Universitetet i Oslo - IN2000 Software Engineering med prosjektarbeid – 2023 – Rapport

JumpTime - Case 4. Åpent case - i lufta

Team 30

Medlemmer

Tobias Hagness Mortensen - tobiashm

Wale Belal Sayed - walebs

Marius Iversen Warlo - mariusiw

Ole Edvard Lømo - oleelom

Duc-vi Nguyen - ducvivn

Veiledere

Sondre Bader Wang - sondrbw

Jamila Mehmandarova - jamilakm

Antall ord: 12 380

Innholdsfortegnelse

1. Presentasjon

1.1 Introduksjon

Denne rapporten er en omfattende beskrivelse av vårt prosjektarbeid innen IN2000 - Software Engineering med prosjektarbeid, hvor blant annet planleggingsfasen, utførelsen og refleksjon er inkludert. Prosjektet varte fra 04/03/2023 til 26/05/2023, og var 12 uker med kontinuerlig arbeid for å danne et produkt som vi kunne være stolte av og som vi føler vi fikk mye erfaring fra.

Målet for prosjektet er at vi som gruppe får oppleve hvordan det er å jobbe som utviklere, og sammen skal vi lage en Android-applikasjon som bruker APIer fra Meteorologisk institutt. I prosjektet utviklet vi en applikasjon som skulle gi data om det er mulig å gjennomføre fallskjermhopp i et område ut ifra værdata fra APIer hentet fra Meteorologisk Institutt.

1.2 Teamet

Teamet skulle originalt bestå av seks medlemmer, men en av studentene trakk seg fra IN2000 emnet, og ble dermed ikke med. Tre av medlemmene går *Informatikk: programmering og systemarkitektur,* mens de andre to går *Informatikk: språkteknologi*. Vi tre som går samme studieløp er kjent med hverandre fra før men gjennom hele prosjektet ble alle godt kjent med hverandre, gjennom teambuilding.

1.3 Presentasjon av prosjektet

I *IN2000-Software Engineering med prosjektarbeid* emnet kreves det at studentene gjennomfører en større prosjektoppgave i gruppe, samt innlevering av obligatoriske oppgaver som må godkjennes for å kunne gå opp til eksamen. Dette er for å gi studentene i studielinjen kjennskap og erfaring om teamarbeid og systemutvikling, slik at vi er bedre forberedt til arbeidslivet etter studiene.

Prosjektoppgaven vi skal jobbe med er basert på en case fra Meteorologisk institutt. Vi valgte *Åpent case - i lufta*, hvor det var fritt valg til å lage hvilken som helst app, så lenge det var relatert til atmosfæren og brukte de obligatoriske datakilder. Kravet var å bruke minst to vær APIer.

Når gruppen først møtte hverandre ble vi enige om å komme med ideer til hva det åpne caset kunne handle om. En uke senere stemte vi over ideer, og en skydiving-værapplikasjon vant i flertall. Når vi ble ferdig med skydiving-aspektet av applikasjonen så ville vi også legge til flere ekstremsporter med tanken om å gjøre den så relevant som mulig for så mange. Ekstremsport er nisje, men ved å legge til mange ulike sporter kan vi dekke mange behov på en gang.

1.4 Løsning

Ekstremsport-værapplikasjonen skal brukes som et verktøy for interesserte, slik at de kan evaluere om det er mulig å utføre et hopp i et området. Applikasjonen skal gi informasjon om værforholdene og en anbefaling fra oss om hvor lurt et hopp er å utføre. Denne anbefalingen er basert på brukerundersøkelser og research vi har gjort innenfor sporten.

Vår løsning for å gjennomføre dette prosjektet var først og fremst å bli kjent med alle i gruppen, gjennom teambuildingaktiviteter. Da sørger vi for at alle i gruppen blir komfortable med hverandre, for prosjektet handler om teamwork for å få det gjennomført. Hvordan vi som gruppe jobber sammen er avgjørende for å lage en god applikasjon. Vi satte opp en plan for tider vi skulle ha møter, og hvordan prosjektplanen skulle se ut de neste 12 ukene.

Først og fremst så jobbet vi med å finne relevante APIer for prosjektet, og samtidig kunne parse dem til å kunne bruke dataen. Vi begynte også å undersøke hva fallskjermhoppere hadde forventet av en skydiving-værforhold app, og hvilke værforhold hopp kunne gjennomføres i. Vi lagde også ferdig et design og struktur i figma slik at alle i gruppen var på samme bølgelengde, og implementerte hvert komponent sakte men sikkert gjennom prosjektplanen. Kartet og APIene ble implementert, og i løpet av sprint 3 var en MVP (minimum viable product) ferdig. Etter MVP-en begynte vi å diskutere navn og logo design for applikasjonen, hvor vi stemte over å holde et blå-tema som representerer himmelen, og en

minimalistisk logo som hopper i fallskjerm. Siste del av prosjektet gikk til å pusse opp kode, og drive med enhetstester.

2. Brukerdokumentasjon

2.1 Applikasjonen

Applikasjonen er en kombinasjon av en vær- og ekstremsportapplikasjon. Den gir brukeren oversikt over stasjoner som tilbyr lignende ekstremsporter, samt informasjon om værforholdene på disse stasjonene. Applikasjonen fungerer som et verktøy for å få en oversikt og en anbefaling om det er trygt å gjennomføre ekstremsporten med de rådende værforholdene på stedet.

2.2 Målgruppe

Det er ingen definert målgruppe for applikasjonen, men den vil trolig appellere mest til de som er interessert i ekstremsport og er villige til å bruke en slik applikasjon til slike formål. De valgte ekstremsportene inkluderer fallskjermhopping, men kan også utvides til bungeejumping, basehopp og paragliding. Applikasjonen er rettet mot de som driver med slike sporter. Imidlertid kan også andre personer som er nysgjerrige på disse sportene eller som vil bruke den som underholdning og testing også være en potensiell målgruppe.

Applikasjonen er i utgangspunktet en skydivingapplikasjon, og vil primært passe for fallskjermentusiaster. Fallskjermhopping er en luftsport der utøveren hopper ut fra store høyder, ofte fra fly, svever i fritt fall, utløser en fallskjerm og lander på bakken (Askheim, 2021).

Ifølge USPA (United States Parachute Association), skjedde det rundt 3.57 millioner fallskjermhopp i 2021 og det anerkjennes at flest folk hopper bare en eller to ganger gjennom livet sitt (USPA, n.d.). Applikasjonen faller inn i en nisjekategori, og ville sannsynligvis bare bli brukt av folk som hopper aktivt eller jobber innenfor fallskjermhopping. Brukere kan da

ha muligheten til å evaluere sikkerheten rundt et hopp, uten å måtte bruke mange forskjellige værtjenester.

2.3 Funksjonaliteter

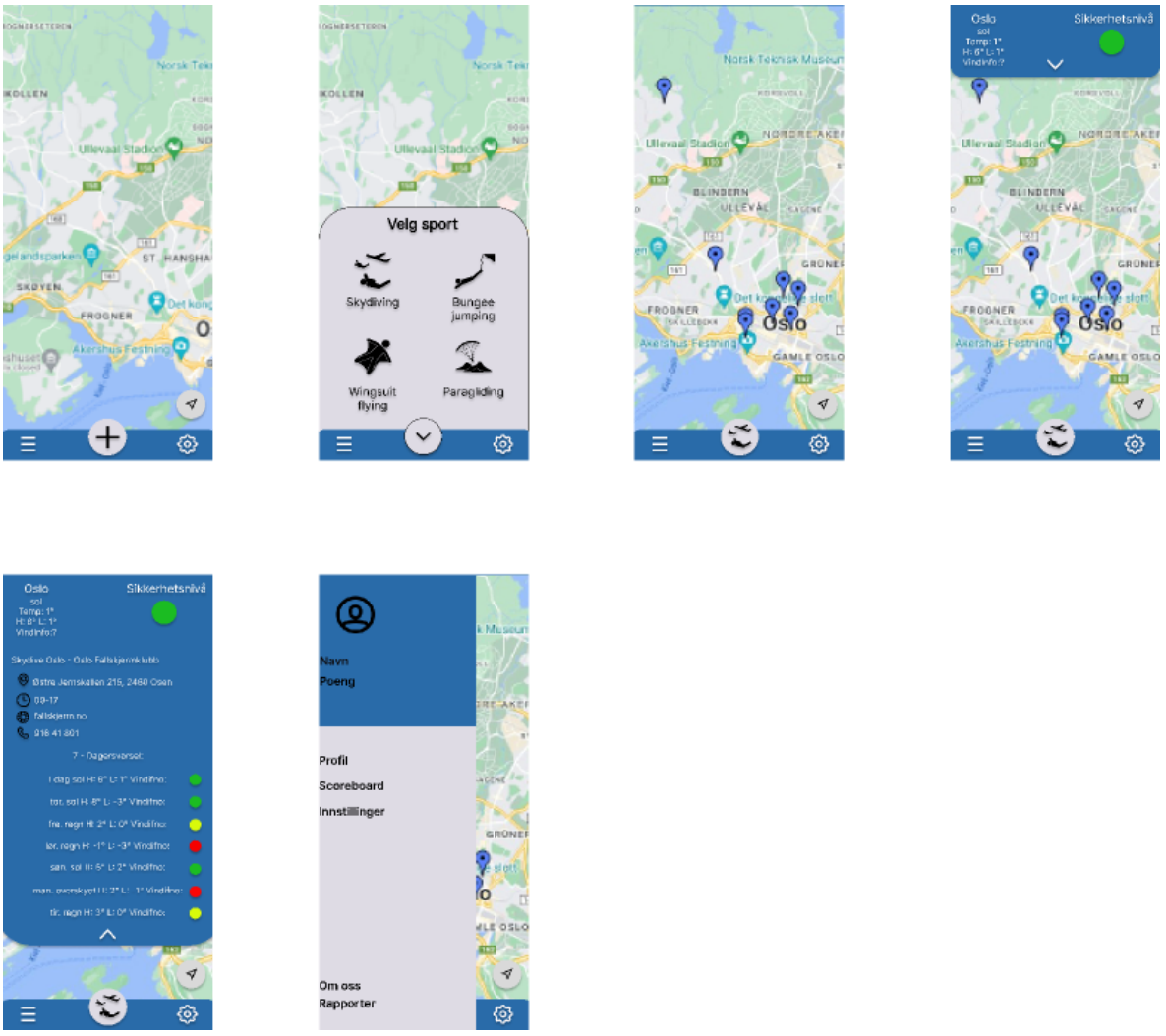
Applikasjonens funksjonaliteter vil være relevant for formålet på en brukervennlig måte slik at den er enkel å bruke for de fleste.

Funksjonalitetene vil omfatte muligheten til å velge en ekstremsportstasjon og få informasjon om værforhold, stasjonsinformasjon og anbefalinger fra oss om det er trygt å utføre et hopp. For hver av stasjonene er det en markør på kartet som kan trykkes på for å vise denne informasjonen. Applikasjonen vil da hente værdata gjennom APIer fra Meteorologisk instituttet og bruke det til å fortelle om værforhold samt et trafikklys som sier noe om anbefaling (grønn, gul, rød). Dette er inspirert av det åpne caset "Trafikklys for flyplasser" fra 2022, hvor flygelederne fikk terskelverdier for værforholdene for å avgjøre om det var trygt å operere flyplassen.

Applikasjonen vil også inneholde flere nyttige funksjoner som for eksempel en Your-location-knapp som gjør at brukeren kan finne sin nåværende posisjon på kartet. Deres nåværende posisjon blir representert med en blå sirkel, og knappen vil føre til at kartet hopper fra hvor det startet til den posisjonen. Videre vil det være mulig å arkivere hoppsteder og gi dem en vurdering og en tekstlig beskrivelse, slik at brukeren kan gå tilbake og se hva de syntes om stedene de har hoppet fra tidligere. Det vil også være mulig å filtrere stasjonene basert på rangering av hjerter.

Vi regner også med å inkludere funksjonaliteter for å kunne endre applikasjonens design til nattmodus, endre språk og kunne rapportere feil som brukere kan oppleve. Disse funksjonene hjelper å forbedre brukeropplevelsen.

2.4 Struktur og design



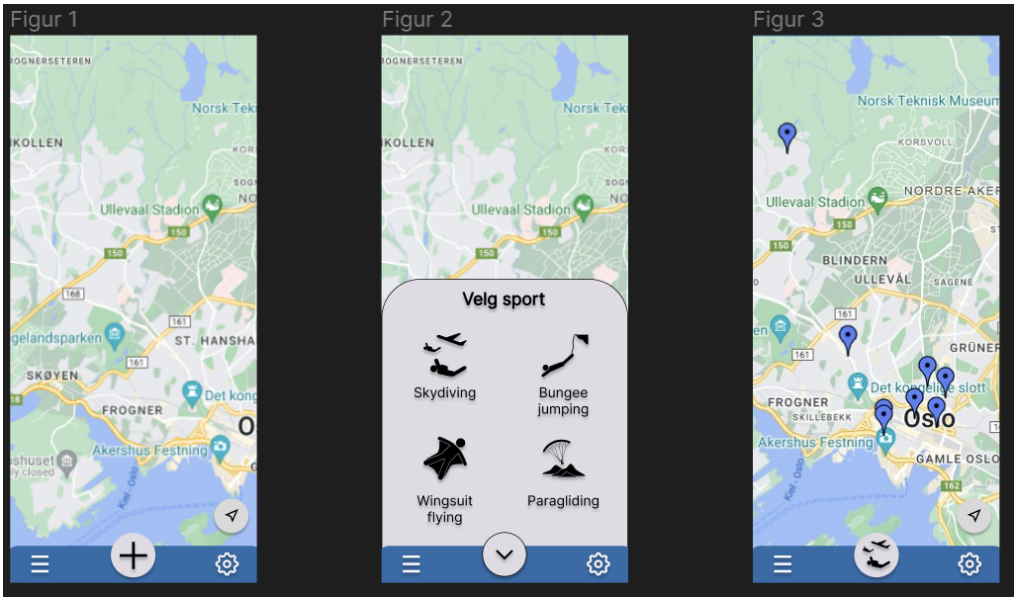
(Figma design av applikasjonen)

I designprosessen ønsket vi å skape et dynamisk og intuitivt design for å gjøre appen brukervennlig for alle Dette inkluderte å ha tydelige ikoner og knapper for flytende navigering i applikasjonen. For at design skal være mer intuitivt, har vi valgt å ha med en bottomNavigation-bar hvor de viktigste funksjonene er lett tilgjengelig for brukerne.

Vi har også valgt at appen vår skal ha et minimalt og konsistent design, der målet er at appen skal være visuelt behagelig. For at appen skal følge dette, har vi valgt at alle skriftene har samme skrifttype og ikonene skal følge samme stil, for eksempel: runde kanter istedenfor spisse. For å unngå for mange farger, har vi valgt 2 hovedfarger for appen: lyseblå og lysegrå. Blåfargen er inspirert av himmelen, og siden appen er basert på sporter som utføres i lufta som skydiving, virker det naturlig å implementere assosierte virkemidler. Den andre

hovedfargen er lysegrå, der hensikten er å gjøre det lettere for brukere å skille fargen hvis man havner i lyse områder på kartet.

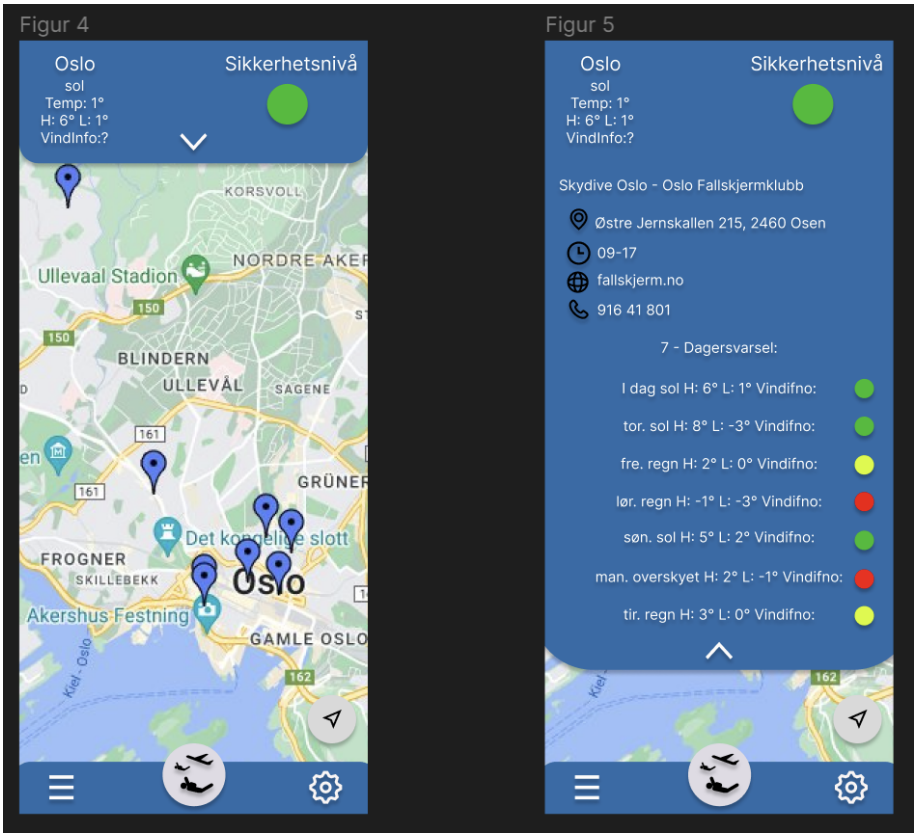
2.4.1 Velge sport



(Figma skisse av applikasjonen)

For å skissere applikasjonen brukte vi verktøyet Figma. Figuren over viser essensen av hovedsiden og deres funksjoner. Designet består av en hovedside med et kart implementert, og en bottom bar som inneholder de viktigste funksjonene. Vi ønsker å gi brukeren muligheten til å velge en sport gjennom en knapp på bottom baren som gir en popup-boks med de ulike ekstremsport-alternativene. Når en ekstremsport har blitt valgt så vil markører på kartet oppdatere seg til å vise alle stasjoner i Norge som har de ekstremhopp mulighetene. Herfra kan brukeren finne sin nærmeste stasjon å besøke. I figmaen har vi brukt en rekke kilder for å legge til ikoner og kartet (Agrawal, 2014; Evericons, 2019; Google, Ukjent dato; Gupta, 2013; Lay, 2016, 2017).

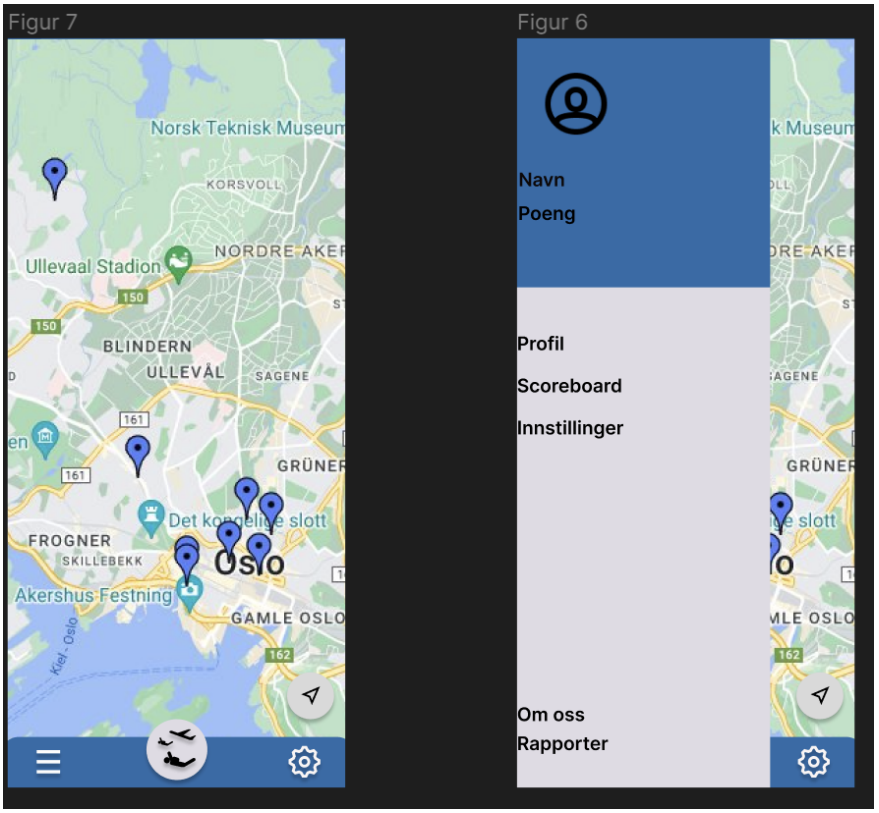
2.4.2 Informasjon om stedet



(Figma skisse av applikasjonen)

Vi ønsker også å gi brukeren enkel tilgang til vær- og stasjonsinformasjon om den stasjonen de har valgt på kartet gjennom en popup-boks på toppen av hovedsiden. I popup-boksen vil sikkerhetsnivået til å utføre ekstremhopp bli visualisert, som er basert på værdata som vi får fra vær-APIene. Popup-boksen inkluderer også en knapp for å utvide boksen for å få mer detaljert informasjon om stasjonen, og været for de neste 7 dagene. Å kunne se fram 7 dager åpner opp muligheten for å planlegge hopp lenger fram i tid. Designet til informasjon om stasjonen har vi tatt inspirasjon fra Google Maps (Google, Ukjent Dato).

2.4.3 Navigation drawer



(Figma skisse av applikasjonen)

På hovedsiden sin bottom bar befinner det seg også en hamburger meny som drar ut enn navigationDrawer. Draweren vil dekke halvparten av skjermen med en header som viser profilen til brukeren med antall hopp de har gjennomført, og en rekke knapper som tar de med til ulike skjermer. De ulike skjermene innebærer profiler, arkiv, scoreboard, innstillinger, om oss og rapportering.

Profil vil ta brukeren til designerte sider hvor de kan få informasjon om seg selv, og antall hopp de har gjennomført i tillegg til ekstra informasjon om dem. Scoreboard inneholder tabell over alle brukere som har utført hopp, og hvem som leder i antall. Innstillinger inneholder egenskaper som kan endres av applikasjonen. Om Oss har informasjon om utviklerne av appen, og rapporter gir brukeren muligheten til å sende inn klager eller feil som de oppdager under bruk av appen.

Navigation draweren kan lukkes ved å trykke på kartet eller dra draweren til venstre. En tilleggsfunksjon som vi valgte å ha for kartet er en Your-location-knapp som tar brukeren til

nåværende posisjon, hvor kartet vil sentreres ut fra brukerens posisjon. Knappen vil befinne seg under knappene for å forstørre eller minke kartet.

2.5 Plattform og aksessering

Applikasjonen er utviklet med Kotlin og Jetpack Compose, som er verktøy for å bygge applikasjoner på Android-plattformen. For å kunne kjøre applikasjonen, så må Android Studio tas i bruk sammen med en emulator. I Android Studio kan man lage sin egen emulator med størrelsen til en telefon eller et nettbrett, og et valgt API-nivå til emulatoren.

Vi anbefaler å bruke en Google Pixel 5 med API-nivå 33 for å få den beste brukeropplevelsen på grunn av at vi utviklet applikasjonen ved å bruke denne som primær test emulator. På slutten av prosjektet ble det testet med andre enheter, og API-nivåer for å få oversikt over hvordan det ville se ut på andre enheter. Pixel 5 API 33 ble brukt fordi det er en ny telefon som ble lansert i 2020, og som vi anså som en gjennomsnittlig plattform for andre Android-enheter. Hvis man eventuelt ikke skal bruke den anbefalte emulatoren, bør man sørge for at emulatoren støtter Google Play Services. Google Play Services består av bakgrunnstjenester og biblioteker for bruk av mobilapper som kjører på enheten (Google, 2023).

Når man laster ned applikasjonen og åpner den vil man få en melding om å endre noe i deploymentTargetDropDown.xml, dette skjer fordi OS-et ditt ikke kjenner igjen det som står på linje 10. Bare trykk OK, også skal det fikse seg på egen hånd.

3. Kravspesifikasjon og modellering

3.1 Krav

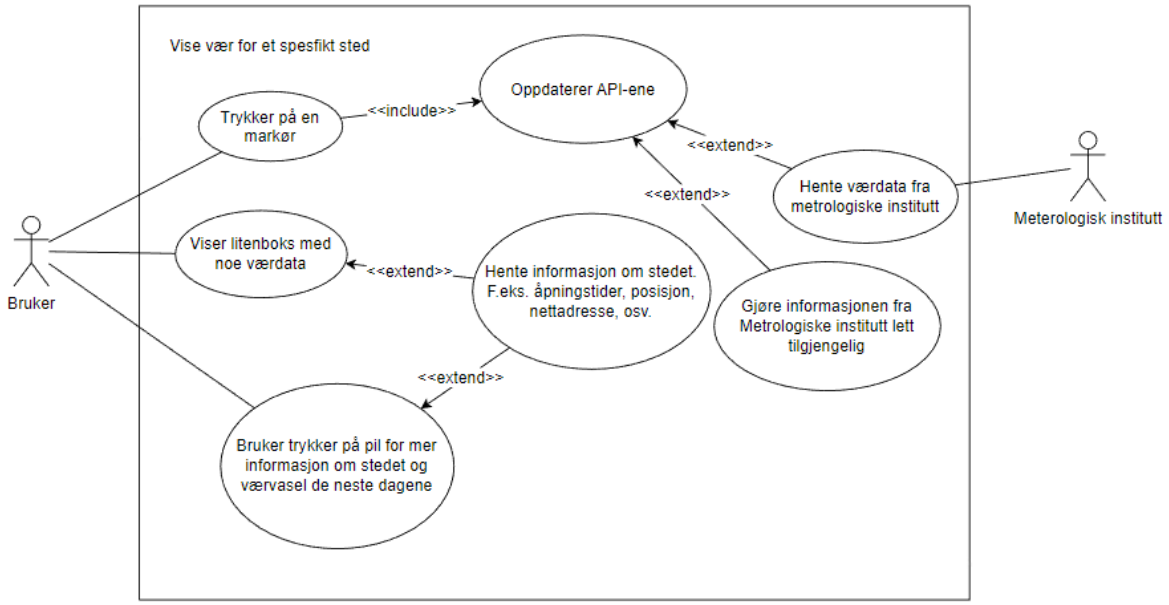
Denne tabellen beskriver de viktigste funksjonelle- og ikke-funksjonelle kravene i applikasjonen.

Må ha med		
	Funksjon som sjekker om det anbefales å utføre sporten	Funksjonelt krav
	Få et svar fra APIene	Funksjonelt krav
	Sette opp navigasjonen mellom skjermene	Funksjonelt krav
	Anbefalinger til steder å skydive	Ikke-funksjonelt krav
	Vise kart	Ikke-funksjonelt krav
	Hente og vise værdata fra APIene	Funksjonelt krav
Burde ha med		
	Ha en loading skjerm	Ikke-funksjonelt krav
	Muligheten til å lagre hopp, og gi de vurderinger	Ikke-funksjonelt krav
Fint å ha med		
	Ha dark theme mulighet til appen	Funksjonelt kravt
	Ha en egen app logo	Ikke-funksjonelt krav
	Ha mulighet til å rapportere feil i applikasjonen	Ikke-funksjonelt krav
	Implementere en Your-location-knapp	Ikke-funksjonelt krav

3.2 Brukerhistorie

Som bruker kan jeg trykke på en markør og den nyeste værdataen vil bli presentert for dagen og de neste tre dagene for stedet jeg trykket, samt en nettadresse og litt annen info om stedet.

3.3 Use-Case



Mål: Få boks med sted- og værinfo en dag og de neste dagene etter

For å nå målet må brukeren trykker på en markør også trykke på nedtrekkspilen inne i den lille informasjonsboksen.

3.4 Klassediagram

Vedlegg [7.2] : Klassediagram til use-caset

3.5 Sekvensdiagram

Navn: Trykke på markør og få nyeste værdata og se værvarsel for de neste dagene

Primæraktør: Bruker at Jumptime applikasjonen

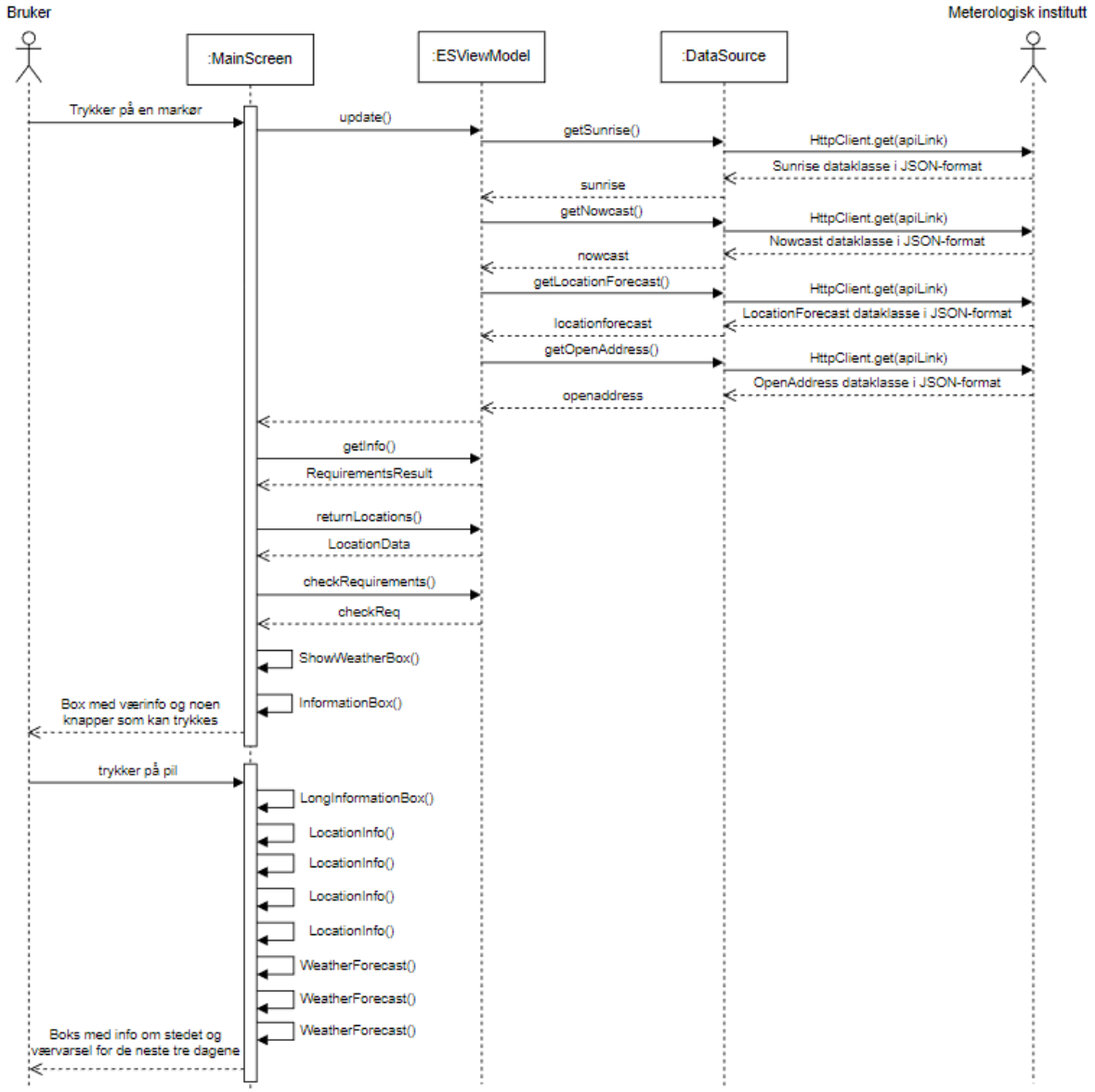
Sekundæraktør: Meteorologisk institutt

Prebetingelse: Brukeren har fått opp kartet med alle markørene

Postbetingelse: Bruker får hele boksen med info om stedet og værvarsel de neste tre dagene

Hovedflyt:

1. Brukeren trykker på en marker
2. Systemet kjører update() som henter alt det nyeste av værdata fra meteorologisk institutt
3. Systemet henter infoen som skal brukes.
4. Systemet henter alle lokasjonene som er på kartet
5. Systemet beregner en verdi som avgjør hvor sikkert det er å hoppe i fallskjerm
6. Viser en liten del av hele boksen med værinfo.
7. Bruker trykker på pil for å se mer info om stedet og værvarsel
8. Systemet henter og bruker den informasjonen den trenger for å vise fullstendig info


Bruker
:MainScreen
:ESViewModel
:DataSource
Meterologisk institutt
Trykker på en markør
update()
getSunrise()
HttpClient.get(apiLink)
Sunrise dataklasse i JSON-format
sunrise
getNowcast()
HttpClient.get(apiLink)
Nowcast dataklasse i JSON-format
nowcast
getLocationForecast()
HttpClient.get(apiLink)
LocationForecast dataklasse i JSON-format
locationforecast
getOpenAddress()
HttpClient.get(apiLink)
OpenAddress dataklasse i JSON-format
openaddress
getInfo()
RequirementsResult
returnLocations()
LocationData
checkRequirements()
checkReq
ShowWeatherBox()
InformationBox()
Box med værinfo og noen knapper som kan trykkes
trykker på pil
LongInformationBox()
LocationInfo()
LocationInfo()
LocationInfo()
LocationInfo()
WeatherForecast()
WeatherForecast()
WeatherForecast()
Boks med info om stedet og værvarsel for de neste tre dagene

3.6 Viktige objektorienterte prinsipper

3.6.1 OOP-prinsipper

Objektorientert programmering er en betegnelse på en programmeringsstil hvor kode organiseres etter en viss struktur med klasser og objekter. Denne strukturen skal representere hvordan vi organiserer virkeligheten rundt oss (Vihovde, 2021). Data og funksjoner som hører sammen er samlet som objekter, og en klasse definerer en datatype og beskriver innhold og egenskaper til objekter av denne datatypen.

De viktigste prinsippene innen objektorientert programmering er for eksempel arv, polymorfisme, innkapsling og abstraksjon. Arv-prinsippet lar en klasse arve egenskaper og metoder fra en annen klasse, hvor de nye klassene om arver kalles for en underklasse eller barneklasse, mens de eksisterende klassene kalles for foreldreklasser. Arv gir mulighet for gjenbruk av kode og hierarkisk strukturering av klasser. Polymorfisme gjør det mulig for objekter av forskjellige klasser å bli behandlet forskjellig avhengig av typen objekt metoden opererer på. Dette tillater til mer fleksibilitet og abstraksjon i designet. Abstraksjon-prinsippet handler om å forenkle komplekse systemet ved å identifisere essensielle egenskaper og ignorerer unødvendige detaljer. Innkapsling handler om å skjule intern implementasjon og tilstand av et objekt.

Disse prinsippene utgjør grunnlaget for OOP og gir retningslinjer for å skape godt strukturerte og fleksible programmer. Noen av disse prinsippene bidrar også til flere OOP-prinsipper som lav kobling og høy kohesjon.

Lav kobling står for avhengigheten mellom moduler og klasser innen et program. Lav kobling vil da tilsi at det er lite avhengighet mellom klassene og da mye enklere å gjøre endringer i koden uten at det påvirker resten av systemet. Høy kohesjon viser til graden av elementer som henger sammen og har en felles funksjon i en klasse eller modul. Desto nærmere dataene og metodene er, jo enklere er det å forstå og vedlikeholde koden. En modul eller klasse med lav kohesjon derimot, vil ha elementer som er spredt rundt og som fokuserer på flere forskjellige oppgaver som ikke nødvendigvis henger sammen. Dette kan føre til at koden blir mer kompleks, vanskeligere å forstå og vanskeligere å vedlikeholde. Både lav kobling og høy kohesjon fører til å bidra med robuste og vedlikeholdbare systemer. Lav

kobling og høy kohesjon er ikke eksplisitte OOP-prinsipper i seg selv, men de er begreper som er tett knyttet til prinsippene og målene innenfor objektorientert programmering.

3.6.2 MVVM

MVVM (Model–view–viewmodel) er en design-arkitektur som fungerer bra med OOP (Objektorientert-programmering). MVVM er et viktig konsept som jobber med å konvertere dataobjekter fra modellen til å administreres og presenteres som et User Interface(UI). Model-delen refererer til brukbar data, hvor den holder på dataen. View-delen viser hva brukeren ser med den formaterte dataen. Viewmodel-delen refererer til koblingen mellom Model og View, hvor den henter data fra Model og viser det frem i View (*Introduction to Model View ViewModel (MVVM)*, 2022). MVVM og OOP henger ofte sammen på grunn av at MVVM bruker OOP-prinsipper til å organisere og samle kode som lav kobling og høy kohesjon. I MVVM-arkitekturen har visningsmodellen for eksempel lav kobling til visningen og modellen, som tilsier at den ikke er direkte avhengig av hverken visningen eller modellen. Høy kohesjon i MVVM-arkitekturen betyr for eksempel at all data er samlet i modellen som er mer naturlig.

3.6.3 Hvordan vi bruker OOP-prinsipper og MVVM

OOP-prinsippene vi tar i bruk er for eksempel ideen om abstraksjon. Det er brukt i visse skjermer som kortene i ArkivScreen. og innstillingene i SettingsScreen. De er composable metoder for kort- og instillingsobjekter, og ved å abstrahere disse gjør vi det enklere for å utvide skjermene senere hvis vi har lyst på flere innstillinger.

Vår app har i hovedsak lav kobling mellom komponentene. Dette kommer fra at hver skjerm er isolert fra resten av koden, bortsett fra Viewmodel-en, og designet til skjermene (utenom hovedskjermen). Dette gjør det enklere å redigere individuelle skjermer uten å påvirke resten av koden, med mindre det er relatert til design. For å ytterligere redusere koblingen kunne vi ha opprettet en overordnet funksjon som definerer det felles designet for skjermene, og deretter brukt denne funksjonen for skjermer som deler samme design.

Appen vår har hovedsakelig høy kohesjon. Dataklassene som brukes av APIene er organisert slik at hver API har sin egen fil, og disse filene er gruppert i en pakke. For eksempel er filene SportRequirements, RequirementsResults og LocationData plassert i samme pakke. Dette bidrar til en viss grad av kohesjon. Imidlertid kunne man vurdert å flytte filer som er mindre relatert til APIene til en separat pakke, eller å flytte de API-relaterte filene til en dedikert pakke. Bortsett fra dette har skjermene generelt høy kohesjon, der kun koden som er relatert til hver enkelt skjerm finnes i skjerm filene.

I MVVM strukturen vår er det Datasource som kan regnes som Model delen fra MVVM. Datasource henter data fra APIene Nowcast, Locationforecast, Sunrise, og Åpent Adresse. Viewmodel-en vår sørger for at APIene er tilgjengelige for resten av app. Den utnytter også APIene ved å gjøre en avgjørelse på om det er trygt å hoppe fra et spesifikt sted. I tillegg prosesserer den informasjonen fra APIene til et lettere format å bruke av View delen. View er bygd opp av en rekke skjermer som kan navigeres mellom. Informasjonen som Viewmodel prosesserer vises i MainScreen i en boks.

4. Produktdokumentasjon

4.1 Teknologier og arkitektur som brukes i løsningen

4.1.1 Arkitektur og teknologi

Når det kommer frem til teknologier og arkitektur i prosjektet, så krever faget at vi bruker kotlin som programmeringsspråk. Kotlin er et høynivåspråk som ligner og er designet til å fungere fullt ut med Java. Vi bruker Kotlin sammen med Jetpack Compose som er Androids anbefalte bibliotek for å bygge brukergrensesnitt. Den gjør applikasjonen vår levende med mindre kode, gode verktøy og intuitive Kotlin APIer. Editoren som ble brukt var Android Studio, som er et integrert IDE brukt i Android-utvikling som kommer med sin egne android emulator. Vi jobbet med Android Studio annerledes enn fjorårets IN2000 studenter ved å implementere Kotlin-kode funksjonaliteter i stedet for å jobbe alene med XML-filer, som er et markup language og filformat for lagring, overføring og rekonstruering av data.

På grunn av at Kotlin og Jetpack Compose var et krav, så ble hele prosjektet utført med det, men samme prosjekt kunne ha blitt gjennomført i XML eller et annet programmeringsspråk. Da kunne man også ha brukt en annen editor som for eksempel Visual Studio Code eller Atom, men det å emulere applikasjonen blir vanskeligere enn å ha den innebygd som for eksempel Android Studio.

Jetpack compose versjonen som ble brukt var Material 3, som er et bibliotek som lar deg bygge Jetpack Compose UI. Material 3 er den nyeste Jetpack compose versjonen, og har bare nylig begynt å bli stabil til bruk, og er designet til å være på linje med den nyeste Android-stilen. Eneste grunnen til at vi valgte Material 3 var på grunn av at de tidligere obligatoriske innleveringene i IN2000 krevde Material 3. Det var ingen grunn til å endre stilen på prosjektet vårt, og vi ville lage en applikasjon som var datert med andre android applikasjoner

4.1.2 API-nivå

Et API-nivå er et tall som er tilordnet en spesifikk versjon av Android-plattformen som identifiserer hvilken versjon av SDK (software development kit), og hvilket sett med egenskaper og funksjoner som er tilgjengelige for utviklere å bruke når de bygger apper for den versjonen av Android. Hver Android-versjon har et spesifikt API-nivå knyttet til seg, og dette API-nivået representerer et unikt rammeverk som tilbys av den versjonen av plattformen. Når utviklere bygger en Android-app, kan de spesifisere hvilket API-nivå appen deres er rettet mot. Dette lar appen bruke spesifikke egenskaper og funksjoner som er tilgjengelige på det API-nivået, samtidig som den opprettholder kompatibilitet med eldre versjoner av Android.

MinSDK er den tidligste utgivelsen av Android SDK som applikasjonen kan kjøre på, og targetSDK er versjonen applikasjonen er målrettet mot. TargetSDK beskriver da det optimale rammeverket for kjøreforhold. Dette er mest for å indikere hvor aktuell applikasjonen din er for bruk på markedsplassen.

Vi valgte å bruke API-nivå 33, og emulerte det hovedsakelig på en Google Pixel 5 telefon. Dette var det høyeste nivå vi kunne velge, og gjør at alle tidligere API-nivå emulatorer også

kan kjøre applikasjonen. Det betyr ikke nødvendigvis at applikasjonen vil kjøre optimalt på alle telefoner med lavere API-nivå, men at det er i teorien mulig. Telefonene vi testet på var for eksempel Google Pixel 2 og Google Pixel C med forskjellige APIer.

4.2 Viktigste kvalitetsegenskapene og begrunnelser

4.2.1 Brukerkvalitet

Når det kommer til kvalitetsegenskapene ved en applikasjon, så er brukervennlighet en av de viktigste. Applikasjonen er ment å være et enkelt verktøy for ekstremsportsentusiaster å bruke, og vi regnet med at de ikke trenger avanserte funksjoner for å få data om værforhold til å utføre hopp. Når applikasjonen startes blir brukeren møtt med hovedsiden, hvor de kan finne stasjoner for å utføre hopp med en gang. De røde markørene på kartet er tydelige, og værboksen tar plass med en gang en markør har blitt valgt. I værboksen får brukeren vite med en gang om det er mulig å gjennomføre hoppet gjennom det fargede ikonet, hvilken stasjon det er snakk om og de viktigste værvariablene. Dette er et simpelt design som virker intuitivt for de fleste vi har utført en brukertest på, og som vi har regnet med som en suksess.

Alle andre funksjoner som å kunne rapportere feil eller komme seg til ulike skjermer i applikasjonen har også blitt designet på en enkel måte. Alle knappene for å komme seg til skjermene befinner seg i enten bottom baren eller i navigationDraweren, og er tilstede i hvilken som helst skjerm brukeren kan befinne seg i. Dette gir en generell struktur, og gjør at brukeren slipper å gå frem og tilbake for å utføre flere funksjoner. Denne ideen kom også fra en brukertest som syntes at det var tungvint å gå tilbake til hovedsiden for å aksessere andre skjermer.

I løpet av prosjektet benyttet vi også en brukerundersøkelse om hvilke funksjoner og værforhold aktive fallskjermhoppere trenger for å bruke en skydiving-værforhold app. Ut ifra resultatene vi samlet gjennom brukerundersøkelsen var det en rekke funksjoner som ble etterspurt, som vi også evaluerte til å være gode å implementere. I ett av spørsmålene skulle deltakerne gi ideer for hva de kunne likt å se i applikasjonen, og en stor andel refererte til funksjonene i Windy. Windy er en interaktiv værvarslingstjeneste som viser data for hele verden, og funksjonene de fleste etterspurte var muligheten til å kunne spå muligheten til å

hoppe 2-3 dager frem i tid, se vindretning og ha tydelige fonter. Mange av de andre svarene hadde vi allerede implementert, men brukerundersøkelsen var nyttig for å øke kvaliteten til brukeropplevelsen på applikasjonen.

Brukervennligheten er applikasjonens hovedgrunn til suksess på grunn av at det gir en positiv opplevelse for brukerne. Når appen er enkel å bruke, intuitiv og gir en smidig navigasjon, vil brukerne føle seg fornøyde og tilfredsstilt. Dette kan føre til økt brukerengasjement, positiv omtale og gjentatte besøk, noe som bidrar til appens suksess. Igjen så fører dette også til en høyere sjanse for at brukere velger vår app fremfor konkurrentene i et marked.

4.2.2 Pålitelighet og tilgjengelighet

For en god brukeropplevelse er påliteligheten og tilgjengeligheten til applikasjonen viktig. Applikasjoner og programmer som krasjer gir ikke gode tilbakemeldinger fra brukere. Fraværet av feil i applikasjonen er en av kvalitetsegenskapene vi har. Generelt har vi få warnings, appen krasjer ikke og minne-stacken fylles ikke. Det eneste som gjør at systemet ikke er fullstendig robust er på grunn av at APIene som står for å vise værdata kan være kilden til ustabilitet, hvor de ikke gir data om værforhold dager senere. Vi har opplevd at når vi kjører applikasjonen på en emulator, så kan vi miste en av dagene på tredagersværvarselet.

Det er alltid den samme dagen og trolig skjer det fordi vi bruker en let-funksjon som blir null når den sjekker om APIet har en verdi for den dagen. Det er sannsynligvis løsninger for å fikse dette, men problemet ble oppdaget relativt sent og mangel på tid førte til at det ikke ble løst.

4.2.3 Brukerundersøkelsen

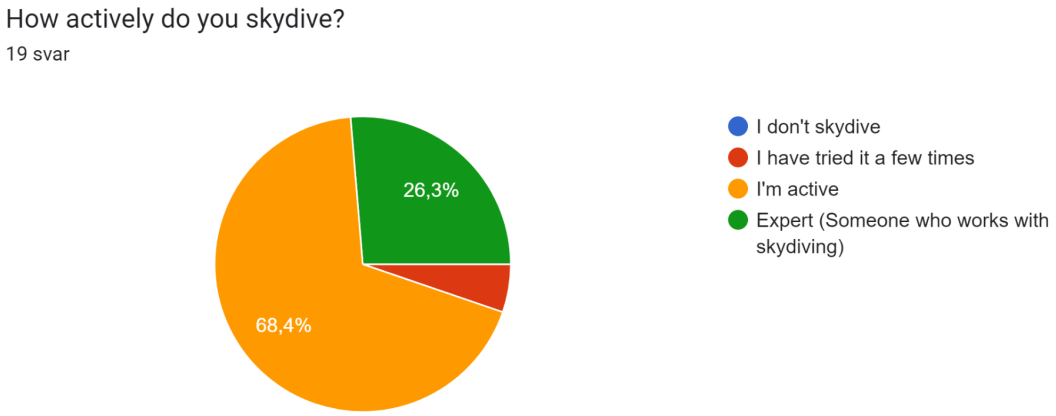
Brukerundersøkelsesverktøyet som ble brukt var Google skjema med spørsmål om hvor aktivt brukerne hopper fallskjerm, hvor mye været påvirker deres hopp, optimale værforhold og funksjoner de ønsker som tradisjonelle værapper mangler. Brukerundersøkelsen var anonym, og alle som hadde tilgang til linkaddressen kunne svare på spørsmålene. Dette ble lagt ut på Reddits skydiving-forum (*Brukerundersøkelsen er i vedlegg*).

Vi var fokuserte på at bare skydivere skulle svare på spørsmålet, men det å verifisere hvem som er aktive hoppere er vanskelig gjennom nettet. Dette førte for eksempel til at vi måtte dømme hvilke svar som virket legitime ut ifra hva vi leste om fallskjermhopping fra nettet og sammenligne det med resten av svarene.

I helhet så fikk reddit posten 4800 visninger, 11 kommentarer og 19 som deltok i selve schema-undersøkelsen. Posten ble godt tatt imot av nettsamfunnet utenom et par kommentarer som mente at vi ikke burde basere applikasjonens værterskler for hopp gjennom en brukerundersøkelse fra Reddit. Problemet med dette er at det ikke er lovgitte regler for hvilke værforhold som må oppfylles for å gjøre hopp. Dette er heller regulert av stasjonens egne fallskjermhoppere og eksperter. Likevel så var responsen god.

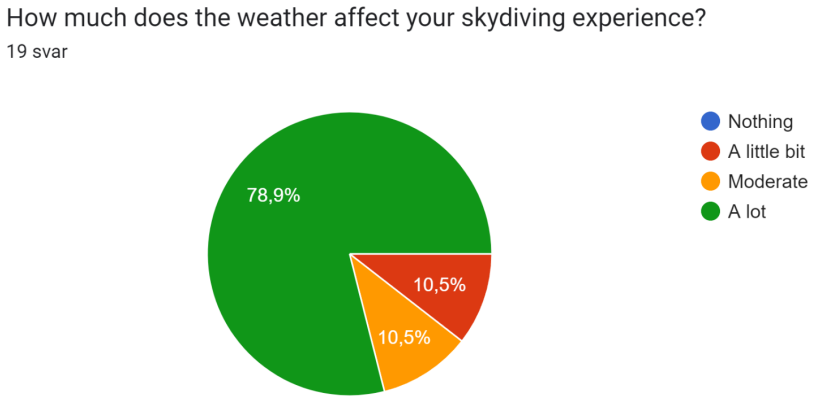
4.2.4 Resultat av brukerundersøkelse

19 fra Reddit-posten deltok i brukerundersøkelsen, og mange av tilbakemeldingene beskrev de samme værforholdene som de andre. I det første spørsmålet om deltakerne er aktive fallskjermhoppere, så mente de fleste at de var aktive som gjør at deres tilbakemeldinger er nyttige for våre formål.



(*Resultat fra første spørsmål i brukerundersøkelsen*)

Videre i brukerundersøkelsen får brukerne spørsmål om hvor mye været påvirker deres fallskjermhopp egenskaper, hvor de fleste mener at det er ekstremt viktig. Dette var vi allerede kjent med, men det som overrasket oss var at 10% av deltakerne syntes det var mer insignifikant. Dette spørsmålet legger opp for de neste tilbakemeldingene vi ville ha.



(Resultat fra andre spørsmål i brukerundersøkelsen)

Neste spørsmål prøver å få svar på optimale og dårlige forhold for å utføre fallskjermhopp. Her fikk vi mange forskjellige svar, med noen felles værforhold som for eksempel at det bør være en klar himmel, vindstyrke under 6 m/s, ingen nedbør, under 3 m/s vindkast, UV-indeks lavere enn 4 og temperatur mellom 10 og 35 celsius.

Værforholdene som brukerne mente var dårlige og man burde unngå hopp på innebærte lave skyer eller tåke, temperaturer over 35- eller mindre enn 5 celsius, vindstyrke over 11 m/s, nedbør over 0,1 mm/t, sjanse for torden, vindkast over 6 m/s og UV-indeks 7 eller høyere.

Moderate forhold betydde alt mellom optimale og dårlige forhold, men bare et aspekt av været kunne føre til at hoppet ble umulig. Brukerne var veldig strenge på det med klar sikt ned til bakken, og at nedbør og torden skaper alt for uforutsigbare forhold. Disse tilbakemeldingene stemte overens med informasjonen vi fant, men i bunn og grunn så er det vanskelig å gi et bestemt svar om en person kan utføre hoppet fordi alle hoppere evaluerer det forskjellige. Terskelverdiene vi tar inn fra brukerundersøkelsen er gjennomsnittet av alle svarene slik at vi kan lage en applikasjon som passer overens med de fleste fallskjermhopperne.

Fjerde spørsmål i brukerundersøkelsen ville finne ut hvilke funksjoner aktive skydivere kunne tenkt seg på en applikasjon som vår. Mange var allerede vant til å bruke værapplikasjoner som for eksempel Windy, men funksjoner som ble nevnt var for eksempel muligheten til å lagre hopp som ble utført, muligheten til å kunne spå været 2-3 dager frem i

tid, se vindretning og skydekning. Noen ville også ha lette brukergrensesnitt og leselige fonter slik at applikasjonen var lett å bruke.

Femte spørsmål handlet om å nevne unntak vi som utviklere burde ta i betraktning når vi lager en skydiving-værforholdapplikasjon hvor vi fikk noen svar. Mye var allerede nevnt i fjerde spørsmål, men noen ville ha muligheten til å kunne bruke applikasjonen over alt i verden.

Siste spørsmål prøvde å finne ut hva fallskjermhoppere mislikte med applikasjoner som Windy, og noen mente at den var komplisert å bruke for spesifikt ekstremsport. Tilbakemeldingene mente også at Windy sine antakelser om været var ofte feil, som kan ha noe å gjøre med den har flere upresise datakilder for å dekke hele verden. Applikasjoner som konsentrerer seg på et land kan ha fordeler som at de har mer lokaliserte og pålitelige datakilder.

I helhet fikk vi gode tilbakemeldinger på undersøkelsen, og vi er fornøyd med å vite at de fleste svarene er fra erfarne hoppere. Mye av dataen vi tok inn fra undersøkelsen vil også være brukbart, spesielt terskelverdiene vi kan bruke for å evaluere tryggheten til å utføre ekstremhopp ut ifra været. Vi forstod at det ikke finnes en fasit for å utføre hopp, men det å kunne få oversikt over erfarne hoppere sine grenser kunne vi lage gjennomsnittlige verdier de hopper på.

4.2.5 Tilbakemelding fra brukertester

Det ble utført noen brukertester gjennom prosessen, hvor det ble gitt ulike tilbakemeldinger. Eksempelvis ble det foreslått å legge bottomNavigationbaren til hovedskjermen på alle skjermene vi har i applikasjonen. Dette var en fornuftig endring, og ble tatt med videre i utviklingen.

En annen tilbakemelding sa at det var ønskelig at når innstillingsskjermen ble navigert, at det skulle være mulig å trykke på innstillingsknappen igjen for å komme tilbake til skjermen der kartet er.

4.3 API

4.3.1 Nowcast

Vi har valgt å ta i bruk tre forskjellige API fra Meteorologisk Institutt. På grunn av at vi utvikler en ekstremsport-værapplikasjon, vil APIene hovedsakelig bestå av vær-APIer. Det første APIet er *Nowcast*. Fra Nowcast henter vi diverse værdata som er målt fra et spesifikt sted. Nowcast har bare data på hvordan været er akkurat nå, men ikke noe informasjon om værforhold fremover i tid. Dataen vi tar i bruk er for eksempel temperatur, vindstyrke, vindkast, nedbør og UV-indeks.

Locationforecast APIet, som vi også har brukt, har de samme dataene som Nowcast, og mer. Vi har allikevel valgt å ta med Nowcast fordi APIet er mye lettere å bruke. Det vil være lettere for en person som ikke er kjent med de forskjellige APIene å ta i bruk Nowcast hvis de skal bruke det et sted i koden.

4.3.2 Locationforecast

Det andre APIet vi bruker er *Locationforecast*. Vi henter også mye værinformasjon herfra, men det som skiller Locationforecast fra Nowcast er at Locationforecast også har data for de neste 12 timene. Dette gir muligheten til å forutsi om det er mulig å gjennomføre ekstremsport dagen før. I tillegg gir APIet informasjon om hvor mye det antakeligvis kommer til å regne, og dette er også relevant til hvorvidt man kan drive med ekstremsport eller ikke.

Til gjengjeld er Locationforecast vanskeligere å bruke enn Nowcast. Locationforecast har mange flere underklasser man må navigere seg igjennom for å ta i bruk dataene vi trenger, og dermed kan det være vanskeligere å finne dataene man trenger.

4.3.3 Sunrise

Det tredje APIet fra Meteorologisk institutt vi bruker er Sunrise. Sunrise inneholder mye informasjon om solen og månen, men det eneste vi bruker herfra er tidspunkt for soloppgang og solnedgang. Dette trenger vi fordi man ikke kan drive på med f.eks. fallskjermhopping når det er mørkt.

4.3.4 Åpent Adresse

Det første av APIene vi bruker som ikke er fra Meteorologisk Institutt er Åpent Adresse-APIet. Dette APIet kommer fra Kartverket, og har mange forskjellige funksjoner relatert til stedsnavn. Funksjonen vi bruker er muligheten til å gi APIet koordinater, og få navnet på stedet tilbake som resultat. Dermed kan vi vise navnet på stedet hvor brukeren sjekker været. Dette lar brukeren være sikker på at de har valgt riktig sted.

4.3.5 Google Maps

Vi valgte å implementere et interaktivt kart i appen slik at brukeren kan ha oversikt over skydivingstasjoner. For at kartet skal vises, valgte vi å bruke Google Maps SDK sitt API (Google, 2023). APIet gir oss tilgang til kartdata fra Google Maps, i tillegg med en rekke andre funksjonaliteter. For å få tilgang til Google Maps APIet måtte vi gå gjennom en registreringsprosess for å få en API-nøkkel, og nøkkelen tas i bruk ved å legge det inn i AndroidManifest-filen slik at alle får tilgang til å se kartet (Google, 2023).

En av grunnene til at vi valgte APIet er fordi Google Maps SDK støtter Android, og har en Maps Compose biblioteket for Jetpack Compose som gjør det lettere for oss å integrere et kart i appen vår. Biblioteket tilbyr compose-funksjoner som vi kan bruke for å tilpasse kartet etter vårt behov. En av de viktigste funksjonene vi bruker er: GoogleMap og Marker. GoogleMap tar inn flere parametere som vi bruker for å bestemme størrelse, kamera posisjon og kart stiler. GoogleMap har også en boks vi benytter for å kalle funksjonen Marker. Marker oppretter en visuell markør på kartet, den tar også imot parameteren “state” som vi bruker for å oppgi koordinater (LatLng) for markøren. LatLng brukes til å sette posisjonen til hver markør på kartet. I tillegg bruker vi “onClick” parameteren for å gjøre markøren klikkbar, slik at informasjonen om stedet vises på skjermen når vi trykker på den.

4.3.6 Hvordan vi bruker APIene

For å hente data fra APIene bruker applikasjonen Ktor-bibliotek. Ktor muliggjør asynkrone API-kall og datahenting. Det er fordelaktig å gjøre dette asynkront, slik at resten av programmet ikke trenger å vente på API-resultatene.

Appen bruker Ktor versjon 2.2.3, som var den nyeste tilgjengelige versjonen da Ktor ble integrert i prosjektet. Siden den gang har det ikke vært noen grunner til å oppdatere til en nyere versjon. Andre biblioteker, som for eksempel Retrofit, kan også utføre disse oppgavene, men Ktor ble valgt fordi det allerede hadde blitt brukt tidligere i faget.

For å deserialisere JSON-dataene fra APIene til brukbare objekter i koden bruker applikasjonen Gson-biblioteket. Gson lar deg konvertere ren JSON hentet fra APIer til dataklasser som kan brukes i programmet.

Gson er et bibliotek som kan importeres gjennom Ktor, og dermed er versjonen som brukes også 2.2.3. Det finnes også andre biblioteker som kan utføre disse oppgavene og som også er inkludert med Ktor, for eksempel kotlinx.serialization og Jackson. Vi har likevel valgt å bruke Gson, siden dette biblioteket allerede hadde blitt brukt tidligere i faget.

4.3.7 Problemer med APIene

Et tidlig problem som vi møtte på med Google Maps APIet var kompatibilitet, ved at API nivået til emulatoren burde være på 30 eller 33 for at kartet skulle vises på skjermen. Feilmeldingen som dukker opp uten det er: “ExtremeSport won’t run without Google Play services, which are not supported by devices".

Det kan føre til potensielle utfordringer for brukere med eldre versjoner av operativsystemet, der kartet ikke vises frem i appen. Dette kan også være et problem for brukere som ikke har muligheten til å oppdatere operativsystemet på enhetene sine.

Et annet problem som oppsto relativt sent i utviklingsprosessen var at eksempel-dataen til Locationforecast ikke samsvarte med dataen man fikk ut av APIet. Dette resulterte i at deler av dataen ble forkastet, og noe data vi forventet å ha ikke var tilgjengelig. For å rette opp i dette, var løsningen så enkel som å endre dataklassen slik at den korresponderte med de dataene som faktisk ble levert av Locationforecast.

4.4 Videreutvikling og drift

I drift og videreutvikling av applikasjonen ser vi for oss at det fortsettes på funksjonene vi ikke fikk gjennomført. Dette inkluderer for eksempel å fullføre composable skjermer som AkrivScreen, FavorittScreen og Instillinger. Sidene har det fundamentale satt på plass, men mangler de ulike funksjonalitetene som gjør de fullverdige.

Det hadde også vært hensiktsmessig å implementere muligheten til å velge flere ekstremsporter som vi planla. Applikasjonen passer inn i en nisjekategori, og hvis applikasjonen vil treffe flere målgrupper uten å miste sin ekstremsportessens, så må den inkludere flere ekstremsporter. Brukeropplevelsen kunne også forbedres ved å gi brukerene mer frihet gjennom å la dem legge til egne ekstremsporter. Denne funksjonen kunne gitt applikasjonen en større brukerbase, og fremdeles være kjent som en ekstremsport-værforhold applikasjon.

I videreutviklingen så ville en drift også justert på terskelverdiene for å utføre ekstremhopp. Vår data på når det er trygt å utføre ulike hopp er ikke en definitiv fasit på hva som faktisk hadde vært trygt. Realiteten er at det varierer veldig fra sted til sted, og riktige verdier burde være satt av erfarne fallskjermhoppere i sine designerte områder.

APIene vi også tar i bruk viser ikke værforhold for så mange dager frem i tid. En videreutvikling kunne ha vært å legge til den funksjonaliteten. Det finnes APIer som ser mye lengre fram i tid som kan være nyttige i en større applikasjon.

Som nevnt tidligere i 4.3.7 Problemer med APIene, kan det potensielt oppstå problemer for brukere med eldre versjoner av operativsystemet. En mulig løsning for dette kan være å implementere enklere funksjonalitet for eldre enheter. Det kan være hensiktsmessig å i stedet for ha et interaktivt kart med markører for stedene, hvor man kan implementere et grensesnitt som viser en liste over stasjonene som brukeren selv kan trykke på for å få vær og stedsinformasjon.

5. Prosessdokumentasjon

5.1 Utviklingsmetode

I løpet av prosjektoppgaven ble det brukt en smidig utviklingsmetode hvor vi kombinerte gode elementer fra både Scrum og Kanban.

Scrum er et rammeverk for å utvikle og levere produkter ved å bryte ned arbeid til sprinter som skal gjennomføres i løpet av tidsperioder. Det er tre tilstander oppgavene kan være fordelt i: "Må gjøres", "Gjøres", "ferdig". I hver sprint velges det ulike oppgaver fra produkt-backloggen, og hver sprint tabell blir sin egen sprint-backlog. Scrum-teamet består av en produkteier, en scrum master og utviklere, som hver har spesifikke ansvarsområder innen prosjekter. Scrumteamet utfører også korte møter kalt daglige scrums i løpet av sprinten hvor de snakker om hva de har gjort siden siste møtet, og hva de skal gjøre videre. På slutten av sprinten utføres det også et møte hvor teammedlemmene viser frem arbeidet sitt, og et annet møte hvor de evaluerer hva som fungerte og ikke.

Kanban er et annet rammeverk som baserer seg på å vise alle oppgaver som skal utføres, oppgaver som gjøres og hvilke som har blitt gjort. Oppgavene blir flyttet fra de ulike kolonnene ut fra tilstanden. Kanban-metoden innebærer også å begrense mengden arbeid som kan utføres på samme tid. Dette kalles ofte "Work in Progress" (WIP)-grensen. WIP-grensen sikrer at teamet ikke tar på seg for mye arbeid samtidig, og fokuserer på å fullføre oppgavene før de starter på nytt. Kanban-metoden blir kontinuerlig evaluert og justert ut fra behovet i prosjektet. Kanban er ikke like tidsorienterte som Scrum.

5.1.1 Scrumban

Vår utviklingsmetode skulle bestå hovedsakelig av kanban og noen scrum-prinsipper (Scrumban). Rammeverket vil for eksempel bestå av sprinter og scrum-møter, men vi ignorerer roller som product owner og gir friheten til å velge ut flere oppgaver i løpet av en sprint fra kanban-boardet. Vi har heller ikke daglige stand-ups, men heller bare et møte i midten av sprinten for å se hvordan arbeidet til teammedlemmene går. Vi valgte også å ha en roterende scrum master slik at alle i teamet fikk litt ledererfaring i løpet av prosjektet. Vi var

heller ikke noen tilhengere av å fordele forskjellige aspekter av prosjektet til forskjellige medlemmer, men heller at alle kan gi innspill til alle deler.

Sprintene skulle vare i to uker, hvor nye sprinter starter på onsdager hvor vi diskuterer hva som er viktig å gjennomføre til neste sprint. Onsdag uken etter er et "daily standup" hvor teammedlemmene diskuterer hva de har gjort og hva de har planer om å gjøre fremover. Her nevner de for eksempel hva som har blitt ferdig, hva som trengs hjelp med og hvilke nye oppgaver de tok opp. Siste dagen av sprinten var et retrospekt møte hvor teamet diskuterte hva som fungerte bra, og hvilke utfordringer vi bør ta til hensyn til neste sprint. Oppgaver som ikke ble ferdige blir overført til neste sprint. I løpet av oblig 3 i IN2000, så lagde vi også en plan om hva hver sprint skulle handle om med rom til å justere hvis det var nødvendig.

Under finner du planen.

Sprint	Notes
1	Lage design av app
2	Lage MVP
3	Starte testing
4	Lage ferdig App
5	Pusse opp Prosjekt

(Sprintplan for hele prosjektet)

5.1.2 Verktøy

Verktøyene som ble brukt var for eksempel et *Trello*-board for å visualisere tilstanden til alle oppgavene som måtte gjennomføres. Trello er et nettbasert kanban-style liste-program hvor man kan lage et kanban board hvor medlemmer kan dra oppgaver fra product-backloggen og tildele dem til hvert medlem. Tildelingen gjør det lett å ta opp de ulike oppgavene med hvert medlem, og vite hvilke oppgaver som allerede er gjort eller er i"doing". Fordelingen blir også diskutert i sprintmøtet i starten av sprinten slik at alle vet hva de har ansvar for.

Andre verktøy som ble brukt var *Github* for å dele kildekode mellom hverandre. Github er viktig for å kunne gjøre kildekode lett tilgjengelig for alle i gruppen og ha egenskapen til å holde styr på endringer som blir gjort i koden. Det gjør det lett å administrere endringer, gjennomgå kode og samarbeide. Github ga også muligheten til å kunne kopiere koden fra hovedgrenen over til separate grener for å gjøre endringer eller teste uten å påvirke hovedkoden (branching). Gjennom branching kunne hvert medlem jobbe med koden uten å påvirke de andre sin prosess, og til slutt smelte sin gren til hovedkoden ved at en annen i teamet godkjenner endringene på grenen.

For kommunikasjon og møter ble det hovedsakelig brukt *Discord*, som er et kommunikasjonsprogram hvor vi kunne lage en desidert server for prosjektet. I serveren hadde vi muligheten til å lage ulike kanaler for kommunikasjon, chatgrupper for ulike oppgaver og muligheten til å dele skjermen til andre medlemmer under møter. Alle i gruppen var kjente med discord, og var naturligvis et lett valg å ta for å kommunisere med hverandre. I møtene hvor vi startet sprinten, så møttes vi fysisk i et grupperom på ifi bygget,

For rapporten brukte vi *Google Docs*, som ga oss muligheten til å jobbe på et dokument sammen. Google doc er også nyttig fordi den konstant lagrer endringer i rapporten automatisk. Endringer i rapporten blir også highlighted, og hvis vi noen gang skal tilbake i rapport-historien er det mulig.

For UI design av applikasjonen ble det brukt *Figma*, som er et nettbasert designverktøy som brukes av designere til å lage brukergrensesnitt, ikoner og logoer, ved hjelp av vektorformer, tekst og bilder. Under prosjektet ble figma brukt til å lage eller endre designet på applikasjonens design, som gjorde det lettere for alle å ha samme tanker om strukturen til applikasjonen.

For modellering ble *Diagrams.net* brukt, som er et kildekode-diagramverktøy som brukes av utviklere og designere til å lage diverse diagrammer, tabeller og flyt. Drag and drop brukergrensesnittet gjør det til et enkelt og nyttig verktøy for å representere våre modeller.

I løpet av prosjektet brukte vi også *Google Forms* som er et gratis undersøkelsesprogram som ble brukt til å samle data inn fra forumer innen for eksempel skydiving. Vår undersøkelse ble hovedsakelig brukt på Reddit, som er et internett-forum hvor brukere kan poste lenker til

annet innhold på nett, og samtidig stemme og diskutere i postene. Reddit inneholder diverse sub-forum, og for vårt mål om å nå ekstremsportentusiaster så var skydiving-siden viktig. På r/skydiving brukte vi google form undersøkelsene til å samle data om værforhold og ønsker for vårt prosjekt.

Vi brukte også Adobe Illustrator brukt for å lage en egen original logo for applikasjonen. Det er et redigeringsverktøy for vektorgrafikk, som er vanlig å bruke innen grafisk design.

5.2 Plan

En overordnet oversikt over prosjektets gang vil være undertittelene som oppsummerer hva som ble gjort i hvert sprint. Dette inkluderer også sprint 0 som gikk hovedsakelig til å planlegge og gjøre oblig 3 i IN2000.

5.2.1 Sprint 0 - Planleggingsfase og obligatorisk innlevering

I starten av prosjektet, i det vi kaller Sprint 0, så ble gruppen tildelt og vi møtte alle medlemmene for å bli kjent og diskutere møtetider som passet alle. Sprinten ble brukt til å finne ideer til hvilken case vi skulle jobbe med, og å fullføre obligatorisk innlevering 3 i IN2000. Vi ble enige om å lage en værapplikasjon for ekstremsport etter forslag og avstemninger.

Veiledningstimene i denne sprinten ga oss verdifull innsikt i hvordan vi skulle jobbe med applikasjonen og hvilke utfordringer vi kunne forvente oss basert på våre ideer. Vi utarbeidet en sprintplan og satt opp et Trello-board for å strukturere arbeidet videre. Det var også viktig å ha en skisse av hvordan applikasjonen skulle se ut, og i oppgaven for Sprint 0 ble alle i gruppen bedt om å lage noen designskisser til brukergrensesnittet. Disse skissene skulle presenteres i Sprint 1, og de beste delene ville bli satt sammen.

5.2.2 Sprint 1 - Design

I den første sprinten satte vi hovedsakelig fokus på å utvikle en design- og strukturplan for applikasjonen vår. Vi lot oss blant annet inspirere av Ryde-applikasjonen, som er en app for å

finne og bruke Ryde sine elektriske scootere i Norge. I forkant av denne sprinten hadde alle i teamet laget egne designforslag i sprint 0. I starten av sprint 1 ble vi enige om en felles design som ville være grunnlaget for videre utvikling av applikasjonen, samtidig som vi forbeholdt oss muligheten til å justere designet basert på fremtidige behov. De fleste i teamet hadde lignende tegninger som ga oss en god start på utviklingsprosessen.

I denne sprinten eksperimenterte vi med APIene vi skulle bruke og prøvde å parse dem. Vi startet også med å lage en brukerundersøkelse som vi ville legge ut på skydiving-forum for å få direkte tilbakemeldinger om hvilke værforhold som er egnet eller ikke egnet for hopping. Kjennskap til reelle fallskjermhoppere ville gi mer innblikk til hva vi burde fokusere mer på. Selv om det ikke var mye programmering i denne planleggingsfasen, fikk vi satt opp arbeid til sprint 2.

5.2.3 Sprint 2 - MVP

I veiledningstimen ble gruppen anbefalt å ha en Minimum Viable Product (MVP) etter påskeferien, og de satte seg som mål å være 50% ferdige med hele applikasjonen siden mange i teamet hadde mer fritid. De opprettet også prosjektrapporten og lærte seg å branche kode på Github.

Vi satte opp APIene og Viewmodel-en med bare noen få warnings, og begynte å implementere noen deler av designet vårt i sprint 2. I tillegg kontaktet vi flere fallskjermstasjoner for å få innsikt i hvilke faktorer som må tas i betraktning når det gjelder fallskjermhopping og værforhold. Dessverre fikk vi bare kontakt med én stasjon i Norge, og tilbakemeldingen vi fikk var ikke så nyttig som vi hadde håpet på. Vi bestemte oss derfor for å lage en undersøkelse og publisere den på et større nettforum for å få innspill fra andre aktive fallskjermhoppere internasjonalt.

I retrospekt var mange av målene som ble satt opp oppfylt og mer. User-interfacen var nesten identisk med designet de hadde laget i Sprint 1, og vi hadde integrert Google Maps i hovedsiden med en rød markør på en stasjon. Google Maps APIet ga oss noen utfordringer, som at den var hakkete i emulatoren, men ellers fungerte den som den skulle. Emulatorer er generelt krevende, så vi antok at dette var normalt.

Vær-APIene var satt opp, og det eneste som manglet var å finne terskelverdier for å vise visse fargekoder. Disse terskelverdiene skulle komme fra brukerundersøkelsen, og egen research om ekstremsporten. Brukerundersøkelsene de begynte på i Sprint 1 ble lagt ut på Reddit sitt skydiving-forum. Vi ventet noen dager for å få mer engasjement.

I samme sprint begynte vi å diskutere navn til applikasjonen.

5.2.4 Sprint 3 - Testing og design

I sprint 3 hadde vi allerede utviklet en MVP av applikasjonen vår, som oppfylte kravene til å bli regnet som en værapplikasjon for ekstremsport, og enda mer. I utgangspunktet skulle denne sprinten dreie seg om testing, men vi var usikre på hvor mye testing som var nødvendig og måtte evaluere situasjonen mens vi arbeidet. Vi innså også at vi lå litt foran planen med backend-arbeidet, så vi prioriterte å ta igjen på UI-designet.

Undersøkelse vi gjennomførte på Reddit-forumet fikk stor respons, og vi utarbeidet en liste over terskelverdier for fallskjermhopping. Disse terskelverdiene var ikke ensartede, men ble delt inn i kategorier som "gode", "moderate" og "dårlige" værforhold. Vi brukte informasjonen vi fikk fra undersøkelsen, samt andre kilder, for å fastslå de beste værforholdene å hoppe i og når det ville være umulig å hoppe. Det er viktig å påpeke at fallskjermhopping ikke er en sport med en fastsatt liste over værforhold som er trygge å hoppe i, og mye av ansvaret ligger på fallskjermhopperen selv for å evaluere sine egne grenser. Likevel var det enighet blant de fleste tilbakemeldingene om hvilke værforhold som var trygge og hvilke som ikke var det, selv blant de som kalte seg "eksperter".

Målet for denne sprinten var å fullføre nesten hele applikasjonen, og det var kun noen få oppgaver som gjensto på produktbacklogen. Vi løste utfordringene vi hadde med at Google Maps hakket, og implementerte en loading-skjerm som vises mens applikasjonen startet opp. Vi startet også arbeidet med å lage et JSON-dokument som kan lagre stasjonenes informasjon.

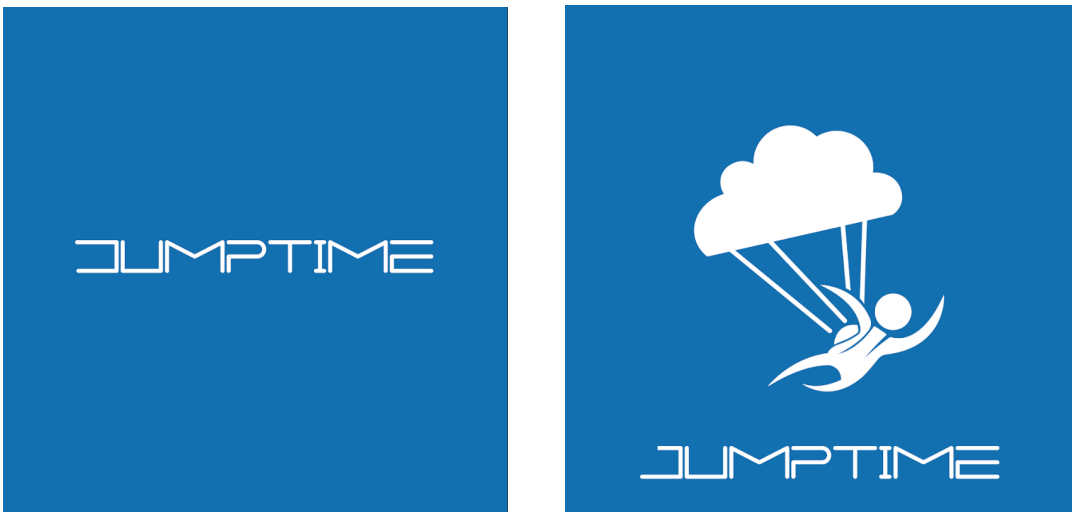
I løpet av sprinten opplevde vi også problemer med prosjektet fordi vi hadde ventet med å slå sammen branchene sammen med main. Dette førte til at applikasjonen krasjet, og vi måtte gå tilbake til tidligere versjon og manuelt legge til de nye kodelinjene som skulle være inkludert. Dette var en tidkrevende prosess som vi følte satte oss tilbake, og som sannsynligvis kunne vært unngått hvis vi var mer kjent med hvordan merging i Github fungerte. Til slutt klarte vi å gjenopprette appen til sin originale tilstand.

Vi diskuterte også om å fokusere på å fullføre skydiving-aspektet av applikasjonen først, og deretter jobbe videre med andre ekstremsporter når alt var ferdig.

5.2.5 Sprint 4 - Ferdig produkt

Målet for sprint 4 var å ha et fullstendig ferdig produkt ved slutten av sprinten. Vi gjorde også noen strukturelle endringer på applikasjonen ved å inkludere en bottom bar på alle sider, med unntak av loading skjermen. Da kan brukeren aksessere alt fra hvilken som helst side.

I begynnelsen av sprinten startet vi også med å designe en logo som skulle representere appen. Vi tok inspirasjon fra en stickman i vektorstil og en enkel sky, og alt ble skapt fra bunnen av i Adobe Illustrator. Vi eksperimenterte med forskjellige fargekombinasjoner, og fant ut at kombinasjonen av blå og hvit fungerte best. Vi utviklet også tekst som kunne brukes på hjørnene av app-skjermene, og samme logo ble også brukt på loading screenen.



(Logo-design)

I løpet av samme sprint fikk vi implementert markørene for hver skydiving-stasjon i Norge, som brukeren kan klikke på for å få opp en dropdown-boks med informasjon om stasjonen. Vi la også til funksjonaliteten for å kunne forstørre og forminske boksen.

En viktig funksjonalitet vi ønsket å inkludere i appen var muligheten til å enkelt finne fallskjermstasjoner rundt om i Norge og sjekke værforholdene der. I tillegg ønsket vi også å gi brukerne muligheten til å legge til egne favorittsteder for hopping. For å oppnå dette bestemte vi oss for å lagre all informasjon om de ulike stedene i et JSON-dokument. Tanken var å deserialisere dette dokumentet ved oppstart av appen, og deretter serialisere det tilbake til JSON når nye steder skulle legges til. Dette gjorde det enkelt for oss, utviklerne, å lese og håndtere informasjonen. Samtidig kunne eventuelle ekstra steder brukerne la til bli lagret lokalt på enhetene deres.

Implementeringen av denne funksjonaliteten tok opp mye tid i sprint 3, hele sprint 4 og en del av sprint 5. Vi eksperimenterte med ulike løsninger for å få tilgang til filen der stedene var lagret, og det ble brukt mye tid på denne utfordringen. I ettertid ser vi at det ville vært mer tidseffektivt å lagre all informasjon om stedene direkte i dataklassene i koden. Dette ville spesielt vært hensiktsmessig siden implementeringen av å legge til ekstra steder ikke engang ble fullført.

Mot slutten av sprinten innså vi at vi ikke hadde nok tid til å implementere de andre ekstremsportene vi hadde planlagt. Opprinnelig var ideen å lage en værapplikasjon for skydiving, men vi utvidet det til å inkludere flere ekstremsporter. Dessverre hadde tre av teammedlemmene hjemmeeksamen i IN2140, som begrenset tiden vi hadde til rådighet for å arbeide med applikasjonen de siste to sprintene.

Vi klarte heller ikke å oppnå målet vårt om å gjøre applikasjonen "ferdig" i denne sprinten på grunn av problemene med JSON. Målet ble overført til neste sprint.

5.2.6 Sprint 5 - Pussing og rapport

I sprint 5 satte vi oss som mål å fullføre prosjektet innen første halvdel av sprinten og holde et møte for å diskutere dette. Etter at alt av nødvendig funksjonalitet var implementert, skulle vi sette applikasjonen til side og fokusere 100% på å fullføre prosjektrapporten.

I tillegg skulle teammedlemmene prøve å få venner og kjente til å teste applikasjonen og gi tilbakemeldinger. Testene skulle for eksempel innebære å se hvordan de navigerte til forskjellige sider og hvordan de tolket informasjonen. Vi observerte hvor de klikket først, og hva som virket intuitivt for dem.Vi fikk gjennomført noen brukertester ved å spørre venner og familie.

I et ekstra møte i midten av sprinten merget vi det siste vi manglet, og applikasjonen var fullstendig ferdig. I løpet av en gjennomgang av koden så ryddet vi også koden, og endret noen av ikonene til android studios sine egne ikoner.

I løpet av sprinten ble JSON-filen fikset og fullført slik at vi kunne hente fallskjermstasjonene.

Resten av sprinten gikk til å fullføre prosjektrapporten ved at teamet skrev hva de hadde gjort på delene av prosjektet de hadde jobbet på.

5.3 Endringer

I løpet av prosjektet var det en del endringer som ble gjort, spesielt når det kom frem til strukturer og hvilke funksjoner vi kunne ta med. Endringene kom hovedsakelig fra ønsker av brukerundersøkelsen, og forbedringsmuligheter medlemmer i gruppen så og stemte over.

5.3.1 Ekstremsporter

I løpet av sprint 3 begynte vi å tvile på om det var realistisk å implementere en værapplikasjon for flere ekstremsporter, i tillegg til fallskjermhopping. I sprint 4 ble vi enige om å begrense fokuset vårt til skydiving alene. Dette er noe vi sannsynligvis kunne ha

oppnådd med mer tid, men de siste to sprintene i prosjektet ble hektiske ettersom flere av gruppemedlemmene også hadde hjemmeeksamener i andre fag å håndtere.

Likevel, da vi startet prosjektet, hadde vi intensjonen om å inkludere flere ekstremsporter i appen vår. Vi har derfor bygget applikasjonen slik at det er mulig å legge til flere ekstremsporter i fremtiden, ved å samle inn informasjon gjennom brukerundersøkelser og datainnsamling. Det er imidlertid viktig å merke seg at brukerundersøkelser og forskning tar tid, spesielt når det gjelder å finne de optimale forholdene for trygge ekstremhopp i hver enkelt sport.

5.3.2 Utviklingsmetode

I 5.1 Utviklingsmetode beskriver vi vår valgte tilnærming som er en kombinasjon av Scrum og Kanban, som vi fulgte gjennom prosjektet. Vi implementerte sprinter og opprettet et Kanban-board for å administrere prosjektoppgavene. I begynnelsen av prosjektet hadde vi også en roterende scrum master. Imidlertid ble kanban-boardet og scrum master-rollen ikke opprettholdt etter de første tre sprintene.

Dette skjedde fordi de viktigste oppgavene på Kanban-boardet ble gjennomført, og gruppemedlemmene utviklet en felles forståelse av hverandres oppgaver uten å være avhengige av Kanban-boardet. Vi hadde diskusjoner om våre planer for hver sprint og opprettholdt god kommunikasjon gjennom Discord, slik at alle var oppdaterte på hverandres arbeidsområder og behov.

Vi innså også etter hvert at det ikke var nødvendig med en roterende Scrum Master for å gjennomføre prosjektet. Vi fant ut at det beste var å diskutere og ta avgjørelser sammen som en gruppe. Opprettelsen av sprints og planlegging av møter kunne gjøres uten en dedikert scrum master. Alle i gruppen var fleksible og vi kom alltid frem til en felles enighet.

Vårt samarbeid som en helhetlig gruppe, hvor vi alle var litt mer fri og ikke hadde spesifikke roller, var mer effektivt for oss. Men vi ser fordelene til en ordentlig struktur og ute i arbeidslivet er dette mye mer nyttig.

5.3.3 Funksjoner

Vi introduserte hvilke funksjoner vi ville ha med i 2.2 Funksjonaliteter i rapporten, men uheldigvis ble ikke alle implementert på grunn av mangel på tid. Vi så at de problemene med å danne en skydiving-værapplikasjon var større enn de funksjonene vi så for oss å implementere ved siden av.

De viktige funksjonene som for eksempel å få anbefalinger om ekstremsporthopp i ulike stasjoner ble implementert, men det å kunne filtrere markører etter ekstremsporter ble ikke. Kartet og markørene fungerer akkurat som de skal, men muligheten til å kunne for eksempel finne sin nåværende posisjon på kartet eller muligheten til å arkivere hopp ble ikke implementert som vi hadde håpet på.

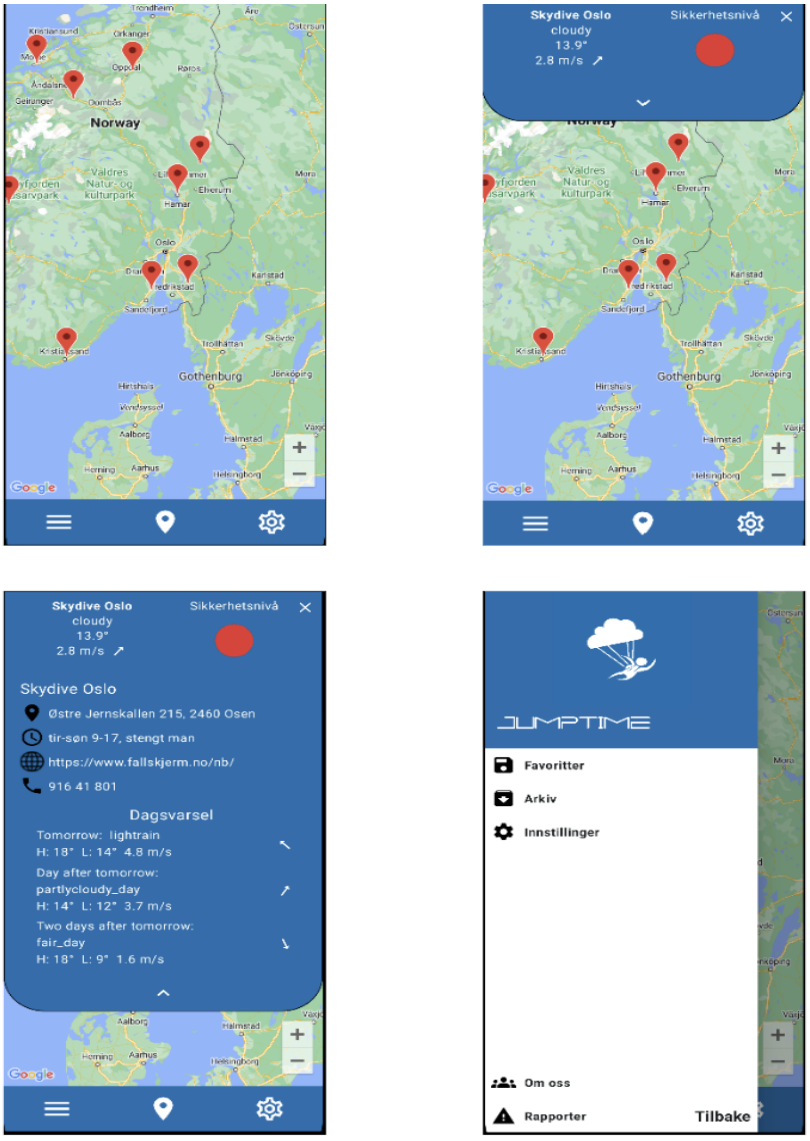
Vi fikk heller ikke implementert muligheten til å endre applikasjonens brukeropplevelse med en nattmodus-funksjon eller andre språkalternativer som engelsk. Vi visste at tilpasninger som de var det siste som skulle prioriteres, og det viktigste var å fokusere på hovedfunksjonene.

Skjermen til arkiverte hopp, og ratingsystemet ble derimot implementert, men igjen så hadde vi ikke nok tid til å få stasjoner til å dukke opp der som elevatedCards.

Your-location-knappen og muligheten til å se sin nåværende posisjon ble heller ikke implementert i hovedsiden.

Nye funksjoner som ble implementert derimot, var bottom baren som nå er til stede i alle skjermene slik at brukeren har muligheten til å rekke alle andre skjermer derfra. Ideen kom fra en brukertest hvor de etterspurte muligheten til å rekke andre skjermer fra hvor som helst. Etter å evaluere denne ideen med gruppen, så kom vi frem til at det var smart å implementere for en lettere brukeropplevelse.

5.3.4 Struktur og design



(Skjermbilder av nåværende app)

En av de største endringene vi gjorde i design og strukturen var å fjerne knappen som skulle gi brukeren muligheten til å velge ekstremsport, og erstattet den med knappen tilbake til hovedsiden.

Til tross for disse endringene er strukturen og designet ganske lik vår figma-tegning fra Sprint 1. Noen andre endringer som vi gjorde var på headeren til navigationdraweren. Istede for å vise en profil med navn og poeng, implementerte vi JumpTime-logoen med en JumpTime skrift under som vi ser fra skjermbildet av den nåværende appen, nederst til høyre.

Vi forstod etterhvert at det var irrelevant å vite noe om brukere når applikasjonen skulle bare være ment som et verktøy. Under headeren har vi erstattet profil med favoritter. Vi fjernet også poengsystemet som vi originalt hadde planlagt å legge til, men på grunn av at vi allerede ikke hadde tid til å arkivere hopp, så tenkte vi at det var meningsløst å ha den med. Så istedenfor Scoreboard har vi nå arkiv istedenfor. Ideen til arkivskjermen er å kunne lagre steder man har tidligere hoppet og samtidig se et vurderingssystem av de forskjellige stasjonene. I hovedskjermen valgte vi heller ikke å implementere Your-location-knappen som flytter kartet til nåværende posisjon. Istedenfor ligger det nå en knapp som er fra Google Maps sitt bibliotek, som lar brukeren forstørre eller forminske kartet.

Det ble også gjort noen endringer i ShowInfoBoxen, som for eksempel implementasjonen av vindretning som mange fra brukerundersøkelsen etterspurte. Windy er hovedsakelig brukt av skydivere og er kjent for sin animerte vindretning egenskap. Vi tenkte at hvis mange fallskjermhoppere bruker applikasjonen for den funksjonaliteten, måtte vi også inkludert det i vår egen app. Windy vinner derimot på at de har vindretningene animert mot sine retninger.

Vi tok også inn implementasjonen til å kunne se værforhold for tre dager frem i tid i ShowInfoBoxen ved å gi informasjon om værtype (regn, sol, overskyet osv), vindretning, vindstyrke og temperatur. Vi hadde opprinnelig ønsket å vise værvarsler for de neste syv dagene, men fant ut at APIet vi skulle bruke ikke ga oss informasjon over tre dager. Dette gir muligheten for folk å planlegge skydiving-turer frem i tid. Vi fikk ikke lagt til sikkerhetsnivå for hver dag. Grunnen til det igjen var at vi ikke hadde god nok tid, og at endringen krevde omskrivning av en del funksjoner som allerede var implementert.

5.4 Testing

5.4.1 Emulatortesting

Vi gjennomførte hovedsakelig testing av applikasjonen på en Pixel 5 med API-nivå 33, og alt fungerte som det skulle. Vi utførte imidlertid også tester på eldre API-nivåer, som API-nivå 30, 27 og 23.

På Pixel 2-telefonen fungerte applikasjonen godt med API-nivå 30, bortsett fra noen problemer med størrelsen på Jumptime-logoen i navigationdraweren. Til tross for noen PNG-relaterte problemer på telefonen, fungerte applikasjonen bra. Fonter og knapper var ikke påvirket til graden hvor de stoppet å være intuitive å bruke, men det var små forskjeller fra den normale Pixel 5 størrelsen.

Vi testet også på en Pixel 5 med API-nivå 23, men opplevde at applikasjonen krasjet på grunn av vector drawables som sannsynligvis ikke eksisterer på det nivået. På API-nivå 27 opplevde vi også krasj på samme telefon på grunn av forsøket på å tegne et for stort bitmap på canvas. Dette skyldes sannsynligvis at noen av PNG-filene har for høy oppløsning. Vi regnet også med at kartet ville slite med å vises frem på eldre APIer på grunn av Google Maps kompatibiliteten som nevnt i 4.3 API.

Vi utførte også tester på en Pixel C-nettbrett med API-nivå 33, og det fungerte godt. Eneste observasjonen var at skriftstørrelsen ble mye mindre ved at vi hadde mer plass, mens UI-deler forble like stor som en telefon. Løsningen til dette er å gjøre alle størrelser dynamiske.

5.4.2 Enhetstester

Enhetstestene som er skrevet for programmet tester flere av backend-funksjonene i programmet. Det er ingen enhetstester som er skrevet for noe av det visuelle, ettersom ingen av funksjonene returnerer noe.

APIene vi bruker har hver sin enhetstest. I stor grad sjekker de bare verdier vi er sikre på hva skal være, som for eksempel koordinater og type resultat. Det er ikke hensiktsmessig å teste noe annet, ettersom alle de andre verdiene avhenger av vær.

Det er også skrevet enhetstest for funksjonen som regner ut om det er greit å hoppe eller ikke. Ettersom funksjonen bruker værdata som endrer seg, er det ikke mulig å teste funksjonen så grundig. Det testene gjør er å sjekke om funksjonen gir ønsket resultat når vi tester for ekstreme terskelverdier. Altså, alle kravene blir satt til verdier som gjør at funksjonen alltid skal regne ut at det er godt nok vær til å hoppe. Gjør den ikke det, vet vi at noe er galt et sted.

Det er også skrevet enhetstester for noen diverse hjelpefunksjoner. Her testes det om hjelpefunksjonene gir det resultatet som er forventet med et gitt input. Input er i noen tilfeller en gitt verdi, i andre tilfeller null. I de tilfellene det er null så er det for å teste om funksjonen ikke krasjer programmet når null er input.

5.4.2 Brukertester

Vi gjennomførte brukertester med personer vi hadde tilgjengelig, da dette var det mest praktiske alternativet. Målet med testene var å få deltakerne til å utføre enkle oppgaver som vi muntlig forklarte. Eksempel på en slik oppgave var, “Finn vindstyrken om to dager på stasjonen i Bergen” eller “Rapporter en feil i applikasjonen”.

Deretter ba vi brukerne om å gi tilbakemeldinger om applikasjonen, samt eventuelle ønsker og endringer de hadde. Vi tok notater mens brukerne arbeidet med oppgavene for å se om de klarte dem. Etter at de var ferdige, spurte vi dem om de syntes at plasseringen av tekst, knapper og andre elementer i appen var intuitive. I avsnitt 4.2.5 har vi beskrevet disse tilbakemeldingene og hvordan vi implementerte noen av dem.

Foruten å samle tilbakemeldinger, så var brukertestene også nyttige for å observere hvordan brukerne navigerte seg gjennom appen for å oppnå målene sine. Vi la merke til at noen av brukerne hadde litt problemer med å finne den større boksen med værvarsel og stasjonsinformasjon. Noen prøvde å dra på skjermen som om de brukte en berøringsskjerm, mens andre bare trykket, og fikk det til. Denne tendensen kommer sannsynligvis fra andre applikasjoner med lignende bokser.

5.5 Teknisk gjeld

5.5.1 MainScreen

Viewmodellen ble ikke benyttet i koden under utviklingen. Det ville ha vært nyttigere hvis den hadde blitt sendt til de stedene hvor den kunne ha blitt brukt, som for eksempel på alle skjermene. Ved å inkludere viewmodellen på forhånd i stedet for å vente til man er sikker på nøyaktig hvordan den skal brukes, vil det være tydeligere at den er ment å være en integrert

del av applikasjonen. Dette vil også gjøre den mer tilgjengelig og enklere å bruke når man trenger den.

Hovedskjermen er komponert av en øvre del som inneholder informasjon, et kart i midten, og en bottom bar. Kodens struktur reflekterer ikke det samme som designets struktur på grunn av at koden er strukturert slik at underbaren er øverst, og deretter kommer informasjonsboksen og kartet. Det blir en bedre sammenheng med hvordan appen faktisk ser ut dersom denne flyttes til bunnen. Gjenspeilingen vil også føre til at det er lettere for andre kodere eller designere å forstå prinsippet.

Noen av beregningene i koden inneholder if-blokker og ligger på et dypere nivå enn nødvendig. For å forbedre ytelsen og gjøre koden mer effektiv, bør disse beregningene heller gjøres om til konstanter tidligere i koden. For eksempel vil en fjerdedel av skjermen alltid være den samme uansett hvor man befinner seg i applikasjonen. Ved å konvertere disse beregningene til variabler, vil det resultere i mindre beregningstid for applikasjonen. Selv om dette virker som ubetydelige endringer, så kan små beregninger samle seg opp og påvirke ytelsen når appen blir kompleks.

Koden inneholder litt overflødig logikk. Informasjonsboksen på toppen av skjermen ble kodet slik at det først ble sjekket for en variabel som bestemte om det skulle vises mye eller lite informasjon i boksen. Variabelen bestemmer mengden informasjon som skal vises til brukeren, og mengden informasjon ble vist i enten en liten eller stor boks. En bedre tilnærming ville være å ha en boks som alltid viser den samme informasjonen, og deretter bruke variabelen til å bestemme om den eksklusive informasjonen fra den lengre boksen skal vises. Dette vil gjøre det lettere å utvide boksene senere hvis man for eksempel ønsker å endre den mindre boksen, siden man bare trenger å redigere ett sted. Denne endringen vil også redusere kodens kompleksitet og gjøre den lettere å vedlikeholde på lang sikt.

Koden vår har en tendens til å hardkode informasjon i funksjoner som antas å bli tilgjengelig senere. Dette gjøres ved å legge til informasjonen direkte i koden på steder hvor den ikke er tilgjengelig, i stedet for å lage en parameter for den informasjonen. For eksempel, i stedet for å lage en parameter for et stedsnavn, skriver vi inn stedsnavnet direkte i funksjonen som en streng: "plass: Tromsø". Dette kan føre til to problemer. For det første, når vi endelig får tilgang til informasjonen senere, vil funksjonen allerede ha informasjonen hardkodet inn, noe

som kan gjøre det vanskelig å implementere den nye informasjonen i koden. For det andre, når vi skal bruke funksjonen senere som den var ment, må vi først finne igjen de hardkodede strengene, noe som kan være tidkrevende og frustrerende. Derfor bør vi i stedet lage parametere for informasjonen i funksjonene våre for å gjøre koden vår mer fleksibel og lettere å vedlikeholde.

5.5.2 LoadingScreen

En lastefunksjon som brukes i LoadingScreen som var utnyttet, kom fra en annen package enn LoadingScreen. Det blir klarere hvor denne kommer til å brukes dersom den er i samme fil som der den er tenkt å brukes.

5.5.3 SettingsScreen

Det virker som om det er noen gode forslag til forbedringer av koden her. For det første kan det være lurt å vurdere om en innstillingsfunksjonen burde være sin egen klasse eller funksjon, slik at det blir enklere å legge til nye innstillinger i fremtiden. Videre kan det være nyttig å lage en funksjon for den gjentakende skjermstrukturen, slik at det blir lettere å gjøre endringer i stil og tekst på alle disse skjermene samtidig.

5.5.4 ArkivScreen og FavorittScreen

Her også kan det være en god idé å bruke objekter for å representere kortene for hoppene i applikasjonen. Å holde kortene i en liste og vise dem ved en for-loop vil gjøre koden mer effektiv og lettere å vedlikeholde senere.

5.5.5 ReportScreen

Skjermen er en stor funksjon og den tjener tre formål. Først til å velge hva slags type problem som skal rapporteres. Så til å skrive problemet og til slutt til å sende rapporten. Hvis skjermen var komponert av tre funksjoner som hver var for et formål, ville koden blitt mer lesbar, og det blir lettere å se hvor koden som gjør hva er, og derfor lettere å revidere senere.

5.5.6 OmOssScreen

Appen inneholder hardkodet tekst som kan skape utfordringer når det kommer til oversettelse av appen. Hvis man ønsker å oversette appen senere, vil man måtte gå gjennom hele koden og oversette hver streng man finner. En bedre tilnærming ville vært å bruke resource-mappen i Android Studio til å lagre teksten. Dette ville gjort oversettelsesprosessen mye enklere ved å åpne den ene filen med teksten og oversette der. Hvis man ønsker å støtte flere språk, kan man lage et system som velger ønsket språk og deretter lage en fil med teksten på det nye språket. På denne måten ville oversettelsesprosessen være enklere og mer skalerbar.

5.5.7 ESViewModel og DataSource

En gjentakende utfordring gjennom koden er at Viewmodel-en virker å bli oversett. Selv om den utvikles, virker det som om mye av informasjonen som den holder på ikke blir brukt på riktig måte. Dette kan skyldes av at mye av informasjonen virker utilgjengelig selv om den er det. Datasource laster inn APIene, og Viewmodel holder på denne informasjonen, inkludert vindstyrke, temperatur og annen relevant informasjon. Imidlertid lagres denne informasjonen i variabler som sunrisedata og locationdata, som fra et frontend-perspektiv er uinteressante. Det som faktisk er relevant, er å kunne hente ut vindstyrken eller temperaturen. For å gjøre denne informasjonen mer tilgjengelig bør informasjonen fra APIene organiseres bedre ved å bruke variabler som windSpeed, eller ved å lage funksjoner i Viewmodel som getWindSpeed(). Da kan man enkelt hente ut relevant informasjon uten å måtte lære seg APIene for å bruke dem.

5.6 Refleksjon

Den overordnede planen for prosjektet var å utvikle en applikasjon som skulle vise tryggheten til å kunne utføre ekstremhopp i ulike deler av Norge. Dette målet ble oppnådd, og vi ser på applikasjonen som en suksess i form av at den utfyller vår visjon som vi hadde i starten av prosjektet.

Teamarbeidet var også en suksess ved at vi generelt følte at vi hadde kontroll gjennom hele prosjektet, og alle i gruppen var begeistret for å lage en applikasjon som vi syntes var interessant. Hele teamet jobbet kontinuerlig med god kommunikasjon, som gjorde at alle i

gruppen prøvde sitt beste. Arbeidet ble fordelt jevnt mellom alle medlemmene i gruppa, og det var ingen som måtte gjøre mye mer enn de andre. Dette skyldtes av at vi ble såpass kjente med hverandre til å alltid komme med innspill og ikke være redde for å ta kontakt hvis de trengte hjelp.

Selv om gruppen fungerte som vi hadde ønsket, så hadde vi en samtale om hva som kunne ha gått bedre. Vi konkluderte med at effektiviteten er noe vi ville ha endret på. Ønsket vårt var å lage en fullstendig ferdig applikasjon, men det ble noe svekket av ineffektivt arbeid. Det var mye som vi anså som viktig å ha med på starten, men uheldigvis måtte settes til side for å rekke og lage et prosjekt som oppfylte kravene til oppgaven. Dette er en lærdom som vi tar med oss videre.

De endringene vi gjorde i henhold til selve applikasjonen var alle relatert til mangel på tid. Hadde vi hatt mer tid, skulle vi implementert flere av funksjonalitetene vi hadde planlagt. Det var ingen sentrale funksjonaliteter vi ikke rakk å implementere, men mange tilleggsfunksjonaliteter uteble. I planleggingsfasen hadde det vært lurt å sette litt mer tid til hver individuelle funksjonalitet.

Til slutt kan det være verdt å nevne at selv om vi hadde et rammeverk for hvordan teamarbeidet skulle fungere, så fulgte vi ikke dette nøye. Vi følte at arbeidet gikk litt av seg selv etter at startfasen var gjennomført, og derfor ble scrumban-systemet vårt unødvendig.

6. Kilder og referanser

Agrawal, H. (2014, 07 19). *bungee jump Icon - Free PNG & SVG 60805*. Noun Project. Retrieved 03 27, 2023, from https://thenounproject.com/icon/bungee-jump-60805/

Askheim, S. (2021, 02 9). *fallskjermhopping*. SNL. Retrieved 03 30, 2023, from https://snl.no/fallskjermhopping

Evericons. (2019, 05 7). *Evericons*. Evericons. Retrieved 03 27, 2023, from https://www.figma.com/file/CPoDgZqWQs73DN3Am1vreK44/Evericons?type=design&node-id=0-1&t=MTQ7SPMd91DTry2r-0

Flaticon. (Ukjent dato, . .). *internet_1011322*. internet_1011322. Retrieved 05 23, 2023, from https://www.flaticon.com/free-icon/internet_1011322

Google. (2023, 03 1). *Maps SDK for Android*. Maps SDK for Android. Retrieved 05 18, 2023, from https://developers.google.com/maps/documentation/android-sdk

Google. (2023, 05 18). *Maps Compose Library*. Maps Compose Library. Retrieved 04 10, 2023, from https://developers.google.com/maps/documentation/android-sdk/maps-compose

Google. (2023, 05 18). *Overview of Google Play services*. Overview of Google Play services. Retrieved 05 24, 2023, from https://developers.google.com/android/guides/overview

Google. (2023, 05 18). *Set up an Android Studio project*. Set up an Android Studio project. Retrieved 04 10, 2023, from https://developers.google.com/maps/documentation/android-sdk/config

Google. (2023, 05 18). *Set up your Google Cloud project*. Set up your Google Cloud project. Retrieved 04 10, 2023, from https://developers.google.com/maps/documentation/android-sdk/cloud-setup

Google. (2023, 05 18). *Using API Keys*. Using API Keys. Retrieved 04 10, 2023, from https://developers.google.com/maps/documentation/android-sdk/get-api-key#console_1

Google. (Ukjent dato, . .). *Google Maps*. Google Maps. Retrieved 03 27, 2023, from https://www.google.com/maps/d/viewer?mid=1NbIT_7wYos0qabv6m-6wTsZIt2M&hl=en_US&ll=59.92333307033322%2C10.795734303466409&z=12

Google. (Ukjent dato, . .). *Oslo - Google My Maps*. Oslo - Google My Maps. Retrieved 03 27, 2023, from https://www.google.com/search?q=oslo+google+maps&rlz=1C1VDKB_noNO1046NO1046&sxsrf=APwXEddNR6xq_M4YQULOD2tp6Sc4AjMypg:1685114804416&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjAqsfqpZP_AhVmkosKHQWfAHsQ_AUoAnoECAEQBA&biw=1440&bih=781&dpr=2#imgrc=8Y6TyBpXrmLXcM

Google. (Ukjent Dato, . .). *Skydive Oslo - Oslo Fallskjermklubb, Østre Jernskallen*. Skydive Oslo - Oslo Fallskjermklubb, Østre Jernskallen. Retrieved 03 27, 2023, from https://www.google.com/maps/place/Skydive+Oslo+-+Oslo+Fallskjermklubb/@61.2568871,11.6684087,17z/data=!3m1!4b1!4m6!3m5!1s0x466a21fbf5061c6b:0xbe886228253a8e9b!8m2!3d61.2568845!4d11.6709836!16s%2Fg%2F11js2ssyfz?entry=ttu

Gupta, R. (2013, 02 20). *Skydiving Icon - Free PNG & SVG 11931*. Noun Project. Retrieved 03 27, 2023, from https://thenounproject.com/icon/skydiving-11931/

Introduction to Model View View Model (MVVM). (2022, 07 13). GeeksforGeeks. Retrieved 04 17, 2023, from https://www.geeksforgeeks.org/introduction-to-model-view-view-model-mvvm/

javaTpoint. (., . Ukjent). *Android API Levels*. Android API Levels. Retrieved 04 25, 2023, from https://www.javatpoint.com/android-api-levels

Lay, G. K. (2016, 09 17). *Wingsuit Icon - Free PNG & SVG 639525*. Noun Project. Retrieved 03 27, 2023, from https://thenounproject.com/icon/wingsuit-639525/

Lay, G. K. (2017, 09 21). *paragliding Icon - Free PNG & SVG 1281393*. Noun Project. Retrieved 03 27, 2023, from https://thenounproject.com/icon/paragliding-1281393/

UiO. (Ukjent dato, . .). *Oversikt over Universitetets bygninger*. Oversikt over Universitetets bygninger. Retrieved 04 24, 2023, from https://admin.uio.no/ta/PDF_fra_Pythagoras_bygningsvis_public/universitetet/gaustad/ga06/ga06.htm

USPA. (Ukjent dato, . .). *How safe is skydiving?* United States Parachute Association. Retrieved 03 30, 2023, from https://www.uspa.org/Discover/FAQs/Safety

Vihovde, E. H. (2021, 12 16). *objektorientert programmering*. SNL. Retrieved 04 17, 2023, from https://snl.no/objektorientert_programmering

Wikimedia Commons. (2020, 04 17). *File:Eo circle red blank.svg*. File:Eo circle red blank.svg. Retrieved 05 23, 2023, from https://commons.wikimedia.org/wiki/File:Eo_circle_red_blank.svg

Wikimedia Commons. (2020, 10 29). *File:Green icon.svg*. File:Green icon.svg. Retrieved 05 23, 2023, from https://commons.wikimedia.org/wiki/File:Green_icon.svg

Wikipedia. (2006, 08 30). *File:Yellow icon.svg*. File:Yellow icon.svg. Retrieved 05 23, 2023, from https://en.m.wikipedia.org/wiki/File:Yellow_icon.svg

7. Vedlegg

7.1 Brukerundersøkelse

Brukerundersøkelse:

https://docs.google.com/forms/d/1ytDwfs4lTlAdPUPd1mZoHI7VWP_V-GgdqxM1445CSAA/edit

Reddit post med brukerundersøkelse:

https://www.reddit.com/r/SkyDiving/comments/12idqka/skydivingweatherapplication_project/

7.2 Klassediagram

7.3 Responsskjema

	API	Vurdering	Kommentar
Værvarsel	AirQualityForecast 0.1		
	Locationforecast 2.0 (classic.xml)		
	Locationforecast 2.0 (compact.json)		
	Locationforecast 2.0 (complete.json)	4	Eksmepel dataen er ikke den samme som det APIet faktisk gir
	MetAlerts 1.1 (CAP/RSS)		
	MetAlerts 1.1 (JSON)		
	Nowcast 2.0	5	
	Subjectiveforecast 1.0		
	Textforecast 2.0 (XML)		
	Textforecast 3.0 (JSON)		
Flyvær	Aviationforecast 1.6		
	FMIroutes 0.1		
	NLAroutes 1.0		
	Offshoremaps 1.0		
	Routeforecast 2.0		
	Sigcharts 2.0		
	Sigmets 2.0		
	Spotwind 1.1		
	Tafmetar 1.0		
	Turbulence 1.1		
	VerticalProfile 1.1		
	Volcanicashforecast 0.1		
Hav og kyst	Gribfiles 1.1		
	Icemap 1.0		
	Oceanforecast 2.0		
	Oslofjord 0.1		
	Soundprofile 1.0		
	Textforecast 2.0		
	Tidalwater 1.1		
Andre	Sunrise 2.0 (XML)		
	Sunrise 2.1 beta (XML)		
	Sunrise 3.0 beta (JSON)	5	
	Weathericon 2.0		
	Frost 0.9		
	Hawarsel-Frost		
	thredds.met.no		

Leverandør	API	URL	Vurdering	
NILU	NILU API	https://api.nilu.no/		
Hva Koster Strømmen?	Strømpris API	https://www.hvakosterstrommen.no/strompris-api		
NVE	Nettleiestatistikk	https://data.norge.no/dataservices/937a0466-3f12-3219-8552-18689cf8d606		
Kartverket	Åpen Adress	https://ws.geonorge.no/adresser/v1/punktsok?lat=59.933333&lon=10.716667&radius=1000	4	Litt vanskelig å implementere