Høgskulen på Vestlandet

BACHELOROPPGAVE

Kryssplattform-applikasjon for varsling av brannrisiko

Cross-Platform Application for Fire Risk Notification

Jørgen Toppen Moen

Johan Ludvig Sørlie

Dat191

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Dataingeniør

Veileder Lars Michael Kristensen

22.05.2023

Jeg bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, *jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 10.*

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Kryssplattform-applikasjon for varsling av brannrisiko	Dato: 22.05.2023
Forfatter(e): Jørgen Toppen Moen, Johan Ludvig Sørlie	Antall sider u/vedlegg: 62 Antall sider vedlegg: 95
Studieretning: Dataingeniør	Antall disketter/CD-er: Ingen
Kontaktperson ved studieretning: Lars Michael Kristensen	Gradering: Ingen
Merknader: Ingen	

Oppdragsgiver: DYNAMIC	Oppdragsgivers referanse: Ingen
Oppdragsgivers kontaktperson: Ruben Dobler Strand	Telefon: 48 35 00 63

Stikkord:

Kryssplattform	Brannrisiko	DYNAMIC

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Sammendrag:

Gjengroing, klimaendringer og ekstremvær forverrer det globale storbrannproblemet, som i Norge utgjør en spesiell risiko i tett trehusbebyggelse. Forskningsprosjektet DYNAMIC ved Høgskulen på Vestlandet ønsker å bidra til å bedre forstå parametere som leder til storbranner. Gjennom DYNAMIC er det blant annet utviklet en brannrisikomodell for trehus.

Denne rapporten beskriver utviklingen av en kryssplattform-applikasjon ved bruk av .NET MAUI, og viser at rammeverket kan brukes til å utvikle løsninger med en stor grad av gjenbrukt kode mellom plattformer. Kryssplattform-applikasjonen som ble utviklet tilbyr predikert brannrisiko i trehusbebyggelse gjennom brukervennlige begrep og visualiseringer på mobil og skrivebord. Forvarsel om periode med høy brannrisiko til både brannvesenet og privatpersoner kan bidra til preventive tiltak. Skal en eksisterende løsning transformeres til nytt rammeverk anbefaler prosjektet gjenbruk av arkitektur og modeller, men nyutvikling av brukergrensesnitt.

Overgrowth, climate change, and extreme weather exacerbate the global issue of large-scale fires, which in Norway, pose a particular risk in densely built-up wooden house areas. The DYNAMIC research project at Western Norway University of Applied Sciences aims to contribute to a better understanding of the parameters leading to such fires. DYNAMIC has, among other things, developed a fire risk model for wooden houses.

This report describes the development of a cross-platform application using the .NET MAUI framework, demonstrating that the framework can be used to develop solutions with a high degree of code reuse across platforms. The cross-platform application developed offers predicted fire risk in wooden homes through user-friendly terms and visualizations on mobile and desktop. Early warnings of periods with high fire risk to both the fire department and private individuals can aid in preventative measures. When transforming an existing solution to a new framework, the project recommends reusing the architecture and models, but developing the user interface from scratch.

FORORD

Prosjektet som beskrives i denne rapporten har tilhørighet til studieprogrammet Dataingeniør ved Høgskulen på Vestlandet (HVL). Arbeidet er gjennomført av Jørgen Toppen Moen og Johan Ludvig Sørlie våren 2023. Gjennom prosjektet ble det utviklet en kryssplattform-applikasjon for varsling av brannrisiko i trehusbebyggelse. Oppgaven ble gitt av forskningsprosjektet "Reducing Fire Disaster Risk Through Dynamic Risk Assessment and Management (DYNAMIC)" ved HVL.

Prosjektgruppen ønsker å rette en stor takk til veileder prof. Lars Michael Kristensen og representant for prosjekteier, stipendiat Ruben Dobler Strand, for verdifull veiledning. Deres engasjement og konstruktive tilbakemeldinger var vesentlige for resultatene som ble oppnådd i prosjektet.

Til slutt ønsker prosjektgruppen å takke venner og familie som har vært god støtte underveis og hjulpet til med gjennomlesing.

INNHOLDSFORTEGNELSE

1 INNLEDNING 1

1.1 KONTEKST 1

1.2 PROSJEKTEIER 2

1.3 MOTIVASJON 2

1.4 MODELL FOR BRANNRISIKO 2

1.5 PROBLEMBESKRIVELSE OG MÅL 3

1.6 FORSKNINGSSPØRSMÅL 3

1.7 OPPBYGGING AV RAPPORTEN 4

2 PROSJEKTBESKRIVELSE 5

2.1 PRAKTISK BAKGRUNN 5

2.1.1 Tidligere og relatert arbeid 5

2.1.2 Initielle krav 6

2.1.3 Initiell løsningsidé 6

2.2 RESSURSER 6

3 DESIGN AV PROSJEKTET 8

3.1 FORSLAG TIL LØSNING 8

3.1.1 Alternativ 1: MAUI 8

3.1.2 Alternativ 2: Flutter 8

3.1.3 Alternativ 3: Individuell utvikling 9

3.1.4 Diskusjon av alternativene 9

3.2 VALGT LØSNING 10

3.3 PROSJEKTMETODIKK 11

3.3.1 Utviklingsmetodikk 11

3.3.2 Prosjektplan 12

3.3.3 Strategi 12

3.3.4 Risikovurdering 12

3.4 EVALUERINGSPLAN 13

4 DETALJERT LØSNING 14

4.1 BRUKSTILFELLEDIAGRAM 14
4.2 DOMENEMODELL 16
4.3 DESIGNMØNSTER 17
4.3.1 MVVM 17
4.3.2 Data Binding 18
4.3.3 Community Toolkit MVVM 19
4.4 PROGRAMVAREARKITEKTUR 21
4.5 VÆRDATA 22
4.5.1 Meteorologisk institutt 22
4.5.2 Frost API 22
4.5.3 WeatherForcast API 23
4.6 BRANNRISIKO 24
4.7 DATAMODELLER 25
4.7.1 Fire Risk Model 25
4.7.2 Models 26
4.8 TJENESTEKLIENTER 27
4.9 APPLIKASJONSFLYT 28
4.10 SQLITE DATABASE 29
4.11 PLATTFORMTILPASSET KILDEKODE 31
4.12 GRAFISK BRUKERGRENSESNITT I .NET MAUI 32
4.12.1 Pages 33
4.12.2 Layouts 33
4.12.3 Views 33
4.13 GRAFISK BRUKERGRENSESNITT I LØSNING 33
4.13.1 Navigering 34
4.13.2 Mine lokasjoner 36
4.13.3 DataTemplate 36
4.13.4 Expander på mobil 38
4.13.5 Alternativ for skrivebord 39

4.13.6 Celler *40*

4.13.7 Celle for stolpediagram *41*

4.13.8 Celle for graf *42*

4.13.9 Celle for tabell *43*

4.14 FARGEVALG OG STILER 44

4.15 OVERSETTING 46

5 EVALUERING 47

5.1 DEMONSTRASJON 47

5.2 TEST AV FERDIG PRODUKT 49

5.3 EVALUERING AV KRYSSPLATTFORM-RAMMEVERK 50

5.3.1 Rammeverkets formål *50*

5.3.2 Oppnådd grad av kryssplattform *50*

5.3.3 Prosjektgruppens erfaringer *51*

6 DISKUSJON 52

7 KONKLUSJON OG VIDERE ARBEID 54

7.1 SVAR PÅ FORSKNINGSSPØRSMÅL 54

7.2 ANTAGELSER OG POTENSIELLE FEILKILDER 55

7.3 VIDERE ARBEID 56

8 REFERANSER 59

9 VEDLEGG 63

9.1 GANTT-DIAGRAM 63

9.2 RISIKOVURDERING 64

9.3 RISIKOMATRISE 65

9.4 INTERVJU TIL BRUKERTEST 66

ORDLISTE

Forkortelser

FMC – Fuel Moisture Content, mengden fuktighet i trevirke.

TTF – Time To Flashover, tid til overtenning (overtenning er fase av brannutvikling hvor brannen er ute av kontroll).

CPU – Central Processing Unit, hovedprosessor på datamaskin.

IFD – Indoor Fire Development, risikobegrep for innendørs brannrisiko.

API – Application Programming Interface, programmeringsgrensesnitt som muliggjør datautveksling mellom ulike programvarer.

MET – Meteorologisk institutt.

XAML – Extensible Application Markup Language, brukes til å definere brukergrensesnitt.

UI – User Interface, brukergrensesnitt.

Begrep

Brannrisiko/brannfare – Sannsynlighet for brann og konsekvenser dersom brann oppstår.

DYNAMIC – "Reducing Fire Disaster Risk Through Dynamic Risk Assessment and Management", forskningsprosjekt ved Høgskulen på Vestlandet.

Mobil – Brukes i denne rapporten om smarttelefoner.

Skrivebord **–** Brukes i denne rapporten om både bærbare og stasjonære datamaskiner. Grunnen er hyppig bruk av begrep i forbindelse med programvare, for å skille fra smarttelefoner og nettbrett.

Kryss-plattform – Flere plattformer kan ta i bruk samme kode som er skrevet i én kodebase.

Plattform – Brukes i denne rapporten både om ulike typer enheter som mobil, skrivebord og nettbrett, og om operativsystemer som for eksempel Android, iOS og Windows.

Proxy Server – Mellomtjener mellom en enhet og servere på internett.

Fog Computing – Tåkedatabehandling, utfører databehandling og lagring nærmere datakilden.

1 INNLEDNING

1.1 Kontekst

Storbranner (brann i flere bygninger) utgjør et økende problem i Norge og resten av verden, hovedsakelig på grunn av klimaendringer og gjengroing i randsonen mellom natur og bebyggelse (Røhr-Staff, 2019). Perioder med tørke og kraftig vind kan føre til rask brannspredning og omfattende ødeleggelser, spesielt i områder med tett trehusbebyggelse. Lærdalsbrannen i 2014 viste hvilke konsekvenser en storbrann kan medføre i et slikt miljø (NRK TV, 2014). Sammen med Flatangerbrannen samme år, utgjorde disse hendelsen noen av de mest omfattende brannene i Norge i antall tapte bygninger siden 1923 (Røhr-Staff, 2019). For å unngå tap av liv, og hindre materielle ødeleggelser, er det viktig å ha et effektivt system for varsling av brannrisiko. Med brannrisiko menes forhøyet sannsynlighet for brann, samt alvorlige konsekvenser dersom en brann skulle oppstå. Brannfare beskriver noe av det samme som brannrisiko, men knyttes opp mot tilstanden til farekilden.

Katastrofebranner i land som Australia, Canada, USA og ved Middelhavet viser at brannrisikoen øker på globalt nivå og understreker behovet for bedre beredskap og forståelse av risikofaktorer (Høgskulen på Vestlandet, 2022). Tverrfaglig forskning er nødvendig for å redusere risikoen for katastrofebranner og bedre forstå hvilke faktorer som bidrar til dem. Forskere arbeider med å utvikle kunnskap om storbranner, risikoreduserende tiltak og dynamiske varslingssystemer. Bruk av teknologi og utvikling av system for varsling kan bidra til bedre og mer strukturert informasjon om innendørs brannrisiko. Dette kan være et nyttig verktøy i det forebyggende arbeidet mot brann i Norge.

1.2 Prosjekteier

Prosjekteier er forskningsprosjektet "*Reducing Fire Disaster Risk Through Dynamic Risk Assessment and Management (DYNAMIC)*". Forskningsprosjektet (heretter kalt DYNAMIC) tilhører Høgskulen på Vestlandet (HVL). Gjennom tverrfaglig forskning skal DYNAMIC bidra til å bedre forstå parametere som leder til storbranner. Et av målene i prosjektet er å utvikle modeller og programvare for å identifisere og varsle kommende risikotopper (DYNAMIC, 2021). Prosjekteier DYNAMIC representeres i dette arbeidet ved stipendiat Ruben Dobler Strand.

1.3 Motivasjon

Motivasjonen for dette bachelorprosjektet var å bidra til effektiv varsling av brannrisiko som muliggjør tiltak for å forebygge alvorlige konsekvenser ved en eventuell brann. En applikasjon på flere plattformer kan kommunisere brannfaren til ulike brukere, slik at risikoreduserende tiltak kan gjennomføres. For eksempel kan brannvesenet strategisk plassere materiell eller enheter i felt for å redusere utrykningstid til risikoutsatte områder. Privatpersoner kan gjøres bevisst på og redusere bruk av åpen ild i eget hjem i perioder med høy risiko.

1.4 Modell for brannrisiko

En brannrisikomodell for trehus ble utviklet av Torgrim Log (Log, 2019). Modellen predikerer innendørs relativ luftfuktighet og fuktighetsinnholdet i trebrensel (FMC) for innendørs trevirke. Dette gjøres basert på målinger og forutsigelser av utendørs relativ luftfuktighet og temperatur. Den modellerte FMC er korrelert med *tiden til overtenning* (TTF) (Kraajeveld, 2016). TTF er overgangstiden mellom vekstperioden og den fullt utviklede fasen i brannutvikling, som indikerer uholdbare forhold (Karlsson og Quintiere, 2000). TTF er vanligvis i størrelsesorden minutter og avtar med synkende FMC, hvor lav TTF tilsvarer økt risiko for storbrann. For tett trehusbebyggelse vil dette indikere både økt sannsynlighet for at et branntilløp utvikler seg til en fullt utviklet brann, samt den videre utviklingen av brannen til andre bygg (konsekvens).

1.5 Problembeskrivelse og mål

Forskningsprosjektet DYNAMIC har et mål om å utvikle et varslingssystem for storbrannfare i trehusbebyggelse. Tidligere arbeid inkluderer blant annet utviklingen av en kryssplattform-applikasjon til mobil for beregning og varsling av brannrisiko (Fisketjøn, Hussain og Svendal, 2022). Et utgangspunkt for prosjektet som beskrives i denne rapporten har vært videreutvikling av denne tidligere løsningen ved hjelp av kryssplattform-rammeverk for mobil og skrivebord. Hovedmålet har vært å ferdigstille applikasjonen og gjøre den mer robust, samt videreutvikle kommunikasjon av risiko mot ulike brukergrupper. Disse brukergruppene inkludere brannvesenet og privatpersoner. Videre har det vært viktig å utforske fordeler og ulemper med kryssplattform-utvikling, og hvorvidt dette bør være en del av strategien for en applikasjon som den DYNAMIC trenger.

1.6 Forskningsspørsmål

Med utgangspunkt i videreutviklingen av mobilapplikasjonen fra tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022), ble det utarbeidet tre forskningsspørsmål.

FS1: Kan rammeverket .NET MAUI brukes til å utvikle en kryssplattform-applikasjon for mobil og skrivebord som gir informasjon om brannrisiko i trehus tilpasset plattform og ulike brukere?

Ønsket utfall for prosjektet er å utvikle en applikasjon som tar i bruk kryssplattform-teknologi for å levere informasjon om brannrisiko i trehus på flere plattformer. Grensesnitt må tilpasses plattform og dermed også brukere, og risiko/farenivå må presenteres på en måte som gir brukeren verdi. For å oppnå dette må applikasjonen være lett å bruke, og det trengs visualiseringsalternativer for risiko/farenivå. I utgangspunktet vil skrivebordapplikasjon benyttes av brannvesenet, mens mobilapplikasjon vil tas i bruk av både brannvesenet og privatpersoner.

FS2: Hvilken grad av kryssplattform er det mulig å oppnå ved bruk av .NET MAUI for aktuell applikasjon?

Graden av kryssplattform bestemmes av hvor stor andel av kildekoden utvikleren skriver som må tilpasses ulike plattformer. Ved 100% kryssplattform gjøres ingen tilpasninger,

rammeverkets egenskaper for kompilering til hver plattform er nok til å oppnå en fungerende applikasjon alle steder. Graden av kryssplattform må også sees i lys av flere faktorer som utviklerens erfaring, kompromisser som er inngått og kvalitet av endelig resultat.

FS3: Er det hensiktsmessig å kopiere arkitektur og kildekode, eller starte utvikling fra bunn ved overgang til kryssplattform-rammeverk?

Applikasjonen skal være en videreutvikling av tidligere arbeid, men skal ta i bruk nytt rammeverk. Det er derfor nødvendig at fremgangsmåte for å ta i bruk dette blir vurdert før, under og etter utvikling av løsningen. På den måten kan det gis et svar på hvordan man skal gjennomføre en slik overgang til et nytt rammeverk på best mulig måte.

1.7 Oppbygging av rapporten

Det antas at leseren har teknologisk kompetanse. Struktur, beskrivelser og løsning har en teoretisk og metodisk tilnærming som krever innsikt i IT-tekniske fagområder.

Bacheloroppgaven er delt inn i følgende kapitler:

Kapittel 2 er en innføring i prosjektet. Kapittelet har som hensikt å gi leseren en bedre forståelse av bakgrunn og ressurser som krevdes for å fullføre prosjektet.

Kapittel 3 omhandler gjennomføringen av prosjektet. Kapittelet redegjør for prosjektmetodikk, struktur og hvordan prosjektet ble gjennomført. En sentral del av dette er beskrivelse av designet, viktige aktiviteter og teknologisk anvendelse i gjennomføringen.

Kapittel 4 handler om detaljert løsning for produktet. Kapittelet gir et innblikk i arkitektur, kildekode, programflyt og utseende for applikasjonen.

Kapittel 5 beskriver resultater fra evaluering av prosjektet.

Kapittel 6 inneholder diskusjon rundt prosjektet som en helhet. Planlegging, gjennomføring og ferdig produkt diskuteres.

Kapittel 7 handler om konklusjon og videre arbeid. Resultater settes i sammenheng med opprinnelige forskningsspørsmål. Det vil også gis kommentarer og anbefalinger med tanke på videre arbeid med problemstillingen.

2 PROSJEKTBESKRIVELSE

2.1 Praktisk bakgrunn

Forskningsprosjektet DYNAMIC har til hensikt å utvikle brannrisikomodeller som skal inngå i et varslingssystem for husbrannfare med fokus på trehus. For å indikere brannfare i trehus benyttes per i dag en modell som estimerer fuktinnhold i innvendig trepanel, herunder spesielt vegger og tak (Log, 2019). Dersom forholdene i trehusene er svært tørre, så vil dette kunne medføre hurtig brannutvikling og økt sannsynlighet for brannspredning til nabobygg. Tidligere arbeid har sett på hvor nøyaktige risikoindikasjoner offentlige værdata gir, og hvordan disse indikasjonene kan leveres til sluttbrukere.

2.1.1 Tidligere og relatert arbeid

I 2019 utførte Stokkenes (Stokkenes, 2019) implementering og validering av modellen (Log, 2019). Dette var den første implementasjonen av modellen, og risikoindikasjoner ble beregnet basert på værdata fra observasjoner og prognoser, herunder utendørs relativ fuktighet og temperatur. Undersøkelsen konkluderte med at Log sin modell (Log, 2019) kan brukes til å gi indikasjoner på brannrisiko. I tidligere arbeid har begrep *Time To Flashover* (TTF) vært benyttet til angivelse av risiko.

I 2021 utviklet Halderaker og Evjenth et web-basert system for brannrisiko (Halderaker og Evjenth, 2021). Dette var Java-basert og tok i bruk sky og mikro-tjenester. Web-tjenesten ble evaluert med tanke på prestasjon og nøyaktighet, og det ble konkludert med at CPU-bruk for applikasjonen var neglisjerbar. Det kom frem av arbeidet at værprognoser for opptil tre dager frem i tid kunne brukes for å indikere den kommende risikoen (husbrannfaren). Dette ble gjort ved å sammenligne beregnet brannrisiko basert på prognoser og observasjoner.

I 2022 utviklet Fisketjøn, Hussain og Svendal en mobilapplikasjon (Fisketjøn, Hussain og Svendal, 2022). Denne var basert på *Edge Computing* og skulle gjøre indikasjoner på kommende brannrisiko enkelt tilgjengelig for brukere ved å beregne og presentere dette lokalt på brukerens mobil. Brukeren var nødt til å oppgi en ID som måtte hentes fra Meteorologisk institutt sine sider, og deretter legge til lokasjoner basert på koordinater eller GPS-system på

mobilen. For hver innlagt lokasjon ble brannrisiko for inneværende dag, samt tre dager frem i tid presentert for brukeren. Brannrisiko ble angitt ved bruk av begrep *Indoor Fire Development* (IFD).

2.1.2 Initielle krav

De initielle kravene for bachelorprosjektet var å lage en kryssplattform-applikasjon for mobil og skrivebord, med formål å varsle brannrisiko i trehus for angitte lokasjoner. Applikasjonen skulle inneholde tilpasninger på hver plattform der det var nødvendig. Skrivebordapplikasjonen var tiltenkt for bruk i brannvesenet, mens mobilapplikasjonen var aktuell for privatpersoner i tillegg. Derfor var det nødvendig å evaluere begrepsbruk for brannrisiko for å best mulig presentere denne til brukere med ulik bakgrunn. Videre var det også et krav at applikasjonen skulle være en videreutvikling av mobilapplikasjonen som ble utviklet i tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022), og bruke eksisterende kildekode for brannrisikomodell.

2.1.3 Initiell løsningsidé

Applikasjonen ble utviklet ved hjelp av kryssplattform-teknologi. Formålet med applikasjonen er å muliggjøre at brukere på mobil og skrivebord kan hente informasjon om brannrisiko i trehus for bestemte lokasjoner. Det ble foreslått at skrivebord skulle ha flere alternativer for visualisering av brannrisikoverdier. Disse alternativene kunne inngå i mobilapplikasjonen dersom det fungerte visuelt.

2.2 Ressurser

Beregning av brannrisiko er avhengig av risikomodell og værdata. Værdata blir hentet fra Meteorologisk institutt (heretter kalt MET) sine programmeringsgrensesnitt (Meteorologisk institutt, u.å. a). Modellen for beregning er Log sin modell (Log, 2019), utviklet gjennom DYNAMIC.

Microsoft Teams ble brukt for å arrangere møter mellom prosjektmedlemmene, veileder og prosjekteier. I tillegg ble denne kanalen brukt til diskusjon og deling av filer. Teknisk veiledning ble gitt av veileder professor Lars Michael Kristensen, mens spørsmål som angikk DYNAMIC

og Log sin modell ble besvart av stipendiat Ruben Dobler Strand. For planlegging av prosjektarbeid ble det brukt et Gantt-diagram. For å sørge for effektiv utvikling og feilsøking ble det gjennom DYNAMIC anskaffet en Android-mobil.

3 DESIGN AV PROSJEKTET

3.1 Forslag til løsning

Det finnes ulike teknologier som muliggjør tjenester på både mobil og skrivebord. Teknologien som ble valgt bestemte strategi for utvikling. Det viktigste å ta hensyn til var hvilken grad av tilpasning som var nødvendig på plattformene og hvor mye tid som kunne spares ved at kildekode automatisk ble gjenbrukt mellom plattformene. Ulike løsninger for å utvikle en applikasjon på flere plattformer ble vurdert.

3.1.1 Alternativ 1: MAUI

.NET MAUI (Multi-platform App UI) er et rammeverk for å bygge kryssplattform-applikasjoner med C# og .NET (Microsoft, 2023a.). Det muliggjør utvikling av applikasjoner som kan kjøre på Android, iOS, macOS og Windows med én enkelt kodebase. MAUI er åpen-kildekode og er en videreutvikling av Xamarin.Forms. For å bygge apper bruker utviklere XAML for å definere brukergrensesnittet og C# for å implementere applikasjonslogikken.

MAUI er en integrert del av .NET. Det finnes flere gode verktøy for .NET-applikasjoner som for eksempel Community Toolkit, samt et stort miljø av utviklere. Innebygd i .NET, og derfor også MAUI, er en funksjon som kalles *Hot Reload*. Denne funksjonen tillater utviklere å se endringer i sanntid når de skriver XAML-kode for applikasjonens brukergrensesnitt. Det finnes dessverre ikke et visuelt redigeringsverktøy for brukergrensesnitt i .NET MAUI. Det er også et relativt nytt rammeverk, lansert i mai 2022, noe som kan bety færre ressurser tilgjengelig for læring og problemløsing.

3.1.2 Alternativ 2: Flutter

Flutter er et rammeverk for å bygge applikasjoner med høy ytelse og attraktivt design (Flutter, u.å.). Det er utviklet av Google og lar utviklere bygge kryssplattform-applikasjoner for iOS, Android, web og skrivebord ved hjelp av én kodebase. Flutter bruker programmeringsspråket DART, som gir utviklere muligheten til å skrive rask og enkel kode.

Flutter har en rekke innebygde kontroller og oppsett som gir utviklere en stor grad av frihet når det gjelder design av brukergrensesnittet. Utviklere kan lage sine egne kontroller og tilpasse dem etter behov. Flutter har også et stort samfunn av utviklere som bygger og deler tilpassede kontroller og pakker. *Hot Reload* er også en funksjon i Flutter.

3.1.3 Alternativ 3: Individuell utvikling

Et alternativ var å utvikle flere separate applikasjoner, én for hver plattform. Fordelen med dette er 100% kontroll over tilpasninger på hver plattform, og gjerne solide og etablerte utviklingsverktøy for hver av dem. Ulempen er økt tidsbruk ved utvikling samt fremtidige oppdateringer. Det antas at et større team er nødvendig for å vedlikeholde noe slikt.

I noen tilfeller kan det være nødvendig med slik individuell utvikling, herunder ved økt behov for sikkerhet og ytelse. Kryssplattform-rammeverk resulterer i applikasjoner som tar mer lagringsplass, kan kjøre noe tregere og det tar lengre tid før oppdateringer i de underliggende operativsystemene implementeres (Decode, 2023).

3.1.4 Diskusjon av alternativene

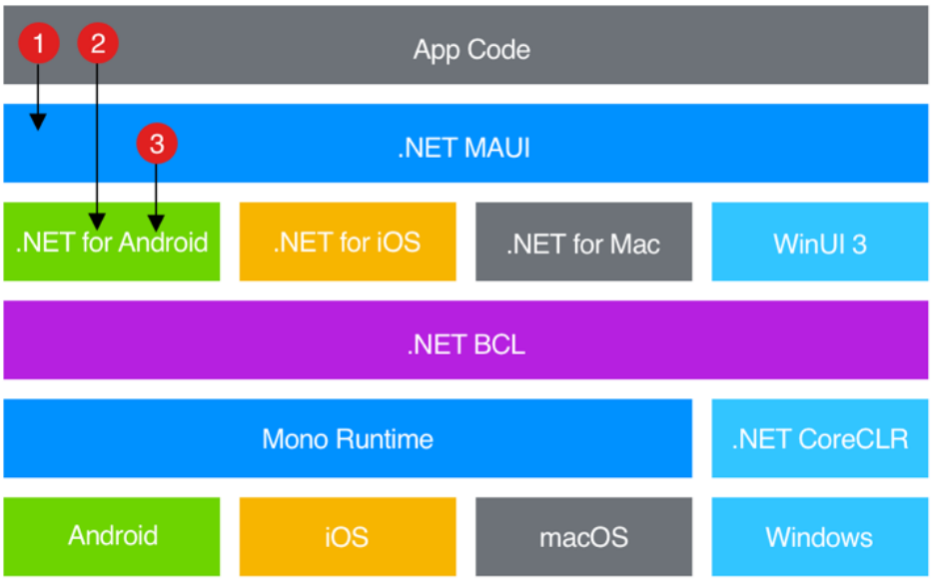
Hver av de introduserte alternativene over har sine fordeler og ulemper. Dersom prosjektgruppen skulle tatt i bruk individuell utvikling på flere plattformer ville det ikke vært tilstrekkelig med tid til å utvikle et ferdig produkt på alle. Flutter er moderne og tilbyr et system for å lage egne skjermelementer som er fleksibelt. Dersom man har god kjennskap til Google sine tjenester innen programutvikling ville dette vært et godt valg.

MAUI er ikke like etablert som Flutter (lansert i mai 2017), men har likevel noen fordeler. Microsoft har mange tjenester og verktøy innen programutvikling, herunder Visual Studio som er et integrert utviklingsmiljø spesifikt for .NET-løsninger. Fordi MAUI er etterfølgeren til Xamarin var det også mer sannsynlig at kildekode og arkitektur kunne gjenbrukes fra tidligere mobilapplikasjon (Fisketjøn, Hussain og Svendal, 2022).

3.2 Valgt løsning

Basert på de diskuterte alternativene ble det besluttet å benytte .NET MAUI i det videre arbeidet. Det ble lagt vekt på mulighet for gjenbruk fra tidligere mobilapplikasjon. Gjennom prosjektbeskrivelsen fra DYNAMIC var det også anbefalt å velge dette rammeverket. Prosjektmedlemmene deltok i et emne som omfattet programutvikling i .NET parallelt med bachelorprosjektet. Dette gjorde at medlemmene fikk relevant faglig kompetanse om rammeverket som inngikk i prosjektet.

Figur 1 viser høynivå-arkitektur for .NET MAUI (Microsoft, 2023a). .NET versjon 6 eller høyere tilbyr plattform-spesifikke rammeverk for å lage applikasjoner: .NET for Android, .NET for iOS, .NET for macOS og Windows UI 3. Disse rammeverkene har alle tilgang til det samme .NET *Basis Class Library* (BCL) som abstraherer bort detaljene fra hver plattform slik at de aksesseres med lik kode på alle plattformer. Koden kommuniserer med .NET MAUI-grensesnittet (pil 1), som deretter kommuniserer med den aktuelle plattformen koden har blitt kompilert til (pil 3). Det er mulig å kommunisere direkte med grensesnitt for hver plattform (pil 2) ved behov.



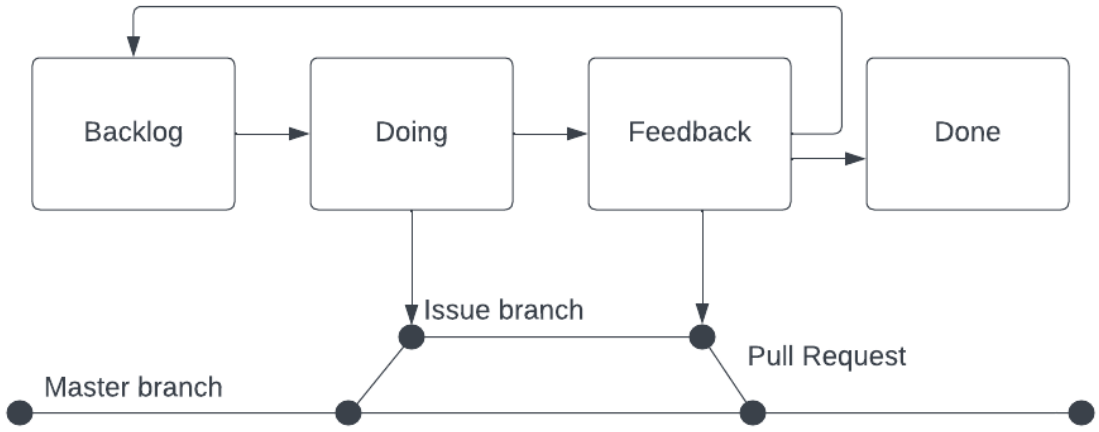
Figur 1. Høynivå-arkitektur for .NET MAUI (Microsoft, 2023a).

3.3 Prosjektmetodikk

3.3.1 Utviklingsmetodikk

For å sikre en strukturert og effektiv utviklingsprosess ble det valgt å bruke elementer fra metodikken Scrum. Scrum er en smidig metode hvor det er tett samarbeid mellom involverte parter om å definere krav, og hvordan disse kan deles opp og leveres i mindre deler for å oppnå prosjektmålene som er satt (Scrum, u.å.). Fordelen med metoden er at den gir økt fleksibilitet og muligheten for fortløpende tilpasninger. I praktisk anvendelse er det vanlig å dele oppgaver inn i mindre deler. Deloppgavene beskrives og planlegges løst i sprinter med en varighet fra én til fire uker.

Under hver deloppgave ble det laget flere konkrete arbeidsoppgaver i form av *Issues*. Disse ble ført opp i *Backlog*-listen i starten av hver sprint. Figur 2 viser arbeidsflyt for hvert *Issue*. Når et prosjektmedlem arbeidet på et *Issue*, ble det flyttet til *Doing*-listen, og korresponderende *Issue* og *Branch* ble opprettet på *Github*. Når arbeidet med *Issue* var ferdig, ble det flyttet til *Feedback*-listen, og det ble opprettet en *Pull-Request* for å *merge* endringene til *Master Branch*. Dersom testingen og tilbakemeldingene var vellykket, ble *Issue* flyttet til *Done*-listen. Hvis ikke, ble det returnert til *Backlog*-listen med nye krav.



Figur 2. Arbeidsflyt for en arbeidsoppgave (Issue).

3.3.2 Prosjektplan

Tidlig i prosjektet ble det laget et Gantt-diagram (vedlegg 9.1) for planlegging og strukturering av fire faser. Disse fasene tar for seg oppstartsfase, selvstudie, utvikling- og testfase og sluttfase. Fasene hadde sine egne mål og milepæler og ga en oversikt over tidsrammer. Gantt-diagrammet var fleksibelt og ble endret etter behov, ved for eksempel nye krav fra prosjekteier.

Oppstartsfasen og selvstudie henger sammen, hvor gruppen planla prosjektet og tilegnet seg nødvendig teori. I utvikling- og testfasen ble det gjennomført flere iterasjoner med utvikling. Målet med sluttfasen var å ferdigstille prosjektet, og sikre at alle ønskede resultater ble oppnådd. Det ble planlagt å få evaluering fra prosjekteier i form av demonstrasjon før siste fase av utvikling. Deretter skulle det gjennomføres testing i slutten av fasen.

3.3.3 Strategi

Det ble utarbeidet en strategi for hvordan tidligere arbeid (Fisketjøn, Hussain og Svendal, 2022) kunne tas i bruk. Applikasjonen som beskrives i denne rapporten er en videreutvikling av dette arbeidet og inkluderer nytt rammeverk, utvidelse til skrivebord og en del nye funksjoner. Del 1 av utviklingsfasen gikk til å undersøke hvilke deler av mobilapplikasjonen som kunne gjenbrukes, noe som fremkommer av Gantt-diagram (vedlegg 9.1). Modeller for data så ut til å kunne overføres i sin helhet. Det ble forsøkt å overføre deler av logikken for brukergrensesnitt, men med mindre suksess. Derfor ble det besluttet at dette skulle bygges i størst mulig grad fra bunn for maksimal kontroll av kildekode. Arkitektur og designmønster skulle videreføres, slik at programflyt og innhold kunne ta inspirasjon fra forrige applikasjon.

3.3.4 Risikovurdering

I begynnelsen av prosjektet ble det gjennomført en risikovurdering (vedlegg 9.2). Denne identifiserte mulige risikoer, og vurdering av sannsynligheten for at de kunne inntreffe. Det ble også gjennomført en evaluering av hver risiko. Etter kartleggingen ble hver risiko plassert i en risikomatrise (vedlegg 9.3) for å vurdere om det var nødvendig å iverksette tiltak. Å plassere risikoene i en risikomatrise gjør det mulig å prioritere tiltak og ressurser for å håndtere de mest kritiske risikoene. For eksempel ble det jobbet iterativt og smidig ved å ta i bruk prinsipper fra

Scrum, noe som begrenset konsekvensen av forsinkelser (se hendelse 2 i risikomatrise vedlegg 9.2).

3.4 Evalueringsplan

Evalueringen av resultatene var kritisk for å innhente innsikt som kunne bidra til å forbedre løsningen. Det ble ikke planlagt en omfattende brukertest av applikasjonen, da testing i forbindelse med tidligere arbeid (Fisketjøn, Hussain og Svendal, 2022) allerede hadde gitt klare indikasjoner på kravene fra brukere i ulike roller. Stipendiat Ruben Dobler Strand hadde kjennskap til disse kravene, og det ble derfor besluttet at evalueringen skulle gjennomføres i form av to omganger med testing og tilbakemelding med ham.

Den første evalueringen, omtalt som demonstrasjon, ble planlagt med fokus på det estetiske aspektet, begrepsbruken og organiseringen av brukergrensesnittet. Eventuelle forslag til forbedringer fra prosjekteier kunne arbeides med til neste fase. Den neste fasen innebar en brukertest der programflyt og brukeropplevelse skulle evalueres. I etterkant av brukertesten var det planlagt å samle tilbakemeldinger og evalueringer gjennom et intervju med forberedte spørsmål om utseende og brukeropplevelse. Denne tilnærmingen til evaluering ble valgt for å sikre en grundig forståelse av brukeropplevelsen og for å identifisere mulige områder for forbedring.

4 DETALJERT LØSNING

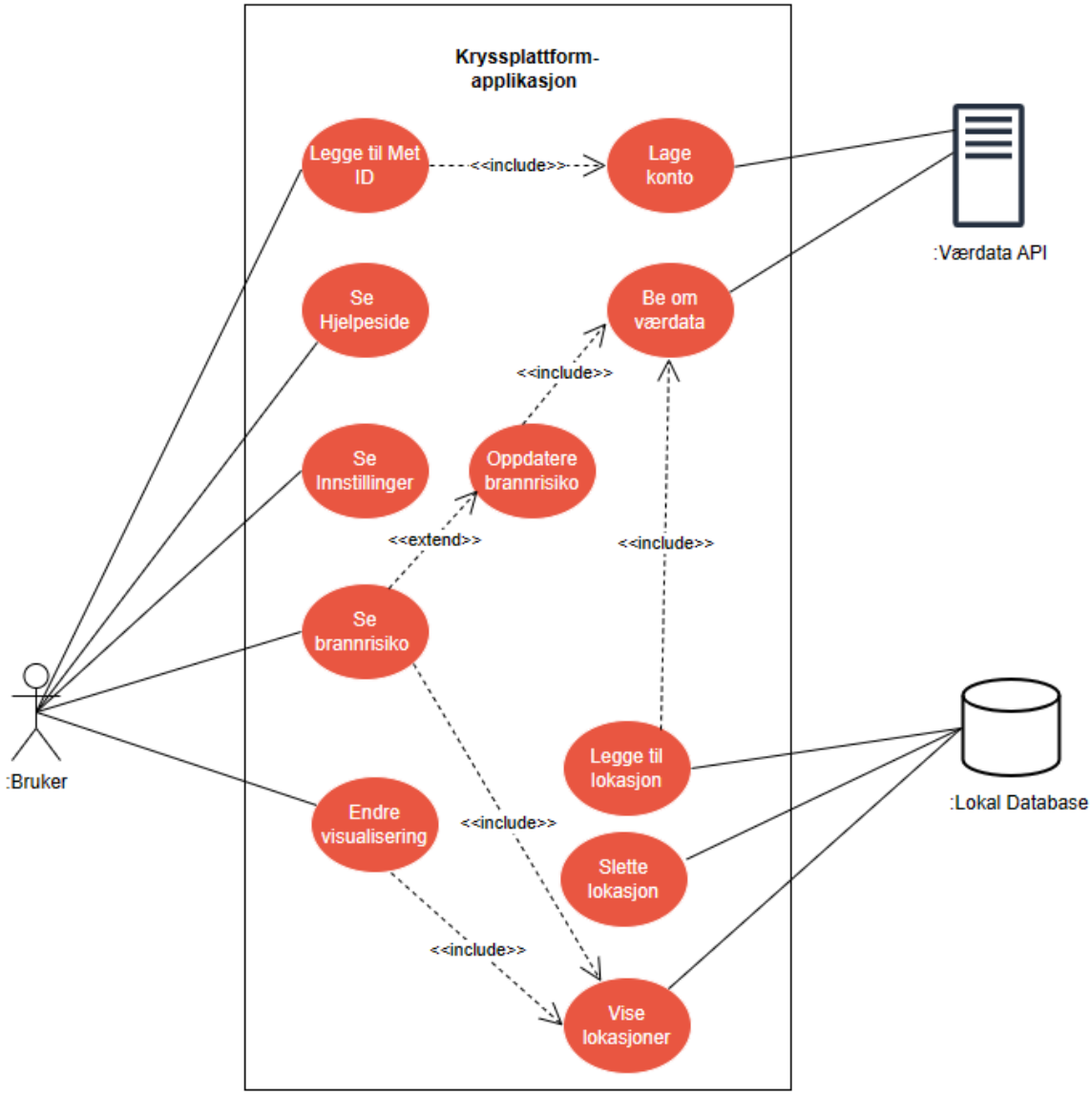
I dette kapittelet vil den omfattende løsningen for kryssplattform-applikasjonen bli grundig beskrevet. For å visualisere systemarkitekturen og designkomponentene, benytter alle diagrammer i kapittelet *Unified Modeling Language* (UML) som en standard metode for modellering. Komponenter og brukergrensesnitt forklares med eksempler fra kildekode og bilder fra kjøring av applikasjonen.

4.1 Brukstilfellediagram

Brukstilfellediagram (Figur 3) i UML er en visuell modelleringsteknikk som anvendes for å fremstille krav og funksjonaliteten til kryssplattform-applikasjonen. Den primære aktøren i applikasjonen er definert som *Bruker*, mens *Værdata API* og *Lokal Database* fungerer som støtteaktører og leverer tjenester til applikasjonen.

Aktører:

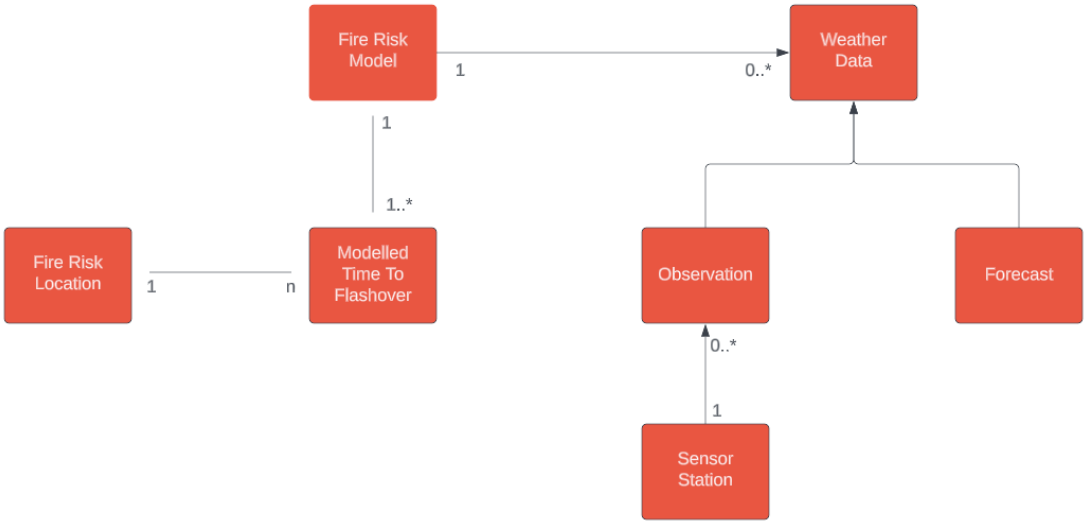
1. *Bruker*: Person som samhandler med applikasjonen.
2. *Værdata API*: En ekstern tjeneste som levere værdata fra Frost og MET.
3. *Lokal Database*: Henter og lagrer informasjon om lokasjoner lokalt på enhet.



Figur 3. Brukstilfellediagram.

4.2 Domenemodell

En domenemodell er en konseptuell modell som representerer de viktigste konseptene og relasjonene innenfor et problemområde (SAFe, 2023). Modellen brukes for å kartlegge hvordan applikasjonens logikk skal være. Figur 4 viser hvordan konseptene og relasjonene fungerer i applikasjonen.



Figur 4. Domenemodell.

Domenemodellen har to sammensetninger, én fra brannrisikomodell til brannrisiko og den andre fra sensorstasjon til observasjon. For eksempel kan ikke brannrisiko eksistere uten en brannrisikomodell. Avhengig av hvilken type værdata som behøves, kan værdata enten være en observasjon eller en prognose. Forholdet mellom brannrisiko og lokasjon indikerer at en lokasjon forventer brannrisikoer for n dager, der n er satt til fire i dagens løsning (inneværende dag og tre dager frem i tid).

4.3 Designmønster

Designmønstre, eller arkitektoniske mønstre, blir brukt i moderne applikasjoner for å skape et nødvendig skille mellom logikk og brukergrensesnitt (Stonis, 2022). Disse er viktige fordi de gir velprøvde løsninger på vanlige problemer og utfordringer i programvareutvikling. Ved å dele inn kildekode etter ansvarsområder bidrar designmønstre til å skape struktur, forutsigbarhet og enklere vedlikehold. For applikasjonen ble *Model-View-ViewModel* (MVVM) benyttet.

4.3.1 MVVM

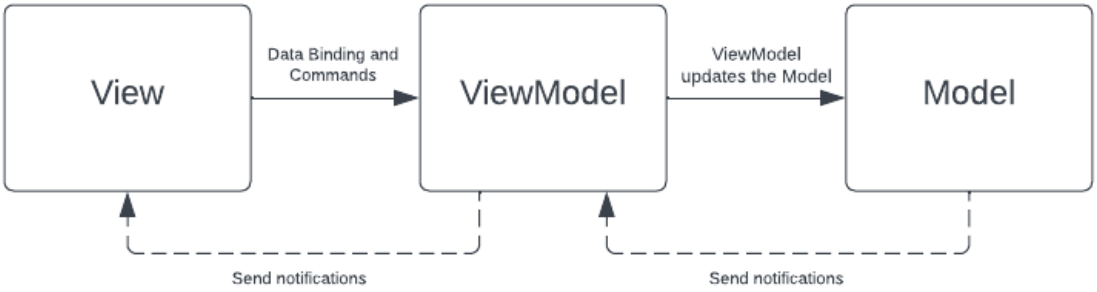
MVVM er et designmønster som brukes i programvareutvikling for å skille grafiske brukergrensesnitt fra forretningslogikk og datamodeller (Stonis, 2022). Målet er å gjøre koden mer modulær, enkel å vedlikeholde og testbar. Forrige prosjekt brukte MVVM, og som en del av strategien for å videreutvikle applikasjonen var det hensiktsmessig å videreføre dette (Fisketjøn, Hussain, Svendal, 2022).

Model representerer data og forretningslogikk.

View håndterer brukergrensesnittet og visning av data.

ViewModel fungerer som en bro mellom modellen og visningen, og eksponerer data og kommandoer som kan bindes til visningselementer.

Figur 5 (Stonis, 2022) viser forhold mellom *Model, View* og *ViewModel*. MVVM ble først introdusert av arkitekter i Microsoft og er en integrert del av Microsoft sine teknologier. (Smith, 2009). Bruken av MVVM fører til en bedre organisering av kode og et klarere skille mellom ulike ansvarsområder, noe som bidrar til at *Single Responsibility* – prinsippet overholdes. Dette er et viktig prinsipp innen programvareutvikling, som skal fasilitere for enklere testing og vedlikehold (Martin, 2006).



Figur 5. Forhold mellom View, ViewModel og Model i MVVM (Stonis, 2022).

4.3.2 Data Binding

Data Binding i .NET MAUI muliggjør effektiv kobling av data og brukergrensesnitt ved å eliminere behovet for eksplisitt kodeoppdatering (Stonis, 2022). *Data Binding* er sentralt i MVVM-mønsteret og forenkler synkronisering av data mellom *ViewModel* og *View*. Figur 6 viser hvordan egenskapen *IsBusy* fra *ViewModel* bindes til en *ActivityIndicator* i *View*. På denne måten kan *IsBusy* settes til *true* når langsomme operasjoner gjøres i *ViewModel* og til *false* når de er ferdige, som vist i figur 7. *ActivityIndicator* vil da vise at applikasjonen laster. Det samme prinsippet blir brukt for all data som skal vises i applikasjonen; *Model* og *ViewModel* endrer tilstand mens *View* endrer fremvisning.

```
1   <ActivityIndicator IsRunning="{Binding IsBusy}" ... />
```

Figur 6. Binding av IsBusy til ActivityIndicator i AddLocationPage.

```
1   async void GetCurrentLocationClickedAsync()
2   {
3     IsBusy = true;
4     try
5     {
6       GeolocationRequest request = new GeolocationRequest(GeolocationAccuracy.Medium, TimeSpan.FromSeconds(10));
7       var _cancelTokenSource = new CancellationTokenSource();
8       var location = await Geolocation.Default.GetLocationAsync(request, _cancelTokenSource.Token);
9       ...
10    }
11  ...
12  IsBusy = false;
13  }
```

Figur 7. IsBusy settes til true i AddLocationViewModel når geolokasjon hentes.

4.3.3 Community Toolkit MVVM

CommunityToolkit.Mvvm-pakken er et moderne, raskt og modulært MVVM-bibliotek for bruk i .NET applikasjoner (Microsoft, 2023b). Pakken er rettet mot .NET Standard og alle kjøretidsversjoner av .NET. Pakkens grensesnitt er likt for alle versjonene og egner seg derfor godt til kryssplattform-applikasjoner. I applikasjonen brukes flere funksjoner fra pakken for å eliminere unødvendig og repetitiv kode, populært kalt *Boilerplate*-kode.

Attributtet *ObservableProperty* fra *CommunityToolkit.Mvvm* er brukt for å forenkle deklarering av egenskaper som kan observeres. Figur 8 viser bruk av attributtet og figur 9 viser generert kode. I tillegg skapes *SetProperty*-metoden automatisk og den genererte koden blir optimalisert slik at den faktisk er raskere enn dersom den ble skrevet eksplisitt av utvikleren (Microsoft, 2023b). Den genererte koden ligger i en egen fil.

```
1   [ObservableProperty]
2   private string visState = "bar";
```

Figur 8. Bruk av ObservableProperty i MyLocationsViewModel.

```
1   public string VisState
2   {
3       get => visState;
4       set => SetProperty(ref visState, value);
5   }
```

Figur 9. Autogenerert kode ved bruk av ObservableProperty.

Et annet attributt fra *CommunityToolkit.Mvvm* er *RelayCommand* som genererer kode for *Command*-metoder som kan kalles fra *View*. Figur 10 viser bruk av attributt mens figur 11 viser generert kode.

```
1  [RelayCommand]
2  async void DeleteLocation(FireRiskLocation location)
3  {
4    ...
5    await databaseService.DeleteLocationAsync(location);
6    ...
7  }
```

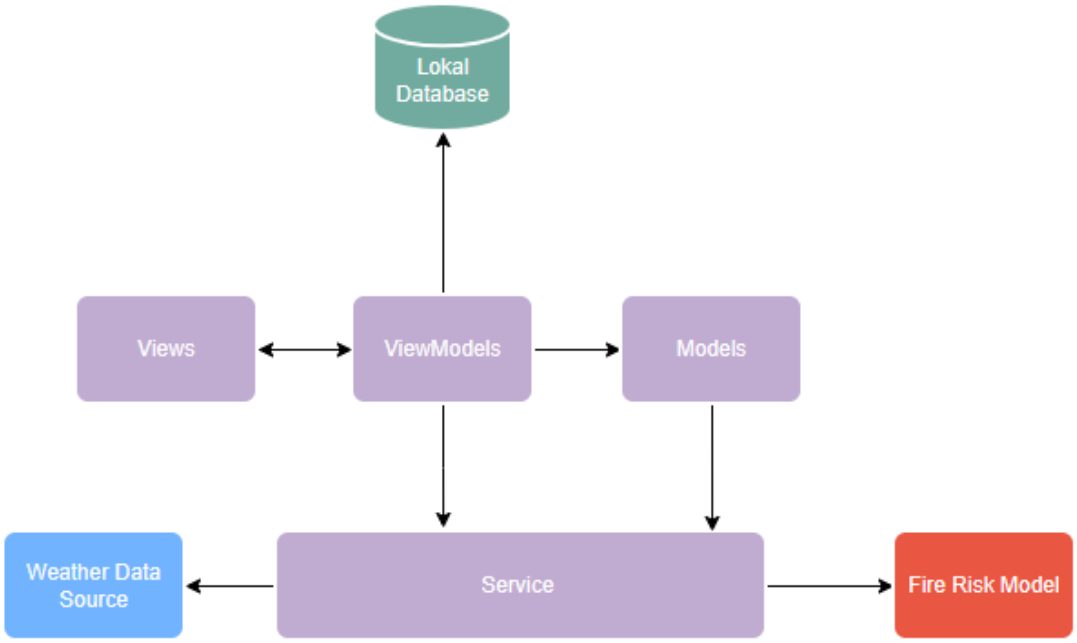
Figur 10. Bruk av RelayCommand i MyLocationsViewModel.

```
1  private RelayCommand? deleteLocationCommand;
2
3  public IRelayCommand DeleteLocationCommand =>
4         deleteLocationCommand ??= new RelayCommand(DeleteLocation);
```

Figur 11. Autogenerert kode ved bruk av RelayCommand.

4.4 Programvarearkitektur

Figur 12 viser arkitekturen for applikasjonen, som er lik for hver plattform applikasjonen kompileres til. Programvarearkitekturen bygger videre på MVVM-designmønsteret med tjenester for beregning av brannrisiko og lagring i database. Service inneholder tjenesteklienter mellom *ViewModels* og kilder for værdata og modell for beregning av brannrisiko. Detaljert prosjektstruktur og programvarearkitektur finnes i *systemdokumentasjon* (vedlegg 4, kapittel 2-3).



Figur 12. Programvarearkitektur.

4.5 Værdata

Værdata og de tilhørende API-ene som benyttes for å beregne brannrisiko tilsvarer det som ble brukt i det forrige prosjektet (Fisketjøn, Hussain og Svendal, 2022). I første fase av utviklingen til ny versjon av applikasjonen ble verktøyet Postman benyttet for å undersøke om de samme forespørselen generte like resultater fra API-ene. Dette ble bekreftet, og bruk av kildene til værdata ble overført fra det tidligere prosjektet. For å få tilgang til API-ene behøves en *client-id*, referert til som MET-ID i applikasjonen.

4.5.1 Meteorologisk institutt

Meteorologisk institutt (MET) varsler været, overvåker klimaet og driver forskning (Meteorologisk institutt, 2017). MET har ansvar for å samle inn og håndtere store mengder værdata. Hovedformålet deres er å bidra til å sikre liv og verdier, samt samfunnets sikkerhet og bærekraft.

4.5.2 Frost API

Frost er et API som gir tilgang til MET sine historiske arkiver for værdata (Meteorologisk institutt, u.å. b). Disse dataene gir daglige, ukentlige og månedlige målinger av temperatur-, nedbør- og vinddata. Forrige prosjekt (Fisketjøn, Hussain og Svendal, 2022) brukte endepunktene og parameterne som er vist i henholdsvis figur 13 og 14 for å hente nærmeste målestasjon og observasjoner.

```
var request = new RestRequest("sources/v0.jsonld")
            .AddParameter("types", "SensorSystem")
            .AddParameter("elements", "air_temperature,relative_humidity,wind_speed")
            .AddParameter("geometry", $"nearest(POINT({longitude} {latitude}))");
```

Figur 13. Forespørsel til endepunkt for målestasjoner.

```
var request = new RestRequest("observations/v0.jsonld")
            .AddParameter("sources", station_id)
            .AddParameter("referencetime", period)
            .AddParameter("elements", "air_temperature,relative_humidity,wind_speed");
```

Figur 14. Forespørsel til endepunkt for observasjoner.

4.5.3 WeatherForcast API

Prognose-API, også kjent som *LocationForecast 2.0*, leverer en fullstendig værmelding for én lokasjon for opptil ni-dagers periode (Meteorologisk institutt, u.å. c). API-et har parametere som inkluderer bredde- og lengdegrad. Figur 15 viser hvordan en prognoseforespørsel ble gjort i forrige prosjekt (Fisketjøn, Hussain og Svendal, 2022).

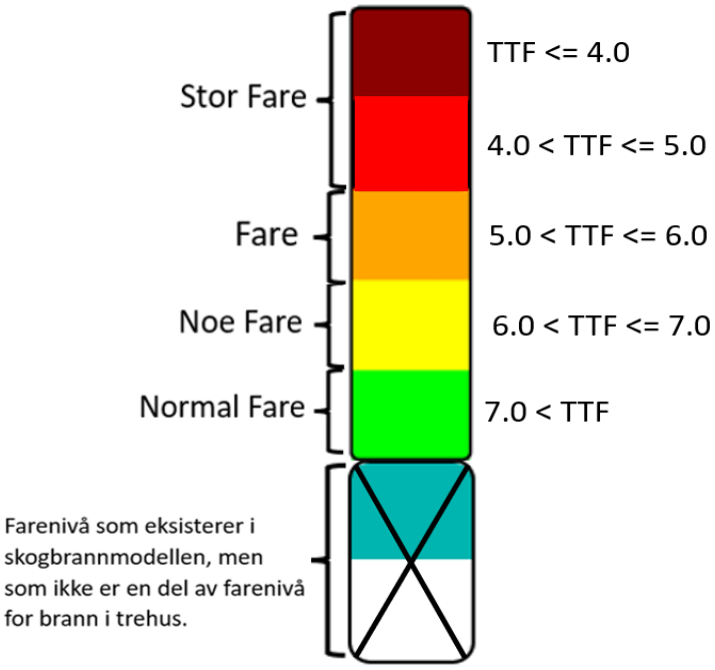
```
1   var request = new RestRequest("weatherapi/locationforecast/2.0/compact.json")
2               .AddParameter("lat", latitude)
3               .AddParameter("lon", longitude);
```

Figur 15. Forespørsel til endepunkt for prognoser.

4.6 Brannrisiko

Som nevnt i avsnitt 2.1.1 ble begrepet IFD benyttet til å kommunisere risiko i første utgave av applikasjonen fordi brukere hadde problemer med å forstå TTF (Fisketjøn, Hussain og Svendal, 2022). Gjennom vurderinger fra prosjekteier ble det bestemt at den nye applikasjonen skulle gå vekk fra begrepet IFD og heller bruke «Husbrannfare». Brannfare er et begrep som beskriver noe av det samme som brannrisiko, nemlig økt sannsynlighet for branntilløp og større konsekvenser ved en eventuell brann. Grunnen til at «Husbrannfare» ble introdusert i applikasjonen er at det gjør det lettere for brukere å koble dette til «skogbrannfare» som er et kjent begrep (Direktoratet for samfunnssikkerhet og beredskap, 2021).

Opprinnelig plan for applikasjonen var å ta i bruk samme fargespekter for risiko som indeksen for skogbrannfare. Ved første evaluering ble det bestemt at fargespekteret for brannrisiko skulle begrenses til mørkerød-til-grønn. Grunnen til dette er at det ikke gir mening å snakke om liten/ingen fare for brann i trehus, den eksisterer alltid til en viss grad. Figur 16 viser fargespekter for brannrisiko i trehus, med korresponderende TTF-verdier for hvert nivå.



Figur 16. Brannrisiko, farger og TTF-verdier i relasjon.

4.7 Datamodeller

I *Model-View-ViewModel*-arkitekturen er modeller ikke-visuelle klasser som innkapsler applikasjonens data (Stonis, 2022). Modeller representerer applikasjonens domenemodell, og inkluderer datamodeller og validering- og forretningslogikk. I applikasjonen er modellene i stor grad gjenbrukt fra tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022), med noen tilpasninger. Modellene og forretningslogikken kan deles inn i to grupper: *Fire Risk Model* og *Models.*

4.7.1 Fire Risk Model

Fire Risk Model inneholder klasser for beregning av *TTF*, basert på værdata i form av *Observation*-objekter som er vist i figur 17. Modellen som benyttes (Log T.) er tilgjengelig i C#. Det refereres til (Fisketjøn, Hussain og Svendal, 2022) for mer informasjon om modellen. Klassen *FireRisk* fra *Fire Risk Model* brukes i *WeatherClient* for å beregne brannrisiko. Grunnet nye krav til applikasjonen var det nødvendig å skille mellom beregning av brannrisiko som et heltall mellom 1-6 og som et desimaltall for TTF. Figur 18 viser grensesnittet *IFireRisk* etter disse revisjonene som la til metodene *FireRiskAsRange* og *FireRiskAsTTF*.

```
1  public class Observation
2  {
3    private DateTime referenceTime;
4    private Temperature temperature;
5    private Humidity humidity;
6    private WindSpeed? windSpeed;
7    ...
8  }
```

Figur 17. Observation-objekt til bruk i Fire Risk Model.

```
public interface IFireRisk
{
  List<FireRiskFactor> fireRiskFactorsFrom(List<Observation> observations);
  FireRiskResult fireRiskTtf(List<Observation> observations);
  List<int> FireRiskAsRange(List<Observation> observations);
  List<double> FireRiskAsTTF(List<Observation> observations);
  List<Observation> interpolateObservations(List<Observation> observations);
}
```

Figur 18. Grensesnitt IFireRisk i Fire Risk Model.

4.7.2 Models

Applikasjonsmodeller, eller bare kalt *Models*, i applikasjonen utgjør de modellene som er involvert i innhenting av data fra MET sine API-er, og endelig presentasjon i form av *FireRiskLocation*-objekter som er vist i figur 19. Disse representerer hver lokasjon brukeren har lagret i applikasjonen, og inneholder oppdatert informasjon om vind, vindretning og modellert TTF.

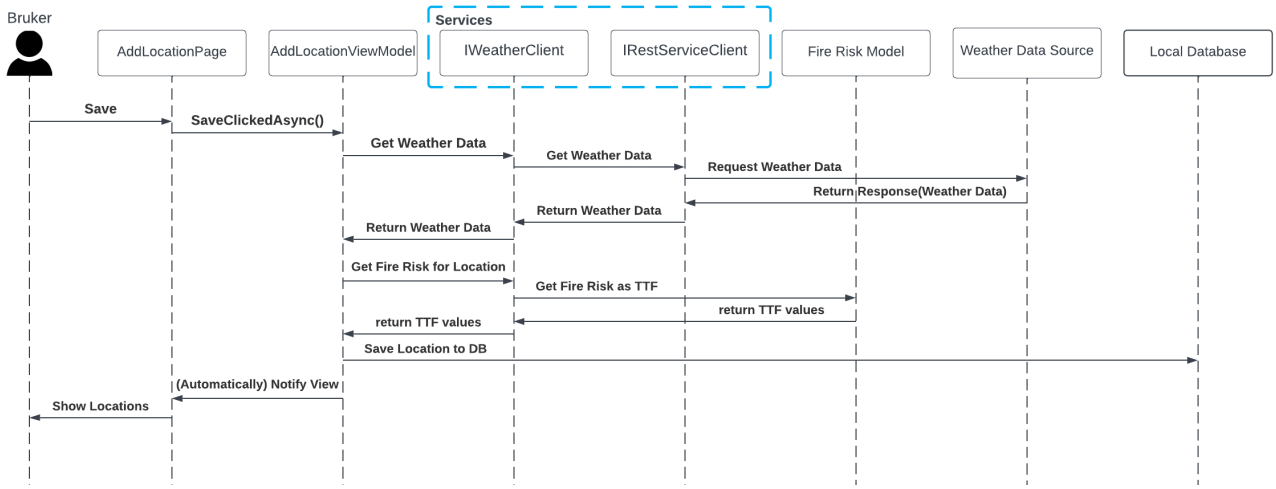
```
public class FireRiskLocation
{
    public string Name { get; set; }
    public string Latitude { get; set; }
    public string Longitude { get; set; }
    public string WeatherStationID { get; set; }
    public List<double> WindSpeed { get; set; }
    public List<double> WindGustSpeed { get; set; }
    public List<double> WindDirection { get; set; }
    public List<double> ModelledTTF { get; set; }
    public double PeakTTF { get; set; }
}
```

Figur 19. FireRiskLocation-objekt fra Models, forenklet.

4.8 Tjenesteklienter

I forbindelse med *Models* behøves forretningslogikk for å hente værdata, transformere data og gjøre beregninger slik at resultatet blir verdier som kan presenteres for brukeren. *Service*-klassene som er ansvarlig for dette er gjenbrukt fra tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022), med tilpasninger gjort for å sørge for at brannrisikoen beregnes til TTF-verdier og ikke IFD-verdier.

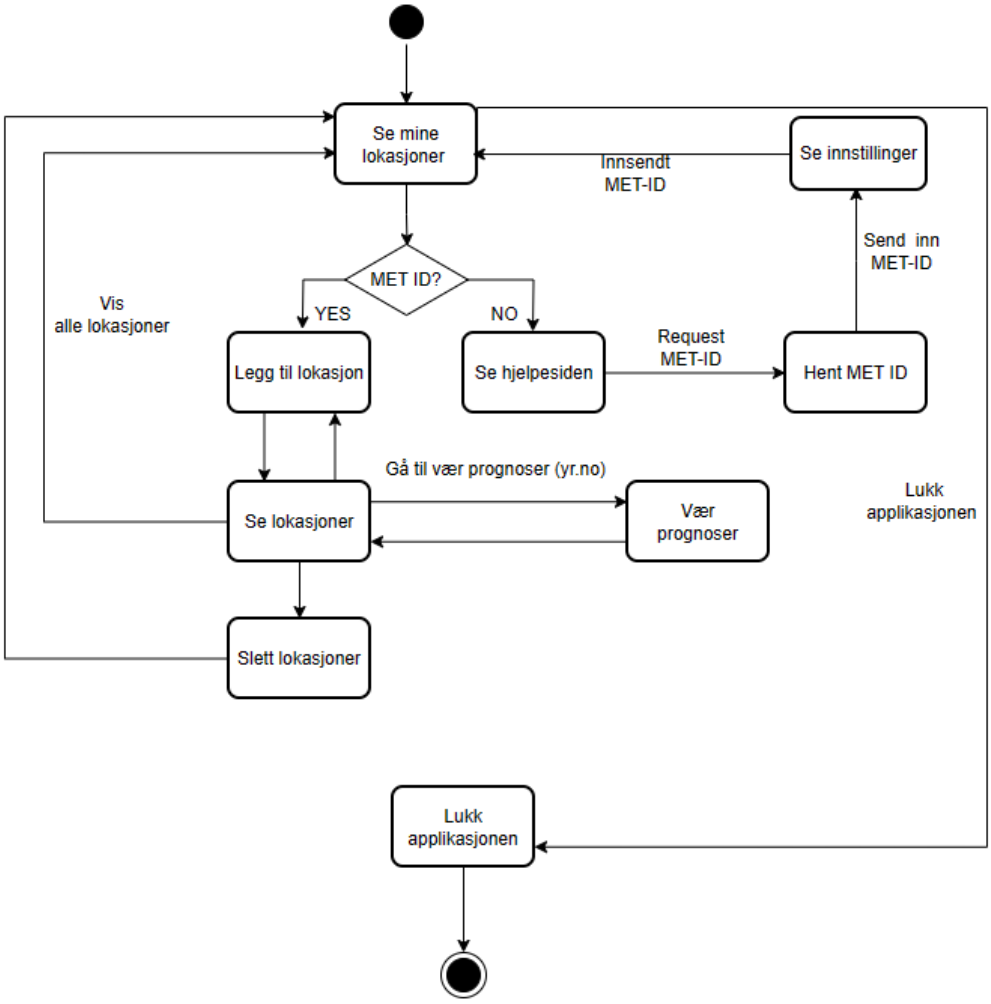
Services består av to grensesnitt, *IRestServiceClient* og *IWeatherClient* og deres implementasjoner i klassene *RestServiceClient* og *WeatherClient*. Figur 20 viser et forenklet sekvensdiagram for samhandlingen mellom klassene i Service når brukeren legger til en ny lokasjon. *Get Weather Data* involverer egentlig tre forespørsler; Nærmeste værstasjon, observasjoner og prognoser, som vist i avsnitt 4.5. All API-kommunikasjon skjer gjennom *IRestServiceClient*, mens kommunikasjon mellom *ViewModel*, *IRestServiceClient* og *Fire Risk Model* skjer gjennom *IWeatherClient*. Grunnen til at disse tjenestene er delt opp i to er for å opprettholde prinsippet om løs kobling (Martin, 2006) i applikasjonen (Fisketjøn, Hussain og Svendal, 2022). Dette er spesielt viktig med tanke på gjenbruk av kode dersom tjenestene for beregning av brannrisiko skal flyttes til skyen.



Figur 20. Sekvensdiagram for lagring av ny lokasjon, med beregnet brannrisiko.

4.9 Applikasjonsflyt

Figur 21 viser applikasjonsflyten i form av en tilstandsmaskin som er lik den for mobilapplikasjonen i tidligere arbeid (Fisketjøn, Hussain og Svendal, 2022). Tilstandsmaskin er ofte brukt for å vise hvordan applikasjonens tilstand går over til en annen. Når applikasjonen starter, vil *MyLocationPage* vises. Bruker må være koblet til en MET-ID for å kunne legge til og slette lokasjoner. Hvis ikke MET-ID er koblet til, kan bruker gå til hjelpesiden for mer informasjon.



Figur 21. Tilstandsmaskin (Fisketjøn, Hussain og Svendal, 2022).

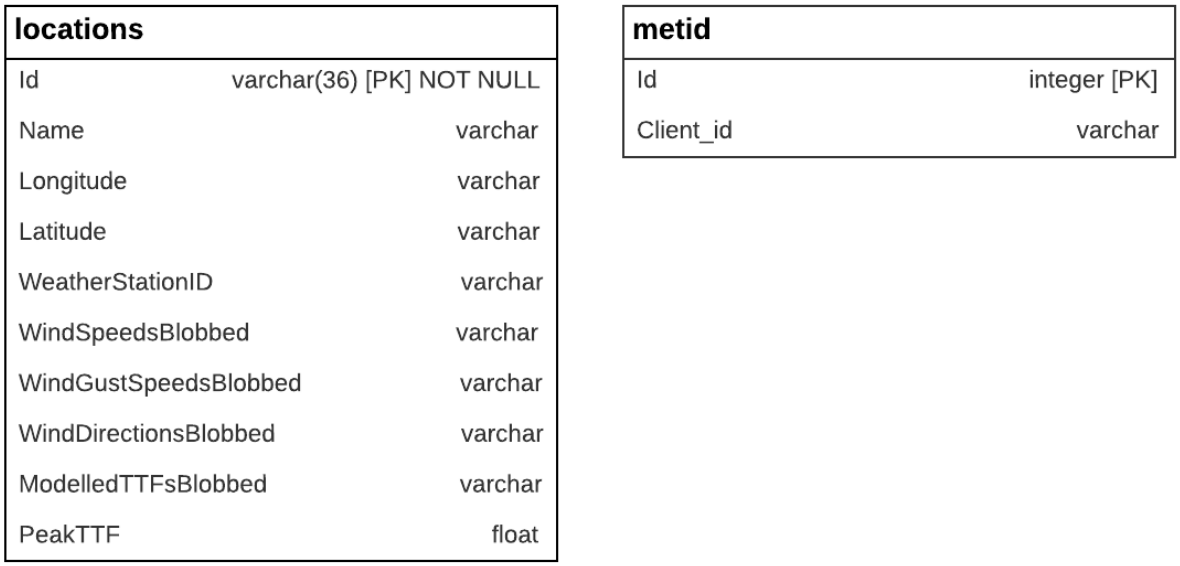
4.10 SQLite Database

For å lagre flere lokasjoner og brukerens MET-ID var det nødvendig å bruke en lokal database, slik som det ble gjort i mobilapplikasjonen fra tidligere arbeid (Fisketjøn, Hussain og Svendal, 2022).

SQLite er en selvstendig, serverløs og nullkonfigurasjonsdatabase, som benytter seg av en transaksjonell SQL-database motor (SQLite, u.å.). Skriving og lesing skjer til og fra vanlige filer på enheten. SQLite støtter ACID-prinsipper som sikrer dataintegritet gjennom atomiske, konsistente, isolerte og holdbare transaksjoner.

Klassen *DatabaseService* håndterer alle transaksjoner til og fra databasen. Slik tilfredsstilles prinsippet om *Single Responsibility* og det er lettere å skalere og endre databasen.

FireRiskLocation-objektene inneholder flere egenskaper som er lister av verdier. Optimalt vil man ved bruk av ORM-databaser (Object Relational Mapping) lagre listedata i egen tabell i databasen og opprette et forhold mellom dem og objektet som «eier» dem. I tilfelle av *FireRiskLocation* inneholder listene et nøyaktig antall verdier (4), og for å forenkle databasen ble det besluttet å bruke *JSON*-serialisering- og deserialisering for å lagre listeverdiene som tekststrenger (varchar). Figur 22 viser tabeller for MET-ID og lokasjoner i databasen.



Figur 22. Tabeller i lokal database.

Figur 23 viser deserialisering av data lagret som tekststrenger i database. I forrige prosjekt (Fisketjøn, Hussain og Svendal, 2022) ble det brukt en ekstern pakke for å oppnå automatisk serialisering. Denne fremgangsmåten skapte problemer da pakken kun fungerer i miljø for mobil (iOS og Android). Fordelen med JSON-serialisering er at det er en plattform-uavhengig fremgangsmåte, slik at lignende problemer ikke vil oppstå i fremtiden.

```
foreach(var location in locations)
{
  //Deserializing of blobbed properties
  location.WindSpeed = JsonConvert.DeserializeObject<List<double>>(location.WindSpeedsBlobbed);
  location.WindGustSpeed = JsonConvert.DeserializeObject<List<double>>(location.WindGustSpeedsBlobbed);
  location.WindDirection = JsonConvert.DeserializeObject<List<double>>(location.WindDirectionsBlobbed);
  location.ModelledTTF = JsonConvert.DeserializeObject<List<double>>(location.ModelledTTFsBlobbed);
  Locations.Add(location);
}
```

Figur 23. Deserialisering ved lasting av data fra database.

4.11 Plattformtilpasset kildekode

Under utvikling ble brukergrensesnitt og forretningslogikk designet med den hensikt at plattformene skulle kreve minimale tilpasninger. Noen steder var det likevel nødvendig. Det presiseres at det med plattform i denne rapporten ofte refereres til mobil, skrivebord, nettbrett og lignende. I .NET MAUI brukes begrepet *Idiom* om disse, mens *Platform* omhandler operativsystemer som Android, iOS, Windows, MacCatalyst og Tizen. I arbeid med applikasjonen ble det ikke brukt kode som aksesserer spesifikke operativsystem direkte, noe som er mulig i .NET MAUI. Det ble heller brukt funksjoner som sjekker hvilken plattform applikasjonen kjører på.

For å sjekke om applikasjonen kjører på *Phone* eller *Desktop* ble *OnIdiom* i XAML og *DeviceInfo.Idiom* i C# brukt. Figur 24 demonstrerer logikk for å skjule et bilde på mobil, mens det forblir synlig på alle andre plattformer. *Default* angir standardverdi for plattformer som ikke er presisert. Figur 25 viser bruk av *DeviceInfo.Idiom* for å sjekke om applikasjonen kjører på mobil.

```
1   <Image ... IsVisible="{OnIdiom Phone='False',Default='True'}" ... />
```

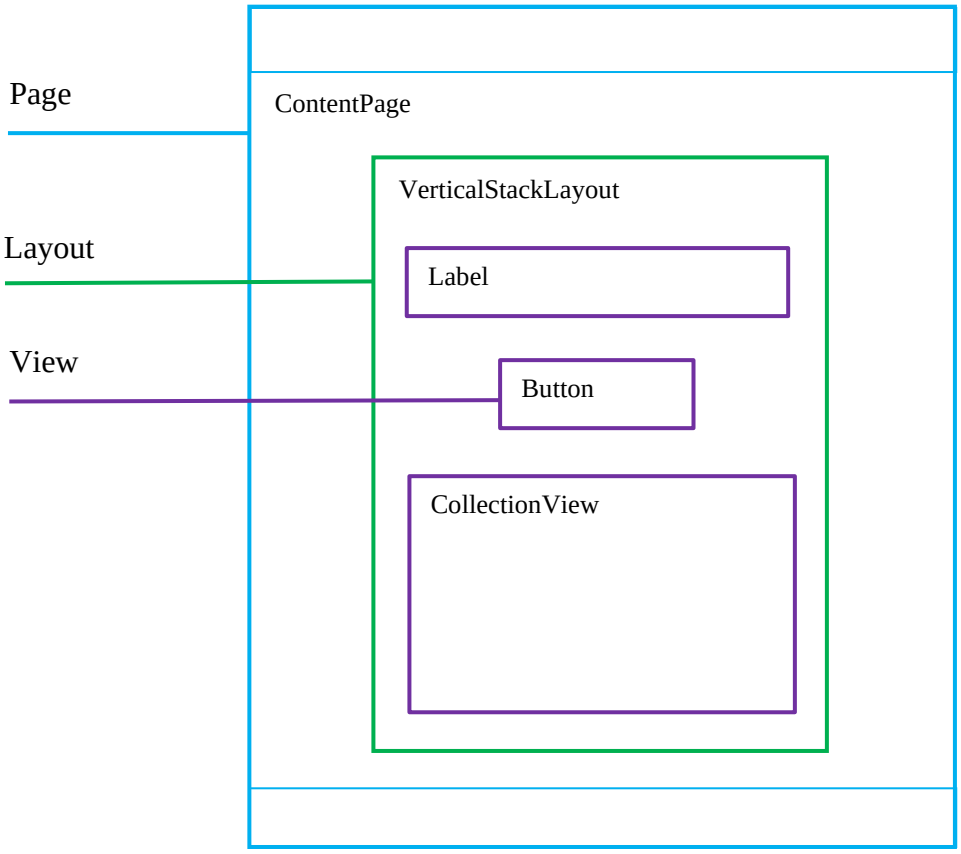
Figur 24. Bruk av OnIdiom.

```
1   bool isPhone = DeviceInfo.Idiom == DeviceIdiom.Phone;
```

Figur 25. Bruk av DeviceInfo.Idiom.

4.12 Grafisk brukergrensesnitt i .NET MAUI

I .NET MAUI blir kontroller i det grafiske brukergrensesnittet kompilert til tilsvarende kontroller på de spesifikke underliggende operativsystemene som Android eller Windows (Microsoft, u.å.). Kontrollene deles inn i tre grupper; *Pages*, *Layouts* og *Views*. Forholdet mellom disse vises i figur 26, med eksempler for hver kontroll. Kode for grafisk brukergrensesnitt i .NET MAUI skrives i språket *XAML* (eXtensible Application Markup Language). Det er mulig å gjøre dette i C#-kode, men *XAML* gir et synlig hierarkisk forhold mellom komponenter som er lettere å lese og forstå.



Figur 26. Hierarki over kontroller for grafisk brukergrensesnitt i .NET MAUI.

4.12.1 Pages

Hver .NET MAUI applikasjon består av en eller flere Pages (sider). Et side-objekt opptar som regel hele skjermen eller vinduet og inneholder minst ett oppsett. Det vanligste side-objektet er *ContentPage*, som inneholder ett enkelt oppsett. Et side-objekt kan sees på som en ramme for det innholdet som skal vises på siden i applikasjonen.

4.12.2 Layouts

Layouts (oppsett) brukes til å ordne det grafiske brukergrensesnittet i visuelle strukturer som bestemmer rekkefølge og posisjon. Hvert oppsett-objekt inneholder typisk mange visninger, men kan også inneholde nøstede oppsett. Et oppsett-objekt kan sees på som en funksjon som plasserer innhold på riktig sted på en side i applikasjonen. For eksempel vil *VerticalStackLayout* plassere innhold nedover i rekkefølge.

4.12.3 Views

Views (visninger) er objekter i det grafiske brukergrensesnittet som utgjør selve innholdet i applikasjonen. Et visning-objekt kan være en knapp, tekstfelt, bilde og mye mer. Noen visning-objekt brukes til å definere utseende på andre visning-objekt, mens andre kan brukes til å oppramse og visualisere data som for eksempel *CollectionView*.

4.13 Grafisk brukergrensesnitt i løsning

Appellen til et kryssplattform-rammeverk som .NET MAUI er at kode skal kunne gjenbrukes mellom plattformer for å lage modeller, brukergrensesnitt og bindinger mellom dem. Det er derfor avgjørende at brukergrensesnittet på mobil og skrivebord har samme filosofi selv om tilpasninger må gjøres. Med filosofi menes det at farger, presentasjon av data, navigasjon og interaksjoner er like i stor grad. Dersom de individuelle tilpasningene blir for store, vil ikke lenger fordelene ved et kryssplattform-rammeverk være til stede. Ressursbruken i forbindelse med å utvikle og feilsøke kildekode vil nærme seg det som trengs ved tradisjonell utvikling på hver enkelt plattform.

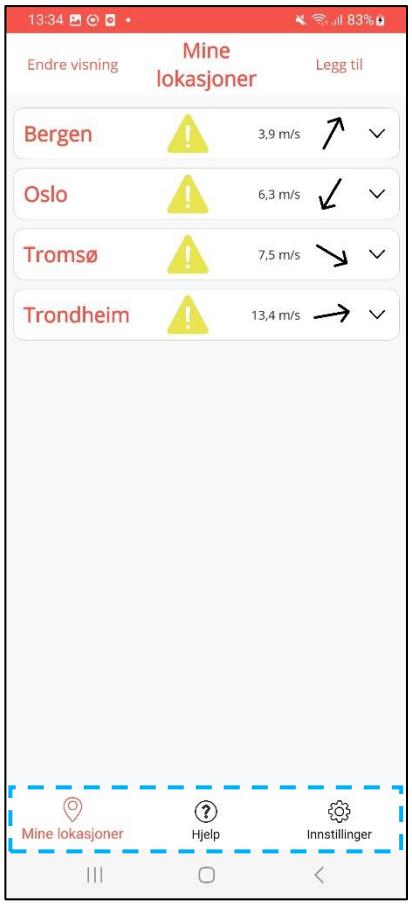
4.13.1 Navigering

Shell i .NET MAUI er til for å redusere kompleksitet i utvikling av applikasjoner ved å tilby funksjonalitet for visuell navigasjon (Microsoft, u.å.). For å gi en navigasjonsopplevelse som er tilpasset plattform, har mobil og skrivebord to forskjellige *Shell*. For å opprette riktig *Shell*, så kontrolleres det i *App.xaml.cs* hvilken plattform applikasjonen kjører på, som vist i figur 27.

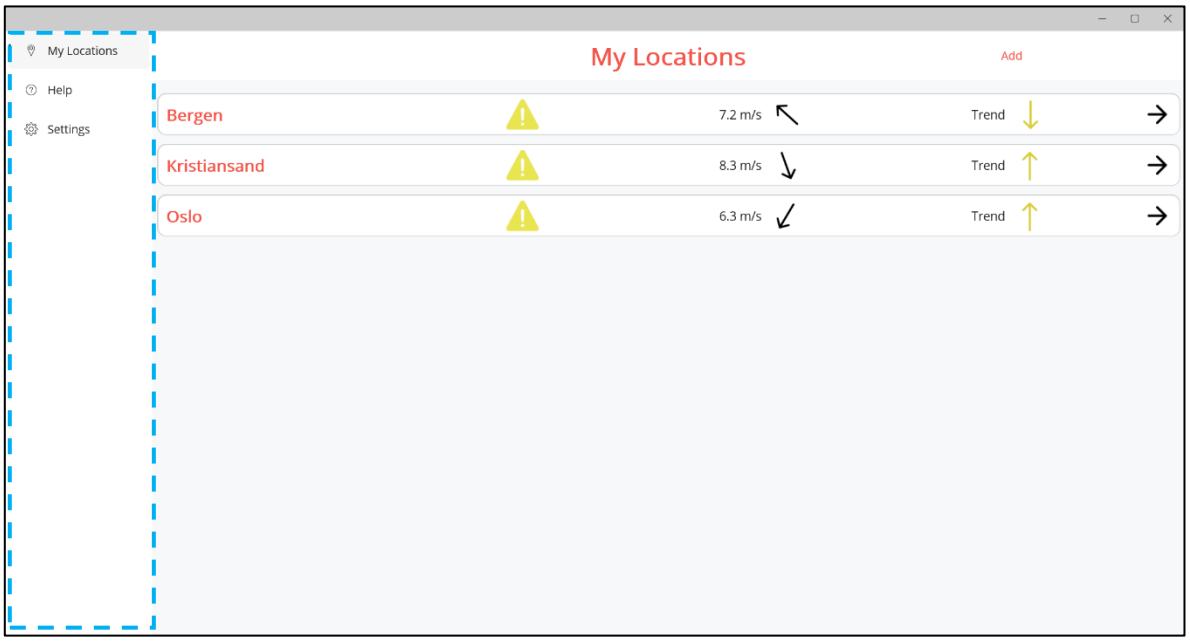
```
1   public App()
2   {
3     InitializeComponent();
4
5     if(DeviceInfo.Idiom == DeviceIdiom.Phone)
6       MainPage = new MobileShell();
7     else
8       MainPage = new DesktopShell();
9   ...
10  }
```

Figur 27. Shell for navigasjon velges ut ifra plattform.

Ved å bruke *Shell* kobles navigasjonsvalgene direkte til innholdssider i applikasjonen, og oppretting og navigering av sidene håndteres automatisk. *Shell* for mobil vises i figur 28 markert med blå farge, mens *Shell* for skrivebord vises i figur 29 markert med blå farge. På mobil markeres siden brukeren er på med annerledes farge på menyvalget i navigasjonslinjen nederst, mens på skrivebord markeres dette ved en svak bakgrunnsfarge bak aktuell side.



Figur 28. Shell for mobil.



Figur 29. Shell for skrivebord.

4.13.2 Mine lokasjoner

Brukeren av applikasjonen har et behov for å se informasjon om innendørs brannrisiko for bestemte lokasjoner. Brukerens lagrede lokasjoner hentes fra databasen på enheten og legges i listen *Locations* i *MyLocationsViewModel*, som vist i figur 30. For å kunne vise informasjon om brannrisiko må listen med lokasjoner settes som *ItemsSource*-kilde til et *CollectionView* i *MyLocationsPage* som vist i figur 31. *CollectionView* tilbyr oppramsing av modell-objekter i en liste, hvor man kan bestemme hvilken data som vises fra hvert objekt, og hvordan den vises (Microsoft, u.å.). Rulling i listen er automatisk implementert.

```
1   public ObservableCollection<FireRiskLocation> Locations { get; } = new();
```

Figur 30. Instansiering av listen Locations i MyLocationsViewModel.

```
1   <CollectionView x:Name="LocationsCollectionView"
2                   ItemsSource="{Binding Locations}"
3                   ... >
```

Figur 31. Locations settes som kilde til CollectionView i MyLocationsPage.

4.13.3 DataTemplate

For å bestemme hvordan brukerens lokasjoner skal presenteres brukes en *DataTemplate*-mal. Denne tildeles til *CollectionView* ved *ItemTemplate*-egenskapen, og defineres som ressurs øverst i *MyLocationsPage* som vist i figur 32. Figur 33 viser hvordan *DataTemplate* med nøkkel *PhoneLocationItemTemplate* velges dersom applikasjonen kjøres på mobil, mens ellers velges *DesktopLocationItemTemplate*. Grunnen til at mobil og skrivebord har ulik *DataTemplate* forklares i de to neste avsnittene.

```
<ContentPage.Resources>
  <ResourceDictionary>
    <DataTemplate x:Key="PhoneLocationItemTemplate">
      <toolkit:Expander>
        ...
      </toolkit:Expander>
    </DataTemplate>

    <DataTemplate x:Key="DesktopLocationItemTemplate">
      <Grid>
        ...
      </Grid>
    </DataTemplate>
  </ResourceDictionary>
</ContentPage.Resources>
```

Figur 32. Definering av DataTemplates som ressurser.

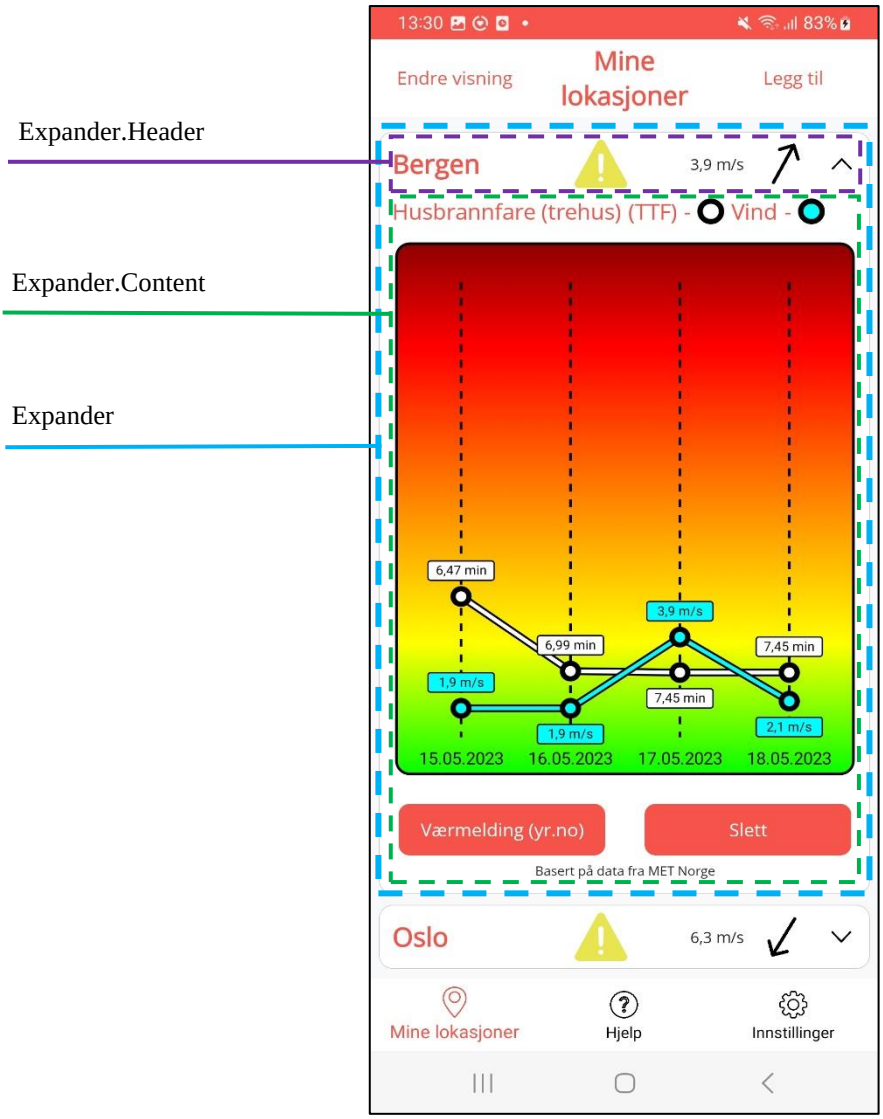
```
var phoneTemplate = (DataTemplate)Resources["PhoneLocationItemTemplate"];
var desktopTemplate = (DataTemplate)Resources["DesktopLocationItemTemplate"];

if (DeviceInfo.Idiom == DeviceIdiom.Phone)
    LocationsCollectionView.ItemTemplate = phoneTemplate;
else
    LocationsCollectionView.ItemTemplate = desktopTemplate;
```

Figur 33. DataTemplate velges ut ifra plattform.

4.13.4 Expander på mobil

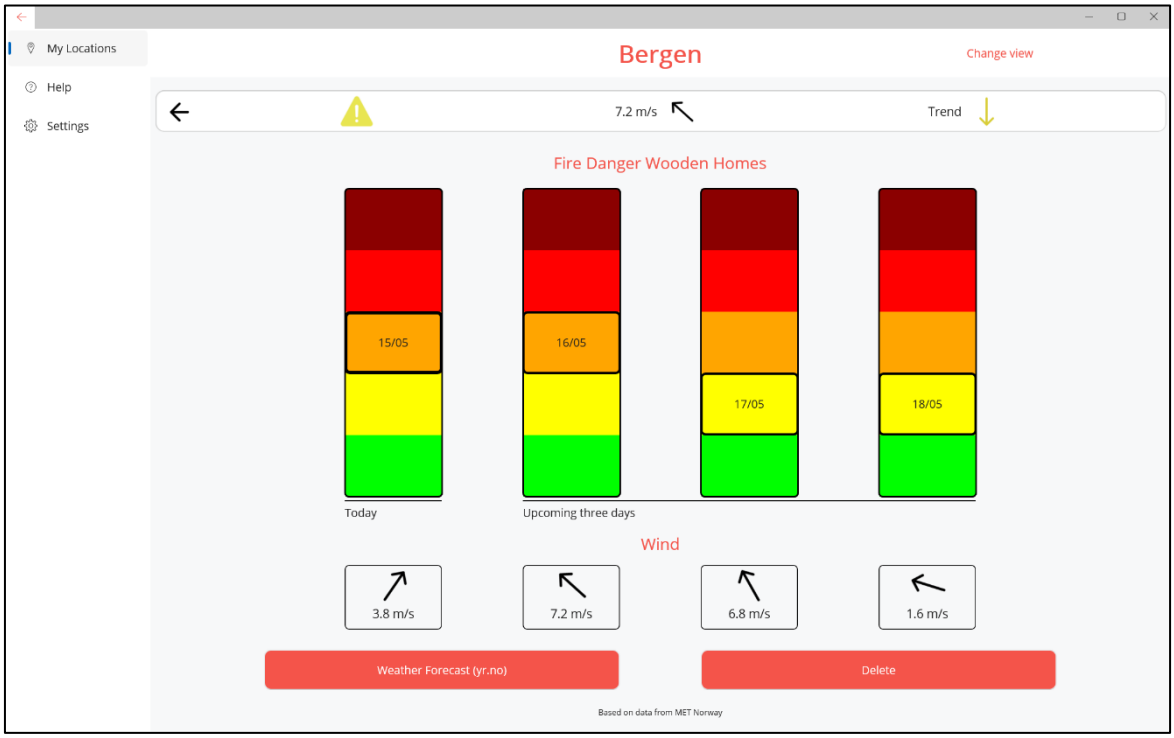
Expander fra .NET MAUI *Community Toolkit* er en ramme for innhold som kan utvides og lukkes (Microsoft, u.å.) som vist i figur 34. Blå ramme viser hele Expander-komponenten. På mobil brukes denne komponenten i listen med *FireRiskLocation*-objekter. I *Expander.Header* defineres data som skal vises øverst i *Expander*, vist med lilla farge i figur. I *Expander.Content* defineres innhold som skal ligge i den utvidede versjonen av rammen, vist med grønn farge i figur. *Header* består av lokasjonsnavn, varseltrekant for brannrisiko og retning og hastighet for vind. *Content* inneholder visualisering av brannrisiko for de 4 aktuelle dagene, samt knapper for å gå til værmelding og slette lokasjon.



Figur 34. Expander på mobil.

4.13.5 Alternativ for skrivebord

Planen var å bruke *Expander* i listen over lokasjoner også på skrivebord, det vil si ikke gjøre noen større individuelle tilpasninger. Uheldigvis førte dette til utfordringer som ikke kunne løses. Utvidelse og lukking av *Expander* på Windows ga oppførsel som gjorde listen over lokasjoner ubrukelig. Det lyktes ikke prosjektgruppen å finne ut gjennom feilsøking hvorfor disse problemene oppstod på Windows, mens alt fungerte utmerket på mobil. Alternativet ble å heller la hver lokasjon i listen være en knapp til en ny side, *LocationPage*. Denne nye siden brukeren tas videre til er vist i figur 35. I listen er hver lokasjon svært lik som i *Expander.Header* på mobil. På skrivebord finnes i tillegg en trend-pil som vist i figur 36. Denne illustrerer om brannrisikoen øker eller synker, og fargen på pilen tilsier farenivå trenden beveger seg mot.



Figur 35. LocationsPage for en bestemt lokasjon på skrivebord.

Figur 36. Trendpil på skrivebord.

4.13.6 Celler

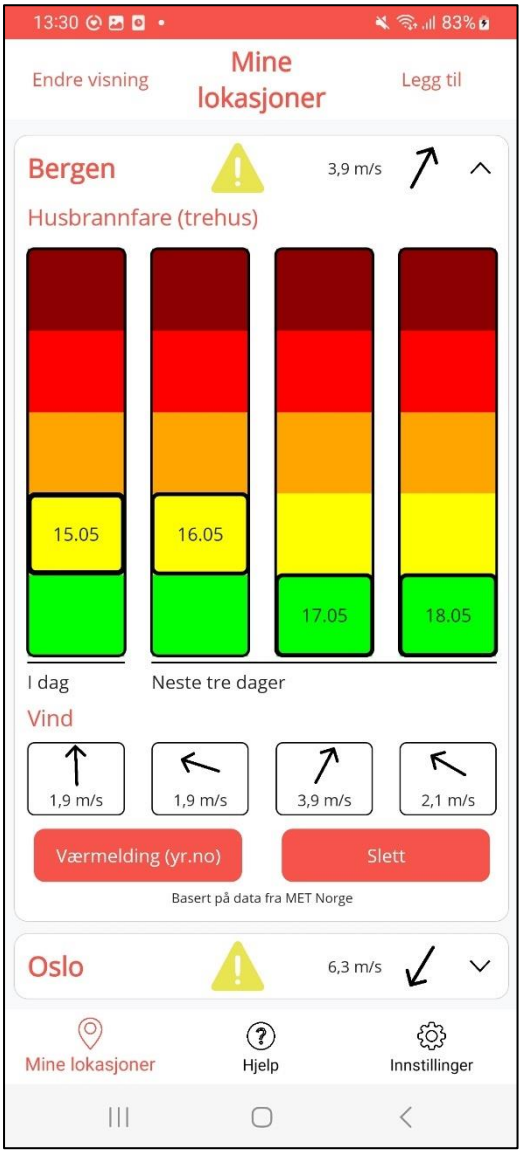
Modell for brannrisiko inneholder data som kan være vanskelig å forstå og sette i sammenheng for brukeren (TTF-verdier spesielt). For å bedre presentere informasjon til brukeren ble det laget visualiseringsalternativer. For å oppnå dette må innholdet i *Expander.Content* på mobil, og i *LocationPage* på skrivebord kunne endres dynamisk etter knappetrykk fra bruker. Figur 37 viser hvordan dette er løst ved å bruke *DataTrigger* som setter innholdet til å være en av tre definerte celler, avhengig av knappetrykk. *Cell* er brukt som et alias for disse delene med innhold, andre begrep som *Widget* eller *Control* kunne vært brukt. Cellene er definert i egne XAML-klasser og er av typen *ContentView* slik at de kan ta plass inne i hvilken som helst *Layout*.

```
1   <toolkit:Expander.Content>
2
3     <cells:BarChartCell IsVisible="False" ...>
4       <cells:BarChartCell.Triggers>
5         <DataTrigger TargetType="cells:BarChartCell"
6                      Binding="{Binding Source={x:Reference locationspage},
7                                Path=BindingContext.VisState}" Value="bar">
8           <Setter Property="IsVisible" Value="True" />
9         </DataTrigger>
10      </cells:BarChartCell.Triggers>
11    </cells:BarChartCell>
12
13    <cells:GraphCell IsVisible="False"...>
14      ...
15    <cells:GraphCell IsVisible="False"...>
16      ...
17
18  </toolkit:Expander.Content>
```

Figur 37. DataTrigger som bestemmer hvilken celle som vises.

4.13.7 Celle for stolpediagram

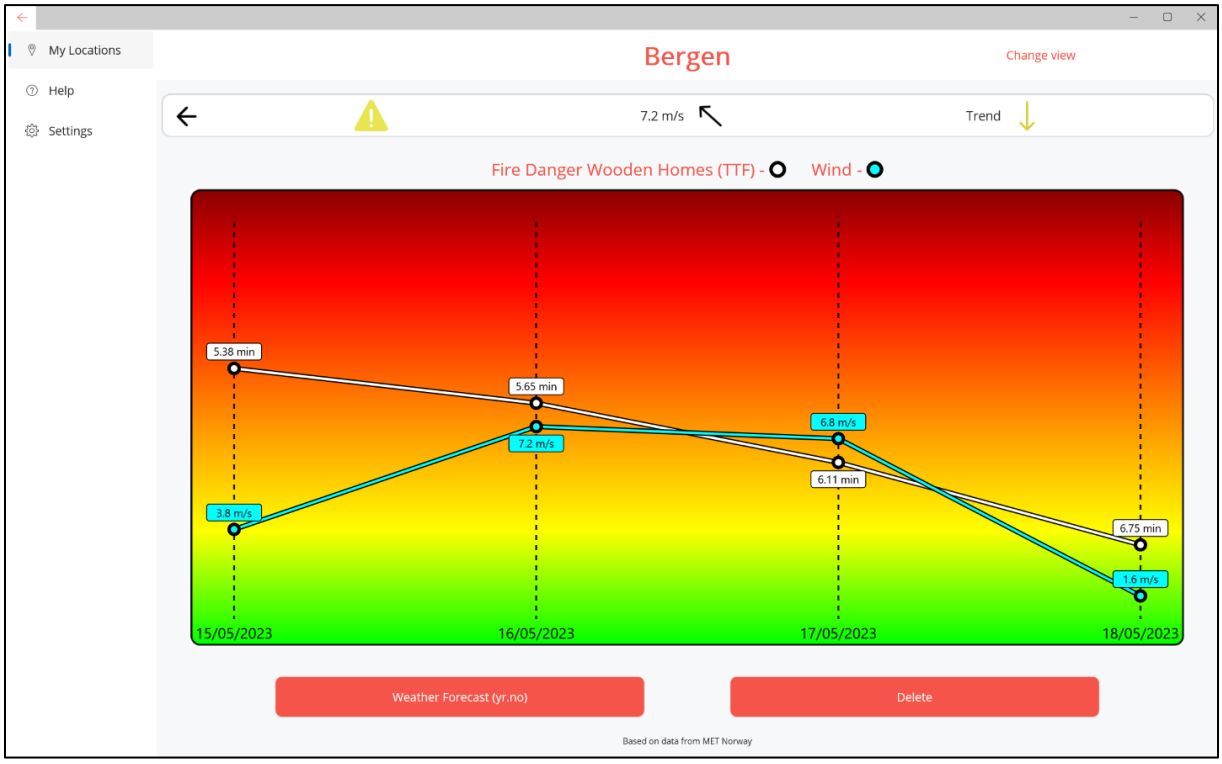
Visualiseringen av brannrisiko som stolpediagram er overført fra tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022). Stolpene bruker farger tilsvarende risiko som er beskrevet i avsnitt 4.6, og aktuell risiko for dato er markert med tykk firkant og dato. Figur 38 viser stolpediagram på mobil. En vesentlig endring for kommunikasjon av risiko er her overgang til begrepet husbrannfare (trehus) i motsetning til det forrige konseptet rundt IFD.



Figur 38. Stolpediagram på mobil.

4.13.8 Celle for graf

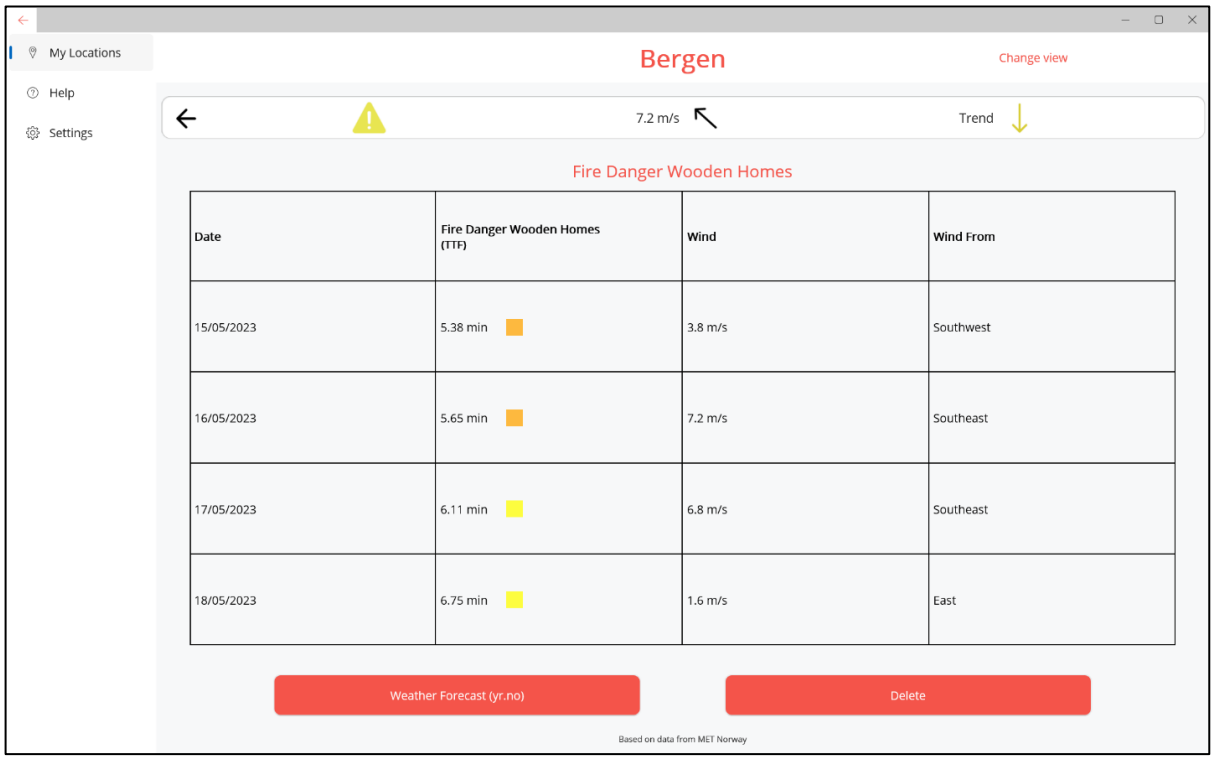
I forbindelse med de nye kravene til visualisering ble det utviklet et alternativ hvor brannrisiko og vind presenteres i en felles graf. Dermed kan brannfare og vind betraktes i samme graf og det oppnås en bedre forståelse av samlet risiko for storbrann. Spesielt for fremstilling ved bruk av graf er at vindhastigheter er blitt klassifisert i fareklasser (fargekoder). Dermed får brukeren et inntrykk av hvilken fare vinden utgjør. Figur 39 viser *GraphCell*-komponenten på skrivebord. Grafen tegnes ved hjelp av et *GraphicsView* som er representert ved et *ICanvas*-objekt i C#. Logikk for å tegne graf er utviklet med enkle tegneelementer. Farge, form, posisjon og størrelse må bestemmes for hver eneste tegneoperasjon. Det ble sett på alternativer til tredjeparts-bibliotek som Telerik som tilbyr komponenter for datavisualisering som for eksempel grafer, men disse er svært dyre (899$ per utvikler per år for Telerik) (Telerik, u.å.).



Figur 39. Graf med TTF-verdier og vind, på skrivebord.

4.13.9 Celle for tabell

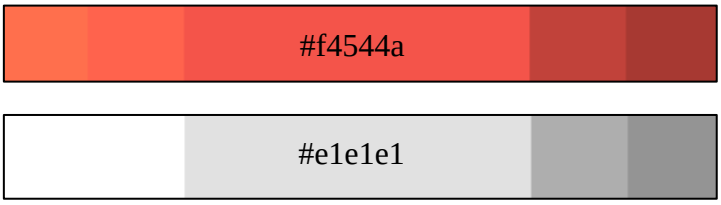
Et krav til visualisering inkluderte en tabell, som vist i figur 40. Denne hjelper brukeren til å se verdier som TTF og risikofarge i sammenheng. Tabell som visuelt verktøy er også kjent for de fleste brukere.



Figur 40. Tabell på skrivebord.

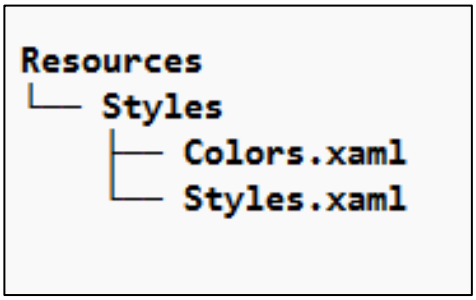
4.14 Fargevalg og stiler

Prosjekteier var åpen for at prosjektgruppen kunne eksperimentere med fargevalg for å gi applikasjonen en sterk identitet. Valget av farger i applikasjonen har spilt en avgjørende rolle for å skape en god brukeropplevelse. Applikasjonenes fargesammensetning inkluderer primærfargen rød og sekundærfargen grå. Figur 41 illustrerer fargepaletten som ble benyttet. Hensikten med fargevalget var å velge farger som assosieres med brannrisiko og fare.



Figur 41. Fargepalett brukt i applikasjonen.

For å oppnå konsistens og brukervennlighet i applikasjonen tilbyr .NET MAUI et godt system for å organisere farger og stiler. Figur 42 viser hvordan filer for farger og stiler oppbevares. Figur 43 viser definisjon av fargen med nøkkel *Primary*. Figur 44 viser hvordan denne fargen brukes til å definere en stil med nøkkel *ToolbarButtonStyle*. Figur 45 viser hvordan denne stilen tas i bruk på en *Button*-kontroll, slik at den får fargen *Primary*. Dette kalles eksplisitt stil. Det er også mulig å definere implisitte stiler for hver type kontroll. Bruk av stiler gjør at endringer i farge (og andre egenskaper) kan gjøres fra en sentral lokasjon, ikke hos hver enkelt kontroll.



Figur 42. Struktur for stiler og farger.

```
<ResourceDictionary
    <Color x:Key="Primary">#f4544a</Color>
    ...
</ResourceDictionary>
```

Figur 43. Fargen med nøkkel Primary defineres i Colors.xaml.

```
<ResourceDictionary>
    <Style x:Key="ToolbarButtonStyle" TargetType="Button">
        <Setter Property="TextColor" Value="{StaticResource Primary}" />
        ...
    </Style>
</ResourceDictionary>
```

Figur 44. Stilen med nøkkel ToolbarButtonStyle defineres i Styles.xaml.

```
<Grid>
  <Button Style="{StaticResource ToolbarButtonStyle}"/>
    ...
</Grid>
```

Figur 45. Bruk av stil for Button i MyLocationsPage.xaml.

4.15 Oversetting

Et krav til applikasjonen var at den skulle være tilgjengelig på engelsk og norsk. .NET MAUI tilbyr et enkelt system for oversetting, eller mer presist *lokalisering*, av tekst i applikasjonen. Figur 46 viser hvordan oversetting for overskriften «My Locations» blir gjort på både norsk og engelsk. Figur 47 viser hvordan teksten hentes ut ved å bruke *AppResources.My_Locations*. Ved kjøretid bestemmer enhetens språkinnstillinger hvilket språk som skal brukes. Engelsk er standardvalg. Eksemplene fra kjøring av applikasjonen i kapittel 4 viser engelsk språk på skrivebord og norsk språk på mobil.

Name	Value
My_locations	Mine lokasjoner

Name	Value
My_locations	My Locations

Figur 46. My_locations tekststreng i AppResources.resx.

```
1   <Label Text="{x:Static resx:AppResources.My_locations}" Grid.Column="1" .. />
```

Figur 47. Bruk av tekststreng My_locations, som blir oversatt ved kjøring.

5 EVALUERING

Resultat for prosjektet ble evaluert i form av demonstrasjon, test av ferdig produkt og en vurdering av kryssplattform-rammeverket.

5.1 Demonstrasjon

For å evaluere og justere de visuelle egenskapene til applikasjonen ble det gjennomført en demonstrasjon med stipendiat Ruben Dobler Strand. Målet var å vurdere foreløpig arbeid, spesielt med tanke på utseende og begrepsbruk i brukergrensesnittet. Forslag til eventuelle endringer ble på denne måten oppdaget før den avsluttende evalueringen.

Hver side i applikasjonen ble demonstrert, og tilbakemeldinger ble gitt på konkrete elementer som burde endres. Dobler Strand utarbeidet noen forslag til endringer. Figur 48 viser skjermbilde til venstre og forslag til endring på høyre, for stolpediagram. Det ble besluttet at applikasjonen skulle gå vekk fra å bruke hele fargespekteret fra indeksen for skogbrannfare. Som nevnt i avsnitt 4.6 ble blått og hvitt nivå fjernet, siden det ikke er hensiktsmessig å benytte betegnelsen «ingen fare» i forbindelse med risiko for husbrannfare. I tillegg ville endringen gi et mer kjent fargespekter for risiko; rød-gul-grønn. Legg også spesielt merke til endringer i begrepsbruk, som blir beskrevet nedenfor.

Figur 48. Forslag til endringer fra prosjekteier etter demonstrasjon.

Følgende er utvalgte punkter over foreslåtte endringer:

- Fjerne en del tekst for å skape en ryddigere overskrift for hver lokasjon.
- Erstatte «Modellert Time To Flashover (mTTF)» med begrepet «Husbrannfare (trehus)».
- Fjerne blå og hvit farge fra fargeskala for brannrisiko.
- Gå bort fra begrepene «Ekstremt rask» og «Sakte» i stolpediagram og i graf.

5.2 Test av ferdig produkt

Etter en kort iterasjon med utvikling for å gjennomføre endringene som ble besluttet i forbindelse med demonstrasjon, ble applikasjonen evaluert i sin endelige versjon. Evalueringen ble gjennomført som brukertest med påfølgende spørreskjema. Stipendiat Ruben Dobler Strand og stipendiat Tord Hettervik Frøland, med bakgrunn innen programutvikling, deltok. Testing ble gjennomført på Android og Windows med reelle data. Spørreskjema bestod av spørsmål som baserte seg på opprinnelige funksjonelle krav samt oppgavens problembeskrivelse og mål. Vedlegg 9.4 inneholder utlevert informasjonstekst og oppgave gitt testpersonene samt spørreskjema benyttet under evalueringen av skrivebordapplikasjon. Identisk intervju rettet mot mobilapplikasjon ble også gjennomført i tillegg til spørsmål om applikasjonen og prosjektet som en helhet. Følgende er en punktvis oppsummering av tilbakemeldinger gitt ved evaluering av ferdig produkt:

- **Navigering** fungerte godt på mobil, mens på skrivebord var det mindre intuitivt å navigere seg mellom listen av lokasjoner og en bestemt lokasjon. Bedre bruk av ikoner for navigering ble foreslått som forbedring.
- **Stolpediagram** fungerte godt på mobil, mens på skrivebord var det usikkert om dette fungerte like bra som opprinnelig tenkt. Plassering og formatering av datoer samt tettere stolper ble foreslått som mulige forbedringer.
- **Graf** fungerte svært godt på begge plattformer med TTF og vind i samme plott. En mulig forbedring er å legge til enheter for TTF og vind på y-akse på skrivebord.
- **Tabell** fungerte fint på begge plattformer.
- Funksjoner for å **legge til** og **slette** lokasjoner fungerte godt.
- Side for **Innstillinger** fungerte fint.
- Side for **Hjelp** fungerte fint, men innhold behøvde en oppdatering.

Prosjekteier svarte også at utformingen av applikasjonen samsvarte med bestillingen, og at nåværende versjon fremstod moden for testing hos brannvesen som er partnere i DYNAMIC-prosjektet. Utfordringer i forbindelse med navigering, datoer, begrepsbruk og innhold på hjelpeside ble utbedret i etterkant av test. Endringer som var tidkrevende ble ikke arbeidet med, men redegjøres for i *systemdokumentasjon* (vedlegg 4, kapittel 8).

5.3 Evaluering av kryssplattform-rammeverk

5.3.1 Rammeverkets formål

Formålet med .NET MAUI er å forenkle og effektivisere utviklingsprosessen ved å tilby en delt kodebase for å bygge apper på tvers av forskjellige plattformer. Abstrahering og gjenbruk av kildekode er et viktig prinsipp innen utvikling av programvare fordi det øker produktivitet, reduserer kostnader og fører til enklere vedlikehold (Martin, 2008). .NET MAUI oppnår dette dersom utvikleren ikke støter på for mange brukstilfeller eller problemer som er spesifikke for hver plattform. Kryssplattform-rammeverk er godt etablert i mobilutvikling i industrien. Applikasjoner som Instagram, Discord og faktisk Microsoft Teams og Outlook er utviklet i React Native (Meta Platforms, u.å). Når det er snakk om kryssplattform-applikasjoner som skal kjøre på mobil og skrivebord finnes det flere utfordringer som påvirker negativt prinsippet om abstrahering og gjenbruk. Forskjellene mellom plattformene er større med tanke på operativsystem, skjermstørrelse og standarder. Deler av kildekoden må nødvendigvis «skrives to ganger».

5.3.2 Oppnådd grad av kryssplattform

Det er utfordrende å kvantifisere den faktiske graden av kryssplattform ved evaluering av den utviklede applikasjonen. For å skape et nøyaktig bilde ville det være nødvendig å utvikle applikasjonen på alle relevante plattformer og deretter evaluere størrelse, kvalitet og utviklerens subjektive erfaringer. Sammenligning av antall kodelinjer er ikke en fullstendig presis metode av ulike grunner, men den ble likevel benyttet for å gi en indikasjon på hvordan kryssplattform-rammeverket har påvirket applikasjonen.

En kvantitativ analyse ble utført for å bestemme prosentandelen av kildekode i applikasjonen som er tilpasset plattform. Denne analysen innebar identifisering av kildekode som utvikleren skrev to ganger; én for mobil og én for skrivebord. Dette ble gjort for å tilfredsstille plattformspesifikke krav, som for eksempel egen side for lokasjon på skrivebord som beskrevet i avsnitt 4.13.5. Ideelt ville disse delene med kildekode vært skrevet bare én gang. Basert på denne identifiseringen ble det estimert at ca. 210 kodelinjer av totalt 3070 faller inn under denne

kategorien. Dette utgjør om lag 7% av den totale koden, noe som indikerer en betydelig grad av kryssplattform, altså at plattformene benytter identisk kildekode.

Ved å innskrenke analysen til bare å inkludere filer som er direkte involvert i brukergrensesnitt og tilhørende klasser, som omfatter 1900 kodelinjer, var prosentandelen av plattformspesifikk kode 11%. Dette resultatet, selv om det er litt høyere, viser fortsatt en betydelig grad av kryssplattform - så mye som 89%. Det er imidlertid viktig å merke seg at disse kvantitative målingene også bør ses i sammenheng med de erfaringene som ble gjort under utviklingsprosessen.

5.3.3 Prosjektgruppens erfaringer

Under utviklingen av dette prosjektet har prosjektgruppen opplevd en rekke positive og negative aspekter ved bruk av .NET MAUI-rammeverket. Det ble oppdaget at det var utfordrende å tilegne seg de nødvendige ferdighetene for å arbeide effektivt med rammeverket, spesielt grunnet mangel på omfattende dokumentasjon og ressurser. Dette kan forstås i lys av rammeverkets relativt korte tid som en allment tilgjengelig programvare (offisielt utgitt 23.05.2022).

Hovedutfordringen med .NET MAUI, og potensielt andre kryssplattform-rammeverk, er at feilsøking er krevende. Problemer kan for eksempel oppstå på kun én plattform, og endringer man gjør som konsekvens av dette kan skape problemer på en annen plattform. Microsoft markedsfører .NET MAUI som en «write-once run-everywhere»-opplevelse for utviklere. Imidlertid har dette begrepet blitt humoristisk omskrevet til "write-once debug-everywhere" på ulike utviklerforum, noe som illustrerer de potensielle utfordringene ved kryssplattform-utvikling.

Til tross for disse utfordringene har prosjektgruppen anerkjent betydelige fordeler ved å bruke .NET MAUI. Utvikling av separate løsninger for hver enkelt plattform ville ha medført tidsbegrensninger som ikke ville være mulig å overkomme innenfor prosjektets tidsramme. XAML, språket som brukes for det grafiske brukergrensesnittet, ble funnet å være relativt intuitivt å lære og effektivt å bruke. Videre ga verktøy som .NET, C# og Visual Studio, som er etablert og anerkjent i bransjen, en positiv utviklingsopplevelse.

6 DISKUSJON

Resultatet av prosjektet er en applikasjon for varsling av brannrisiko i trehusmiljøer. Ved å benytte seg av .NET MAUI-rammeverket, kan den kjøres på flere plattformer og gi brukeren tilpasset informasjon. Applikasjonen funger på Windows og Android og gir brukeren mulighet til å visualisere brannrisiko for ulike lokasjoner, noe som kan bidra til tiltak som forebygger brann i trehus. Prosjektgruppen har ikke hatt tilgang på eller prioritert testing på iOS og Mac, men det antas at det skal være en overkommelig jobb å få applikasjonen til å fungere på disse plattformene. Det er ikke gjort noen tilpasninger i applikasjonen som skal tilsi noe annet.

Evaluering av applikasjonen var svært nyttig for å avdekke mangler og ønskede funksjoner. Det ville vært ideelt å utføre den første demonstrasjonen tidligere i prosessen. Imidlertid var det ikke passende å evaluere applikasjonen før utviklerne mente at den oppfylte de funksjonelle kravene. Arbeidet med forbedringer etter evaluering ble hektisk, men ga den ferdige løsningen verdi. Prosjekteier evaluerte applikasjonen som klar for videre testing i brannvesenet siden den har oppnådd de fastsatte målene.

For prosjektet var det avgjørende å utarbeide en detaljert prosjektplan for å bevare oversikten over de fastsatte tidsfristene. Gantt-skjema (vedlegg 9.1) fungerte godt til dette. Utviklingsfasen tok lengre tid enn antatt, og frister måtte revideres kontinuerlig. Dette skyldtes i hovedsak tidkrevende feilsøking. Resten av fasene fra planleggingen ble utført som planlagt og tidsfristene ble møtt.

Ved å ta i bruk prinsipper fra Scrum som utviklingsmetodikk kunne prosjektet raskt tilpasse seg endringer i krav fra prosjekteier og problemer underveis. I startfasen av prosjektet hvor tidligere arbeid skulle undersøkes for gjenbruk var det derimot utfordrende å inkludere de samme prinsippene. Når dette arbeidet var ferdig, og et skjelett for applikasjonen ble lastet opp på Github, økte arbeidstempoet. Gruppen omgjorde funksjonelle krav fra prosjekteier til deloppgaver som kunne arbeides med. Ved å bruke *Issues* på Github for hver deloppgave var prosjektmedlemmene oppdatert på hverandres fremgang, og justeringer ble gjort der det var nødvendig for eksempel med tanke på tidsbruk.

Prosjektet støtte på en del utfordringer under utvikling som i stor grad må tilskrives rammeverket. Det ble konstant vurdert om man skulle rette opp et problem som var spesifikt for én plattform, eller utsette dette for å opprettholde fremdriften i utviklingen. Et eksempel er *Expander*-komponenten fra *Maui.CommunityToolkit*-pakken som forklares i avsnitt 4.13.5.

Flere andre MAUI-komponenter inneholdt også feil på enkelte plattformer. Likevel er prosjektgruppen fornøyd med de alternative løsningene som ble utarbeidet. Mange av utfordringene kunne potensielt vært unngått dersom gruppen hadde valgt et mer modent og veletablert rammeverk. Dette ville imidlertid gått på bekostning av gjenbruk fra tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022).

Et godt samarbeid med prosjekteier har vært en viktig del av prosjektet. Gjennom regelmessige møter ble tett dialog opprettholdt, noe som bidro til å oppnå prosjektets mål og forventinger. Prosjekteier bidro med faglig innsikt som var viktig for at prosjektgruppen skulle forstå behovene til DYNAMIC. Nye krav fra prosjekteier underveis, knyttet til for eksempel begrep og farger for risiko i brukergrensesnittet, ga prosjektmedlemmene erfaring med å håndtere endringer i programvarekrav.

Hvis prosjektet skulle vært gjennomført på nytt ville en del endringer blitt gjort i lys av erfaringene. En evaluering av applikasjonen burde vært gjort underveis og tidligere i prosjektet. Som følge av dette måtte prosjektet hatt fokus på å utvikle en fungerende applikasjon tidligere, med mindre hensyn til kvalitet. Slik ville problemer og endringer blitt identifisert raskt. I startfasen av prosjektet burde det ha vært mindre fokus på å gjenbruke kildekode fra tidligere arbeid. Brukergrensesnittet burde blitt utviklet fra bunn.

7 KONKLUSJON OG VIDERE ARBEID

Denne bacheloroppgaven har undersøkt hvordan et kryssplattform-rammeverk kan brukes til å utvikle en applikasjon for varsling av innendørs brannrisiko i trehus. Det har blitt gjort tilpasninger som gjør at applikasjonen kan benyttes av ulike brukergrupper. Prosjektgruppen har utviklet en fungerende applikasjon for mobil og skrivebord, og samtidig vurdert .NET MAUI som rammeverk. Konklusjon av rapporten tar for seg svar på forskningsspørsmål som ble stilt i avsnitt 1.6.

7.1 Svar på forskningsspørsmål

FS1: Kan rammeverket .NET MAUI brukes til å utvikle en kryssplattform-applikasjon for mobil og skrivebord som gir informasjon om brannrisiko i trehus tilpasset plattform og ulike brukere?

Applikasjonen som ble utviklet fungerer godt til å presentere informasjon om innendørs brannrisiko i trehus til ulike brukere på ulike plattformer. Ved å justere begrepsbruk for risiko og tilby visualiseringsalternativer kan brukere med ulik bakgrunn benytte applikasjonen. Gjennom evaluering og tilbakemeldinger ble det avdekket at applikasjonen er klar for videre testing hos brannvesen som er partnere i DYNAMIC-prosjektet. Dette betyr at prosjektet har lyktes med å nå de målene som ble satt for applikasjonen, ved bruk av .NET MAUI.

FS2: Hvilken grad av kryssplattform er det mulig å oppnå med .NET MAUI?

.NET MAUI muliggjør en høy grad av kryssplattform, men dette avhenger mye av kompleksiteten til applikasjonen. I kildekoden til applikasjonen er det som beskrevet i avsnitt 5.3.2 89% som *ikke* er tilpasset individuelle plattformer, når det kommer til brukergrensesnitt. Utvikling av applikasjonen derimot ga et negativt bilde av hvor stor grad av kryssplattform man kan oppnå. Feilsøking og testing på flere plattformer samt feil som viste seg kun på enkelte plattformer gjorde at applikasjonen ikke kunne dra fullt nytte av fordelene i .NET MAUI.

FS3: Er det hensiktsmessig å kopiere arkitektur og kildekode, eller starte utvikling fra bunn ved overgang til kryssplattform-rammeverk?

Dersom kryssplattform blir valgt som strategi for å utvikle en applikasjon basert på tidligere arbeid, anbefales det å planlegge nøye hvilke deler som skal gjenbrukes. Gjennom prosjektet ble det erfart at videreføring av arkitektur er en fordel, fordi applikasjonens logikk kan ta inspirasjon fra tidligere arbeid. Modeller bør gjenbrukes i størst mulig grad, mens brukergrensesnitt bør utvikles fra bunn. I MVVM-designmønsteret betyr dette at både *View* og *ViewModel* bør implementeres på nytt ved å følge standarder for aktuelt rammeverk.

7.2 Antagelser og potensielle feilkilder

I dette avsnittet vil det presenteres antagelser og potensielle feilkilder som kan ha påvirket resultatene og konklusjonene i denne bacheloroppgaven. Det er viktig å være klar over disse faktorene for å forstå og vurdere påliteligheten av funnene som er presentert.

Brukertest av applikasjonen ble kun gjennomført med to personer. Dette er en begrensning fordi man i en brukertest gjerne vil ha nyanserte tilbakemeldinger for å styre videre utvikling best mulig. I dette tilfelle var det allerede gjort omfattende brukertest med flere brukergrupper av bachelorprosjektet som dette bygger videre på (Fisketjøn, Hussain og Svendal, 2022). Programflyt for applikasjonen var ikke endret, selv om den var utvidet til å fungere på skrivebord. De tydeligste tilbakemeldingene på den forrige applikasjonen hadde å gjøre med bruk av MET-ID, noe dette prosjektet ikke skulle endre på. En større ulempe ved mangel på storskala brukertest er mangelen på tilbakemelding på de nye visualiseringsalternativene.

Konklusjoner om grad av kryssplattform og overgang til nytt rammeverk er i stor grad basert på egne erfaringer. Prosjektmedlemmene hadde lite erfaring med .NET utvikling og ingen erfaring med MAUI, og en del av utfordringene som er beskrevet i rapporten kunne vært unngått med mer erfaring. Utvikling som krevde løsninger utenom de eksemplene som fantes i Microsoft sin dokumentasjon kan ikke garanteres å ha fulgt anbefalt praksis og standard. Erfaringene presentert i rapporten er likevel av verdi for andre i samme posisjon som vurderer om teknologien passer til deres prosjekt.

Mens .NET er godt etablert i industrien er MAUI relativt nytt og det lyktes ikke å finne forskningsartikler som omhandlet rammeverket. Kildene som er brukt omhandler kryssplattform-utvikling for mobil, som ikke kan sammenlignes med tanke på utfordringer med

selve brukergrensesnittet. Forskjellene mellom mobil og skrivebord er naturligvis større med tanke på skjermstørrelse, konvensjoner for navigering og annen oppførsel.

7.3 Videre arbeid

Evaluering med prosjekteier viste at applikasjonen er moden for videre testing i brannvesen. Dette vil være første oppgave for videre arbeid. Visualiseringsalternativer behøver tilbakemelding fra brukere slik at de kan forbedres ytterligere. Det vil også være nyttig å høre hva brukere tenker om begrepene og fargene som skal forklare brannrisiko i trehus. Hvordan applikasjonen oppleves på skrivebord vil være viktig informasjon, fordi bruk av .NET MAUI fører til løsninger som ikke nødvendigvis følger like standarder som de som utvikles med skrivebord-spesifikke rammeverk.

Videre arbeid med problemstillingen bør også ha fokus på mulighetene for *Proxy Server* (mellomtjener) og *Fog Computing* (tåkedatabehandling) for beregning og lagring av brannrisiko. Dette er nødvendig før applikasjonen kan tilbys til offentligheten, fordi mengden datatrafikk generert til MET sine API-er må begrenses. I forbindelse med dette vil også behovet for oppretting og kopiering av MET-ID utelukkes. Det vil måtte gjøres noen endringer i tjenesteklientene i applikasjonen for å flytte dem til skytjenester, men dette vil også begrense størrelsen på applikasjonen, noe som er bra. Flere mulige endringer og forbedringer beskrives i detalj i *systemdokumentasjon* (vedlegg 4, kapittel 8).

FIGURLISTE

FIGUR 1. HØYNIVÅ-ARKITEKTUR FOR .NET MAUI (MICROSOFT, 2023A). 10
FIGUR 2. ARBEIDSFLYT FOR EN ARBEIDSOPPGAVE (ISSUE). 11
FIGUR 3. BRUKSTILFELLEDIAGRAM. 15
FIGUR 4. DOMENEMODELL. 16
FIGUR 5. FORHOLD MELLOM VIEW, VIEWMODEL OG MODEL I MVVM (STONIS, 2022). 18
FIGUR 6. BINDING AV ISBUSY TIL ACTIVITYINDICATOR I ADDLOCATIONPAGE. 18
FIGUR 7. ISBUSY SETTES TIL TRUE I ADDLOCATIONVIEWMODEL NÅR GEOLOKASJON HENTES. 18
FIGUR 8. BRUK AV OBSERVABLEPROPERTY I MYLOCATIONSVIEWMODEL. 19
FIGUR 9. AUTOGENERERT KODE VED BRUK AV OBSERVABLEPROPERTY. 19
FIGUR 10. BRUK AV RELAYCOMMAND I MYLOCATIONSVIEWMODEL. 20
FIGUR 11. AUTOGENERERT KODE VED BRUK AV RELAYCOMMAND. 20
FIGUR 12. PROGRAMVAREARKITEKTUR. 21
FIGUR 13. FORESPØRSEL TIL ENDEPUNKT FOR MÅLESTASJONER. 22
FIGUR 14. FORESPØRSEL TIL ENDEPUNKT FOR OBSERVASJONER. 22
FIGUR 15. FORESPØRSEL TIL ENDEPUNKT FOR PROGNOSER. 23
FIGUR 16. BRANNRISIKO, FARGER OG TTF-VERDIER I RELASJON. 24
FIGUR 17. OBSERVATION-OBJEKT TIL BRUK I FIRE RISK MODEL. 25
FIGUR 18. GRENSESNITT IFIRERISK I FIRE RISK MODEL. 26
FIGUR 19. FIRERISKLOCATION-OBJEKT FRA MODELS, FORENKLET. 26
FIGUR 20. SEKVENSDIAGRAM FOR LAGRING AV NY LOKASJON, MED BEREGNET BRANNRISIKO. 27
FIGUR 21. TILSTANDSMASKIN (FISKETJØN, HUSSAIN OG SVENDAL, 2022). 28
FIGUR 22. TABELLER I LOKAL DATABASE. 29
FIGUR 23. DESERIALISERING VED LASTING AV DATA FRA DATABASE. 30
FIGUR 24. BRUK AV ONIDIOM. 31
FIGUR 25. BRUK AV DEVICEINFO.IDIOM. 31
FIGUR 26. HIERARKI OVER KONTROLLER FOR GRAFISK BRUKERGRENSESNITT I .NET MAUI. 32
FIGUR 27. SHELL FOR NAVIGASJON VELGES UT IFRA PLATTFORM. 34
FIGUR 28. SHELL FOR MOBIL. 35
FIGUR 29. SHELL FOR SKRIVEBORD. 35
FIGUR 30. INSTANSIERING AV LISTEN LOCATIONS I MYLOCATIONSVIEWMODEL. 36
FIGUR 31. LOCATIONS SETTES SOM KILDE TIL COLLECTIONVIEW I MYLOCATIONSPAGE. 36
FIGUR 32. DEFINERING AV DATATEMPLATES SOM RESSURSER. 37
FIGUR 33. DATATEMPLATE VELGES UT IFRA PLATTFORM. 37
FIGUR 34. EXPANDER PÅ MOBIL. 38
FIGUR 35. LOCATIONSPAGE FOR EN BESTEMT LOKASJON PÅ SKRIVEBORD. 39

FIGUR 36. TRENDPIL PÅ SKRIVEBORD. 39
FIGUR 37. DATATRIGGER SOM BESTEMMER HVILKEN CELLE SOM VISES. 40
FIGUR 38. STOLPEDIAGRAM PÅ MOBIL. 41
FIGUR 39. GRAF MED TTF-VERDIER OG VIND, PÅ SKRIVEBORD. 42
FIGUR 40. TABELL PÅ SKRIVEBORD. 43
FIGUR 41. FARGEPALETT BRUKT I APPLIKASJONEN. 44
FIGUR 42. STRUKTUR FOR STILER OG FARGER. 44
FIGUR 43. FARGEN MED NØKKEL PRIMARY DEFINERES I COLORS.XAML. 45
FIGUR 44. STILEN MED NØKKEL TOOLBARBUTTONSTYLE DEFINERES I STYLES.XAML. 45
FIGUR 45. BRUK AV STIL FOR BUTTON I MYLOCATIONSPAGE.XAML. 45
FIGUR 46. MY_LOCATIONS TEKSTSTRENG I APPRESOURCES.RESX. 46
FIGUR 47. BRUK AV TEKSTSTRENG MY_LOCATIONS, SOM BLIR OVERSATT VED KJØRING. 46
FIGUR 48. FORSLAG TIL ENDRINGER FRA PROSJEKTEIER ETTER DEMONSTRASJON. 48

8 REFERANSER

Decode (2023) *Top challenges of developing cross-platform mobile apps*. Tilgjengelig fra: https://decode.agency/article/cross-platform-development-challenges/ (Hentet: 8. mai 2023)

DYNAMIC. (2021) *Reducing fire disaster risk through dynamic risk assessment and management.* Tilgjengelig fra: https://app.cristin.no/projects/show.jsf?id=2495578 (Hentet: 23. februar 2023)

Fisketjøn, E.H., Hussain, A.T. og Svendal, T. (2022) *A Mobile Application for Fire Risk Notification based on Edge Computing*. Bacheloroppgave. Høgskulen på Vestlandet.

Flutter (uten år) *Flutter documentation*. Tilgjengelig fra: https://docs.flutter.dev/ (Hentet: 20. februar 2023)

Halderaker, E.D. og Evjenth, A. (2021) *Development and Evaluation of a Software System for Fire Risk Prediction*. Masteroppgave. Høgskulen på Vestlandet.

Høgskulen på Vestlandet (2022) *Storbranner*. Tilgjengelig fra: https://www.hvl.no/forsking/gruppe/storbranner/ (Hentet: 22. februar 2023)

Karlsson, B. og Quintiere, J.G. (2000) *Enclosure Fire Dynamic*. Boca Raton: CRC Press LLC.

Kraaijeveld, A. (2016) *Burning Rate and Time to Flashover in Wooden ¼ scale Compartments as a Function of Fuel Moisture Content.* 10th International Fire Science & Engineering Conference; Windsor, UK.

Log, T. (2019) *Modeling Indoor Relative Humidity and Wood Moisture Content as a Proxy for Wooden Home Fire Risk*. Sensors

Martin, R.C. og Martin, M. (2006) *Agile Principles, Patterns, and Practices in C#.* US: Pearson Education.

Martin, R.C. (2008) *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall.

Meta Platforms (uten år) *Who is using React Native?* Tilgjengelig fra: https://reactnative.dev/showcase (Hentet 8. mai 2023)

Meteorologisk institutt (2017) *Om Meteorologisk institutt.* Tilgjengelig fra: https://www.met.no/om-oss/om-meteorologisk-institutt (Hentet 5. mai 2023)

Meteorologisk institutt (uten år, a) *Welcome to the MET Weather API*. Tilgjengelig fra: https://api.met.no (Hentet: 15. februar 2023)

Meteorologisk institutt (uten år, b) *What is Frost?* Tilgjengelig fra: https://frost.met.no/index.html

(Hentet 5. mai 2023)

Meteorologisk-Institutt (uten år, c) *Locationforecast*. Tilgjengelig fra: https://api.met.no/weatherapi/locationforecast/2.0/documentation (Hentet: 5.mai 2023)

Direktoratet for samfunnssikkerhet og beredskap (2021) *Ny og bedre farevarsling for skogbrann.* Tilgjengelig fra: https://www.dsb.no/nyhetsarkiv/2021/ny-og-bedre-farevarsling-for-skogbrann/ (Hentet 8. mai 2023)

Microsoft (2023a) *What is .NET MAUI.* Tilgjengelig fra: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui (Hentet: 8. mai 2023)

Microsoft (2023b) *Introduction to the MVVM Toolkit*. 2023. Tilgjengelig fra: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/ (Hentet: 20. april 2023)

Microsoft (uten år) *.NET Multi-platform App UI documentation.* Tilgjengelig fra: https://learn.microsoft.com/en-us/dotnet/maui/ (Hentet: 8. mai 2023)

Røhr-Staff, M. (2019). Det viktig arbeidet med å unngå katastrofebrann. *Brennaktuelt.no.* Tilgjengelig fra: https://www.brennaktuelt.no/brann-brannforskning-brannsikkerhet/det-viktige-arbeidet-med-a-unnga-katastrofebrann/108831 (Hentet: 23. februar 2023)

SAFe (2023) *Domain Modeling* Tilgjengelig fra: https://scaledagileframework.com/domain-modeling/ (Hentet 21. mai 2023)

Scrum (uten år) *What is Scrum?* Tilgjengelig fra: https://www.scrum.org/learning-series/what-is-scrum/what-is-scrum (Hentet: 9. februar 2023)

Smith, J. (2009) *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern.* Tilgjengelig fra: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (Hentet: 19. april 2023)

Stokkenes, S. (2019) *Implementation and Evaluation of a Fire Risk Indication Model*. Masteroppgave. Høgskulen på Vestlandet.

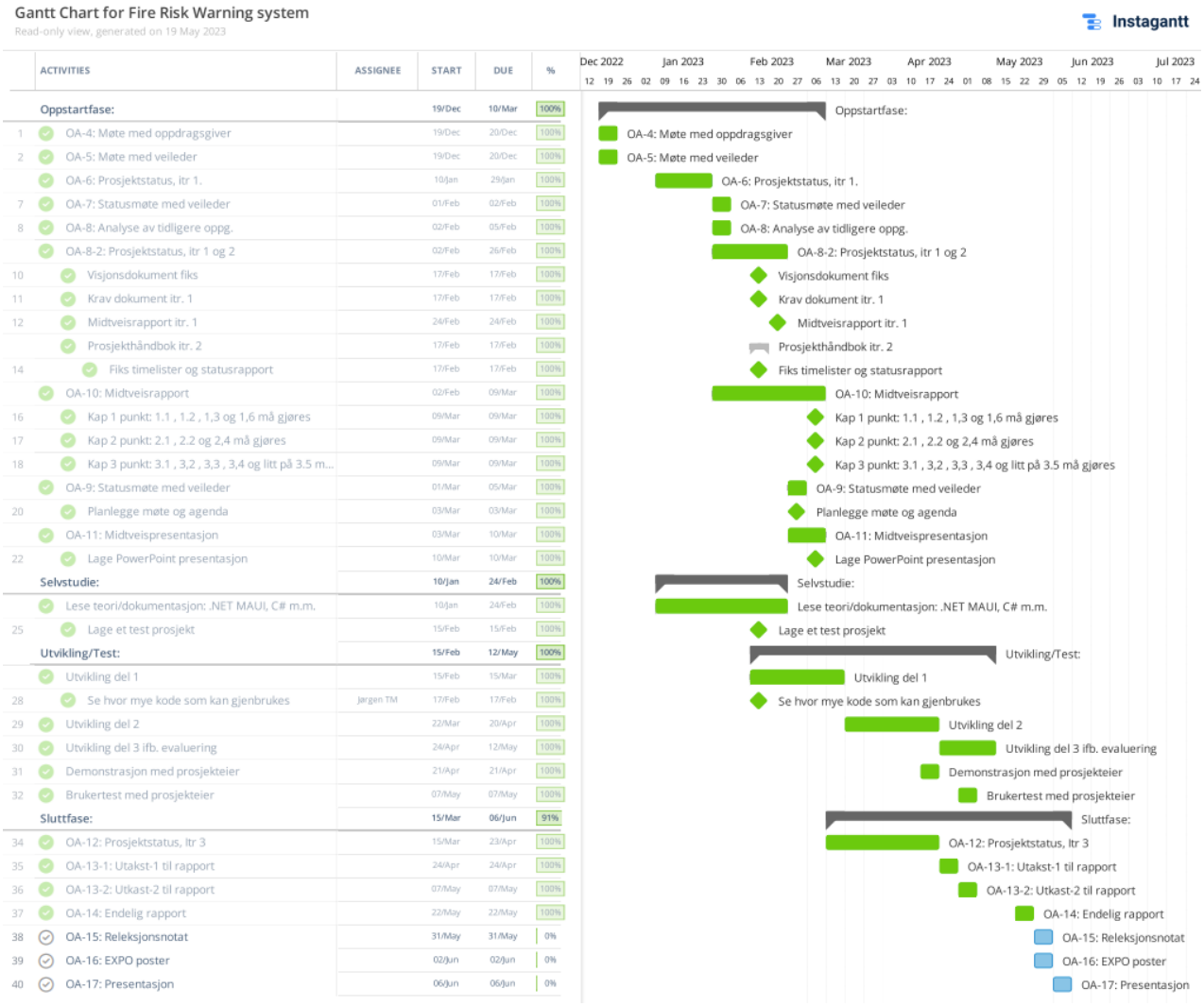
Stonis, M. (2022) *Enterprise Application Patterns Using .NET MAUI.* Redmond, Washington: Microsoft Developer Division, .NET, and Visual Studio product teams.

SQLite (uten år) *About SQLite*. Tilgjengelig fra: https://www.sqlite.org/about.html (Hentet: 15.mars 2023)

Telerik (uten år) *The Most Comprehensive UI Library of .NET MAUI Controls*. Tilgjengelig fra: https://www.telerik.com/maui-ui (Hentet 5. april 2023)

9 VEDLEGG

9.1 Gantt-diagram



9.2 Risikovurdering

	Hendelse /Risiko	Årsak	Sannsynlighet	Konsekvens	Risiko-produkt	Tiltak
1	.NET MAUI utfordringer vi ikke klarer å løse	Rammeverk er nytt, kan støte på begrenset dokumentasjon	2	5	10	Undersøke kvaliteten på dokumentasjon på forhånd.
2	Applikasjonen blir ikke komplett i tide	Forsinkelser i utvikling	5	3	15	Jobbe iterativt og smidig, slik at vi hele tiden har en fungerende løsning
3	Applikasjonen tas ikke i bruk	Den treffer ikke brukerbehov	3	4	12	God dialog med stipendiat hos HVL som har kunnskap og vet brukerbehov
4	Gruppemedlem mer står fast i sine oppgaver	Mangel på kunnskap, dårlig definering av oppgaver	3	4	12	Hyppige møter for å oppdatere team/veileder. Sette seg inn i det tekniske
5	Involverte er utilgjengelig for arbeid	Sykdom	4	3	12	Verktøy som tillater individuelt arbeid (github, gantt online)
6	Tap av kildekode	Hardware (utstyr) slutter å fungere	2	5	10	Lagring av kildekode på github
7	Ufullstendig testing av plattformer	Ikke tilgang / tid til å teste på alle 4 plattformer	3	4	12	Skaffe tilgang, evt. avgrense mål om testing på alle plattformer

9.3 Risikomatrise

Sannsynlighet	Svært Høy (5)	5	10	15	20	25
	Høy (4)	4	8	12	16	20
	Middels (3)	3	6	9	12	15
	Lav (2)	2	4	6	8	10
	Svært Lav (1)	1	2	3	4	5
		Svært Lav (1)	Lav (2)	Middels (3)	Høy (4)	Svært Høy (5)
	Konsekvens					

9.4 Intervju til brukertest

Test FireGuard2023

Testen vil gjennomføres i to deler; En på Android mobil og en på Windows skrivebord. Etter hver del vil det være et intervju med spørsmål til brukeropplevelsen, svar må gjerne utdypes. Til slutt følger et kort intervju med spørsmål som angår applikasjonen som en helhet. Dersom tiden ikke strekker til, vil intervjuene gjennomføres ved en senere anledning.

Agenda: Test skrivebord, intervju skrivebord, test mobil, intervju mobil, intervju felles for applikasjon.

Applikasjonen testes i reell tilstand, uten bruk av liksom-data, på begge plattformer. Applikasjonen vil testes med engelsk språk på skrivebord, og norsk språk på mobil (begge språk er egentlig tilgjengelig på begge plattformer). Oppgavene brukeren skal gjennomføre er like for begge plattformer.

Oppgaver for brukeren:

1. Følg instruksjoner for å legge til MET-ID : «…».

(Skal være tilgjengelig på utklippstavle/autofullføring, pass på at mellomrom ikke inkluderes)

2. Følg instruksjoner for å gå til side for å legge til lokasjon.

3. Skriv navn på lokasjon: «Bergen», trykk på knapp for å «hente posisjon».

4. Trykk på lagre-knapp.

5. Se at lokasjon legges til i listen for «mine lokasjoner».

6. Trykk på lokasjon for å se mer informasjon om brannrisiko.

7. Endre visning for brannrisiko ved å trykke på knappen «endre visning». Prøv flere ganger.

8. Trykk på knapp for å slette lokasjon, og se at lokasjon forsvinner fra listen.

9. Utforsk «Hjelp»-siden.

Etter at disse oppgavene er gjennomført er brukeren fri til å eksperimentere med applikasjonen dersom det er funksjoner som er uklare. Etter dette gjennomføres intervju.

Intervju i forbindelse med test med prosjekteier, FireGuard2023 – Skrivebordapplikasjon

Deltaker:

Dato:

Sted:

Til stede:

Plattform:

1. Hvor responsiv er applikasjonen? (lite-middels-svært)
2. Hvor godt fungerer navigasjonen i applikasjonen? (dårlig-ok-godt)
3. Er det noe du forventet å finne/se som du ikke gjorde?
4. I hvilken grad gir stolpediagrammet en god visualisering av brannrisiko?
5. I hvilken grad gir grafen en god visualisering av brannrisiko?
6. I hvilken grad gir tabellen en god visualisering av brannrisiko?
7. Hvor godt fungerer funksjon for å hente posisjon automatisk?
8. Hvor effektiv er funksjonen for å legge til flere lokasjoner?
9. Hvor effektiv er funksjonen for sletting av lokasjon?
10. Hvor godt fungerer siden for innstillinger (siden for MET-ID)?
11. Hvor godt fungerer hjelpesiden?
12. Noe du vil tilføye?