

Skumring

IN2000 - Prosjektoppgave i

Software Engineering



Team 3

Adrian Emil Chambe -Eng: adrianch

Anna Hansgård: annhansg

Mari Svennevik Notland: marisnot

Synnøve Nordvik Helgesen: synnovnh

Viljar Drevland Hardersen: viljardh

Veiledere: Jamila Mehmandarova & Camilla Thogersen Kielland Lund

Presentasjon	3
Introduksjon	3
Bakgrunn.....	3
Vår løsning	3
Beskrivelse av teamet	3
Brukerdokumentasjon.....	4
Målgruppe	4
Funksjonalitet.....	4
Plattform	6
Aksessering	7
Produktdokumentasjon	7
Kvalitetssegenskaper	7
Evalueringsfunksjonalitet	7
Arkitektur	7
Best practice	8
Beregning av værforhold.....	8
Håndtering av bilder.....	8
Brukskvalitet	8
Responstid.	8
WCAG og Universell utforming	9
Mulig å oppfatte	9
Mulig å betjene	9
Forståelig	9
Robusthet	9
Pålitelighet/Fravær av feil	9
Mangler i systemet:	10
API og EDR	10
Prosessdokumentasjon.....	11
Valg av smidige metoder - Scrumban.....	11
Designprosessen.....	12
1. Empati – Intervju.....	13

2. Definere – Affinity diagram.....	13
3. Utvikle idéer – crazy 8 m/ brainstorming	14
4. Prototyping.....	14
5. Teste –Geriljatesting og brukertesting	14
Geriljatesting.	14
Brukertesting.	15
Implementeringsprosess.....	16
Første fase: fra Kickoff til MVP	16
Refleksjon: Første fase	17
Andre fase: Hovedfasen.....	18
Kode - utforsking og tilspisning.....	19
Refleksjon: Andre fase	19
Avslutningsfase.....	20
Kravspesifikasjon - endringer og beslutninger	21
Refleksjon: kravspesifikasjonen.....	22
Testing.....	22
Refleksjon.....	23
Utviklingsprosessen	23
Refleksjon: Teamarbeidet	24
Konflikt og håndtering	25
Aktiviteterne	26
Avslutning	26
Veien videre.....	26
Vår læring	27
Tips til neste års studenter	27
Kilder	28

Presentasjon

Introduksjon

Har du noen gang sett en så fantastisk solnedgang du skulle ønske du kunne oppleve den på nytt? Kanskje du har gått forbi et sted du vet ville være det perfekte bakteppet for et solnedgangsbilde, men du var for sent ute til å fange øyeblikket? Velkommen til løsningen som lar deg forutsi og fange de mest uforglemmelige solnedgangene. Løsningen vår er designet for å tilfredsstille et ønske om å oppleve naturens mest spektakulære øyeblikk.

Bakgrunn

Inspirert av veien hjem en kald ettermiddag i januar fikk et gruppemedlem idéen til vår app da en nydelig solnedgang lå i horisonten over Oslogryta. Med det kommende kurset IN2000 i bakhodet ga det en idé til å lage en app som forutser når disse omstendighetene ligger til rette. Etter pitching av forslaget til de andre gruppemedlemmene på Kick-off, viste barometeravstemningen at alle ønsket å gå videre med en åpen case for å følge denne idéen.

Vår løsning

Skumring henter vær- og soldata via MET sine APIer, sjekker værforholdene rundt solnedgangstider de tre neste dagene og presenterer de på en intuitiv måte for brukeren. Appen viser tidspunkt for blå og gyllen time og beregner værforhold for fotografering basert på luftfuktighet og tettheten av skyer i tre forskjellige høyder. Appen har allerede en håndfull lokasjoner lagret, og brukerne kan legge til egendefinerte posisjoner og se avstanden til disse fra nåværende posisjon. Brukerne kan se disse som pins på et kart og i listeform, og kan lagre lokasjoner i en favorittliste.

Beskrivelse av teamet

Vi er fem ambisiøse, hardtarbeidende, blide og sosiale personer. Anna, Viljar og Mari studerer informatikk: design, bruk og interaksjon, mens Synnøve og Adrian studerer informatikk: digital økonomi og ledelse. I løpet av prosjektet har vi blitt en sammensveisa gjeng som trives med å jobbe sammen.



Brukerdokumentasjon

Målgruppe

Dette produktet er rettet mot hobbyfotografer uansett utstyr og ferdighetsnivå. I tillegg ønsker vi at applikasjonen skal fungere like godt for turister som byens innbyggere.

Funksjonalitet

Det første brukeren ser når en åpner appen er en SplashScreen, fulgt av en kjapp guide til appens funksjonaliteter som vil vises hver gang appen starter til bruker har gått gjennom hele guiden. Deretter vises hjemskjermen (Figur 1). Her man får info om dagens solnedgang ved brukers lokasjon, angitt på toppen av skjermen. Her vises temperatur, klokkeslett, og værforholdene ved solnedgang. I tillegg vises tidspunktene til den gylne -og blå time. Når bruker har favorittmerket en lokasjon vil den dukke opp på hovedskjermen i form av et kort med bilde og beskrivelse av værforhold. Trykker brukeren på kortet åpnes en informasjonsskjerm for lokasjonen (Figur 6/7), og brukeren blir presentert for en beskrivelse av stedet, i tillegg til data for solnedgangsforhold de tre neste dagene. Her kan en se avstand til lokasjonen fra sin nåværende posisjon.




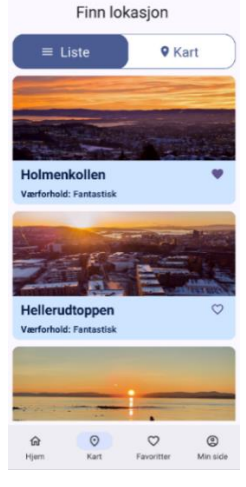
På bunnen av skjermen er en navigasjonsbar som lar bruker navigere til de forskjellige hovedskjermene. På “Kart/Map” vil det åpnes et kart ved brukerens lokasjon (Figur 2), og noen forhåndslagrede lokasjoner visualiseres som pins på kartet, i tillegg til de brukeren har lagret. Pins kan trykkes på og et infokort vil dukke opp (Figur 3), der man kan lagre/fjerne stedet som favoritt og navigere til informasjonsskjermen. Trykker og holder bruker på kartet kommer en dialog opp der bruker kan lagre stedet som lokasjon med tilhørende bilde om ønskelig. Ved trykk på knappen over kartet vises lagrede lokasjoner i et listeforamt med navn, bilde og værforhold

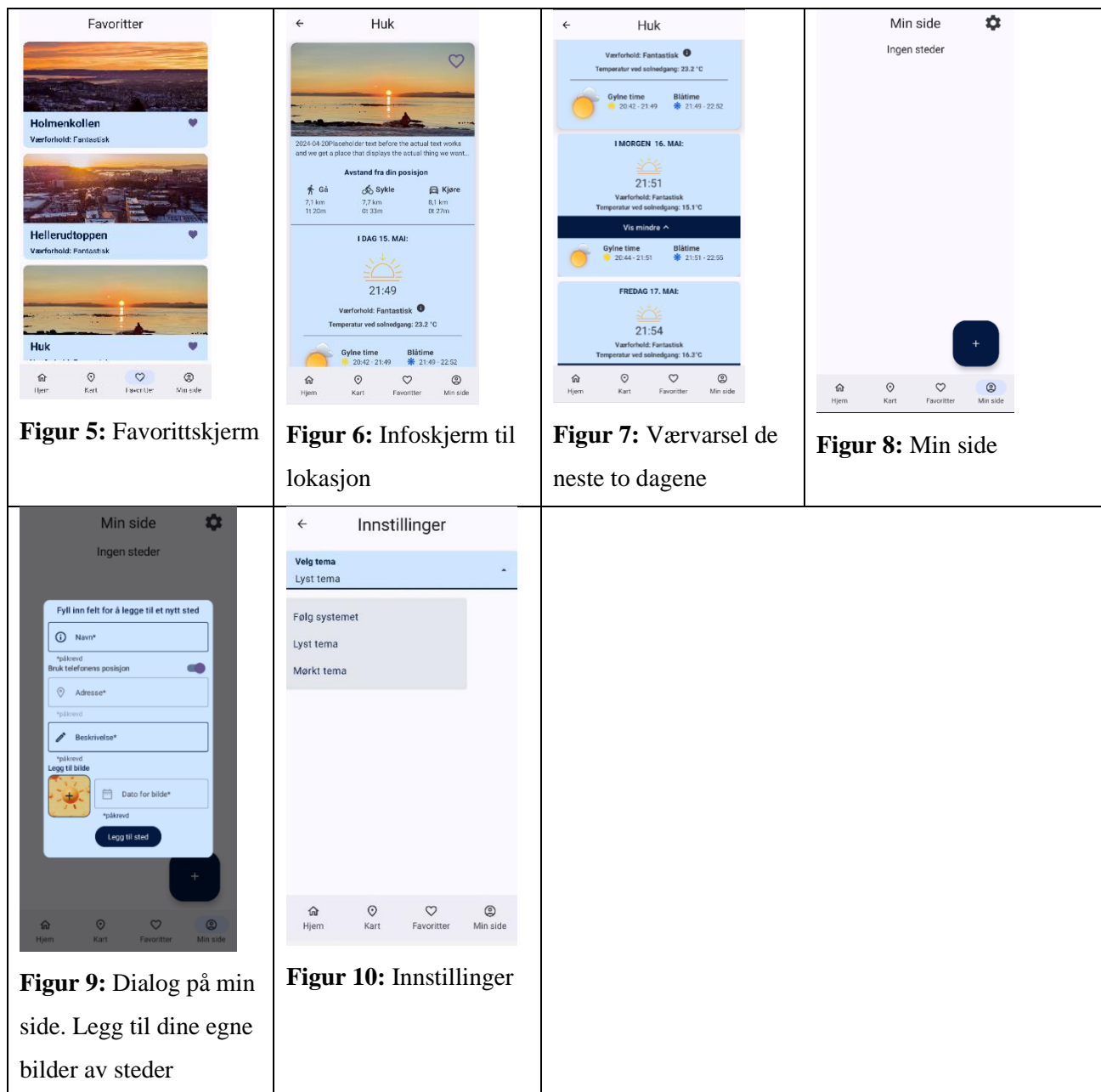
den kvelden. Kortene i listen kan lagres/fjernes som favoritt, og ved trykk kan en navigere til lokasjonens infoskjerm (Figur 6/7).

Fra navigasjonsbaren kan man navigere til den dedikerte skjermen for favorittlokasjoner (Figur 5), vist i et listeformat. På “Min Side/My Page” kan man legge til egne lokasjoner og se disse i en liste. Når bruker trykker på plussknappen nederst i høyre hjørne dukker en dialog opp (Figur 9), der man må fylle inn navn på sted, beskrivelse, adresse (eller automatisk via brukerens posisjon), dato for bildet, og til slutt kan legge inn selve bildet. Hvis brukeren ikke har fylt inn alle de påkrevde feltene vil ikke lokasjonen bli lagret. Øverst til høyre er det et tannhjul hvor man kan endre innstillinger for appen (Figur 8/10) som å endre appens tema. Denne endringer vil ikke skje før appen restarter. Det skulle også være en funksjon for å bytte språk og standardlokasjon, men dette prioriterte vi vekk grunnet tidsbegrensning. Språk følger automatisk systeminnstillinger, mellom norsk og engelsk.

Tabell 1

Skjermbilder fra appen i lyst tema

 <p>Figur 1: Hjemskjerm</p>	 <p>Figur 2: Kartskjerm</p>	 <p>Figur 3: Kort med beskrivelse av pin</p>	 <p>Figur 4: Listefremvising av lagrede lokasjoner</p>
---	---	---	--



Plattform

Produktet er utviklet til Android uten planer om å lage støtte for iOS, nettleser eller annet. Dette er for enkelhetens skyld - alle på gruppen er helt ferske utviklere, og vi ønsker å kunne fokusere på å få selve apputviklingsprosessen godt innarbeidet og sørge for at vi kommer i mål med et stabilt og fungerende produkt. I tillegg legger kurset opp til utvikling for Android.

Aksessering

Appen kan hentes under “releases” i github-repoet. All kode og ressurser er lastet opp på github slik at om en ønsker kan man klonе det, åpne prosjektet i Android Studio, og kompilere appen selv. Appen kjører på API-nivå 31. Det gjør at appen ikke vil fungere på eldre versjoner av Android, men var nødvendig for SplashScreen og MapBox sin stabilitet.

Produktdokumentasjon

Kvalitetssegenskaper

Evaluering av funksjonalitet

Grunnidéen til appen er utformet av oss, men funksjonaliteten er utviklet i samarbeid med brukernes tilbakemeldinger fra intervju, geriljatesting og brukertesting, som vist i designprosessen. Den sentrale funksjonen til appen er å vise værforholdene rundt solnedgang, men for å skille dette fra en alminnelig værapp undersøkte vi mer spesifikt hva slags forhold som ligger til rette for en estetisk solnedgang.

Arkitektur. Arkitekturen baserer seg på MVVM-arkitektur. Hvert API har en tilhørende DataSource-fil og repository. Databasen består av tre tabeller for henholdsvis værdata, stedsinfo og bildereferanser. Et større repository, PlaceRepository samler de mindre repositoriene og gir et enhetlig grensesnitt for dataaksess. Dette samler logikken og reduserer kompleksitet og potensial for feil i koden da vi unngår å håndtere mange ulike repositorygrensesnitt i Viewmodels.

Dataklassen PlaceInfo er sentral, og inneholder data om lokasjon, værforhold og angir om stedet er favoritt eller et av de forhåndsinnsatte lokasjonene. Andre sentrale dataklasser er SunEvent, som holder informasjon om tid for solnedgang, blå og gyllen time, og værforhold.

WeatherConditions holder informasjon om luftfuktighet og de forskjellige skylagene, og logikken som håndterer den ligger i ForecastRepository. Ved API-kall lagres informasjonen i databasen og databasen kan akkсессeres gjennom ForecastRepository.

Vi bruker Viewmodel Factory og dependency injection til å opprette Viewmodelene, slik at de alle deler den samme instansen av PlaceRepository.

I interesse av å ha kode som er lett å lese og vedlikeholde har vi forsøkt å unngå «code smell» (Lindsjörn Y., 2024, p. 27) og oppnå høy kohesjon, lav kobling. Dette var fra starten et ledende prinsipp grunnet usikkerhet på hvor mange «features» vi kom til å få gjennomført. Det var viktig at det skulle være lett å føye til mer informasjon og flere funksjoner på ferdige objekter. Vi føler vi har fått dette til i grei grad – Objektene i datalaget har veldig spesifikke roller

(høy kohesjon), og er ikke veldig avhengige av hverandre (lav kobling) i den forstand at de opererer fritt – Alt går ikke under om én API er nede eller brukerposisjon ikke er aktiv.

Best practice. Som uerfarne utviklere visste vi ikke hva som var “best practice” da vi begynte å strukturere koden vår, og vi endte dermed opp med noen konvensjoner som vi senere oppdaget var dårlig kodeskikk. Blant annet sender vi instanser av NavHostController til alle skjermene våre fordi de har hver sin nav bar, noe vi oppdaget sent i løpet at man burde unngå. Noen av problemene våre endte vi opp med å løse før innlevering. Den første versjonen av dialogen som kommer opp når bruker vil legge til et nytt sted var knyttet direkte til skjermens Viewmodel og tok inn en instans av denne som et argument. I slutfasen skrev vi om denne koden til å benytte seg av en event handler og ta inn alle funksjonskall som argumenter. Slik kunne den kobles av MyPageScreen og vi kunne bruke den samme koden for å legge til nye steder fra kartet som befinner seg på en helt annen skjerm.

I koden har vi laget en pakke som heter sharedcomponents, der vi utformet modulær og gjenbrukbar kode for UI-komponenter. Den ovennevnte dialogen er en del av denne pakken, samt ListCard (som brukes på alle skjermer for å vise flere steder i en liste), Top- og Bottom bar, som brukes til navigasjon på alle skjermer.

Beregning av værforhold

En av brukerne spesifiserte at hen ikke ville tolke værddata selv, så vi valgte å representere det med ikoner i tillegg til en gradering. Via LocationForecast får vi informasjon om tettheten av skyer i forskjellige høyder, og ifølge artikler, blant annet Photography Academy (2019) er det ideelt med lav luftfuktighet og litt tetthet av skyer i middels og høye skylag. Vi tilegnet de ulike variasjonene vektorer, og designet en algoritme som basert på disse verdiene gir en vurdering.

Håndtering av bilder

Bilder til de innebygde lokasjonene er lagret internt i assets-mappen. Ellers lagres bilder i appens tildelte «local storage»-mappe når de lastes opp.

Brukskvalitet

Responstid. Vi har fokusert på hastighet i appen, og parallellisering av API-kall gjør at alle kallene for hvert sted sendes når kartet åpnes og svarene lastes inn på under ett sekund hvis internett er tilgjengelig. Resultatene lagres internt i databasen, og det gjøres ikke nye kall før dataen er over én time gammel. Alle skjermene i appen bruker denne databasen, som gjør at navigering flyter fint.

WCAG og Universell utforming. I løpet av prosjektet hadde vi fokus på universell utforming med ønske om en brukervennlig og intuitiv app. Ved å følge WCAG 2.1- standarden har det vært tydelig hvilke aspekter som burde tas hensyn til i utviklingen og hvordan de kan implementeres på best mulig måte.

Mulig å oppfatte. Ikonene vises med både farge, tekst og farge, og der tekst skal fylles inn er obligatoriske felt merket med stjerne, i tillegg til at de endrer farge og gir beskjed dersom de ikke er utfylte. Contrast checker (WebAIM web accessibility in mind, 2024) er brukt for å sjekke kontrasten til appen i lyst tema og mørkt tema, og MaterialTheme sitt fargetema er benyttet i store dele av appen.

Mulig å betjene Det er overskrifter på alle skjermer som gir brukeren en beskrivelse av hvor hen er. Overflødig informasjon er lagt til lokasjonenes informasjonsskjerm for at brukeren ikke skal bli overveldet eller distraheret på listeskjermen. Ellers er innholdet på alle skjermer delt opp i mindre deler med underoverskrifter som gjør det enklere å sortere hva som «hører sammen» av informasjon. Det er mulig å dobbeltrykke for å zoome inn på kartet og på den måten kan brukeren velge å benytte seg av kun en hånd for å navigere på skjermen. Vi hadde som krav at mest mulig interagering med appen skulle skje via trykk og ikke med “bevegelse” av fingeren. Dette er dog ikke en innebygget mulighet for kartet eller der brukeren må skyve for å se flere kort på favorittskjermen, listeskjermen og favorittvisning på hjemskjermen. Dette kravet kunne vært fulgt på en bedre måte som ved å legge inn en kontroller som lar deg navigere ved trykk.

Forståelig Etter ønske fra brukere støtter appen både norsk og engelsk, og appens språk baseres på telefonens systemspråk. For å holde symbolikken forståelig og konsekvent har alle ikoner samme betydning på tvers av skjermer og språk. Kjente grensesnittmetaforer er også brukt, for eksempel hjerte for å symbolisere favoritter og hus for å symbolisere hjemskjerm. På hjemskjermen vises en animasjon mens den laster inn data for å signalisere til bruker at den jobber med å få informasjon frem fra bak kulissene.

Robusthet All tekstlig informasjon og ikoner i appen er formatert slik at brukeren kan benytte hjelpeteknologi til å lese opp teksten.

Pålitelighet/Fravær av feil Vi har fulgt MVVM som hjelper «separation of concerns» og sikrer et tydeligere skille mellom lagene. Dette gjør koden lettere å lese og vedlikeholde. Bilder lagres i intern mappe som gir rask og sikker tilgang, og bildefunksjonen vil fungere uavhengig av hva som skjer med bildene i brukernes galleri. Alle Viewmodel funksjoner som interagerer med

datalaget er skrevet med try/catch, og viser deskriptive feilmeldinger på norsk eller engelsk til bruker basert på hvilken type feil de fanger opp. Funksjonene i datalaget vil kaste exceptions av forskjellige typer når de oppdager feil. F. eks. Vil getNewForecastData kaste en InternetException hvis den ikke får sendt http-forespørselen. For å enkelt kunne vite hva slags feil som oppsto når Viewmodel fanger de opp, har vi laget våre egne Exception underklasser. Feilmeldingene vises på en snackbar hvor bruker enten kan laste inn data på nytt, eller krysse ut om dette ikke er ønskelig. Appen kan da brukes med eksisterende data selv om man mangler internett. Dette gir meningsfulle feilmeldinger, uten at appen kræsjer.

Mangler i systemet: Når man åpner appen første gang vil koden lete etter brukerens posisjon samtidig som systemet ber om tillatelse til å bruke posisjonen. Dette gjør at brukere må trykke på “last inn på nytt” etter å ha gitt tilgang til posisjonen, eller navigere til en annen skjerm og tilbake til home, før HomeScreen kan laste inn værdata på brukerens posisjon.

Videre er en mangel at fargen til snackbar ikke er WCAG-vennlig. Dette ble ikke prioritert siden vi først oppdaget det mot slutten av prosjektet. I tillegg vil ikke snackbar lastes inn om det er feil på maplistskjermen.

Vi ønsket å få til at bruker kunne endre språkinnstillinger manuelt, men dette utgikk på grunn av tidspress og den følger kun systemspråk, enten norsk eller engelsk.

En spesifikk bug som sannsynligvis aldri vil finne sted (dog vi oppfordrer bruker til å prøve) er at vi benytter oss av koordinatene til Null Island (0°N 0°E) til feilhåndtering og for å sjekke om brukerlokasjon har blitt oppdatert. Skulle bruker befinne seg presist (innen få meter) ved disse koordinater vil hovedskjerm ikke laste og kartskjerm vise Norge.

En siste mangel er at endringer i settings vil ikke skje før appen restartes.

API og EDR

I utgangspunktet var det krav at API-kall skulle gjøres via IFI sin proxy til MET, men siden var en av veldig få grupper som valgte case 5 ble vi i begynnelsen av prosjektet fortalt at det ikke var nødvendig å sette opp proxy. Kvelden to dager før levering ble vi fortalt av en av de andre gruppene at det var mulig og forventet av oss allikevel, men grunnet tidspresset valgte vi å holde oss til API-kallene som de er.

Et av kravene til case 5 var å anvende Environmental Data Retrieval (EDR), som er en ny API-standard for søk i geodata. Via MET sine dokumentasjonssider fant vi API-ene Sunrise 3.0 for informasjon om solnedganger og Locationforecast 2.0 for værdata. Disse var svært enkle i

bruk – gi de koordinater på WKT-format og eventuelt en dato, og den returnerer informasjonen på JSON-format. Skjelettene for API-responsene ble lagret i filer med mange klasser for å gjøre det enklere å holde styr på. Siden API-responsene inneholdt mer informasjon enn vi trengte lagde vi våre egne objekter som kun holdt på informasjonen vi ville jobbe med videre. Fra LocationForecast tar vi kun vare på data for de tre nærmeste dagene fordi det ble unødvendig mange API-kall på Sunrise 3.0 da den ikke støtter spørring om flere dager av gangen, i tillegg til at brukerne ikke så nytten. Vi ble nødt til å bruke LocationForecast sin «Complete» respons da «Compact» manglet info om tettheten i de forskjellige skylagene som var relevant for oss.

Sunrise 3.0 manglet data for den blå og gylne time, men vi fant SunriseSunset.io som gav oss informasjonen vi trengte for å bestemme dette. Blå/gyllen time er noe løst definert, men vi definerer blå time som timen etter solen har krysset horisonten, mens gylle time som timen før.

Brukerne forespurte karttjenester, og vårt valg falt på Mapbox da det tilbød tjenestene vi trengte gratis, og vi hadde hørt det var greit å bruke. Vi benyttet oss blant annet av deres kart, “forward geocoding”, “reverse geocoding,” og “directions” som gjorde det enkelt å orientere brukeren i forhold til lokasjoner på kartet, i tillegg til å anslå avstand og reisetid. Mapbox var ikke alltid helt optimal da dens «composable»-funksjonalitet fortsatt er eksperimentell, under utvikling, og mangelfullt dokumentert. Den krevde en del eksperimentering for å implementeres. I tillegg kommer «forward geocoding» med suboptimale og noen ganger feil forslag til adresse.

Både Sunrise 3.0 og SunriseSunset.io gav oss et litt overraskende «problem». En dag krasjet appen da vi kjørte samme koden vi hadde testet hele uken. Vår forhåndslagrede lokasjon ved Nordkapp prøvde å laste blåtime og solnedgang da det var midnattssol, og API-en responderte med «null». Dette stod forklart i dokumentasjonen vi hadde skummet litt for fort, og vi fikk håndtert responsen og gitt fornuftig melding til bruker.

Ellers opplevde vi ingen utfordringer med EDR i forhold til andre format vi brukte. Spørringer og håndtering av JSON-responser var like greit å jobbe med det var Sunrise 3.0, SunriseSunset.io eller Reverse Geocoding. Responsklassene ble skrevet automatisk via «JSON To Kotlin Class», og vi lagde våre egne objekter som hentet ut infoen vi trengte.

Prosessdokumentasjon

Valg av smidige metoder - Scrumban

I starten av prosjektet satte vi oss som mål å følge en smidig praksis for arbeidet. Fordi vi ønsket en fast struktur for møter slik Scrum tilbyr i tillegg til en kontinuerlig flyt av arbeid slik

Kanban gir, falt valget på Scrumban. Vi brukte elementer fra Scrum som Sprint Planning, standup og retrospektiv, akkompagnert av Kanbans visuelle metode for oppgavehåndtering. I stedet for et scrumbanbrett som kun brukes for én sprint, har kanban-brettet blitt brukt for hele prosjektet, der vi har flyttet arbeidsoppgaver hver sprint fra «To-Do» og «Doing» til «Done». Her brukte vi Trello, og hadde ett brett for koding og ett brett for rapport og design, der vi ukentlig organiserte oppgaver etter prioritet. Hver sprint varte i en uke med Sprint planning og retrospektiv på tirsdager og Standup på fredager.

Samtlige hadde lite erfaring med Scrumban som rammeverk, og det ble derfor en læringskurve å integrere denne formen for struktur. De første ukene innebar mye “learning by doing”, men vi fikk fort forståelse av arbeidsflyten, samt hvordan benytte Kanban-brettet.

Prosjektet var en god arena for å teste rollen som scrummaster og vi bestemte derfor at alle skulle prøve seg i rollen. Som scrummaster skulle man skrive møteagenda i Discord dagen før, bestille grupperom og være ordstyrer i løpet av møtene. Scrummaster måtte ha god oversikt over sine egne og andres arbeidsoppgaver, noe som gjorde at man fikk et overordnet blikk på prosjektet. Det var derfor ekstra nyttig at alle fikk bidra i rollen. I tillegg hadde samtlige forskjellig stil og fokus på møtene som gjorde det spennende å oppleve hverandres lederstiler.

Sprintene ga oss frister å forholde oss til som har gjort at vi har sett fremgang og hatt god flyt i prosjektet uke for uke. Standup har vært nyttig for å holde resten av gruppen oppdatert med prosjektets fremgang, men også for å be om hjelp om en skulle trenge hjelp med oppgaver.

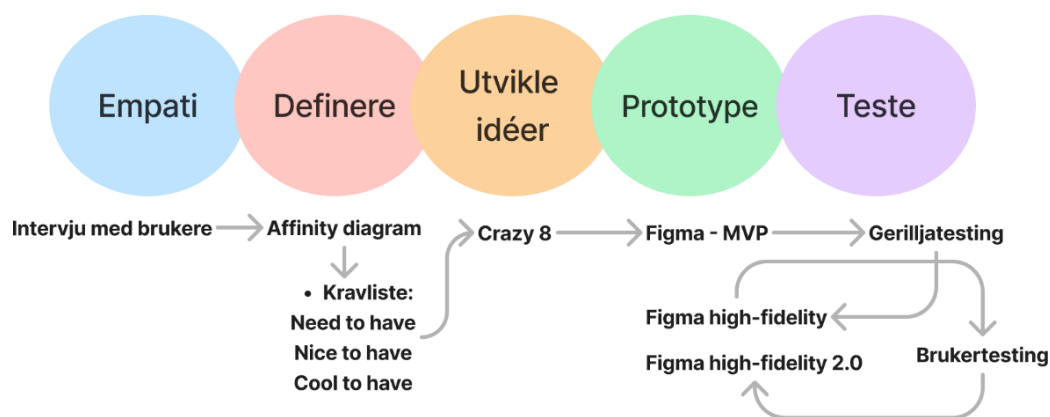
Selv om det var en del som måtte på plass i starten for å få til en god integrering av Scrumban, slik som scrummasters rolle og tid for møter, så vi nytten av et godt rammeverk for struktur etter hvert som prosjektet gikk. I tillegg ser vi viktigheten av å ha gjennomført et prosjekt med smidig arbeidsmetodikk da det gir oss en idé av arbeidslivets struktur.

Designprosessen

For designprosessen er design thinking (Dam, R. F. ,2024) brukt som metode. Metodikken egner seg godt til problemer kjennetegnet av ufullstendige, motstridende krav som stadig endrer seg, og var derfor en fin måte å utforske caset fra ulike vinkler. I tillegg er design thinking en fleksibel og iterativ metodikk hvor man kan hoppe frem og tilbake mellom de ulike fasene etter behov. Det satte vi pris på da vi ønsket at brukerne skulle være i fokus i løpet av prosessen (Dam, R. F. (2024, March 1)). På figur 11 illustreres det hvordan våre faser har inngått i design thinking prosessen.

Figur 11:

Illustrasjon av design thinking og dens faser i forhold til vår prosess.



1. Empati – Intervju

I første fase var det fokus på å forstå brukerne. Tilfeldigvis var en av prosjektmedlemmene hobbyfotograf og med i styret for Studentenes Fotoklubb på Chateau Neuf. Dermed hadde vi tilgang til intervjuobjekter i tillegg til breddekunnskap fra en på teamet. Gjennom et semi-strukturert gruppeintervju stilte vi åpne spørsmål som ga rom for diskusjon og oppfølgingsspørsmål. En intervjuguide (vedlegg 1) og samtykkeskjema (vedlegg 2) ble utformet før intervjuet. Samtykkeskjema ble gjenbrukt i brukertesting. Lydopptak og notater ble brukt som metode for datainnsamling og sikret nok materiale til å etablere behov. Å møte engasjerte hobbyfotografer som var villig til å gi oss uvurderlig innsikt i deres behov og krav til appen var veldig motiverende. Intervjuet ga oss en bedre forståelse av hvilke aspekter vi burde fokusere på, og hvilke som var av mindre interesse for brukerne. Mange av idéene hadde vi ikke tenkt på fra før, og vi fikk flere aha-opplevelser i løpet av intervjuet. For eksempel var de interesserte i tidspunktet for den blå og gylne time. Kartleggingsfasen gikk dermed ut på å få breddeforståelse fremfor å låse seg til en endelig løsning. I retrospekt burde vi hatt en forventningsavklaring før intervjuet da noen av funksjonalitetene brukerne ønsket var utenfor omfanget til caset. Dette var en viktig lærdom som ble tatt med videre i brukertesting.

2. Definere – Affinity diagram

Defineringsfasen gikk ut på å finne den viktigste innsikten fra kartleggingsfasen for å komme frem til et klarere bilde av brukernes behov og krav til appen. Her ble affinity diagram brukt som metodikk for å utforme temaer ved å samle tanker, sitater og idéer fra intervjuene på

post-it lapper. Basert på funnene fant vi sammenhenger og dro linjer på tvers av de forskjellige brukerne (vedlegg 3). Dette ga oss forståelse for brukernes behov som igjen ga oss idéer til funksjonaliteter og hvilke aspekter av appen det var lurt for oss å fokusere på videre i designet. Innsikten fra defineringsfasen ga oss dermed mulighet til å definere de funksjonelle og ikke-funksjonelle kravene som ga oss en samlet forståelse av hvordan vi skulle bevege oss videre med tanke på appens utforming. Kravene ble deretter hierarkisk sortert ut fra prioritet.

3. *Utvikle idéer – crazy 8 m/ brainstorming*

Til nå hadde vi hatt et åpent sinn og forsøkt å få konkret data. Nå kunne vi basere oss på denne dataen og brainstorme idéer med hjelp av idémyldringsmetoden Crazy Eight (vedlegg 4). Det var viktig at alle i gruppen fikk delta på aktiviteten slik at de som implementerte low-fi prototypen i Figma hadde et godt utgangspunkt og felles forståelse av gruppens ønsker. Her fikk vi skissert ned tanker om screens, pop-ups og oppsett som igjen ga nye idéer til krav. Videre ga det oss et godt grunnlag for brainstorming og idemyldring rundt appens utforming og funksjon. I tillegg ble det tydelig at vi var samstemte rundt flere likhetstrekk ved designet, og med bakgrunn i dette kunne vi gå videre og implementere designet i Figma.

4. *Prototyping*

Prototypingsfasen skjedde i flere omganger (vedlegg 5). Basert på funnene fra crazy eight lagde vi den første prototypen i Figma som ble utgangspunkt for low-fidelity demo. Senere ble denne geriljatestet med venner og familie slik at vi igjen kunne videreutvikle den til en mer high-fidelity prototype. Denne versjonen ble utgangspunktet for brukertesting. Under brukertesting vi i tillegg til det visuelle, opptatt av at prototypen skulle være interaktiv. Dette for å se om brukerne likte flyten i appen. Her viste vi frem to “flows” som ble testet ut og gitt tilbakemeldinger på. Etter brukertesting hadde vi et enda klarere bilde av hvordan vi ønsket å gå videre med designet av appen og kunne med det lage en siste versjon i Figma som vi brukte som mal for det endelige produktet.

5. *Teste –Geriljatesting og brukertesting*

Geriljatesting. I påskeferien geriljatestet vi vår low-fidelity Figma-prototype med venner og kjente. Dette for å sikre at den generelle layouten og konseptet bak appen var intuitiv. Vi lagde en uformell mal (vedlegg 6) som vi brukte under hver test slik at vi systematisk kunne samle inn tilbakemeldinger om appens oppsett, funksjoner og design. Totalt fikk vi geriljatestet med 14 personer. Ved å gjennomføre denne aktiviteten klarte vi å tydeliggjøre vår visjon og samtidig

sikre at den kommende brukertesting i større grad fokuserte på appens funksjonalitet og brukervennlighet.

Ved første sprintplanlegging etter påsken samlet vi alle tilbakemeldingene og fant fort ut at samtlige brukere var forvirret av hjemskjermen. Den var ikke tydelig nok, og flere var usikre på hvordan de skulle tolke ikonene. Var det en vær-app eller en app for solnedganger? I tillegg var det forvirring om hva som lå til grunn for stjernerangeringen av solnedgangene. Til brukertesting valgte vi derfor å ha med en popup-boble som skulle forklare det nærmere når brukeren trykket på stjernene. Med dette i bakhodet gikk vi videre med å utvikle en high-fidelity-prototype som vi deretter kunne teste på målgruppen vår – hobbyfotografer.

Brukertesting. Med vår high-fidelity prototype gjennomførte vi tre brukertester (vedlegg 7) der prototypens funksjonalitet og brukskvalitet stod i fokus. Ved å teste prototypen i Figma kunne vi få innsikt uten å måtte bruke tid på å implementere funksjonalitet i kode som ikke ville blitt brukt. Vi ga en kort introduksjon til appen før vi startet opptak og ga deltakerne mulighet til å klikke seg gjennom to forskjellige versjoner av prototypen. Spørsmål slik som “Hva synes du om layouten?”, “Hva tror du knappene betyr?” og “ville du brukt appen?” ble spurt for å få en forståelse for brukernes opplevelse.

Ting som for oss var selvsagte var ikke like intuitivt for brukerne. Blant annet synes de det var forvirrende med ikonene som ble brukt til “rating” og ønsket at vi skulle bytte ut disse med skrift i stedet. I tillegg var ordet «rating» forvirrende da intervjuobjektene assosierte dette med «bruker anmeldelser», og ble derfor byttet ut med “weather conditions”. Brukerne ønsket ellers en beskrivelse av hvordan vi kategoriserte gode og dårlige forhold. De syntes fargene i designet var for sterke og distraherende. I tillegg var samtlige brukere enige om at infoskjermen med kamerainnstillinger var unødvendig og ikke kom til å bli brukt. Ellers var det viktig for brukerne å ha all nyttig funksjonalitet lett tilgjengelig på hjemskjermen som gjorde at favoritt-funksjonen fikk prioritet over “nice places today”. Brukerne nevnte videre at de likte å kunne se avstand fra sin posisjon til en lokasjon. Dermed ble denne funksjonaliteten med videre i designet. I tillegg var de opptatte av å kunne legge til sine steder direkte fra kartet med “long-press” slik at de slapp å gjøre det via «Min Side/My Page». Alle ovennevnte forslagene ble tatt med videre og implementert i det endelige designet.

Vi hadde til nå vært usikre på flere aspekter ved appen, slik som om vi skulle ha settings lett tilgjengelig i navigasjonsbaren og hvor vi skulle plassere “Mine Lokasjoner/MyLocations” og

“Favoritter/Favourites”. Her var brukerne enige i hvilken versjon av prototypen de foretrakk, som igjen gjorde oss i større grad sikre på hva vi burde implementere. Alt i alt var brukerne positive til appen, og ga oss verdifulle tips til hvordan å gjøre den bedre og mer rettet mot målgruppen vår.

Implementeringsprosess

Første fase: fra Kickoff til MVP

Idémyldringen begynte ved første møte på kick-off etter å ha vært gjennom Stinky Fish (rolleetableringslek) og blitt litt kjent med hverandre. Vi diskuterte de forskjellige casene, hva vi likte og mislikte med dem og landet til slutt på egen case - «*Skumring*». Denne oppgaven gav oss frihet til å utforske forskjellige API-er og løsninger, noe vi muligens brukt litt for mye tid på. Dermed ble vi forsinket i å forene våre tanker om det endelige produktet.

Verktøyene vi brukte gjennom prosessen ble fort bestemt. For kommunikasjon valgte vi Discord som er godt egnet til lavterskel kommunikasjon, og Google Drive ble brukt som samlepunkt for alt av tekst- og bildedokumenter. Trello ble brukt som Kanban-brett, Figma til prototyping og skissering, og UiO sin GitHub ble brukt som VCS. Android Studio ble brukt til å utvikle produktet, og draw.io til å modellere dataflyten.

Første sprintplanlegging var uken etter kickoff hvor vi ble enige om hvilken smidig praksis vi skulle følge. Vi ønsket å lage en grov disposisjon av hvordan vi så for oss at oppstartsfasen skulle være - En plan for «the fuzzy front end» (D. G. Reinertsen, 1999). Vi kartla hvilke erfaringer hvert medlem tok med seg fra tidligere emner og prosjekter. Et ønske var at alle skulle ha likt eierskap til alle aspekter av prosjektet, og vi presiserte at vi ville unngå å låse oss til ett arbeidsområde. Målet var at alle skulle være innom alle «poster» - slik som design, koding og rapportskrivning. Det var viktig å ha en åpen kultur der det var rom for å ta initiativ til å starte på oppgaver og prøve noe nytt, selv om det var innenfor et område man ikke var kjent med fra før.

Vi ville involvere brukerne så tidlig som mulig for å forsikre oss om at vi var på riktig vei og utviklet i samsvar med brukernes behov. I tråd med dette hadde vi idédugnad og fikk skissert noen forslag til hva appen skulle gjøre.

Parallelt med intervjuene ble et kodeskjelett og API-modeller satt opp, og etter input fra brukerintervju lagde vi et «affinity diagram». Vi tegnet opp en skisse over arkitektur og dataflyt, og med dette hadde vi en disposisjon å jobbe mot. Deretter kunne vi sette i gang med å implementere deler av arkitektur hver for oss med felles rammeverk og mål.

Nå som vi offisielt var i gang ønsket vi å få formalisert konvensjoner og skikker når det kom til kodedelen. Dette handlet i hovedsak om hvordan man navngir objekter, funksjoner og branches i Git, men også hvordan man tester for og håndterer feil. Vi prøvde tidlig å formulere krav til MVP, men grunnet at alle var ferske til denne typen utvikling var de noe utydelige. Det var vrient å bestemme detaljer, men etter «Crazy Eight» kom vi til enighet om et grunnlag vi kunne bygge videre på.

Under mobbprogrammering begynte vi med et repository som skulle håndtere lokasjoner og værvarsel for disse. Det var ikke veldig produktivt da forskjellen i ferdighetsnivået innen koding var ganske vesentlig og ikke alle i gruppen var like kjent med strukturen vi jobbet mot. Det ble heller brukt som en mulighet til å ta en gjennomgang av hva som hadde blitt gjort, hvordan API-ene fungerte og hva som skulle gjøres kodemessig. En viktig leksjon videre var at vi skulle planlegge slike aktiviteter bedre og at gruppemedlemmene skulle stille mer forberedt.

Videre ble parprogrammering gjennomført som aktivitet med enhetstesting av noen API-responser. Dataen ble sett over manuelt for å sørge for at den ble konvertert riktig. Leksjonen fra parprogrammeringen var at forberedelse og planlegging var viktig for å få mest ut av tiden sammen. Uformelle parprogrammeringer ble gjennomført gjennom hele prosjektets gang.

Uken før ferdigstilling av MVP-en ble en kortere sprint enn vanlig, i tillegg til at det var en del løse tråder som måtte samles. Mange forskjellige elementer var designet og kodet, men ikke lagt til i main branch. I den anledning valgte vi å gjennomføre en aktivitet vi kalte «Merge Workshop» (eller Mergeshop) som ble en utvidet stand-up og en slags «code review» hvor vi i plenum gikk gjennom pull requests og løste merge-konflikter. Dette var nyttig som teamaktivitet, men også for å kartlegge hva som manglet før vi var fornøyde med appen som MVP. Etter dette ble noen små justeringer gjort og MVP-en ble presentert med fire skjermer.

Refleksjon: Første fase

For flere i gruppen var det første gangen en jobbet med et prosjekt av slikt omfang, og mange av planene ble til mens vi holdt på. Forsøk på struktur, slik som arkitektur, tidslinje (vedlegg 8) og oversikter ble ikke brukt slik vi ønsket - De ble laget, men i mindre grad tatt frem under møtene og ikke referert til mens vi planla sprinter. Gruppen brukte mye tid på møter for å få en felles forståelse av hvordan prosjektet skulle gå frem. I retrospekt ser vi at planene burde vært tydeligere kommunisert og strengere fulgt.

Videre kan valget av åpen case ha bidratt til usikkerhet da vi ikke hadde en ordentlig kjøreplan. På grunn av denne friheten var det vanskeligere å si hva appen skulle være og gjøre. Vi var enige om noen grunnfunksjoner, men samtidig opptatte av at det var målgruppens behov som skulle være styrende. Noe vi kunne startet med tidligere var oppsett av API-er, implementering av Mapbox og datastruktur. Grunnet dette følte vi at vi lå etter tidsskjema, som ble en unødvendig stressfaktor vi kunne unngått. Ellers opplevde vi å være senere ute med å formalisere arkitekturen enn vi ønsket, som førte til tidspress og at vi forhastet oss i koden. Vi gjorde noen feil med behandling av koordinater hvor vi hadde forvekslet longitude- og latitudeverdiene, som plasserte lokasjonen midt i det Arabiske hav. En annen svakhet ved koden var manglende try/catch, som vi måtte bruke påsken på å få på plass. Disse feilene var en påminnelse om at vi måtte kode oversiktlig og kommentere koden. Vi pådro oss noe teknisk gjeld da vi utformet den første arkitekturen. Fordi den var alt for enkel, endte vi opp med å lage flere små repositories og dataklasser som vi senere måtte bruke tid på å slå sammen og omstrukturere.

Vi ble heller ikke enige om en struktur på hva møtene burde være og inneholde før midten av prosjektet. I tillegg valgte vi tidlig at ingen skulle låses fast i sin “naturlige arbeidsoppgave” og noen tok på seg flere oppgaver uten å delegerer til andre. Dette førte til at flere i gruppen opplevde det som vanskelig å finne sin plass og bidra i starten, noe som ble tatt opp og adressert etter hvert som vi kom mer i gang og planer tok form.

Andre fase: Hovedfasen

Etter å ha ryddet i koden gjennom påsken og geriljatestet bestemte vi oss for å forberede oss til brukerintervju. Med bakgrunn i funn fra geriljatestingen gikk vi gjennom MVP-prototypen skjerm for skjerm og kartla hva som måtte forbedres. Denne måten å gå gjennom skjermene på viste seg å være svært verdifull og noe vi fortsatte med. En high-fidelity-prototype tok form og denne ble brukt i brukertesten. Funnene ble brukt til å finne en balanse mellom utforskning og begrensning. Etter brukertesting fikk vi snakket med veiledere om features vi hadde lyst til å ha med i den ferdige appen, og som et resultat hadde gruppa et klarere bilde av hva vi skulle jobbe med fremover. Gjennom prioritering landet vi på å utvikle mer robuste og elegante funksjoner, og bestemte oss for å fokusere på kjernefunksjonalitet.

I løpet av denne fasen ble skjermene ferdige én etter én, samtidig som flere funksjonaliteter falt på plass. Selv om pushvarsel om fin solnedgang var noe vi valgte vekk fikk

vi på plass de aller viktigste funksjonene, slik som språk, lys- og mørkt tema, parallellisering av API-kallene og en database til å lagre informasjon.

Kode - utforsking og tilspisning

Etter å ha implementert databasen bestemte vi oss for å refaktorere hele kodestrukturen til at all data om lokasjoner og vær ble hentet fra databasen. Slik kunne vi redusere API-kall og implementere caching. Deretter skrev vi om datalaget slik at vi kombinerte flere små repositories til et nytt og stort repository kalt PlaceRepository. Denne tar for seg kommunikasjon med databasen, API-kall og andre funksjoner som er nødvendig for Viewmodels.

I starten av fasen utforsket vi muligheter for ekstern database og server. I tillegg hadde både teamet og brukere nevnt en sosiale-medier-funksjon der man kan følge andre brukere og se deres lokasjoner og bilde, men med tanke på hvor mye ressurser det ville krevd valgte vi å gå bort fra disse. Vi valgte heller å gå for en Room-database som ikke krevde server, og var derfor lettere å sette opp. Implementasjonen av denne endte opp med å bli en flaskehals fordi flere planlagte funksjonaliteter var avhengige av databasen. Vi jobbet rundt flaskehalsen ved å utvikle skjermer og Viewmodels underveis, og koblet disse til databasen når den var ferdig. Flaskehalsen kunne vært løst ved å begynne på implementasjonen av databasen tidligere.

I denne fasen oppstod noe teknisk gjeld, men vi la også inn ressurser for å løse dette. Det var her at vi skrev den første versjonen av NewPlaceDialog, som var tett knyttet til MyPageViewModel og avhengte av å få inn en instans av denne som argument. Denne dialogen ble skrevet om til å bruke event handlers og ta inn funksjonskall som argumenter i slutfasen.

Refleksjon: Andre fase

Funksjoner og kode tar mer tid enn tenkt, derfor har fasen vært noe kaotisk. Fra starten ble feilhåndtering glemt og mer fokus ble lagt på fungerende kode. Dette ble implementert etter MVP-en og satt som en prioritet videre for å få en robust og pålitelig kode. Generelt burde vi vært flinkere til å skrive ned tekniske beslutninger slik som applikasjonens struktur. Samtidig nådde vi mange milepæler og så appen ta form. I retrospekt kunne vi spart tid ved å benytte labtilbud for å løse tekniske problemer vi slet unødvendig lenge med. Vi burde vært bedre på å lese dokumentasjon, koden ble til tider preget av noe slurv som førte til lett unngåelige feil, for eksempel: Databasen ville ikke fylles, appen ble startet i to instanser, og at vær ved feil tidspunkt ble vist til solnedgang. Grunnet tidspress ble koden preget av «første løsning som fungerer» uten

at det nødvendigvis er den beste løsningen. I tillegg burde vi vært flinkere til å debugge og bruke logcat i koden, slik at vi kunne unngått unødvendig ekstraarbeid i slutfasen.

Ellers kan ønsket om at alle skulle være med på alt ha gått ut over designprosessen. Grunnet en treg kodelast ble det dårlig tid mellom brukertesting til en måtte starte å kode det endelige designet i appen. Det var dog et ønske om en high-fidelity prototype som kunne brukes som mal for UI og disposisjon til appen. Prototypingen ble derfor gjort parallelt med coding, og designet ble mer mangelfullt enn ønsket. Valg som hadde blitt tatt på møtet ble ikke reflektert i designet og det ble dermed forvirring rundt hva som skulle med i appen. Flere gruppemedlemmer burde bidratt med å avlaste de som satt med design.

På en annen side oppdaget vi en svært nyttig gruppeaktivitet, som var å gå gjennom prototypen i Figma skjerm for skjerm i fellesskap- Da fikk vi vedtatt akkurat hva som skulle være med videre, hva som faktisk var implementert i kode og ikke. Dette var for å forsikre oss om at alle var enige i hvordan appens utforming og funksjonalitet skulle være. Det gjorde at design og kode ble mer samkjørt og at det var lettere å skrive konkrete kodeoppgaver.

Uansett ser vi tilbake på en fin periode der vi hadde noen store teknologiske seire, og vi ble godt kjent med hverandre ettersom vi har sittet mye og jobbet fysisk sammen.

Avslutningsfase

Opprinnelig var planen å avslutte all programmering to uker før frist, men det var stadig småting vi hadde lyst å få på plass, samt bugs vi ønsket å fikse. Dermed definerer vi avslutningsfasen som de tre siste ukene før fristen, der vi ble ferdig med å implementere designet og oppfylte alle «must-have» krav og «need-to-have» krav.

Vi fikk ferdigstilt en del viktige funksjoner slik som tid for den blå og gylne time, språk, snackbar, tid- og avstandsberegning, og geocoding.

Videre ble Kotlin refactoring gjennomført og tester skrevet. I denne fasen begynte vi å jobbe mer med rapporten. Vi hadde opprettet et dokument for å organisere og visualisere status på de ulike rapportdelene, og delte disse ut til alle gruppemedlemmer. Rapportdelene lå i ulike mapper i forhold til hvorvidt de var påbegynt, klare for gjennomgang eller ferdige. Til slutt samlet vi alt i ett dokument for å flette sammen teksten og håndtere dobbeltskriving.

I tillegg gikk vi over aktivitetene vi manglet. Vi valgte vekk TDD og Backlog Enrichment fordi vi tenkte de ikke ville gi oss noe verdi så sent i løpet. Hadde oppskriften til TDD blitt tilgjengelig tidligere ville den vært mer aktuell.

Kravspesifikasjon - endringer og beslutninger

I starten av prosjektet tok vi utgangspunkt i åtte krav vi tenkte ville være relevante for vår app. De fleste har fulgt oss gjennom hele prosjektet, mens noen har blitt modifisert eller valgt vekk med bakgrunn i brukernes behov eller som følge av prosjektets begrensede tidsperspektiv. Vårt «hovedkrav» og det som kan sees på som en byggestein for appen var kravet om å bruke vær-og soldata for å foreslå hvor og når man kan ta fine bilder. Videre krav, slik som å kunne lagre favorittsteder, ble en viktig tilleggsfunksjon som viste seg å være av høy relevans for brukerne. Ellers har vi hatt fokus på WCAG, som beskrevet i større detalj i avsnittet om universell utforming. Herunder kommer fokuset i appen på å gi fornuftige feilmeldinger.

Etter intervjuet med brukerne ble det laget en kravliste hvor kravene ble rangert som «must-have», «need-to-have», «nice-to have» og «cool-to have». «Must-have» var krav vi ville ha på plass til MVP, mens resten var noe vi så for oss til et ferdig produkt. «Need-to-have» var høyt prioriterte krav som var ønsket av brukerne, mens i andre enden var «cool-to-have» som var mer ettertanker og ønsker. Vi var flinke til å prioritere de viktigste kravene og fant etter hvert ut at kvalitet måtte prioriteres over kvantitet, og noen krav ble derfor valgt bort. På tabell 2 vises vår nåværende kravliste. Som tidligere nevnt ble noen krav for kompliserte og tidkrevende, slik som push-varsler. Mens andre krav, som “anbefalte lokasjoner” og “hente data for langtidsvarsel” ble valgt vekk da det ikke var behov hos brukerne og de mente at hjemskjermen ble for rotete.

Tabell 2

Kravoversikt prioritert fra need to have, nice to have til cool to have.

Krav	Funksjonell/ ikke- funksjonell	Prioritet
Vise om det er bra forhold for solnedgang på et spesifikt sted	Funksjonell	Need to have
Hente data for korttidsvarsel (3 dager frem)	Funksjonell	Need to have
Presentere temperatur	Funksjonell	Need to have
En oversikt på kart over steder å ta bilde	Funksjonell	Need to have
Egen skjerm for lokasjoner	Funksjonell	Need to have
Appen skal ikke kræsje under noen omstendigheter	Ikke-funksjonell	Need to have
Gi forståelige feilmeldinger	Ikke-funksjonell	Need to have
Følger WCAG (minstekravet)	Ikke-funksjonell	Need to have
Gi en oversikt over steder å ta bilde i en liste	Funksjonell	Nice to have

Steder brukeren har lastet opp kan vises i en liste	Funksjonell	Nice to have
Appen skal lagre værdata	Funksjonell	Nice to have
Appen skal ha tilgang til værdata offline hver gang den henter fra API	Funksjonell	Nice to have
Presentere værdata for gitte lokasjoner	Funksjonell	Nice to have
Brukeren kan markere lokasjoner som favoritter	Funksjonell	Nice to have
Appen kan bytte mellom forhåndsinnstilte og live lokasjon	Funksjonell	Nice to have
Appen kan vise informasjon på Engelsk og Norsk	Funksjonell	Nice to have
Appen kan byttes mellom mørkt og lyst tema	Funksjonell	Nice to have
Appen skal benytte seg av parallelliserte API-kall	Funksjonell	Nice to have
Kartet skal komme opp innen 3 sekunder	Ikke-funksjonell	Nice to have
Presentere skydekke på en mer forståelig/håndterlig måte	Ikke-funksjonell	Nice to have
Bruker kan lage pins med egne lokasjoner	Ikke-funksjonell	Nice to have
Bruker skal kunne legge til bilder på en lokasjon	Funksjonell	Cool to have

Refleksjon: kravspesifikasjonen

I starten av hovedfasen la vi opp til at kravspesifikasjonen skulle være styrende for valgene vi tok i løpet av prosjektet. Dette sklei ut etter hvert som beslutninger ble tatt fortløpende i møter og på Discord. Vi tok også avgjørelser mellom møtene som gitt imot det vi opprinnelig var blitt enig om. Ingen hadde den overordnede rollen om å endre på kravspesifikasjonen, og når det ble mer å diskutere på møtene ble den ofte glemt. Vi burde heller gått gjennom kravene hvert møte slik at vi kunne ta vekk eller tilføye krav etter hva som dukket opp. Kravspesifikasjonen ble derfor ikke den styringsmekanismen vi først så for oss. Den ble ikke oppdatert annet enn rett etter intervju og brukertesting, for så å bli glemt igjen. Noen av kravene fra tidlig i prosjektet viste seg å være mer tidkrevende å implementere enn forventet. Selv om vi ikke fulgte kravlisten klarte vi å forholde oss til de fleste kravene, dog med noen bortfall.

Testing

Testing ble gjort til ansvarsområde for et gruppemedlem som dessverre forlot gruppen, og da andre gruppemedlemmer satt med fulle hender med andre deler av prosjektet, ble ikke testene skrevet før i slutten av hovedfasen. På en annen side har det vært noe refaktorering av kodestruktur og repositories som gjorde at flere av de tidligere enhetstestene ikke lenger fungerte.

Selv om det ikke ble skrevet mange enhetstester underveis har alt av funksjoner og kall blitt testet, stresstestet, debugget og logget via LogCat.

De første testene ble gjort i påsken da vi hadde fokus på å kvalitetssikre koden vi hadde skrevet til MVP-en så langt i prosjektet. Disse testene sjekket om DataSource-filen til Sunrise-APIet returnerer riktig verdi og om instansen inneholder riktig verdi i forhold til et API returverdiobjekt. Vi lagde dummydata i en JSON-fil og brukte Android Studio sine tester for å «asserte» at objektet som ble opprettet inneholdt riktig verdier. Videre har noen skjermer og komponenter blitt testet kontinuerlig ved bruk av preview. De første skjermene og komponentene vi skrev fulgte ikke “best practice” og var avhengige av argumenter som vi ikke kunne sende inn som en del av Preview. Mot slutten av prosjektet ble vi bedre på å følge anbefalingene, og komponenter som NewPlaceDialog og DeletePlaceDialog har fungerende previews.

Igjennom utviklingen og spesielt på slutten av prosjektet har vi utført stresstester på appen. Dette innebærer å finne alle tenkelige måter appen kan kræsje på. For eksempel ved å teste med og uten nett eller brukerlokasjon. Dette har vært en effektiv måte å finne feil man overser under kodingen.

Refleksjon

Utviklingsprosessen

Som uerfarne programmerere holdt vi oss til MVVM-arkitekturen som ga oss klare og sikre rammer. Vi møtte dog på noen utfordringer, for eksempel ved implementasjon av brukerlokasjon og MapBox da disse ikke er typiske kall/respons APIer, men hadde data som var kritisk at ble håndtert riktig og var tilgjengelig til forskjellige repositories. I tillegg var den overordnede oversikten tilbudt av kanbanboard veldig hjelpsom i at den lot oss identifisere en flaskehals. Mye avhengte av at databasen var klargjort, men arkitekturen og MVVM lot oss delvis jobbe rundt det da vi hadde klasser og funksjoner allerede modellert. En ting vi ikke har vært like flinke på gjelder spesielt i views/skjermer, der duplikater av kode finnes flere steder. Det vil si at hvis det er endringer i hva vi ønsker å vise og hvordan, vil det føre med en del manuelt arbeid.

Underveis har vi merket viktigheten av å gå gjennom kode i plenum og vise hva man har gjort fremfor å bare fortelle om det. Vi skulle ønske vi viste mer kode under møtene både i oppstartsfasen og hovedfasen. Da ville det vært lettere for gruppe medlemmene å henge med i hva som skjer på ulike fronter i prosjektet, som senker terskelen for å komme med forslag, innspill og

hjelp dersom et gruppemedlem står fast. På en annen side er det begrenset hvor mye en kan sette seg inn i ny kode som kun blir presentert foran en. Vi har opplevd at man må sitte litt med selv koden for å kunne gi noe verdifullt innspill, og da oppstår en tidsbegrensning som påvirker hvor mye man kan sette seg inn i andres problemer med koden.

I perioder kom mange pull requests på én gang, som resulterte i “mega merges”. Her oppsto merge conflicts, og vi gikk sammen for å håndtere disse, noe som var tidkrevende. Dette oppstod på grunn av dårlige vaner med oppdatering av branches mot main og usikkerhet om hvordan Github fungerer. Senere ble vi bedre på dette og fikk minsket antall konflikter.

En annen utfordring var mangel på standarder innen UI-koding. Gruppemedlemmene som lagde designet kodet i tillegg Viewmodels og screens. Vi burde fordelt dette arbeidet mer på andre gruppemedlemmer, siden dette var to ganske store oppgaver, der hvert av områdene ikke fikk den tiden som trengtes. Det ble dermed ikke tid til å sette standarder innen UI-koden, som gjør at vi har fått litt mer komplisert kode enn nødvendig. Det er småfeil i koden som kunne vært unngått dersom denne prosessen hadde fått mer tid.

Kvalitet og modularitet til koden har til tider blitt nedprioritert, for eksempel ser vi kode som repeterer seg selv hvor den kunne blitt delt i moduler. Som nye i Kotlin og Android Studio hadde vi en tendens til å prioritere funksjonalitet over effektivitet, altså så lenge koden gjorde det den skulle var det bra nok. Dette har ført til mer kompleks og uoversiktlig kode enn nødvendig.

Ellers var vi noe delt i hvordan vi skulle utføre lagring av bilder. Til slutt kom vi frem til to alternativer - å lagre de internt i appen eller å lagre en referanse til hvor bildet lå på telefonen. Problemet med sistnevnte var at referansen ikke ville fungere dersom bruker flyttet eller slettet bildet på telefonen. For å sikre robusthet sørget vi derfor for å lagre internt i minnet på appen. Dermed lagres referansen når bildet blir lastet opp, og hver entitet i databasen deler stedsId fra tilhørende stedsentitet i stedsdatabasen.

Igjen angående kodeskikk og hva som er ansett som «best practice» i apputvikling var det mye vi ikke fant ut før helt i slutten av prosjektet, og som resultat er koden i den forbindelse noe varierende. For eksempel får Views og composables tilsendt UI-states, Viewmodels og objekter om hverandre, mens de egentlig skulle hatt Event Handlers og fått funksjonskall som parametere.

Refleksjon: Teamarbeidet

På starten var gruppen noe delt da vi var to grupper som kjente hverandre fra før, derfor var det viktig å legge til rette for positive opplevelser sammen for å styrke teamdynamikken. Et

tidlig ønske fra samtlige gruppemedlemmer var å sitte så mye fysisk sammen som mulig, blant annet for å senke terskelen for å spørre om hjelp, forbedre arbeidsflyt, og naturligvis bli kjent med hverandre. Vi brukte blant annet Discord aktivt, og det ble for vane å si ifra om man var på skolen dersom andre fra gruppen skulle ønske å komme og jobbe i fellesskap. Dette har hatt en positiv effekt på gruppen, og mange av våre avgjørelser har blitt tatt utenfor de avtalte møtene og de digitale kommunikasjonsdelingskanalene. Dette var på godt og vondt da det sikret en smidig praksis, men disse avgjørelsene ble ikke alltid skrevet ned og formalisert.

I ettertid ser vi dog at vi kunne brukt mer avstemning ved uenigheter fremfor lange diskusjoner. En mulig ulempe av å ha rullerende scrummaster var at vi aldri fikk noen trygge nok i rollen til å ta styring over diskusjonene når de gikk for langt og sluttet å være produktive. Selv om vi i det store og det hele hadde god kommunikasjon og var trygge på hverandre var det til tider vanskelig å spørre om hjelp. Det handlet for det meste om at en ikke ønsket å være en «byrde» for de andre gruppemedlemmene, da det meste var nytt for alle. Et lite problem var at de fleste satt med veldig spesifikke og komplekse oppgaver alene, og det var vanskelig å be om hjelp til disse da det ville ta verdifull tid og energi fra andre som også slet med sitt. Det har dog vært stor velvilje blant medlemmene til å hjelpe hverandre når en har blitt spurt. Blant annet har de mest kodekyndige vært flinke til å forklare kode, samt skrevet dokumenter slik at alle skal forstå og snakke det samme språket. I tillegg har vi vært flinke til å rose hverandre for arbeidet vi har gjort og feiret små og store seiere i alt fra design til koding og rapportskriving.

Figur 12:

Bilder fra chat i Discord



Konflikt og håndtering

I løpet av hovedfasen blusset det opp en konflikt vi hadde adressert med veilederne tidligere i prosjektet. Den gang ble en ny teamavtale skrevet med presisering av begreper,

forventninger, ansvar, prioritering og fordeling av oppgaver. Mye tid hadde blitt brukt på konflikten, og ønsket fra samtlige gruppe-medlemmer var å fokusere på prosjektarbeidet. Vi så det derfor som nødvendig å involvere veilederne igjen for å løse opp i uenigheter vedrørende fordeling, mengde og kvalitet på arbeidet. Veilederne har et nyttig utenforperspektiv og det var viktig at alle gruppe-medlemmers frustrasjon, ønsker og behov kom frem. Resultatet til slutt ble at gruppen gikk fra seks til fem medlemmer. Selv om konflikten påvirket vår progresjon, har den bidratt til å styrke vår forståelse av teamdynamikk og viktigheten av å løse konflikter tidlig og på en konstruktiv måte i fremtidige samarbeidssituasjoner. Det er aldri én side av en sak, og det kan godt være at konflikten hadde blitt løst annerledes om det fra starten hadde vært en bedre struktur og forventningsavklaring.

Etter konflikten var løst ble fokuset fremover på produktet vi ønsket å lage, og det var enighet om å legge dette bak oss. Møtene kunne igjen gi rom for åpen diskusjon og gode samtaler om produktet vi ønsket å levere.

Aktivitetene

Gruppen opplevde at aktivitetene og deres gjennomføringskrav var et forstyrrende element. Vi måtte bruke unødvendig tid på å planlegge aktivitetene, og det var en stressfaktor å sørge for at vi sanket nok poeng. De fleste i gruppen hadde erfaring med prosjektarbeid og brukerinvolvering, og vi tror vi ville gjort mange av aktivitetene på eget initiativ.

Avslutning

Veien videre

Vi fikk ikke gjort alt vi ønsket å utrette med appen, men vi er fortsatt stolte av resultatet. Neste mål hadde vært å brukerteste den endelige løsningen for å finne ut om endringene fulgte brukernes ønsker. Ellers hadde neste funksjonalitet vært å få til at hjemskjerm oppdateres ordentlig når appen kjører første gang, i tillegg til pushvarsler for godvær ved lagrede lokasjoner. I utgangspunktet var det et håp om å lage en mer elegant algoritme som tok for seg vær over lengre tid, da visse vær-phenomen over perioder kan gi spesielle forhold. Videre ønsket vi å melde om forskjellige skytyper. I tillegg var det et ønske om en plattform for å dele bilder og lokasjoner med andre brukere. Vi ønsket å la brukerne kategorisere pins med tags og farger, og et brukerønske var en funksjonalitet som foreslo lokasjoner i nærheten. Et ønske var også at bruker kan velge standardlokasjon og språk uavhengig av systemspråk i appen. Vi kom langt med dette,

men dessverre ikke helt i mål. Til slutt kunne vi hatt med en søkefunksjon i skjermene med lister, hvor man kunne filtrert stedene etter tags.

Vår læring

Dette prosjektet har vært en øvelse først og fremst i samspill og koordinasjon. Vi har lært mye om viktigheten av å ta tak i konflikter tidlig, men kanskje det viktigste av alt har vært et felles mål å jobbe mot – Tidlig bestemme seg for hva man ønsker å oppnå, og tidlig konkretisere stegene som må tas. Vel så viktig er det at man faktisk holder seg til og gjør det som blir avtalt. En annen ting er å bruke mindre tid på diskusjon, mer tid på produksjon.

Viljar: Det er utrolig viktig å holde fokus på målet man har blitt enige om, og å være tidlig ute med konkretisering av stegene dit slik at man får avklart usikkerheter så fort som mulig.

Synnøve: Det er stor verdi i å visualisere kode og dataflyt fra starten av. Skriv kommentarer i koden underveis så alle kan henge med. Ikke vær redd for å be om hjelp.

Mari: Det er viktig å komme fort i gang med prosjektet, oppgaver tar vanligvis lenger tid enn man tror. I tillegg er det viktig med åpen kommunikasjon, spesielt ved konflikter eller uenigheter.

Anna: Forsøk å få på plass en struktur tidlig. Ikke vær redd for å utforske og prøve noe nytt. Vi er alle her for å lære og det er ingen bedre mulighet for det enn i prosjektarbeid.

Adrian: Det er verdt å bruke tid i starten på å lese seg opp om best practices før man begynner å skrive kode. Vi har noen strukturelle trekk med koden vår som jeg skulle ønske jeg hadde mer kunnskap om da jeg skrev skjelettet til prosjektet.

Tips til neste års studenter

Vårt råd er at det er viktig å tidlig bestemme seg for hva slags funksjonaliteter appen skal ha. I hvert fall i starten er det en god ide å holde omfanget lite, og heller bygge et solid utgangspunkt man kan legge på funksjonaliteter siden.

Kilder

Dam, R. F. (2024, March 1). The 5 Stages in the Design Thinking Process. Interaction Design Foundation - IxDF. <https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process>

Lindsjørn, Y. (2024, February 9). *IN2000: Kravhåndtering, modellering, design og prinsipper*. [PowerPoint slides].
https://www.uio.no/studier/emner/matnat/ifi/IN2000/v24/forelesninger/in2000.2024.02.08.krav_design_modellering.pdf

Photography Academy. (2019). *Photo Planning 101: Let's Talk about the Weather*. PhotographyAcademy.com. <https://www.photographyacademy.com/photo-planning-101-lets-talk-about-the-weather>

Reinertsen, D. G. (1999). Taking the Fuzziness Out of the Fuzzy Front End. *Research Technology Management*, 42(6), 25–31. <http://www.jstor.org/stable/24132855>

WebAIM web accessibility in mind (2024). *Institute for Disability Research, Policy, and Practice*. Contrast checker. Hentet 13.05.24 fra <https://webaim.org/resources/contrastchecker/>