Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Kryssplattform-applikasjon for varsling av brannrisiko

(Cross-Platform Application for Fire Risk Notification)

Systemdokumentasjon

Versjon <2.0>

Dokumentet er basert på Systemdokumentasjon utarbeidet ved NTNU. Revisjon og tilpasninger til bruk ved IDER, DATA-INF utført av Carsten Gunnar Helgesen, Svein-Ivar Lillehaug og Per Christian Engdal. Dokumentet finnes også i engelsk utgave.

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
27.04.2023	1.0	Første versjon	Jørgen
19.05.2023	2.0	Endelig versjon	Jørgen og Johan

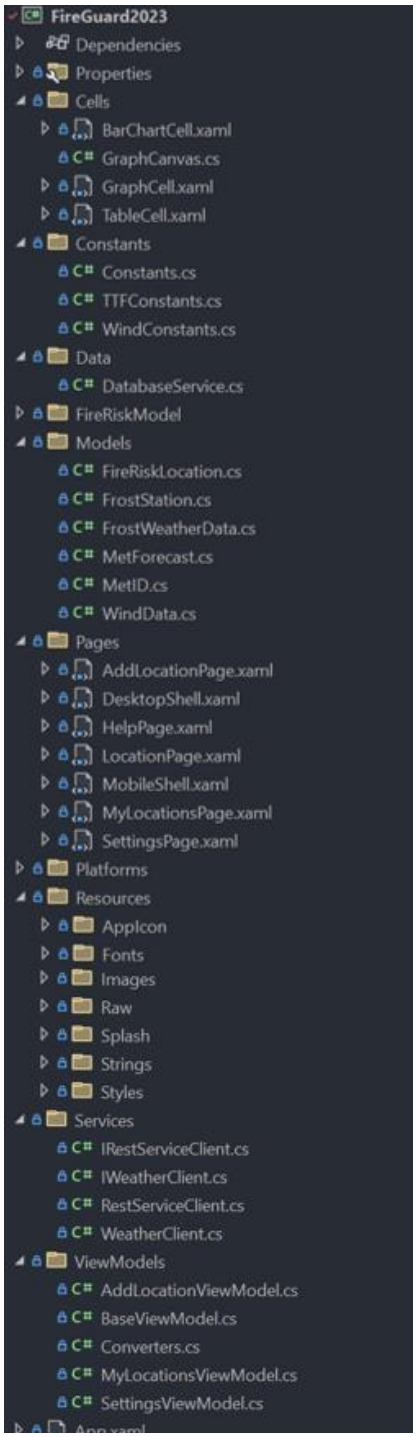
INNHOLDSFORTEGNELSE

1 INNLEDNING

Dette dokumentet beskriver systemet som inngår i applikasjon utviklet i forbindelse med bachelorprosjekt «Kryssplattform-applikasjon for varsling av brannrisiko». Applikasjonen er utviklet i rammeverket .NET MAUI som muliggjør utvikling av applikasjon til Android, iOS, Windows og Mac ved hjelp av en samlet kodebase. Hensikten med dokumentet er å gi teknisk innsikt i løsningen som er utviklet, og veilede de som skal kjøre applikasjonen og eventuelt arbeide videre med den.

2 PROSJEKTSTRUKTUR

Figur 1 viser mappestruktur for prosjektet. *FireRiskModel* ligger som mappe i prosjektet, men burde egentlig være et bibliotek som legges til i *solution* med en avhengighet. Grunnen til at det ligger som mappe nå er endringer som måtte gjøres med en del metoder underveis i utvikling for å fasilitere for ny bruk av enhet for brannrisiko.

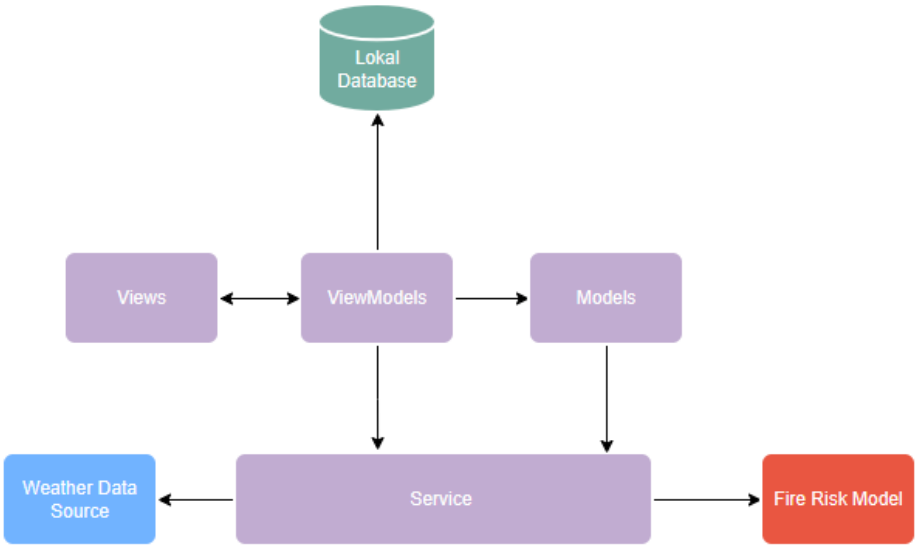


Figur 1. Mappestruktur for prosjekt.

3 ARKITEKTUR

3.1 Programvarearkitektur

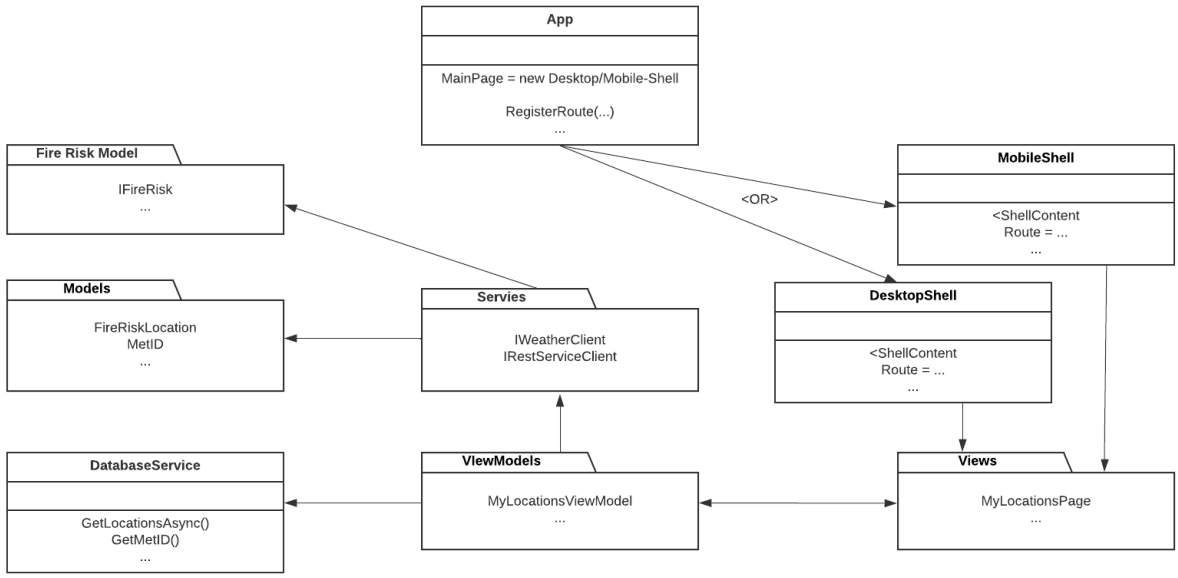
Applikasjonen benytter designmønsteret *Model-View-ViewModel*. Figur 2 viser systemarkitektur for applikasjonen, hvor komponentene fra MVVM utgjør hoveddelen av applikasjonen (i lilla). I tillegg kommuniserer ulike *ViewModels* med databasen gjennom *DatabaseService*, og *Service*-klasser (klientklasser i mappen Services) kommuniserer med MET-API for å hente værdata, og *Fire Risk Model* for å beregne brannrisiko ut ifra disse.



Figur 2. Programvarearkitektur.

3.2 Overordnet klassediagram

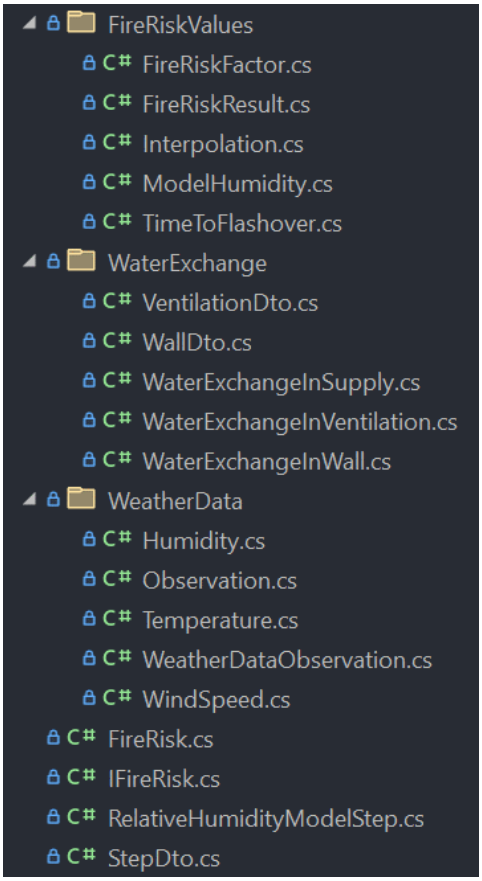
Figur 3 viser overordnet klassediagram for applikasjonen. *Views* er i applikasjonen kalt *Pages*. Alle *Pages*, *ViewModels* og *Services* registreres i *MauiProgram.cs* for *dependency injection*; Applikasjonen injiserer disse komponentene automatisk der det trengs i konstruktører. I *App.xaml.cs* bestemmes det ut ifra plattform hvilken Shell som skal opprettes, *Mobil*-eller-*Desktop*.



Figur 3. Klassediagram for applikasjonen.

3.3 Fire Risk Model

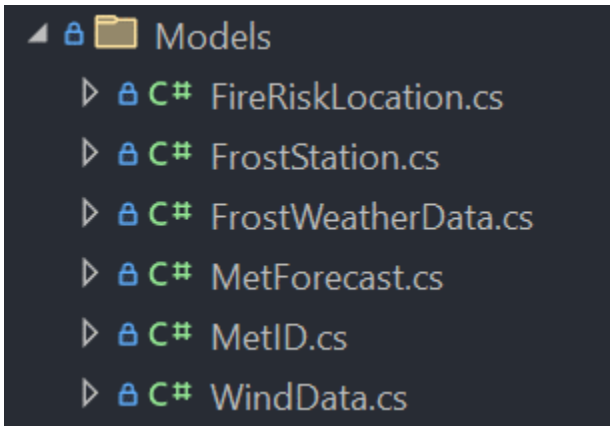
I forbindelse med modell for beregning av innendørs brannrisiko i trehus for i dag og tre dager fremover behøves flere klasser. Modellen som benyttes (Log T.) er tilgjengelig i C#. Det refereres til (Fisketjøn, Hussain og Svendal, 2022) for mer informasjon om modellen. Klassediagram for *Fire Risk Mode*l vil ikke vises her da denne er omfattende, men figur 4 viser mappestruktur for modellen:



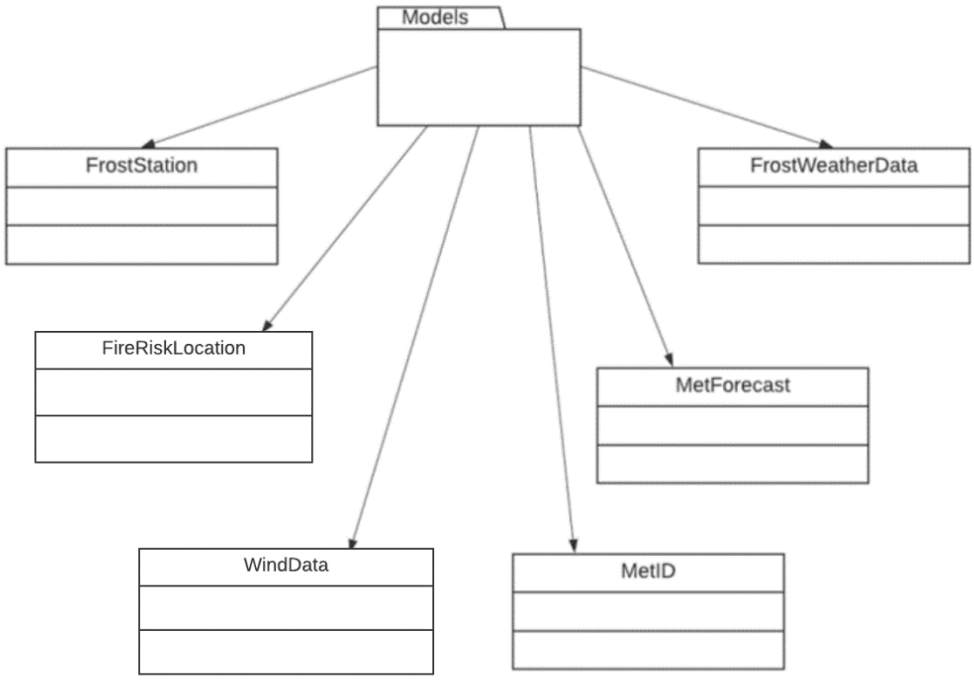
Figur 4. Mappestruktur for Fire Risk Model.

3.4 Models

De klassene og metodene som inngår i *Models* håndterer tilstanden til data som brukes i applikasjonen. Dette er data for lokasjoner med brannrisiko, MET-ID og klasser som brukes når applikasjonen skal hente værdata fra MET og bruke *Fire Risk Model* til beregninger. Figur 5 viser mappestruktur for *Models* mens figur 6 viser klassediagram for *Models*.



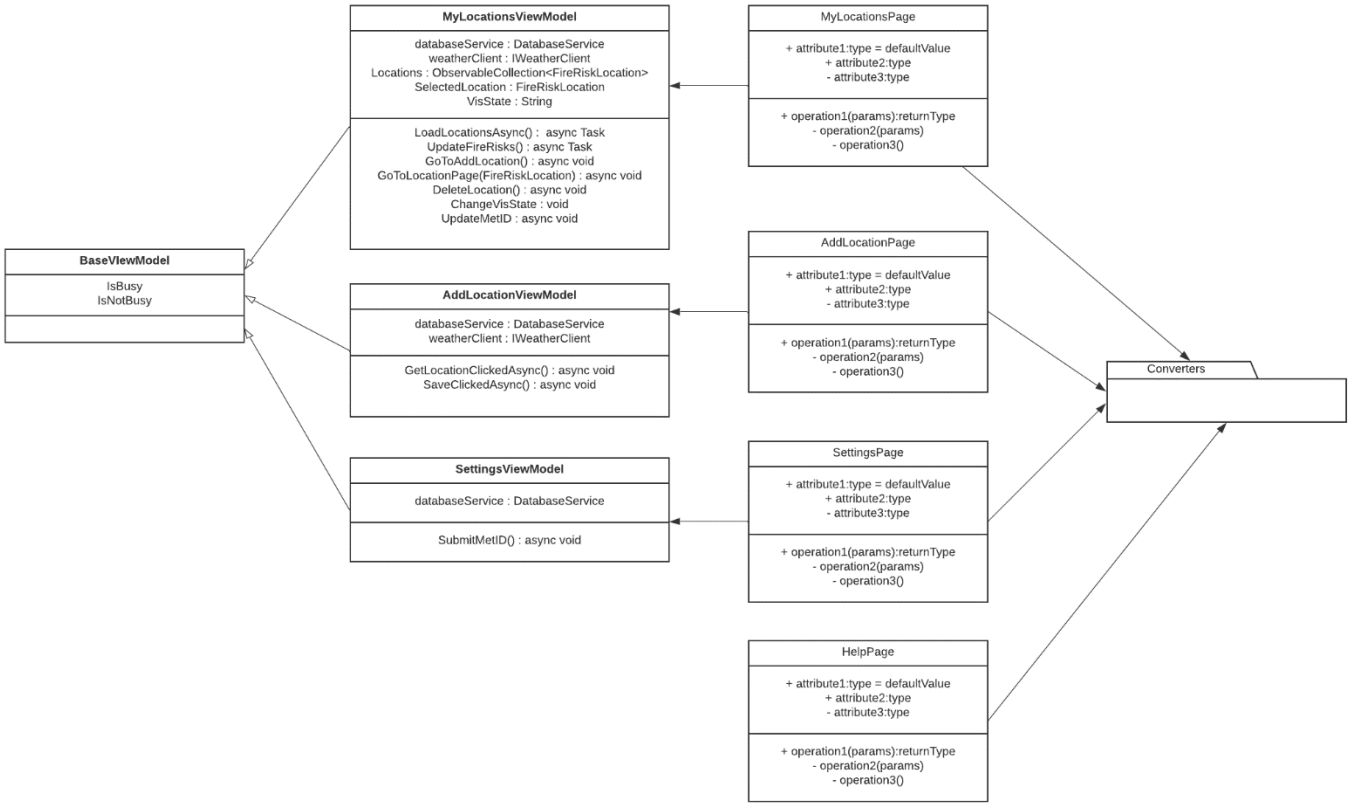
Figur 5. Mappestruktur for Models.



Figur 6. Klassediagram for Models.

3.5 Views (Pages) og ViewModels

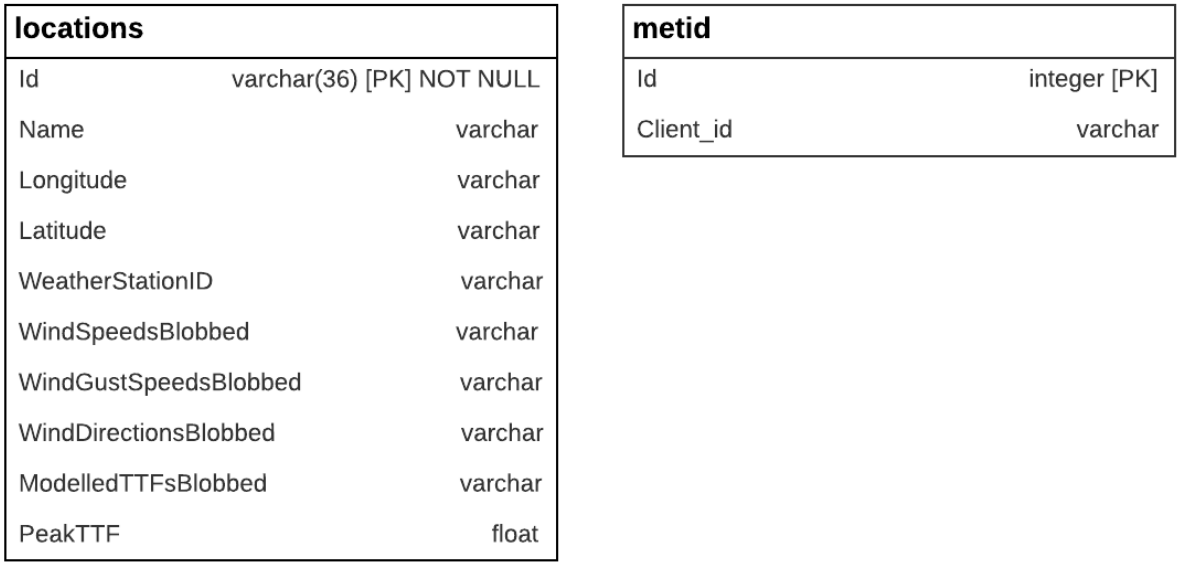
Figur 7 viser forhold mellom *Views* og *ViewModels*. Alle *ViewModels* arver klassen *BaseViewModel* som har to *Observable Properties*, *IsBusy* og *IsNotBusy*. Disse blir brukt i *Views* for å vise at lokasjonen holder på med en tidkrevende operasjon i bakgrunnen.



Figur 7. Klassediagram for Pages og ViewModels.

4 DATABASEMODELL

Applikasjonen bruker en SQLite-database som lagrer data i filsystemet lokalt på hver enhet der den kjører (SQLite, 2023). I databasen lagres lokasjoner med beregnet brannrisiko og brukerens MET-ID som trengs for å gjøre API-kall til værtjenester. Klassen *DatabaseService* håndterer alle transaksjoner til og fra databasen. Figur 8 viser tabeller i lokal database. Merk at listeverdier for vind, vindretning og modellert TTF lagres som tekststrenger (dette forklares på neste side).



Figur 8. Tabeller i database.

For å lagre listene med *ModelledTTF*, *WindDirection* og *WindSpeed* brukes JSON-serialisering for å gjøre dem om til *strings*, som vist i figur 10. Dette er for å unngå å opprette egne tabeller for disse verdiene, som bare er 4 stykker per lokasjon. Verdiene må da selvfølgelig og deserialiseres når de hentes ut fra database. Figur 9 viser hvordan *FireRiskLocation*-klassen ser ut med disse ekstra egenskapene for serialiserte verdier (OBS legg merke til [*Ignore*]-attributt som sier at denne egenskapen ikke skal lagres i databasen, det er nemlig bare den serialiserte *stringen* som skal lagres).

```
//creating a new location object
FireRiskLocation newlocation = new()
{
  Id = Guid.NewGuid(),
  Name = Name,
  Longitude = Longitude,
  Latitude = Latitude,
  WeatherStationID = stationId,
  WindDirection = windDirection,
  WindSpeed = windSpeed,
  WindGustSpeed = windGust,
  ModelledTTF = modelledTTF,
  PeakTTF = peakTTF
};

  //Serializing blobbed properties
  newlocation.WindSpeedsBlobbed = JsonConvert.SerializeObject(newlocation.WindSpeed);
  newlocation.WindGustSpeedsBlobbed = JsonConvert.SerializeObject(newlocation.WindGustSpeed);
  newlocation.WindDirectionsBlobbed = JsonConvert.SerializeObject(newlocation.WindDirection);
  newlocation.ModelledTTFsBlobbed = JsonConvert.SerializeObject(newlocation.ModelledTTF);
```

Figur 10. Oppretting av FireRiskLocation-instans, med serialisering.

```
[Table("locations")]
public class FireRiskLocation
{
    [PrimaryKey, AutoIncrement]
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Latitude { get; set; }
    public string Longitude { get; set; }
    public string WeatherStationID { get; set; }

    [Ignore]
    [JsonProperty("WindSpeed")]
    public List<double> WindSpeed { get; set; }
    public string WindSpeedsBlobbed { get; set; }

    [Ignore]
    [JsonProperty("WindGustSpeed")]
    public List<double> WindGustSpeed { get; set; }
    public string WindGustSpeedsBlobbed { get; set; }

    [Ignore]
    [JsonProperty("WindDirection")]
    public List<double> WindDirection { get; set; }
    public string WindDirectionsBlobbed { get; set; }

    [Ignore]
    [JsonProperty("ModelledTTF")]
    public List<double> ModelledTTF { get; set; }
    public string ModelledTTFsBlobbed { get; set; }

    public double PeakTTF { get; set; }
}
```

Figur 9. FireRiskLocation-klasse, med attributter.

5 SERVER-TJENESTER

Applikasjonen tar i bruk API-er fra Meteorologisk Institutt (MET) for å skaffe værdata (MET, 2022a). Disse blir brukt på samme måte som forrige applikasjon, og det henvises til tidligere bachelorprosjekt (Fisketjøn, Hussain og Svendal, 2022) for mer utfyllende informasjon om API-ene. Det vil gis en kort introduksjon til de to som brukes her.

5.1 Weather API Location Forecast Service

Weather API Location Forecast Service gir prognoser for vær basert på lokasjon for 9 dager fremover i tid. De første to-til-tre dagene gis timesvise prognoser, mens etter dette er de for hver 6. time (MET, 2022a).

Et eksempel på en HTTP GET-forespørsel til *locationforecast complete*-endepunktet:

```
https://api.met.no/weatherapi/locationforecast/2.0/complete.json?lat=60.3691&lon=5.3505
```

Parametrene "lat" og "lon" representerer breddegraden og lengdegraden til geokoordinatene. MET anbefaler ikke å bruke mer enn fire desimaler av hensyn til effektiv *caching* og for å unngå blokkering. Parameterne som vanligvis inkluderes i en HTTP GET-forespørsel er følgende: høyde, breddegrad og lengdegrad. Breddegrad og lengdegrad må inkluderes i hver forespørsel, mens høyde er valgfritt.

5.2 Frost API

Frost er et REST-API som gir fri tilgang til MET sine historiske arkiver for værdata (MET, 2022b). Disse dataene inneholder kvalitetssikrede observasjoner. Log sin modell (Log, 2019) bruker historiske værdata for aktuell lokasjon for å kalibrere modellen slik at den gir mest mulig nøyaktige prediksjoner for de enste tre dagers brannrisiko.

For applikasjonen brukes *sources*-endepunkt for å hente nærmeste værstasjon til de aktuelle koordinatene for lokasjonen, og *observations*-endepunktet for å hente ut historiske værdata for denne stasjonen (MET, 2022c).

Et eksempel på en HTTP GET-forespørsel til endepunktet *sources*:

```
https://frost.met.no/sources/v0.jsonld?types=SensorSystem&elements=air_temperature,relative_humidity,wind_speed&geometry=nearest(POINT(5.3327 60.383))
```

Parameteren "types" spesifiserer stasjonstypen. Man kan ekskludere stasjoner som ikke oppfyller kravene ved å spesifisere parameteren "elements" med: *air_temperature*, *relative_humidity*, *wind_speed*.

Når det gjelder parameteren "geometry", spesifiserer den geometrien til en stasjon. Den geografiske plasseringen uttrykkes enten som et enkelt punkt eller som et polygonområde. Syntaksen "nearest(POINT(lon lat))" refererer til elementet nærmest disse koordinatene.

Et eksempel på en HTTP GET-forespørsel til *observations*-endepunktet:

```
https://frost.met.no/observations/v0.jsonld?sources=SN50540&referencetime=2022-04-01/2022-04-02&elements=air_temperature,relative_humidity,wind_speed
```

«Sources» angir hvilken målestasjon man ønsker værdata fra. «Referencetime» angår tidsperiode, angitt i UTC og ISO-8601 (MET, 2022d).

6 TILGANG OG SIKKERHET

For å skaffe tilgang til MET sine API-er, herunder Frost, må brukeren lage en konto (MET, 2022a). Ved å gi epostadresse får brukeren en Client-ID og en Client-Secret. For de offentlige tilgjengelige dataene trengs kun Client-ID, referert til som MET-ID i forbindelse med applikasjonen. Programflyten i applikasjonen leder brukeren til https://frost.met.no/auth/requestCredentials.html for å opprette bruker og ta vare på Client-ID.

Applikasjonen lagrer ingen personlige data, bortsett fra MET-ID for brukeren. Det ble gjennom tidligere prosjekt (Fisketjøn, Hussain og Svendal, 2022) vurdert at MET-ID ikke er sensitiv informasjon, fordi anskaffelse av denne Client-ID-en gjennom MET sine ressurser **kun krever epostadresse**. MET-ID lagres derfor i klartekst i database.

Bruk av kun Client-ID som identifikasjon i forespørsler til MET sine API-er klassifiseres av dem som *Basic Authentication* (MET, 2022e). Det er og mulig å bruke OAuth2 som autentisering, men dette er valgfritt dersom man forespør ikke-konfidensielle data. Basic Authentication regnes som sikkert når web-klient og servertilkobling er sikre (IBM, 2023). Derfor antast det, i dette prosjektet som hos Fisketjøn, Hussain og Svendal sitt, at MET overholder disse forutsetningene med tanke på at de anbefaler *Basic Authentication*.

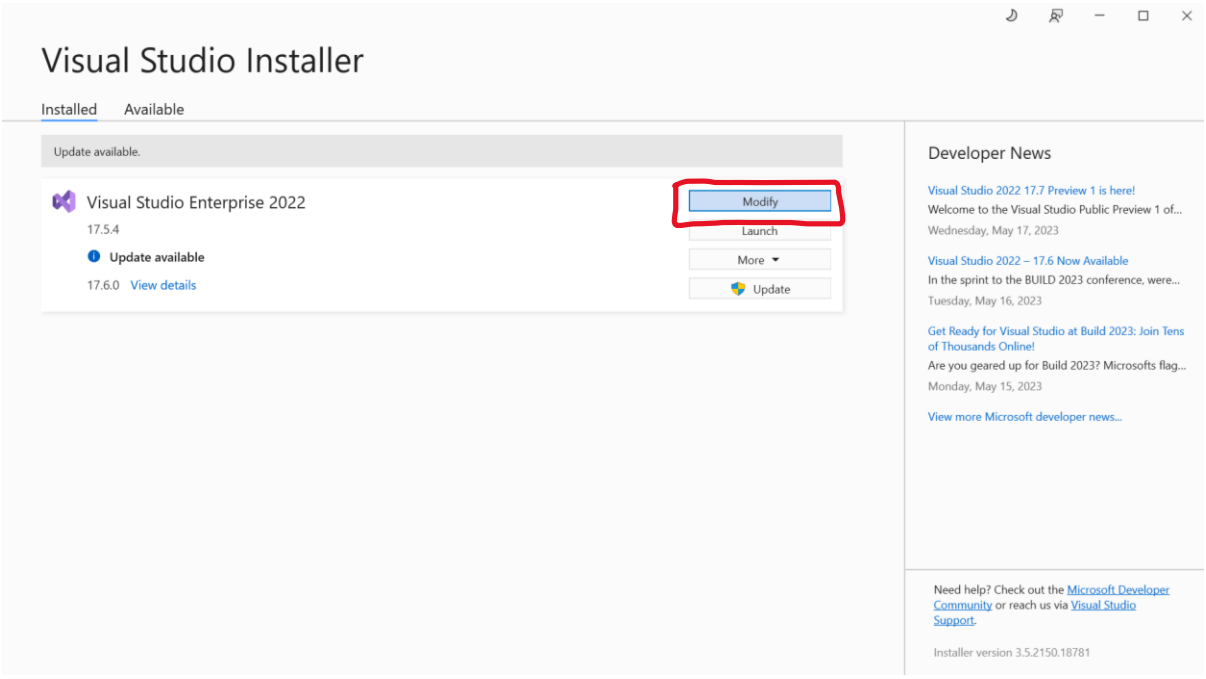
Forespørsler til API-er for værdata i *RestServiceClient* tar alle i bruk *https*-tilkobling.

7 INSTALLASJON OG KJØRING

7.1 Verktøy

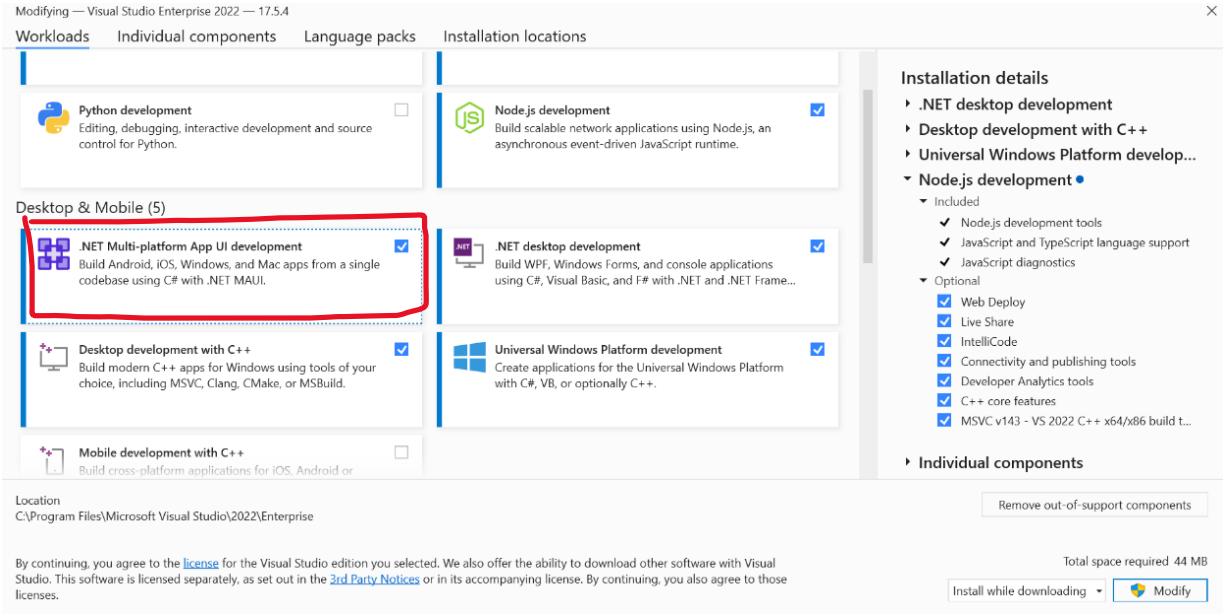
Løsningen er laget ved bruk av Visual Studio 2022 og det anbefales å fortsette å bruke dette som verktøy. Det er mulig å sette opp sitt eget miljø i VSC (Visual Studio Code), men dette krever nok en del tilpasninger og innsats. Visual Studio 2022 har alt som trengs for å utvikle og teste .NET MAUI apper, samt enkelt håndtere pakker.
For å få tilgang til .NET MAUI verktøy i Visual Studio må man laste ned *workload* for dette. Dette kan hukes av når man først laster ned Visual Studio 2022, men dersom man allerede har programmet lastet ned kan man legge til *workload* i Visual Studio Installer, slik som dette:

1. (Figur 11) Trykk på Modify (Dette eksempelet har *Enterprise-edition* av VS2022, men det skal ikke være noen forskjell i fremgangsmåte for *Community-Edition*).



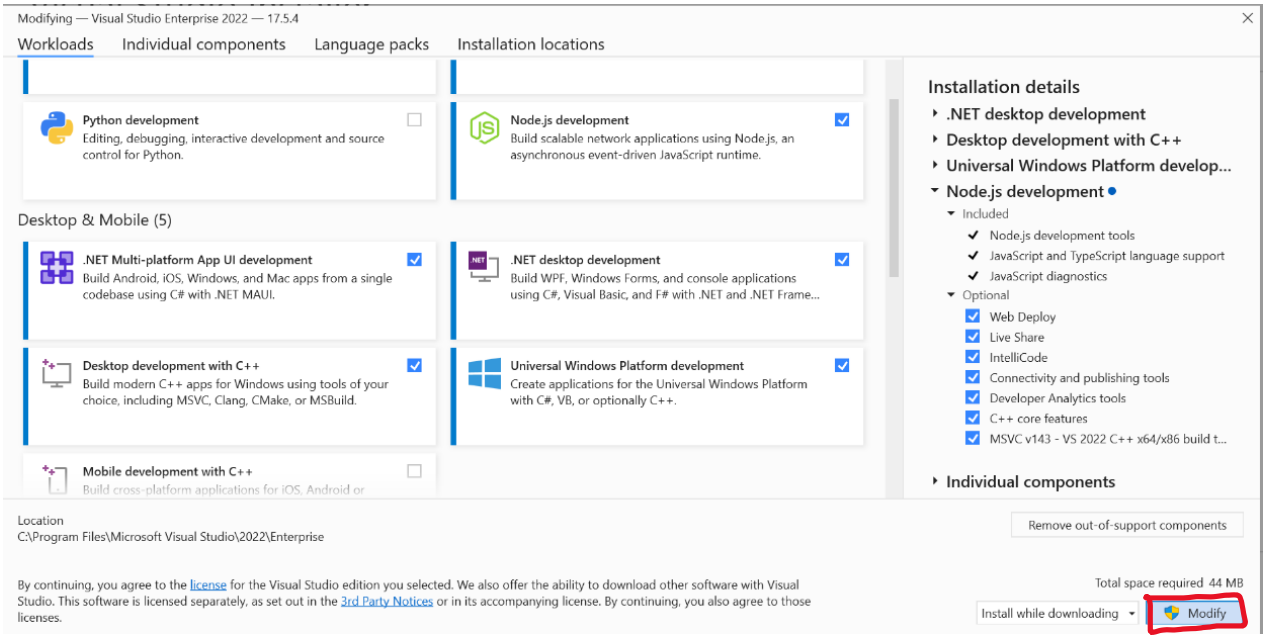
Figur 11. Knapp for redigering av workload for Visual Studio 2022.

2. (Figur 12) Velg *workload* for .NET Multi-platform App UI development.



Figur 12. Valg av workload for .NET MAUI.

3. (Figur 13) Trykk på Modify og godta endringer. Nedlastning av *workload* vil ta en liten stund.



Figur 13. Knapp for lagring av endringer.

7.2 Kloning

GitHub-*repository* for applikasjonen krever autorisert tilgang (spør Lars Michael Kristensen eller evt. Annen ansvarlig veileder). Link til *repository*:

https://github.com/selabhvl/dynamic-mobileap-2023

For å klone prosjekt gjelder det å bruke SSH. Dersom man skal arbeide videre med prosjektet kan det være lurt å lage en fork, men dette har sikkert veileder/prosjekteier noen meninger om. Det kommer an på hvordan det eksisterende arbeidet skal brukes videre.

Prosjektet i løsningen har ingen avhengigheter til bibliotek eller lignende. Prosjektet hadde i utgangspunktet en avhengighet til et bibliotek med kildekode for brannrisikomodell, men klassene fra dette biblioteket ble heller lagt til direkte i prosjektet for å kunne endre et par metoder i *IFireRisk*, slik at brannrisiko nå beregnes som TTF. Det vil være en liten jobb å implementere de samme endringene i det eksterne biblioteket, og deretter lage en ny avhengighet til dette (og da slette mappene med klasser som ligger direkte). Bibliotek for brannrisikomodell er en del av *repository* for bachelorprosjekt 2022, altså forrige applikasjon (mobilapplikasjon). Følgende er lenke til *repository*, som og krever tilgang som nevnt tidligere:

https://github.com/selabhvl/dynamic-mobileapp

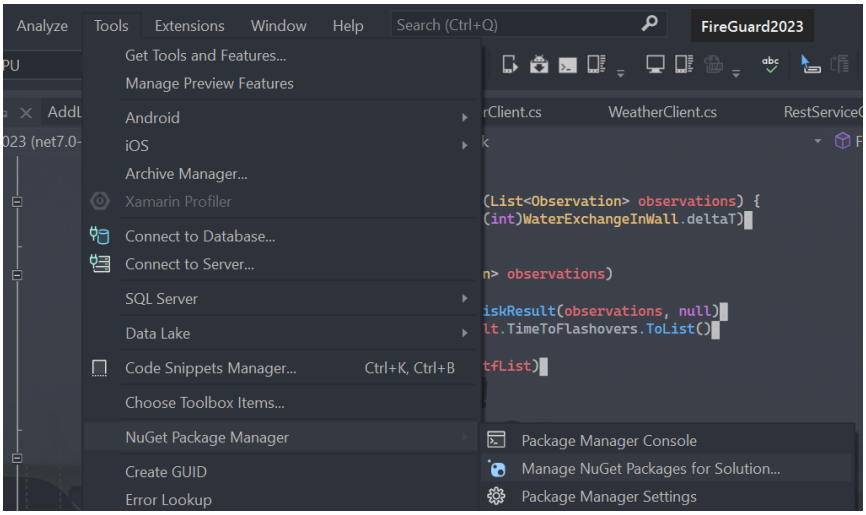
7.3 Pakker

Prosjektet bruker en rekke pakker. Som nevnt tidligere anbefales det å bruke Visual Studio 2022, fordi dette gjør det enklere å vedlikeholde pakker og se oppdateringer. Figur 14 viser listen av pakker i løsningen, som kan sees i *FireGuard2023.csproj*-filen:

```
<ItemGroup>
  <PackageReference Include="CommunityToolkit.Maui" Version="5.0.0" />
  <PackageReference Include="CommunityToolkit.Mvvm" Version="8.1.0" />
  <PackageReference Include="Newtonsoft.Json" Version="13.0.2" />
  <PackageReference Include="RestSharp" Version="108.0.3" />
  <PackageReference Include="sqlite-net-pcl" Version="1.8.116" />
  <PackageReference Include="SQLiteNetExtensions" Version="2.1.0" />
  <PackageReference Include="SQLitePCLRaw.provider.dynamic_cdecl" Version="2.1.4" />
</ItemGroup>
```

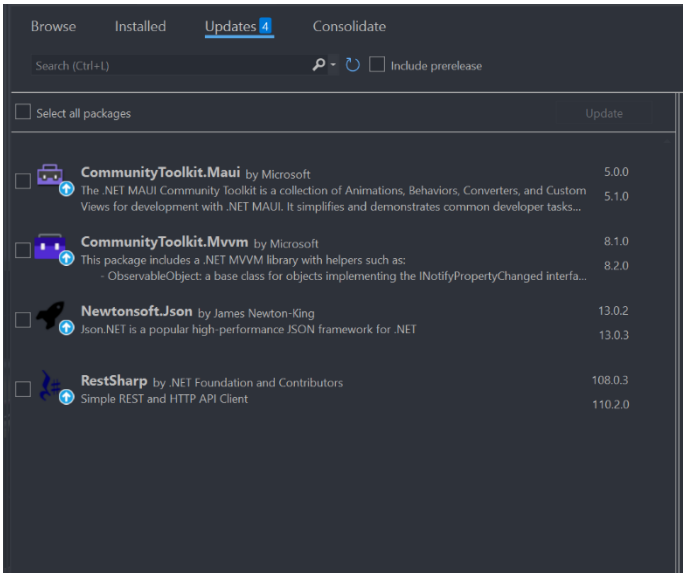
Figur 14. Pakker spesifikke for prosjektet.

I tillegg er også en del pakker installert automatisk gjennom .NET MAUI *Workload* og annet. Full liste av pakker kan lett aksesseres gjennom NuGet *Package Manager*, som vist i figur 15.



Figur 15. Lokasjon for NuGet Package Manager.

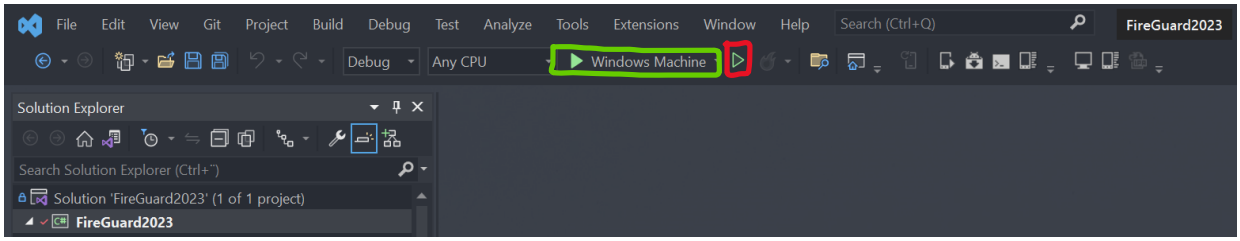
Fra dette verktøyet er det og lett å administrere oppdateringer for pakkene, som vist i figur 16. Fire av pakkene er ikke oppdatert til nyeste versjon. Fra det som er undersøkt inneholder *CommunityToolkit.Maui* forbedringer av feil, men den nye versjonen har blitt testet og løser ikke nevnte problemer i rapporten med for eksempel *Expander*-komponent. *CommunityToolkit.Mvvm* inneholder så vidt gruppen vet mindre endringer. Det forventes at begge *CommunityToolkit*-oppdateringene kan gjennomføres uten krav til store forandringer. For *RestSharp* og *Newtonsoft*-pakkene må det undersøkes nærmere hvilke endringer disse fører med seg. Av det gruppen har undersøkt kan det hende *RestSharp*-oppdateringen endrer hvordan *http*-forespørsler skal gjennomføres.



Figur 16. Pakker som har nye oppdateringer.

7.4 Windows

Kjøring av testing av applikasjon er enkelt på Windows, noe som er selvfølgelig med tanke på at .NET og Visual Studio er Microsoft sine teknologier. Figur 17 viser kontroller i Visual Studio for å kjøre med *debugging* (grønn markering) og uten *debugging* (rød markering).



Figur 17. Knapp for kjøring med og uten debugging.

Ved kjøring med *debugging* på Windows kan også *profiling tool* brukes til å se ytelse og minnebruk for applikasjonen, under Debug > Windows > Show Diagnaostic Tools. *Hot Reload* er automatisk en funksjon ved kjøring med *debugging*, som gjør at endringer i XAML-filene vil vises under kjøring i form av endringer i brukergrensesnitt. Merk at endringer i *Shell*, eller endringer av *Bindings* ikke alltid vil vises automatisk med *Hot Reload*. I tilfeller der man ikke forstår hvorfor endringene ikke synes kan det være lurt å starte applikasjonen på nytt. Endringer i C#-filer vil ikke gi noen effekt før applikasjonen startes på nytt.

7.5 Android

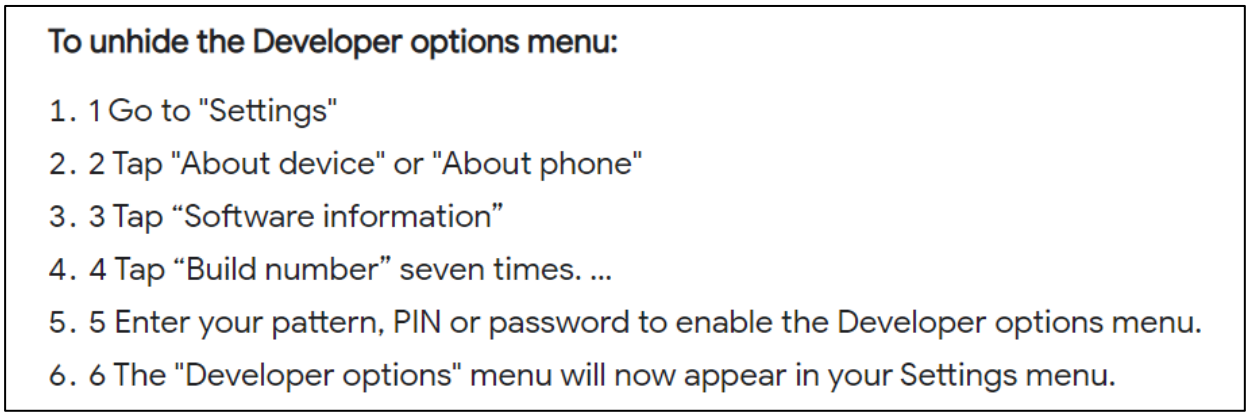
Ved kjøring og testing på Android anbefales det sterkt å anvende en fysisk mobil som kobles til ved USB eller USB-C. Bruk av emulator vil også bli beskrevet her, men prosjektgruppen støtte på store problemer som tok mye tid for å få satt dette opp.

Med en fysisk enhet (OBS! DYNAMIC har anskaffet en Samsung Galaxy til dette formålet, undersøk med veileder/prosjekteier om denne er tilgjengelig) blir *debugging* på mobil effektivt. En Android-*tablet* kan og kobles til for å *debugge* på denne måten, men applikasjonen er foreløpig ikke tilpasset *tablet*. Det er mulig å *debugge* over internett, altså uten USB-kabel, men dette er ikke undersøkt.

7.5.1 Debugging med fysisk mobil:

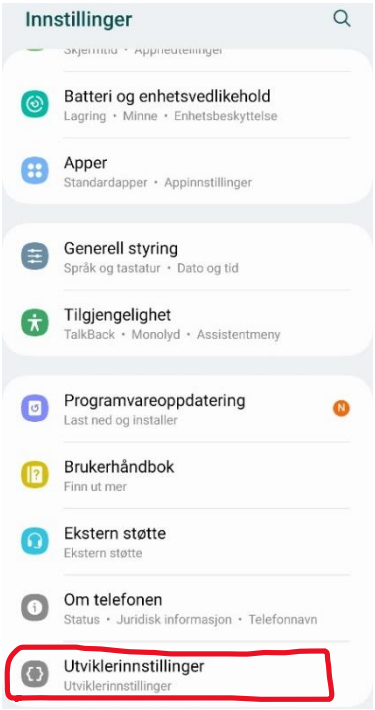
Fremgangsmåte:

1. Sørg for at utviklerinnstillinger på enheten er skrudd på. Det kan være forskjellige måter å gjøre dette på avhengig av enhet, men på mobil som ble anskaffet til prosjektet ble det brukt fremgangsmåte fra Samsung, vist i figur 18:



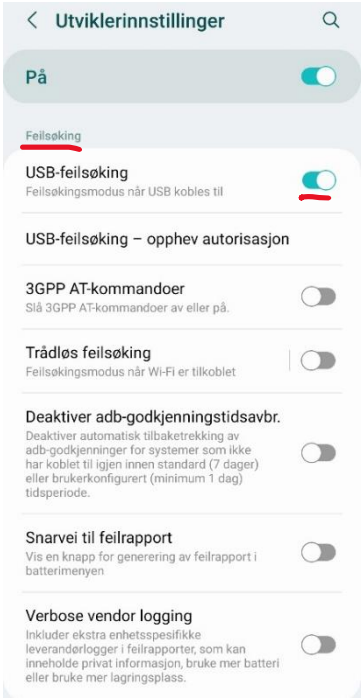
Figur 18. Fremgangsmåte for aktivering av utviklermodus på Samsung.

Figur 19 viser hvordan menyen for utviklerinnstillinger (Developer Options) skal dukke opp:



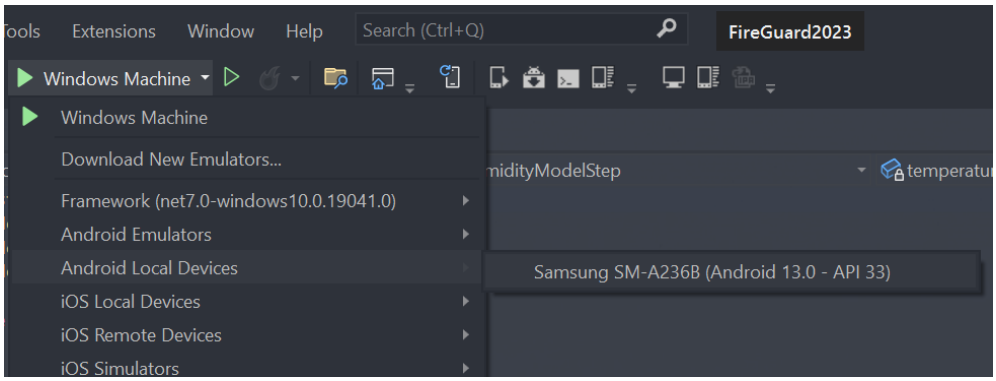
Figur 19. Utviklerinnstillinger under innstillinger på Samsung.

2. Trykk på «Utviklerinnstillinger», og ble ned til «Feilsøking» som vist i figur 20. Aktiver USB-feilsøking.

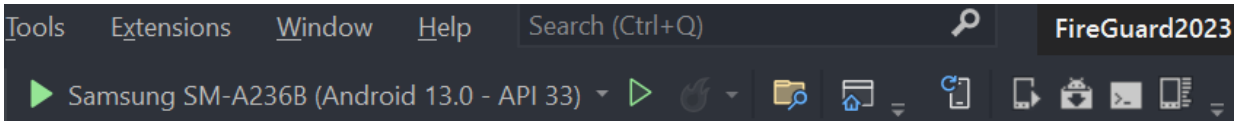


Figur 20. Feilsøking og aktivering av USB-feilsøking.

3. Koble enheten til datamaskin med USB. Nå skal enheten dukke opp under Pil ved navn på aktuell plattform > Android Local Devices > navn på mobil, som vist i figur 21. Velg enheten, slik at den havner der det før stod «Windows Machine», som vist i figur 22.



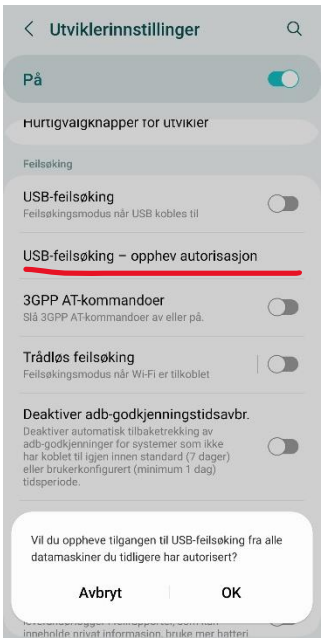
Figur 21. Valg av tilkoblet mobil for kjøring av applikasjon.



Figur 22. Menyvalg for kjøring etter tilkobling av mobil.

Dersom enheten ikke dukker opp når man kobler til med USB gjør følgende:

1. Koble fra USB
2. Skru av USB-feilsøking
3. Trykk på «opphev autorisasjon» (figur 23), og velg ok.



Figur 23. Opphev autoriasasjon for å fikse problemer med tilkobling.

4. Skru på USB-feilsøking
5. Koble til enhet med USB på nytt

4. Enheten er nå klar for å kjøre applikasjonen, med *debugging* eller uten. Trykk på knapp for å kjøre applikasjonen. Første gang, eller etter mange endringer, kan det ta en del tid å bygge og laste ned applikasjonen på enheten. Til slutt skal applikasjonen starte på telefonen.

Hot Reload er også fungerende når man kjører med *debugging* på fysisk enhet. Gjør endringer i XAML-kode og se at endringene skjer i applikasjonen på mobilen. Merk tidligere kommentarer fra Windows-delen angående endringer som muligens ikke reflekteres i applikasjonen, men som krever omstart.

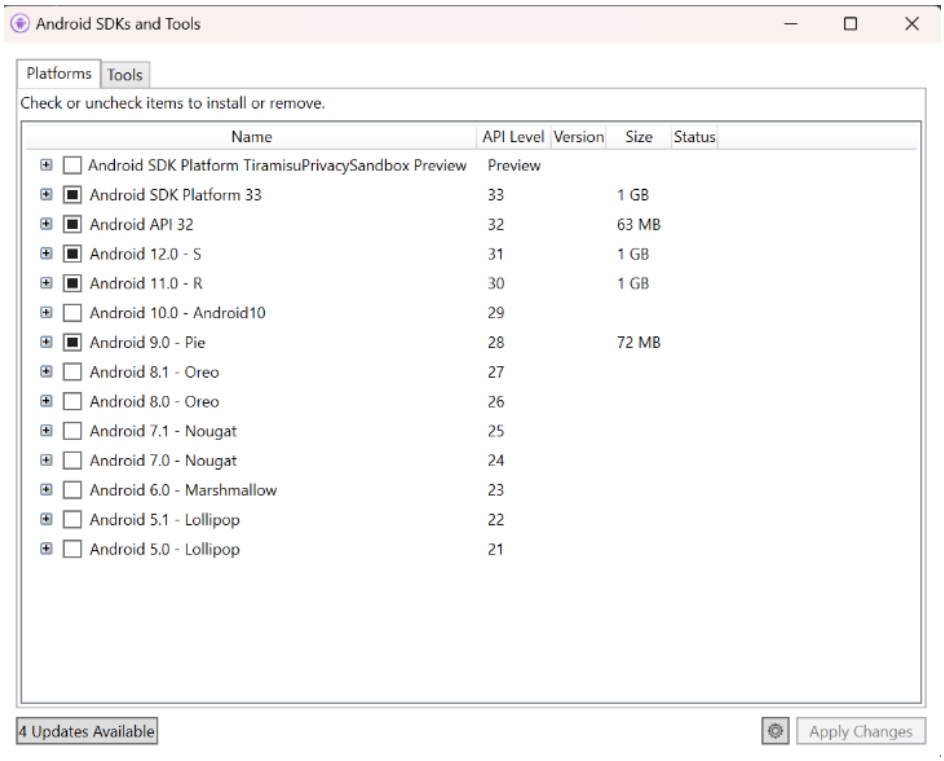
7.5.2 Debugging med emulator

Som nevnt hadde prosjektgruppen store problemer med å få emulator til å fungere optimalt. Det er også viktig å nevne at applikasjonen **MÅ** testes på fysisk enheten fra tid til annen uansett, da emulatoren ikke nødvendigvis klarer å gjenskape all atferd som oppstår med en fysisk mobil (for eksempel er det vanskelig å gjenskape touch-effekten når man bruker museklikk i emulatoren).

Følgende er lenke til Microsoft sin dokumentasjon av hvordan sette opp Android-Emulator på Windows til *debugging* av .NET MAUI applikasjoner:

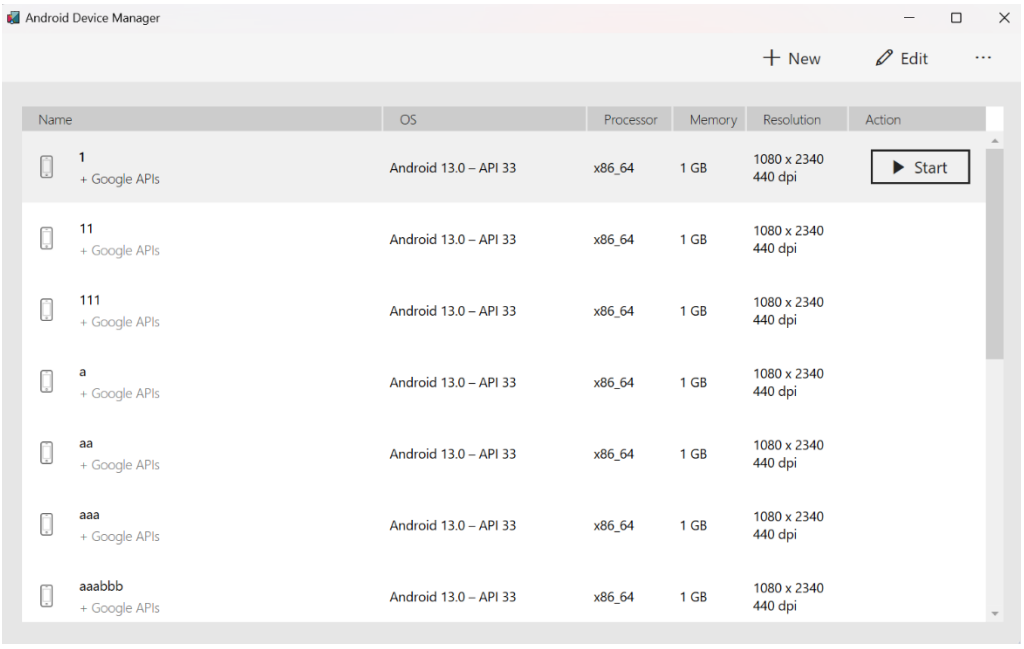
https://learn.microsoft.com/en-us/dotnet/maui/android/emulator/device-manager

For å håndtere Android-SDK versjoner som trengs for å kjøre emulatorer på Windows brukes «Android SDK Manager». Denne finnes i Visual Studio 2022 under Tools > Android > Android SDK Manager. Hvilke versjoner som kreves beskrives i dokumentasjonen som ble gitt over. Vindu for redigering av versjon skal se slik ut (figur 24):



Figur 24. Android SDK Manager.

For å håndtere Android-emulatorer brukes «Android Device Manager». Denne finnes i Visual Studio 2022 under Tools > Android > Android Device Manager. Hvordan legge til emulatorer beskrives i dokumentasjonen som ble gitt over. Vindu skal se slik ut (figur 25, men uten noen enheter lagt til):



Figur 25. Android Device Manager.

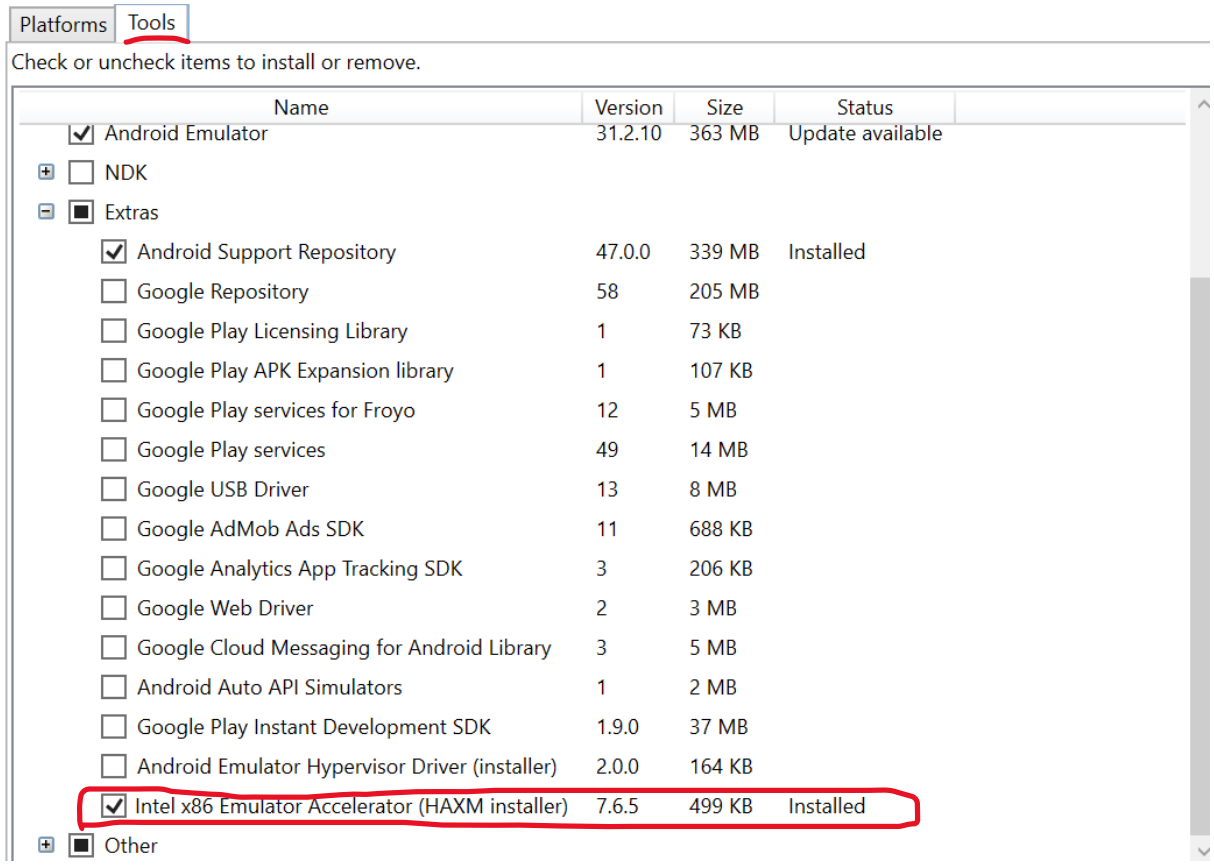
7.5.3 Feil med emulator

Dersom det er behov for testing av mobilapplikasjon og man har fått fysisk enhet til å fungere, anbefales det å ignorere emulator. Dersom man er avhengig av emulator, men ikke får det til å fungere, nevnes det noen tips:

1. **Emulator vil ikke starte i det hele tatt, eller krasjer umiddelbart etter start.** Sørg for at riktige *Tools* er installert, ved å følge dokumentasjon som er gitt over. Dersom man fortsatt har problemer kan det være fordi emulatoren trenger «Hardware Acceleration» for i det hele tatt å klare å starte applikasjonen. Følgende lenke inneholder guide til hvordan få til dette:

https://learn.microsoft.com/en-us/dotnet/maui/android/emulator/hardware-acceleration

For prosjektmedlemmene hjalp det å installere følgende pakke under Tools i Android SDK Manager (vist med rødt i figur 26). Det er ikke sikkert dette er riktig Accelerator for ditt system, så les nøye gjennom guide som ble gitt ved lenke.



Figur 26. Tillegg under Tools > Extra, i Android SDK Manager.

2. **Ved oppstart av eksisterende emulator, viser emulator feil klokkeslett og krasjer eller vil ikke kjøre applikasjon**. I dette tilfellet er det nødvendig å slette emulatoren og opprette en ny emulator. Prosjektgruppen har hatt erfaring med at ny emulator må opprettes for hver gang Visual Studio lukkes og startes på nytt. Dette tar en del tid, men kan fungere dersom det er kritisk å bruke emulator.

7.6 Generell feiloppretting ved feilet kjøring

Ved flere tilfeller kan applikasjonen feile ved *debugging*, på grunn av forskjellige prosjekt-filer som gir *error*. Dette kan være filer som bare trengs å bygges på nytt.

Build > Clean Solution, deretter, Build > Rebuild Solution, og deretter kjøring av løsning kan løse problemer med følgende filer:

- Plattform-spesifikke prosjektfiler som for eksempel *AndroidManifest*
- *Resx*-filer, dette er filer for oversetting.
- *MauiProgram.cs*, men sjekk og at *UseMauiCommunityToolkit()* er en del av *MauiApp.CreateBuilder*
- Ved feilmelding "*Property xxx does not exist in the current context*" kan samme fremgangsmåte hjelpe. Denne feilen kommer som regel av *properties* som skal autogenereres vha. *CommunityToolkit.Mvvm*.

7.7 iOS og Mac

Prosjektgruppen har ikke lyktes å kjøre applikasjonen på iOS eller Mac. Det ble heller ikke gjort veldig mange forsøk, men det forventes at dette ikke skal være veldig vanskelig så lenge man er nokså kjent med Visual Studio 2022 for Mac. Det er viktig at operativsystemet på Mac ikke er for gammelt, men det er usikkert hvilken versjon som er minstekrav. Det refereres til følgende dokumentasjon dersom det skal gjøres et forsøk på å teste og utvikle applikasjonen på iOS og Mac:
https://learn.microsoft.com/en-us/dotnet/maui/ios/cli
https://learn.microsoft.com/en-us/dotnet/maui/mac-catalyst/cli

Det er ikke gjort endringer i prosjektfiler som skal tilsi at kjøring på Mac og iOS er umulig. .NET MAUI støtter dette «out of the box», men som det kommer frem av dokumentasjonen kreves det litt mer for å få teknologien til å fungere på Apple sine operativsystemer. Github-*repository* må selvfølgelig *klones* eller *Forke*s-og-*klones* til Mac-datamaskin før man kan gjennomføre stegene i dokumentasjonen.

8 VIDERE ARBEID

Dette kapittelet gir en detaljert guide til hvordan videre arbeid med applikasjonen bør gjennomføres. Det presenteres hvordan ulike deler av applikasjonen som er viktig for videre arbeid er organisert, og hvilke feil og mangler som kan/bør rettes opp i. Nøyaktig plan og målsetning for videre arbeid vil selvfølgelig bestemmes av prosjekteier og veileder dersom det skal gjennomføres i forbindelse med bachelorprosjekt.

8.1 Converters

For å omgjøre modell-data til den dataen som ønskes å vises frem brukes Converters. Slike konverteringer gjøres av egne Converters-klasser som ligger i *Converters.cs*-filen, under mappen *ViewModels*. Det er spesielt en *Converters* som må merkes; *WDToTextConverter* som brukes i *TableCell.xaml*, vist med henholdsvis figur 27 og 28. Denne brukes for å gjøre vindretning om til en string, som for eksempel «sørøst». Problemet er at *Converteren* vil skape stor *overhead* dersom applikasjonen utvides til flere språk. Det anbefales, der det er mulig, å heller bruke filer for oversetting for å oppnå dette (men da blir det nødvendigvis svært mange tekststrenger for de ulike himmelretningene i oversettingsfilene).

```
/// <summary>
/// Class converts wind direction to text string.
/// By default returns english string i.e "Souteast", but if language setting is Norwegian (nb-NO)
/// returns norwegian string i.e "Sørøst".
/// If device is Phone string is truncated to abbreviation of wind direction, i.e "SE"
/// </summary>
public class WDToTextConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        if (value is double windDirection)
        {
            bool isPhone = DeviceInfo.Idiom == DeviceIdiom.Phone;
            bool isNorwegian = CultureInfo.CurrentCulture.Name == "nb-NO";
            isPhone = false;

            string cardinalDirection = windDirection switch
            {
                <= 22.5 or >= 337.5 => isNorwegian ? (isPhone ? "N" : "Nord") : (isPhone ? "N" : "North"),
                < 67.5 => isNorwegian ? (isPhone ? "NØ" : "Nordøst") : (isPhone ? "NE" : "Northeast"),
                < 112.5 => isNorwegian ? (isPhone ? "Ø" : "Øst") : (isPhone ? "E" : "East"),
                < 157.5 => isNorwegian ? (isPhone ? "SØ" : "Sørøst") : (isPhone ? "SE" : "Southeast"),
                < 202.5 => isNorwegian ? (isPhone ? "S" : "Sør") : (isPhone ? "S" : "South"),
                < 247.5 => isNorwegian ? (isPhone ? "SV" : "Sørvest") : (isPhone ? "SW" : "Southwest"),
                < 292.5 => isNorwegian ? (isPhone ? "V" : "Vest") : (isPhone ? "W" : "West"),
                < 337.5 => isNorwegian ? (isPhone ? "NV" : "Nordvest") : (isPhone ? "NW" : "Northwest"),
                _ => "Unknown",
            };

            return cardinalDirection;
        }
        else
        {
            return "Unknown";
        }
    }
```

Figur 27. Converter for vindretning til tekststreng.

```
<Label Grid.Row="2" Grid.Column="4" Text="{Binding WindDirection[0], Converter={StaticResource WDtoTextConverterKey}}" />
<Label Grid.Row="3" Grid.Column="4" Text="{Binding WindDirection[1], Converter={StaticResource WDtoTextConverterKey}}" />
<Label Grid.Row="4" Grid.Column="4" Text="{Binding WindDirection[2], Converter={StaticResource WDtoTextConverterKey}}" />
<Label Grid.Row="5" Grid.Column="4" Text="{Binding WindDirection[3], Converter={StaticResource WDtoTextConverterKey}}" />
```

Figur 28. Bruk av Converter i TableCell.

8.2 Filer for oversetting

Filer for oversetting ligger under Resources > Strings. Per nå inneholder applikasjonen oversetting for norsk, og *default*-filen er på engelsk. Hver eneste tekststreng i applikasjonen skal være i disse filene. Følgende ressurs ble brukt som guide for å sette opp oversetting, eller egentlig lokalisering, av applikasjonen:

https://cedricgabrang.medium.com/localize-your-net-maui-application-6722fb21e60d

Oversetting brukes i applikasjonen for å angi riktig dato-format avhengig av språk, noe som sannsynligvis vil være aktuelt med tanke på videre arbeid. Dato, tid og andre verdier kan kreve ulik fremstilling. Figur 29 viser ressurs for oversetting i *AppResources.nb-NO.resx* (norsk) og *AppResources.resx* (*default*-engelsk) for dato-format, mens figur 30 viser bruk av format i *BarChartCell.xaml*.

DateFormat_Long	{}{0:dd.MM.yyyy}
DateFormat_Short	{}{0:dd.MM}
DateFormat_Long	{}{0:dd/MM/yyyy}
DateFormat Short	{}{0:dd/MM}

Figur 29. Ressurs for oversetting av datoformat.

```
<Label Text="{Binding Path=Days[3], Source={RelativeSource AncestorType={x:Type local:MyLocationsViewModel}}, StringFormat={x:Static resx:AppResources.DateFormat_Short}}"
```

Figur 30. Bruk av ressurs for oversetting.

Generelt brukes oversettingene mest i *Text*-egenskapen til *labels*, for å oversette enkle strenger. Det er ventet at begreper for spesielt brannrisiko vil endres etter hvert som applikasjonen blir testet videre, og det er da i *AppResources*-filene at de korrekte oversettingene må legges til.

8.3 System for farger og stiler

Under Resources > Styles ligger *Colors.xaml* og *Styles.xaml*. I *Colors* defineres farger med nøkkel for navn, slik at de kan brukes senere (figur 31). I *Styles* defineres stiler som påvirker egenskapene til ulike kontroller i applikasjonen (figur 32). De kan defineres eksplisitt, da brukes de med en nøkkel-verdi, eller de kan defineres implisitt, da gis de ikke en nøkkel-verdi men påvirker alle kontroller av den gitte typen i hele applikasjonen (så lenge de aktuelle kontrollene ikke bruker en eksplisitt stil).

```
<Color x:Key="Primary">■#f4544a</Color>
<Color x:Key="Secondary">□#454c5a</Color>
<Color x:Key="Tertiary">■#ff634d</Color>
<Color x:Key="White">■White</Color>
<Color x:Key="Black">■Black</Color>
```

Figur 31. Definering av farger.

```
<Style x:Key="ToolbarGridStyle" TargetType="Grid">
    <Setter Property="BackgroundColor" Value="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Gray950}}"/>
    <Setter Property="HeightRequest" Value="60"/>
</Style>
```

Figur 32. Definering av Style.

I *Styles* kan man og definere ulike farger for stilen ut ifra systeminnstillinger for *Light* eller *Dark-mode*. Dette er **IKKE** implementert på en gjennomført måte, og ble ikke prioritert i applikasjonen fordi det ikke var et krav. Derfor vil kjøring av applikasjonen i *Dark-mode* gi en del rar oppførsel med farger, spesielt i *Shell*-komponenter. Det er og noen stiler som er definert på sidene de brukes på. En forbedring er å flytte disse til *Styles.xaml*, og abstrahere slike stiler til et par samlede stiler som gjelder for samme komponent på alle sidene. For eksempel at alle overskrifter bruker samme stil i større grad.

8.4 Tilpasninger til plattform

Tilpasninger er ikke gjort til hvert operativsystem, men de er gjort for å skille mellom Mobil og Skrivebord. Figur 33 viser hvordan *Grid* i *BarChartCell* inneholder tilpasninger. Dette fungerer, men roter til lesbarheten av kildekoden, og vanskeliggjør vedlikehold.

```
<!--Body bar chart content start -->
<Grid Grid.Column="0" Grid.Row="1" ColumnSpacing="{OnIdiom Phone='20', Default='100'}" Margin="{OnIdiom Phone='0,0,0,0',Default='200,0,200,0'}"
      ColumnDefinitions="*,*,*,*" RowDefinitions="*,*,*,*,*,10,20">
```

Figur 33. Tilpasninger til plattform med bruk av OnIdiom.

8.5 Graf (GraphCell)

For å tegne graf med verdier for TTF og vind ble det brukt et *GraphicsView* som er .NET MAUI sitt *view* for tegning. I *graphicsview* legges et *canvas*, som defineres som en C#-klasse som implementerer *IDrawable*-interface. I prosjektet heter denne klassen *GraphCanvas* og ligger i *Cells*-mappen. Figur 34 viser forenklet struktur for *Draw*-metoden som utfører tegningen.

```
For hver vind/ttf-verdi
  Tegn datapunkt
  Tegn linje til neste datapunkt
  Tegn label med verdi for datapunkt

  Hvis siste datapunkt
    Tegn datapunkt
    Tegn label med verdi for datapunkt
```

Figur 34. Struktur i tegne-metode for graf.

Hver tegne-operasjon er laget som egne metoder nederst i klassen. Ved tegning av *label* gjøres det beregninger som forskyver *label* dersom det kolliderer med et annet datapunkt. Disse beregningene gjøres med **hardkodede** verdier. Andre ting som er verdt å nevne er at det gjøres noen forskjeller mellom desktop og mobil, pga. rar oppførsel med størrelsene i *graphicsview* på mobil. Det burde gjøres et forsøk på å generalisere hele klassen til å kun bruke skjermstørrelse, men det er ikke sikkert dette er mulig. Generelt for *GraphCanva*s er at det er en ganske uorganisert klasse, mye på grunn av mengden tegneoperasjoner som må gjennomføres. Det burde derfor undersøkes om tredjeparts-bibliotek kan brukes til graf-tegning. I forbindelse med rapport ble det avdekket at disse kan være svært dyre, så det må undersøkes nøye hvilke alternativer som finnes.

8.6 Forslag til forbedringer, fra test med prosjekteier

I forbindelse med evalueringen av applikasjonen ble det gjennomført brukertest med stipendiat Ruben Dobler Strand (fungerende prosjekteier) og stipendiat Tord Hettervik Frøland (bakgrunn innen programutvikling). I forbindelse med intervju (spørreskjema) i etterkant ble det gitt forslag til forbedringer. Noen ble utført, mens de det ikke var tid/mulighet til blir presentert her:

På mobil:

- Innføre akseverdier for graf hvis det er plass og ser bra ut (mest sannsynlig er det for liten plass)
- Navigering: Når brukeren trykker på «My Locations» («Mine Lokasjoner») i tab-baren nederst, burde brukeren bli tatt tilbake til hovedside for lokasjoner, uavhengig om brukeren befinner seg på side for «Add Location».
-

På skrivebord:

- Innføre akseverdier for graf (ttf og vind altså)
- Utvide graf (og muligens tabell og stolpediagram) til å inneholde flere dager i forveien. Dette vil kreve mye jobb i brukergrensesnitt (per nå er visualisering for 4 dager «hardkodet» i både stolpediagram og tabell).
- Vurdere om stolpediagram fungerer godt nok på skrivebord. Kanskje bør stolpene tettere sammen, og akseverdier innføres. Det er forventet at videre brukertesting vil avdekke dette.

På begge plattformer:

- Tekst på hjelp-side må oppdateres kontinuerlig etter nye krav fra prosjekteier (det vil sannsynligvis avdekkes nye behov gjennom videre brukertesting).

8.7 Andre områder for forbedring

Noen av områdene for forbedring er allerede nevnt. Følgende er en oppramsing av noen områder for forbedring, uten bestemt rekkefølge.

a) Fullstendig implementasjon av stiler, for å kunne benytte Dark- og Lightmode på korrekt vis.

b) Generalisering av dataframvisning i *BarChartCell* og *TableCell* slik at data kan presenteres uavhengig av hvor mange dager som inkluderes. Da må det brukes

Views for oppramsing (*CollectionView*, *TableView* eller andre) i kombinasjon med *DataTemplates*.

c) Tilpasning av kontroller for navigering på skrivebord, for å gi en følelse som ligner mer tradisjonelle Desktop-applikasjoner (piler, overskrifter etc.).

d) Tilpasning av *WeatherClient* og *RestServiceClient* for å fasilitere for tilkobling til skytjeneste (Fog Computing og Proxy Server).

e) Generalisering av spesielt skriftstørrelse og farger i sentrale Stiler for å simplifisere vedlikehold og endringer. Verdier for *Margin* og *Spacing* etc. kunne og hatt godt av å være sentralisert, for å skape konsistente mellomrom og innrykk overalt.

f) Generalisering av brukergrensesnitt for å unngå altfor mange forskjeller mellom skrivebord og mobil. Heller ta utgangspunkt i skjermstørrelse for eksempel.

8.7.1 Overgang til skytjeneste

Bachelorprosjektet rakk ikke å gjennomføre migrering av brannrisiko-beregning-og-lagring til skytjeneste. Det var snakk om å benytte seg av en *Proxy Server* (mellomtjener) i skyen for lagring av beregnede brannrisiko-data, og *Fog Computing* (tåkedatabehandling) for å gjennomføre beregningene. Dette er svært viktig for å begrense mengden datatrafikk som genereres til MET sine API-er. Dette må implementeres før applikasjonen kan tilbys offentligheten, altså hvem som helst.

Mellomtjeneren må ta over alt ansvar for direkte forespørsler til MET sine API-er. Hvor beregningene av brannrisiko skal gjennomføres er opp til valgt strategi, men i *Fog Computing*-mønsteret kan det tenkes at de gjøres på kant-enheter (mobil/skrivebord som kjører applikasjonen). Deretter vil beregninger sendes til mellomtjener og være tilgjengelig for nye kant-enheter som ønsker brannrisiko for samme lokasjon. Dette vil sannsynligvis beskrives i større detalj ved neste prosjekt.

Endringer må nødvendigvis gjøres i *RestServiceClient*-implementasjonen. Sannsynligvis vil det kun kreves små endringer i *WeatherClient*-implementasjonen, fordi det er såpass lav kobling mellom disse klient-klassene. Det er uansett ikke forventet at brukergrensesnittet i form av *View* og *ViewModel* – klasser behøver noen forandringer.

FIGURER

9 REFERANSER

- MET (2022a) *WeatherAPI*. Tilgjengelig fra: https://api.met.no. (Hentet 12.05.23)
- MET (2022b) *What is FROST?* Tilgjengelig fra: https://frost.met.no/index.html (Hentet 12.05.23)
- MET (2022c) *Frost*. Tilgjengelig fra: https://frost.met.no/api.html (Hentet 12.05.23)
- MET (2022d) *API Concepts*. Tilgjengelig fra: https://frost.met.no/concepts2.html (Hentet 12.05.2023)
- MET (2022e) Authentication. Tilgjengelig fra: https://frost.met.no/authentication.html (Hentet 12.05.23)
- IBM (2023) *HTTP basic authentication*. Tilgjengelig fra: https://www.ibm.com/docs/en/cics-ts/5.4?topic=concepts-http-basic-authentication (Hentet 12.05.23)
- Log, T. (2019) *Modeling Indoor Relative Humidity and Wood Moisture Content as a Proxy for Wooden Home Fire Risk*. Sensors
- 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.
- SQLite (2023) *About SQLite*. Tilgjengelig fra: https://www.sqlite.org/about.html (Hentet: 15 mars 2023)