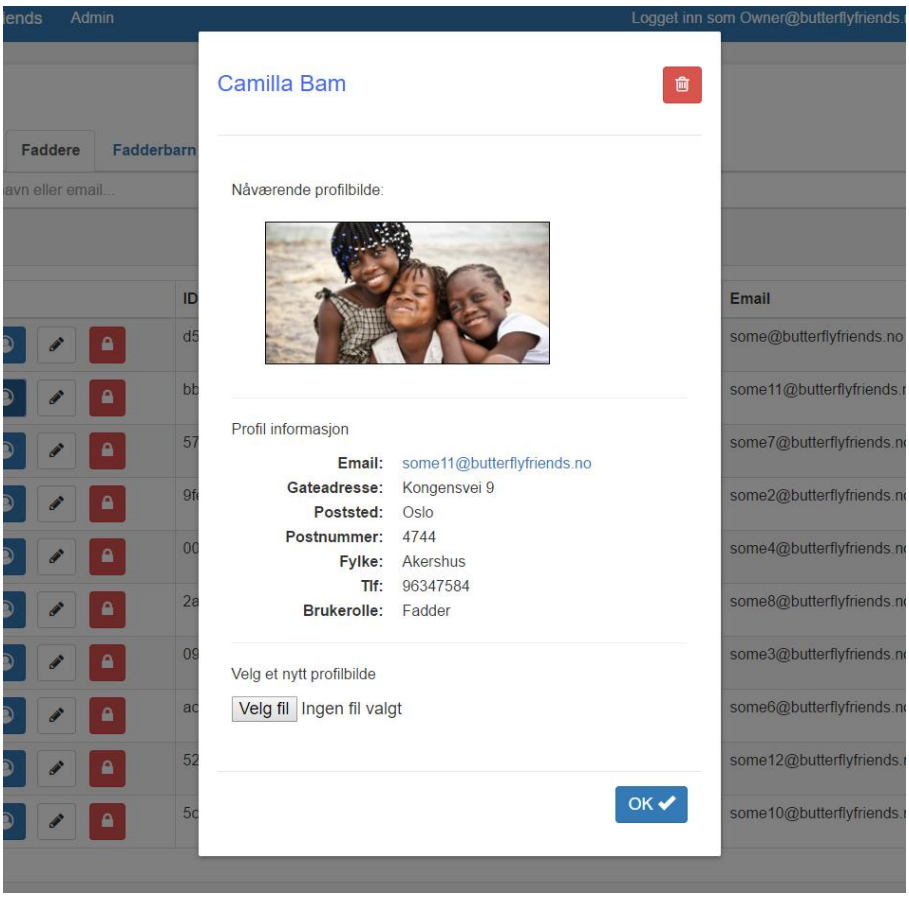
Som vist over er siden delt inn i tre seksjoner eller faner, som enkelt kan byttes mellom ved å trykke oppe til høyre. Alle sidene er komplette med avansert søk- og filtreringsmetoder, og paging slik som forklart i kapittel 3.3.1. Her kan man sortere på, for eksempel, etternavn eller vise alle brukere som har en spesifikk postkode, gateadresse, eller folk som har begge. For fadderbarn kan man også se alle fadderbarn som er født på spesifikke datoer siden det er relevant å holde orden på alderen til organisasjonens fadderbarn.

Siden er altså delt opp i 3 faner, og når siden opprinnelig lastes inn hentes 30 elementer, som tilsvarer 10 ansatte, 10 faddere og 10 fadderbarn. Etter dette oppdateres kun en liste av gangen, og det er den listen du jobber med for øyeblikket. For eksempel dersom du går til neste siden på fadderlisten er det kun den som prosesseres og oppdateres.

For å oppnå mest mulig brukervennlighet på dette panelet fungerer hele panelet på en side. Alle funksjonaliteten er presset inn på en side slik at når man filtrerer, søker og bytter sider i listene blir du værende på akkurat samme side. Ingen lange og tungvinte fullstendige sideinnlastninger eller behov for å gå til en ny side hver gang du skal se informasjon om en spesiell entitet. Det er dermed viktig, med så mye funksjonalitet på siden at ting ikke blir overveldende og kun relevant informasjon for øyeblikket vises.

Når en operasjon skjer mot serveren, som bytting av listeside eller henting av brukerinformasjon blir brukeren presentert med et innlasting symbol som spinner rundt og gir tilbakemelding til brukeren om at prosessen er undervei og brukeren må vente, på lik linje med alle andre AJAX operasjoner i applikasjonen. Siden oppdateres så der det er relevant med ny informasjon. Det hele er gjort veldig effektivt og brukervennlig med AJAX som

håndterer alle kall til serveren og oppdaterer der det er nødvendig, og bare der det er nødvendig.



Figur 18: Eksempel på en modalboks med brukerinformasjon.

Informasjonen som vises utover den mest grunnleggende informasjonen i listene blir fylt inn i relevante modalbokser som vises med en pen animasjon, og legger seg oppå listen med en "maske" mellom liste og boks for å sette boksen i fokus. Dette gjelder operasjoner for alle entiteter som å se informasjon, endre profilbilder, slette brukere, lage brukere og endre all informasjon om dem. Dersom man trykker på den blå brukerknappen som vist i figur 17 vil en boks som ligner figur 18 vises. Funksjonaliteten til disse boksene håndteres av klientrammeverket Vue og er laget og designet av meg. Dette er elementer av HTML som legger seg på en pen måte over innholdet som ligger bak. I en slik detaljboks, som vist i Figur 17 kan du også, i tillegg til å se gjeldende informasjon, laste opp profilbilde og laste dette opp asynkront mot databasen. Dette vil da automatisk finne det nåværende profilbildet, dersom det eksisterer og oppdatere dette med den nye dataen. Caching er skrudd av for denne funksjonen, for hvis ikke vil ikke nettleseren vise det oppdaterte bildet med en gang siden nettlesere lagrer allerede innlastede bilder i minnet slik at innlasting går raskere neste gang. Du kan også slette enheten herfra ved å klikke den røde søppelkassen. Dette vil bringe opp enda en modalboks som spør om du er sikker på dette. All annen informasjon om brukeren som email, passord og annen personlig informasjon kan også endres, men dette er blitt flyttet inn i sin egen modalboks som kan hentes frem ved å trykke på redigeringsymbolet ved siden av hver bruker, og viser alle databasevariabler og gir deg mulighet til å endre på disse. Funksjonaliteten har blitt flyttet litt rundt til logiske plasser for å ikke overvelde bruker.

Profilbildeopplastningen er vesentlig fordi her er det viktig å optimalisere innlastingen. Profilbildet brukes også som identifikator når vi bruker autofullfør til å søke etter brukere som

diskutert i kapittel 3.3.1. Det er derfor ikke gunstig å måtte laste inn et fullt størrelse bilde når søket skal være raskt og vise bildet på et veldig lite format i en liste. Dette leder meg opp til at bildet må skaleres ned når det lagres i databasen. Etter litt testing finner jeg ut at profilbildet trenger ikke være bredere eller høyere enn 300 piksler, som er passe stort for visning av denne typen bilde. Dersom dette er tilfellet skaleres bildet ned til å ikke overstige 300 piksler. Denne bildestørrelsen er fin for kun å skulle vise bildet så det er mulig å se hva det er i en informasjonsboks, men det er fortsatt for stort til å raskest mulig hente og vise bildet på et veldig lite format som i autofullfør søkeboksen. Derfor lagres det et ekstra bilde i tillegg til profilbilde som er en miniatyrversjon av profilbildet. Miniatyr versjoner av bilder, som ofte er på rundt 40 piksler i bredden, blir lagt i en egen tabell i databasen kalt “thumbnails”, og enhver thumbnail kobles til den større versjonen av bildet i databasen. En profilbildethumbnail lager også en relasjon direkte til brukeren den tilhører slik at når man søker på brukerens navn i et optimalisert søk kan man med en gang sjekke om det er et miniatyr profilbilde tilknyttet denne brukeren, og da også så returnere denne og vise i søket. En kort test viser at et stort bilde på 2MB i størrelse kan komprimeres ned til cirka 10 kb noe som dramatisk effektiviserer disse søkene.

For å skalere bilder samtidig som man beholder størrelsesforholdet krever det noen metoder. Første må man finne ut av er hvilken side man skal sette ned til minimum. For eksempel dersom bredden settes ned til 40 piksler må vi også finne størrelsesforholdet mellom bredde og høyde for å finne ut av hvor høy den nye høyden blir når forholdet mellom sidene bevares.

```
        private static Bitmap ResizeImage(Bitmap image, int width, int
height)
        {
            Bitmap resizedImage = new Bitmap(width, height);
            using (Graphics gfx = Graphics.FromImage(resizedImage))
            {
                gfx.DrawImage(image, new Rectangle(0, 0, width, height),
                new Rectangle(0, 0, image.Width,
image.Height),
                                                        GraphicsUnit.Pixel);
            }
            return resizedImage;
        }
```

Kodesnutten over viser en metode for å skalere ned eller opp et bilde. For det første tar den inn en Bitmap. Bilder i prosjektet lagres i bytearrays og ikke bitmaps siden slike er enklere å hente og vise i nettleseren, men i dette tilfellet trenger vi en bitmap siden denne klassen har mulighet for å hente høyde og bredde til bildet. Derfor må bildet først konverteres til en bitmap før det sendes inn til funksjonen.

```
var ProfileImageWidth = 40;
double ratio = (double) ((double) bmp.Width/(double) bmp.Height);
int height = (int) ((double) ProfileImageWidth/ratio);
```

Over er et eksempel på å først finne ration til bilde ved å dele bildets bredde på høyden og så finne den nye høyden i relasjon til den nye bredden ved å dele bredden som er bestemt til å settes ned til 40 piksler på størrelsesforholdet. Her er alle double-konverteringene

nødvendig for å tvinge frem en double-verdi i C# og ikke ende opp med en integerverdi som da blir 0. Disse verdiene må sendes inn til ResizeImage funksjonen som lager en ny, tom bitmap basert på høyde og bredde. Deretter brukes Graphics biblioteket til å tegne det gamle bildet over på den nye tomme bitmappen ved hjelp av bilde, rektangelområdet det skal tegnes inn på og rektangelområdet det blir hentet fra. Funksjonen returnerer så det behandlede bildet som etter dette igjen må konverteres til en bytearray som lagres i databasen.

Denne autofullfør optimaliserte søkefunksjonen brukes også i brukeradministrasjonspanelet ved å her gjøre det mulig å legge faddere til fadderbarn med denne funksjonen. Her fungerer den på samme måte som i bildeopplastningspanelet bare at den kun filtrerer faddere og ansatte siden det ikke er hensiktsmessig å legge fadderbarn til som faddere til andre fadderbarn. Foreløpig er det kun funksjonalitet for at et fadderbarn kan ha én fadder siden dette er alt som kreves for å betale for skolegang for dette fadderbarnet. Derimot kan en fadder ha så mange fadderbarn som er ønskelig. Alle fadderbarn tilhørende en bestemt bruker kan også filtreres ut og vises med filtreringsmetoden i fadderbarnpanelet.

Entiteter kan også legges til i databasen her. Det er slik organisasjonen for eksempel legger til en ansatt brukere som ikke blir medlem på samme måte som vanlige faddere. Dette fungerer på samme måte som resten av panelet ved at all informasjon fylles inn i en modalboks, og listen oppdateres automatisk med den nye brukeren. Dette er såklart dersom all validasjon av verdier går gjennom testene for å sjekke at verdiene ikke er ulovlige og feil format. Feilmeldinger blir vist om dette.
Panelet håndhever altså full kontroll over sidens brukere og fadderbarn. Det er helt vesentlig for en side med mange medlemmer å kunne endre alle databasevariabler til brukere ettersom behovet oppstår. Selv om det kanskje ikke alltid trengs å ende informasjon om faddere er det veldig nyttig å kunne endre email og passord når det er behov for dette, som for eksempel når en bruker mister sin gamle mail og trenger å bytte. Dette gjør systemet mindre statisk og mer dynamisk siden det ikke krever at brukeren lager noen ny bruker for å løse problemet, men heller bare kan ta kontakt med administrator.

I tillegg til all annen nevnt funksjonalitet har jeg lagt til en knapp for å deaktivere brukere. Denne knappen krever ingen modalboks-spørsmål om bekreftelse, for alt den gjør er å endre én variabel i databasen. Når den trykkes (vist som en rød lås i figur 17) endres entiteten til inaktiv eller aktiv alt ettersom knappen viser en lås som er åpen og grønn eller rød og lukket. Når en entitet i databasen er satt til inaktiv dukker den for eksempel ikke opp i søk når man skal tagge bilder og brukeren vil få en feilmelding om at brukeren er inaktiv dersom han prøver å logge inn på en inaktiv bruker. Dette er slik at brukeren kan stenges ute uten å måtte slette all informasjon om denne brukeren. Dette er nyttig dersom en bruker kanskje slutter å være fadder en stund, men kan komme tilbake uten at informasjonen er slettet. Sletting av brukere setter også en ganske innviklet prosess der, naturlig nok, brukeren slettes, men også thumbnails, profilbilder, taggebokser, og dersom brukeren som slettes er den eneste som er tagget i et bilde, blir også hele bildet slettet. Dermed er det greit og av og til bare kunne deaktivere denne brukeren og da heller filtrere ut og kun velge å se aktive brukere i administrasjonspanelet. Kun deaktiverte brukere kan også velges å filtreres ut dersom det er ønskelig. Når en bruker settes til inaktiv settes det også en variabel som forteller datoen brukeren ble deaktivert slik at administratorer i systemet for eksempel kan slette alle brukere som har vært aktiv i mer enn tre måneder.

Brukerhierarki

Nettsiden har et brukerhierarki for å håndheve brukeres rettigheter på forskjellige deler av siden. Dette begrenser også hvor en bruker kan bevege seg på siden og dersom han går til et panel han ikke har tillatelse til å se må han logge inn med en administrator bruker for å komme videre. I tillegg til dette har jeg i burkeadministrasjonspanelet lagt til funksjonalitet for

hvem som kan redigere hva. Det ville vært dumt hvis en administrator som hadde en veldig dårlig dag begynte å slette entiteter slik at ingen lenger kunne bruke siden. Derfor er hierarkiet bygget slik at man kun kan redigere informasjon til brukere som er lavere i hierarkiet enn en selv. Den eneste bruker på ditt hierarkinivå som du kan endre på er deg selv, siden det er rimelig at du kan administrere deg selv, selv om du ikke får lov til nedgradere eller deaktivere din egen bruker for dersom du gjør dette må du ha hjelp til å fikse det igjen av en bruker på et høyere nivå.

Hierarkiet er som følger:

Brukertype	**Funksjon**
Eier	Nettsidens eier, denne brukeren fins det bare én av og blir levert med den ferdiglagde siden til prosjekteier. Denne brukeren er på toppen av hierarkiet; ingen kan redigere denne brukeren unntatt den selv og den kan redigere alle.
Administrator	Dette er en type ansatt i organisasjonen som har flere rettigheter enn en vanlig ansatt. Denne har tilgang til alle paneler på siden. Det eneste denne bruker ikke kan gjøre er å utnevne nye administratorer eller endre administrator- og eierbrukere.
Ansatt	Vanlig ansatt i organisasjonen. Har tilgang til å skrive og håndtere blogginnlegg og bilder med tags.
Fadder	Laveste brukernivå og har ikke tilgang til noen deler av administrasjonspanelet. Men kan nå alle offentlige deler av siden og også se bilder av seg selv og sine fadderbarn.

Brukernivåene blir håndtert med en simpel variabel i databsen kalt “UserNr”. Denne variabelen sier hvor i hierarkiet du er. Den øverste brukeren, nettsidens eier, får verdien 0 og jo høyere brukertallet er jo lavere i hierarkiet er brukeren. Dersom det er behov for et større hierarki i fremtiden kan man fylle på med høyere tall etter behov. Jeg har valgt å fjerne redigeringsapper, deaktiveringsknapper, og sletteknapper dersom en bruker du ser detaljer om er på et nivå som gjør at du ikke har rett til å endre denne brukeren. Mangel på en knapp stopper derimot ikke folk som virkelig vil endre verdiene slik at på serversiden sammenlignes også rollenummeret til brukeren som endrer informasjonen med brukeren som blir endret. Dersom ingen tillatelse blir innvilget returneres en feilmelding om nettopp dette. Eieren av siden kan fremme ansatte til administratorer eller degradere administratorer til vanlige ansatte. Da får eller mister de også tilgang til de respektive sidene som er tilgjengelig for de forskjellige brukernivåene.

Innlasting

Tilbake litt til innlastingen av de 30 databaseelemntene som først vises når siden lastes inn. Dersom alle tilhørende modalbokser skal lastes inn samtidig ville dette være dårlig praksis. For alle 3 panelene til sammen blir det 93 modalbokser på en gang. Dette var faktisk mitt opprinnelige forsøk og det fungerte dårlig. Innlastingen ble treg og svært stygg med modalbokser som flimrer foran skjermen før de blir usynlige. Dermed løser jeg det på en helt annen måte: Når en knappen ved siden av et databaseelement trykkes går det et AJAX kall til serveren som returnerer et PartialView som er et HTML og Javascript segment som kan injiseres der det er behov på siden. Dette ParialViewet inneholder en modalboks og tilhørende Vue.js håndteringsfunksjoner for å håndtere funksjonaliteten. Dette kodesegmentet legges så til et bestemt tomt div-element på siden og er konfigurert slik at det

kjøres med en gang det blir injisert på siden og dermed vises redigeringsboksen med relevante variabler og funksjoner. Når bruker da er ferdig å redigere lagres informasjonen, boksen lukker seg, informasjonen sendes og oppdaterer databasen på serveren og returnerer en melding om suksess eller feil samtidig som den endrete entiten markeres i grønt for å markere suksess og rødt dersom noe gikk galt. Når bruker da igjen trykker på en annen knapp skjer det samme for denne brukeren også og den lukkede modalboksen som ligger i div-elementet som vi er ferdig med blir fjernet og erstattet med den nye modalboksen som nettopp er hentet fra serveren. På denne måten går innlastning mye raskere og ser mye bedre ut når vi bare laster inn kun den informasjonen vi ber om for øyeblikket, og fjerner den informasjonen vi ikke lenger ha bruk for å se. Dette gjør også at siden generelt kjører og føles mye bedre ut å bruke i og med at for mange modalbokser på en gang bruker opp minnet til datamaskinen ettersom alle krever en del eventhandlers og mer eller mindre innviklet kode for å håndtere variabler og kommunisere med serveren. Kun det som brukes av kode faktisk kjøres.

3.3.4 PR-administrasjonsystem

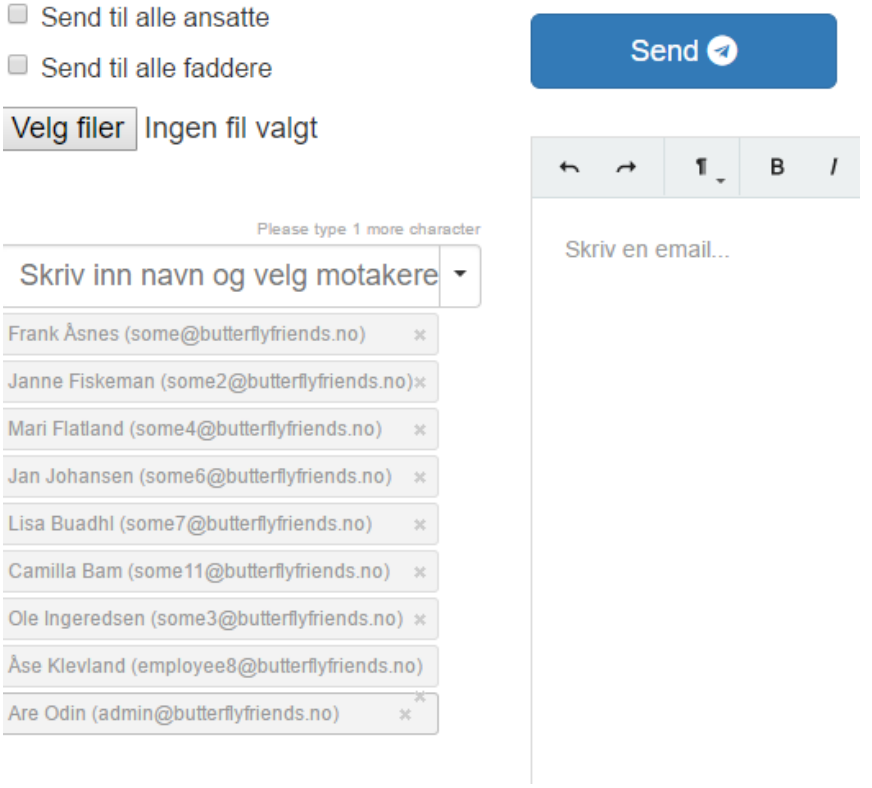
PR, eller Public Realtions systemet skal ta seg av kommunikasjon med faddere og omverden ellers.

Email

Email skal sendes til brukeren, så under PR-panelet har bruker mulighet til å gå videre til to forskjellige funksjoner; email og blogginnlegg. Emailfunksjonen består, klientsidemessig, av en teksteditor, mulighet for å velge hvem som skal motta emailen og mulighet for å laste opp vedlegg sammen med emailen. FormData objektet støtter alle mulige filtyper for samtidig opplastning, så dette er ikke vanskelig.

For å skrive emails har jeg valgt HTML teksteditoren Trumbowyg. Det er idiéelt om mailene som lages også kan styles slik at de blir seende profesjonelle for de som måtte motta dem. Derimot så støtter de aller færreste mail-tjenester eskterne og interne stylesheets for CSS. [41] Derfor må man ha inline CSS for å style mailen, noe Trumbowyg nettopp gjør. Mailen kan da legges til tabeller, overskrifter, youtube-videoer (dette er ikke støttet hos alle mailtjenester heller så den må brukes på egen risiko) og bilder. Bildene lastes opp til Imgur.com gjennom deres API. Dette gjør ting enklere for min del ettersom jeg slipper å laste opp og lagre bildene som sendes med mailene i databasen. Dersom de skulle lagres på serveren må man ta høyde for hvor lenge de bør ligge der før det er hensiktsmessig å slette dem igjen siden de bare blir liggende å ta plass. Mail er ikke i utgangspunktet ment for å vise frem bilder, det er bare en nyttig tilleggsfunksjon. Med imgur kan bildene enkelt bli lastet opp til APIet og da får jeg en link som kan legges til i emailen for å vise frem bildet i en image tag. Bilder på imgur ligger på deres servere så lenge Imgur ønsker det med mindre de blir bedt om å slettes. [42] Sletting kan derimot være vanskelig så det er viktig for ansatte i bedriften å tenke på hvilke bilder de legger til i mailene. Bilder, og alle andre filtyper, kan også legges til mailen som vedlegg.

Når bruker skal velge hvem emailen skal sendes til har de tre valg. De kan huke av boksen som sier “send til alle ansatte” eller “send til alle faddere”, eller skrive manuelt in emailadresser. Begge boksene kan hukes av om dette er ønskelig og da vil mailen gå ut til alle brukere i databasen som er aktive.



Figur 19: Eksempel på valg av mottakere

Denne siden har også en autofullfør funksjon som når du skriver inn navn eller email i mottakerfeltet som vist i figur 19 så vises tilsvarende entiteter i databasen med navn, email og profilbilde. Her brukes jQuery pluginen MagicSuggest til å filtrere vekk og vise resultater. Her sjekkes det at det ikke er dobbelt opp av like mailadresser, og søkekallet mot serveren går bare dersom det skrives inn 2 eller flere karakterer for å spare belastningen mot serveren. Denne pluginen, på lik linje med jQueryUI sin autofullfør funksjon har også en custom render metode som lar meg vise listen med brukere på det viset jeg selv ønsker, og kan legge til profilbilde-thumbnails for lettere identifikasjon av bruker. Når en bruker velges fra listen legges et element med navn og mailadresse horisontalt nedover etter hverandre som vist i figur 19. Det var her mest hensiktsmessig å bygge elementene nedover vertikalt på en side der det ikke er så mye plass til andre ting. Den horisontale listen kan fortsette nedover så langt det er ønskelig, og elementer kan også når som helst fjernes fra listen ved å trykke krysset på høyre side av elementet. Kodemessig legges emailadressen i en Javascript array når et element velges. Emailadresser som ikke ligger i databasen fra før av kan også legges til om det er ønskelig, men her brukes en regular expression til å sjekke at mailadressen er gyldig.

Når bruker så er klar for å sende mailen popper det opp en Vue modalboks der bruker må skrive inn emne til mailen. Emailen kan ikke sendes før emne er valgt. Når dette er gjort samles alle elementer av emailen; mailtekst og CSS, motakerliste, hvorvidt alle faddere eller ansatte skal få mailen og vedlegg i et FormData objekt og postes til serveren med AJAX der de puttes i sine respektive plasser i mailobjektet og filene blir lest inn og lagt til Attachment objektet til mailen før den sendes avgårde med SendGrid APIet.

Blogg- og nyhetsfunksjon

Denne funksjonen er for å legge ut blogg og nyheter på forsiden av nettsiden og dermed vise omverdenen hva organisasjonen jobber med. For dette målet har jeg funnet en god Javascript plugin som heter ContentTools som gjør det mulig å lage en editor som har en

veldig enkel “what you see is what you get” funksjonalitet og lar deg direkte editere paragraf-elementer slik som de vises i nettleseren uten at teksteditoren blir seende for mye ut som en klassisk teksteditor. ContentTools er den fineste, mest oversiktlige og best fungerende gratis teksteditor pluginen som jeg har funnet etter å ha sett gjennom flere enn 15 forskjellige. Trumbowyg kommer på andreplass og fungerer veldig fint for emailsending med sine inline CSS styles.

Du kan her enkelt se hvordan artikkelen blir seende ut når den legges ut på forsiden. Her har jeg tatt i en del inspirasjon fra den svært enkle, men effektive teksteditoren til Medium, slik som nevnt i kapittel 2.1.4, som også fint viser brukeren hvordan deres artikkel blir seende ut før publisering og tilbyr hensiktsmessig funksjonalitet samtidig som enkle men svært funksjonelle redskaper for redigering tilbys.

Figur 20: Eksempel på editering av en artikkel

Artikkelen er enkelt lagt opp som vist i Figur 20. På venstre side er det funksjonalitet for å publisere, slette, endre filnavn, lagre dokumentet og få tips. Publikasjon funksjonen bestemmer om artikkelen skal publiseres på forsiden av nettsiden eller ikke. Artikkelen må lagres minst en gang før den kan publiseres for det må være noe i databasen å faktisk publisere. Når dette skjer blir artikkelen stemplet med en tid og dato for når den først blir publisert og på forsiden blir artikler vist i den rekkefølgen de ble publisert i og ikke den rekkefølgen de ble lagret i. Dersom en artikkel blir endret i ettertid havner den derimot ikke på toppen av listen igjen som de ville om de ble sortert etter sist lagret dato. Publiseringen kan når som helst trekkes for å gjemme artikkelen fra forsiden. Slettefunksjonen er ganske klar på hva den gjør. Du må svare på at du er sikker på at du ønsker dette, og dersom du vil slette blir hele artikkelen og alle bilder som tilhører den slettes fra databasen. Bruker vil så bli dirigert tilbake til PR-siden når artikkelen slettes. Lagre fil funksjonen henter elementene artikkelen består av, samler dem sammen og lagrer den i databasen med et AJAX kall. En animasjon som flasher et ok eller ikke ok-tegn indikerer at lagringer var suksessfull eller ikke. Dersom den ikke er suksessfull blir en mer detaljert feilmelding vist. Tips funksjonen består av en dialog som gir brukeren tips på hvordan bruke editoren dersom det er første gang den blir brukt. Dette inkluderer tips som at dersom man trykker Crtl+S så lagres dokumentet. Er det dermed første gang dokumentet lagres vil en modalboks også poppe opp og be bruker gi artikkelen et navn. Navnet må ikke være unikt, det er bare en måte å identifisere artikkelen sammen med andre variabler når man skal finne den igjen senere, men det kan heller ikke være tomt.

Artikkelen har også en del funksjonalitet som kan aksesseres fra panelet som opprinnelig er plassert til høyre artikkelen, men denne toolbaren kan flyttes til der bruker måtte ønske å ha den. Den, sammen med alle andre knapper følger med vinduet etterhvert som bruker blar nedover. Toolbaren har funksjonalitet som å laste opp og legge til bilder i artikkelen, velge skriftstørrelse, legge til linker, videoer, tabeller og lister, samt midtstille eller høyrestille tekst. ContentTools baserer seg på egendefinerte CSS klasser som brukeren selv lager, noe som gir skriveverktøyet et unikt design basert på hvem som har laget CSSen. Når ContentTools starter laster den inn en del stiler som jeg har definert i en fil som heter editor.css der all CSSen for editoren er definert. Denne CSSen blir også hentet opp lagt til når artiklene skal vises på forsiden. ContentTools lar meg også definere en del stiler som bruker kan velge å bruke eller ikke når det skrives. Alt dette initialiseres under oppstart av editoren. Disse stilene kan legges til ved å klikke på tagene nede i venstre hjørne av editoren. Disse stilene er spesifikke til de forskjellige elementene. For eksempel kan man legge til forskjellige fonter og farger til paragraf-elementer, og andre stiler for andre elementer som lister og tabeller. Jeg har i min editor definert en del stile for paragrafer som font-stiler og mulighet for å legge til sitatseksjoner og separere tekst slik at artikkelen får litt forskjellige stilige elementer som gjør det behagelig å skrive og lese artikkelen. Nede i høyre hjørne kan man også se hvor mange ord man har skrevet.

Man velger helt å sette opp og designe dokumentet slik man selv ønsker, og min metode har gått på å være enkel og forståelig for en best mulig og behagelig skriveopplevelse med nok muligheter for å gjøre artiklene engasjerende og pene. Jeg har delt artikkelen inn i tre hovedregioner. Dette er tittel, ingress og hovedtekst. Når en bruker først lager et nytt dokument blir han møtt med et tomt dokument med tre seksjoner med “placeholder”-tekst som beskriver hva de er ment å representere før brukeren begynner å skrive. Tittelen står fast med største skriftstørrelse og kan ikke legges til så mange stiler unntatt funksjonalitet som å midtstille teksten og legge til linker. Under tittelen kommer en ingress seksjon som også står fast med en litt mindre skriftstørrelse. Her er det meningen å oppsummere innholdet slik at ingressen kan vises på forsiden og lokke lesere til å lese. Disse to elementene står fast og kan ikke slettes. Den mest modulære elementet er resten av teksten der paragrafer kan legges til og lettes helt etter behov. Når dokumentet initialiseres legges det til en liten seksjon øverst i hovedteksten som indikerer hvem som er forfatter av denne artikkelen og hvilken dato den ble skrevet på. Denne seksjonen er bare et forslag, og kan endres eller slettes etter behov. Etterhvert som flere brukere jobber på samme artikkel vil alle disse brukerne legges til som forfattere av artikkelen i databasen, og det er opp til disse å bestemme hvem som skal krediteres som forfattere.

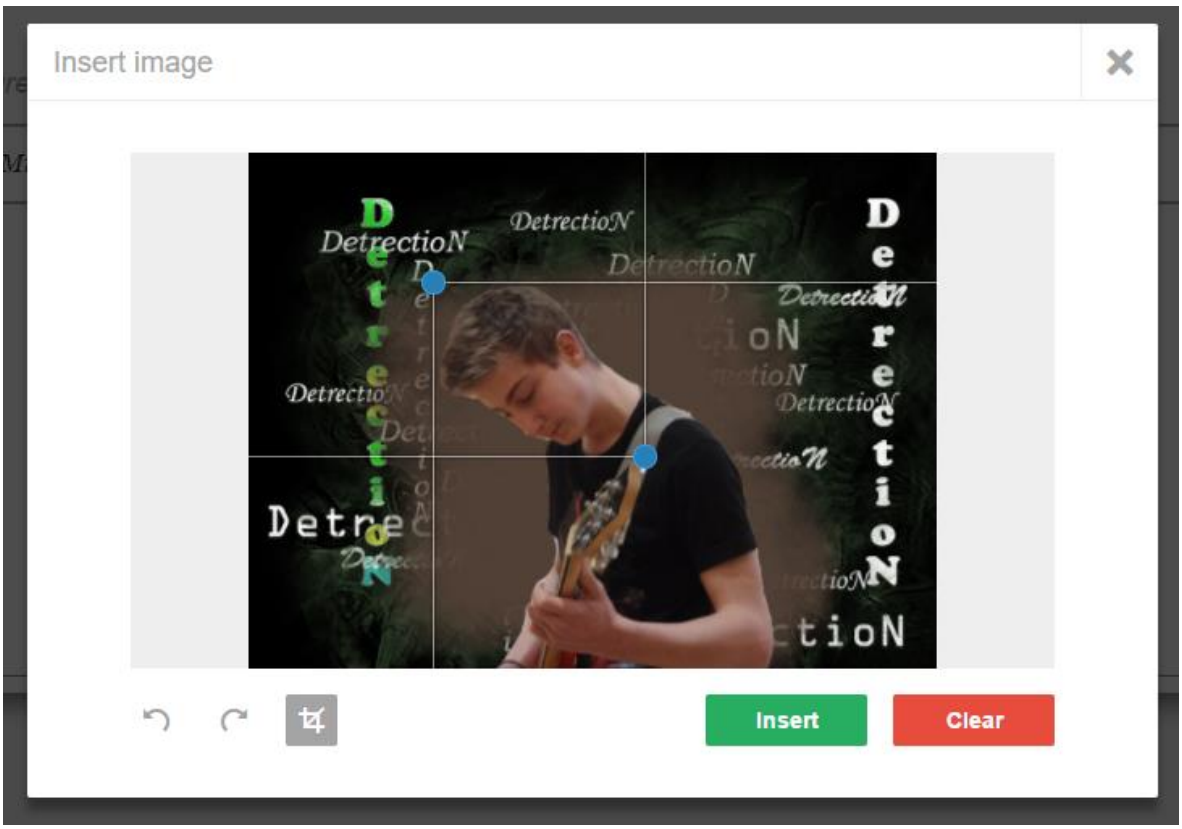
Når artikkelen skal lagres samles all tekst og HTML i disse forskjellige regionene før de blir gitt hvert sitt navn og postet til serveren. På serveren lagres de i sine respektive databasevariabler siden det er variabler for tittel, ingress og vanlig tekst. I tillegg er det egne variabler for å lagre tittel og ingress uten HTML slik at det på renest mulig måte kan vises på forsiden som en forhåndsvisning. Editoren har ikke kjempe mye funksjonalitet, men den funksjonaliteten som er der er enkel å bruke og gir kraftfulle verktøy for å gjøre artikkelen pen og leselig uten å tilby forvirrende mye funksjonalitet. Det må også nevnes at ContentTools har funksjonalitet for å flytte bilder og videoer rundt på siden etter behov. Et bilde bil kan for eksempel når som helst skaleres opp eller ned og flyttes rundt på siden der det måtte passe å ha den. Den kan legges til slik at den bryter teksten i en paragraf eller den kan legges pent på siden av teksten slik at teksten beveger seg rundt bildet, alt for gjøre dokumentet pent og leselig.

Bildeopplastning

Bildeopplastningen i dokumentet er en ganske omfattende prosess for håndtering av artikkelens bilder. ContentTools tilbyr en del events man kan lytte til for å håndtere bilder når forskjellige hendelser skjer, men selve bildehåndteringskoden må skrives på egenhånd.

Når en bruker laster opp et bilde og legger det til dokumentet lastes dette bildet opp, igjen med AJAX, til serveren der bildet lagres på serveren som et vanlig bilde. På samme måte som bilder ellers i databasen lagres. Det blir også tatt høyde for at bilde ikke må være for stort. Maksimumbredden til en artikkel er satt til 800 piksler slik at det er ikke noe poeng i å lagre bilder som er større enn dette. Dermed blir bildet konvtertert til en bitmap der vi kan sjekke om bildet er bredere enn 800 piksler. Dersom dette er tilfellet blir bildet skalert ned basert på størrelsesforholdet slik som beskrevet i kapittel 3.3.3. Deretter returneres et JSON objekt med størrelsen, bredde og høyde, i en string der verdiene er separert med komma. Bildet som legges til dokumentet vises med en default bredde på 600 piksler og bildene har funksjonalitet for å skaleres opp eller ned i dokumentet etter behov, og derfor er det viktig og også returnere makstørrelsen på bildet slik at ikke bildet blir skalert opp større enn det resolusjonen til originalbildet tillater. Dersom bildets makstørrelse er mindre enn 600 piksler vises bildets maksimale størrelse automatisk i dokumentet. URLen til bildet og IDen blir også returnert hvor vi på klientsiden fyller dette bildet inn i dialogboksen for bildeopplastning og redigering ved å legge til URLen til det nylige opplastede bildet og splitte strengen med størrelsesverdier opp i en array slik at verdiene enkelt kan bli aksessert. Redigeringen er derimot ikke ferdig her for bilde skal kunne videre redigeres i dialogboksen der det opplastede bildet nå er vist.

Bilde vil så kunne beskjæres og roteres etter brukers ønske. Når bildet roteres sendes bildes ID og hvilken retning dette bildet skal roteres til serveren. Dersom alt går etter planen og systemet finner bildet (ellers vil feilmeldinger bli returnert) henter jeg bildet fra databasen og igjen konverterer det til en bitmap for at den skal være enkel å jobbe med. Bitmaps i C# har en metode som heter “RotateFlip” som gjør det mulig å rotere bildet. Basert på om rotasjonsretningen er i klokkeretning eller mot klokkeretning roterer jeg bildet 90 eller 270 grader for å simulere denne handlingen, konverterer bitmapen tilbake til en byte array og lagrer den nye roterte bildet ved å overskrive verdiene til det gamle bildet. Deretter returneres igjen bildets id, URL og de nye høyde- og breddeverdiene. På klientsiden fyller vi igjen bildebehandlingsdialogen med det nylig behandlede bildet. Siden URLen til det gamle og det nye bildet er i utgangspunktet akkurat det samme vil ikke bilde endre seg siden nettleseren allerede har cached det gamle bildet med samme URL av effektiviseringsgrunner slik at når du bruker URLen igjen vil bare det samme bildet vises selv om det ikke er riktig. Derfor legges en nytt parameter på URLen som er '&_ignore=' + Date.now() som gir oss en unik dato for akkurat det sekundet. Når vi legger på en nytt parameter, selv om det i dette tilfellet ikke gjør noen ting så henter nettleseren bildet på nytt siden den ikke kan vite om parameteret betyr noe eller ikke.
I tillegg til rotering skal bildet kunne beskjæres for å velge ut den mest interessante delen av bildet. ContentTools har et fint grensesnitt for å velge ut hvilken region du vil ha og har en metode som heter dialog.cropRegion(). Dersom det er en region av bildet som er lagt til for beskjæring når bildet lagres, altså at når bruker trykker på “insert” for å legge bilde til i dokumentet så legges den beskjærte regionen til i skjemaet som så sendes til serveren. Beskjæringsregionen er en streng med 4 tall separert med komma. Når strengen er “0,0,1,1” så er bildet i sitt opprinnelige format og skal ikke beskjæres. Strengen “0,0,1,1” representerer verdier for topp, venstre, bunn og høyre. Disse representerer forhold på hvordan hjørnene oppe til venstre og nede til høyre i forhold til de originale verdiene. Verdiene kalkuleres ut fra diagonalpunktene i beskjæringsboksen som vist på figur 21 under.



Figur 21: Eksempel på hvordan beskjæring gjøres på klientsiden.

disse forholdene må så ganges opp med verdiene til den fulle størrelsen av bildet. Altså er oppe til venstre koordinat (0,0) og nede til høyre (1,1). Dersom det er gitt en beskjæringsregion vil for eksempel oppe til venstre nå være (0.546,0.745) som er forholdet mellom de originale koordinatene og de nye. Derfor må beskjæringskoordinatene ganges opp med høyden og bredden av bildet, der bildets høyde for eksempel ganges opp med y-koordinaten til beskjæringspunktet oppe til venstre for å finne den nye startkoordinaten til bildet.

```
var x1 = (int) (bmp.Width*cropList[1]);
var y1 = (int) (bmp.Height*cropList[0]);
var y2 = (int) (bmp.Height*cropList[2]);
var x2 = (int) (bmp.Width*cropList[3]);
var newWidth = x2 - x1;
var newHeight = y2 - y1;
```

Over vises et kodeeksempel der de nye startverdiene til bildet bil funnet og konvertert til integerverdier siden piksler ikke kan være doubles. Den nye høyden og bredden til bildet må også regnes ut basert på koordinatene. Deretter bruker jeg en metode for å beskjære bildet som minner veldig mye om scaleringsfunksjonen i kapittel 3.3.3. Her tegner jeg også bildet over på en ny bitmap ved å definere et rektangel (beskjæringsregionen) og tegner så bildet over på denne nye firkanten, bare at bildet ikke blir skalert ned denne gangen for å passe inn i firkanten og heller blir tegnet rett over som fører til et bilde der kun det som ble tegnet inn i

firkanten vises, altså beskjæringsboksen. Bildet lagres så i databasen for så å bli sendt tilbake til klienten der bildet legges til i dokumentet.

Foreldreløse bilder

Med en applikasjon som dette kan foreldreløse bilder fort bli et problem. Foreldreløse bilder er bilder som ikke tilhører noen artikkel, men som likevel bare ligger i databasen og tar opp plass. Dette kan for eksempel skje dersom brukeren laster opp et bilde, bildet vises i bildebehandling dialogen, men så bestemmer bruker seg for at han ikke vil bruke bilde lenger og fjerner det fra dialogboksen. Da er bilde borte fra dokumentet, men det ligger fortsatt i databasen uten at det brukes til noe. ContentTools tilbyr noen eventlisteners for å håndtere slike situasjoner. For eksempel når bruker trykker clear for å fjerne et bilde trigger det et event som gjør at en request kan sendes til serveren med bildets ID der bildet slettes. Dette løser derimot ikke alle problemene, for det er flere scenarier der for eksempel bildet legges til dokumentet, men så visker bruker det ut igjen. Elementet fjernes da fra dokumentet, men det er ingen event som kan fortelle oss om dette. Dette løses ved at når vi lagrer et dokument kjører vi også en funksjon som henter alle bildene i det dokumentet, legger bildeIDene inn i en liste og poster denne til serveren sammen med resten av artikkelen. På servere lager vi så relasjoner mellom den lagrede artikkelen og disse bildene. En artikkel er koblet til 0 eller mange bilder og disse bildene kobles altså til artiklen de befinner seg i. Første gangen dette skjer er det ingen bilder tilhørende denne artikkelen så alle bildeneIDene legges enkelt å greit inn i tabellen og lager relasjonene. Derimot når dette skjer neste gang må bildene sammenlignes. Listen som sendes inn og listen over relaterte bilder i databasen skal være den samme slik at disse listene må sammenlignes. Dersom de er like skjer ingenting, dersom det er elementer i den nye listen som ikke finnes i listen i databasen legges disse til i listen og dersom det er elementer i listen i databasen som ikke ligger i listen som ble postet til serveren så må disse bildene hentes fra og slettes siden dette betyr at disse bildene ikke lenger befinner seg i dokumentet og er dermed foreldreløse. Entity Framework har en god RemoveRange metode som gjør det mulig å enkelt slette en gruppe databaseentiteter.

Dette løser en del problemer, men det løser ikke problemet der bildet legges til dokumentet, men dokumentet lagres aldri. Da er det et bilde som er lastet opp i databasen, men siden dokumentet aldri ble lagret etter at bildet ble lagt til så ble det aldri knyttet til artikkelen og når artikkelen hentes opp igjen vil bilde være borte, men det vil fortsatt eksistere i databasen. Mye kan skje når man skriver en artikkel. Kanskje strømmer går, kanskje brukeren lukker fanen med et uhell, eller kanskje brukeren er misfornøyd med endringene å ikke ønsker å lagre. Jeg har prøvd å ta høyde for fane lukkingen ved at dersom brukeren lukker fanen med artikkelen bil nettleseren først prompte brukeren og si at ikke lagrede endringer kanskje går tapt dersom du lukker fanen. Dette er heller ingen ultimat løsning så det jeg gjør er å markere ikke lagrede bilder med en database variabel kalt “temporary”. Når temporary er satt til true i databasen så er bildet ikke tilknyttet noen artikkel. Dette er tilfellet når bildet akkurat er lagt til uten at artikkelen er lagret. Hvis bruker nå lukker dokumentet uten å lagre vil bilde bli liggende i databasen foreldreløst, men nå er det markert som foreldreløst og det kan slettes senere. Koden bør dermed jevnlig gå gjennom alle bildene og slette de som er markert som midlertidige bilder. Dette kan gjøres gjennom å trykke på en knapp i administrasjonspanelet for å starte prosessen, men dette er ikke en spesielt profesjonell løsning. Beste løsningen er å bruke “scheduled tasks” som kjører en gang iblant, kanskje en gang i uken og rensker opp i systemet. Dette er ikke en smart å kjøre i selve kildekoden, men bør heller håndteres av hostingtjenesten som nettsiden er hosted på. Azure har for eksempel støtte for schedulers som holder ved like systemet og kjører planlagte oppgaver. [43] Jeg har ikke tilgang til hostingtjenesten til oppdragsgiver så jeg får ikke satt i gang den planlagte koden, men jeg har skrevet en metode som går gjennom og sletter midlertidige bildefiler så alt som trengs er å sette denne opp til å kjøre i faste intervaller.

Artikkelgjennopphenting

Når en artikkel er lagret skal den også kunne hentes opp igjen. Det skal ikke være slik at artikkelen må skrives ferdig i ett kjør. Istedenfor skal det være mulig å lagre en artikkel for å gjøre noe helt annet, for så igjen å finne artikkelen i databasen og fortsette å skrive. Når en bruker først klikker seg inn på PR-panelet, vil panelet være delt i to der en side viser artikler som ligger i databasen, opprinnelig sortert fra sist lagret til først. På den andre siden kan man velge å skrive mail eller lage en ny artikkel. Artikkellisten fungerer som som mange andre lister i dette prosjektet og er delt inn i sider med AJAX paging og listen inkluderer også et søkefelt på toppen med avanserte søkealternativer dersom man for eksempel kun vil vise publiserte artikler eller kun vise artikler av en bestemt forfatter. Her fremheves også artiklene med litt CSS3 animasjoner for å vise hvilken artikkel som er i fokus samtidig som den viser grunnleggende informasjon om artikkelen som sist lagret dato og først publisert, i tillegg til tittel og navn. Det er også verdt å nevne at sist lagret variabelen vises også på bunnen av artikkelen slik at lesere kan se når artikkelen sist var modifisert.

Dersom en bruker trykker på en artikkel henter jeg denne artikkelen fra serveren og fyller inn de forskjellige regionene i artikkelen med sine respektive database verdier. Dermed blir det akkurat som at man hentet opp akkurat samme dokumentfila og kan fortsette å skrive å redigere på samme måte som når man lagde en helt ny artikkel.

3.3.5 Donasjonsystem

Siste store system som har blitt implementert i applikasjonen er et donasjon system slik at organisasjonen kan ta imot engangs- og månedlige donasjoner. Dette er, som diskutert tidligligere i kapittel 3.2 implementert med betalingstjenesten Stripe.

Som diskutert i kapittel 2 tar jeg inspirasjon fra Redd Barna sine nettsider. Når du først går inn på nettsidens forside vil du bli møtt av bilder fra organisasjonens arbeid dersom disse er lagt til i administrasjonspanelet. Dersom brukeren ikke er logget inn vil det i tillegg til donasjoner også være et segment som spør om du ønsker å bli medlem av siden og som deretter tar deg videre til et innmeldingsskjema dersom det er ønskelig. Dersom bruker er logget inn vises ikke medlemskapsfunksjonen. Her, som på Redd Barnas nettsider, forsøker jeg å skape følelser og engasjement for saken ved å vise bilder på en pen og hensiktsmessig måte (jeg vil diskutere bildefunksjonen i neste delkapittel) samtidig som jeg gjør det mulig for bruker å veldig enkelt og brukervennlig donere penger dersom de ønsker det.

Donasjonsfeltet er delt opp i to Bootstrap-faner. En fane for engangs donasjoner og en annen fane for faddere som ønsker å betale månedlig til organisasjonen. Donasjonsprosessen er delt opp i tre deler: Først en del der du blir spurt om hvor mye du vil donere, så en del der du blir spurt om informasjon om deg selv og hvorfor du donerer og til sist en del der du oppgir kortinformasjon og betaler. Disse delene kan bruker bevege seg enkelt mellom ved å skrive inn verdiene de vil oppgi for så å trykke neste. Dette vil trigge en Fadeout-fadein animasjon som på en pen måte fader ut innholdet i donasjon segmentet og erstatter det med neste trinn i prosessen. Hvilket trinn i donasjonsprosessen du er på blir også vist slik at bruker ikke tenker at dette blir for mye arbeid. Bruker kan gå frem og tilbake mellom disse segmentene av donasjonsprosessen med frem- og tilbakeknappene som tar deg mellom segmentene ved å fade-in/fade-ut innhold.

Etter at bruker har valgt hvor mye de vil donere en gang eller månedlig må de gi opp informasjon om seg selv. Dette er helt frivillig og bruker kan være anonym dersom dette er ønskelig. Dersom bruker er logget inn kan man også huke av for at donasjonen skal linkes til deres brukerkonto og slipper dermed å skrive inn noe. Dersom brukeren ikke er et medlem av siden og heller ikke vil være anonym kan han skrive inn sin informasjon manuelt. Dette er nyttig dersom du ønsker kvittering på epost, eller at organisasjonen skal ha mulighet til å kontakte deg. Du kan også oppgi personnummer her dersom du skal donere 500 kroner eller over og ønsker skattefradrag.[39] Her er det en del skjemavalidasjon som kjøres for å sjekke

at alt blir riktig. Det er viktig når man jobber med betalinger. For eksempel må donasjonsbeløp være et tall og over 3 kroner. Stripe tar et behandligsgebyr på 2.4% + 2 kroner så det er ingen poeng i å donere 2 kroner eller under. Felt som personnummer må også være tall og 11 siffer langt og mailadresser må være gyldige adresser. Brukere kan her også skrive en donasjonsmelding som for eksempel kan inneholde hva brukeren ønsker at donasjonen skal bli brukt til.

Figur 22: Informasjon om donasjonen.

Siste steg i donasjonsprosessen er å gi kortinformasjon. Her er det viktig å nevne at kortinformasjonen ikke behandles av organisasjonen, men utelukkende av betalingstjenesten stripe. Ingen kortinformasjon blir postet til serveren. Dette blir forklart til brukeren med en enkel melding som linker til Stripe sine nettsider. På dette punktet skriver bruker inn sin kortinformasjon som er kortnummer, CVC og utløpsdato. utløpsdato skrives i feltet på formatet MM/YY. Denne strengen må deles opp for Stripe APIet må ha variablene måned og år hver for seg slik at strengen må splittes på “/” og 20 må legges til YY variabelen for å få det eksakte årstallet.

For å håndtere kortinformasjon brukers så Stripe sitt klientside API. Dette APIet må lastes inn som ekstern kode og det er ikke anbefalt å ha scriptet lagret på serveren av sikkerhetsgrunner. Det lastes altså inn for hver donasjon som gjøres. Når brukeren har skrevet inn sine kortinformasjon verdier kjører jeg først validasjon på disse verdiene ved å bruke validasjonsmetodene i klientside APIet til stripe som kan si om dette er gyldige verdier til betaling. Dersom sjekken passerer lager jeg et paymentinfo objekt i Javascript som inneholder informasjon om brukeren. Dersom bruker har oppgitt adresse og mail legges dette også inn i objektet, men disse verdiene er frivillige og kan ikke gjøre at betalingen feiler. Dette objektet sendes så inn i Stripe.card.createToken metoden. Denne metoden går til Stripe sine serverene som validerer at det er et legitimt kort som er brukt. APIet aksepterer VISA og Mastercard. Dersom kortet ikke er legitimt vises en feilmelding om dette til bruker, men dersom det er legitimt får vi tilbake en paymenttoken som kan brukes til å belaste kortet til bruker på serversiden uten at serveren får tak i kortinformasjonen. Kodeutklippet nedenfor viser hvordan jeg får at i en token.

```
Stripe.card.createToken(paymentInfo, function (status, response) {
                if (response.error) {
                    $("#OncePaymentError").text(response.error.message);
                    return;
                } else {
                    $("#OncePaymentError").text("");
                    token = response.id;
                    processPayment(token);
                }
            });
```

Stripe har en pakke for håndtering av serverside betaling i .NET, men jeg syntes dokumentasjonen og hjelpemidler for dette på nett var litt manglende så jeg valgte rett og slett å bruke APIet direkte og uten noen pakke. Når token og kortinformasjon sendes til serveren lager jeg et donasjonsobjekt dersom det er en engangsdonasjon. Her puttes informasjon som kontaktinformasjon dersom det er oppgitt, beskrivelse av donasjonen og donasjonsmengde. Dersom bruker har bedt om å linke donasjonen til deres bruker kan dette også gjøres her. Deretter lager jeg en C# NameValueCollection(), som er mer eller mindre det samme som et JSON objekt med nøkler og verdier. I dette objektet puttes forskjellig informasjon som tokenen, type valutta, mengde og beskrivelse som er donasjonsIDen og beskrivelsen fra kundens side. Donasjonsmengden må oppgis i øre for norsk valuta med Stripe så donasjonsmengden må ganges med 100. Her legges det også til mail dersom organisasjonen ønsker å bruke Stripe sin innebygde kviteringsmailtjeneste. Hvis ikke brukes SendGrid dersom SendGrid er skrudd på for applikasjonen.

Videre sjekker vi om Stripe faktisk er konfigurert for applikasjonen, og dersom dette ikke er tilfellet returneres en feilmelding om dette. I utgangspunktet skal man ikke donasjon delen av siden vises dersom Stripe ikke er konfigurert, men å etterprøve på serversiden er viktig, for kanskje Stripe har blitt skrudd av for applikasjonen i den tiden bruker brukte på å skrive inn sin informasjon. Dersom dette er tilfellet blir en feilmelding returnert.

Videre lager jeg et WebClient objekt som gjør det mulig å kommunisere med APIet og legger inn den hemmelige delen av API nøkkelen i WebClient.Credentials feltet. Jeg forsøker så å laste opp NameValue objektet med donasjonsverdiene til API endpointet https://api.stripe.com/v1/charges. Dersom dette skaper en WebException, altså at APIet returnerer en feilmelding håndterer vi dette og returnerer responsestrengen. Dersom det ikke blir kastet noe unntak var donasjonen suksessfull og vi kan lagre databasen med den nye donasjonen som er markert som suksessfull. Dersom SendGrid er konfigurert for

applikasjonen og slått på sendes en kvitteringsmail til mailadressen dersom en er oppgitt. Uansett returneres et PartialView tilbake til klientsiden med kvitteringen som erstatter innholdet i donasjons-diven. Kvitteringen består av et referansenummer som er donasjonens ID i databasen, hvor mye som ble donert og mulighet for å dele på sosiale medier dersom dette er konfigurert og slått på for applikasjonen.

Dersom det er et donasjonsabonnement som blir inngått så er prosessen mye det samme, bare at her lages en kunde som lagres på Stripe sine nettsider ved hjelp av APIet. Denne kunden legges så til planen som er valgt ved å bruke planens unike navn. Abonnementer lagres ikke i databasen fordi disse kan fort bli avbrutt. Derimot lagres donasjonsmaler.

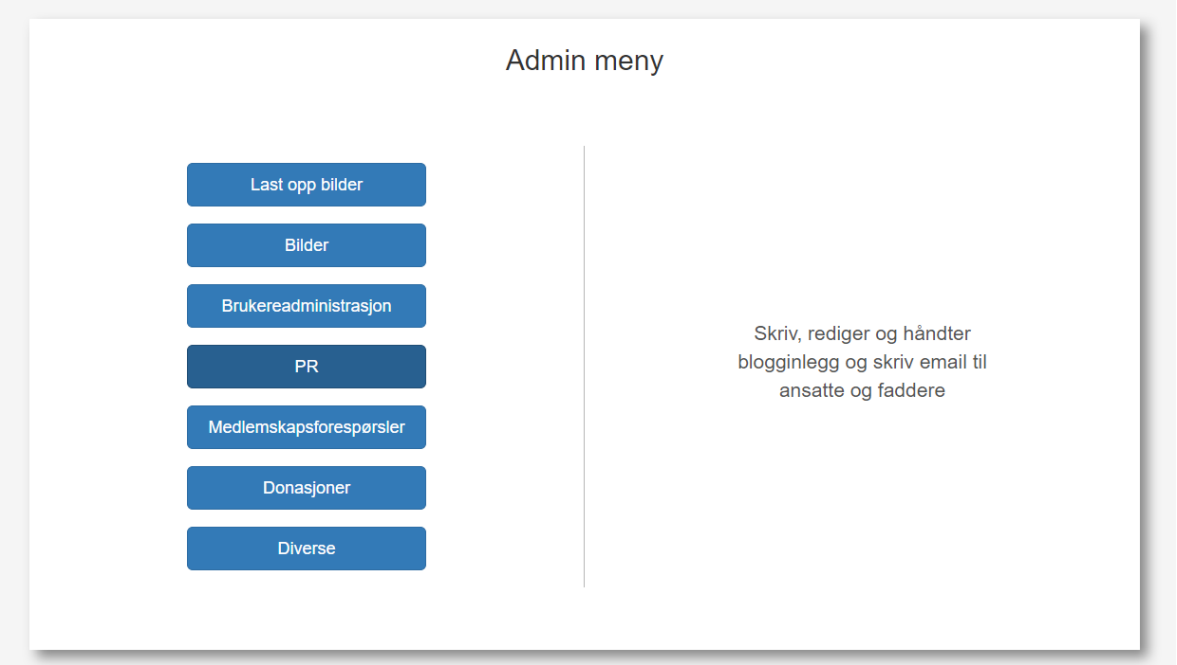
Stripe har sitt eget administrasjonssystem på sine nettsider som tilbyr veldig mye funksjonalitet som refusjoner og anmelding av bedrageri. Jeg vil nok oppfordre organisasjonen til å bruke dette verktøyet til å holde styr på det meste av donasjoner, men jeg har alikevel laget et panel i administrasjonsdelen av siden som tilbyr litt grunnleggende funksjonalitet. Først og fremst kan du se donasjoner som er lagret i databasen og detaljer om disse, samt søke og filtrere på referansenummer og bruker, i utgangspunktet filtrert fra nye til eldre donasjoner. Disse donasjonene kan i utgangspunktet ikke slettes og er bare et verktøy for å søke opp og se informasjonen. En annen del av dette panelet er abonnementene som hentes fra APIet og listes ut fra https://api.stripe.com/v1/subscriptions ved å bruke den unike IDen til de aktive abonnementsmalene. Abonnementmaler er de planene for betaling som bruker velger fra på forsiden når de skal sette opp månedlig betaling til organisasjone. Her kan også maler lages, deaktiveres og slettes. Når jeg lager en mal gjør jeg et kall til https://api.stripe.com/v1/plans med et unikt navn på planen, hvor ofte og hvor mye som skal betales. Dersom dette går gjennom blir planen lagret i databasen og planen blir aktivert. En plan kan også deaktiveres for å ikke være tilgjengelig for brukere eller den kan slettes fra systemet om det er ønskelig. Donasjonspanelet gjøres på like linje med andre paneler i applikasjonen med AJAX, paging og donasjoner.

3.3.6 Navigasjon og design

Navigasjon

Det er har vært viktig at systemet er enkelt å navigere i. Jeg har fulgt 3-klikksregelen fra kapittel 2 for å ikke gjøre navigasjonen for innviklet. Navigasjonen er enkelt delt opp og dersom det er forskjellig type innhold på de forskjellige sidene er de naturlig delt opp i sine seksjoner med faner eller avgrensede segmenter. Navigasjonen på fadder sidene er veldig enkel fordi det ikke er veldig mye funksjonalitet på disse sidene og dermed er det naturlig nok enkelt å finne frem

På administrasjonssiden er det mye funksjonalitet som må håndteres. Derfor har jeg logisk delt opp funksjonalitet i paneler og forsiden av adminpanelet viser disse forskjellige delene.



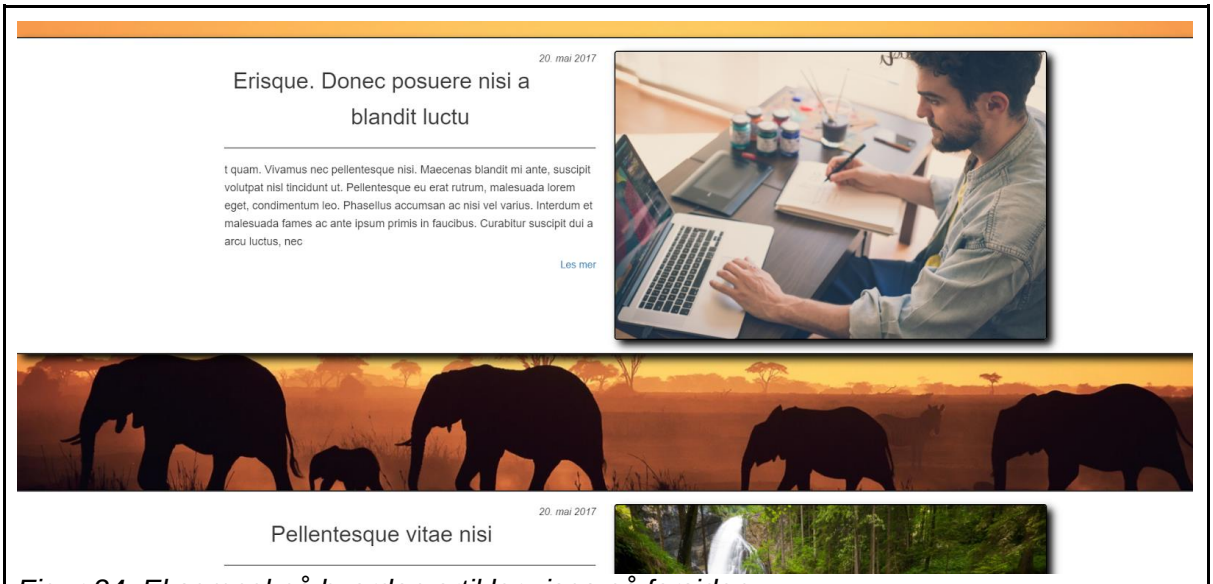
Figur 23: Administrasjonsmenyen.

Som vist i figur 23 er administrasjonsdelen av siden delt opp i 7 deler og du vil fra menyen kunne navigere deg til disse forskjellige delene av siden. På høyre side av menyen er det en beskrivelse av hva funksjonaliteten til de forskjellige panelene er. Denne dukker opp når bruker beveger musepekeren over en av disse sidene. Ut fra forsiden på kan du komme til en hvilken som helst funksjon i administrasjonspanelet i 3 klikk eller mindre. Noen av disse navigasjonsknappene fører til et panel med flere valg. Dersom du for eksempel velger PR vil du komme til en side der du kan administrere og se alle artikler i databasen, men også velge å navigere deg videre til å skrive en ny mail eller artikkel. Alt er ganske nøye strukturert og ingen av sidene er forsøkt å kun vise den informasjonen som er ønsket å vise for øyeblikket. Dersom du for eksempel er interessert i å se oversikt over API nøkler blir ikke bilder også vist.

Design

Design for administrasjonspanelet er ment å være enkelt, men behagelig for øynene å jobbe med. Dette inkluderer en hvit boks som vist i figur 22 og en komplementær grå farge rundt. Her skal det være enkelt å finne frem og bare et enkelt, men effektivt design er viktig. Jeg har også i applikasjonen brukt mye farger og ikoner fra Font Awesome og Bootstrap for å gjøre knappene mest mulig intuitive og selvforklarende. For eksempel så kan en knapp for å lagre være grønn med et lagre symbol, mens en sletteknapp kan være rød med en søppelkasse.

På forsiden og faddersiden av nettsiden er designet litt mer stilig for nettopp å inspirere engasjement og lokke potensielle faddere til å bli medlem. Den vanlige skriftstørrelsen er satt til å være 16 piksler som er trolig den mest behagelige skriftstørrelsen å lese på en nettside [1]. Når en ny bruker kommer til forsiden av nettsiden vil han først bli møtt med bilder og valget om å donere eller bli medlem, plassert side ved side. Dersom brukeren ikke ønsker dette kan han også bla ned der han kan bla gjennom artikler. Disse artiklene viser overskrift og ingress sammen med et bilde fra artikkelen dersom artikkelen er tilknyttet ett eller flere bilder (første bilde i som er listet i databasen tilhørende artikkelen brukes).



Figur 24: Eksempel på hvordan artikler vises på forsiden.

Som vist på figur 24 vises artiklene nedover samtidig som det vises bilder i bakgrunnen dersom dette er konfigurert. Disse artiklene lastes inn slik som bildene på fadder sidene lastes inn ved at de ikke lastes inn alle på en gang, men etterhvert som brukeren blar nedover på siden hentes eldre og eldre artikler og legges til nedover.

3.3.7 Dynamisk design

Det har vært viktig å gjøre siden dynamisk til forskjell fra oppdragsgivers nettside som i dag er veldig statisk. Dette vil si at innhold må kunne endres uten at man må gå inn og endre kildekoden. Det er naturlig at databaseentiteter som brukere, artikler, bilder og så videre kan hele tiden endres eller fjernes, men tekst og bilder på forsiden av nettsiden må også være redigerbare. I administrasjonspanelet er det et panel som heter “diverse”. Dette panelet er for funksjonalitet som ikke er stor nok til å få sitt eget panel og som ikke helt passer inn noe sted. Dette panelet er pent delt opp med faner av typer funksjonalitet og her kan for eksempel organisasjonen skrive inn grunnleggende informasjon om arbeidet og deres kontaktinformasjon som mailadresse og telefonnummer. Her kan også adresse til organisasjonens hovedkontor legges inn. De kan her hele tiden endre på disse variablene etterhvert som det er nødvendig og fylle på informasjon om bedriften som de vil dele. Dette vises på “Om oss” siden som linkes til i toolbaren på forsiden. Her kan leserne altså lese om organisasjonen og se informasjon som adresse og kontaktinformasjon. Alt dette er frivillig å legge inn og dersom bedriften ikke vil legge inn en adresse trenger de heller ikke dette og adressedelen av om oss siden vil dermed fjernes fra siden. Dersom de kun ønsker å oppgi by er dette også mulig. Her kan bedriften skrive en hel stil med flere avsnitt om seg selv om de ønsker det, eller bare si litt eller si ingenting i det hele tatt. Adresse og kontaktinformasjon blir også vist i footeren på forsiden. Footeren er segmentet som ligger helt på bunnen.

Sosiale medier kan også konfigureres på diverse sidene og her kan man legge til Facebook- og Twittersider etter ønske og velge om disse funksjonene skal være skrudd på eller ikke. De fleste av funksjonene på forsiden kan skrus av dersom organisasjonen ikke ønsker å ha dem der lenger. Sosiale medier, dersom de er konfigurert, vises på om oss sidene og gir brukere mulighet til å like den eller følge dem på twitter. Sosiale medier brukes også til å dele donasjoner. Dersom sosiale medier er slått av laster siden heller ikke inn scriptet som håndterer sosiale medier APIene. Disqus er et veldig populært kommentarnettverk for å enkelt og pent legge til kommentarfelt på siden din. Diqus URL kan også legges til under sosiale medier. Dersom Disqus funksjonen for nettsiden er skrudd på kan brukere kommentere på artikler ved å bruke sine sosiale medier kontoer eller sin Disqus konto

dersom de har det. Disqus har også gode kommentar administrasjon systemer som kan aksesseres gjennom deres nettsider.

I om oss delen av diversepanelet kan brukere med administratortillatelse (vanlige ansatte kan ikke nå dette panelet) legge til tekst ved donasjon- og medlemskapsdelene av forsiden. Dette er slik at organisasjonen selv kan prøve å appellere til besøkende for å få dem til å bli medlemmer eller donere. For eksempel så kan administrator legge inn en tekst om hvorfor det hjelper så mye at folk donerer penger og hvorfor dette er viktig. Denne teksten vil deretter vises sammen med resten av donasjonsdelen av siden for å appellere besøkende.

Alle API nøkler kan også administreres i denne delen av siden. Organisasjonen blir nødt til å lage sine egne kontoer for de forskjellige tjenestene som brukes. På denne måten får de også tilgang til den avanserte funksjonaliteten dashbordene brukersidene til Stripe og SendGrid tilbyr. Disse nøklene kan altså i diversepanelet legges inn slik at de blir oppdatert i databasen. Eventuelt trengs de ikke å legges inn eller funksjonaliteten kan skrus av dersom organisasjonen ikke lenger vi bruke denne funksjonaliteten.

Som nevnt tidligere kan brukervilkår også lastes opp i diversepanelet, men her kan også bilder til visning på fadder sidene lastes opp. Dersom ingen bilder lastes opp vil bare den veldig grunnleggende siden i blått, hvitt og grått vises. Grått er bakgrunnsfargen i nettsiden og kan byttes ut med bilder. Her kan det for eksempel lastes opp et bilde som vises som bakgrunn i applikasjonen. Dette bildet vil bli da vist i bakgrunnen på alle fadder sidene. Dette er ikke nødvendig for administrator sidene så her beholdes det grunnleggende designet, men sidene til fadderne blir med dette penere og mindre kjedelige dersom et hensiktsmessig bakgrunnsbilde med god resolusjon velges. I tillegg kan man laste opp en bildekarusell i det samme panelet. Bildekarusellen er en serie med bilder og/eller videoer som vises på forsiden av siden og kun forsiden. Disse elementene vises i rekkefølge med animasjoner som forfrisker bildevisningen. Dette er kun en funksjon for forsiden siden det kan være litt plagsomt å lese lange artikler med en bildekarusell i bakgrunnen. Dersom funksjonen er konfigurert med bilder og videoer og slått på vil URLene til elementene sendes til en jQuery plugin som heter Vegas som håndterer og viser bakgrunnsbildene.

3.4 Validering og testing

Testingen i prosjektet har blitt gjort i stor grad ved hjelp av en database initialiserer som har gjort det mulig å legge inn testverdier go deretter videre videre jobbe. APIer som betalingstjenesten laget med Stripe blir testet med Stripe sine testkort som ikke er ordentlige kort, men kan likevel brukes til å gjøre test betalinger.

Prosjektet har beveget seg i sprint sykluser og testing har skjedd periodevis etterhvert som funksjonalitet har blitt lagt til i systemet. Ekstra testing har blitt utført på slutten av prosjektet for å verifisere at alle krav består testene. Siden testplanen er ganske stor har jeg besluttet å legge den i et vedleggsdokument (Vedlegg F). Mange flere tester kan også bli utført for å vise at systemet virker ned til hver minste detalj. Ikke alle testene som har blitt gjort har blitt dokumentert av hensyn til lesbarhet og relevans til mål og krav for prosjektet. Kun de viktigste testene som beviser at krav og også deler av målene for brukervennlighet som er enkle å teste har blitt tatt med i testplanen. Jeg har ikke funnet noen tester som gjør at systemet krasjer, i og med at svært mye feilhåndtering skode har blitt tatt til hjelp.

Resultatet av den utførte testplanen viser at alle tester oppfører seg som forventet med unntak av den automatiske bildetaggingsfunksjonen som er litt uberegnelig og legger ofte til bokser på feil sted. Denne funksjonen har jeg derimot ikke kontroll over siden den i hovedsak styres av en jQuery plugin.

4 Drøfting

Prosjektet er satt i gang for å lage en nettsideløsning for en mindre fadderorganisasjon som har funksjonalitet for å håndtere på en brukervennlig og effektiv måte de fleste aspektene av driften til en slik organisasjon. Jeg vil påstå at alle problemer og krav til prosjektet er løst i dette prosjektet. Hvorvidt disse er de beste løsningene kan derimot diskuteres.

4.1 Brukervennlighet

Jeg har, som diskutert i kapittel 2.2 forsøkt å følge en del designprinsipper for å gjøre løsningen min så effektiv og brukervennlig som mulig. Det var viktig at løsningen skulle gi tilbakemeldinger til brukeren på handlinger. Når det gjelder generelle AJAX kall til serveren, noe det er veldig mye av i dette prosjektet så blir alle disse markert med i det minste et loading symbol som indikerer at serveren jobber med kallet og dersom det er en filopplastning prosess som pågår så indikerer jeg arbeidet og opplastningsprosessen med en fremdriftslinje som viser nøyaktig hvor langt prosessen har kommet. På ingen tidspunkt blir brukeren sittende uvitende om hvorvidt handlingen de foretok seg fungerte eller ikke. Med AJAX kall så er det ingen sideinnlastning slik når kallet er over er det veldig viktig med tilbakemeldinger på suksesser og feil. Jeg har tatt høyde for dette og alle AJAX kallene i prosjektet unntatt kallene som henter lister og modalbokser sender tilbake meldinger om enten suksess eller feil. Alle kallene har derimot mulighet for å vise feilmeldinger. Feilmeldinger er det mye av og de er ikke bare på serversiden de sendes, det er også mye klientside validering som resulterer i feilmeldinger i prosjektet for å effektivisere prosessen og slippe å legge all validasjon over til serveren. Alle feilmeldinger vises i rødt og suksess meldinger i grønt for å indikere resultatet visuelt for brukeren. Jeg vil påstå at disse tingene er viktige for brukervennligheten til systemet og dermed for å løse mitt problem.

Gjør derimot dette siden brukervennlig? Jeg vil i høyeste grad påstå dette. Det løser ikke alle problemer for optimalisering av brukervennlighet, men det gjør at innlastningstider er mye kortere, siden mindre deler av all informasjonen som inngår på en side hentes av gangen fra servere og oppdateres der det trengs. Tradisjonelt sett oppdateres nettsider for hver gang noe hentes eller sendes til serveren. Etter opplasting kan oppdatert informasjon vises. Med min AJAX baserte metode skjer alt når du vil uten at du må vente på at serveren skal bli ferdig. Det er dette som er det flotte med asynkrone kall som AJAX; resten av siden stopper ikke opp. Det å programmere uten AJAX hadde derimot spart prosjektet massivt mye tid ettersom det er veldig tidkrevende og til tider vanskelig å utføre alle disse kallene og delvise oppdateringene på en god måte. Jeg vil likevel argumentere for at dette er verdt det i en så brukervennlighet- og effektivitetsorientert prosjektoppgave.

Jeg har også tatt valg om å bruke mye modalbokser i mitt prosjekt. All informasjon som kan redigeres, med unntak av bildeopplastning- og tagginspanelet er styrt med modalbokser som gjør at brukeropplevelsen blir veldig mye bedre. Det er liten tvil om at både effektivitetsmessig, brukervennlighetsmessig og designmessig at modalbokser tilbyr mer til funksjonaliteten av prosjektet enn det å vise hver enkelt database entitet på en ny side. Lastetidene er mindre når man ikke trenger å laste inn en hel ny side, modalboksene viser stilige animasjoner når de aktiveres og de vises kun dersom brukeren trykker på en knapp som “tilkaller” dem. Spørsmålet blir igjen om hvorvidt det er verdt de lange timene som er brukt for å programmere modalboksenes funksjonalitet slik at de fungerer for ser ut som de skal i hele prosjektet. Jeg tror det er nødvendig selv om det kan påpekes at funksjonaliteten kanskje kan tones ned litt og ikke implementeres over hele applikasjonen. Problemet med dette igjen blir at designet, som også er en viktig del av oppgaven blir inkonsekvent med forskjellige løsninger for samme funksjonalitet. Det gjør at systemet ikke blir seende like bra ut, samtidig som dette kan skape forvirring hos brukere og skade brukervennligheten til systemet som jeg setter så høyt.

4.2 Design

Farger og fonter har blitt nøye valgt ut. Jeg valgte en veldig mild blåfarge som hovedfargen til applikasjonen. Dette er gjort for å ha en rolig og behagelig farge for øynene samtidig som organisasjonen Butterfly Friends har brukt blå som hovedfarge i logoen sin i årevis. Bakgrunnsfargen til nettsiden er en veldig lys grå som er kontemplær til de hvite boksene som nettsidens innhold plasseres i. Jeg kunne valgt en annen farge enn grå til å gå med det hvite, men sammen med det blåe som den siste fargen ser det best ut på dette viset. Grå er en veldig kjedelig bakgrunnsfarge, og andre lyse farger som veldig lys blå eller lilla er betydelig livligere farger å ha i bakgrunnen, men den er i det minste høyst nøytral. Det er lagt opp til at administratorer skal kunne legge til egne bilder som vises i bakgrunnen slik at applikasjonen blir mer spennende for brukeren, men da er det også greit å ha en veldig nøytral farge som grå som standardfarge dersom ikke det ønskes å gjøre applikasjonen ekstra spennende på dette viset. Dersom det er ønskelig uansett administrator også legge til et bilde i hvilken som helst farge de ønsker for å dekke bakgrunnen.

Fontstørrelsen har jeg satt til å være større enn standard. Fonten er 16 piksler på det minste, og for deler av applikasjonen, som ingressen til artiklene brukes 18. Standard skriftstørrelse er 14 piksler men jeg føler dette ikke er like behagelig å lese. Dette gjør at innhold tar litt mer plass, men jeg tror at når folk skal jobbe og lese mye på en slik nettside er det viktig med litt stor og behagelig skriftstørrelse i en behagelig font. Fontstørrelse 16 er faktisk regnet som, av profesjonelle utviklere, som den ideelle skriftstørrelse på nett, som ikke hverken tar for mye eller for lite plass. [1]

Navigasjonen forsøker å oppfylle krav om intuitivitet uten for mye kompleksitet. Dette gjøres med farger, ikoner og beskrivelser for hva de forskjellige panelene inneholder. Ingen funksjon er heller langt fra en annen. Det jeg derimot mangler linker i verktøylinjen på nettsiden. På faddersidene linkes alle relevante sider i verktøylinjen på toppen av siden, men dette er fordi her er det ikke så mange linker å velge mellom. Dersom du er innlogget er det bare 3 linker som vises i verktøylinjen siden fadderdelen av nettsiden ikke er så komplekst. 4 linker vises dersom du er innlogget som administrator. Administrasjonsdelen av siden er delt opp i 7 underseksjoner, hvor underseksjonen PR i tillegg er delt mellom email og blogg. Dersom alle disse funksjonene skal linkes til i verktøylinjen ville det bli 8 linker i tillegg til forsidelinkene og utloggingslinken. Dette føler jeg blir i overkant rotete som ledet meg til å heller lage et stort panel for navigasjon inn i andre paneler der det tydelig vises beskrivelse av hva hvert panel gjør. Dette gjør at du alltid må tilbake til forsiden av adminpanelet for å navigere rundt noe som til tider kan føles litt tungvindt. Alternativt kunne jeg laget en dropdown meny for navigasjon i aminpanelet og gjøre det mulig å navigere ut fra dropdown menyen som vanligvis gjemmes og dermed ikke gjør navigasjonen rotete og uoversiktlig.

Jeg har også valgt, når det gjelder kommentarsystemet på artiklene og ikke utvikle mitt eget, men heller bruke Disqus. Ulempen med dette er kanskje at det ikke tilbyr noen funksjonalitet for å kommentere med innloggede brukere. Jeg synes likevel dette er greit i og med at nettsiden ikke er et sosialt medie og kommentarer er noe som vises til offentligheten. Det er nok derfor mer hensiktsmessig å la faddere engasjere seg gjennom sine offentlige sosiale medier kontoer dersom de ønsker dette, samtidig som det gjør at utenforstående også kan engasjere seg i arbeidet og potensielt inspireres til å bli faddere. Disqus, som driver kommentarsystemet er også allerede godt utviklet med moderasjonsverktøy slik at jeg har spart veldig mye tid på å på veldig enkelt implementere kommentarsystemet med eksisterende systemer. Hadde jeg hatt tid til å bygge systemet helt fra bunnen av ville jeg likevel trolig ikke gjort det siden jeg føler funksjonaliteten passer fint slik den er nå.

4.3 Engasjement

Systemet er laget for å være høyst engasjerende med kommentarsystem, donasjon system tett tilknyttet bilder for å få frem følelser, bildesystem som lar faddere ta del i deres fadderbarns liv, email system som lar organisasjonen enkelt kommunisere med medlemmer og blogg og nyhetsfunksjon som legges ut i offentligheten som tilbyr åpenhet og tilstand på arbeid som skal hjelpe å introdusere organisasjonen for nye og allerede eksisterende medlemmer. Alt dette legger til rette for at organisasjonen skal kunne øke drastisk i støtte og medlemstall i og med at den nåværende løsningen tilbyr nesten ingenting. Dersom det hadde vært mer tid i prosjektet kunne jeg trolig ha implementert enda flere funksjoner for å hjelpe engasjement på vei, som chattefunksjon slik at organisasjonen kan raskt svare på spørsmål fra potensielle medlemmer og forum der medlemmer kan snakke om interesser, organisasjonsarbeid og bli bedre kjent med miljøet. Likevel tror jeg, men ressursene og tiden jeg ble tildelt for dette prosjektet at prioriteringen av hvilke funksjoner som ble implementert var riktig.

Om jeg startet prosjektet på nytt ville jeg gitt mer tid til rapportskrivning, men det er også det eneste. Jeg er generelt veldig fornøyd med mitt produkt og personlig føler jeg problemet løses på en veldig engasjerende og god måte. Produktet tilbyr ikke en fullstendig løsning på driften av en fadderorganisasjon, men den tilbyr en fullstendig løsning for de rammene som er gitt dette prosjektet med tanke på tidsfrister, mål og arbeidskraft. Jeg vil så absolutt si at mine resultater er valide for å oppnå et brukervennlig, effektivt, dynamisk og engasjerende system ved at det aller meste av sidens innhold kan endres så dø funksjonalitet vil trolig ikke bli et problem med mindre det innføres med vilje. Effektiviteten ligger i lastetider, delvis sideoppdateringer, filtreringer og søk, bildehenting og databasesøk optimalisering, og asynkron kode basert på AJAX kall som er over brukt nesten hele løsningen når det gjeller kommunikasjon mellom klient og server. Engasjementet er som forklart i seksjon 4.3 og brukervennligheten i 4.1. Kravene til prosjektet oppfylles også i høyeste grad og viser at all planlagt funksjonalitet ble levert som forventet (Se vedlegg F).

Prosjektet har lært meg mye om webdesign, prinsipper for webdesign, effektivisering av både innlasting informasjon og databasesøk, og også en hel masse om brukervennlighet i forhold til asynkron kode og oppdateringer og modalbokser for å vise innhold i stedet for å laste inn hele nye sider.

5 Konklusjon

Problemet i dette prosjektet har vært å finne ut av og implementere et effektivt, dynamisk og brukervennlig håndteringssystem for en mindre fadderorganisasjon som også forsøker å engasjere faddere og potensielle faddere til å bidra og bry seg om arbeidet.

Løsningen min hadde intensjoner om å være dynamisk, altså at siden ikke består av statisk, uforanderlig innhold, men heller at det meste, med unntak av enkelte rammer skulle kunne endres på etter hvert som organisasjonen har behov for å oppfriske informasjon og innhold. Løsningen skulle også forsøke å inspirere og engasjere gjennom bilder, artikler, kommentarsystem og velplasserte og enkle donasjon systemer. Effektiviteten har blitt oppnådd gjennom asynkron kode som returnerer og oppdaterer kun der bruker har behov for dette og databaseoptimalisering spiller en stor rolle i å kutte ned på søk- og innlastningstider, samt modalbokser har blitt brukt for å dynamisk, brukervennlig og effektivt laste inn og vise informasjon om entitetene i databasen.

Løsningen baserer seg på svært mye databaseadministrasjon av tekst på siden og også databasevariabler til brukere, bilder og artikler. Det er svært få entiteter i databasen som ikke enten kan undergå omfattende redigeringer eller fjernes fra databasen fullstendig.

Navigasjon og design er også laget for å være enkel og intuitivt for å gjøre opplevelsen mest mulig brukervennlig.

Ansatte i bedriften får et mye enklere og mer effektivt system å håndtere enn det som finnes i dag, noe som forhåpentligvis øker effektiviteten til det administrative arbeidet til organisasjonen, gjør arbeidet mer motiverende å jobbe med og at organisasjonen får både flere medlemmer og mer støtte gjennom systemene for kommunikasjon, medlemskap, administrasjon og donasjoner. Faddere i organisasjonen vil forhåpentligvis engasjere seg i arbeide i en helt ny grad enn det som ble gjort før når systemer for åpenhet, engasjement og donasjonsmuligheter er på plass.

Dersom prosjektet skulle utvikles videre ville jeg forsøke å også lage et system som administrerer og håndterer lønn til ansatte på en sikker, effektiv og brukervennlig måte og på denne måten virkelig håndtere alle aspekter at driften til en mindre fadderorganisasjon.

Referanseliste

1. Sofia Woods, “Top 10 principles for effective web design”, Shortie Designs, Brisbane, Quneensland, Australia, 5. mars 2014. [Online]. Tilgjengelig: https://shortiedesigns.com/2014/03/10-top-principles-effective-web-design/
2. “Number of monthly active Facebook users worldwide as of first quarter 2017(in millions)”,[Online]. Tilgjengelig: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
3. Hanne Bjugstad, “Medlemmene viser vei”, 16. September 2013. [Online]. Tilgjengelig: https://www.reddbarna.no/nyheter/medlemmene-viser-vei
4. A Medium Corporation, “Medium.com”, [Online]. Tilgjengelig: https://www.linkedin.com/company/medium-com
5. Alex Ivanovs, “WYSIWYG Website Builders for Online Business” , Huffington Post, 15. desember 2015, [Online]. Tilgjengelig: http://www.huffingtonpost.com/alex-ivanovs/wysiwyg-website-builders_b_8814882.html
6. “Introducing JSON”, [Online]. Tilgjengelig: http://www.json.org/
7. “About”, [Online]. Tilgjengelig: https://git-scm.com/about
8. Bill Wagner, Luke Latham og Maira Wenzel, “Introduction to the C# Language and the .NET Framework”, Microsoft, 5. mars 2017, [Online]. Tilgjengelig: https://docs.microsoft.com/en-us/dotnet/articles/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework
9. “What is reCAPTCHA?”, Google, [Online]. Tilgjengelig: https://www.google.com/recaptcha/intro/
10. “SendGrid Overview”, SendGrid, [Online]. Tilgjengelig: https://sendgrid.com/docs/User_Guide/index.html
11. “Walkthrough: Organizing an ASP.NET MVC Application using Areas”, Microsoft, [Online]. Tilgjengelig: https://msdn.microsoft.com/en-us/library/ee671793(v=vs.100).aspx
12. “What is MVC?”, TutorialsPoint, ,[Online]. Tilgjengelig: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm
13. “Introduction to Entity Framework”, Microsoft, [Online]. Tilgjengelig: Introduction to “ASP.NET Identity”, Microsoft, [Online]. Tilgjengelig: https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx
14. Jon Galloway, Andy Pasic og Tom Dykstra, “Introduction til ASP.NET Identity”, Microsoft, 17 oktover 2013, [Online]. Tilgjengelig: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity
15. “HTML”, Mozilla.org, [Online]. Tilgjengelig: https://developer.mozilla.org/en-US/docs/Web/HTML

16. http://getbootstrap.com/
17. “Ajax”, Mozilla, [Online]. Tilgjengelig: https://developer.mozilla.org/en-US/docs/AJAX
18. “What is jQuery?”, [Online]. Tilgjengelig: http://jquery.com/
19. “Introduction”, Vue.js, Tilgjengelig: http://vuejs.org/v2/guide/
20. “About jQuery UI”, http://jqueryui.com/about/
21. Sofia Woods, “10 top principles of effective web design”, Shortie Designs, 4 mars 2014, [Online]. Tilgjengelig: https://shortiedesigns.com/2014/03/10-top-principles-effective-web-design/
22. Vitality Friedman, “10 principles of effective web design”, Smash Magazine, 31 januar 2008, [Online]. Tilgjengelig: https://www.smashingmagazine.com/2008/01/10-principles-of-effective-web-design/#top
23. “Entity SQL Language”, Microsoft, [Online]. Tilgjengelig: https://msdn.microsoft.com/en-us/library/bb399560(v=vs.110).aspx
24. “Entity Framework 6 Runtime Licence” , Microsoft, [Online]. Tilgjengelig: https://msdn.microsoft.com/en-us/library/dn467385(v=vs.113).aspx
25. “Queries in LINQ to Entities”, Microsoft, [Online]. Tilgjengelig: https://msdn.microsoft.com/en-us/library/bb399367(v=vs.110).aspx
26. Mithun Pattankar, “Difference between ASP.NET Core MVC and ASP.NET MVC 5”, 26 september 2017, [Online]. Tilgjengelig: http://www.mithunvp.com/difference-between-asp-net-mvc6-asp-net-mvc5/
27. Jani Hyytiäinen, “MVC 5 or ASP.NET Core Now?”, 17 september 2016, [Online]. Tilgjengelig: https://www.linkedin.com/pulse/mvc-5-aspnet-core-now-jani-hyyti%C3%A4inen
28. “Limits to Sending and Recieving Mail”, Google Inc., [Online]. Tilgjengelig: https://support.google.com/mail/answer/22839?ctx=gsidentifier
29. “Prices”, Sendpulse, [Online]. Tilgjengelig: https://sendpulse.com/prices
30. “Gmail Sending Limits in G Suite”, Google Inc. [Online]. Tilgjengelig: https://support.google.com/a/answer/166852?hl=en
31. “Simple, Flexible Pricing”, SendGrid, [Online]. Tilgjengelig: https://sendgrid.com/pricing/
32. Elmer Thomas, Caro Caserio, Glenn Gailey, “How to Send Email Using SendGrid with Azure”, Microsoft, [Online]. Tilgjengelig: https://docs.microsoft.com/en-us/azure/app-service-web/sendgrid-dotnet-how-to-send-email
33. “Plans Designed to Grow Your Business”, Chargify, [Online]. Tilgjengelig: https://www.chargify.com/pricing/
34. “Forståelige Gebyrer”, Paypal, [Online]. Tilgjengelig: https://www.paypal.com/no/webapps/mpp/paypal-fees
35. “Simple, transparent pricing”, Stripe, https://stripe.com/no/pricing
36. “Developer First”, Stripe, [Online]. Tilgjengelig: https://stripe.com/no
37. “Comparison with Other Frameworks”, Vue.js, [Online]. Tilgjengelig: https://vuejs.org/v2/guide/comparison.html
38. “Best practices for securely using API keys”, Google Inc. ,[Online]. Tilgjengelig: https://support.google.com/cloud/answer/6310037?hl=en
39. “Gaver til frivillige organisasjoner”, Skatteetaten, [Online]. Tilgjengelig http://www.skatteetaten.no/no/Tabeller-og-satser/Gaver-til-frivillige-organisasjoner/
40. “Deling av bilder”, Datatilsynet, [Online]. Tilgjengelig: https://www.datatilsynet.no/Teknologi/Internett/Bilder-pa-nett/
41. “Using CSS in HTML Emails”,Bemchmarkemail , [Online]. Tilgjengelig: https://www.benchmarkemail.com/help-FAQ/answer/Using-CSS-in-HTML-emails

42. “Our Privacy Policy Has Changed”, Imgur, [Online]. Tilgjengelig: http://imgur.com/tos

43. “Scheduler”, Microsoft Azure, [Online]. Tilgjengelig: https://azure.microsoft.com/nb-no/services/scheduler/

44. Butterfly Friends, [Online]. Tilgjengelig: http://butterflyfriends.no/

45. “Normalisering”, Høyskolen i østfold, [Online]. Tilgjengelig: http://www.ia.hiof.no/~elinkaan/databaser/2003/normalisering.html

46. “Code style”, ReShaper, [Online]. Tilgjengelig: https://www.jetbrains.com/resharper/features/code_formatting.html

47. “ ”, Mozilla, [Online]. Tilgjengelig: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div

48. Ben Emmet, “Entity Framework Performance And What You Can Do About It”, 16. Desember 2015, [Online]. Tilgjengelig: https://www.simple-talk.com/dotnet/net-tools/entity-framework-performance-and-what-you-can-do-about-it/

49. Tom Dykstra og Andy Pasic, Microsoft, “Sorting, Filtering, and Paging with the Entity Framework in an ASP.NET MVC Application”, 6. Januar 2015, [Online]. Tilgjengelig: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/sorting-filtering-and-paging-with-the-entity-framework-in-an-asp-net-mvc-application
50. “What is Nuget”, nuget.org, [Online]. Tilgjengelig: https://www.nuget.org/
51. Anders Abel, “Secure Account Activation In ASP.NET Identity”, Kentor, 17. Mai 2015 [Online]. Tilgjengelig: https://coding.abel.nu/2015/05/secure-account-activation-with-asp-net-identity/

Vedleggliste

Vedlegg A: Databasediagram
Vedlegg B: Timelister
Vedlegg C: Prosjektplan
Vedlegg D: Møtereferater
Vedlegg F: Testplan