kjøre-hjelperen

Hovedprosjekt i informasjonsteknologi

Prosjektnr: 8

Henrik Hermansen **s171179**
Lars Smeby **s171200**

HØGSKOLEN I OSLO OG AKERSHUS

BEKK

Statens vegvesen

HØGSKOLEN I OSLO
OG AKERSHUS

PROSJEKT NR.
2013 - 8

TILGJENGELIGHET
Åpen

Studieprogram: Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

Telefon: 22 45 32 00
Telefaks: 22 45 32 05

HOVEDPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Kjørehjelperen	**26.05.13**
	ANTALL SIDER / BILAG
	151 / 7
PROSJEKTDELTAKERE	INTERN VEILEDER
Henrik Hermansen s171179 **Lars Smeby s171200**	**Eva Hadler Vihovde**

OPPDRAGSGIVER	KONTAKTPERSON
BEKK Consulting AS	**Christian Schwarz**

SAMMENDRAG

Kjørehjelperen er en applikasjon til iPhone som viser deg informasjon om vegen du befinner deg på når du er ute og kjører. Brukeren kan ved hjelp av applikasjonen enkelt se hvilken fartsgrense som gjelder, om han/hun er på en forkjørsvei og om det er noen farestrekninger eller –punkter lenger fremme.

3 STIKKORD
iPhone
Nasjonal Vegdatabank
Statens vegvesen

Innholdet i rapporten

Rapporten inneholder følgende dokumenter, i denne rekkefølgen:

Presentasjon

Prosessdokumentasjon

Produktdokumentasjon

Testdokumentasjon

Brukerveiledning

Kravspesifikasjon

Ordforklaringer og kilder

2013

Kjørehjelperen

Presentasjon

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Oppgaven vår i dette hovedprosjektet gikk ut på å lage en mobilapplikasjon som skal benytte Statens vegvesens datasett Nasjonal Vegdatabank.

Dette dokumentet er en presentasjon av hovedprosjektet for gruppe 8 ved Høgskolen i Oslo og Akershus, våren 2013. Gruppen, samt bakgrunn og mål for oppgaven presenteres her i sin helhet. Det gis også en kortfattet presentasjon av sluttproduktet og dets funksjonalitet, men dette vil vi gå nærmere inn på i produktdokumentasjonen.

Dette dokumentet bør leses i sin helhet før man fortsetter med de andre dokumentene i rapporten. Deretter bør man fortsette med prosessdokumentasjonen. Prosessdokumentasjonen inneholder flere referanser til produktdokumentasjonen og testdokumentasjonen. Disse kan leses i sin helhet, eller brukes som oppslagsverk. Den endelige kravspesifikasjonen og et dokument med ordforklaringer og kilder følger til slutt.

Ord og uttrykk forklares fortløpende i fotnoter. I tillegg finnes ordforklaringene samlet i dokumentet "Ordforklaringer og kilder" bakerst i rapporten.

NB: Fullstendig kildekode, samt en videodemonstrasjon av produktet er tilgjengelig på prosjektets nettside: http://student.iu.hio.no/hovedprosjekter/data/2013/08/.

Innholdsfortegnelse

1 Gruppen og oppgaven

Denne delen av presentasjonen omfatter gruppen og apparatet rundt, samt oppgaven med bakgrunn og mål for denne.

1.1 Gruppen

Gruppen vår består av Henrik Hermansen og Lars Smeby. Vi tar begge Bachelor i Informasjonsteknologi ved Høgskolen i Oslo og Akershus, og har jobbet sammen ved flere tidligere prosjekter. I tillegg har vi gått på skole sammen tidligere, og har også der jobbet sammen på prosjekter. Gjennom disse tidligere anledningene har vi erfart at vi har en god kommunikasjon oss imellom, og at vi samarbeider godt. Vi var også trygge på hverandre med hensyn til ambisjonsnivå, noe som vil gjøre det lettere å samarbeide og stille krav og forventninger til hverandre. I tillegg til at vi kjenner hverandre godt, både faglig og sosialt, ønsket vi begge å benytte denne erfaringen til å lære så mye som mulig. På bakgrunn av alt dette var vi trygge på hvilke mål og forventninger vi skulle ha til hverandre og oppgaven.

Figur 1: Oppdragsgiveren vår var BEKK Consulting.

1.2 Oppdragsgiveren

Oppgaven er utført for BEKK Consulting AS, som er et norsk konsulentselskap. De gjennomfører prosjekter for private og offentlige virksomheter innen strategisk rådgivning, utvikling av IT-systemer og design av digitale tjenester. De er i dag i overkant av 300 ansatte, og har kontorer i Oslo og Trondheim. For sjette år på rad er BEKK i tillegg rangert som det mest attraktive konsulentselskapet i Norge blant IT-studentene i Universum Student Survey.

Figur 2: Statens vegvesen var kunden i prosjektet.

Ansvarlig for oppgaven hos BEKK var *Christian Schwarz*, mens vår faglige veileder var *Christoffer Marcussen*. Christoffer er med i BEKKs faggruppe for applikasjonsutvikling på mobiltelefon og ville derfor kunne bistå oss i arbeidet.

1.3 Kunden

Oppgaven ble som sagt gitt oss av BEKK, men det var Statens vegvesen som i utgangspunktet har ønsket at denne applikasjonen skal utvikles. Dette innebar at det var BEKK vi skulle forholde oss til

med hensyn til utviklingsprosessen og det tekniske, men at Statens vegvesen stilte sin kompetanse til rådighet når det gjaldt innholdet i datasettet vi skulle arbeide med.

Vår kontaktperson hos Statens vegvesen var *Jan Kristian Jensen*.

Figur 1: Prosjektet ble utført ved Høgskolen i Oslo og Akershus.

1.4 Veileder og institusjon

Hovedprosjektet ble utført ved Institusjon for informasjonsteknologi ved Høgskolen i Oslo og Akershus.

Vår interne veileder ved høgskolen var høgskolelektor *Eva Hadler Vihovde*.

1.5 Bakgrunn for oppgaven

BEKK utviklet i 2012 et REST[1] API[2] for Statens vegvesen. Tanken var at dette APIet skulle gjøre data fra NVDB[3] gratis tilgjengelig for alle. Ettersom APIet var en helt ny tjeneste ønsket både BEKK og Statens vegvesen at folk skulle se mulighetene i dette produktet, og ta det i bruk ved å utvikle produkter og tjenester som utnytter data fra APIet.

En måte å synliggjøre mulighetene i APIet på er å utvikle et produkt som tar i bruk de forskjellige tjenestene APIet tilbyr. Det er nettopp dette vår oppgave gikk ut på. Statens vegvesen ønsket en applikasjon som enkelt demonstrerer noen av mulighetene i APIet, og BEKK ønsket at vi gjennom dette kunne utforske og teste funksjonaliteten de har utviklet.

1.6 Mål for oppgaven

Etter idémyldring og drøfting i samarbeid med begge våre kontakter hos BEKK, kom vi frem til at målet for oppgaven skulle være en applikasjon som gir nyttig informasjon om vegen man kjører på. Vi bestemte at målgruppen skulle være "alle som kjører bil" for å gjøre den så allment nyttig som mulig. Den konkrete idéen ble altså en applikasjon som viser gjeldene og kommende skilt langs veien man kjører på, med en målsetning om at applikasjonen skal være enkel å bruke, ikke kreve interaksjon under kjøring og presentere informasjonen på en så lettfattelig måte som mulig.

[1] Representational State Transfer, REST, er en programvarearkitektur for distribuerte systemer som World Wide Web. REST har etter hvert blitt den dominerende designmodellen for web-APIer.
[2] Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.
[3] Nasjonal vegdatabank, database med vegobjekter forvaltet av Statens vegvesen.

2 Sluttproduktet

Kjørehjelperen er en applikasjon til iPhone[4] som gir brukeren informasjon om veien han/hun kjører på. Telefonen legges eller monteres på dashbordet i bilen. Applikasjonen bruker så telefonens GPS[5] til å finne ut hvilken vei den befinner seg på, og viser deretter brukeren informasjon om denne veien.

Figur 2: En Navstar-2-satellitt brukt i GPS i bane rundt jorden (Foto: Wikimedia Commons).

Brukeren kan blant annet bli informert om hvilken fartsgrense det er der han/hun befinner seg, om vedkommende kjører på en forkjørsvei eller en motorvei og om det er fare for at elg, hjort eller rein krysser vegbanen. Det varsles også om farer og andre objekter av interesse lenger fremme: høydebegrensninger, jernbanekryssinger, farlige svinger, rasfare, osv. Totalt kan brukeren varsles om 47 ulike hendelser og interessepunkter langs norske veier.

Figur 3: Kjørehjelperen i bruk.

Informasjonen vises i form av kjente norske trafikkskilt, og skilt basert på disse. Størrelsen på skiltene varierer etter type telefon[6], og etter hvor mange skilt som skal vises samtidig. Applikasjonen kan maksimum vise syv skilt av gangen (fem på iPhone 4 eller eldre modeller). For objekter som ligger lengre fremme vises også avstand til disse. Denne avstanden oppdateres kontinuerlig, inntil objektet er passert. Når det dukker et nytt objekt på skjermen varsles dessuten brukeren med en lyd.

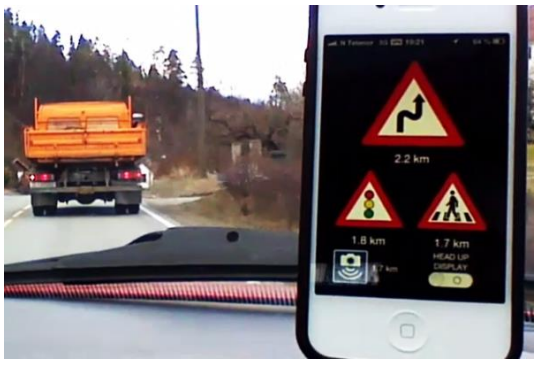
Figur 4: iPhone 4S til venstre, og iPhone 5 til høyre (Foto: ITProPortal.com).

[4] iPhone er en smarttelefon fra Apple Inc.
[5] NAVSTAR Global Positioning System (GPS) er et nettverk av satellitter som er plassert i bane rundt Jorden av det amerikanske forsvaret. Systemet gjør det mulig for en mottaker å fastsette egen posisjon med svært stor nøyaktighet overalt i verden, under nær sagt alle værforhold.
[6] Tidligere modeller av iPhone har en skjerm på 3,5", mens nyeste modell (iPhone 5) har en skjerm på 4".

Ved førstegangsbruk viser applikasjonen i utgangspunktet alle objekter og hendelser på skjermen. Brukeren kan imidlertid selv velge hva han/hun vil varsles om, og kan slik tilpasse applikasjonen til eget behov og ønske. Det er også mulig å deaktivere lydvarsling hvis ikke dette er ønskelig. I tillegg kan brukeren styre hvor mye lagringsplass Kjørehjelperen får lov til å bruke på telefonen[7].

Figur 5: Applikasjonen kan tilpasses etter eget ønske og behov.

Kjørehjelperen kan brukes både i portrett- og landskapsmodus[8]. Sensorer i telefonen registrerer hvilken vei enheten er orientert, og tilpasser visningen dertil. Applikasjonen kan også vise informasjon speilet i frontruten på bilen, kjent som head-up display (HUD)[9]. Ved å aktivere dette speilvendes all informasjon på telefonens skjerm. Ved å så legge telefonen på dashbordet, vil man kunne se informasjonen som en refleksjon i frontruten. Når HUD er aktivert låses telefonen i landskapsmodus. Dette gjøres for å hindre uønsket endring av orientering når telefonen ligger flatt på dashbordet.



Figur 6: Kjørehjelperen i portrettmodus.

Figur 9: Kjørehjelperen i landskapsmodus.

[7] Applikasjonen mellomlagrer data på enheten for å begrense datatrafikk. Les mer om mellomlagringen av data i produktdokumentasjonen, kapittel 3.1: "Core Data".
[8] Henviser til om telefonen orienteres henholdsvis stående eller liggende.
[9] Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.

3 Apps4Norge

Figur 10: Kjørehjelperen med HUD aktivert.

Onsdag 8. mai 2013 kom Kjørehjelperen på 3. plass i konkurransen Apps4Norge. Dette var en konkurranse om å lage en applikasjon som benytter seg av åpne, offentlige data. Ettersom vår applikasjon gjør nettopp dette, valgte vi å melde den på konkurransen. Selv om vi hadde meldt applikasjonen vår på en konkurranse arbeidet vi målrettet med hensyn til hovedprosjektet, for å gjøre et best mulig prosjekt. Konkurransen var i regi av Difi[10] og IKT-Norge. Mer informasjon om konkurransen finnes i prosessdokumentasjonen, kapittel 2.3.7.

Figur 7: Henrik og Lars med premien for 3. plass i konkurransen Apps4Norge. (Foto: Statens vegvesen)

[10] Direktoratet for forvaltning og IKT

2013

Kjørehjelperen

Prosessdokumentasjon

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Det forutsettes at leseren har lest presentasjonen av dette prosjektet før denne rapporten leses. Presentasjonen gir innblikk i viktig informasjon angående gruppen og oppdragsgiveren, samt bakgrunnen for og mål med prosjektet. Når presentasjonen er lest kan denne rapporten leses som et selvstendig dokument.

Videre forutsettes det at leseren har kompetanse innen programmering og systemutvikling, og er kjent med datatekniske begreper. Vi har valgt å legge det tekniske innholdet på et slikt nivå da dette dokumentet i hovedsak er beregnet på sensor og veileder. For ordens skyld vil vi allikevel kort forklare en god del av de ord og uttrykk som kan være fremmed for noen, ettersom man ikke kan forvente at en datakyndig person kjenner til alle datatekniske begreper. Disse forklares i fotnoter, og finnes også i "Ordforklaringer og kilder" bakerst i rapporten.

Prosessdokumentasjonen inneholder en detaljert gjennomgang av arbeidsprosessen vår gjennom hele prosjektet. Vi har her valgt å dele opp denne dokumentasjonen i fire hoveddeler:

- **Planlegging og metode** tar for seg forhold rundt planlegging og samarbeid, kommunikasjon, kompetansebygging i forhold til prosjektet, metodikk og verktøy og teknologier vi har benyttet underveis.
- **Utviklingsprosessen** beskriver prosjektets faser i detalj, og tar også for seg hver enkelt sprint under produksjonsfasen. Denne delen vil også ta for seg større utfordringer vi har hatt underveis, og hvordan vi gikk frem for å løse disse.
  I denne delen vil vi gjerne trekke frem kapittel 2.3.6, "Utfordringer underveis", som et spesielt interessant kapittel for den eller de som skal vurdere dette prosjektet.
- **Kravspesifikasjonen og dens rolle** vil omtale hvordan vi har benyttet kravspesifikasjonen og hvilken rolle den har spilt underveis i prosjektet. Her vil vi også se nærmere på oppfylte krav, avvik fra kravspesifikasjonen, samt endringer av kravspesifikasjonen underveis.
- **Avsluttende del** tar for seg refleksjoner og konklusjoner vi har rundt prosjektet. Vi vil her gå inn på nytteverdien ved produktet, muligheter for videre utvikling, samt hvilket læringsutbytte vi har hatt av dette prosjektet.

Innholdsfortegnelse

1 Planlegging og metode

I denne delen av dokumentasjonen vil vi ta for oss forhold rundt planlegging og samarbeid, kommunikasjon, kompetansebygging i forhold til prosjektet, metodikk og verktøy og teknologier vi har benyttet underveis.

1.1 Arbeidsforhold og samarbeid

Da vi skulle danne grupper til hovedprosjektet var det ikke vanskelig å vite at vi skulle jobbe sammen på dette prosjektet. Vi har under hele studiet jobbet sammen på flertallet av prosjektene vi har hatt, og vi kjenner derfor godt til hverandre fra før. Vi har gjennom de tidligere prosjektene erfart at vi oppfører oss jevnbyrdige i en gruppesammenheng, og at vi er flinke til å drøfte utfordringer og finne de beste løsningene. Ved utfordringer og uenigheter kan vi alltid argumentere vår sak ovenfor hverandre, og vi er da flinke til å kombinere alle disse argumentene slik at begge bidrar til den endelige avgjørelsen. I tillegg kjenner vi hverandres styrker og svakheter, og vet at vi utfyller hverandre godt. Basert på dette, samt at vi begge har det samme ambisjonsnivået, var valget om at vi skulle samarbeide om hovedprosjektet veldig enkelt.

Vi bor begge i Stor-Oslo-området, og det har derfor ikke vært noe problem å samarbeide på skolen eller hos hverandre, noe som har gjort det enklere å tilpasse arbeidssituasjonen og hvor vi har arbeidet fra dag til dag.

Vår oppgave var i utgangspunktet en utfordring for oss begge siden ingen av oss hadde jobbet med utvikling til iPhone[1] før. Utvikling til iPhone innebar at vi måtte lære oss et nytt programmeringsspråk, Objective-C[2], og et nytt rammeverk, Cocoa Touch[3]. Vi måtte også lære oss et nytt IDE[4], Xcode[5], og til og med et nytt operativsystem, OS X[6]. Man må nemlig bruke Mac om man skal utvikle programvare til OS X eller iOS[7]. Dette var mye å sette seg inn i, men det var også et bevisst valg av oss da vi valgte oppgave. Vi ønsket begge å utvide kunnskapsplattformen vår og å utfordre oss selv i hovedprosjektet.

Figur 1: iOS 6.1 er det nyeste operativsystemet til iPhone, og det vi måtte utvikle til.

[1] Smarttelefon fra Apple Inc.
[2] Objective-C er Apples programmeringsspråk for OS X og iOS. For detaljer se kapittel 1.5.5.
[3] Cocoa Touch er et rammeverk for å utvikle programvare for iOS. For detaljer se kapittel 1.5.6.
[4] Integrated Development Environment er programvare som tilbyr omfattende verktøy for programmering og programvareutvikling.
[5] Xcode er Apples IDE for å utvikle programvare til OS X og iOS. For detaljer se kapittel 1.5.4.
[6] OS X er det gjeldende operativsystemet til Mac.
[7] iOS er Apples operativsystem for iPhone, iPad og iPod.

1.2 Kommunikasjon

Vi vil her beskrive nærmere hvordan vi kommuniserte med interessenter og med hverandre underveis i prosjektet. Generelt har ikke kommunikasjon vært noe problem, og vi har som oftest fått tak i de personer vi har hatt behov for innen rimelig tid, slik at ikke arbeidet har lidd unødig på grunn av dette.

1.2.1 I gruppen

Innad i gruppen har kommunikasjon fungert godt under hele prosjektet. Under planleggingen av prosjektet arbeidet vi mye sammen, og vi avtalte da ofte fra gang til gang når vi skulle jobbe videre.

I produksjonsfasen var vi flinke til å fordele oppgaver, og begge hadde derfor alltid noe å gjøre, slik at vi ikke hadde behov for å kommunisere mye hver eneste dag. Under denne fasen holdt vi jevnlig kontakt via Skype[8], eller telefon dersom noe hastet og den ene ikke var pålogget Skype. Utover dette møttes vi ved behov under produksjonsfasen, men det var det ikke mye behov for, ettersom mange avgjørelser kunne tas via Skype.

Vi har også hatt en internettside til prosjektarbeidet vårt, og denne har også fungert som en form for kommunikasjon oss imellom, men har her ikke spilt noen stor rolle. Nettsiden har blitt benyttet til å føre logg for prosjektet, samt å tilgjengeliggjøre diverse dokumenter i forbindelse med planlegging av prosjektet.

1.2.2 Med oppdragsgiver og kunde

Med oppdragsgiver og kunde har all fjernkommunikasjon foregått via e-post. Vi har hatt jevnlig kommunikasjon med vår veileder hos oppdragsgiver, *Christoffer Marcussen*, samt *Frode Reinertsen* som har vært vår kontakt hos BEKK dersom vi har hatt spørsmål eller forespørsler i forhold til det tekniske i NVDB[9] API[10]. Utover dette har vi hatt noe kommunikasjon med *Jan Kristian Jensen* hos Statens vegvesen når vi har hatt spørsmål angående datasettet i NVDB. Samtlige av disse kontaktpersonene har vi hatt god kommunikasjon med per e-post, og nesten uten unntak fått raske og gode svar.

Figur 2: Vår oppdragsgiver var BEKK Consulting AS.

I tillegg har vi jevnlig hatt møter med Christoffer underveis i både planlegging og under produksjon. Møtene har som oftest funnet sted i BEKKs lokaler på Vippetangen. På disse møtene har vi fått faglig og teknisk veiledning, samt veiledning i arbeidsprosessen og fremdriften. Vi har på disse møtene

[8] Programvare for IP-telefoni. For detaljer se kapittel 1.5.1.
[9] Nasjonal vegdatabank, database med vegobjekter forvaltet av Statens vegvesen.
[10] Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.

også hatt demo for Christoffer, og han har veiledet oss i å presentere produktet, samt gitt oss konkrete tilbakemeldinger på selve produktet vi da har presentert. Dette har vært til god hjelp for oss og representert en trygghet for oss for å sikre at vi gjør et solid arbeid.

1.2.3 Med veileder

Kommunikasjonen vår med veileder *Eva Hadler Vihovde* har for det meste bestått av faste, avtalte møter. I planleggingsfasen var dette veldig nyttig da Eva veiledet oss med å gjøre valg og prioriteringer underveis i planleggingen, og hun ga nyttige råd under arbeidet med kravspesifikasjonen. Ettersom Eva ikke er kjent med utvikling for iPhone og iOS kunne hun naturlig nok ikke gi oss teknisk veiledning under produksjonsfasen, men hun ga oss nyttig veiledning med tanke på brukervennlighet og funksjonalitet i produktet. I tillegg til dette fungerte hun som en pådriver for fremgang i prosjektet, ettersom vi viste frem produktet vårt ved hver sprint til henne også.

HØGSKOLEN I OSLO OG AKERSHUS

Figur 3: Vår veileder ved Høgskolen i Oslo og Akershus var Eva Hadler Vihovde.

Utover dette har det vært sporadisk kommunikasjon med Eva per e-post, sms eller telefon, når det har vært behov for dette. Totalt sett har vi hatt god kommunikasjon med Eva og fått god veiledning.

1.3 Tilegning av kunnskap

Det var mye ny kunnskap vi måtte tilegne oss til dette prosjektet. Før vi kunne starte med noe utvikling for iOS, måtte vi først lære oss programmeringsspråket Objective-C. Dette var noe vanskeligere enn andre språk, da syntaksen er veldig annerledes fra andre språk vi har lært. Da vi etter hvert behersket Objective-C kunne vi begynne å sette oss inn i bibliotekene og rammeverkene for iOS. Vi jobbet hele tiden med forskjellige tutorials[11] på Internett, som tok for seg alt fra hvordan man lager sin første applikasjon, til mer avanserte og konkrete temaer som Core Data[12] og RestKit[13].

Statens vegvesen sier selv at NVDB API er et kraftig verktøy, men krever at man bruker tid på å sette seg inn i det for å kunne utnytte det på en god måte. Vi brukte derfor mye tid på å sette oss inn i APIet, forstå dataene, hvordan man benyttet dem og hvordan de hang sammen. En kort forklaring på hvordan NVDB API fungerer og hvordan det brukes finnes i kapittel 1.2 i produktdokumentasjonen.

[11] En tutorial er en stegvis gjennomgang på hvordan man lager eller utfører noe.
[12] Core Data er et rammeverk for databaselagring på iOS.
[13] RestKit er et Objective-C-rammeverk for iOS som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen av objekter.

Ettersom BEKK var vår oppdragsgiver ga de oss noen retningslinjer for hvordan vi skulle bedrive prosjektutvikling. BEKK er opptatt av smidig[14] utvikling, og benytter Scrum[15] i sine prosjekter. Vi fikk derfor en innføring av dem i grunnleggende Scrum og bruk av brukerhistorier. Les mer om Scrum i "1.4.3 Utviklingsmodell".

1.4 Planleggingsverktøy og metodikk

I dette kapittelet vil vi ta for oss de verktøy og hjelpemidler vi har benyttet til planlegging og prosjektstyring, samt hvilken metodikk vi har fulgt i denne prosessen.

1.4.1 Dokumentasjon

I alle prosjekter er det viktig å ha en viss plan på hva om skal gjøres og hva som skal produseres. I starten er det behov for overordnede planer, og nærmere produksjonsstart er det behov for mer spesifikke planer. Hvor detaljerte og langsiktige disse planene er avhenger av hvilken utviklingsmodell man ønsker å følge, men det vil uansett være behov for en form for planlegging. Denne planleggingen ble dokumentert i diverse styringsdokumenter (se kapittel 1.4.1.2, "Styringsdokumenter", for detaljer).

I tillegg til styringsdokumenter er det viktig å dokumentere arbeidet underveis slik at man kan gå tilbake og se hvem som gjorde hva, og når. Dette valgte vi å gjøre i en egenprodusert, nettbasert løsning (se "1.4.2.1 Prosjektlogg").

1.4.1.1 Dokumentasjonsstandard

Dokumentet "Dokumentasjonsstandard for bachelor-prosjekter (hovedprosjekt) for Institutt for informasjonsteknologi Høgskolen i Oslo og Akershus" av Ann-Mari Torvatn har fungert som en veiledning underveis i arbeidet med dokumentasjonen. Både når det gjaldt styringsdokumentene, men også når det gjaldt sluttrapporten. Vi valgte bevisst å ikke følge dokumentasjonsstandarden slavisk, men heller bruke den som en inspirasjon og veiledning til hva som burde omtales og hvordan det burde struktureres. Dette valgte vi fordi det var elementer i dokumentasjonsstandarden vi ikke følte at passet helt til vår dokumentasjon. Dette valget ble også tatt i samråd med veileder Eva Vihovde.

1.4.1.2 Styringsdokumenter

Under utrednings- og forprosjektfasen ble det opprettet noen dokumenter som skulle si noe om fremgangen i prosjektet, samt fungere som retningslinjer for den senere utviklingen. Under disse fasene ble følgende styringsdokumenter produsert:

[14] Agile, refererer til en iterativ og inkrementell systemutviklingsmodell.
[15] Scrum er en iterativ og inkrementell systemutviklingsmodell. Les mer i kapittel 1.4.3.

- **Statusrapport** var et dokument som ble levert i oktober 2012. Dette var en kort rapport der vi beskrev hvor langt vi var i prosessen med å finne en oppgave. På dette tidspunktet hadde vi sendt en søknad til BEKK, og avventet svar.
- **Prosjektskisse** var det første dokumentet som beskrev hvilken retning prosjektet faktisk skulle ta. Det var avtalt at BEKK skulle være oppdragsgiver og vi hadde skrevet første utkast av omfanget til prosjektet.
- **Forprosjektrapport** ble skrevet i januar 2013, noe tid etter prosjektskissen. I dette dokumentet var flere detaljer på plass, og vi hadde utformet noen mål og rammebetingelser for prosjektet.
- **Kravspesifikasjon** ble skrevet i starten av februar. Av styringsdokumentene har dette vært det viktigste dokumentet under produksjonsfasen, da det var dette dokumentet som konkret spesifiserte hvilke krav som gjaldt til funksjonalitet, brukervennlighet og design. Les mer om kravspesifikasjonen og dens rolle i del 3 av dette dokumentet.

1.4.2 Prosjektside på nett

Under hele prosjektet har vi hatt en prosjektside på nett. Her har vi hatt generell informasjon om prosjektet, publisert styringsdokumentene underveis og loggført arbeidet vårt i en prosjektlogg. I tillegg har vi hatt lenker til oppdragsgiver og veileder, samt GitHub[16]-repositoriet[17] vårt, på denne siden. Mot slutten av produksjonsfasen la vi også ut en demonstrasjonsvideo av applikasjonen vår på denne siden.

1.4.2.1 Prosjektlogg

Som nevnt over førte vi en prosjektlogg på prosjektsiden vår. Her skrev vi individuelle innlegg hver gang vi hadde jobbet hver for oss, samt innlegg for gruppen om vi hadde vært i møter eller jobbet sammen. Kort sagt har alt arbeid vi har gjort blitt loggført i denne loggen, sammen med tidsbruk for det aktuelle arbeidet. Dette har vært til nytte ved at vi har kunnet gå tilbake og se når vi gjorde hva og hvem som gjorde hva.

[16] GitHub er en nettbasert tilbyder av tjenester for prosjekter som bruker Git versjonskontrollsystem. Les mer om Git under kapittel 1.5.3.
[17] Repository er et begrep fra versjonskontroll og omhandler en datastruktur som bl.a. inneholder filer, mapper, historikk og referanser til tidligere versjoner.

2013-03-18	
	La til et vilttrekk-objekt, lagde et superobjekt som alle vegobjektene arver fra, flyttet kode som gikk igjen inn i passende klasser og metoder, og omstrukturerte tolkingen av returnerte objekter i VegObjektKontroller til å utnytte fordelene ved arv og kunne dermed fjerne mye unødvendig kode.
210 minutter	
2013-03-15	
Selvstudie om brukerpreferanser/innstillinger i iOS.	
120 minutter	
2013-03-15	
Skype-møte ang prosessen videre, drøfting rundt veiskilt og app-innstillinger.	
60 minutter	
2013-03-14	
	Fikset problemene rundt landskapsmodus og HUD ved å opprette 3 forskjellige viewcontrollere hvor den ene tar seg av all logikk, og de to andre arver fra den første og tar seg av henholdsvis portrett- og landskapsvisning.
240 minutter	

Figur 4: Et utdrag fra prosjektloggen på nettsiden.

1.4.3 Utviklingsmodell

Vi har ikke hatt en konkret utviklingsmodell som vi har fulgt helt etter boka, men vi har jobbet etter prinsipper fra smidige utviklingsmodeller og i hovedsak tatt elementer fra Scrum.

Kjennetegnet ved agile utviklingsmodeller er at de er iterative og inkrementelle, og at krav og løsninger utvikles og velges underveis i prosjektet, i samarbeid med de involverte parter. Dette innebærer at det blir mindre planlegging i forkant av prosjektet, mindre byråkrati og enklere å gjøre endringer i rammer og mål underveis, etter hvert som man ser endringer i behov og muligheter underveis i prosjektet. Ved slutten av hver iterasjon skal det alltid leveres en kjørbar versjon av programvaren, slik at denne kan demonstreres.

Som allerede nevnt jobbet vi ikke konkret etter Scrum, men brukte følgende elementer fra modellen:

- **Iterasjonsbasert arbeid**, i Scrum kalt sprinter. Ved starten av hver sprint plukkes det arbeidsoppgaver og funksjoner fra produktkøen, og disse oppgavene skal så sorteres i prioritert rekkefølge. Antallet oppgaver velges basert på grovestimering.
- **Grovestimering** av tidsbruk av arbeidsoppgaver og funksjonalitet som skal implementeres. I praksis estimeres det ikke antall timer per oppgave, men kompleksitet og omfang evalueres, og så estimeres det hvor mye man rekker i løpet av en sprint.
- **Stand-up** er daglige møter der team-medlemmene koordinerer arbeidet seg imellom. Vi hadde ikke alltid daglige møter, men hadde kommunikasjon oss i mellom tilnærmet daglig. Dersom man følger Scrum skal disse Scrum-møtene gjennomføres stående, være relativt korte og team-medlemmene skal omtale følgende tre spørsmål:
    - Hva er gjort siden forrige Scrum-møte?

- Hva skal gjøres før neste møte?
- Hva har (eventuelt) vært til hinder for at team-medlemmet var effektiv i implementeringen av funksjonalitet?

Vi gjennomførte ikke denne delen av Scrum-møtene, men for erfaringens skyld bakte vi den inn i sprint-møtet ved slutten av hver sprint, sammen med sprint retrospective.

- **Sprint review og sprint retrospective** utføres ved slutten av hver sprint. I sprint review demonstreres funksjonaliteten som ble lagt til i det foregående inkrementet, slik at man kan få tilbakemeldinger fra de partene som er interessenter i prosjektet. Deretter gjennomføres sprint retrospective der man samler erfaringer fra siste sprint og finner måter å jobbe litt mer effektivt på neste sprint.

Utover det ovennevnte kan vi legge til at vi arbeidet etter sprinter som varte i to uker, og delte produksjonsfasen vår inn i fem sprinter. Les mer om dette i kapittel 2.3, "Produksjonsfasen".

1.4.4 Versjonskontroll og backup

Til styringsdokumenter og dokumentasjon arbeidet vi begge på hver våre lokale filer i lokale mapper. Disse mappene hadde vi begge synkronisert med hver vår sky-tjeneste, slik at det automatisk alltid var en backup av dokumentene i nettskyen. For ordens skyld kan det nevnes at Lars brukt Microsoft SkyDrive, mens Henrik brukte norske Jottacloud. Ingen av oss opplevde noen problemer med disse tjenestene, og vi var godt fornøyde med denne måten å ha backup på.

Figur 5: Prosjektet har et eget åpent repository på GitHub, her representert av maskoten Octocat (Foto: GitHub).

Under produksjonsfasen brukte vi Git for versjonskontroll, og hadde et åpent repository på GitHub. Vi valgte å bruke dette i hovedsak fordi det er et kjent og solid system, samt at vi hadde benyttet det ved tidligere anledninger, og var derfor noe kjent med det. Vi var fornøyd med dette valget og hadde svært få problemer underveis. De problemene vi hadde skyldtes utelukkende manglende kunnskap fra vår side, men dette fant vi løsninger til på Internett, og tok således lærdom av dette.

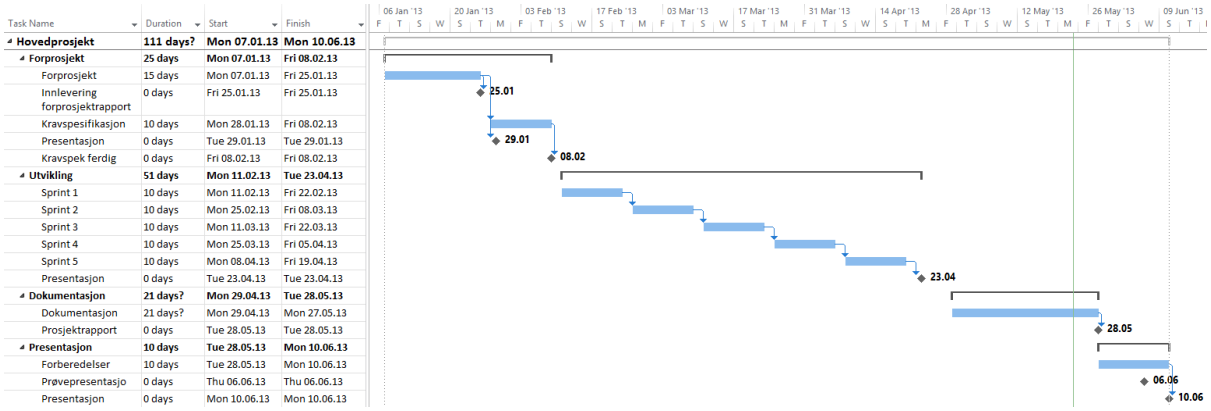
1.4.5 Diagrammer

Som en del av planleggingen tegnet vi også diverse diagrammer underveis. Dette ble ikke gjort på bakgrunn av den utviklingsmodellen vi jobbet etter, men vi ønsket å ha noen overordnede planer for

diverse aspekter av prosjektet. Alle planene er noe revidert underveis, basert på erfaringer og lærdom vi fikk etter hvert som vi jobbet, men ideene er fortsatt de samme. Vi vil her presentere de diagrammene vi har benyttet, slik de så ut etter siste revisjon.

1.4.5.1 Gantt-diagram

Et Gantt-diagram er ikke noe man egentlig skal lage når man arbeider smidig og etter Scrum. Vi utarbeidet heller ikke et Gantt-diagram slik det skal brukes, der man legger inn alle oppgaver og alt som skal utvikles, og så estimerer tidsbruk for hele prosjektet. Istedenfor brukte vi Gantt-diagrammet som en overordnet plan for å holde styr på diverse frister, samt å kunne fordele sprintene våre jevnt utover den tiden vi hadde til produksjonsfasen.



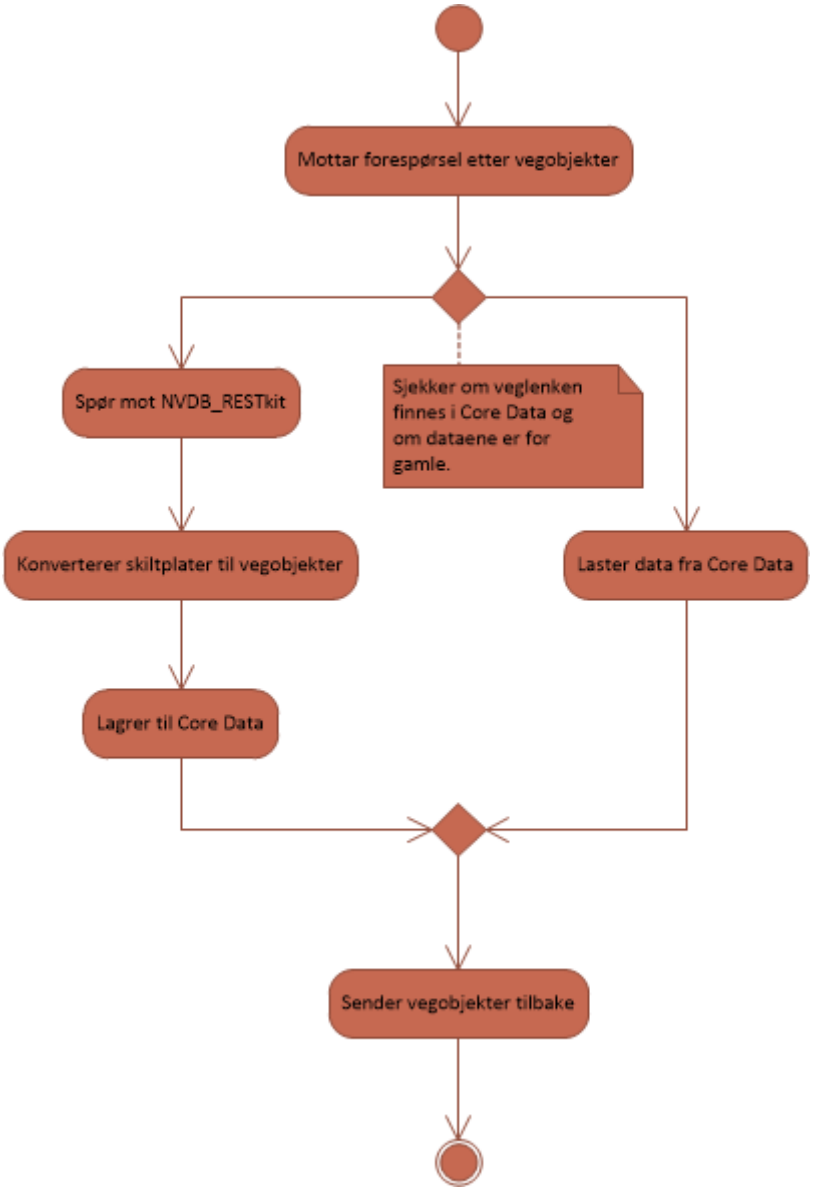
Task Name	Duration	Start	Finish
Hovedprosjekt	**111 days?**	**Mon 07.01.13**	**Mon 10.06.13**
Forprosjekt	**25 days**	**Mon 07.01.13**	**Fri 08.02.13**
Forprosjekt	15 days	Mon 07.01.13	Fri 25.01.13
Innlevering forprosjektrapport	0 days	Fri 25.01.13	Fri 25.01.13
Kravspesifikasjon	10 days	Mon 28.01.13	Fri 08.02.13
Presentasjon	0 days	Tue 29.01.13	Tue 29.01.13
Kravspek ferdig	0 days	Fri 08.02.13	Fri 08.02.13
Utvikling	**51 days**	**Mon 11.02.13**	**Tue 23.04.13**
Sprint 1	10 days	Mon 11.02.13	Fri 22.02.13
Sprint 2	10 days	Mon 25.02.13	Fri 08.03.13
Sprint 3	10 days	Mon 11.03.13	Fri 22.03.13
Sprint 4	10 days	Mon 25.03.13	Fri 05.04.13
Sprint 5	10 days	Mon 08.04.13	Fri 19.04.13
Presentasjon	0 days	Tue 23.04.13	Tue 23.04.13
Dokumentasjon	**21 days?**	**Mon 29.04.13**	**Tue 28.05.13**
Dokumentasjon	21 days?	Mon 29.04.13	Mon 27.05.13
Prosjektrapport	0 days	Tue 28.05.13	Tue 28.05.13
Presentasjon	**10 days**	**Tue 28.05.13**	**Mon 10.06.13**
Forberedelser	10 days	Tue 28.05.13	Mon 10.06.13
Prøvepresentasjo	0 days	Thu 06.06.13	Thu 06.06.13
Presentasjon	0 days	Mon 10.06.13	Mon 10.06.13

Figur 6: Dette Gantt-diagrammet ble laget i begynnelsen av prosjektet.

Underveis i prosjektet endret vi den siste sprinten da vi så at vi hadde behov for mer tid. I utgangspunktet var planen vår å ta fri i påskeferien, men vi så at vi fikk knapt med tid, og valgte derfor å inkludere påskeuka i sprint 5, slik at også denne fikk en varighet på to uker.

1.4.5.2 Aktivitetsdiagram

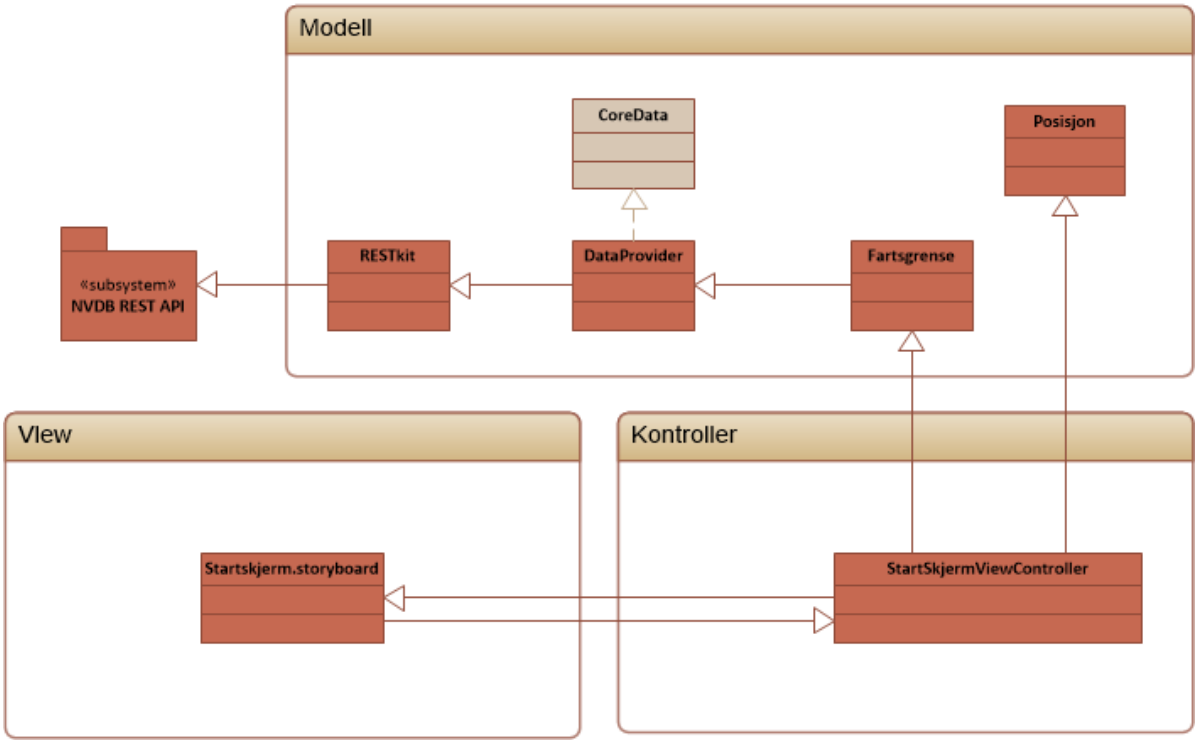
For å hjelpe oss med å se gangen i databehandlingen internt i applikasjonen tegnet vi opp et aktivitetsdiagram. Dette handler altså ikke om aktiviteter som brukeren av applikasjonen vil foreta seg. Dette er heller ikke en naturlig del av smidig utvikling, men vi ønsket å ha dette på papir, slik at vi hadde en felles forståelse av hva målet skulle være da vi utviklet dette.



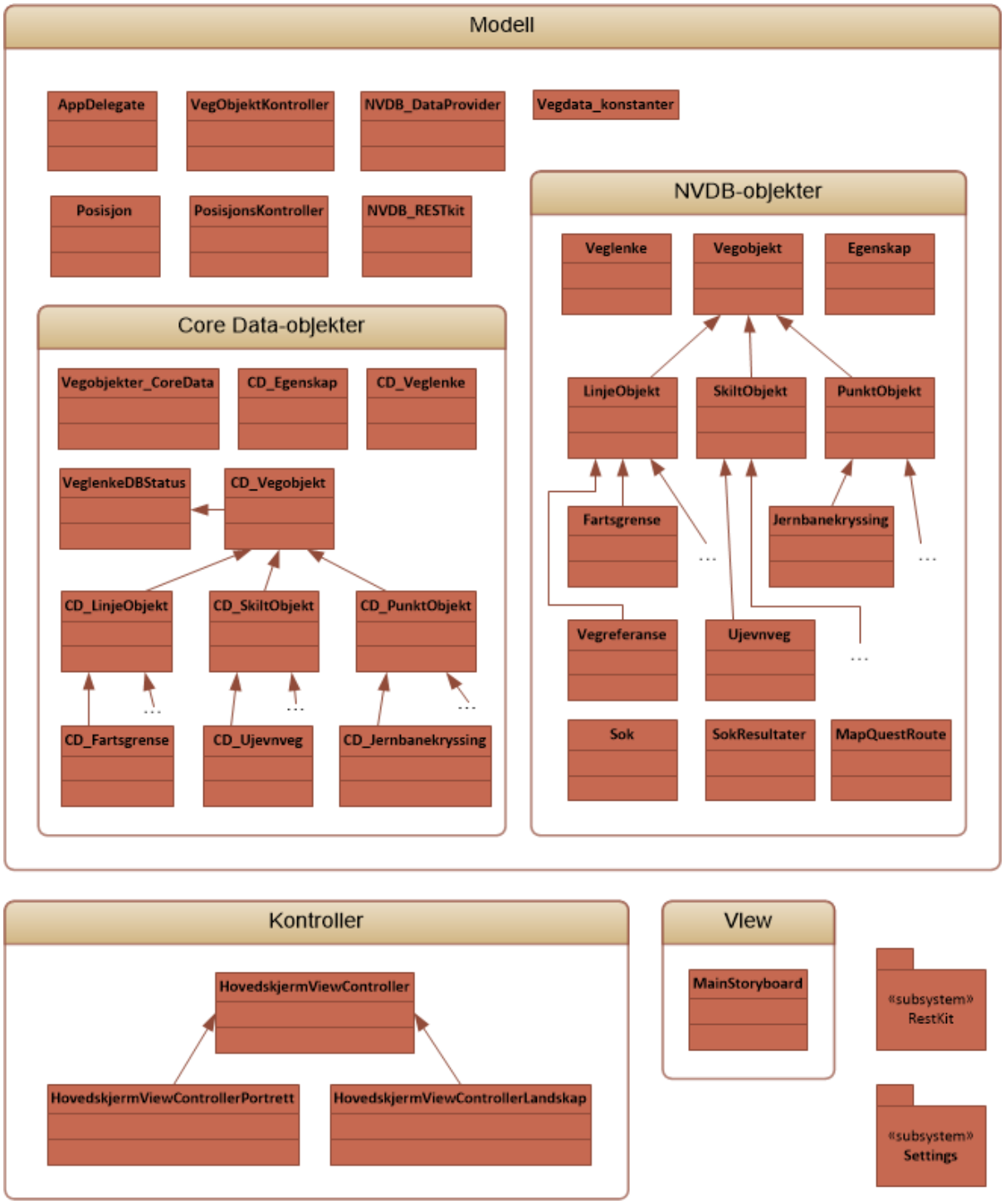
Figur 7: Aktivitetsdiagrammet viser hva som skjer i NVDB_DataProvider-klassen når den mottar en forespørsel etter vegobjekter. Les mer om dette i produktdokumentasjonen, kapittel 4.3.

1.4.5.3 Klassediagram

Ettersom vi ganske raskt så at det ville bli en del klasser å holde styr på, opprettet vi et klassediagram for å få oversikt. Dette diagrammet ble i første omgang konstruert med de klasser vi så behov for i starten. Etter hvert som vi gjorde oss erfaringer med tekniske utfordringer og hvilke behov vi faktisk hadde, ble diagrammet revidert. Revisjonen skjedde da alltid i forkant av endringene, slik at vi hadde en klar plan for vegen videre.



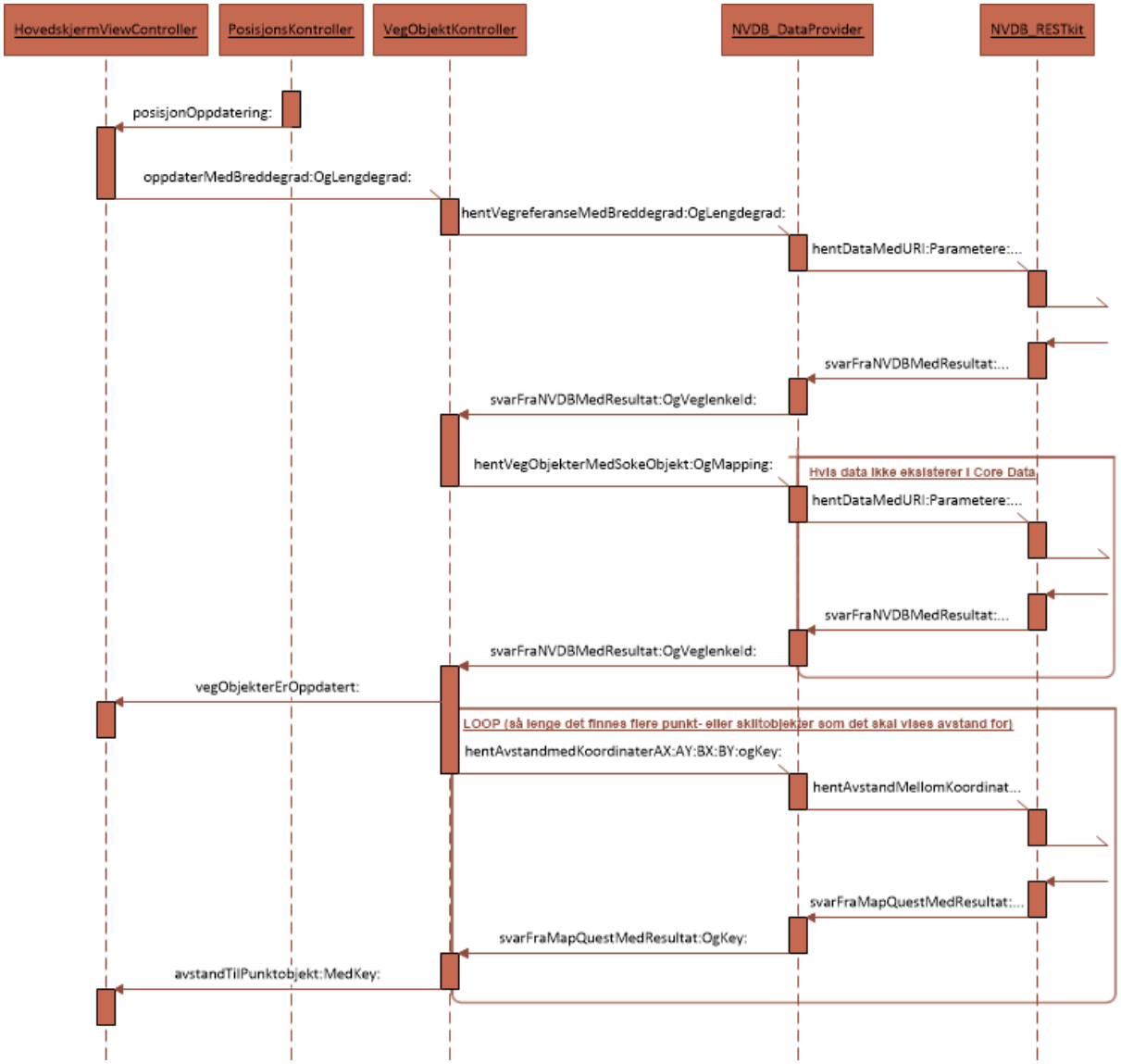
Figur 8: Slik så klassediagrammet ut til første sprint.



Figur 9: Dette er det endelige klassediagrammet i prosjektet, og viser hvordan Kjørehjelperen ser ut i dag.

1.4.5.4 Sekvensdiagram

I forhold til å holde oversikt over kommunikasjon mellom klasser og den logiske sammenhengen i programmet ønsket vi også å benytte et sekvensdiagram. Dette ble opprettet nokså sent i prosjektet. Diagrammet har ikke hatt spesiell stor nytteverdi for oss, men det er inkludert i produktdokumentasjonen, og vil der kunne ha stor nytteverdi for eventuelle utviklere som skal videreutvikle prosjektet.



Figur 10: Sekvensdiagrammet viser en normalkjøring i applikasjonen.

1.4.6 Brukerhistorier

Som en del av planleggingen benyttet vi oss også av brukerhistorier. Dette er noe BEKK er vant med å gjøre, så de ønsket tidlig at vi skulle sette oss litt inn i dette, ettersom de er kjent med nytteverdien av dette. Brukerhistoriene ble skrevet ved starten av hver sprint, og fungerte således som en rettesnor for målet med den funksjonaliteten vi til en hver tid arbeidet med å implementere. Vi vil i de følgende underkapitler gjengi brukerhistoriene vi skrev, sprint for sprint.

1.4.6.1 Sprint 1

Fartsgrenser

Som sjåfør skal jeg kunne se fartsgrensen på vegen jeg kjører på slik at jeg kan justere farten min.

Akseptansekrav:

- Jeg skal kunne se fartsgrensen på vegen jeg kjører på

- Jeg skal slippe å interagere med telefonen
- Fartsgrensen skal presenteres i samme stil som fartsgrenseskiltet

1.4.6.2 Sprint 2

Hindre telefonen fra å sovne

Som sjåfør skal jeg kunne se applikasjonen hele tiden slik at jeg ikke behøver å interagere med den mens jeg kjører.

Akseptansekrav:

- Mens applikasjonen kjører skal telefonen ikke skrus av automatisk

Forkjørsveg

Som sjåfør skal jeg kunne se om jeg er på en forkjørsveg eller ikke slik at jeg vet om jeg må være ekstra oppmerksom på trafikk fra høyre.

Akseptansekrav:

- Jeg skal kunne se om vegen jeg er på er en forkjørsveg eller ikke
- Informasjonen skal presenteres i samme stil som forkjørsvegskilt

Landscape

Som sjåfør skal jeg kunne velge om jeg vil ha telefonen i portrett eller landskapsmodus slik at jeg kan legge fra meg telefonen slik jeg vil og fremdeles kunne lese informasjonen.

Akseptansekrav:

- Jeg skal kunne vende telefonen og fremdeles få presentert informasjonen riktig veg
- Informasjonen skal tilpasses telefonens visningsmodus best mulig

1.4.6.3 Sprint 3

HUD (Head-up display)

Som sjåfør skal jeg kunne legge telefonen på dashbordet slik at jeg får presentert informasjonen som en refleksjon i frontruten[18].

Akseptansekrav:

- Jeg skal kunne se informasjonen på riktig måte reflektert i frontruten
- Informasjonen skal presenteres i samme stil som ellers, men tilpasses for klarest mulig refleksjon

[18] Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.

Varselskilt

Som sjåfør skal jeg kunne få informasjon om kommende og gjeldende varselskilt* om diverse farer og annet som varsles om slik at jeg kan gjøre nødvendige forberedelser når jeg nærmer meg og være tilstrekkelig oppmerksom når jeg er i et område der et varselskilt er gjeldende.

Akseptansekrav:

- Jeg skal få et varsel om kommende varselskilt* i rimelig tid før det vil være gjeldende.
- Jeg skal se hvilke varselskilt* som er gjeldende der jeg er nå.
- Informasjonen skal presenteres i samme stil som varselskiltet den refererer til.

* Det er kun krav om utvalgte varselskilt. Hvilke varselskilt dette gjelder vil bestemmes nærmere under utvikling.

1.4.6.4 Sprint 4

Brukerpreferanser

Som sjåfør skal jeg kunne velge hvilken informasjon jeg skal få presentert, og hvilke varsler jeg ønsker å motta slik at applikasjonen fungerer etter mine ønsker og behov.

Akseptansekrav:

- Jeg skal kunne endre hvilken informasjon jeg får presentert av applikasjonen.
- Jeg skal kunne endre hvilke varsler jeg får presentert, bl.a. varsler om varselskilt.

Lydvarsling

Som sjåfør skal jeg bli varslet ved hjelp av lyd når det på skjermen bli presentert nye skilt for strekningen jeg er på slik at jeg ikke behøver å konstant følge med på skjermen, men kan bli gjort oppmerksom på dette gjennom lydvarsling.

Akseptansekrav:

- Jeg skal ved hjep av audio bli gjort oppmerksom på at applikasjonen oppdager og presenterer nye skilt underveis på streknigen jeg er på.

Avstandsvisning

Som sjåfør skal jeg få informasjon om avstanden til skilt og objekter på strekningen jeg er på slik at jeg kan ta høyde for dette og gjøre eventuelle forberedelser i god tid.

Akseptansekrav:

- Jeg skal kunne se avstanden til et skilt eller det objektet skiltet representerer.

1.4.6.5 Sprint 5

Ingen brukerhistorier.

1.5 Verktøy og teknologi

1.5.1 Skype

Skype er programvare som brukes til IP-telefoni. Brukere som har installert programvaren og opprettet konto kan ringe til hverandre via datamaskinen. Ettersom samtalen skjer over Internett er dette helt kostnadsfritt med tanke på antall samtaler og lengden på samtalene. Programmet har også funksjonalitet for IM[19], filoverføring og videosamtaler. Med unntak av videosamtaler benyttet vi oss også av disse funksjonene.

Figur 11: Skype ble ofte benyttet til kommunikasjon.

1.5.2 Trello

Trello er en nettbasert applikasjon for prosjektstyring. Funksjonalitet som skal implementeres, eller spesifikke oppgaver som skal gjøres, opprettes som et kort og legges i en liste over oppgaver som skal gjøres, en såkalt "backlog". Hvert kort kan tildeles en eller flere deltakere i prosjektet, som skal løse den spesifikke oppgaven. Når en oppgave er påbegynt flyttes kortet til listen over oppgaver som jobbes med, og når den er ferdig flyttes kortet videre til en liste over ferdige oppgaver. Slik kunne vi enkelt holde oversikt over hva som skulle gjøres, hvem som gjorde hva og når oppgaver var utført.

1.5.3 Git

Git er et distribuert versjonskontrollsystem med mulighet for alle-til-alle synkronisering. Git tilbyr desentralisert arbeidsflyt med en tillitsbasert pyramidestruktur. Ved hjelp av Git var det enkelt å utvikle hver for oss, selv om vi til tider gjorde endringer på de samme filene, ettersom Git for det meste håndterer sammenfletting av disse endringene.

1.5.4 Xcode

Xcode er Apples integrerte utviklingsmiljø for utvikling av programvare til OS X og iOS. Ettersom Xcode kun kan benyttes i OS X er man avhengig av å ha en Mac for å kunne bruke det. Xcode har også støtte for andre språk enn C og Objective-C, men det var kun dette vi hadde behov for i vårt prosjekt.

[19] Instant Messaging, eller lynmelding på norsk, er en type samtale over et nettverk der man sender tekstbeskjeder til hverandre i sanntid.

1.5.5 Objective-C

Objective-C er Apples objektorienterte høynivå-programmeringsspråk[20] for OS X og iOS som tilfører Smalltalk[21]-lignende kommunikasjon til programmeringsspråket C. Objective-C ble opprinnelig utviklet av NeXT på starten av 80-tallet for deres operativsystem NeXTSTEP.

1.5.6 Cocoa Touch

Cocoa Touch er et rammeverk for å utvikle programvare som kan kjøre på iPhone, iPod Touch og iPad. Cocoa Touch tilbyr et abstraksjonslag for iOS og er basert på MAC OS X Cocoa API. I likhet med Cocoa følger også Cocoa Touch designmønsteret MVC[22]. Verktøy for å utvikle applikasjoner basert på Cocoa Touch er inkludert i iOS SDK[23].

[20] Et høynivåspråk har stor abstraksjon fra måten datamaskinen fungerer.
[21] Smalltalk er et dynamisk, objektorientert og reflekterende programmeringsspråk fra 1980.
[22] Model-view-controller er et designmønster for programvareutvikling. For detaljer se produktdokumentasjonen kapittel 2.1, "Model View Controller".
[23] Software development kit (SDK) er verktøy som lar deg utvikle programmer til en spesifikk programvarepakke, et rammeverk, en hardware-plattform, et operativsystem e.l.

2 Utviklingsprosessen

I denne delen av prosessdokumentasjonen ønsker vi å belyse utviklingen av prosjektet. Noen av de følgende kapitler er også dels omtalt i andre kapitler, men der med et annet fokus. I denne delen vil det være et kronologisk fokus der vi vil belyse fremdriften i prosjektet.

Den kronologiske fremdriften kan også ses på figur 6 i kapittel 1.4.5.1, "Gantt-diagram".

2.1 Utredningsfasen

Utredningsfasen var en noe vag fase på slutten av 2012. Den begynte allerede da vi satte sammen grupper, og varte frem til vi kunne begynne å konkretisere oppgaven vår.

I denne fasen skrev vi først en statusrapport, og senere en prosjektskisse. Vi begynte også smått å forberede oss for iOS og Objective-C da prosjektskissen var klar og det var klart at det var dette vi skulle jobbe med.

2.2 Forprosjektsfasen

Denne fasen begynte rett over nyttår, og det var nå vi måtte begynne å sette de første konkrete målene og rammebetingelsene for prosjektet. Dette var en myk start på denne fasen, med en glidende overgang til utformingen av konkrete krav til prosjektet.

I denne fasen skrev vi i januar en forprosjektrapport der overordnede mål og rammebetingelser ble formulert. Da dette var klart, i slutten av januar, begynte vi arbeidet med kravspesifikasjonen. 8. februar var denne ferdig, og vi nærmet oss da å kunne gå over i produksjonsfasen.

Underveis i denne fasen startet vi også arbeidet med å tilegne oss nødvendig kunnskap om de språk og rammeverk vi ville få bruk for i produksjonsfasen.

2.3 Produksjonsfasen

Kort tid etter at kravspesifikasjonen var klar gikk vi over i produksjonsfasen. Vi hadde på det tidspunktet tilegnet oss tilstrekkelig med kunnskap innen Objective-C og iOS SDK til å kunne begynne på basisen til produktet. Som nevnt i "1.4.3 Utviklingsmodell", arbeidet vi iterativt i såkalte sprinter. Vi hadde totalt fem sprinter, og vi vil her presentere grovt hvordan arbeidet ble fordelt over de fem sprintene. Dette var ikke noe vi kunne planlegge på forhånd – hver sprint ble planlagt ved starten av den aktuelle sprinten.

2.3.1 Sprint 1

Mandag 11.02.13 – fredag 22.02.13

I den første sprinten var det mye grunnarbeid som måtte gjøres, og produktet vi leverte ved slutten av sprinten hadde derfor lite funksjonalitet. Den vesentlige basisfunksjonaliteten som ble implementert her var:

- Finne lokasjonskoordinater ved hjelp av telefonens GPS[24]
- Et kommunikasjonslag for å kommunisere med NVDB API
- Et skjermbilde å vise data i

Figur 12: Etter første sprint gjennomførte vi en testtur av applikasjonen.

Videre benyttet vi denne basisfunksjonaliteten til å implementere følgende funksjonalitet:

- Benytte kommunikasjonslaget mot NVDB API for å hente ut informasjon om hvilken veg man er på
- Benytte informasjonen om hvilken veg man er på for å hente ut fartsgrensen på den aktuelle vegen
- Vise mottatte data om veg og fartsgrense på skjerm. Fartsgrense ble her vist grafisk som et skilt

Da denne sprinten var over var vi godt fornøyd med å være i gang, og å ha noe konkrete data å vise. Vi ble også raskt bevisste på hvor lang tid utvikling tok med et språk vi var såpass ferske i, spesielt når vi også jobbet mot et API vi ikke var drevne i, men dette var nyttige erfaringer vi tok med oss til de følgende sprintene. Vi avsluttet denne sprinten med en testtur, noe som var veldig nyttig ettersom vi ble oppmerksom på hvordan GPSen oppførte seg, hvordan kommunikasjonen mot NVDB API gikk over tid og hvordan vi kunne forbedre disse aspektene. Les mer om selve testen i testdokumentasjonen, kapittel 2.1.

[24] NAVSTAR Global Positioning System (GPS) er et nettverk av satellitter som er plassert i bane rundt Jorden av det amerikanske forsvaret. Systemet gjør det mulig for en mottaker å fastsette egen posisjon med svært stor nøyaktighet overalt i verden, under nær sagt alle værforhold.

2.3.2 Sprint 2

Mandag 25.02.13 – fredag 08.03.13

Da neste sprint startet nølte vi ikke med å ta fatt på utfordringene. I denne sprinten tok vi fatt på diverse problemer vi oppdaget på testturen i slutten av sprint 1, og mot slutten av sprinten fikk vi implementert noe ny funksjonalitet. De sentrale forandringene i denne sprinten var som følger:

- Hindre telefonen fra å "sovne", slik at applikasjonen alltid er åpen og på når den først er åpnet
- Diverse forbedringer i oppdateringen av GPS-posisjon, som optimalisert oppdateringsfrekvens og økt presisjon
- Endret måten vi hentet informasjon fra NVDB API for å begrense databruk og øke nøyaktigheten i resultatet
- La til landskapsmodus for applikasjonen
- La til forkjørsveg og presenterte denne informasjonen på samme måte som fartsgrense

Etter denne sprinten valgte vi å ikke ta noen testtur, ettersom det var såpass lite ny funksjonalitet. Vi fikk dekket de behov vi hadde for testing gjennom iOS-simulatoren i Xcode.

2.3.3 Sprint 3

Mandag 11.03.13 – fredag 22.03.13

Ved tredje sprint begynte vi å bli varme i trøya. Vi så at grunnmuren i applikasjonen nå begynte å være solid, og vi fokuserte nå mer på å utvide applikasjonen til å kunne presentere mer informasjon. I løpet av denne sprinten fikk vi gjort følgende endringer:

Figur 13: I sprint 3 la vi blant annet til varsling av vilttrekk.

- Fikset bug der telefonen "spammet" forespørsler mot NVDB API om man sto helt stille
- La til funksjonalitet for HUD
- La til noen fareskilt: vilttrekk, høydebegrensning, jernbanekrysning og fartsdemper
- Tilpasset skjermbildet for opptil 5 skilt (7 på iPhone 5)[25]

Av samme grunner som etter sprint 2 valgte vi også her å ikke ta noen testtur. Det var dessuten noen logistikkutfordringer knyttet til fysisk testing (les mer om dette i testdokumentasjonen). Etter denne sprinten begynte vi å se applikasjonen ta form, men vi så at det fortsatt gjensto mye arbeid, og gikk derfor raskt i gang med sprint 4.

[25] iPhone 5 har en 4" skjerm, mens tidligere modeller har 3,5" skjermer.

2.3.4 Sprint 4

Mandag 25.03.13 – fredag 05.04.13

I den fjerde sprinten bestemte vi oss for at det aller meste av funksjonalitet nå måtte implementeres, slik at vi hadde sprint 5 til diverse bugfiksing og finpussing. I løpet av denne sprinten fikk vi gjort følgende arbeid med applikasjonen:

Figur 14: Under testturen etter sprint 4 testet vi blant annet HUD-modus.

- Core Data ble implementert slik at data vi hentet fra NVDB API kunne caches[26] for gjenbruk
- Lydvarsling ble implementert, slik at brukeren varsles når det dukker opp et nytt skilt på strekningen
- Brukerpreferanser ble implementert for de skilt og funksjoner som allerede var implementert
- Avstandsvisning til punktobjekter ble implementert
- Filtersøk mot NVDB API ble implementert for å redusere mengden datatrafikk

Ved slutten av denne sprinten følte vi at det var behov for en ny testtur. Vi følte nå at vi hadde gjort grunnmuren i applikasjonen solid nok, samt at vi hadde nok skilt og funksjonalitet til at det var hensiktsmessig å teste det på en kjøretur. I tillegg var dette viktig med tanke på at neste sprint ville være den siste.

2.3.5 Sprint 5

Mandag 08.04.13 – fredag 19.04.13

Da vi kom til den siste sprinten så vi at vi fremdeles hadde mye arbeid foran oss, og vi måtte derfor arbeide ekstra effektivt her. Vi hadde nå også god kontroll på koden vår, og det var tydelig lettere å utvikle mer funksjonalitet når vi hadde blitt så mye stødigere på språket og miljøet siden første sprint. I denne sprinten fikk vi gjort følgende tillegg og endringer:

- Diverse brukerinnstillinger ble lagt til, som visning av HUD-knappen, lydvarsling og begrensning av lokale data i cache
- Gikk over til MapQuest[27] for å finne avstander. Endret fra Google Maps av hensyn til Googles bruksvilkår[28]

[26] Catching er mellomlagring av data.

- Mange nye skilt ble lagt til (ca. 30 nye skilt) og brukerinnstillinger for disse ble lagt til
- Grafikk for ikon og lasteskjerm ble implementert
- Diverse bugfiksing og små optimaliseringer

Figur 15: Vi laget et eget ikon til applikasjonen som vises på startskjermen på telefonen og i App Store.

Etter denne sprinten tok vi en tredje testtur for å se at det ikke var noen alvorlige feil eller mangler. Det vi fant kan man lese om i testdokumentasjonen, kapittel 2.3. Generelt var vi godt fornøyd med både siste sprinten og produktet etter denne turen.

2.3.6 Utfordringer underveis

Underveis i produksjonsfasen støtte vi på flere utfordringer som gjorde at vi måtte endre eller tilpasse løsningen vår. Noen var vanskelige å løse, noen krevde ikke like mye innsats, mens noen rett og slett ikke lot seg løse. Vi ønsker gjennom dette kapitlet å belyse det faktum at dette ikke har vært en triviell prosess. Selv om applikasjonen vår ikke har flust med funksjonalitet for brukeren å utforske, så er det viktig å være oppmerksom på at det er mye data som skal lagres, behandles og presenteres, og dette krever en god del funksjonalitet og en del mekanismer som vil være skjult for brukeren.

De følgende punktene er omtalt i tilfeldig rekkefølge.

2.3.6.1 Brukerinnstillinger

Da vi begynte arbeidet med brukerinnstillingene oppdaget vi tidlig at denne delen av iOS opererte med en veldig banal løsning.

Når man skal definere innstillinger for en applikasjon i iOS gjøres dette via et enkelt grensesnitt, som så genererer XML. Når man så starter Settings-applikasjonen i iOS, og åpner innstillinger for applikasjonen man har laget, vil Settings-applikasjonen tolke XML'en og presentere innstillingene man har ønsket.

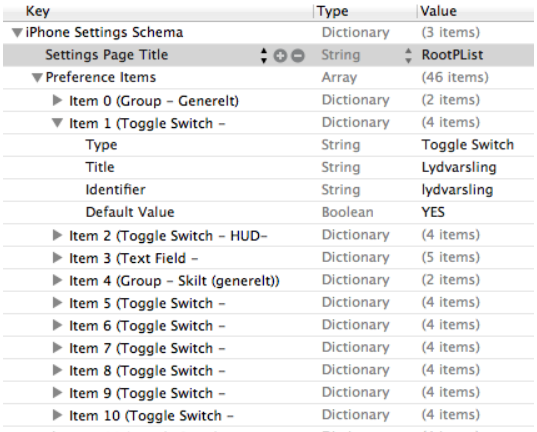
Dessverre legger dette en alvorlig begrensning på innstillings-skjermen, ettersom man ikke kan gjøre den dynamisk på noen måte. Dette la følgende begrensninger på brukerinnstillingene:

- Vi ønsket en knapp som kunne slette lokale data i applikasjonen om man trykket på den. Dette lot seg ikke gjøre. Vi fant ingen løsning på problemet.

[27] MapQuest er en online karttjeneste som tilbyr blant annet vegkart og navigasjonshjelp.
[28] Google krever at deres navigasjonstjeneste benyttes i deres eget kart.

- Vi ønsket en trinnbasert slider for å begrense cache size. Slidere i iOS Settings er trinnløse og viser ikke sin nåværende verdi på noen måte. Vi valgte da å la brukeren definere en cache size limit ved hjelp av et tekstfelt, noe vi selv anser som en stygg løsning.

Vi er klar over at det finnes rammeverk for iOS som kunne løst dette problemet for oss, der man kan lage egendefinerte, dynamiske og iOS-lignende brukerinnstillinger. Grunnet den gitte tidsrammen kunne vi dessverre ikke prioritere å tilegne oss kunnskap om disse rammeverkene, og måtte derfor frastå fra å forsøke å bruke dem.



Key	Type	Value
iPhone Settings Schema	Dictionary	(3 items)
Settings Page Title	String	RootPList
Preference Items	Array	(46 items)
Item 0 (Group – Generelt)	Dictionary	(2 items)
Item 1 (Toggle Switch –	Dictionary	(4 items)
Type	String	Toggle Switch
Title	String	Lydvarsling
Identifier	String	lydvarsling
Default Value	Boolean	YES
Item 2 (Toggle Switch – HUD–	Dictionary	(4 items)
Item 3 (Text Field –	Dictionary	(5 items)
Item 4 (Group – Skilt (generelt))	Dictionary	(2 items)
Item 5 (Toggle Switch –	Dictionary	(4 items)
Item 6 (Toggle Switch –	Dictionary	(4 items)
Item 7 (Toggle Switch –	Dictionary	(4 items)
Item 8 (Toggle Switch –	Dictionary	(4 items)
Item 9 (Toggle Switch –	Dictionary	(4 items)
Item 10 (Toggle Switch –	Dictionary	(4 items)

Figur 16: Det er veldig begrenset hvordan innstillingene til en applikasjon kan se ut. Her fra grensesnittet i Xcode.

2.3.6.2 Avstander

En sentral og veldig viktig del vi ønsket å få med i produktet vårt var avstander til skilt og objekter. Da vi valgte denne oppgaven gikk vi løs på den i den tro at NVDB API inneholdt tilstrekkelig informasjon slik at dette skulle være en enkel oppgave. Dette viste seg raskt å ikke stemme – NVDB API inneholder ingen informasjon om avstander eller lengder på vegstrekninger.

Løsningen ble da å involvere et eksternt API for å håndtere avstander. Da vi først kom til denne løsningen brukte vi Google Maps. Vi fant snart ut at Google krevde at for å bruke APIet deres til å hente ut kjøreavstanden mellom to punkter, så måtte dette brukes sammen med Google Maps. Siden vi ikke gjorde dette, måtte vi finne et annet API for denne jobben Vi endte da med å bruke MapQuest.

Baksiden med denne løsningen er at vi nå må hente informasjon om avstanden asynkront med oppdateringen av skilt og objekter. Dette kan i noen tilfeller føre til at avstandene ikke oppdaterer seg så jevnt og korrekt som vi ønsker.

2.3.6.3 Kobling mellom veglenker

En annen informasjon vi også antok at var en del av NVDB API var referanser mellom veglenker[29], slik at vi alltid kunne sett en viss avstand fremover på vegen. Dessverre fungerer NVDB API slik at når man er på en veglenke, så kan man kun se objekter på den aktuelle veglenken. Dette innebærer at når man befinner seg på de siste hundre meterne av en veglenke, så vil man ikke få informasjon om kommende skilt og objekter, selv om de i noen tilfeller kan ligge noen titalls meter lengre frem.

[29] Et objekt i NVDB som representerer en sammenhengende vegstrekning med en unik ID.

Vi så en mulig løsning i å se på koordinatene til de tre siste punktene på veglenken, for deretter å beregne hvor vegen vil gå videre, slik at vi kunne hentet ut informasjon om veglenken på den beregnede posisjonen. Grunnet den gitte tidsrammen var dette dessverre ikke noe vi kunne prioritere å gjøre.

2.3.6.4 Manglende data i datasettet

Selv om NVDB API er et kraftig verktøy med mye informasjon, viste det seg etter hvert at det var sentrale data som rett og slett manglet for en del områder. For eksempel fant vi ingen forkjørsveger da vi lette i hele Oslo og Akershus. Vi kontaktet Statens vegvesen angående dette, og fikk til svar at mye av informasjonen i NVDB er det lokale avdelinger av Statens vegvesen som har ansvaret for å bidra med. Dette vil altså si at det ikke er NVDB API som har en feil i seg selv, men at Statens vegvesen fremdeles har en stor jobb med å ferdigstille datasettet i NVDB.

Statens vegvesen

Figur 17: Lokale avdelinger i Statens vegvesen er ansvarlige for å bidra med informasjon til NVDB.

Dette var selvsagt ikke et problem vi kunne gjøre noe med, og vi må derfor dessverre godta at applikasjonen ikke kan være bedre enn datasettet den er bygget på.

2.3.6.5 APIet er i demoversjon

Da vi tok på oss dette prosjektet i november/desember 2012 ble vi informert om at APIet foreløpig kun eksisterte som en lukket demoversjon, men at det ikke var noe problem å ta utgangspunkt i denne. Vi ble videre informert om at APIet skulle komme ut for offentligheten hos Statens vegvesen i løpet av desember.

Det viste seg dog at APIet ikke ble frigitt for offentligheten før i mars 2013. I mellomtiden hadde demoversjonen fått ny funksjonalitet for søk, funksjonalitet som vi valgte å benytte oss av fordi den gjorde det mye enklere å foreta søk i databasen.

Da APIet ble frigitt, og vi ble oppmerksomme på at dette var den første demoversjonen vi hadde sett, og ikke den vi nå jobbet mot, tok vi kontakt med BEKK og Statens vegvesen for å forhøre oss om når den versjonen vi jobbet mot ville settes ut i produksjon. Vi fikk da til svar at det burde skje innen en måned, men selv om vi nå er i slutten av mai, jobber vi fremdeles mot en demoversjon som ikke er offentlig tilgjengelig.

Dette har for det meste ikke vært til noe stort hinder for arbeidsprosessen, men satt en stopper for vårt ønske om å få publisert produktet i Apples App Store[30] innen prosjektets frist.

2.3.6.6 Automatisk parsing av JSON for søk

Da vi arbeidet med å automatisere søkeprosessen opprettet vi klasser i Objective-C som skulle settes sammen til en struktur lik den som benyttes for JSON[31]-baserte søk i NVDB API. Vi brukte innebygde funksjoner fra iOS SDK, men fikk ikke dette til å fungere. Vi forsøkt også å traversere søke-objektet og dets objektorienterte egenskaper manuelt, men det fungerte fremdeles ikke.

Etter grundig feilsøking viste det seg at vi hadde truffet på en ukjent bug i Objective-C. Da vi satte sammen klassene slik vi gjorde ble den ene klassen ubrukelig. Vi forsøkte mye frem og tilbake, men det lot seg ikke gjøre å opprette objekter av den aktuelle klassen. Vi fant ingen hjelp for dette på Internett, og vår veileder Christoffer Marcussen fra BEKK hadde aldri sett noe lignende.

Resultatet ble at vi laget en "workaround" hvor vi opprettet JSON-strengen manuelt. Dette ble noe mer tungvint med tanke på å utvide søkeobjektet, men når koden først var skrevet ble resultatet tilfredsstillende, ettersom JSON-strengen for søk ble generert korrekt slik den skulle.

2.3.6.7 Mange objekter langs vegen er ikke objekter i NVDB

Mange av objektene langs vegen vi ønsket å varsle om i applikasjonen eksisterer ikke som objekter i NVDB. Implementasjon av objekter som også er objekter i NVDB var en relativt triviell prosedyre, men vi måtte finne andre løsninger for å varsle om de øvrige objektene.

Figur 18: I applikasjonen er bl.a. "Farlig sving" tolket fra et skiltobjekt, og ikke et eget objekt i NVDB.

Løsningen her ble at vi måtte se på skiltplater i NVDB. Selv om mye ikke er lagt inn som objekter i NVDB er i hvert fall alle skiltplater registrert som skiltplate-objekter. Vi kunne således søke opp de skiltplatene vi hadde behov for og benytte denne informasjonen for å varsle brukere om de aktuelle objektene. Det kan for øvrig nevnes at for å få til dette var vi avhengige av den noe mer avanserte søkefunksjonaliteten som foreløpig ikke er offentlig tilgjengelig, som omtalt i kapittel 2.3.6.5, "APIet er i demoversjon".

2.3.6.8 Nedetid på APIet

Ettersom vi hele tiden jobbet mot en demoversjon av APIet hadde vi ingen garanti på at ting var oppe og kjørte hele tiden. Dette innebar at vi ved enkelte anledninger ble forsinket i produksjonsarbeidet da APIet kunne være utilgjengelig i opptil flere timer. I kortere perioder kunne

[30] Apple App Store er Apples distribusjonsplattform for applikasjoner til iOS.
[31] JavaScript Object Notation, en enkel tekstbasert standard for datautveksling.

vi også oppleve at APIet svarte ekstremt tregt, noe som gjorde at simulering av applikasjonen ble en langtrukken prosess.

Ettersom dette var utenfor vår kontroll var det ikke annet vi kunne gjøre enn å kontakte Frode Reinertsen hos BEKK for å høre om han kunne forbedre situasjonen.

2.3.6.9 Upresis bounding-box

Under testturen etter første sprint oppdaget vi at applikasjonen hadde problemer med å "holde seg på vegen". Da det var sideveger eller paralleltgående veger virket det nesten tilfeldig hvilken veg APIet trodde vi var på.

Dette var en problemstilling vi tok opp med Frode Reinertsen hos BEKK, og han utførte så forbedringer rundt dette i APIet. I tillegg kom det ny funksjonalitet i demoversjonen av APIet. Da vi utforsket og tok i bruk denne funksjonaliteten, samtidig som vi forbedret våre egne metoder for posisjon- og veglenkesøk, fikk vi økt presisjonen betraktelig.

Selv etter de ovennevnte forbedringene ser vi fremdeles at det er vanskelig for APIet å vite nøyaktig hvilken veg vi er på dersom det er flere veger innenfor en radius på noen få meter, eller dersom man kjører over/under en bro. Selv på vår siste testtur, som varte i ca. 2 timer, var dette et relativt sjeldent problem.

2.3.7 Apps4Norge

Apps4Norge var en konkurranse i regi av Difi og IKT-Norge. Konkurransen gikk ut på å lage en applikasjon som benytter eller er basert på åpne, offentlige data. Konkurransen ble arrangert for å få fokus på de mange nye offentlige datasettene, fra offentlige etater, som har blitt lansert i 2012/2013.

I starten av april ble vi tilfeldigvis tipset om konkurransen av Christoffer Marcussen i BEKK, og vi kikket litt på vilkår og regler for å delta. Da vi oppdaget at applikasjonen vår faktisk var midt i blinken for denne konkurransen, ettersom vi benytter åpne, offentlige data, bestemte vi oss for å melde oss på. Etter dette arbeidet vi videre som normalt, og beholdt fokuset på at prosjektet skulle gjennomføres etter de betingelser og vilkår vi hadde tatt sikte på fra starten av. Den eneste forandringen vi gjorde i prosjektet på

Figur 19: Kjørehjelperen vant 3. plass i lagkonkurransen i Apps4Norge (Foto: Cecilie Vaagen Smestad, Statens vegvesen).

grunn av konkurransen, var å fremskynde den siste testturen med noen få dager, slik at vi kunne få laget en god demovideo.

Da vi deltok på prisutdelingen 8. mai var overraskelsen derfor stor da vi gledelig nok ble tildelt 3. plass i kategorien for applikasjoner laget av grupper. I ettertid har applikasjonen vår og konkurransen blitt omtalt på nettsidene til bl.a. Statens vegvesen[32], Høgskolen i Oslo og Akershus[33], IKT-Norge[34] og Teknisk Ukeblad[35].

2.4 Dokumentasjonsfasen

Etter at produksjonsfasen var endt, og produktet var ferdig utviklet, var det klart for å dokumentere arbeidet. Vi satte av ca. en måned til denne fasen, og begynte på den mandag 29. april, med innleveringsfrist for prosjektet tirsdag 28. mai.

I likhet med arbeidet med styringsdokumentene, arbeidet vi også under dokumentasjonsfasen med dokumentasjonsstandarden[36] til Ann-Mari Torvatn som en veiledning, uten at den ble fulgt slavisk. Også her var dette med overlegg, ettersom det var aspekter vi følte burde omstruktureres, kuttes eller legges til. Også disse avvikene ble gjort i samråd med veileder Eva Vihovde.

[32] http://www.vegvesen.no/Om+Statens+vegvesen/Media/Siste+nyheter/Vis?key=472208
[33] http://www.hioa.no/Aktuelle-saker/Fikk-premie-for-kjoerehjelper
[34] http://ikt-norge.no/2013/05/vinnere_apps4norge/
[35] http://www.tu.no/it/2013/05/08/norges-mest-lovende-teknologier
[36] For detaljer om dokumentasjonsstandarden se kapittel 1.4.1.1, "Dokumentasjonsstandard".

3 Kravspesifikasjonen og dens rolle

I denne delen vil vi beskrive rollen kravspesifikasjonen har hatt under utviklingsarbeidet med produktet vårt. Vi vil også se nærmere på hvilke krav og rammer vi har klart å oppfylle, hvilke områder vi har avvik fra kravspesifikasjonen, samt hvilke utvidelser eller endringer vi har gjort i kravspesifikasjonen underveis.

3.1 Kravspesifikasjonens rolle

Kravspesifikasjonen skal fungere som et styringsdokument for hovedprosjektet. Den skal definere rammer og retningslinjer for prosjektet, utarbeides i samarbeid med oppdragsgiver, og sikre at både vi og oppdragsgiver får klarlagt hva betingelsene for prosjektet skal være. Dette gjelder både funksjonelle og ikke-funksjonelle krav, samt krav til arbeidsprosess og kvalitetssikring.

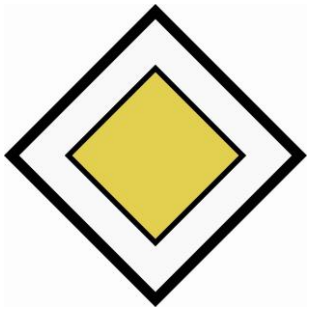
Kravspesifikasjonen skal opptre som en bindende avtale mellom oss og oppdragsgiver, men det skal også tas hensyn til krav fra Høgskolen i Oslo og Akershus. Kravspesifikasjonen, og alle revisjoner, skal derfor være godkjent av alle parter før det kan anses som gyldig.

3.2 Samsvar med kravspesifikasjonen

I dette kapittelet tar vi for oss alle kravene i kravspesifikasjonen, og diskuterer i hvilken grad disse har blitt oppfylt. Kravspesifikasjonen det refereres til her er den endelige kravspesifikasjonen, som er vedlagt som et eget dokument i denne oppgaven. Endringer som er gjort i kravspesifikasjonen underveis kan man lese om i kapittel 3.4.

3.2.1 Funksjonelle krav

Vi delte opp de funksjonelle kravene[37] i 3 deler: Prioritert funksjonalitet, ønsket funksjonalitet og eventuell tilleggsfunksjonalitet. Dette var for å gi en pekepinn på hvordan vi prioriterte de forskjellige forslagene til funksjonalitet. Underveis har funksjonalitet blitt omprioritert, flyttet og lagt til.



Figur 20: Dette skiltet vises på skjermen hvis brukeren kjører på en forkjørsveg.

Brukeren skal kunne se gjeldende fartsgrense for vegstrekningen

Så lenge det er registrert en fartsgrense på vegen brukeren befinner seg på i NVDB vises denne på skjermen. Fartsgrensen vises som et fartsgrenseskilt.

[37] Funksjonelle krav er krav til funksjonalitet som skal implementeres i systemet.

Brukeren skal kunne se hvorvidt han/hun er på en forkjørsveg eller ikke

Hvis brukeren er på en veg som er registrert som en forkjørsveg i NVDB vil det vises et forkjørsvegskilt på skjermen. Hvis vegen ikke er en forkjørsveg vil det derfor heller ikke vises noe forkjørsvegskilt. Manglende forkjørsvegskilt kan dog også skyldes manglende data i NVDB. Hvis brukeren er på en motorveg eller motortrafikkveg[38] vil disse skiltene vises i stedet for forkjørsvegskiltet.

Brukeren skal kunne se diverse gjeldende varselskilt

Brukeren blir presentert følgende fareskilt:

- Vilttrekk
- Farlig sving
- Smalere veg
- Steinsprut
- Farlig vegskulder
- Tunnel
- Trafikklyssignal
- Syklende
- Møtende trafikk
- Sidevind
- Annen fare
- Fartsdemper
- Farlige svinger
- Ujevn veg
- Rasfare
- Bevegelig bru
- Farlig vegkryss
- Avstand til gangfelt
- Ku
- Kø
- Skiløpere
- Jernbanekryssing
- Bratt bakke
- Vegarbeid
- Glatt kjørebane
- Kai, strand eller ferjeleie
- Rundkjøring
- Barn
- Sau
- Fly
- Ridende

Figur 21: Dette skiltet varsler om glatt kjørebane.

I tillegg blir brukeren presentert for:

- Høydebegrensning
- Videokontroll/-overvåkning
- Automatisk trafikkontroll
- Særlig ulykkesfare

Figur 22: Dette skiltet brukes til å varsle om nødtelefoner langs vegen. Det benyttes ikke i applikasjonen, men var inspirasjonen til skiltet som varsler om SOS-lommer.

Brukeren skal kunne endre hvilken informasjon som skal vises

Alle vegobjektene som støttes i applikasjonen kan deaktiveres i applikasjonens innstillingsmeny.

Brukeren skal kunne endre hvorvidt det brukes lyd for å varsle brukeren

Lydvarsling kan deaktiveres i applikasjonens innstillingsmeny.

Figur 23: Uoffisielt skilt som brukes til å varsle om SOS-lommer i applikasjonen.

Informasjonen på skjermen skal kunne speilvendes med det formål å lage et HUD i frontruten

Head-up display er implementert. En av/på-knapp speilvender informasjonen på skjermen og låser orienteringen til landskapsmodus.

Brukeren skal kunne se avstand til førstkommende rasteplass

Førstkommende rasteplass varsles med et rasteplasskilt og avstanden dit.

[38] Motorveg og motortrafikkveg er de nye navnene på henholdsvis motorveg klasse A og motorveg klasse B.

Brukeren skal kunne se avstand til førstkommende SOS-lomme

Førstkommende SOS-lomme varsles med et serviceskilt med bokstavene SOS. Dette er ikke et offisielt skilt. Avstanden dit står under eller ved siden av.

Brukeren skal kunne se avstand til nærmeste toalett (rasteplass eller bensinstasjon)

Førstkommende toalett varsles med et serviceskilt med bokstavene WC. Avstanden dit står under eller ved siden av. Det opplyses kun om toaletter forvaltet av Statens vegvesen, da NVDB ikke inneholder data om bensinstasjoner.

3.2.2 Ikke-funksjonelle krav

De ikke-funksjonelle kravene[39] er delt opp i produktkrav, prosesskrav og eksterne krav.

3.2.2.1 Produktkrav

Produktkrav omhandler krav til det ferdige produktet som ikke er en direkte funksjon. Dette omhandler krav til brukervennlighet, effektivitet og pålitelighet.

Applikasjonen skal være på norsk

Dette kravet er oppfylt.

Applikasjonen skal i så stor grad som mulig følge de 5 E'ene

Vi vil påstå at den gjør det[40]. Grunnet at applikasjonen kun består av ett skjermbilde, begrenses mulighetene til å bryte disse prinsippene betraktelig. Les mer om dette i produktdokumentasjonen, kapittel 5.3: "De fem E'ene".

Applikasjonen skal ikke kreve interaksjon under bruk (kjøring)

Første gang man starter applikasjonen må man godkjenne at den får bruke telefonens posisjonstjeneste, samt at man må godta en ansvarsfraskrivelse for bruk. Hvis telefonen ikke klarer å finne posisjonen under bruk kommer det dessuten opp en dialogboks[41] som informerer om dette. Utenom dette krever applikasjonen ingen interaksjon.

Applikasjonen skal begrense egen datatrafikk i så stor grad som mulig

Applikasjonen mellomlagrer data på enheten. Hvis applikasjonen ser at den er på en veg den har vært på før, og dataene ikke er eldre enn 30 dager, bruker den de mellomlagrede dataene. Dette sparer mye datatrafikk, siden enkelte vegstrekninger inneholder mye data.

[39] Ikke-funksjonelle krav er krav til systemets egenskaper og rammeverk.
[40] Prinsipper for brukervennlighet: effective, efficient, engaging, error tolerant og easy to learn (Stone, Jarret, Woodroffe, Minocha, "User Interface Design and Evaluation", 2005, Elsevier Inc.).
[41] Liten boks med informasjon som legger seg over skjermbildet. Man må trykke på ”OK” for at den skal forsvinne.

Figur 24: Applikasjonen skal begrense datatrafikk i så stor grad som mulig (Foto: Jollygoo.com).

Hvilken veg man befinner seg på er det NVDB som avgjør ved hjelp av telefonens GPS-koordinater. Det må derfor gjøres en spørring mot serveren for hver oppdatering. I tillegg må det spørres mot MapQuest for hver oppdatering for å beregne avstand til punktene lenger fremme. Det vil derfor brukes noe datatrafikk hele tiden, men mellomlagringen av data reduserer datatrafikken med 95%.

Applikasjonen skal være så energigjerrig som mulig

Det er ikke gjort noen egne tiltak for å begrense strømbruken. Telefonen sørger selv for å skru av tjenester når applikasjonen minimeres. Når den er i bruk benyttes den høyeste graden av nøyaktighet ved posisjonering. Dette krever en del strøm, men er nødvendig for at posisjonen skal være nøyaktig nok. Apple anbefaler at telefonen er koblet til en strømkilde når denne tjenesten benyttes. Vi anslår allikevel at applikasjonen kan kjøre i omtrent 5 timer på et fulladet batteri[42].

Applikasjonen skal begrense bruk av telefonens minne

Det er under implementasjonen forsøkt å designe applikasjonen på en måte som gjør at det ikke befinner seg store mengder data i minnet til enhver tid. Det brukes i stor grad arrays[43] og dictionaries[44] til å holde på informasjon istedenfor store objekter.

Med introduksjonen av ARC[45] i iOS 5[46] overføres dessuten minnehåndteringen til kompilatoren[47]. Vi trengte derfor ikke bekymre oss for minnelekkasjer[48] i noen stor grad.

Posisjonering av bruker skal være innenfor en nøyaktighet på 10 meter

Ved bruk av telefonens lokasjonstjeneste med høyeste nøyaktighet leverer den posisjoner med 5 meters nøyaktighet.

Posisjonering av objektdata skal være innenfor en nøyaktighet på 1 meter

Vi har ikke utført noen konkrete tester på hvor nøyaktig posisjonene i NVDB er, men ut i fra eget inntrykk er posisjonene langt mer nøyaktige enn 1 meter.

---

[42] Basert på egen testing på en iPhone 4S.
[43] Et array er en endimensjonal tabell.
[44] En dictionary er en datastruktur bestående av nøkkel- og verdipar.
[45] Automatic Reference Counting (ARC) er at ansvaret for minnehåndtering flyttes fra programmereren til kompilatoren.
[46] iOS er Apples operativsystem for iPhone, iPad og iPod.
[47] En kompilator er et dataprogram som oversetter et dataprogram fra kildekode til maskinkode.
[48] Feil ved minnehåndteringen i programmet.

Dataene som presenteres skal være oppdaterte

Hvis de mellomlagrede dataene på telefonen er eldre enn 30 dager, oppdateres dataene. Hvorvidt dataene i NVDB er oppdatert eller ikke har vi ingen kontroll over.

3.2.2.2 Prosesskrav

Prosesskrav omhandler krav til prosessen rundt utviklingen av produktet, deriblant metodikk, leveranser, implementasjon og standarder.

Applikasjonen skal være kjørbar på iPhone 3GS, 4, 4S og 5 (oppdatert til iOS 6.1)

Ved å utvikle applikasjonen til iOS 6.1 vil den være kjørbar på alle enheter med dette operativsystemet. iPhone 3GS og nyere støtter iOS 6.x.

Figur 25: Applikasjonen skal kunne kjøre på en iPhone 3GS eller nyere.

Kildekoden for leveransen skal lisensieres under GNU GPL[49] og være tilgjengelig via GitHub

Kildekoden er lisensiert under GLP 3 og er tilgjengelig på https://github.com/lsmeby/Vegdata.

Applikasjonen skal utvikles for iOS 6.1

Applikasjonen skal utvikles med Xcode 4.6 på OS X

Objective-C benyttes som programmeringsspråk

Applikasjonen skal benytte Cocoa Touch og de andre rammeverkene i iOS 6.1 SDK

Disse kravene er selvskrevne og kan ikke omgås. Ved utvikling av applikasjoner til iOS tvinges du til å bruke Xcode på OS X, der du igjen må bruke overnevnte rammeverk og programmeringsspråk. Vi brukte nyeste versjon av Xcode og iOS SDK, som på det tidspunktet var henholdsvis 4.6 og 6.1.

Applikasjonen skal testes på så mange fysiske iOS-enheter som mulig

Per i dag er applikasjonen kun testet på en iPhone 4S i tillegg til simulatoren i Xcode. Det planlegges testing på flere enheter før endelig lansering i App Store.

Applikasjonen skal benytte det åpne biblioteket RestKit for kommunikasjon og parsing av data med NVDB APIet

Applikasjonen benytter RestKit. Se kapittel 3.2 i produktdokumentasjonen for mer informasjon om RestKit.

[49] GNU General Public License (GNU GPL eller GPL) er en lisens som tillater fri bruk, kopiering og endring av koden.

Brukerpreferanser skal lagres som "User preferences"

Vegdata skal lagres som Core Data

Disse kravene er innfridd. Se "3.1 Core Data" og "4.2.12 Settings" i produktdokumentasjonen for detaljer rundt lagring av data.

Det skal benyttes JSON for kommunikasjon med NVDB API

Datautveksling med både NVDB og MapQuest kan gjøres med JSON og XML. Kjørehjelperen benytter JSON. Les mer om dette i første kapittel i produktdokumentasjonen.

Applikasjonen skal benytte MVC-patternet (som forventes i iOS)

Strukturen i et iOS-prosjekt er i utgangspunktet basert på MVC-patternet. Vi har utvidet dette ytterligere ved å gjøre en enda tydeligere lagdeling. Les mer om dette i produktdokumentasjonen, kapittel 2.

3.2.2.3 Eksterne krav

Eksterne krav omhandler krav som ikke passer inn under produkt- eller prosesskrav, deriblant estetiske og lovmessige krav.

Applikasjonen skal være i samsvar med Apples retningslinjer for design på iOS

Retningslinjene[50] er til dels subjektive, men det er forsøkt å følge disse så godt som mulig. Les mer om dette i produktdokumentasjonen under kapittel 5.2.

Dataene som presenteres skal være store nok til å kunne sees på 1 meters avstand

Dataene er godt synlige for en person med normalt syn på 1 meters avstand.

Det skal i så stor grad som mulig benyttes ikoner og grafikk fremfor tekst

Det benyttes skilt til å varsle om ulike farer og hendelser. Avstander må naturlig nok oppgis som tekst.

Designet skal følge de 7 designprinsippene

Vi mener designet følger prinsippene[51]. Les en vurdering av dette i kapittel 5.1 i produktdokumentasjonen.

[50] https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

[51] Structure, simplicity, consistency, tolerance, visibility, affordance og feedback (Stone, Jarret, Woodroffe, Minocha, "User Interface Design and Evaluation", 2005, Elsevier Inc.).

Applikasjonen skal ikke kreve interaksjon under kjøring som strider mot norsk vegtrafikklov

Applikasjonen krever i utgangspunktet ingen interaksjon under kjøring. Les mer om dette i testdokumentasjonen, kapittel 1.

Applikasjonen skal inneholde teksten: "Inneholder data under norsk lisens for offentlige data (NLOD) tilgjengeliggjort av Statens vegvesen."

Applikasjonen skal opplyse om, og avskrive seg selv og Statens vegvesen ansvar for, eventuelle feil og mangler som måtte forekomme i NVDB APIet

Ved første gangs oppstart vises overnevnte tekst, en link til MapQuest og en kort ansvarsfraskrivelse. Ansvarsfraskrivelsen gjelder både for Kjørehjelperen selv, samt data fra NVDB og MapQuest.

Figur 26: Denne meldingen vises første gang Kjørehjelperen starter.

3.3 Avvik fra kravspesifikasjonen

Vi vil her ta for oss de krav som vi ønsket å oppfylle, men som av diverse grunner ikke har blitt oppfylt. Vi vil også forklare kort hvorfor det aktuelle kravet ikke ble oppfylt.

Brukeren skal kunne endre hvor tidlig varsler skal vises

Dette kravet ble vanskelig å innfri. Grunnene for dette handler dels om det omtalt i kapittel 2.3.6.1, "Brukerinnstillinger", men i hovedsak handler det om utfordringen omtalt i kapittel 2.3.6.3, "Kobling mellom veglenker".

Den ferdige applikasjonen skal være godkjent og tilgjengelig i Apples App Store

Det er to vesentlige grunner til at dette kravet ikke har blitt innfridd. Den mest konkrete grunnen er at NVDB API enda ikke er offentlig tilgjengelig i den versjonen vi har arbeidet mot. Les mer om dette i "2.3.6.5 APIet er i demoversjon".

Den andre grunnen er at selv om vi er godt fornøyde med arbeidet vi har gjort, og synes vi har arbeidet godt gjennom hele prosessen, så er vi begge av den oppfatning at applikasjonen har behov for litt finpussing før vi stolte kan lansere den for verden. Les mer om dette i kapittel 4.3, "Videre utvikling".

3.4 Endringer i kravspesifikasjonen

Selv om noen avvik fra kravspesifikasjonen ikke er til å unngå, har vi til en viss grad forsøkt dette ved å revidere kravspesifikasjonen underveis. Disse endringene ønsker vi også at skal være omtalt i denne delen av dokumentasjonen.

Brukeren skal kunne se avstand til neste mulige forbikjøring

Denne funksjonaliteten ble droppet både på grunn av dårlig datagrunnlag i NVDB, og at det datagrunnlaget som fantes gjaldt noen få skiltede forbikjøringsfelt. Vi følte at dette ikke ville gi noe særlig nytteverdi til brukeren.

Brukeren skal kunne se hva klokka er

Dette var funksjonalitet vi anså som relativt lite viktig, ettersom de fleste har andre klokker tilgjengelig i bilen, og kravet frafalt derfor underveis.

Brukeren skal kunne se et varsel ved overtredelse av fartsgrense

Dette kravet ble fjernet fordi det var funksjonalitet vi anså som relativt lite viktig. Vi ønsket å prioritere fokuset vårt mot den funksjonaliteten vi anså som mest relevant slik at vi kunne spisse inn applikasjonen mot å være en skilt-påminnelses-applikasjon. Vi ville dessuten bruke all tilgjengelig plass på skjermen til skilt.

Brukeren skal kunne se kommende bomstasjoner på vegen

Dette kravet ble i hovedsak frafalt fordi det ville være lite hensiktsmessig for brukere av applikasjonen å bli varslet om dette. Som beskrevet i 2.3.6.3, "Koblinger mellom veglenker", var det vanskelig å kunne se veldig langt fremover på en vegstrekning. Ettersom vi da ikke kunne varslet sjåføren før det var et par kilometer igjen til bomstasjonen, følte vi at dette var noe sent å varsle om en bomstasjon.

Det skal være mulig å få opp utvidet informasjon om fasiliteter ved rasteplasser

Dette kravet ble droppet av samme grunn som vi droppet kravet om varsel ved overtredelse av fartsgrense: denne funksjonaliteten var noe avvikende fra kjernen i applikasjonen, og var derfor påtenkt som en mulig utvidelse dersom vi fikk god tid.

Brukeren skal kunne se avstand til førstkommende bensinstasjon

Da vi formulerte dette kravet var det en ren teoretisk mulighet dersom vi fikk veldig god tid. Ettersom bensinstasjoner ikke er en del av datasettet i NVDB ville dette krevd innhenting av data fra andre hold, og ville derfor vært en større oppgave å utføre. Kravet ble derfor fjernet.

Varsler skal kunne dikteres for brukeren

Dette kravet omhandlet funksjonalitet som vi anså som en spennende mulighet. Allikevel var vi fullt klar over, da vi formulerte det, at det ville vært en stor jobb. Kravet ble derfor skrevet ned for å sikre at vi hadde tenkt ut nok funksjonalitet til å unngå at vi fikk for lite å gjøre i prosjektet. Det ble etter hvert tydelig at vi hadde planlagt veldig mye arbeid for oss selv, og dette kravet ble derfor fjernet.

Det skal leveres en kjørbar applikasjon ved hver leveranse

Det ble gjort en endring i detaljene til dette kravet: Den ene leveransedatoen ble endret fra 12.04 til 05.04, dette valgte vi fordi vi så et behov for å også jobbe i påskeferien, for å få mer tid til utviklingen.

4 Avsluttende del

Etter et langt, utfordrende og spennende prosjekt har vi her kommet til slutten på dette eventyret, så i denne delen vil vi gjøre oss noen tanker og oppsummeringer som vil være naturlige å gjøre ved slutten av et prosjekt.

4.1 Vurderinger av data og teknologi

I prosjektet har vi benyttet nye verktøy og data, som vi hadde som et mål at vi skulle vurdere.

4.1.1 NVDB

NVDB inneholder mye data, noe som gir mange muligheter til implementasjon og bruk av APIet. Noe vi riktig nok savner i APIet er avstander og lengder på veglenker. I Kjørehjelperen har vi måttet ty til en løsning der vi benytter MapQuest for å finne avstanden til kommende objekter. Hvis NVDB hadde oppgitt lengden på veglenkene, hadde vi både fått en smidigere og mer nøyaktig avstandsangivelse i applikasjonen.

En annen ting vi savnet i NDVB API var en sammenheng mellom veglenkene. Slik NVDB fungerer i dag finnes det ingen direkte link mellom veglenker som henger sammen, og vi kan derfor ikke på noen enkel måte forutse hva som dukker opp på kommende veglenker.

Selv om NVDB inneholder over 250 objekttyper, skulle vi gjerne sett at det var noen flere. Flesteparten av objektene langs vegen som vi benytter i Kjørehjelperen eksisterer ikke som egne objekter i NVDB, men som et skiltplate-objekt som vi måtte tolke og konvertere til egne objekter. Dette er en løsning som fungerer for øyeblikket, men er noe svak siden den utelukkende baserer seg på sammenligning av tekststrenger.

4.1.2 iOS

Etter at vi begge tidligere har jobbet med både Android[52] og Windows Phone[53], var det en spennende utfordring å jobbe med iOS. Vi ser flere likheter mellom plattformene i hvordan prosjektet struktureres, men også vesentlige ulikheter, da spesielt syntaksen i Objective-C.

iOS og Cocoa Touch-rammeverket bygger på gamle C-klasser og kan derfor oppleves som noe utdatert og ustrukturerte. For øvrig ser vi at Apple har lagt mye arbeid i å bygge et solid SDK som

[52] Android er et operativsystem til mobile enheter basert på Java. Android utvikles av Google.
[53] Windows Phone er Microsofts operativsystem til smarttelefoner.

enkelt gir tilgang til telefonens hardware og egenskaper. Vi var også godt fornøyd med delegater[54], og hvordan dette benyttes for å kommunisere asynkront innad i applikasjonen.

4.2 Nytteverdi

I dette delkapittelet vil vi se nærmere på hvilken nytteverdi prosjektet og produktet vil ha for sentrale parter, utenom oss som har utført arbeidet. Vi vil se nærmere på vårt eget utbytte av dette i kapittel 4.3.

4.2.1 For brukere av produktet

Brukere av produktet er helt klart den parten i dette prosjektet som vil ha størst nytteverdi av produktet i ettertid. Brukere har ikke vært en direkte interessent i prosjektet, men ettersom de er målgruppen må de også være å anse som en sentrale.

I retrospekt, føler vi at vi har produsert et godt produkt der vi har oppnådd vårt mål om et allment nyttig produkt. Anerkjennelsen vi fikk gjennom vår plassering i konkurransen Apps4Norge utgjør også en ekstra trygghet om at vi har utviklet et solid og nyttig produkt. Vi velger derfor å konkludere med at brukere av produktet vil kunne ha stor nytte av det i ettertid av dette prosjektet.

4.2.2 For oppdragsgiver

Ettersom oppdragsgiver tradisjonelt ikke produserer kommersielle produkter av denne typen vil ikke oppdragsgiver ha noen nytteverdi av produktet i ettertid. Dette har heller ikke vært noe mål for dem, og må derfor ikke anses som noe negativt.

Derimot kan selve prosessen rundt prosjektet ha en viss nytteverdi for oppdragsgiver i ettertid. Vi antar ikke at BEKK som firma vil kunne dra spesiell nytte av dette prosjektet, men det vil være rimelig å anta at erfaringen for de involverte personene hos BEKK kan være nyttig å ta med seg videre. Vi håper også at denne dokumentasjonen kan være til nytte for oppdragsgiver ved en senere anledning.

BEKK har riktignok fått både oppmerksomhet rundt og en nyttig testing av APIet de har utviklet for Statens vegvesen, noe som utvilsomt er noe de kan ta med seg videre.

4.2.3 For kunde

Til forskjell fra mange andre oppgaver har ikke denne handlet om å utvikle et produkt som skal brukes av kunden, altså Statens vegvesen. Derimot har denne oppgaven handlet om å synliggjøre

[54] En delegat i Objective-C er en klasse som implementerer en gitt protokoll. En annen klasse kan kalle denne delegaten uten å vite noe om hvilken klasse den kaller. Les mer om delegater i produktdokumentasjonen, kapittel 4.1.4.

mulighetene i et allerede eksisterende produkt, NVDB API. I lys av disse forutsetningene kan vi så vurdere nytteverdien dette produktet vil ha for Statens vegvesen i ettertid.

For å vurdere nytteverdien av produktet for kunden vil vi se på de samme argumentene som vi så på under vurderingen av nytteverdien for brukere. For det første føler vi selv at vi har utviklet et solid produkt. I tillegg kan vi med sikkerhet si at anerkjennelsen vi fikk gjennom vår plassering i Apps4Norge utgjør en god publisitet for Statens vegvesen og deres API med åpne data, NVDB. De har også fått erfaringer og tilbakemelding på hvordan det er å bruke APIet deres. Vi vil derfor konkludere med at kunden i ettertid vil ha god nytteverdi av produktet vi har utviklet og publisiteten det skaper for NVDB.

4.3 Videre utvikling

På kort sikt er det ikke store endringer vi ser for oss at det er behov for i applikasjonen. Vi er godt fornøyd med funksjonalitet og stabilitet, men ønsker allikevel å utføre noen endringer før vi vil anse produktet nå som en god førsteversjon, med tanke på lansering. I det vi skriver denne dokumentasjonen benytter applikasjonen vår fremdeles en demoversjon av APIet, og det kan derfor ikke publiseres for allmennheten i Apples App Store. Når denne demoversjonen blir satt i produksjon har vi som mål å finpusse på applikasjonen og få den publisert. Les mer om dette i produktdokumentasjonen, kapittel 6: "Gjenstående arbeid".

4.4 Læringsutbytte

I løpet av dette semesteret og dette prosjektet føler vi begge at vi har fått kunnskap og erfaring innen flere områder. Vi ser at prosessen har utviklet oss, og modnet oss med tanke på å være forberedt på arbeidslivet. Ikke bare har dette vært et omfattende prosjekt som har tatt tid å gjennomføre, men vi har også hatt et ansvar med å sette mål og rammer for prosjektet. Det har også vært en viktig prosess med å planlegge og strukturere tiden, der vi har sett hvor utfordrende det kan være å ha frister å overholde og leveranser man skal rekke. Sist, men ikke minst, har det også vært en verdifull erfaring å ha en konkret oppdragsgiver fra arbeidslivet, noe som har gitt oss god innsikt i hvordan et prosjekt gjennomføres i "den virkelige verden".

I tillegg til den viktige erfaringen med prosjektarbeid og prosjektstyring, har vi også fått et stort faglig læringsutbytte av dette prosjektet. Som nevnt i "1.1 Arbeidsforhold og samarbeid" ønsket vi begge "å utvide kunnskapsplattformen vår" samt å "utfordre oss selv i hovedprosjektet". Dette var altså et bevisst valg fordi vi ønsket at hovedprosjektet skulle være noe mer enn å bare gjøre noe vi allerede hadde lært på studiet; vi ønsket å lære mer, og det gjorde vi ved å lære oss et nytt programmeringsspråk, et nytt rammeverk, et nytt utviklingsmiljø og et nytt operativsystem! Vi er

derfor stolte av å se tilbake på dette prosjektet der vi ikke bare utviklet et produkt vi er stolte av, men også tilegnet oss mye ny faglig kunnskap som vil gi oss tyngde og bredde når vi skal ut i arbeidslivet.

4.5 Konklusjon

Vi har nå gjennomført et omfattende prosjekt der vi både har fått erfare hvordan slikt skal gjennomføres profesjonelt i arbeidslivet, men også tilegnet oss mye ny kunnskap og erfaring. Vi har utfordret oss selv, både faglig og med tanke på omfang, og kommet frem til et solid og anerkjent produkt. Derfor er vi veldig fornøyde når vi nå har kommet til vegs ende, og vi kan se tilbake på prosjektet.

Når vi nå kan reflektere over denne prosessen i retrospekt så ser vi at dette ikke bare har vært et semester og et prosjekt, men heller en kulminasjon av en tre år lang prosess med læring. Grunnen til at vi ser på det slik er at vi her har fått bruk for alt vi har lært underveis i studiet, både programmeringsferdigheter, evnen til å lære seg nye språk, prosjektplanlegging, visuelt design, interaksjonsdesign, og mye, mye mer.

På bakgrunn av graden av måloppnåelse, produktets anerkjennelse, samt vår egen tilfredshet med både prosess, produkt og egen utvikling, vil vi si at prosjektet har vært svært vellykket.

2013

Kjørehjelperen

Produktdokumentasjon

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Produktbeskrivelsen inneholder en beskrivelse av ”Kjørehjelperen”. De første kapitlene redegjør for ulike teknologier og rammeverk som er benyttet. Deretter gjennomgås programmet i detalj. Her går vi inn på hvordan programmet er bygget opp, hvordan kommunikasjonen foregår internt i programmet og hvilke datastrukturer som benyttes. Til slutt ser vi på det grafiske brukergrensesnittet i applikasjonen.

Denne delen av dokumentasjonen er skrevet for en person med kompetanse innen programmering og systemutvikling. Dokumentasjonen er ment som et oppslagsverk for å gi en utenforstående innsikt i applikasjonens oppbygning og funksjonalitet. Ved hjelp av produktdokumentasjonen skal vedkommende kunne fortsette arbeidet med applikasjonen, både vedlikehold og videre utvikling. Dokumentasjonen er derfor meget teknisk.

Ord og uttrykk forklares fortløpende i fotnoter. I tillegg er samtlige fremmedord og forkortelser samlet i dokumentet "Ordforklaringer og kilder" bakerst i rapporten.

Innholdsfortegnelse

1 Datasett

Data i Kjørehjelperen hentes i hovedsak fra Nasjonal vegdatabank, NVDB, forvaltet av Statens vegvesen. I tillegg benyttes data fra MapQuest[1] til å bestemme avstanden til kommende objekter. I de følgende punktene tar vi for oss innholdet og strukturen i disse datasettene.

1.1 JSON

JavaScript Object Notation, kalt JSON, er en enkel tekstbasert standard for datautveksling. Den er opprinnelig avledet fra JavaScript for å representere enkle datastrukturer. Standarden er imidlertid uavhengig av JavaScript eller andre programmeringsspråk. (…) JSON-formatet brukes ofte til serialisering og overføring av strukturert data over en nettverkstilkobling, primært mellom en server og en webapplikasjon.[2]

JSON brukes ofte som et alternativ til XML[3]. Data kodet i JSON er generelt mindre enn tilsvarende data kodet i XML, blant annet på grunn av XML's avsluttende tagger. I tillegg mener mange at JSON parses[4] raskere enn XML.

```
{
  "firstName" : "John",
  "lastName" : "Smith",
  "age" : 25,
  "address": {
    "streetAddress" : "21 2nd Street",
    "city" : "New York",
    "state" : "NY",
    "postalCode" : "10021"
  },
  "phoneNumber" : [
    {
      "type" : "home",
      "number" : "212 555-1234"
    },
    {
      "type" : "fax",
      "number" : "646 555-4567"
    }
  ]
}
```

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>
      21 2nd Street
    </streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">
      212 555-1234
    </phoneNumber>
    <phoneNumber type="fax">
      646 555-4567
    </phoneNumber>
  </phoneNumbers>
</person>
```

Kodesnutt 1: Data formatert som JSON til venstre, og som XML til høyre. Hvis vi fjerner unødvendige mellomrom og linjeskift (brukt her for å gjøre koden lettleselig) består JSON-formatet av 240 tegn mens XML-formatet består av 344 tegn.

[1] MapQuest er en online karttjeneste som tilbyr blant annet vegkart og navigasjonshjelp.
[2] Hentet fra artikkelen om JSON på Wikipedia.org (http://no.wikipedia.org/wiki/JSON).
[3] Extensible Markup Language (XML) er et universelt og utvidbart markeringsspråk. Brukes til deling av strukturerte data mellom informasjonssystemer og til koding av dokumenter og som kommunikasjonsmiddel mellom ulike informasjonssystemer og dataformater.
[4] Prosessen med å generere eller lese data fra (i dette tilfellet) en JSON- eller XML-stuktur.

Både APIet[5] fra NVDB og APIet fra MapQuest tilbyr data både som JSON og XML. Vi har valgt å bruke JSON konsekvent, på grunn av fordelene knyttet til størrelse og hastighet. Det er i tillegg et krav at søkeobjekter[6] sendt til NVDB er på JSON-formatet, så det er derfor naturlig at det er samsvar mellom utgående og inngående objekter i applikasjonen.

1.2 Nasjonal vegdatabank

Nasjonal vegdatabank (NVDB) er en database med informasjon om riks- og fylkesveger, kommunale veger, private veger og skogsbilveger. Databasen brukes aktivt i Norges vegforvaltning. Den inneholder blant annet informasjon om vegnett med geometri og topologi som danner grunnlaget for kartløsninger og ruteberegnere på internett. I tillegg finnes en oversikt over utstyr og drenering langs vegen, ulykker og trafikkmengder og grunnlagsdata for bruk i støyberegning og trafikkmodellering.

Statens vegvesen

Figur 1: Dataene i NVDB eies og forvaltes av Statens vegvesen.

APIet er basert på REST[7], og kan brukes til å hente de fleste grunnlagsdata i NVDB. Dataene leveres på XML- eller JSON-format. I dag inneholder APIet mer enn 16 millioner objekter, fordelt på over 250 objekttyper.

1.2.1 Dataeier og lisens

Statens vegvesen eier og forvalter dataene i NVDB. Dataene er lisensiert under Norsk Lisens for Offentlige Data (NLOD)[8]. Denne lisensen er utarbeidet av Fornyings-, administrasjons- og kirkedepartementet og er ment brukt av offentlige virksomheter ved tilgjengeliggjøring av offentlige data. Lisensen gir kort oppsummert rett til fri bruk, også kommersielt, mot navngivelse av lisensgiver og at dataene ikke fremstilles villedende. I tilfellet med NVDB må applikasjonen inneholde teksten ”Inneholder data under norsk lisens for offentlige data (NLOD) tilgjengeliggjort av Statens vegvesen.”.

[5] Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.
[6] Les mer om dette under "1.2.2.2 Finne objektene på den vegen vi befinner oss".
[7] Representational State Transfer, REST, er en programvarearkitektur for distribuerte systemer som World Wide Web. REST har etter hvert blitt den dominerende designmodellen for web-APIer.
[8] http://data.norge.no/nlod/no/1.0

1.2.2 Bruk av Nasjonal vegdatabank API

Når vi spør mot NVDB APIet bruker vi GET som innebærer at vi sender med informasjon til serveren i URL'en[9]. Vi sender i tillegg med en header[10] som forteller at vi ønsker å motta JSON-data. APIet har en grunn-URL som er følgende:

https://www.vegvesen.no/nvdb/api/

Ved spesifikke søk mot APIet legger vi til data på slutten av denne URLen.

1.2.2.1 Finne ut hvor brukeren befinner seg

NVDB APIet har funksjonalitet for å finne ut på hvilken veg brukeren befinner seg. Ved å sende koordinater til APIet returnerer det en *vegreferanse*, så sant det eksisterer en veg i nærheten av dette punktet. En vegreferanse inneholder informasjon om en veg, som for eksempel hva slags veg det er, i hvilken kommune den ligger osv. Viktigst for oss så inneholder den også en *veglenke*.

Følgende URL spør etter nærmeste veg til punktet 10.198832, 59.625669[11]:

…/vegreferanse?srid=WGS84&x=10.198832&y=59.625669

"srid=WGS84" angir at koordinatsystemet som brukes er WGS84, som er det systemet som brukes i lokasjonstjenesten i iOS. Spørringen over gir følgende svar fra NVDB:

```
{
  "sokePunkt" : "POINT (10.198832 59.625669)",
  "sokePunktSrid" : "WGS84",
  "punktPaVegReferanseLinjeUTM33" : "POINT (229343.56174456686 6619511.951859929)",
  "punktPaVegReferanseLinjeWGS84" : "POINT (10.198944775244298 59.62563354138125)",
  "meterVerdi" : 3702,
  "visningsNavn" : "EV 18 HP2 m3702",
  "veglenkeId" : 2172077,
  "veglenkePosisjon" : 0.8449232402200674,

  ...
}
```

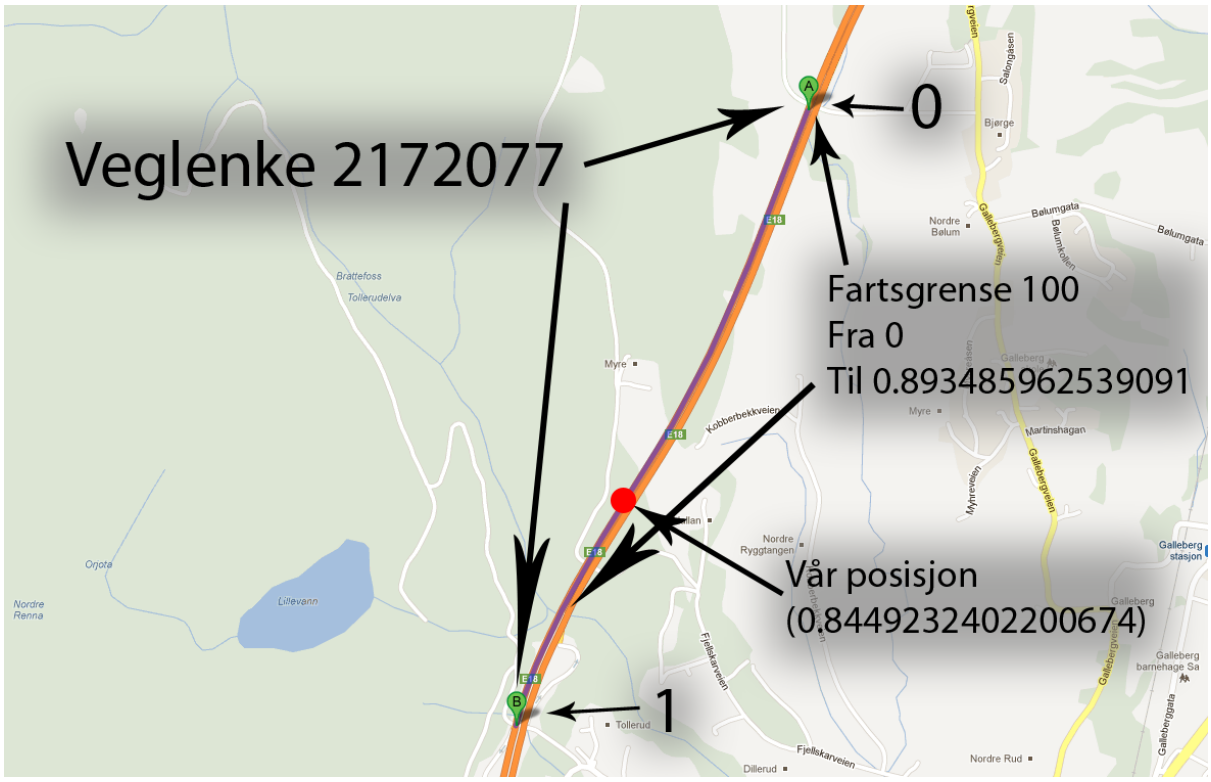
Kodesnutt 2: De første linjene av svaret fra NVDB når det spørres etter en vegreferanse på punkt 10.198832, 59.625669.

En veglenke er en sammenhengende vegstrekning med en unik ID. Posisjoner langs en veglenke defineres med et tall mellom 0 og 1. Vegreferansen som mottas når vi spør APIet om posisjonen vår forteller oss hvilken veglenke vi er på og hvilken posisjon vi befinner oss på på denne veglenken.

Hvis vi bruker eksempelet over ser vi at vi befinner oss på en veglenke med id 2172077 på posisjon 0.8449232402200674. Sett at vi senere finner en fartsgrense hvor det er oppgitt at den gjelder på

[9] Uniform Resource Locator.
[10] Supplerende data plassert først i en datablokk som blir lagret eller sendt.
[11] I Norge oppgis lengde- og breddegrader i motsatt rekkefølge av f.eks. USA. I Norge skriver vi "10.198832, 59.625669", mens i USA skriver de "59.625669, 10.198832".

veglenke 2172077 fra posisjon 0 til posisjon 0.893485962539091. Vi kan dermed vite at fartsgrensen gjelder der vi er, og trenger ikke forholde oss til avstander eller koordinater.



Figur 2: Bildet viser veglenken fra eksempelet. Vi ser at vår posisjon er mellom fra- og til-posisjonen til fartsgrensen med verdi 100, og kan dermed vite at den gjelder der vi er. (Kart: Google Maps)

1.2.2.2 Finne objektene på den vegen vi befinner oss

Når vi nå skal finne objekter på vegen vi er på, benytter vi veglenkeidentifikasjonen og –posisjonen vi mottok da vi spurte etter vegreferansen. Når vi søker etter objekter i NVDB må vi riktignok sende med et søkeobjekt på JSON-format.

```
{
  lokasjon:
  {
    veglenker:
    [
      {
        id: 2172077,
        fra: 0,
        til: 1
      }
    ]
  },
  objektTyper:
  [
    {
      id: 105,
      antall: 0,
      filter: []
    }
  ]
}
```

Kodesnutt 3: Et søkeobjekt på JSON-format, brukt til å søke etter objekter i NVDB.

Søkeobjektet i kodesnutt 3 søker etter fartsgrenser på vegen vi er på. Under lokasjon er det oppgitt en veglenke med id 2172077, med fra- og til-verdier på henholdsvis 0 og 1. Dette betyr at vi søker på hele veglenken med id 2172077. Under objekttyper søker vi etter et objekt med id 105. Dette er den numeriske id'en til fartsgrenser i NVDB. Når antall er satt til 0 betyr det at vi vil ha alle objektene den finner. Filteret står tomt her, men kan brukes til å filtrere hvilke objekter vi ønsker, eller ikke ønsker. I denne sammenhengen kunne vi for eksempel bedt om å kun motta fartsgrenser som har verdien 50.

Satt inn i URL'en ser spørringen mot NVDB slik ut:

…/sok?kriterie={lokasjon:{veglenker:[{id:2172077,fra:0,til:1}]},objektTyper:[{id:105,antall:0,filter:[]}]}

Med denne spørringen får vi følgende svar fra NVDB:

```
{
  "sokeObjekt": {...},
  "totaltAntallReturnert": 4,
  "resultater":
  [
    {
      "typeId": 105,
      "statistikk": {...},
      "vegObjekter":
      [
        {
          "objektId": 87557714,
          "objektTypeId": 105,
          "objektTypeNavn": "Fartsgrense",

          ...

          "egenskaper":
```

```
[
  {...},
  {
    "id": 2021,
    "navn": "Fartsgrense",
    "verdi": "100",
    "enumVerdi": {...},
    "definisjon": {...}
  },

  ...

],
strekningslengde: 8630,
lokasjon:
{

...

veglenker:
[
  {
    "id": 2172077,
    "fra": 0,
    "til": 0.893485962539091,
    "direction": "WITH"
  }
],

...
```

Kodesnutt 4: Svar fra NVDB på JSON-format. Her har vi spurt etter fartsgrenser på veglenke 2172077.

Svaret vi mottok fra NVDB over inneholder 4 fartsgrenser. Det som gjenstår for å finne ut hvilken fartsgrense som gjelder akkurat der vi er vil derfor være å gå gjennom fartsgrensene og se om en av dem dekker det punktet på veglenken vi befinner oss på.

Vi finner en fartsgrense som gjelder på veglenke 2172077 fra 0 til 0.893485962539091. Som vi vet fra eksempelet over befinner vi oss på den samme veglenken, og på posisjon 0.8449232402200674. Vi vet dermed at dette er gjeldende fartsgrense-objekt der vi er. Under egenskaper i svaret ser vi at fartsgrensen er 100.

1.2.2.3 Utvidelser og praktisk bruk

Måten vi søker etter en vegreferanse i eksempelet over er det samme som gjøres i Kjørehjelperen. Vi gjør denne spørringen hver gang vi mottar nye koordinater fra posisjonstjenesten i telefonen. Dette er nødvendig, siden vi trenger hjelp av NVDB til å avgjøre hvilken veg vi befinner oss på til enhver tid.

Når vi søker etter objekter langs en veg i eksempelet over, sender vi kun med én objekttype, altså fartsgrense. "Objekttyper" i søkeobjektet er et array[12], noe som betyr at vi kan sende med flere objekttyper. Det er dette vi gjør i applikasjonen. Vi søker etter alle 40 objekttypene samtidig, uavhengig av om brukeren ønsker å bli varslet om disse eller ikke. Alle objektene lagres deretter på

[12] Et array er en endimensjonal tabell.

telefonen, slik at neste gang brukeren befinner seg på vegen med denne id'en, lastes dataene fra telefonen i stedet.

1.3 MapQuest

MapQuest er en online karttjeneste som tilbyr blant annet vegkart og navigasjonshjelp. De leverer tjenester til bruk med blant annet JavaScript[13], Flash[14] og webtjenester, i tillegg til egne kartapplikasjoner til forskjellige smarttelefoner. MapQuest eies av amerikanske AOL[15].

Figur 3: Avstandene til objekter i applikasjonen benytter MapQuest Open Directions API Web Service.

I Kjørehjelperen benyttes MapQuest Open Directions API Web Service. Denne tjenesten benytter OpenStreetMap[16] til å levere navigasjonsdata direkte over HTTP[17].

1.3.1 Valg av MapQuest

NVDB inneholder ingen avstander eller lengder. Vi var derfor nødt til å benytte en annen tjeneste for å beregne avstanden frem til punkter av interesse. Google Maps ble også vurdert som mulig leverandør av navigasjonstjeneste, men ble forkastet grunnet lisensieringsproblemer[18]. Valget falt på MapQuest på grunn av den åpne lisensen.

1.3.2 Bruk av tjenesten

På samme måte som NVDB APIet sendes spørringen til MapQuest APIet som en URL med parametere. Vi bygger også her et søkeobjekt som JSON:

```
{
  locations:
  [
    {
      latLng:
      {
        lat:59.652683,
        lng:10.226554
      }
    },
    {
```

[13] JavaScript er et skriptspråk som er best kjent for å tilføre dynamiske elementer til nettsider.
[14] Adobe Flash er programvare som utvikles og distribueres av Adobe Systems. Flash brukes mest til å vise dynamisk innhold på nettsider og støtter vektor- og pikselgrafikk.
[15] AOL Inc. (Tidligere kjent som America Online).
[16] OpenStreetMap (OSM) er et fritt redigerbart kart over hele jorden, laget og redigert av brukerne. OpenStreetMap er lisensiert under Open Data Commons Open Database License (ODbL), se http://opendatacommons.org/licenses/odbl/1.0/ for mer info.
[17] Hypertext Transfer Protocol.
[18] Google krever at deres navigasjonstjeneste benyttes i deres eget kart.

```
      latLng:
      {
        lat:59.633401,
        lng:10.205687
      }
    }
  ],
  options:
  {
    unit:k
  }
}
```

Kodesnutt 5: Et søkeobjekt på JSON-format som sendes til MapQuest APIet.

Som man ser av eksempelet over inneholder søkeobjektet to koordinatpunkter, det første er vår posisjon, mens det andre er posisjonen til objektet vi ønsker å finne avstanden til. Under "options" settes "unit" til "k". Dette sørger for at vi mottar et svar i kilometer. Vi legger søkeobjektet inn i parameteren "json" og får følgende URL som sendes til MapQuest:

http://open.mapquestapi.com/directions/v1/routematrix?key=X&json={locations:[{latLng:{lat:59.652683, lng:10.226554}},{latLng:{lat:59.633401,lng:10.205687}}],options:{unit:k}}

Legg merke til den ekstra parameteren "key=X" i URLen over. For å bruke APIet kreves det at man registrerer applikasjonen hos MapQuest. Kjørehjelperen har en lang og unik nøkkel som sendes med i spørringen i stedet for X.

Spørringen over gir følgende svar fra MapQuest:

```
{
  "allToAll" : false,
  "distance" :
  [
    0,
    2.465
  ],

  ...
```

Kodesnutt 6: Et eksempel på svar fra MapQuest.

Svaret inneholder en del mer informasjon enn det som kommer frem av eksempelet over. Vi er allikevel kun interesserte i én ting: Det andre tallet under "distance". Dette tallet representerer avstanden mellom de to punktene vi oppga. Det er altså 2,465 km fra vår posisjon til objektet foran oss.

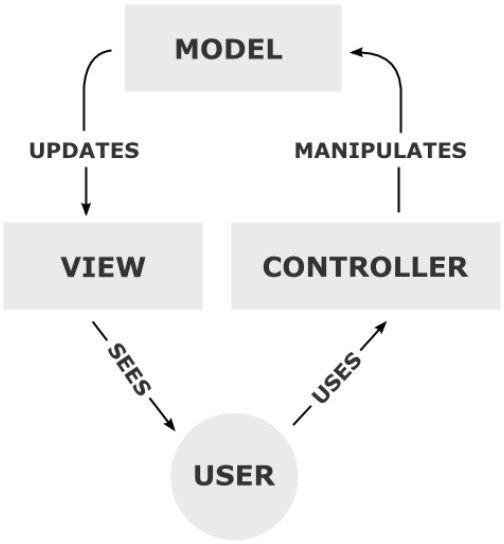
2 Design pattern

I dette kapittelet tar vi for oss MVC, hvordan det er implementert i iOS[19] og hvordan vi bruker det i Kjørehjelperen.

2.1 Model View Controller

Model View Controller, eller MVC, er et programvarearkitektur-pattern[20] som separerer representasjonen av data fra hvordan det fremstilles for brukeren.

Modellen består av selve datastrukturen, i tillegg til regler, logikk og funksjoner. I objektorientert programmering faller selve objektene inn under modellen. I tillegg får funksjoner som interagerer med databaser eller andre eksterne tjenester plass her.



Figur 4: Modellen, viewet og kontrolleren i forhold til brukeren (Foto: Wikimedia Commons).

Viewet er det brukeren ser, altså brukergrensesnittet. Frontend-kode, som HTML[21], Javas[22] Swing-bibliotek eller iOS's UIView-klasse er eksempler på kode som kun har noe med hvordan brukeren ser informasjon.

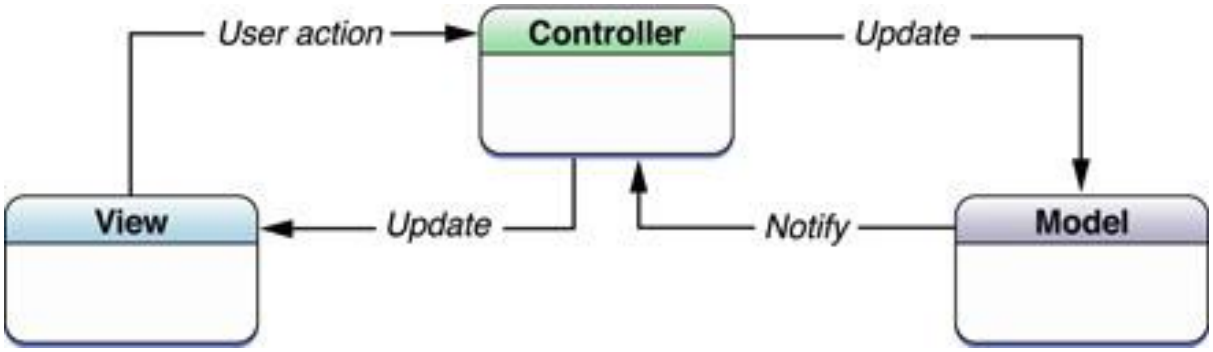
Kontrolleren (Controller) er bindeleddet mellom modellen og viewet. Kontrollerens oppgave er å formidle informasjonen fra modellen til viewet, og å sende eventuelle endringer fra brukeren tilbake til modellen.

De sentrale ideene bak MVC er gjenbruk av kode og separering av ansvar. Modellen trenger ikke tenke på hvordan dataene fremstilles for brukeren, og viewet trenger ikke bry seg om hvor dataene det viser kommer fra. Det er kontrollerens oppgave å binde de to andre sammen. Tanken er dermed at man kan bytte ut enkeltdeler av koden ved bytte av f.eks. datagrunnlag eller plattform, uten å måtte skrive all kode på nytt.

[19] iOS er Apples operativsystem for iPhone, iPad og iPod.
[20] Et designpattern er en kjent gjenbrukbar løsning på et vanlig problem i systemutvikling.
[21] HyperText Markup Language (HTML) er et markeringsspråk for formatering av nettsider.
[22] Java er et objektorientert programmeringsspråk utviklet av Sun Microsystems (nå kjøpt av Oracle Corporation).

2.2 MVC i iOS

iOS følger også MVC-patternet. Det har modeller, views og kontrollere. Det spesielle er at i en iOS-applikasjon er gjerne viewet slått sammen med kontrolleren i en såkalt ViewController. En ViewController er en kontroller som fokuserer mer på viewet enn modellen. Den "eier" et eller flere views, og hovedoppgaven dens er å administrere brukergrensesnittet. Den er riktignok fremdeles to separate elementer, en kontroller med et tilhørende view. Grunnen til denne fremgangsmåten er at i en iOS-applikasjon oppdateres ofte viewet med en gang det skjer en endring i modellen. Slettes det f.eks. en innføring i en tabell må modellen oppdateres, og i tillegg forventes det at raden forsvinner fra viewet med en gang. Det er i tillegg ønskelig at raden forsvinner uten at hele viewet må tegnes på nytt.



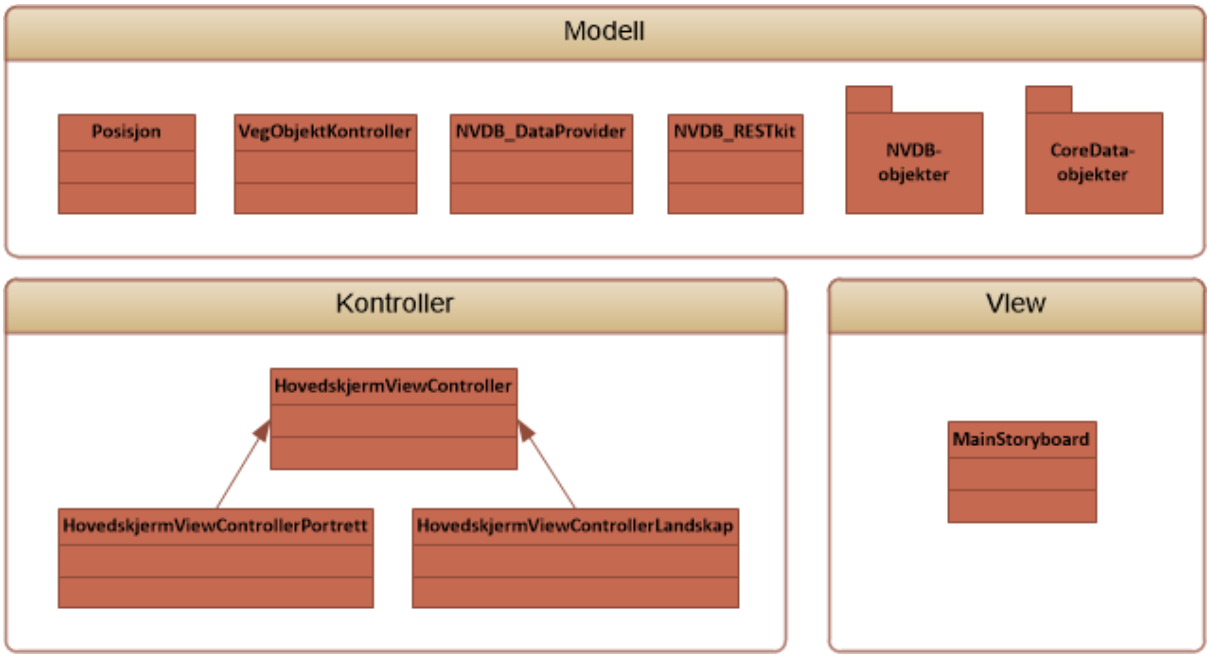
Figur 5: Denne figuren viser hvordan Apple beskriver forholdet mellom modellen, viewet og kontrolleren (Foto: Apple).

2.3 MVC i Kjørehjelperen

Kjørehjelperen bruker naturlig nok også MVC-patternet. Figur 22 gir en forenklet oversikt over hvor de ulike klassene i applikasjonen passer inn i de tre MVC-lagene. For å gi en forståelse av hvorfor de forskjellige klassene er plassert der de er, gis det også en kort beskrivelse av de ulike klassenes funksjon og oppgave.

Modellaget er det absolutt største laget. Her har vi Posisjon, en klasse som finner GPS-posisjonen til telefonen. VegObjektKontroller inneholder logikken som bestemmer hva som skal vises på skjermen til brukeren. NVDB_DataProvider sørger for å levere data til VegObjektKontroller, enten fra NVDB eller fra Core Data[23]. NVDB_RESTkit sørger for kommunikasjon med NVDB APIet og MapQuest APIet. NVDB-objekter og CoreData-objekter er objekter som representerer konkrete objekter, f.eks. en fartsgrense eller en jernbaneovergang.

[23] Core Data er et iOS-rammeverk som lagrer data i en database lokalt på enheten. Les mer under "3.1 Core Data".



Figur 6: De ulike klassene i Kjørehjelperen tilhører enten modell-, kontroll- eller view-laget i MVC.

Kontrollaget inneholder tre klasser, hvorav HovedskjermViewControllerPortrett og HovedskjermViewControllerLandskap er subklasser[24] av HovedskjermViewController. Subklassene gjør bare ting som er spesifikt for orienteringen av telefonen, det meste av logikken ligger i HovedskjermViewController. Denne klassen er ansvarlig for å vise informasjonen den mottar fra VegObjektKontroller på skjermen. I tillegg utfører den oppgaver knyttet til oppstart og avslutning av applikasjonen, endring av innstillinger og endring av orientering.

Viewet inneholder kun en klasse, MainStoryboard. Denne klassen, eller storyboardet[25], beskriver hvordan skjermbildet er bygget opp ved hjelp av XML. Storyboardet lages med et grafisk verktøy i Xcode[26], som så genererer XML-koden automatisk.

En mer detaljert beskrivelse av de forskjellige klassene og interaksjonen mellom dem er forklart under kapittel 4.2, Klassene i Kjørehjelperen.

[24] En subklasse er en klasse som arver alle egenskapene til en superklasse, eller en foreldreklasse.
[25] Et storyboard er et dokument som beskriver layoutet til et skjermbilde og overgangene mellom dem i iOS.
[26] Xcode er et program til Mac for utvikling, testing og publisering av programmer til iOS og OS X.

3 Sentrale rammeverk

Når man utvikler applikasjoner til iOS eller andre mobile plattformer er det mye som skjer i bakgrunnen som man slipper å tenke på som utvikler. Takket være solide SDKer[27] får man tilgang til posisjon, kontaktlister og kamera for å nevne noe, med enkle funksjonskall. Skulle vi gått i detalj på alle rammeverkene vi har benyttet oss av i iOS ville dette dokumentet blitt alt for omfattende. Vi har derfor valgt å gå inn på to sentrale rammeverk som har spart oss for mye arbeid.Det ene er Core Data, et rammeverk i iOS som lagrer data lokalt på enheten. Det andre er RestKit, et åpent rammeverk som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen[28] av data til objekter.

3.1 Core Data

Core Data is an object graph and persistence framework provided by Apple.[29] Med dette menes det at Core Data lagrer instanser av klasser på et gitt tidspunkt, og forholdet mellom dem. Det forteller også at Core Data fungerer som et abstrakt lag mellom en applikasjon og de lagrede dataene, for eksempel en database.

Figur 7: Ikonet til Core Data (Foto: Apple).

For å si det på en enklere måte: Core Data lar deg opprette objekter i applikasjonen på vanlig måte. Det er så Core Datas oppgave å lagre disse objektene, oppdatere eventuelle endringer som gjøres, og å hente dem inn igjen neste gang applikasjonen startes. Som utvikler trenger man ikke tenke på hvor eller hvordan disse lagres.

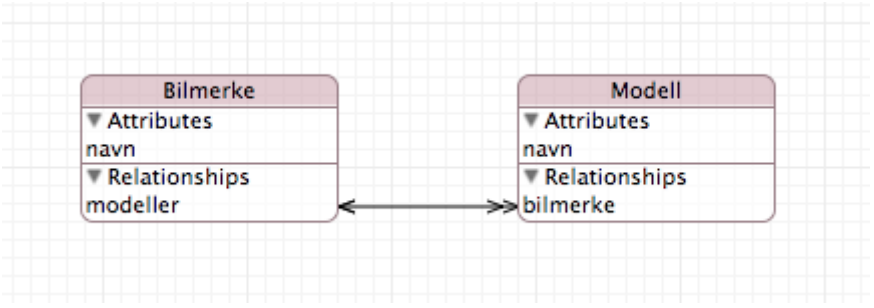
Core Data bruker gjerne en SQLite-database[30] til å lagre data, men man trenger altså ikke skrive SQL[31] for å benytte seg av dette. Det er allikevel noe arbeid som må gjøres for å komme i gang med Core Data.

Figur 8: Core Data bruker gjerne en SQLite-database (Foto: Wikimedia Commons).

For å benytte Core Data i et iOS-prosjekt må man tegne opp alle entitetene[32] som ønskes og forholdet

[27] Software development kit (SDK) er verktøy som lar deg utvikle programmer til en spesifikk programvarepakke, et rammeverk, en hardware-plattform, et operativsystem e.l.
[28] Mapping er prosessen med å overføre data fra en datastruktur til en annen, f.eks. fra JSON til et NSObject (objekt i Objective-C).
[29] Hentet fra artikkelen om Core Data på Wikipedia.org (http://en.wikipedia.org/wiki/Core_Data).
[30] SQLite er et relasjonsdatabasesystem i et lite C-bibliotek. I motsetning til andre databasesystemer er SQLite integrert i klientapplikasjonen og ikke en separat prosess.
[31] Structured Query Language (SQL) er et programmeringsspråk brukt til å kommunisere med relasjonsdatabaser.
[32] En entitet kan defineres som noe som er i stand til selvstendig eksistens og som kan identifiseres.

mellom dem. Dette gjøres med et grafisk verktøy i Xcode. Man må gi entitetene navn, attributter og relasjoner. For å ta et klassisk eksempel: Du oppretter en entitet som heter "Bilmerke" med et attributt kalt "navn". Deretter oppretter du en ny entitet kalt "Modell" med et annet attributt også kalt "navn". "Bilmerke" får så en en-til-mange-relasjon til "Modell" kalt "modeller" siden hvert bilmerke som oftest har mange modeller, mens hver modell kun har ett "Bilmerke".



Figur 9: Entitetene "Bilmerke" og "Modell" i Xcode.

Xcode genererer deretter klasser for deg basert på disse entitetene.

```
// Bilmerke.h

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Modell;

@interface Bilmerke : NSManagedObject

@property (nonatomic, retain) NSString * navn;
@property (nonatomic, retain) NSSet * modeller;
@end

@interface Bilmerke (CoreDataGeneratedAccessors)

- (void)addModellerObject:(Modell *)value;
- (void)removeModellerObject:(Modell *)value;
- (void)addModeller:(NSSet *)values;
- (void)removeModeller:(NSSet *)values;

@end


// Bilmerke.m

#import "Bilmerke.h"
#import "Modell.h"

@implementation Bilmerke

@dynamic navn;
@dynamic modeller;

@end
```

Kodesnutt 7: Autogenerert klasse i Xcode basert på entiteten "Bilmerke". En klasse i Objective-C består av to filer, en deklarasjonsfil (*.h) og en implementasjonsfil (*.m). Les mer om Objective-C under 4.1.1 Objective-C.

I kodesnutt 7 ser vi den autogenererte klassen Bilmerke. Klassen har fått en variabel "navn" og et sett[33] av modeller. I tillegg har Core Data generert fire metoder for å legge til eller fjerne en eller flere modeller.

Man kan nå opprette objekter i applikasjonen på samme måte som man ellers ville gjort. Man må riktignok kalle en spesiell metode ved opprettelse av et nytt objekt, og man må huske å fortelle Core Data at man vil lagre endringene i det nye objektet.

```
Bilmerke * nyttMerke = [NSEntityDescription
                                    insertNewObjectForEntityForName:@"Bilmerke"
                                    inManagedObjectContext:managedObjectContext];
[nyttMerke setNavn:@"Toyota"];

Modell * nyModell = [NSEntityDescription
                                    insertNewObjectForEntityForName:@"Modell"
                                    inManagedObjectContext:managedObjectContext];
[nyModell setNavn:@"Avensis"];

[nyttMerke addModellerObject:nyModell];

NSError * error;
[managedObjectContext save:&error];
```

Kodesnutt 8: Eksempel på opprettelse av et bilmerke "Toyota" med en modell "Avensis".

I den første linjen i kodesnutt 8 oppretter vi et Bilmerke-objekt. "insertNewObjectForEntityName:@"Bilmerke"" betyr at vi vil knytte Bilmerke-objektet til entiteten "Bilmerke" i Core Data. "inManagedObjectContext:managedObjectContext" betyr at Core Data-laget vi vil bruke er definert i variabelen managedObjectContext. Denne variabelen ble satt da applikasjonen startet. Deretter setter vi navnet på merket, og oppretter et Modell-objekt på samme måte. Modell-objektet legger vi deretter til merket med "addModellerObject:nyModell". Til slutt må vi fortelle Core Data at vi ønsker å lagre de nye objektene. Dette gjøres gjennom managedObjectContext. Vi må sende med en referanse til NSError når vi lagrer for å plukke opp eventuelle feil som skulle oppstå.

Når vi ønsker å hente ut objekter fra Core Data må vi opprette en NSEntityDescription. Denne legges inn i et NSFetchRequest som kjøres av managedObjectContext. Dette returnerer et array bestående av alle entitetene av denne typen i Core Data.

```
NSEntityDescription * bilmerkeEntity = [NSEntityDescription
                                    entityForName:@"Bilmerke"
                                    inManagedObjectContext:managedObjectContext];
NSFetchRequest * request = [[NSFetchRequest alloc] init];
[request setEntity:bilmerkeEntity];
```

[33] Et sett (NSSet i Objective-C) er en datastruktur som holder på mange elementer. Elementene er ikke knyttet til en fast plass slik som i et array.

```
NSError * error;
NSArray * resultat = [managedObjectContext executeFetchRequest:request
                                                          error:&error];
```

Kodesnutt 9: Eksempel på uthenting av bilmerker fra Core Data.

Variabelen "resultat" peker nå på et array som inneholder alle bilmerkene i Core Data. Innholdet i "Bilmerke"-entiteten er gjort om til Bilmerke-objekter. På grunn av relasjonene vi oppga da vi designet entitetene inneholder også Bilmerke-objektene sine tilhørende Modell-objekter.

3.2 RestKit

RestKit er et Objective-C[34]-rammeverk for iOS som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen av objekter. Rammeverket fungerer slik at man oppgir URLen til APIet, forteller hvilke objekter man forventer å motta, og beskriver hvordan de mottatte dataene skal mappes til disse objektene. Resten ordner RestKit. Man kjører et asynkront metodekall til RestKit, og deretter leverer RestKit de ferdig mappede objektene når ting er ferdig.

RestKit er veldig kraftig, og har mange funksjoner, blant annet støtte for Core Data. Eksemplene i de påfølgende kodesnuttene er gjort så enkle som mulig for å forstå den grunnleggende funksjonaliteten i RestKit.

```
{
  personer :
  [
    {
      fornavn : "Lars"
      etternavn : "Smeby"
    },
    {
      fornavn : "Henrik"
      etternavn : "Hermansen"
    }
  ]
}
```

Kodesnutt 10: Et JSON-objekt bestående av to personer.

I kodesnutt 10 ser vi et eksempel på hva vi forventer å motta fra APIet. Her har vi to person-objekter, hver med et fornavn og et etternavn.

```
// Person.h

#import <Foundation/Foundation.h>

@interface Person : NSObject

@property (nonatomic, strong) NSString * fornavn;
@property (nonatomic, strong) NSString * etternavn;
```

[34] Objective-C er programmeringsspråket som brukes av Apple. Det er basert på C.

```
@end
```

Kodesnutt 11: Deklarasjonen av Person-objektet vårt i Xcode.

I Xcode deklarerer vi så et Person-objekt med fornavn og etternavn. Kodesnutt 11 viser deklarasjonsfilen (headerfilen[35]) til Person-objektet.

```
NSURL * grunnURL = [NSURL URLWithString:@"http://eksempel.no/api"];
AFHTTPClient * klient = [AFHTTPClient clientWithBaseURL:grunnURL];
[klient setDefaultHeader:@"Accept" value:RKMIMETypeJSON];

RKObjectMapping * personMapping = [RKObjectMapping
                                              mappingForClass:[Person class]];
[personMapping addAttributeMappingsFromDictionary:@{@"fornavn" : @"fornavn",
                                                    @"etternavn" : @"etternavn"}];

RKResponseDescriptor * responseDescriptor = [RKResponseDescriptor
                                    responseDescriptorWithMapping:personMapping
                                                         pathPattern:nil
                                                             keyPath:@"personer"
          statusCodes:RKStatusCodeIndexSetForClass(RKStatusCodeClassSuccessful)];

RKObjectManager * objectManager = [[RKObjectManager alloc]
                                                    initWithHTTPClient:klient];
[objectManager addResponseDescriptor:responseDescriptor];

[objectManager getObjectsAtPath:@"http://eksempel.no/api/personer" parameters:nil
   success:^(RKObjectRequestOperation *operation, RKMappingResult *mappingResult)
   {
      // Hvis alt går bra kommer vi hit. Resultatet ligger i parameteren
      // mappingResult.
   }
   failure:^(RKObjectRequestOperation *operation, NSError *error)
   {
      // Hvis det har skjedd en feil kommer vi hit. Beskrivelse av feilen ligger i
      // parameteren error.
   }];
```

Kodesnutt 12: Eksempel på bruk av RestKit for å hente ned JSON-data og mappe det til Person-objekter.

Kodesnutt 12 viser bruken av RestKit for å hente ned JSON-data og mappe det til Person-objekter. Eksempelet inneholder mange statiske[36] metodekall til RestKit-klasser som vi ikke skal gå inn på. Vi tar kun for oss den overordnede gangen i eksempelet.

Først oppretter vi en klient med URLen til APIet og sier at vi ønsker å motta JSON-data. Deretter oppretter vi en mappingklasse. Her forteller vi at mottatte data skal mappes til Person-objekter og at "fornavn" i JSON-dataen skal mappes til "fornavn" i Person-objektet, og tilsvarende med "etternavn". Vi oppretter så en ResponseDescriptor som bestemmer hva som skal gjøres med resultatet av spørringen mot APIet. Med parameterne sier vi at vi skal bruke det nye mappingobjektet vi opprettet til å mappe det vi finner under "personer" i JSON-svaret. Til slutt oppretter vi en ObjectManager. Vi legger ResponseDescriptoren til ObjectManageren og sender

[35] En headerfil er en fil som forteller hvilke egenskaper og metoder en klasse har, men ikke hvordan de er implementert. Headerfiler benyttes bl.a. i C, C++ og Objective-C.
[36] Statiske metoder er metoder som ikke er knyttet til en spesifikk instans av en klasse.

avgårde en forespørsel. Hvis alt går bra havner vi i "success"-delen etter kallet med alle Person-objektene liggende i parameteren "mappingResult".

4 Programmets oppbygging og virkemåte

I dette kapittelet tar vi for oss hvordan Kjørehjelperen er bygget opp. Først tar vi for oss noen sentrale elementer i iOS og Objective-C som man må kjenne til. Deretter tar vi for oss hvilke klasser applikasjonen består av og deres funksjoner. Vi går ikke inn på hver enkelt metode, dette hadde blitt for omfattende. Applikasjonen inneholder nesten 13.000 kodelinjer, selv om mye av dette er autogenerert av Xcode. Til slutt følger vi gangen i applikasjonen fra den spør etter posisjonen til den viser vegskilt på skjermen.

4.1 iOS og Objective-C

Når man utvikler til iOS eller OS X[37] benytter man Objective-C. Vi skal se nærmere på syntaksen i dette språket og sammenlikne det med Java. Vi skal også se på iOS-spesifikke elementer som protokoller, delegater og storyboards.

4.1.1 Objective-C

Objective-C er et objektorientert høynivåspråk[38]. Det er basert på Smalltalk[39] og C. Objective-C er hovedprogrammeringsspråket brukt av Apple i OS X og iOS. Språket ligner på C++, C# og Java, men har en litt annerledes syntaks på metodekall.

```
- (int)adderTallA:(int)a MedB:(int)b
{
   return a + b;
}

...

int c = [adderTallA:4 MedB:5];
```

```
public int adder(int a, int b)
{
   return a + b;
}

...

int c = adder(4, 5);
```

Kodesnutt 13: En metodedeklarasjon og et kall på metoden i Objective-C (til venstre) og Java (til høyre).

På samme måte som i C og C++ deler Objective-C klasser opp i to filer, en deklarasjonsfil (headerfil) og en implementasjonsfil. Grunnen til dette er at når man inkluderer klassen i andre filer trenger man ikke vite noe om logikken i klassen, kun navnet på egenskaper og metoder. Det holder derfor å inkludere klassens headerfil (som gjerne er mye mindre enn implementasjonsfilen).

```
// Person.h

#import <Foundation/Foundation.h>

@interface Person : NSObject
```

[37] OS X er det gjeldende operativsystemet til Mac.
[38] Et høynivåspråk har stor abstraksjon fra måten datamaskinen fungerer.
[39] Smalltalk er et objektorientert programmeringsspråk. Det spesielle med Smalltalk er at sjekking av typer og objekter skjer under kjøring i motsetning til under kompilering.

```
@property (nonatomic, strong) NSString * fornavn;
@property (nonatomic, strong) NSString * etternavn;
- (NSString *)hentFulltNavn;

@end

// Person.m

#import "Person.h"

@implementation Person

@syntesize fornavn, etternavn;

- (NSString *)hentFulltNavn
{
   return [@{fornavn, etternavn} componentsJoinedByString:@" "];
}

@end
```

Kodesnutt 14: Header- og implementasjonsfilen til objektet Person i Objective-C.

I Objective-C deklarerer man objektet som et interface (som ikke må forveksles med et interface i Java) i headerfilen. Man angir deretter egenskapene med "@property". Nøkkelordet "nonatomic" betyr litt forenklet at egenskapen kan aksesseres av flere tråder[40]. "strong" har erstattet "retain" i ARC[41], og kreves når egenskapen er en peker[42] til et objekt. Nøkkelordet betyr at programmet skal holde rede på hvor mange objekter som er opprettet og slette dem fra minnet når de ikke brukes lenger. I interfacet deklarerer man også eventuelle metoder som skal være synlige utenfor klassen.

I implementasjonsfilen benyttes "@implementation" foran navnet på klassen. "@syntesize" gjør så variablene kan aksesseres direkte, uten å måtte bruke klassenavnet foran variabelnavnet.[43] Metodene deklarert i headerfilen må implementeres.

```
public class Person
{
   private String fornavn, etternavn;

   public String getFornavn {return fornavn;}
   public String getEtternavn {return etternavn;}
   public void setFornavn(String fornavn){this.fornavn = fornavn;}
   public void setEtternavn(String etternavn){this.etternavn = etternavn;}

   public String hentFulltNavn
   {
      return fornavn + " " + etternavn;
   }
}
```

Kodesnutt 15: Den samme klassen, Person, her skrevet i Java.

[40] En tråd er en sekvens av programinstrukser som kan kjøres uavhengig av andre instrukser i et program (en lettvekts-prosess).
[41] Automatic Reference Counting (ARC) er at ansvaret for minnehåndtering flyttes fra programmereren til kompilatoren.
[42] En peker er en referanse til et sted i minnet hvor objektet ligger.
[43] I tidligere versjoner av iOS måtte man dessuten bruke "@syntesize" for at variablene skulle være tilgjengelige utenfor klassen, tilsvarende å skrive get- og set-metoder i Java.

4.1.2 Cocoa Touch

Når man utvikler applikasjoner til iPhone benyttes Cocoa Touch, et brukergrensesnitt-rammeverk for iOS. Det bygger på Cocoa-rammeverket brukt i utvikling til OS X, og er et abstrakt lag som gir tilgang til hardware-egenskaper på enheten.

4.1.3 Protokoller

En protokoll i Objective-C kan minne om et interface i Java. Det er en kontrakt som spesifiserer hvilke metoder en klasse må implementere. Med protokoller kan du dermed "standardisere" klasser og vite at en klasse som implementerer en protokoll også har implementert protokollens metoder.

Det er riktignok mulig å ha valgfrie metoder i en protokoll med nøkkelordet "@optional". Hvis man velger å benytte dette bør man teste om en klasse har implementert denne metoden før man eventuelt kaller den.

Selv om det er mulig å bruke en protokoll på samme måte som et interface i Java, brukes det som oftest til å deklarere delegater.

```
@protocol EksempelProtokoll
@required
- (void) eksempelMetodeA:(NSString *)enStreng;
- (void) eksempelMetodeB:(NSNumber *)etTall;
@end

...

@interface EnKlasse : NSObject <EksempelProtokoll>

@end
```

Kodesnutt 16: Deklarasjonen av en protokoll og deklarasjonen av en klasse som implementerer denne protokollen. Denne klassen må implementere protokollens metoder i implementasjonsfilen for at programmet skal kompilere.

4.1.4 Delegater

Delegater er en viktig del av iOS. Kommunikasjonen mellom biblioteker og klasser er basert på delegater. Det muliggjør asynkrone kall til klasser uten å låse hele systemet mens man venter på et resultat.

I C og C++ har man funksjonspekere[44] som kan minne noe om delegater. Det finnes ikke noe direkte tilsvarende i f.eks. Java. Man kan riktignok få til noe liknende i Java ved å lage sin egen event[45] og lyttere. Når eventen kjøres vil alle klasser som lytter etter denne eventen kjøre en navngitt metode. Til sammenlikning vil en delegat kun kjøre en navngitt metode i en spesifikk instans av en klasse.

[44] En funksjonspeker er en peker som peker til kjørbar kode i minnet.
[45] En hendelse i et program. Klasser og metoder kan "lytte" etter hendelser.

La oss klargjøre med et eksempel fra Kjørehjelperen: Vi har deklarert en protokoll i Posisjon-klassen som heter PosisjonDelegate. Denne protokollen krever at klasser som implementerer den også implementerer en metode "- (void) posisjonOppdatering(Posisjon *)posisjon". I Posisjon-klassen har vi også en peker "delegate" som peker til en klasse som implementerer denne protokollen. Posisjon-klassen vet ingenting om klassen denne peker til.

Når systemet kommer frem til en ny posisjon kalles denne metoden i Posisjon-klassen. Klassen som implementerer denne protokollen kjører dermed denne metoden.

```
// Posisjon.h

@protocol PosisjonDelegate
@required
- (void) posisjonOppdatering:(Posisjon *)posisjon;
@end

...
```

Kodesnutt 17: PosisjonDelegate deklareres i Posisjon.h.

```
// Posisjon.m

...

if([self.delegate conformsToProtocol:@protocol(PosisjonDelegate)])
{
   Posisjon * posisjon = [Posisjon alloc];

   ...

   [self.delegate posisjonOppdatering:posisjon];
}

...
```

Kodesnutt 18: I Posisjon.m sjekkes det at "delegat"-variabelen implementerer PosisjonDelegate-protokollen før "posisjonOppdatering"-metoden kalles med en ny posisjon.

```
// EksempelKlasse.h

@interface EksempelKlasse : NSObject <PosisjonDelegate>

...

@end
```

Kodesnutt 19: EksempelKlasse.h implementerer PosisjonDelegate-protokollen.

```
// EksempelKlasse.m

@implementation EksempelKlasse

...

   Posisjon * pos = [[Posisjon alloc] init];
   pos.delegate = self;

...
```

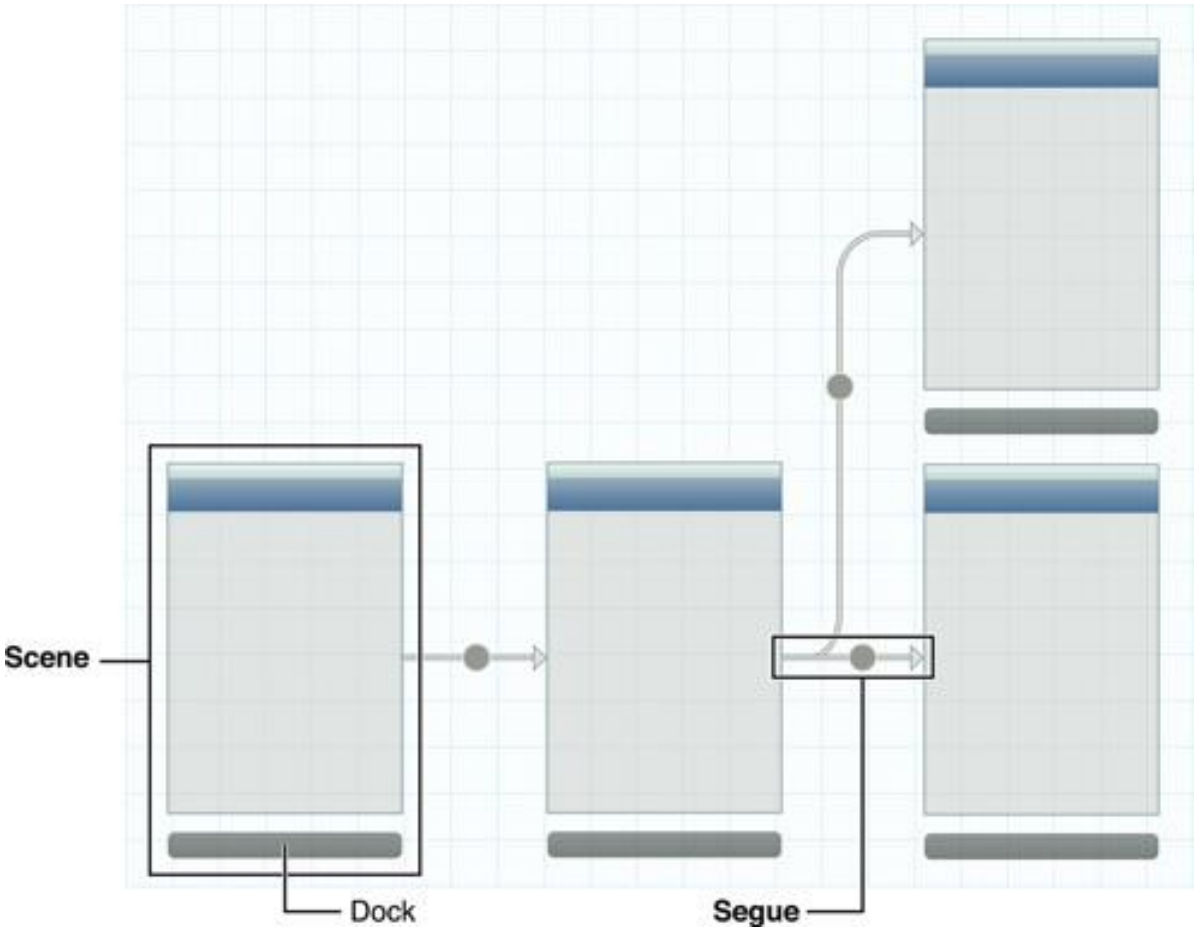
```
- (void) posisjonOppdatering(Posisjon *)posisjon
{
   // Gjør noe med posisjonen.
}

@end
```

Kodesnutt 20: I EksempelKLasse-klassen opprettes en instans av Posisjon-klassen. Her settes Posisjon-klassens "delegate"-variabel til instansen av EksempelKlasse-klassen. "posisjonOppdatering"-metoden er også implementert. Denne kjøres dermed når den kalles fra Posisjon-klassen.

Den største fordelen med delegater er at man kan separere ansvarsområder i applikasjonen. En applikasjon kan kjøre et metodekall og sende med en peker til seg selv. Denne klassen trenger dermed ikke tenke mer på hva som skjer før svaret kommer i form av et metodekall til den selv. Oppgavene kan dessuten utføres i forskjellige tråder. Dette er spesielt viktig i en klasse som styrer brukergrensesnittet. Hadde tråden måttet vente på svar ville man oppleve brukergrensesnittet som "fryst" inntil svaret kom. Dette løses ofte med å manuelt opprette flere tråder i andre programmeringsspråk.

4.1.5 Storyboards

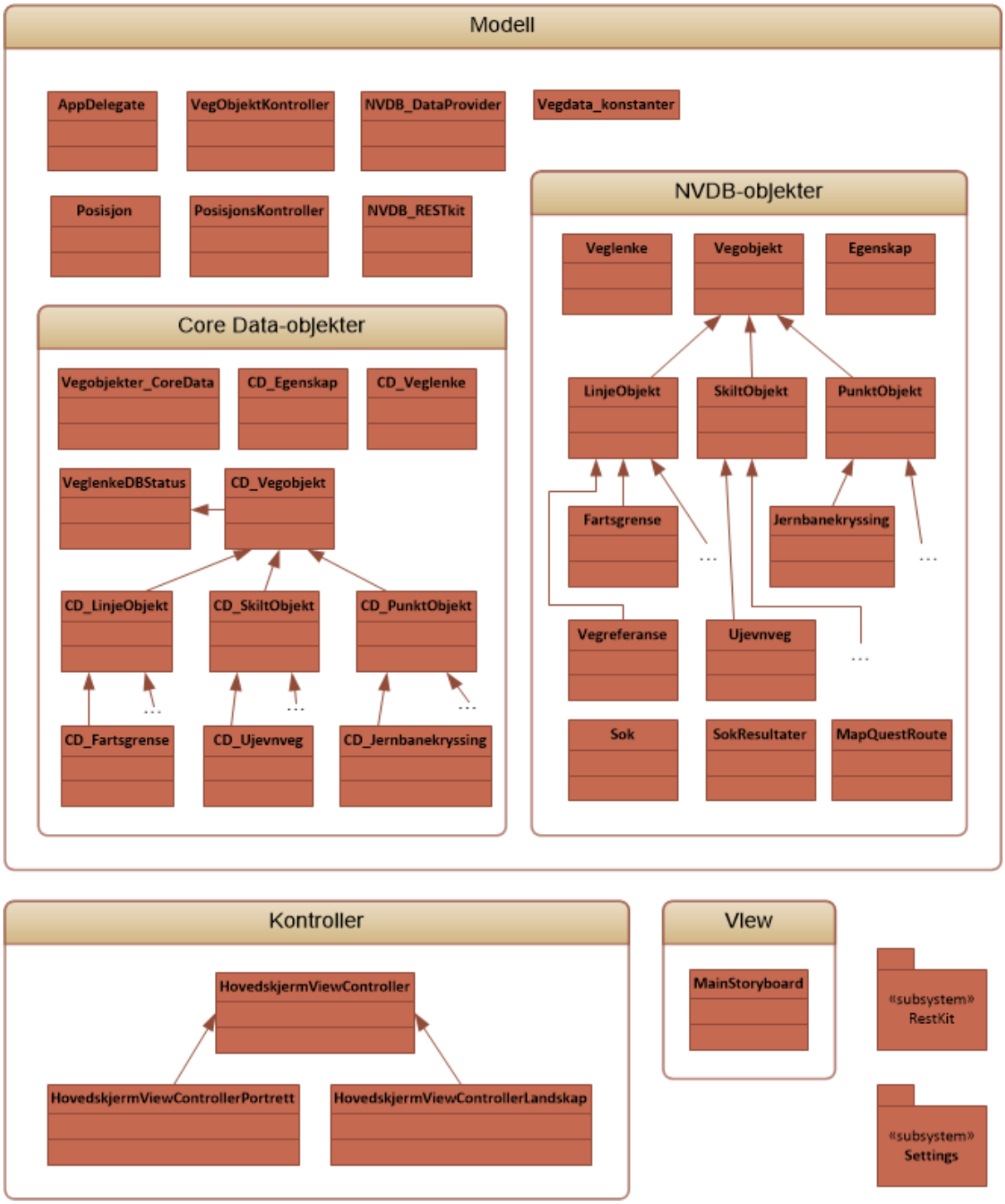
Et storyboard er en grafisk fremstilling av brukergrensesnittet i en iOS-applikasjon. Det viser skjermbildene i applikasjonen og koblingene mellom dem. I Xcode designes brukergrensesnittet i et eget grafisk storyboard-redigeringsprogram. Alt som gjøres i det grafiske redigeringsprogrammet konverteres automatisk til XML i en egen *.storyboard-fil.



Figur 10: Et eksempel på et storyboard i Xcode (Foto: Apple).

4.2 Klassene i Kjørehjelperen

Kjørehjelperen består av veldig mange klasser. Dette er først og fremst fordi alle objektene som vises på skjermen har en egen klasse, en egen Core Data-klasse og en egen container-klasse for å holde på et sett av objekter. I dette delkapittelet ser vi på klassenes viktigste funksjoner. Det forklares også kort hva hver enkelt metode gjør. Kapittelet bør derfor brukes som et oppslagsverk.



Figur 11: Figuren viser klassene i Kjørehjelperen.

4.2.1 AppDelegate

AppDelegate-klassen genereres automatisk av Xcode når man starter et nytt iOS-prosjekt. Klassen er som navnet tilsier en delegat, og mottar viktige hendelser i livssyklusen til applikasjonen. Den brukes gjerne til å utføre handlinger når applikasjonen starter, avsluttes eller minimeres.

Vi benytter AppDelegate til å lese inn innstillinger og klargjøre Core Data. Vi oppretter et NSManagedObjectContext-objekt[46] og sjekker størrelsen på SQLite-databasen. Hvis denne er større enn hva brukeren har satt som grense slettes de eldste innføringene inntil databasens størrelse er 90 % av denne. Når applikasjonen avsluttes kjøres det en ekstra lagring mot Core Data.

Første gang applikasjonen starter settes innstillingene i applikasjonen til standard-verdier. I tillegg vises det en dialogboks til brukeren med pålagte erklæringer fra NVDB og MapQuest.

Metodene i AppDelegate er:

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

Denne metoden setter standardinstillinger og viser pålagte erklæringer hvis applikasjonen kjøres for første gang.

- (BOOL)isForstegangsOppstart

Metoden sjekker om applikasjonen kjøres for første gang.

- (void)applicationWillTerminate:(UIApplication *)application

Kjører "saveContext"-metoden. Autogenerert.

- (void)saveContext

Lagrer endringer i Core Data. Autogenerert.

- (void)applicationDidBecomeActive:(UIApplication *)application

Sender en notifikasjon til andre klasser om at applikasjonen akkurat har blitt åpnet. I tillegg slettes data fra databasen hvis denne er for stor.

- (NSManagedObjectContext *)managedObjectContext

- (NSManagedObjectModel *)managedObjectModel

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator

Disse tre metodene oppretter hver sine Core Data-relaterte objekter. Autogenererte.

- (NSNumber *)coreDataSize

Finner størrelsen på databasen.

- (NSURL *)applicationLibraryCachesDirectory

Returnerer stien til SQLite-filen.

[46] Les mer om hva NSManagedObjectContext-objektet brukes til under 3.1 Core Data.

4.2.2 Posisjon og PosisjonsKontroller

Posisjon er en klasse som samler den viktigste informasjonen fra telefonens posisjonstjeneste. Den inneholder egenskapene breddegrad, lengdegrad, hastighetIMeterISek, retning, meterOverHavet og presisjon. I tillegg inneholder klassen hjelpemetoden hastighetIKmT som konverterer hastigheten til km/t.

PosisjonsKontroller implementerer CLLocationManagerDelegate. Dette tvinger den til å implementere metodene locationManager:didUpdateLocations: og locationManager:didFailWithError:. PosisjonsKontroller mottar altså posisjoner fra enheten, oppretter et Posisjon-objekt og sender det videre med sin egen delegat, PosisjonDelegate.

PosisjonDelegate defineres altså i denne filen. Protokollen krever implementasjon av metodene posisjonOppdatering: og posisjonFeil:.

```
@protocol PosisjonDelegate
@required
- (void) posisjonOppdatering:(Posisjon *)posisjon;
- (void) posisjonFeil:(NSError *)feil;
@end

@interface PosisjonsKontroller : NSObject <CLLocationManagerDelegate>
@property (nonatomic, strong) CLLocationManager * lokMan;
@property (nonatomic, assign) id delegate;
@property (nonatomic, strong) CLLocation * sisteOppdatering;
@end

@interface Posisjon : NSObject
@property (nonatomic, strong) NSDecimalNumber * breddegrad;
@property (nonatomic, strong) NSDecimalNumber * lengdegrad;
@property (nonatomic, strong) NSDecimalNumber * hastighetIMeterISek;
@property (nonatomic, strong) NSDecimalNumber * retning;
@property (nonatomic, strong) NSDecimalNumber * meterOverHavet;
@property (nonatomic, strong) NSDecimalNumber * presisjon;
- (NSDecimalNumber *) hastighetIKmT;
@end
```

Kodesnutt 21: I Posisjon.h deklareres PosisjonDelegate og klassene PosisjonsKontroller og Posisjon.

Metodene i PosisjonsKontroller er:

`- (id) init`

Konstruktør, kalles når en instans av klassen opprettes. Initialiserer et CLLocationManager-objekt.

`- (void) locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations`

Mottar ny posisjon. Hvis enheten har beveget seg nok og det har gått nok tid siden forrige posisjon, basert på satte konstanter, opprettes det et Posisjon-objekt som sendes med delegatens posisjonOppdatering:-metode.

- (void) locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error

Kjøres hvis enheten ikke klarer å finne posisjonen. Kaller delegatens posisjonFeil:-metode.

- (void) oppdaterPresisjonMedFart:(NSDecimalNumber *)meterISekundet

Sjekker enhetens fart og oppdaterer deretter ønsket presisjon til å samsvare med ønsket tidsintervall, basert på satt konstant.

4.2.3 VegObjektKontroller

Oppgaven til VegObjektKontroller er å motta en posisjon, og deretter levere det som skal vises på denne posisjonen. Den deklarerer VegObjektDelegate som inneholder metodene vegObjekterErOppdatert: og avstandTilPunktobjekt:MedKey:. Disse brukes til å levere det som skal vises på skjermen basert på posisjonen. Den implementerer selv NVDBResponseDelegate som brukes til å motta svar på forespørsler fra NVDB_DataProvider.

Klassen sender en forespørsel videre for å finne en vegreferanse til posisjonen. Deretter oppretter den et søkeobjekt basert på denne vegreferansen som sendes videre. Den mottar så alle vegobjektene som er knyttet til denne vegen. Klassen går så gjennom disse objektene og plukker ut de som er aktuelle å vise. Samtidig sender den forespørsler videre for å finne avstanden til noen av disse. Til slutt sender den informasjon om hvilke objekter som skal vises med vegObjekterErOppdatert:-metoden og avstander fortløpende når de mottas med avstandTilPunktobjekt:MedKey:-metoden.

Metodene i VegObjektKontroller er:

- (id)initMedDelegate:(id)delegat OgManagedObjectContext:(NSManagedObjectContext *)context

Konstruktør. Setter delegaten som skal motta oppdateringene, oppretter en instans av NVDB_DataProvider og kaller settOppObjektreferanseArray-metoden.

+ (NSArray *)settOppObjektreferanseArray

Setter opp et array bestående av alle objekttypene som kan vises på skjermen i applikasjonen.

- (void)oppdaterMedBreddegrad:(NSDecimalNumber *)breddegrad OgLengdegrad:(NSDecimalNumber *)lengdegrad

Sender en forespørsel til NVDB_DataProvider etter en vegreferanse.

- (void)sokMedVegreferanse

Oppretter et søkeobjekt med vegreferanse og objekttyper. Sender deretter en forespørsel til

NVDB_DataProvider etter vegobjekter og sender med søkeobjektet og objektmappingen (se hentObjektMapping-metoden).

- (NSArray *)hentObjekttyper

Returnerer et array med Objekttype-objekter som brukes i søkeobjektet.

- (RKDynamicMapping *) hentObjektMapping

Returnerer et RKDynamicMapping-objekt som definerer hvordan RestKit skal mappe data til vegobjektene.

- (void) leggTilDataIDictionary:(NSMutableDictionary *)returDictionary FraSokeresultater:(SokResultater *)resultater MedAvstandsArray:(NSMutableArray *)avstand

Metoden sjekker om mottatt objekt skal vises til brukeren. Hvis brukeren befinner seg på linjen som defineres i et LinjeObjekt kalles leggTilLinjeDataIDictionary:MedVegObjekt:-metoden. Hvis objektet er et PunktObjekt eller et SkiltObjekt kjøres følgende test:

```
else if(self.forrigePosisjon &&
        self.forrigePosisjon.doubleValue >= 0 &&
        (naermestePosisjon.doubleValue < 0 ||
         [VegObjektKontroller diffMellomA:posisjon OgB:vLenke.fra].doubleValue <=
         [VegObjektKontroller diffMellomA:naermestePosisjon
                                    OgB:vLenke.fra].doubleValue) &&
        ((posisjon.doubleValue > self.forrigePosisjon.doubleValue &&
          vLenke.fra.doubleValue > posisjon.doubleValue &&
          ([obj isKindOfClass:[PunktObjekt class]] ||
           ([obj isKindOfClass:[SkiltObjekt class]] &&
            [((SkiltObjekt *)obj).ansiktsside
                              isEqualToString:SKILTPLATE_ANSIKTSSIDE_MED]))) ||
         (vLenke.fra.doubleValue < posisjon.doubleValue &&
          ([obj isKindOfClass:[PunktObjekt class]] ||
           ([obj isKindOfClass:[SkiltObjekt class]] &&
            [((SkiltObjekt *)obj).ansiktsside
                               isEqualToString:SKILTPLATE_ANSIKTSSIDE_MOT])))))
```

Kodesnutt 22: En komplisert test i leggTilDataIDictionary:FraSokeresultater:MedAvstandsArray:-metoden.

Dette er en veldig komplisert test. Forklaringen på testen er følgende:

A - [obj isKindOfClass:[PunktObjekt class]]

B - [obj isKindOfClass:[SkiltObjekt class]]

C - self.forrigePosisjon

D - self.forrigePosisjon.doubleValue >= 0

E - naermestePosisjon.doubleValue < 0

F - [VegObjektKontroller diffMellomA:posisjon OgB:vLenke.fra].doubleValue

<= [VegObjektKontroller diffMellomA:naermestePosisjon OgB:vLenke.fra].doubleValue

G - posisjon.doubleValue > self.forrigePosisjon.doubleValue

H - vLenke.fra.doubleValue > posisjon.doubleValue

I - vLenke.fra.doubleValue < posisjon.doubleValue

*J - [((SkiltObjekt *)obj).ansiktsside isEqualToString:SKILTPLATE_ANSIKTSSIDE_MED]*

*K - [((SkiltObjekt *)obj).ansiktsside isEqualToString:SKILTPLATE_ANSIKTSSIDE_MOT]*

Hvis vi kjører i stigende retning på en veglenke, og objektet enten er det første eller det nærmeste objektet til enhetens posisjon, samtidig som det har en høyere posisjon enn enheten (altså ligger foran enheten). Hvis objektet er et skiltobjekt må ansiktssiden være med metreringsretning[47]:

C && D && G && H && (E || F) && (A || (B && J))

Hvis vi kjører i synkende retning på en veglenke, og objektet enten er det første eller det nærmeste objektet til enhetens posisjon, samtidig som det har en lavere posisjon enn enheten (altså ligger foran enheten). Hvis objektet er et skiltobjekt må ansiktssiden være mot metreringsretning:

C && D && I && (E || F) && (A || (B && K))

Setter vi sammen disse får vi:

(C && D && G && H && (E || F) && (A || (B && J))) || (C && D && I && (E || F) && (A || (B && K)))

Med boolsk algebra finner vi at dette er ekvivalent med:

C && D && (E || F) && ((G && H && (A || (B && J))) || (I && (A || (B && K))))

Hvis testen er sann kalles enten leggTilPunktDataIDictionary:MedVegObjekt:OgAvstandsArray:-metoden eller leggTilSkiltDataIDictionary:MedVegObjekt:OgAvstandsArray:-metoden avhengig av om objektet er et PunktObjekt eller et SkiltObjekt.

- (void)leggTilLinjeDataIDictionary:(NSMutableDictionary *)returDictionary MedVegObjekt:(LinjeObjekt <VegobjektProtokoll> *)objekt

Metoden legger til navnet på LinjeObjekt-objektet i returDictionary-objektet[48]. Dette betyr at det skal vises for brukeren. Hvis det er nødvendig med mer informasjon, f.eks. hvilken fartsgrense det er, legges også dette til.

- (void)leggTilPunktDataIDictionary:(NSMutableDictionary *)returDictionary MedVegObjekt:(PunktObjekt <VegobjektProtokoll>

[47] Metreringsretningen er den stigende retningen på egn veglenke, altså når posisjonstallet går fra 0 mot 1.
[48] En dictionary er en datastruktur bestående av nøkkel- og verdipar.

***)objekt OgAvstandsArray:(NSMutableArray *)avstand**

- (void)leggTilSkiltDataIDictionary:(NSMutableDictionary *)returDictionary MedVegObjekt:(SkiltObjekt <VegobjektProtokoll> *)objekt OgAvstandsArray:(NSMutableArray *)avstand

Metodene legger til navnet på PunktObjekt- eller SkiltObjekt-objektet I returDictionary-objektet, og eventuell ekstra informasjon som er nødvendig. Metoden legger også til informasjon om objektets posisjon i avstand-objektet.

- (NSDecimalNumber *)kalkulerVeglenkePosisjon

Metoden finner enhetens posisjon på en veglenke, representert ved et nummer mellom 0 og 1. Les mer om posisjonering på veglenker under "1.2.2.1 Finne ut hvor brukeren befinner seg".

+ (NSDecimalNumber *)diffMellomA:(NSDecimalNumber *)desimalA OgB:(NSDecimalNumber *)desimalB

Metoden regner ut differansen mellom to desimaltall.

- (void)svarFraNVDBMedResultat:(NSArray *)resultat OgVeglenkeId:(NSNumber *)lenkeId

Metode definert i NVDBResponseDelegate. Metoden kalles når et svar fra NVDB_DataProvider er klart. Hvis svaret inneholder en vegreferanse kalles sokMedVegreferanse-metoden. Hvis svaret inneholder vegobjekter kjøres leggTilDataIDictionary:FraSokeresultater:MedAvstandsArray:-metoden for hvert objekt. Forespørsler om avstander sendes herfra for de objektene det gjelder. Til slutt kalles delegatens vegObjekterErOppdatert-metode med returDictionary-objektet.

- (void)svarFraMapQuestMedResultat:(NSArray *)resultat OgKey:(NSString *)key

Metode definert i NVDBResponseDelegate. Metoden kalles når et svar fra NVDB_DataProvider er klart. Delegatens avstandTilPunktobjekt:MedKey:-metode kalles med resultatet.

- (NSMutableDictionary *) opprettReturDictionaryMedDefaultVerdier

Metoden oppretter et NSMutableDictionary-objekt og setter standardverdier for alle vegobjektene.

4.2.4 NVDB_DataProvider

Oppgaven til NVDB_DataProvider er å levere dataene den blir bedt om. Den fungerer som et abstrakt lag som leverer data fra NVDB og MapQuest. Klasser som ber denne klassen om data trenger ikke tenke på hvor dataene kommer fra. NVDB_DataProvider leverer data fra Core Data hvis de eksisterer, ellers henter den data fra NVDB. Den sørger også for å lagre nye data til Core Data så de kan hentes derfra neste gang.

```
@interface NVDB_DataProvider : NSObject <NVDBResponseDelegate>
@property (nonatomic, strong) NSManagedObjectContext * managedObjectContext;

- (id)initMedManagedObjectContext:(NSManagedObjectContext *)context
                       OgAvsender:(NSObject *)aAvsender;
- (void)hentVegreferanseMedBreddegrad:(NSDecimalNumber *)breddegrad
                          OgLengdegrad:(NSDecimalNumber *)lengdegrad;
- (void)hentVegObjekterMedSokeObjekt:(Sok *)sok OgMapping:(RKMapping *)mapping;
- (void)hentAvstandmedKoordinaterAX:(NSDecimalNumber *)ax
                                 AY:(NSDecimalNumber *)ay
                                 BX:(NSDecimalNumber *)bx
                                 BY:(NSDecimalNumber *)by
                              ogKey:(NSString *)key;
@end
```

Kodesnutt 23: Headerfilen til NVDB_DataProvider. Selv om implementasjonsfilen er på over 1200 linjer og inneholder mange metoder, er det kun fire metoder som deklareres i headerfilen og dermed er synlige utenfor klassen.

Metodene i NVDB_DataProvider er :

- (id)initMedManagedObjectContext:(NSManagedObjectContext *)context OgAvsender:(NSObject *)aAvsender

Konstruktør. Mottar Core Data-konteksten og oppretter en instans av NVDB_RESTkit.

- (void)hentVegreferanseMedBreddegrad:(NSDecimalNumber *)breddegrad OgLengdegrad:(NSDecimalNumber *)lengdegrad

Spør videre mot NVDB_RESTkit etter å ha satt opp parametere og mapping for søk etter vegreferanse.

- (void)hentVegObjekterMedSokeObjekt:(Sok *)sok OgMapping:(RKMapping *)mapping

Sjekker om det finnes data for vegen i Core Data. Hvis det gjør det kalles hentVegObjekterFraCoreDataMedVeglenkeCDObjekt:-metoden. Hvis ikke kalles hentVegObjekterFraNVDBMedSokeObjekt:OgMapping:-metoden.

- (void)hentAvstandmedKoordinaterAX:(NSDecimalNumber *)ax AY:(NSDecimalNumber *)ay BX:(NSDecimalNumber *)bx BY:(NSDecimalNumber *)by ogKey:(NSString *)key

Spør videre mot NVDB_RESTkit etter å ha satt opp parametere og mapping for søk etter avstand med MapQuest.

- (void)hentVegObjekterFraNVDBMedSokeObjekt:(Sok *)sok OgMapping:(RKMapping *)mapping

Spør videre mot NVDB_RESTkit etter å ha satt opp parametere og mapping for søk etter vegobjekter.

- (void)hentVegObjekterFraCoreDataMedVeglenkeCDObjekt:
(VeglenkeDBStatus *)vlenke

Henter ut alle vegobjektene i Core Data for gitt veglenke. Objektene gjøres om fra Core Data-objekter til NVDB-objekter og settes inn i samme datastruktur som returneres fra NVDB_RESTkit-klassen. Kaller deretter delegatens svarFraNVDBMedResultat:OgVeglenkeId:-metode.

- (void)svarFraNVDBMedResultat:(NSArray *)resultat
OgVeglenkeId:(NSNumber *)lenkeId

Hvis mottatt svar inneholder vegobjekter lagres disse til Core Data. Dette gjøres ved å konvertere NVDB-objektene til Core Data-objekter. Skiltplater gjøres også om til NVDB- og Core Data-objekter. Til slutt kalles delegatens svarFraNVDBMedResultat:OgVeglenkeId:-metode.

- (void)svarFraMapQuestMedResultat:(NSArray *)resultat
OgKey:(NSString *)key

Sender svaret direkte videre ved å kalle delegatens svarFraMapQuestMedResultat:OgKey:-metode.

+ (NSArray *)gjorOmTilSkiltobjekterMedResultat:(NSArray *)resultat

Gjør om mottatte skiltplater til NVDB-objekter.

+ (NSDictionary
***)parametereForKoordinaterMedBreddegrad:(NSDecimalNumber**
***)breddegrad OgLengdegrad:(NSDecimalNumber *)lengdegrad**

Konverterer koordinater til URL-parametere i en NSDictionary.

+ (NSDictionary
***)parametereForBoundingBoxMedBreddegrad:(NSDecimalNumber**
***)breddegrad OgLengdegrad:(NSDecimalNumber *)lengdegrad**

Konverterer koordinater til parametere oppgitt som en bounding box i en NSDictionary. Metoden benyttes ikke i siste versjon av Kjørehjelperen.

+ (NSDictionary *)parametereForMapQuestAvstandMedAX:(NSDecimalNumber
***)ax AY:(NSDecimalNumber *)ay BX:(NSDecimalNumber *)bx**
OgBY:(NSDecimalNumber *)by

Konverterer koordinater til parametere i en NSDictionary.

+ (NSDictionary *)parametereForSok:(Sok *)sok

Konverterer et søkeobjekt til parametere.

4.2.5 NVDB_RESTkit

NVDB_RESTkit er ansvarlig for kommunikasjon med web-APIene. Den benytter RestKit til å gjøre spørringer mot det aktuelle APIet.

```
@protocol NVDBResponseDelegate
@required
- (void) svarFraNVDBMedResultat:(NSArray *)resultat
                    OgVeglenkeId:(NSNumber *)lenkeId;
- (void) svarFraMapQuestMedResultat:(NSArray *)resultat OgKey:(NSString *)key;
@end

@interface NVDB_RESTkit : NSObject

@property (nonatomic, assign) id delegate;

- (void) hentDataMedURI:(NSString *)uri
             Parametere:(NSDictionary *)parameter
                Mapping:(RKMapping *)mapping
                KeyPath:(NSString *)keyPath
           OgVeglenkeId:(NSNumber *)lenkeId;
- (void) hentAvstandMellomKoordinaterMedParametere:(NSDictionary *)parameter
                                           Mapping:(RKMapping *)mapping
                                             OgKey:(NSString *)key;
@end
```

Kodesnutt 24: I NVDB_RESTkit.h deklareres en protokoll som brukes til å sende svar på mottatte forespørsler når de er klare. NVDB_RESTkit inneholder to metoder, en som spør mot NVDB og en som spør mot MapQuest.

Begge metodene deklarert i headerfilen (se kodesnutt 24) bruker RestKit til å spørre mot APIer. De initialiserer RestKit, setter opp nødvendig info og mapping og utfører deretter spørringen. Når de mottar svar kjører de delegatens svarFra…-metode. Den siste parameteren i begge metodene, "lenkeId" og "key" brukes ikke i metoden, men sendes tilbake med svaret for at svaret skal kunne identifiseres med riktig spørring.

4.2.6 Vegdata_konstanter

Vegdata_konstanter er ikke en klasse, men en fil som definerer konstanter som brukes i Kjørehjelperen. Det benyttes mye testing på strenger, både i Kjørehjelperen og i NVDB APIet. Det er derfor viktig å ha disse definert på ett sted for å unngå feil og å gjøre endringer vesentlig lettere.

```
#define INGEN_OBJEKTER @"-1"
#define INGEN_EGENSKAPER @"Ingen egenskaper"
#define SKILLETEGN_SKILTOBJEKTER @"#"

#define VEGREFERANSE_BREDDEGRAD @"breddegrad"
#define VEGREFERANSE_LENGDEGRAD @"lengdegrad"

#define FARTSGRENSE_ID 105
#define FARTSGRENSE_KEY @"fartsgrense"
#define FARTSGRENSE_CD @"CD_Fartsgrense"
#define FARTSGRENSE_FART_KEY @"Fartsgrense"
#define FARTSGRENSE_BRUKERPREF @"skilt_fartsgrense"
```

Kodesnutt 25: Et utdrag fra Vegdata_konstanter. Konstantene brukes både til sammenlikninger i Kjørehjelperen, tilknytning av objekter til Core Data og brukerpreferanser, og til spørringer mot NVDB APIet.

4.2.7 NVDB-objekter

NVDB-objektene er objekter som brukes til å mappe data til eller fra JSON med RestKit. Sok-klassen definerer objektet som sendes med spørringer til NVDB APIet. MapQuestRoute er objektet svar fra MapQuest mappes til. De andre objektene er stort sett vegobjekter som data fra NVDB mappes til.

Vegobjektene er subklasser av enten LinjeObjekt, PunktObjekt eller SkiltObjekt. Disse klassene er igjen subklasser av Vegobjekt hvor de viktigste egenskapene defineres. Vegobjekt inneholder blant annet to arrayer med Egenskap- og Veglenke-objekter. Grunnen til at vegobjektene arver fra enten LinjeObjekt, PunktObjekt eller SkiltObjekt er at disse typene behandles ulikt i VegObjektKontroller-klassen. Man kan dermed teste hvilken av disse typene objektet er en subklasse av, og dermed slippe å teste på alle mulige NVDB-objekttyper.

```
@protocol VegobjektProtokoll <NSObject>
@required
+ (RKObjectMapping *)mapping;
+ (NSArray *)filtere;
+ (NSNumber *)idNr;
+ (NSString *)key;
+ (BOOL)objektSkalVises;
@end

@interface Vegobjekt : NSObject
@property (nonatomic, strong) NSArray * egenskaper;
@property (nonatomic, strong) NSArray * veglenker;
@property (nonatomic, strong) NSString * lokasjon;
+ (RKObjectMapping *)standardMappingMedKontainerKlasse:(Class)kontainerklasse;
@end

@interface LinjeObjekt : Vegobjekt
@property (nonatomic, strong) NSNumber * strekningsLengde;
@end

@interface PunktObjekt : Vegobjekt
@end
```

Kodesnutt 26: I Vegobjekt.h defineres VeobjektProtokoll, Vegobjekt, LinjeObjekt og PunktObjekt. SkiltObjekt defineres i en annen fil (SkiltObjekt.h).

VegobjektProtokoll implementeres av de fleste vegobjektene. Denne protokollen sørger for at klassene implementerer metoder som returnerer info om objektet til RestKit. Man kan dermed behandle alle vegobjekter likt uten å vite hva slags objekt det er snakk om.

```
@implementation Fartsgrense

@synthesize strekningsLengde, egenskaper, veglenker, lokasjon;

- (NSString *)hentFartFraEgenskaper
{
    for (Egenskap * e in egenskaper)
    {
        if([e.navn isEqualToString:FARTSGRENSE_FART_KEY])
        {
            return e.verdi;
```

```
        }
    }
    return INGEN_OBJEKTER;
}

+ (RKObjectMapping *)mapping
{
    return [self standardMappingMedKontainerKlasse:[Fartsgrenser class]];
}

+ (NSArray *)filtere {return nil;}

+ (NSNumber *)idNr {return [[NSNumber alloc] initWithInt:FARTSGRENSE_ID];}

+ (NSString *)key {return FARTSGRENSE_KEY;}

+ (BOOL)objektSkalVises
{
    return [[NSUserDefaults standardUserDefaults]
             boolForKey:FARTSGRENSE_BRUKERPREF];
}

@end
```

Kodesnutt 27: Fartsgrense arver fra LinjeObjekt og implementerer VegobjektProtokoll. Vi trenger derfor ikke deklarere egenskapene strekningsLengde, egenskaper, veglenker og lokasjon i headerfilen. Vi må implementere metodene fra VegobjektProtokoll som vi ser at er gjort her i implementasjonsfilen.

Metodene i et standard vegobjekt er:

`+ (RKObjectMapping *)mapping`

Returnerer mappingen som brukes for å mappe JSON-data til objektet med RestKit.

`+ (NSArray *)filtere`

Hvis det benyttes filter[49] ved søk etter objektet i NVDB returneres dette av denne metoden.

`+ (NSNumber *)idNr`

Returnerer identifikasjonsnummeret til objektet i NVDB. Nummeret er definert i Vegdata_konstanter.

`+ (NSString *)key`

Returnerer strengen som brukes til å identifisere objektet innad i Kjørehjelperen.

`+ (BOOL)objektSkalVises`

Returnerer hvorvidt objektet skal vises for brukeren eller ikke, basert på brukerinnstillingene i applikasjonen.

4.2.8 SokResultater

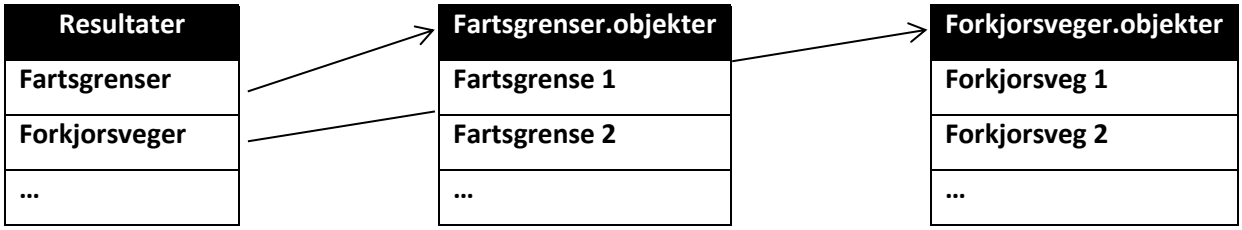
```
@interface SokResultater : NSObject
@property (nonatomic, strong) NSArray * objekter;
-(id)initMedObjekter:(NSArray *)aObjekter;
```

[49] Les om bruk av filter i NVDB under "1.2.2.2 Finne objektene på den vegen vi befinner oss".

```
@end

@interface Fartsgrenser : SokResultater
@end

@interface Forkjorsveger : SokResultater
@end

...
```

Kodesnutt 28: SokResultater.h. Her ser vi SokResultater og to av subklassene, Fartsgrenser og Forkjorsveger.

SokResultater er en klasse som kun inneholder et array. Den har en subklasse for hver enkelt NVDB-objekttype. Klassen Fartsgrenser inneholder for eksempel kun et array bestående av Fartsgrense-objekter. Grunnen til at vegobjektene pakkes inn i sine respektive kontainerklasser i stedet for et vanlig array er at dette muliggjør testing på type på et høyere nivå. Det lar oss også mappe mange objekter til en sortert struktur i RestKit.



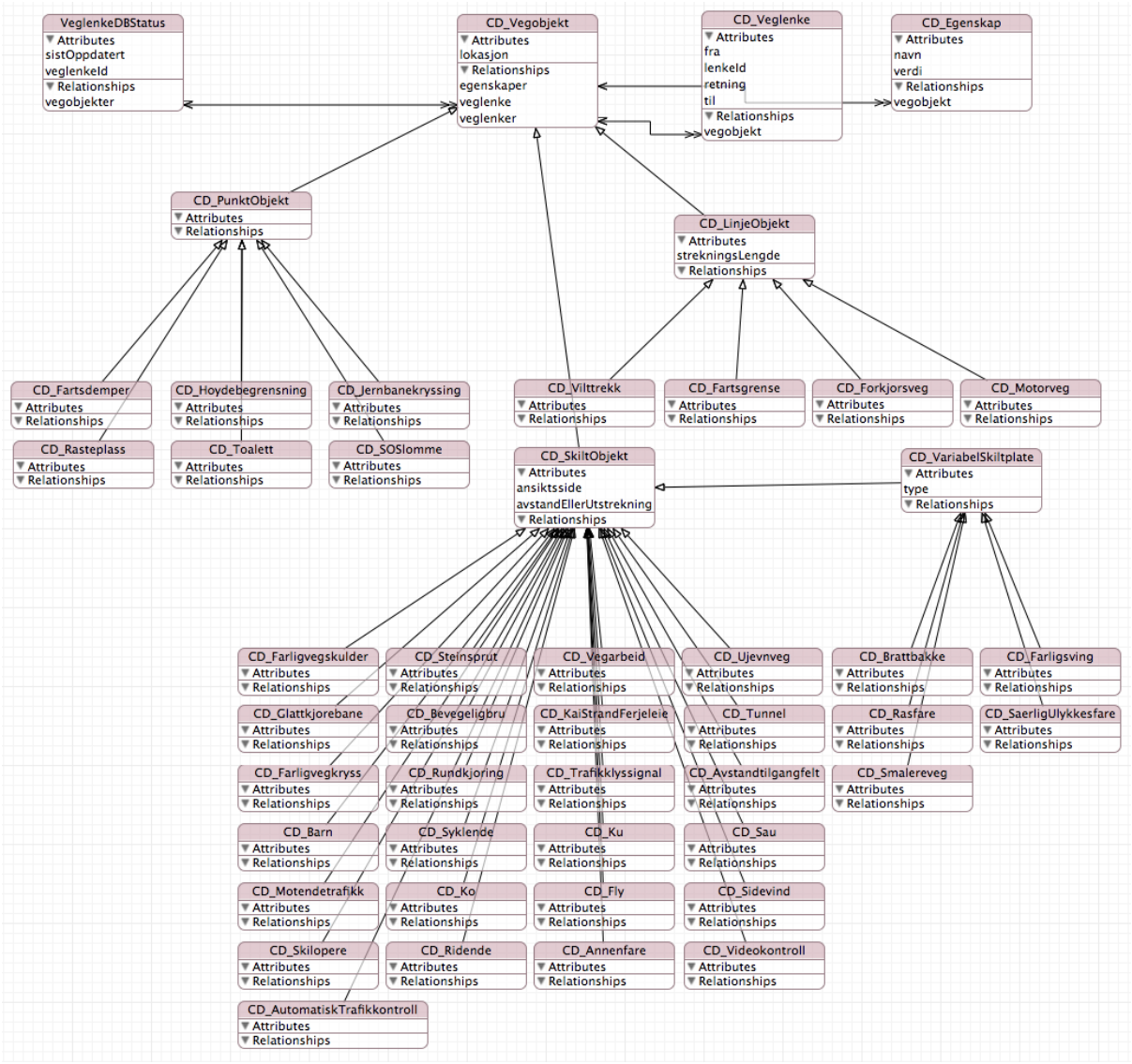
Figur 12: Arrayet "Resultater" som returneres av RestKit inneholder et SokResultater-objekt for hver av NVDB-objektene. Disse objektene inneholder igjen et array med sine respektive Vegobjekt-objekter.

4.2.9 Core Data-objekter

Core Data-objektene er autogenerert av Core Data. De speiler entitetene og forholdet mellom dem. Entitetene speiler igjen NVDB-objektene. Unntaket er VeglenkeDBStatus som representerer én veglenke, et tidspunkt for oppdatering og et sett med alle de tilhørende vegobjektene.

```
@interface VeglenkeDBStatus : NSManagedObject
@property (nonatomic, retain) NSDate * sistOppdatert;
@property (nonatomic, retain) NSNumber * veglenkeId;
@property (nonatomic, retain) NSSet * vegobjekter;
@end

@interface VeglenkeDBStatus (CoreDataGeneratedAccessors)
- (void)addVegobjekterObject:(CD_Vegobjekt *)value;
- (void)removeVegobjekterObject:(CD_Vegobjekt *)value;
- (void)addVegobjekter:(NSSet *)values;
- (void)removeVegobjekter:(NSSet *)values;
@end
```

Kodesnutt 29: Den autogenererte headerfilen til VeglenkeDBStatus.



Figur 13: Den grafiske fremstillingen av entitetene og relasjonene mellom dem i Core Data-datamodellen i Kjørehjelperen.

I Kjørehjelperen er NVDB- og Core Data-objektene separate. Det konverteres manuelt fra den ene typen til den andre ved lasting og lagring. Tidsbegrensninger førte til denne implementasjonen i Kjørehjelperen, men man kunne implementert vegobjekter som både lot seg mappe med RestKit og lagre til Core Data.

```
@interface CD_Vegobjekt : NSManagedObject
@property (nonatomic, retain) NSString * lokasjon;
@property (nonatomic, retain) NSSet *egenskaper;
@property (nonatomic, retain) VeglenkeDBStatus *veglenke;
@property (nonatomic, retain) NSSet *veglenker;
@end

@interface CD_Vegobjekt (CoreDataGeneratedAccessors)
- (void)addEgenskaperObject:(CD_Egenskap *)value;
- (void)removeEgenskaperObject:(CD_Egenskap *)value;
- (void)addEgenskaper:(NSSet *)values;
- (void)removeEgenskaper:(NSSet *)values;
```

```
- (void)addVeglenkerObject:(CD_Veglenke *)value;
- (void)removeVeglenkerObject:(CD_Veglenke *)value;
- (void)addVeglenker:(NSSet *)values;
- (void)removeVeglenker:(NSSet *)values;
@end
```

Kodesnutt 30: Headerfilen til CD_Vegobjekt.

4.2.10 HovedskjermViewController

HovedskjermViewController styrer hele applikasjonen. Den oppretter instanser av PosisjonsKontroller og VegObjektKontroller. Videre har den kontroll over alle de grafiske elementene på skjermen. Klassen mottar posisjoner fra PosisjonsKontroller og sender disse videre til VegObjektKontroller. Den mottar deretter informasjon om vegobjekter og viser dette på skjermen.

HovedskjermViewControllerPortrett og HovedskjermViewControllerLandskap er to subklasser av HovedskjermViewController. De inneholder altså de samme metodene og egenskapene som sin forelder. Kjørehjelperen startes i "Hoved…Portrett". Samtidig som denne gjør forelderens oppgaver, lytter den samtidig etter orienteringsendringer. Første gang den oppdager at telefonen er i landskapsmodus oppretter den "Hoved…Landskap". Deretter flyttes data og pekere frem og tilbake mellom disse to avhengig av telefonens orientering.

Grunnen til at Kjørehjelperen har en egen ViewController for hver orientering er at viewene, altså skjermbildene, er ganske forskjellige. Apple anbefaler[50] i slike tilfeller å bruke to ViewControllere i stedet for å flytte på elementer programmatisk.

HovedskjermViewController inneholder følgende metoder:

- (void)viewDidLoad

Kjøres når applikasjonen starter. Initialiserer klasser og strukturer, blant annet opprettes VegObjektKontroller og PosisjonsKontroller.

-(BOOL)shouldAutorotate

Forteller applikasjonen om den skal skifte orientering automatisk. Hvis HUD[51] er aktivert svarer metoden NO.

- (void)settOppLayoutArray

Setter opp et flerdimensjonalt array som holder på alle GUI-elementene[52] i applikasjonen.

[50] http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/RespondingtoDeviceOrientationChanges/RespondingtoDeviceOrientationChanges.html

[51] Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.

[52] Graphical user interface.

- (IBAction)hudKnappTrykket:(UISwitch *)knapp

Kjøres når HUD skrus av eller på. Speilvender alle elementer eller setter dem tilbake til normalt perspektiv.

- (BOOL)erFireTommerRetina

Forteller om telefonen har en 4" skjerm eller ikke.

- (BOOL)skalViseHUDKnapp

Leser fra brukerinnstillingene og forteller om HUD-knappen skal vises på skjermen eller ikke.

- (void) posisjonOppdatering:(Posisjon *)posisjon

Mottar en ny posisjon som sendes videre til VegObjektKontroller.

- (void) posisjonFeil:(NSError *)feil

Mottar feilmelding fra posisjonstjenesten og viser en feilmelding til brukeren.

- (void) vegObjekterErOppdatert:(NSDictionary *)data

Mottar en NSDictionary med data som skal vises til brukeren. Fjerner gamle data og fyller alle plassene på skjermen med ny data og grafikk.

- (void)avstandTilPunktobjekt:(NSDecimalNumber *)avstand MedKey:(NSString *)key

Mottar en avstand til et Punkt- eller SkiltObjekt. Finner riktig objekt på skjermen og viser avstanden.

+ (NSString *)finnTypeMedStreng:(NSString *)streng

Hjelpemetode som tolker en streng tilhørende et SkiltObjekt sendt fra VegObjektKontroller.

+ (NSString *)finnAvstandstekstMedStreng:(NSString *)streng ErVariabeltSkilt:(BOOL)erVariabelt

Hjelpemetode som tolker en streng tilhørende et SkiltObjekt sendt fra VegObjektKontroller.

I tillegg inneholder HovedskjermViewControllerPortrett følgende metoder:

- (void)viewDidLoad

Kaller forelderens metode med samme navn, og setter bakgrunnsbilde på skjermen.

- (void)awakeFromNib

Som viewDidLoad kalles også denne metoden automatisk når applikasjonen starter. I denne metoden settes det opp et NSNotificationCenter som lytter etter endringer i orienteringen.

- (void)orienteringEndret:(NSNotification *)notifikasjon

Kalles når orienteringen endres. Oppretter HovedskjermViewControllerLandskap ved første kjøring, overfører deretter data og pekere til den ViewControlleren som nå skal vises for brukeren.

-(NSUInteger)supportedInterfaceOrientations

Returnerer orienteringene som støttes av denne ViewControlleren, altså portrett.

- (void)hudKnappTrykket:(UISwitch *)knapp

Kalles når HUD-knappen trykkes. Siden HUD-modus alltid vises i landskapsmodus endrer denne metoden til landskapsmodus, og kaller HovedskjermViewControllerLandskaps hudKnappTrykket:-metode.

HovedskjermViewControllerLandskap inneholder i tillegg følgende metoder:

- (void)viewDidLoad

Kaller settOppLayoutArray:-metoden og setter bakgrunnsbilde på skjermen.

- (void)hudKnappTrykket:(UISwitch *)knapp

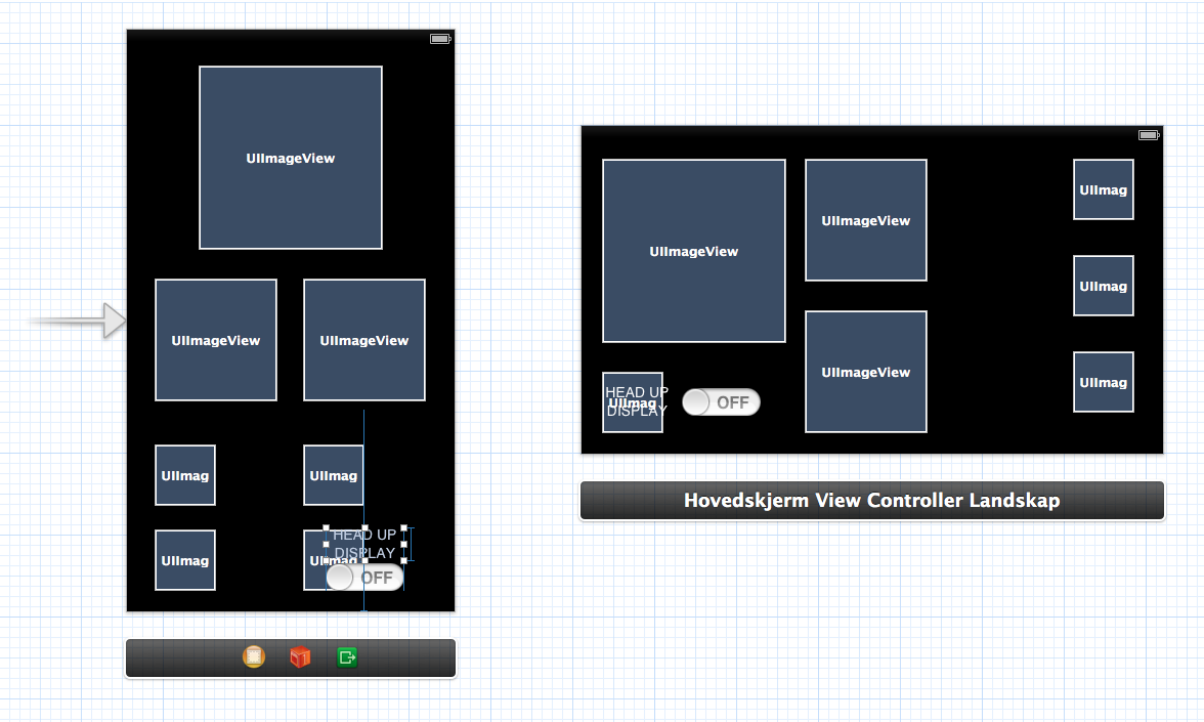
Kaller forelderens hudKnappTrykket:-metode og endrer bakgrunnsbildet.

-(NSUInteger)supportedInterfaceOrientations

Returnerer orienteringene som støttes av denne ViewControlleren, altså landskap.

4.2.11 MainStoryboard

Storyboardet inneholder informasjon om viewene i applikasjonen. Det består som sagt av autogenerert XML basert på en grafisk editor i Xcode.



Figur 14: Figuren viser den grafiske storyboard-editoren i Xcode og de to viewene for portrett- og landskapsmodus.

4.2.12 Settings

Under Settings.bundle/ ligger Root.plist. Her kan man endre hvilke innstillinger som gjelder for applikasjonen. I likhet med MainStoryboard består denne av autogenerert XML basert på en editor i Xcode. Denne editoren er tekstbasert, men tilbyr et forenklet grensesnitt for brukeren, i forhold til å redigere innholdet. Her er det feltet Key som benyttes når en innstilling så skal leses og benyttes av selve applikasjonen. Det finnes flere typer innstillinger, men vi benytter kun to: ToggleSwitch (bryter) og TextField (tekstfelt). I tillegg benytter vi Group (gruppe), men dette fungerer kun som en grupperingsmekanisme for innstillingene, og gir en overskrift for gruppen.

Key	Type	Value
iPhone Settings Schema	Dictionary	(3 items)
Settings Page Title	String	RootPList
Preference Items	Array	(46 items)
Item 0 (Group – Generelt)	Dictionary	(2 items)
Item 1 (Toggle Switch –	Dictionary	(4 items)
Type	String	Toggle Switch
Title	String	Lydvarsling
Identifier	String	lydvarsling
Default Value	Boolean	YES
Item 2 (Toggle Switch – HUD–	Dictionary	(4 items)
Item 3 (Text Field –	Dictionary	(5 items)
Item 4 (Group – Skilt (generelt))	Dictionary	(2 items)
Item 5 (Toggle Switch –	Dictionary	(4 items)
Item 6 (Toggle Switch –	Dictionary	(4 items)
Item 7 (Toggle Switch –	Dictionary	(4 items)
Item 8 (Toggle Switch –	Dictionary	(4 items)
Item 9 (Toggle Switch –	Dictionary	(4 items)
Item 10 (Toggle Switch –	Dictionary	(4 items)

Figur 15: Grensesnittet for opprettelse av brukerinnstillinger i Xcode.

Når en innstilling skal leses og benyttes i applikasjonen hentes den ut på denne måten:

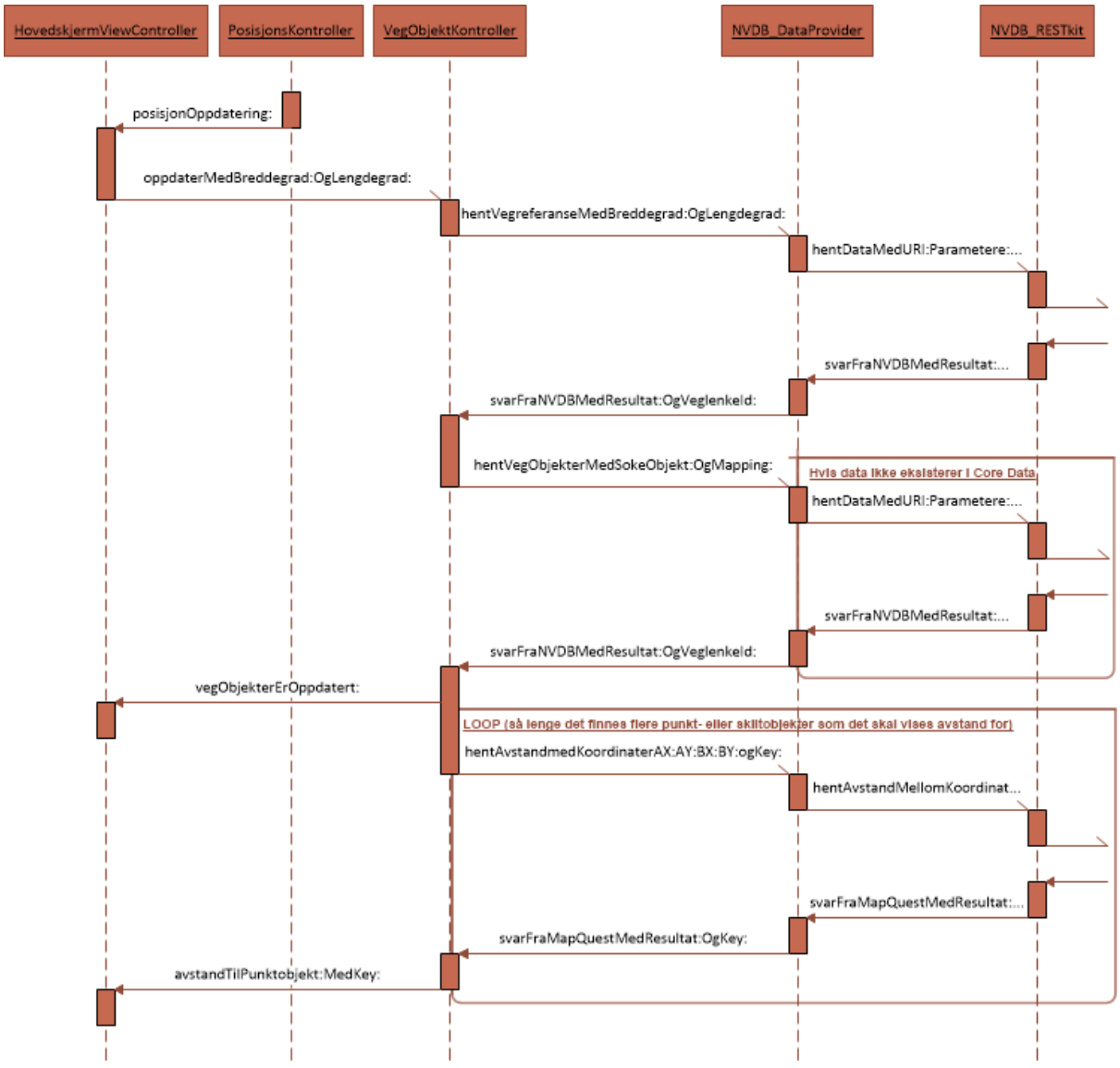
*NSString * enInnstilling = [[NSUserDefaults standardUserDefaults] stringForKey:@"keyTilInnstilling"];*

Innstillinger kan også leses ut som bool, int, osv. Da bytter man enkelt ut stringForKey med boolForKey, osv.

4.3 Gangen i applikasjonen

Når applikasjonen starter kjøres *application:didFinishLaunchingWithOptions:* og *applicationDidBecomeActive:* i AppDelegate-klassen. Deretter kjøres *viewDidLoad* i HovedskjermViewController-klassen. Les mer om hva som gjøres i disse metodene i "4.2.1 AppDelegate" og "4.2.10 HovedskjermViewController".

Nå er Kjørehjelperen klar til å levere data til brukeren. Det første som skjer er at PosisjonsKontroller leverer et nytt Posisjon-objekt til HovedskjermViewController. Denne klassen sender deretter Posisjon-objektet videre til VegObjektKontroller og er ferdig med sin oppgave inntil videre.



Figur 16: Sekvensdiagrammet viser en normalkjøring fra HovedskjermViewController mottar en posisjon til alle data vises på skjermen for brukeren.

VegObjektKontroller sender posisjonen videre til NVDB_DataProvider for å få en vegreferanse knyttet til posisjonen. NVDB_DataProvider sender dette videre til NVDB_RESTkit, som igjen sender en forespørsel til NVDB APIet. Siden det benyttes delegater er det ingen klasser som venter på svar. De våkner først når delegatens metode kalles.

Når RESTkit mottar et svar fra NVDB APIet mappes dette automatisk til et Vegreferanse-objekt, og sendes til NVDB_DataProvider. NVDB_DataProvider sender svaret videre til VegObjektKontroller. VegObjektKontroller bygger deretter et søkeobjekt med den aktuelle veglenke-id'en og alle vegobjektene med deres respektive mapping. Søkeobjektet sendes så til NVDB_DataProvider.

Når NVDB_DataProvider mottar søkeobjektet sjekker klassen om det eksisterer data for denne veglenken i Core Data og om den eventuelt er utdatert (se figur 16). Hvis data eksisterer og ikke er utdatert lastes dette ut av Core Data og sendes tilbake til VegObjektKontroller. Ellers sendes forespørselen videre til NVDB_RESTkit.

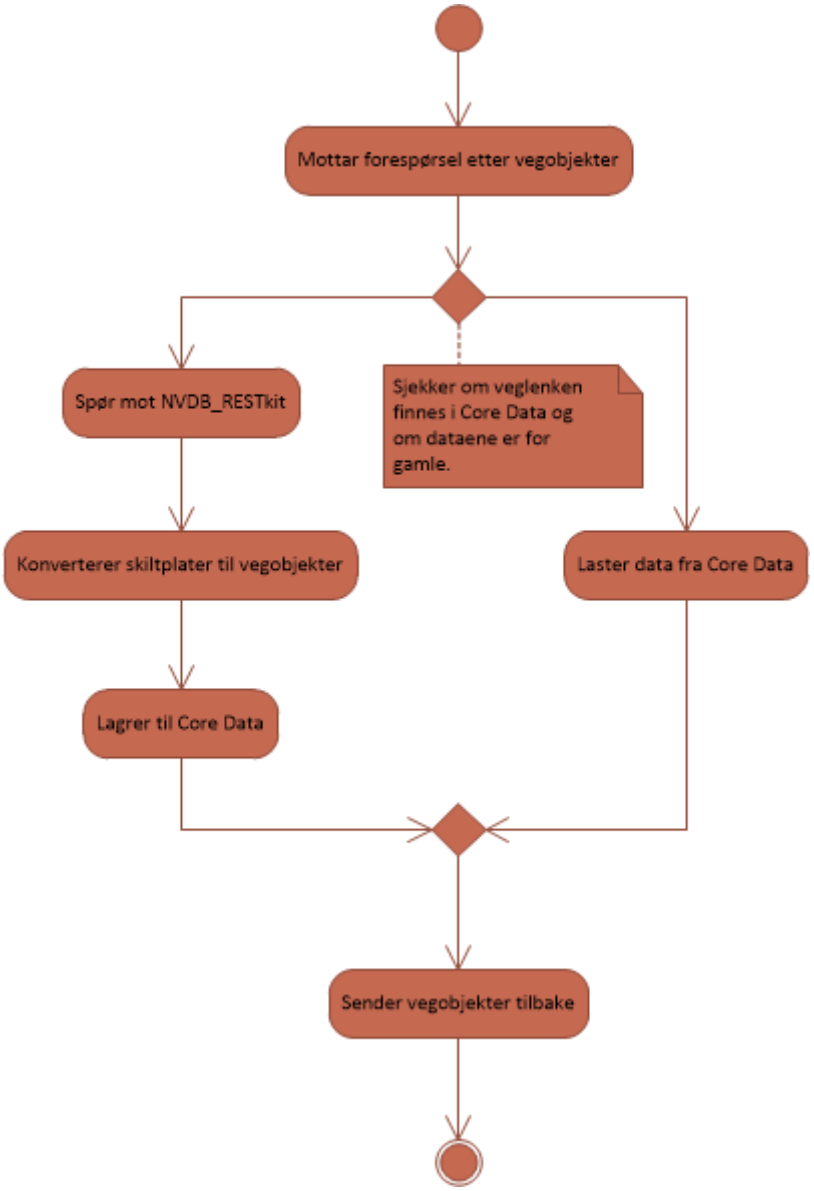
NVDB_RESTkit sender forespørselen videre til NVDB APIet. Når svar mottas, mappes det til vegobjekter og sendes tilbake til NVDB_DataProvider. Når NVDB_DataProvider mottar dette gjøres først skiltobjektene om til vegobjekter, før alt lagres til Core Data. Så sendes resultatet tilbake til VegObjektKontroller.

Når VegObjektKontroller mottar vegobjektene vet ikke klassen noe om de kom fra Core Data eller NVDB APIet. Den løper gjennom dataene og finner frem til det som skal vises til brukeren. Dette legges i en NSDictionary og returneres til HovedskjermViewController. Samtidig kjøres det en forespørsel mot NVDB_DataProvider for å finne avstanden til alle Skilt- og Punktobjektene som skal vises på skjermen.

For hver enkelt avstandsforespørsel sender NVDB_DataProvider denne videre til NVDB_RESTkit. NVDB_RESTkit sender svaret tilbake til NVDB_DataProvider, som igjen sender svaret tilbake til VegObjektKontroller. Denne klassen formaterer svaret og sender det til HovedskjermViewController.

HovedskjermViewController mottar en NSDictionary med alle vegobjektene som skal vises på skjermen. Klassen tegner ut informasjonen på skjermen og stopper opp igjen. Med jevne mellomrom mottar den deretter avstander som skrives ut på skjermen etter hvert som de mottas.

Dette er gangen i applikasjonen når alt går bra. Det hele repeteres omtrent hvert andre sekund. Et sekvensdiagram (figur 15) viser hvordan klassene kommuniserer. Etter alle forespørsler mot venstre i diagrammet stopper klassene opp og gjør ikke noe mer før en delegatmetode kalles med svar.



Figur 17: Aktivitetsdiagrammet viser hva NVDB_DataProvider gjør når den mottar en forespørsel etter vegobjekter.

5 Grafisk brukergrensesnitt

Flere faktorer var viktige med tanke på det grafiske brukergrensesnittet. Vi skulle følge de syv designprinsippene[53], og i tillegg ta hensyn til Apples retningslinjer for design på iOS[54]. Det var også viktig å tenke på brukervennlighet. Vi skulle følge de fem E'ene[55] og i tillegg tenke på trafikksikkerhet under bruk.

5.1 De syv designprinsippene

De syv designprinsippene er structure, simplicity, consistency, tolerance, visibility, affordance og feedback. De er prinsipper for hva som er god design.

- **Structure** refererer til strukturen og organiseringen av innholdet
- **Simplicity** dreier seg om at informasjonen som vises er tydelig og relevant
- **Consistency** går på at designet er gjennomgående i applikasjonen
- **Tolerance** er toleransen for, og konsekvenser av brukerfeil
- **Visibility** dreier seg om hvorvidt det er opplagt hva de forskjellige funksjonene gjør
- **Affordance** dreier seg om hvorvidt det er opplagt hvordan man bruker funksjonaliteten
- **Feedback** refererer til hvorvidt brukeren får passende tilbakemelding på det han/hun gjør

Siden applikasjonen består av ett skjermbilde med skiltplater som vises til brukeren, og i utgangspunktet ingen krav om interaksjon, oppfylles de fleste av disse prinsippene automatisk. Innholdet er organisert på en strukturert måte, det vises ingen unødvendig informasjon, designet er gjennomgående, og funksjonaliteten og bruken er selvforklarende.

Det er få muligheter for brukeren til å gjøre feil i applikasjonen. Hvis posisjonstjenesten ikke klarer å levere en posisjon får brukeren beskjed om dette, med et hint om at applikasjonen trenger tilgang til denne tjenesten. Denne feilen kan riktignok også oppstå hvis brukeren f.eks. befinner seg i en tunnel. Det kreves interaksjon når denne feilen inntreffer (brukeren må trykke på "OK"), men vi har vurdert det til at denne feilen sjeldent forekommer. Skjer det en feil med Core Data, NVDB APIet eller liknende vil brukeren få en melding om at applikasjonen ikke fant noe data. Applikasjonen vil prøve igjen etter ca. to sekunder. Meldingen forsvinner så snart applikasjonen klarer å hente data. Les mer om feilhåndteringen i Testdokumentasjonen.

[53] Structure, simplicity, consistency, tolerance, visibility, affordance og feedback (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).
[54] https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html.
[55] Prinsipper for brukervennlighet: effective, efficient, engaging, error tolerant og easy to learn (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).

Figur 18: Denne feilmeldingen dukker opp når posisjonstjenesten ikke klarer å finne enhetens posisjon.

Figur 19: Denne feilmeldingen dukker opp når det skjer en feil i NVDB APIet eller Core Data.

5.2 Apples retningslinjer for design på iOS

Apples retningslinjer for design på iOS er tilgjengelig på https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html. De går i korthet ut på at man må tenke på hvilke tjenester man tilbyr og til hvilken målgruppe, og tilpasse designet deretter. Designet er viktig, og man må legge vekt på detaljer. Man må også, når man benytter telefonens tjenester og teknologier, bruke dem på en måte som brukere forventer i iOS. Det er også et krav om at applikasjonen må ha unike ikoner og grafikk som skal vises i App Store[56].

I Kjørehjelperen bruker vi trafikkskilt til å vise informasjon. Siden målgruppen er sjåfører og informasjonen som vises dreier seg om veg og trafikk føler vi at dette er et riktig valg. Vi har også designet et eget ikon til applikasjonen som vises på telefonens startskjerm og i App Store.

5.3 De fem E'ene

De fem E'ene er prinsipper for god brukervennlighet. E'ene står for effective, efficient, engaging, error tolerant og easy to learn.

[56] App Store er Apples digitale butikk for applikasjoner og spill.

- **Effective** dreier seg om hvorvidt applikasjonen lar brukeren foreta seg de handlingene han/hun har som mål å utføre
- **Efficient** går på hastigheten og nøyaktigheten utførelsen av handlinger kan gjøres
- **Engaging** dreier seg om hvor harmonisk, engasjerende og tilfredsstillende brukergrensesnittet oppleves for brukeren
- **Error tolerant** går på hvor godt designet hindrer feil, og eventuelt hjelper brukeren videre etter en feil
- **Easy to learn** refererer til hvor lett det er å orientere seg og forstå applikasjonens muligheter

Som med de syv designprinsippene tilfredsstilles de fleste av disse kravene automatisk siden applikasjonen kun har ett skjermbilde og det i utgangspunktet ikke kreves noen interaksjon. Hvor harmonisk, engasjerende og tilfredsstillende brukergrensesnittet oppleves for brukeren er i stor grad subjektivt. Det er allikevel forsøkt å designe skjermbildet på en måte som er oversiktlig og passer til innholdet. Skiltene er store og tydelige, og de viktigste skiltene er størst når det vises mange skilt på skjermen. Bakgrunnen er en svak asfalttekstur. Denne skal være med på å forsterke "veg"-tematikken i applikasjonen, samtidig som at den ikke skal være forstyrrende og ta oppmerksomheten bort fra skiltene. Som nevnt under "5.1 De syv designprinsippene" skal ikke applikasjonen stoppe opp ved feil. Brukeren skal få en passende feilmelding, og applikasjonen skal deretter fortsette å kjøre som normalt.

Figur 20: Ikonet til Kjørehjelperen. Dette vises på telefonens startskjerm og i App Store.

5.4 Trafikksikkerhet

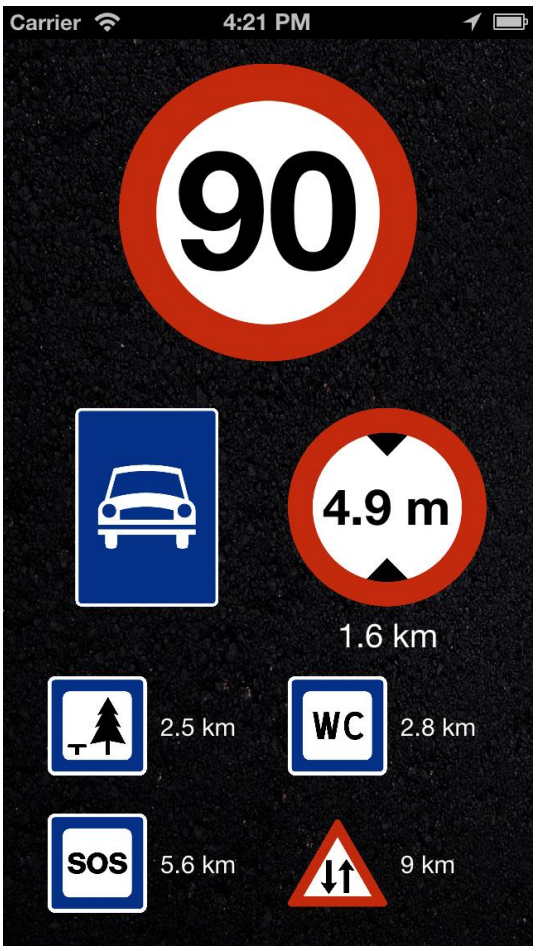
De nye reglene[57] for bruk av mobiltelefon i bil slår fast at mobiltelefonen ikke bare må være fastmontert, men også at det eneste du kan foreta deg under kjøring er å ta samtaler ut og inn og legge på. Applikasjonen er derfor designet slik at du starter den mens du står stille, og deretter skal den ikke kreve noen interaksjon. Unntaket er som nevnt en feilmelding som må klikkes "OK" på når telefonen ikke klarer å finne en posisjon, men dette skal i liten grad skje.

[57] De nye reglene tredde i kraft 01.05.2013.

5.5 Skjermbildet i Kjørehjelperen

Skjermbildet i Kjørehjelperen har en svak asfalttekstur som bakgrunn, og skiltplater med tilhørende tekst under eller ved siden av. De viktigste skiltplatene vises størst, mens når det blir mange skilt samtidig blir de påfølgende skiltene mindre.

Figur 21: Et skjermbilde av applikasjonen i bruk. På en 3,5" skjerm er det kun plass til fire skilt når HUD-knappen vises.



Figur 22: Dette skjermbildet viser den samme posisjonen som figur 20, men på en 4" iPhone 5. Her er også HUD-knappen skult, så her er det plass til syv skilt.

HUD-knappen vises på skjermen for å gjøre bytte mellom HUD-modus og vanlig modus raskt. Hvis brukeren ikke benytter seg av HUD-modusen lar knappen seg skjule i brukerinnstillingene. Dette gir plass til et ekstra skilt. Eier brukeren en iPhone 5 er det i tillegg plass til to ekstra skilt. Dette er fordi iPhone 5 er 4", mot de tidligere modellenes 3,5", og er derfor litt lengre.

I landskapsmodus er det like mange skiltplasser som i portrettmodus. De er riktignok plassert litt annerledes for å få plass til teksten som viser avstanden til objektet.

Figur 23: En 3,5" skjerm i landskapsmodus med HUD-knappen synlig.

Figur 24: En 4" skjerm i landskapsmodus med HUD-knappen skjult.

Når HUD-modus aktiveres tvinges telefonen over i landskapsmodus. Deretter speiles all informasjonen (bortsett fra HUD-knappen) slik at den fremstår riktig når den reflekteres i frontruten.

Figur 25: En 3,5" skjerm i HUD-modus.

Innstillingsskjermen til applikasjonen følger standarddesignet til iOS. Når man benytter brukerinnstillinger i en applikasjon blir disse plassert sammen med innstillinger for andre applikasjoner på telefonen, og får samme utforming som disse.

Figur 26: Innstillingene til Kjørehjelperen ligger sammen med innstillingene til andre applikasjoner på telefonen.

Figur 27: Et utdrag av innstillingene til Kjørehjelperen.

6 Gjenstående arbeid

Selv om Kjørehjelperen er tilnærmet ferdig er det noen ting som gjenstår. NVDB APIet som benyttes er ikke lansert enda. Når dette lanseres er ikke kjent, men det vil trolig skje i løpet av noen få måneder. Inntil dette lanseres kan heller ikke Kjørehjelperen lanseres i App Store.

Før lansering ønsker vi også å gjøre størrelsen på skiltplatene mer dynamiske. Vi ønsker at alle skiltene skal være like store, og at størrelsen reguleres etter hvor mange skilt som vises på skjermen. Dagens løsning fungerer, men kan oppfattes litt rotete, spesielt i landskapsmodus.

Det vil også foretas flere tester av applikasjonen for å finne feil i både den og NVDB APIet. Feil i applikasjonen er alvorlig, ikke bare kan dette gi et negativt syn på Kjørehjelperen og i verste fall Statens vegvesen, men det kan også være et spørsmål om trafikksikkerhet. Selv om applikasjonen frasier seg ansvar, kan det være meget uheldig om den skulle vise uriktig informasjon om en vegstrekning.

2013

Kjørehjelperen

Testdokumentasjon

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Dette dokumentet tar for seg to forskjellige ting. Først forklares det hvordan programmet håndterer feil og melder dette til brukeren. Deretter dokumenteres de fysiske testene av programmet, og resultatet av disse.

Innholdsfortegnelse

1 Feilhåndtering i programmet

Det er mange feil som kan oppstå i et program, og det er viktig å gi informativ tilbakemelding til brukeren. Samtidig har vi forsøkt å lage en applikasjon som krever minst mulig interaksjon under bruk med tanke på trafikksikkerhet. Vi har derfor valgt å samle feilene under to ulike feilmeldinger.

Hvis telefonen ikke klarer å finne posisjonen den befinner seg på, opprettes det en feilmelding. Dette er en dialogboks som sier: "Klarte ikke finne enhetens posisjon. Har Kjørehjelperen tillatelse til å bruke posisjonstjenester?". Brukeren må trykke "OK" for at meldingen skal forsvinne. Denne feilen kan oppstå hvis brukeren ikke har gitt Kjørehjelperen tillatelse til å bruke telefonens posisjonstjeneste. Den kan også oppstå hvis posisjonstjenesten er deaktivert på telefonen. Hvis brukeren befinner seg et sted GPS-tjenesten[1] ikke får kontakt med satellittene vi også denne feilmeldingen dukke opp, som f.eks. inne i en lang tunnel.

Figur 1: Denne feilmeldingen dukker opp hvis telefonen ikke klarer å finne posisjonen den befinner seg på.

Selv om denne feilmeldingen krever interaksjon fra brukeren er det antatt at feilen vil oppstå sjeldent under bruk. Feilmeldingen vil dessuten ikke fortsette å komme opp etter den først er vist én gang.

Hvis det skjer en annen feil, for eksempel at NVDB[2] APIet[3] ikke svarer, at RestKit[4] ikke klarer å mappe[5] resultatet, at det er en feil med Core Data[6] eller at det rett og slett ikke finnes noe data for posisjonen, vises "Fant ingen data om din lokasjon. Prøver igjen straks!". Denne feilmeldingen krever ingen interaksjon, og forsvinner så snart applikasjonen klarer å hente nye data. Vi har kommet frem

[1] NAVSTAR Global Positioning System (GPS) er et nettverk av satellitter som er plassert i bane rundt Jorden av det amerikanske forsvaret. Systemet gjør det mulig for en mottaker å fastsette egen posisjon med svært stor nøyaktighet overalt i verden, under nær sagt alle værforhold.
[2] Nasjonal vegdatabank, database med vegobjekter forvaltet av Statens Vegvesen.
[3] Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.
[4] RestKit er et Objective-C-rammeverk for iOS som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen av objekter.
[5] Mapping er prosessen med å overføre data fra en datastruktur til en annen, f.eks. fra JSON til et NSObject (objekt i Objective-C).
[6] Core Data er et rammeverk for databaselagring på iOS.

Figur 2: Denne feilmeldingen vises hvis det skjer en feil med NVDB APIet, RestKit eller Core Data.

til at brukeren ikke behøver å informeres om hva som gikk galt. Dette er både med tanke på trafikksikkerhet, og at det uansett ikke er noe brukeren kan gjøre med saken.

Detaljerte feilmeldinger logges allikevel til konsollen[7] slik at feil kan oppdages og løses under testing.

[7] Programvindu som viser tekst, ofte status- eller feilmeldinger.

2 Praktisk testing av Kjørehjelperen

Det ble foretatt tre testturer av applikasjonen. Det optimale hadde selvfølgelig vært å organisere brukertester med ulike tredjeparter. Dette lot seg dessverre ikke gjøre av ulike årsaker. Det er ikke bare å lansere en beta-versjon av en applikasjon til iOS[8]. Telefonen må kobles til utviklermaskinen, og det må installeres utviklerprogramvare på den. I tillegg må det opprettes en "provisioning"-profil som knytter telefonen til utviklerens Developer-konto på apple.com.

Siden ingen av gruppemedlemmene eier en iPhone måtte dette lånes til testen. Organiseringen rundt dette begrenset derfor antallet testturer til tre. Det er planer om å foreta flere tester før applikasjonen lanseres i App Store[9].

Alle testene ble foretatt på en iPhone[10] 4S.

2.1 Testtur 1

Den første testen ble foretatt etter sprint 1[11], søndag 24. februar. Den eneste funksjonaliteten som var implementert var visning av fartsgrense.

Testen viste at GPS'en på telefonen oppdaterte seg for sjelden. Vi kom frem til at en løsning hvor oppdateringsintervallet justeres kontinuerlig basert på farten ville være en bedre løsning.

Det viste seg at fartsgrensene som ble vist på skjermen var tilfeldige, og ofte viste fartsgrensen på kryssende og parallelle veger. På dette tidspunktet ble fartsgrensene valgt kun på koordinater, og ikke sammenliknet med veglenken[12] man befant seg på. Vi kom frem til at vi måtte sammenlikne fartsgrensens veglenke med den vi befant oss på for å unngå dette problemet.

Figur 3: Dette bildet er tatt under testtur 1.

Telefonens skjerm skrudde seg av etter noen minutter uten direkte interaksjon. Slik kunne det selvfølgelig ikke være, så dette måtte overstyres i programmet.

[8] iOS er Apples operativsystem for iPhone, iPad og iPod.
[9] App Store er Apples digitale butikk for applikasjoner og spill.
[10] Smarttelefon fra Apple Inc.
[11] En sprint er et begrep i Scrum og omfatter en kort periode i utviklingen med bestemte mål om funksjonalitet som skal implementeres.
[12] Et objekt i NVDB som representerer en sammenhengende vegstrekning med en unik ID.

Når fartsgrensen ble tresifret, altså 100, fikk ikke tallet plass på fartsgrense-skiltet. Dette var en pinlig feil som lett lot seg løse.

2.2 Testtur 2

Den andre testturen ble utført etter sprint 4, søndag 14. april. På dette tidspunktet var fartsgrense, forkjørsveg, motorveg, vilttrekk, høydebegrensning, fartsdemper og jernbanekryssing implementert. Lydvarsling, Core Data, HUD[13], avstand og landskapsmodus var også implementert.

Vi oppdaget at flere objekttyper manglet definert type i NVDB. F.eks. har vilttrekk typene "elg", "hjort" og "rein", mens fartsdempere har typer som "rumlefelt", "busshump", "fartsdump" osv. Når typen ikke var oppgitt ignorerte applikasjonen objektet. Vi kom frem til at applikasjonen heller burde bruke en standardtype hvis dette manglet. F.eks. skulle et vilttrekk uten type defineres som "elg" da dette er vanligst.

Figur 4: Under testtur 2 var HUD-visning implementert.

En annen ting vi la merke til var at applikasjonen fjernet avstandsteksten når den mottok nye skilt. Siden det er en forsinkelse mellom skiltene og tilhørende tekst opplevdes det at avstandsteksten blinket av og på. Det måtte tenkes ut en bedre løsning på dette.

Vi oppdaget også en mindre feil, nemlig at applikasjonen alltid viste motortrafikkveg-skiltet når man var på en motorveg, selv om man var på en vanlig motorveg[14]. Dette måtte undersøkes og løses.

2.3 Testtur 3

Den tredje og foreløpig siste testturen ble gjennomført etter sprint 5, onsdag 24. april. På dette tidspunktet var all funksjonalitet som er nevnt i dokumentasjonen implementert.

På denne testturen så vi at problemet med at avstandsteksten blinket av og på enda ikke var løst. Det var blitt bedre, men enda ikke helt optimalt.

Vi la også merke til at avstanden som oppgis til objekter ofte er litt unøyaktig. Vi tror dette har noe med at avstanden vi får fra MapQuest[15] er navigasjonsavstand, og ikke direkte avstand. Den kan

[13] Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.
[14] Motorveg og motortrafikkveg er de nye navnene på henholdsvis motorveg klasse A og motorveg klasse B.
[15] MapQuest er en online karttjeneste som tilbyr blant annet vegkart og navigasjonshjelp.

Figur 5: Under testtur 3 var all funksjonalitet implementert.

dermed finne på å inkludere at man må snu kjøretøyet i avstanden. Det må undersøkes om dette lar seg forbedre.

Det største problemet vi oppdaget var at applikasjonen ikke alltid klarer å forstå hvilke objekter som befinner seg foran og bak kjøretøyet på veglenken. Det er lagt mye arbeid i å finne ut nettopp dette, i tillegg til hvilken veg kjøretøyet beveger seg. Tydeligvis er det fremdeles noen problemer knyttet til dette. Løsningen blir å simulere kjøring på en veglenke på datamaskinen, og så analysere hva som skjer hele vegen. Hvis feilen ligger i applikasjonens algoritmer lar dette seg enkelt løse, men hvis det viser seg at feilen ligger i NVDB kan dette by på større utfordringer.

2013

Kjørehjelperen

Brukerveiledning

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Dette dokumentet tar for hvordan man bruker Kjørehjelperen. Det tar også for seg hvordan man endrer innstillinger, og til slutt hvilke feilmeldinger man kan støte på og hva dette betyr.

Dokumentet er beregnet på en vanlig bruker, og prøver derfor å være så lite teknisk som mulig. Ord og uttrykk er forklart i fotnoter, og finnes også i "Ordforklaringer og kilder" bakerst i rapporten.

Innholdsfortegnelse

1 Bruk av applikasjonen

For å bruke Kjørehjelperen trenger man en iPhone[1] 3GS eller nyere, oppdatert til iOS[2] 6.1. For mer info om dette, vennligst se www.apple.com. Telefonen bør være tilkoblet en strømkilde under bruk. Kjørehjelperen benytter datatrafikk. Sjekk derfor priser hos din operatør før bruk.

1.1 Første gangs bruk

Etter at Kjørehjelperen er lastet ned fra App Store[3] finner man Kjørehjelperen på startskjermen. Når applikasjonen starter første gang får man spørsmål om man vil tillate at Kjørehjelperen får tilgang til posisjonen din. Dette må tillates, ellers vil ikke applikasjonen fungere. Dette kan endres senere under "Innstillinger" -> "Personvern" -> "Lokasjon" på telefonen.

Deretter får man opp en melding som redegjør for bruken av applikasjonen. Les dette nøye, og trykk deretter "OK". Denne meldingen vil ikke vises flere ganger.

Figur 1: Du må tillate at Kjørehjelperen får bruke lokasjonen din.

Figur 2: Denne teksten vises første gang applikasjonen starter.

Kjørehjelperen er nå klar til bruk.

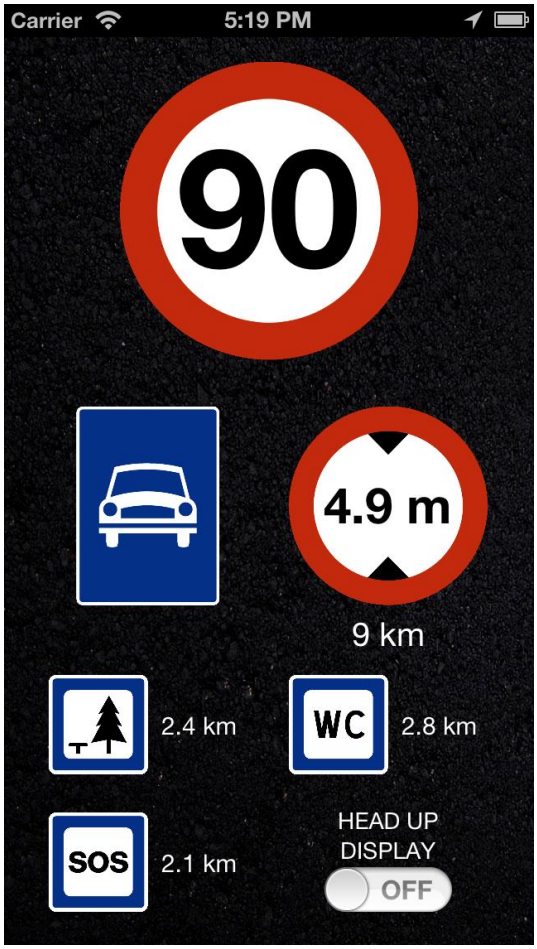
[1] Smarttelefon fra Apple Inc.
[2] iOS er Apples operativsystem for iPhone, iPad og iPod.
[3] App Store er Apples digitale butikk for applikasjoner og spill.

1.2 Kjørehjelperen i bruk

Kjørehjelperen starter å vise informasjon om vegen man befinner seg på så snart applikasjonen åpnes. Skjermbildet viser et stort skilt øverst (som oftest fartsgrensen), deretter to litt mindre skilt. Nederst vises et enda mindre skilt til venstre, og en bryter merket "HEAD UP DISPLAY" til høyre. Hvis applikasjonen kjøres på en iPhone 5 vises i tillegg to ekstra små skilt nest nederst. Hvis skiltet viser til et gitt punkt, vises også avstand til punktet under eller ved siden av skiltet. Når det dukker opp et nytt punkt på skjermen, vil telefonen spille av en liten varslingslyd.

Merk: Hvis det ikke finnes nok informasjon om vegen man er på til å fylle alle plassene, vil de nederste plassene være tomme.

Figur 3: Kjørehjelperen i bruk på en iPhone 4.



Figur 4: Kjørehjelperen i bruk på en iPhone 5.

Hvis man dreier telefonen horisontalt, vil skjermbildet automatisk følge etter. Elementene er plassert noe annerledes i landskapsmodus, men det er de samme elementene som vises på skjermen.

Figur 5: Kjørehjelperen i bruk på en iPhone 4, holdt horisontalt.

Figur 6: Kjørehjelperen i bruk på en iPhone 5, holdt horisontalt.

Dataene som vises på skjermen oppdateres kontinuerlig. Merk: Hvis enheten ikke beveger seg oppdateres dataene sjeldnere.

Head-up display er en gjennomsiktig skjerm som viser informasjon uten at man må se bort fra sitt vanlige synspunkt. Kjørehjelperen støtter dette ved å speilvende informasjonen på skjermen slik at den kan reflekteres i frontruten.

For å aktivere head-up display, trykk på "HEAD UP DISPLAY"-knappen på skjermen. Kjørehjelperen vil bytte til landskapsmodus uavhengig av telefonens orientering. Plasser så telefonen liggende på dashbordet innerst mot frontruten. For å deaktivere head-up display, trykk en gang til på "HEAD UP DISPLAY"-knappen.

Figur 7: Kjørehjelperen med head-up display aktivert.

Figur 8: Kjørehjelperen i bruk med head-up display aktivert på en iPhone 4S.

2 Endring av innstillinger

Innstillingene til Kjørehjelperen finner du sammen med andre innstillinger på telefonen, altså under "Innstillinger" -> "Kjørehjelperen".

"Lydvarsling" står på som standard. Ved å deaktivere dette vil ikke lenger Kjørehjelperen spille av en varslingslyd når nye punkter dukker opp på skjermen.

"HUD-bryter" referer til om head-up display-knappen skal vises i applikasjonen. Denne er også på som standard. Ved å deaktivere denne vil man ikke lenger kunne sette Kjørehjelperen i head-up display-modus, men man vil til gjengjeld få et ekstra skilt på skjermen.

"Telefonminne (MB)" viser til hvor mye plass på telefonen Kjørehjelperen får bruke til å mellomlagre data. Standardverdien er 100 MB. Hvis denne verdien er veldig lav vil det føre til økt datatrafikk. Endringer trer i kraft så fort Kjørehjelperen åpnes igjen.

Figur 9: Innstillingene til Kjørehjelperen ligger under "Innstillinger" på telefonen.

Figur 10: Et utdrag av innstillingene til Kjørehjelperen. "Skilt (generelt)" fortsetter nedover på skjermen, etterfulgt av "Skilt (fareskilt)".

Under "Skilt (generelt)" og "Skilt (fareskilt)" listes alle tingene Kjørehjelperen kan varsle om. Alle verdiene står også på som standard. Ved å deaktivere enkelte verdier vil man ikke lenger bli varslet om dette. Man kan dermed skreddersy applikasjonen til å kun vise den informasjonen man er interessert i.

3 Feilmeldinger

Det er to ulike feilmeldinger man kan møte på når man bruker Kjørehjelperen. Den første er en hvit tekst som vises på skjermen som sier: "Fant ingen data om din lokasjon. Prøver igjen straks!". Dette kan skyldes at Kjørehjelperen ikke klarer å hente data for posisjonen man befinner seg på, eller at det rett og slett ikke finnes data for denne posisjonen. Applikasjonen vil prøve å hente data på nytt ganske umiddelbart. Vedvarer denne feilen kan det hende at applikasjonen ikke får tilgang til internett. Da kan man sjekke om telefonen er koblet til nettverket eller om den eventuelt står i flymodus.

Figur 11: Denne feilmeldingen betyr at Kjørehjelperen ikke klarer å hente noen data.

Figur 12: Denne feilmeldingen betyr at Kjørehjelperen ikke klarer å finne enhetens posisjon.

Den andre feilmeldingen man kan få er en dialogboks som sier: "Klarte ikke finne enhetens posisjon. Har Kjørehjelperen tillatelse til å bruke posisjonstjenester?". Hvis dette skjer plutselig under kjøring kan det tyde på at GPSen[4] ikke klarer å finne enhetens posisjon. Dette kan skje hvis man f.eks. kjører inn i en lang tunnel. Da er det bare å trykke på "OK" og vente til GPS-signalet kommer tilbake. Hvis denne meldingen kommer opp når man starter Kjørehjelperen kan det tyde på at applikasjonen ikke får tilgang til telefonens GPS. Da bør man sjekke om GPSen er deaktivert eller om Kjørehjelperen ikke har tillatelse til å bruke lokasjonstjenesten under "Innstillinger" -> "Personvern" -> "Lokasjon".

[4] NAVSTAR Global Positioning System (GPS) er et nettverk av satellitter som er plassert i bane rundt Jorden av det amerikanske forsvaret. Systemet gjør det mulig for en mottaker å fastsette egen posisjon med svært stor nøyaktighet overalt i verden, under nær sagt alle værforhold.

2013

Kjørehjelperen

Kravspesifikasjon

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
22.03.2013

Forord

Kravspesifikasjonen skal fungere som et styringsdokument for hovedprosjektet. Den skal definere rammer og retningslinjer for prosjektet, utarbeides i samarbeid med oppdragsgiver, og sikre at både vi og oppdragsgiver får klarlagt hva betingelsene for prosjektet skal være. Dette gjelder både funksjonelle og ikke-funksjonelle krav, samt krav til arbeidsprosess og kvalitetssikring.

Kravspesifikasjonen skal opptre som en bindende avtale mellom oss og oppdragsgiver, men det skal også tas hensyn til krav fra Høgskolen i Oslo og Akershus. Dette dokumentet, og alle revisjoner, skal derfor være godkjent av alle parter før det kan anses som gyldig.

Leserveiledning

Kravspesifikasjonen kommer først med en kort presentasjon av de ulike partene, bakgrunnen for prosjektet og en generell beskrivelse av oppgaven. Deretter følger funksjonelle krav, etterfulgt av ikke-funksjonelle krav. Til slutt tar vi for oss håndteringen av endringer og feil som måtte oppstå underveis i prosjektet.

Tekniske ord og forkortelser er forklart i fotnoter. Alle tekniske ord er dessuten samlet i dokumentet "Ordforklaringer og kilder" bakerst i rapporten.

Innholdsfortegnelse

1 Presentasjon

Dette kapittelet inneholder en kort presentasjon av de forskjellige partene i prosjektet.

1.1 Gruppen

Gruppen består av Henrik Hermansen og Lars Smeby. Vi tar begge Bachelor i Informasjonsteknologi ved Høgskolen i Oslo og Akershus, og har jobbet sammen ved flere tidligere prosjekter. På bakgrunn av tidligere erfaringer er vi trygge på hvilke mål og forventninger vi har til hverandre og oppgaven.

1.2 Oppdragsgiveren

Oppgaven utføres for BEKK Consulting AS, som er et norsk konsulentselskap. De gjennomfører prosjekter for private og offentlige virksomheter innen strategisk rådgivning, utvikling av IT-systemer og design av digitale tjenester. De er i dag omkring 300 ansatte, og har kontorer i Oslo og Trondheim.

Ansvarlig for oppgaven hos BEKK er Christian Schwarz, mens vår faglige veileder er Christoffer Marcussen. Christoffer er med i BEKKs faggruppe for applikasjonsutvikling på mobiltelefon og vil derfor kunne bistå oss i arbeidet.

1.3 Kunden

Oppgaven er som sagt gitt oss av BEKK, men det er Statens vegvesen som i utgangspunktet har ønsket denne applikasjonen. Dette innebærer at det er BEKK vi forholder oss til, men at Statens vegvesen stiller sin kompetanse til rådighet.

Vår kontaktperson hos Statens vegvesen er Jan Kristian Jensen.

1.4 Veileder og institusjon

Hovedprosjektet utføres ved Institusjon for informasjonsteknologi ved Høgskolen i Oslo og Akershus. Vår interne veileder ved høgskolen er Eva Hadler Vihovde.

2 Bakgrunn for oppgaven

BEKK har i 2012 utviklet et REST[1] API[2] for Statens vegvesen. Tanken er at dette APIet skal gjøre data fra NVDB[3] tilgjengelig for alle. Vårt oppdrag er å utforske, teste og demonstrere dette APIet slik at andre inspireres til å ta dette i bruk.

[1] Representational State Transfer, REST, er en programvarearkitektur for distribuerte systemer som World Wide Web. REST har etter hvert blitt den dominerende designmodellen for web-APIer.

3 Overordnet systembeskrivelse

Det skal utvikles en applikasjon til iPhone[4] som skal gi brukeren nyttige data fra Nasjonal Vegdatabank når han/hun er ute og kjører i hverdagen. Applikasjonen skal ved hjelp av brukerens geografiske posisjon innhente og presentere informasjon som er relevant for det området eller den strekningen som brukeren kjører på, basert på brukerens preferanser.

Informasjonen skal presenteres på en oversiktlig måte, som i så liten grad som mulig tar sjåførens oppmerksomhet vekk fra trafikken. Dette er viktig med tanke på trafikksikkerhet og norsk vegtrafikklov. Tanken er at sjåføren, ved hjelp av en fastmontert telefonholder eller refleksjonen av telefonen i frontruten, enkelt skal kunne få den informasjonen han/hun trenger ved et kort blikk.

APIet for Nasjonal Vegdatabank inneholder veldig mye data, så vår oppgave blir derfor å velge informasjon som demonstrerer APIets muligheter, samtidig som det skal være en applikasjon som er interessant for folk å bruke i hverdagen.

3.1 Mål

- Demonstrere nytteverdien i et åpent API med vegdata fra NVDB
- Utvikle en applikasjon som når en så stor målgruppe som mulig
- Evaluere funksjonalitet og brukervennlighet hos Statens vegvesen sitt nye REST API
- Vurdere iOS[5] som plattform sammenlignet med tilsvarende plattformer

3.2 Rammebetingelser

- Applikasjonen skal i hovedsak benytte data fra NVDB
- Applikasjonen skal utvikles for iOS
- Utviklingen må skje på OS X[6]

4 Funksjonelle krav

Funksjonelle krav er krav til funksjonalitet som skal implementeres i systemet. De funksjonelle kravene er delt opp i tre deler, fordelt etter prioritet. Prioriteringen er først basert på hvor viktig vi anser en funksjon for å være, og deretter hvor sannsynlig vi anser det at vi får tid til å implementere den, gitt tidsrammene for prosjektet. Den første delen beskriver funksjonaliteten vi anser som

[2] Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.
[3] Nasjonal vegdatabank, database med vegobjekter forvaltet av Statens Vegvesen.
[4] Smarttelefon fra Apple Inc.
[5] iOS er Apples operativsystem for iPhone, iPad og iPod.
[6] OS X er det gjeldende operativsystemet til Mac.

essensiell for å nå målene, og den andre delen beskriver det vi håper å få til. Den tredje delen beskriver forslag til tilleggsfunksjonalitet dersom vi blir tidligere ferdig enn ventet med implementasjonen av den øvrige funksjonaliteten.

4.1 Prioritert funksjonalitet

- Brukeren skal kunne se:
  - Gjeldene fartsgrense for vegstrekningen
  - Hvorvidt han er på forkjørsveg eller ikke
  - Diverse gjeldende varselskilt (som elgfare og farlig vegkryss)
- Brukeren skal kunne endre:
  - Hvilken informasjon som skal vises
  - Hvor tidlig varsler skal vises
  - Hvorvidt det brukes lyd for å varsle brukeren

4.2 Ønsket funksjonalitet

- Informasjonen på skjermen skal kunne speilvendes med det formål å lage et HUD[7] i frontruten
- Brukeren skal kunne se:
  - Avstand til førstkommende rasteplass
  - Avstand til førstkommende SOS-lomme

4.3 Eventuell tilleggsfunksjonalitet

- Brukeren skal kunne se avstand til nærmeste toalett

5 Ikke-funksjonelle krav

Ikke-funksjonelle krav er krav til systemets egenskaper og rammeverk. Disse er delt opp i produktkrav, prosesskrav og eksterne krav.

5.1 Produktkrav

Produktkrav omhandler krav til det ferdige produktet som ikke er en direkte funksjon. Dette omhandler krav til brukervennlighet, effektivitet og pålitelighet.

[7] Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.

5.1.1 Brukervennlighet

- Applikasjonen skal være på norsk
- Applikasjonen skal i så stor grad som mulig følge de 5 E'ene[8]
- Applikasjonen skal ikke kreve interaksjon under bruk (kjøring)

5.1.2 Effektivitetskrav

- Applikasjonen skal begrense egen datatrafikk i så stor grad som mulig
- Applikasjonen skal være så energigjerrig som mulig
- Applikasjonen skal begrense bruk av telefonens minne

5.1.3 Pålitelighetskrav

- Posisjonering av bruker skal være innenfor en nøyaktighet på 10 meter
- Posisjonering av objektdata skal være innenfor en nøyaktighet på 1 meter
- Dataene som presenteres skal være oppdaterte

5.2 Prosesskrav

Prosesskrav omhandler krav til prosessen rundt utviklingen av produktet, deriblant metodikk, leveranser, implementasjon og standarder.

5.2.1 Utviklingsmetodikk

- Det skal benyttes Scrum[9] som utviklingsmodell

5.2.2 Leveransekrav

Her følger krav til leveransene i prosjektet.

5.2.2.1 Underleveranser

- Det skal leveres en kjørbar applikasjon ved hver leveranse (22.02, 08.03, 22.03, 05.04 og 19.04)

5.2.2.2 Endelig leveranse

- Den ferdige applikasjonen skal være godkjent og tilgjengelig i Apples App Store[10]
- Applikasjonen skal være kjørbar på iPhone 3GS, 4, 4S og 5 (oppdatert til iOS 6.1)
- Kildekoden for leveransen skal lisensieres under GNU GPL[11] og være tilgjengelig via GitHub[12]

[8] Prinsipper for brukervennlighet: effective, efficient, engaging, error tolerant og easy to learn (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).
[9] Scrum er en iterativ og inkrementell systemutviklingsmodell. Les mer i prosessdokumentasjonen, kapittel 1.4.3.
[10] App Store er Apples digitale butikk for applikasjoner og spill.
[11] GNU General Public License (GNU GPL eller GPL) er en lisens som tillater fri bruk, kopiering og endring av koden.
[12] GitHub er en hosting-tjeneste for systemutviklingsprosjekter som benytter Git.

5.2.3 Implementasjonskrav

- Applikasjonen skal utvikles for iOS 6.1
- Applikasjonen skal utvikles med Xcode[13] 4.6 på OS X
- Applikasjonen skal testes på så mange fysiske iOS-enheter som mulig
- Objective-C[14] benyttes som programmeringsspråk
- Applikasjonen skal benytte Cocoa Touch[15] og de andre rammeverkene i iOS 6.1 SDK[16]
- Applikasjonen skal benytte det åpne biblioteket RestKit[17] for kommunikasjon og parsing[18] av data med NVDB APIet
- Brukerpreferanser skal lagres som "User preferences"[19]
- Vegdata skal lagres som Core Data[20]

5.2.4 Krav til standarder

- Det skal benyttes JSON[21] for kommunikasjon med NVDB API
- Applikasjonen skal benytte MVC-patternet (som forventes i iOS)[22]

5.3 Eksterne krav

Eksterne krav omhandler krav som ikke passer inn under produkt- eller prosesskrav, deriblant estetiske og lovmessige krav.

5.3.1 Estetiske krav

- Applikasjonen skal være i samsvar med Apples retningslinjer[23] for design på iOS
- Dataene som presenteres skal være store nok til å kunne sees på 1 meters avstand
- Det skal i så stor grad som mulig benyttes ikoner og grafikk fremfor tekst
- Designet skal følge de 7 designprinsippene[24]

5.3.2 Lovmessige krav

- Applikasjonen skal ikke kreve interaksjon under kjøring som strider mot norsk vegtrafikklov

[13] Xcode er et program til Mac for utvikling, testing og publisering av programmer til iOS og OS X.
[14] Objective-C er programmeringsspråket som brukes av Apple. Det er basert på C.
[15] Cocoa Touch er et brukergrensesnitt-rammeverk som brukes til utvikling til iOS.
[16] Software development kit (SDK) er verktøy som lar deg utvikle programmer til en spesifikk programvarepakke, et rammeverk, en hardware-plattform, et operativsystem e.l.
[17] RestKit er et Objective-C-rammeverk for iOS som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen av objekter.
[18] Prosessen med å generere eller lese data fra f.eks. en JSON- eller XML-stuktur.
[19] User preferences er en lagringsmetode på iOS.
[20] Core Data er et rammeverk for databaselagring på iOS.
[21] JavaScript Object Notation, en enkel tekstbasert standard for datautveksling.
[22] Model View Controller (MVC) er en måte å dele opp og strukturere koden på i en applikasjon.
[23] https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html
[24] Structure, simplicity, consistency, tolerance, visibility, affordance og feedback (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).

- Applikasjonen skal inneholde teksten: "Inneholder data under norsk lisens for offentlige data (NLOD) tilgjengeliggjort av Statens vegvesen."
- Applikasjonen skal opplyse om, og avskrive seg og Statens vegvesen ansvar for, eventuelle feil og mangler som måtte forekomme i NVDB APIet

6 Endringshåndtering

Dette kapittelet går ut på hvem som kan komme med forslag til endringer i prosjektet, og hvilke betingelser som settes for at disse skal kunne tas til følge.

6.1 Interessentene

I utgangspunktet er det Statens vegvesen og BEKK som er interessentene og stakeholders i dette prosjektet. Det er disse som vil kunne fremme forslag eller krav om endringer til produktet som lages. Samtidig er også prosjektgruppen en interessent som kan komme med forslag til endringer hvis dette sees fordelaktig.

6.2 Krav ved forslag til endringer

- Endringer i prioritert funksjonalitet skal estimeres av prosjektgruppen og drøftes sammen med alle interessenter før det tas en avgjørelse. Denne typen endringer må godkjennes skriftlig av alle parter.
- Endringer i ønsket funksjonalitet, eventuell tilleggsfunksjonalitet og ikke-funksjonelle krav skal vurderes av prosjektgruppen underveis. Dersom forslaget kommer fra prosjektgruppen skal dette godkjennes av øvrige interessenter.
- Alle interessenter må godkjenne og være enige om eventuelle endringer før de kan inkluderes i kravspesifikasjonen.

6.3 Feilhåndtering

Ved feil i systemet skal det ikke startes på ny funksjonalitet før feilen er rettet. Hvis det arbeides med funksjonalitet av lavere prioritet enn der feilen er oppdaget, skal arbeidet avbrytes inntil feilen er rettet. Unntak fra dette må godkjennes av alle interessenter.

2013

Kjørehjelperen

Ordforklaringer og kilder

Høgskolen i Oslo og Akershus

Henrik Hermansen og Lars Smeby
Gruppe 8
26.05.2013

Forord

Dette dokumentet er kun ment som et oppslagsverk. Alle ord og uttrykk forklart i fotnoter i de andre dokumentene er samlet alfabetisk i dette dokumentet. I tillegg er kildene benyttet i dokumentasjonen samlet her.

Innholdsfortegnelse

1. Ordforklaringer

Android	Android er et operativsystem til mobile enheter basert på Java. Android utvikles av Google.
API	Application Programming Interface (API) er et grensesnitt for kommunikasjon mellom programvare. APIet beskriver de metoder som en gitt programvare eller et bibliotek kan kommunisere med.
App Store	App Store er Apples digitale butikk for applikasjoner og spill.
ARC	Automatic Reference Counting (ARC) er at ansvaret for minnehåndtering flyttes fra programmereren til kompilatoren.
Array	Et array er en endimensjonal tabell.
Catching	Mellomlagring av data.
Cocoa Touch	Cocoa Touch er et brukergrensesnitt-rammeverk som brukes til utvikling til iOS.
Controller	"C" i MVC. Bindeleddet mellom modellen og viewet.
Core Data	Core Data er et rammeverk for databaselagring på iOS.
De fem E'ene	Prinsipper for brukervennlighet: effective, efficient, engaging, error tolerant og easy to learn (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).
De syv designprinsippene	Structure, simplicity, consistency, tolerance, visibility, affordance og feedback (Stone, Jarret, Woodroffe, Minocha, «User Interface Design and Evaluation», 2005, Elsevier Inc.).
Delegat	En delegat i Objective-C er en klasse som implementerer en gitt protokoll. En annen klasse kan kalle denne delegaten uten å vite noe om hvilken klasse den kaller.
Designpattern	Et designpattern er en kjent gjenbrukbar løsning på et vanlig problem i systemutvikling.
Dialogboks	Liten boks med informasjon som legger seg over skjermbildet. Man må trykke på ”OK” for at den skal forsvinne.
Dictionary	En dictionary er en datastruktur bestående av nøkkel- og verdipar.
Entitet	En entitet kan defineres som noe som er i stand til selvstendig eksistens og som kan identifiseres.
Event	En hendelse i et program som klasser og metoder kan plukke opp.
Flash	Adobe Flash er programvare som utvikles og distribueres av Adobe Systems. Flash brukes mest til å vise dynamisk innhold på nettsider og støtter vektor- og pikselgrafikk.

Funksjonspeker	En funksjonspeker er en peker som peker til kjørbar kode i minnet.
Git	Git er et verktøy for versjonskontroll i systemutviklingsprosjekter.
GitHub	GitHub er en hosting-tjeneste for systemutviklingsprosjekter som benytter Git.
GPL	GNU General Public License (GNU GPL eller GPL) er en lisens som tillater fri bruk, kopiering og endring av koden.
GPS	NAVSTAR Global Positioning System (GPS) er et nettverk av satellitter som er plassert i bane rundt Jorden av det amerikanske forsvaret. Systemet gjør det mulig for en mottaker å fastsette egen posisjon med svært stor nøyaktighet overalt i verden, under nær sagt alle værforhold.
GUI	Graphical user interface.
Header	Supplerende data plassert først i en datablokk som blir lagret eller sendt.
Headerfil	En headerfil er en fil som forteller hvilke egenskaper og metoder en klasse har, men ikke hvordan de er implementert. Headerfiler benyttes bl.a. i C, C++ og Objective-C.
HTML	HyperText Markup Language (HTML) er et markeringsspråk for formatering av nettsider.
HTTP	Hypertext Transfer Protocol.
HUD	Head-up display eller heads-up display, også kjent som HUD, er en gjennomsiktig skjerm som viser informasjon uten at brukerne må se bort fra sine vanlige synspunkter. Opprinnelsen til navnet stammer fra at en pilot skal kunne se informasjon med hodet "opp" og se frem, i stedet for å se ned på instrumentene i cockpiten.
Høynivåspråk	Et høynivåspråk har stor abstraksjon fra måten datamaskinen fungerer.
IDE	Integrated Development Environment er programvare som tilbyr omfattende verktøy for programmering og programvareutvikling.
IM	Instant Messaging, eller lynmelding på norsk, er en type samtale over et nettverk der man sender tekstbeskjeder til hverandre i sanntid.
Interface	I Objective-C er et interface deklarasjonen av en klasse med dens egenskaper og metoder. I Java er et interface et sett med egenskaper en klasse må implementere hvis interfacet benyttes, tilsvarende en protokoll i Objective-C.
iOS	iOS er Apples operativsystem for iPhone, iPad og iPod.
iPhone	Smarttelefon fra Apple Inc.
Java	Java er et objektorientert programmeringsspråk utviklet av Sun

	Microsystems (nå kjøpt av Oracle Corporation).
JavaScript	JavaScript er et skriptspråk som er best kjent for å tilføre dynamiske elementer til nettsider.
JSON	JavaScript Object Notation, en enkel tekstbasert standard for datautveksling.
Kompilator	En kompilator er et dataprogram som oversetter et dataprogram fra kildekode til maskinkode.
Kontroller	Se Controller.
Mapping	Mapping er prosessen med å overføre data fra en datastruktur til en annen, f.eks. fra JSON til et NSObject (objekt i Objective-C).
MapQuest	MapQuest er en online karttjeneste som tilbyr blant annet vegkart og navigasjonshjelp.
Metreringsretning	Metreringsretningen er den stigende retningen på egn veglenke, altså når posisjonstallet går fra 0 mot 1.
Minnelekkasje	Feil ved minnehåndteringen i programmet.
Modell	"M" i MVC. Selve datastrukturen, i tillegg til regler, logikk og funksjoner.
MVC	Model View Controller (MVC) er en måte å dele opp og strukturere koden på i en applikasjon.
NVDB	Nasjonal vegdatabank, database med vegobjekter forvaltet av Statens vegvesen.
Objective-C	Objective-C er programmeringsspråket som brukes av Apple. Det er basert på C.
OpenStreetMap	OpenStreetMap (OSM) er et fritt redigerbart kart over hele jorden, laget og redigert av brukerne. OpenStreetMap er lisensiert under Open Data Commons Open Database License (ODbL), se http://opendatacommons.org/licenses/odbl/1.0/ for mer info.
OS X	OS X er det gjeldende operativsystemet til Mac.
Parse	Prosessen med å generere eller lese data fra f.eks. en JSON- eller XML-stuktur.
Pattern	Se designpattern.
Peker	En peker er en referanse til et sted i minnet hvor objektet ligger.
Protokoll	En protokoll i Objective-C kan minne om et interface i Java. Det er en kontrakt som spesifiserer hvilke metoder en klasse må implementere.
Repository	Repository er et begrep fra versjonskontroll og omhandler en datastruktur som bl.a. inneholder filer, mapper, historikk og referanser

	til tidligere versjoner.
REST	Representational State Transfer, REST, er en programvarearkitektur for distribuerte systemer som World Wide Web. REST har etter hvert blitt den dominerende designmodellen for web-APIer.
RestKit	RestKit er et Objective-C-rammeverk for iOS som forenkler kommunikasjonen med REST-baserte webtjenester og mappingen av objekter.
Scrum	Scrum er en iterativ og inkrementell systemutviklingsmodell.
SDK	Software development kit (SDK) er verktøy som lar deg utvikle programmer til en spesifikk programvarepakke, et rammeverk, en hardware-plattform, et operativsystem e.l.
Sett	Et sett (NSSet i Objective-C) er en datastruktur som holder på mange elementer. Elementene er ikke knyttet til en fast plass slik som i et array.
Skype	Programvare for IP-telefoni.
Smalltalk	Smalltalk er et objektorientert programmeringsspråk. Det spesielle med Smalltalk er at sjekking av typer og objekter skjer under kjøring i motsetning til under kompilering.
SQL	Structured Query Language (SQL) er et programmeringsspråk brukt til å kommunisere med relasjonsdatabaser.
SQLite	SQLite er et relasjonsdatabasesystem i et lite C-bibliotek. I motsetning til andre databasesystemer er SQLite integrert i klientapplikasjonen og ikke en separat prosess.
Statisk	Statiske metoder er metoder som ikke er knyttet til en spesifikk instans av en klasse.
Storyboard	Et storyboard er et dokument som beskriver layoutet til et skjermbilde og overgangene mellom dem i iOS.
Subklasse	En subklasse er en klasse som arver alle egenskapene til en superklasse, eller en foreldreklasse.
Tråd	En tråd er en sekvens av programinstrukser som kan kjøres uavhengig av andre instrukser i et program (en lettvekts-prosess).
Tutorial	Stegvis gjennomgang på hvordan man lager eller utfører noe.
URL	Uniform Resource Locator.
User preferences	User preferences er en lagringsmetode på iOS.
Veglenke	Et objekt i NVDB som representerer en sammenhengende vegstrekning med en unik ID.
Vegreferanse	Et objekt i NVDB som inneholder informasjon om en veg, blant annet en

	veglenke.
View	"V" i MVC. Det brukeren ser, altså brukergrensesnittet.
ViewController	Klassetype i iOS. En kontroller som fokuserer mer på viewet enn på modellen.
Windows Phone	Windows Phone er Microsofts operativsystem til smarttelefoner.
Xcode	Xcode er et program til Mac for utvikling, testing og publisering av programmer til iOS og OS X.
XML	Extensible Markup Language (XML) er et universelt og utvidbart markeringsspråk. Brukes til deling av strukturerte data mellom informasjonssystemer og til koding av dokumenter og som kommunikasjonsmiddel mellom ulike informasjonssystemer og dataformater.

2. Kilder

Tema	Kilde
Adobe Flash	http://no.wikipedia.org/wiki/Flash
ARC	http://en.wikipedia.org/wiki/Automatic_Reference_Counting
Cocoa Touch	http://en.wikipedia.org/wiki/Cocoa_Touch
Core Data	http://en.wikipedia.org/wiki/Core_Data
De fem E'ene	Stone, Jarret, Woodroffe, Minocha, "User Interface Design and Evaluation", Elsevier Inc., 2005.
De syv designprinsippene	Stone, Jarret, Woodroffe, Minocha, "User Interface Design and Evaluation", Elsevier Inc., 2005.
GPL	http://en.wikipedia.org/wiki/GNU_GPL
GPS	http://no.wikipedia.org/wiki/Gps
Head-up display	http://en.wikipedia.org/wiki/Head-up_display
Interface (Java)	http://en.wikipedia.org/wiki/Interface_(Java)
JavaScript	http://no.wikipedia.org/wiki/Javascript
JSON	http://en.wikipedia.org/wiki/Json
JSON	http://no.wikipedia.org/wiki/JSON
Kompilatorer	http://no.wikipedia.org/wiki/Kompilator
MapQuest	http://en.wikipedia.org/wiki/MapQuest
MVC	http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
MVC i iOS	http://blog.teamtreehouse.com/ios-design-patterns-model-view-controller-part-3 http://developer.apple.com/library/ios/#documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html http://kentnguyen.com/ios/ios-beginner-series-understand-mvc/
NVDB	http://data.norge.no/data/nasjonal-vegdatabank-api
Object Graph	http://en.wikipedia.org/wiki/Object_graph
Objective-C	https://en.wikipedia.org/wiki/Objective-C
Persistence framework	http://en.wikipedia.org/wiki/Persistence_framework
REST	http://en.wikipedia.org/wiki/Representational_state_transfer

RestKit	http://restkit.org/
Scrum	http://no.wikipedia.org/wiki/Scrum
SDK	http://en.wikipedia.org/wiki/Software_development_kit
Smalltalk	https://en.wikipedia.org/wiki/Smalltalk
Smidig utvikling	http://en.wikipedia.org/wiki/Agile_software_development
SQLite	http://en.wikipedia.org/wiki/SQLite
UML (Unified Modeling Language)	Craig Larman, "Applying UML and Patterns", Prentice Hall, 3rd ed., 2005.
XML	http://no.wikipedia.org/wiki/XML