Operasjonalisering av en maskinlæringsmodell
for å detektere avvik og farer for grunnstøtinger i sanntid

# Vedlegg C - Systemdokumentasjon

Versjon 1.1

Revisjonshistorie

| Dato | Versjon | Beskrivelse | Forfatter |
|---|---|---|---|
| 19.04.2019 | 0.1 | Startet å fylle ut | Joakim Solheim |
| 29.04.2019 | 1.0 | 1. Versjon. Alle kapitler er fylt ut | Joakim Solheim, Hallvard Sælthun |
| 18.05.2019 | 1.1 | Revidering før innlevering. Oppdatert referanser | Brage Snarud, Joakim Solheim, Hallvard Sælthun |

# Innholdsfortegnelse

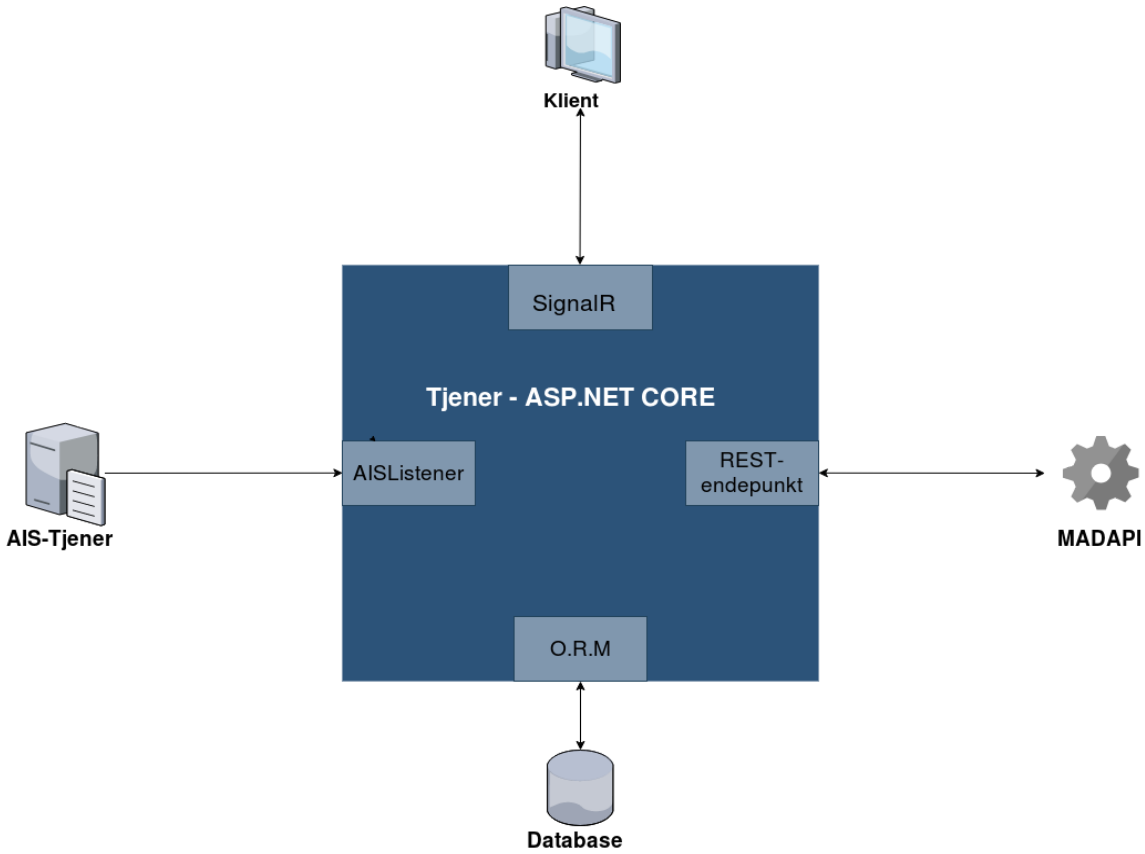
# 1. Introduksjon

Systemdokumentasjonen har som hensikt å beskrive produktet MADART Operational UI. Dokumentet inneholder modeller, diagrammer og beskrivelser av de essensielle arkitektoniske og strukturelle aspektene i produktet. I tillegg viser dette dokumentet hvordan du installerer, tester, kjører og avinstallerer produktet.

Dokumentet er utarbeidet etter mal fra Institutt for Datateknologi og Informasjonssystemer, NTNU.

Vi anbefaler å studere kapittel Hovedrapporten, kapittel 2, for å få et innblikk designmønstre og begreper som anvendes i dette dokumentet.

# 2. Arkitektur



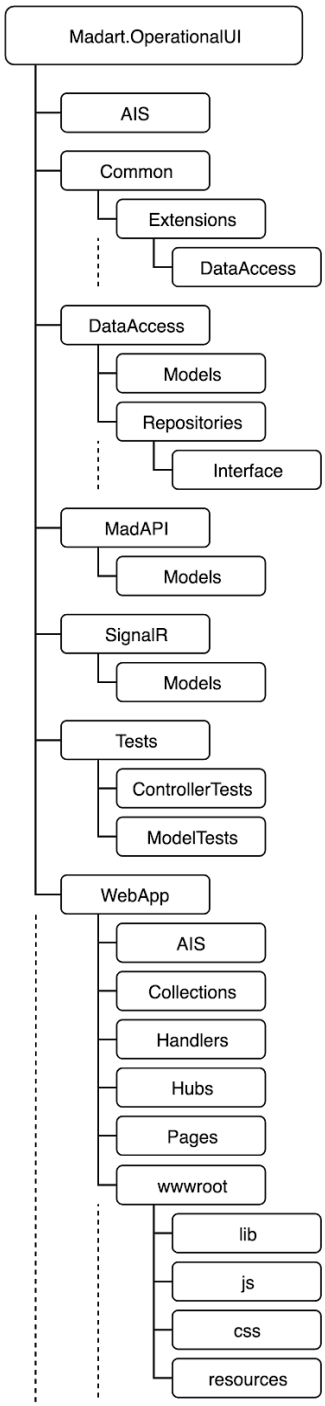

*Figur 2-1: Systemarkitektur for produktet, med hovedkomponentene systemet består av.*

Figuren over viser de fem hovedkomponentene til systemet:  Klient, Tjener, Database, AIS-tjener og API-et til MADART.

**Tjener**: Tjeneren er bygget på ASP.NET Core og er bindeleddet til alle de andre komponentene i arkitekturen.
**Klient**: Klienten sin oppgave er å fremstille dataene som blir sendt fra tjeneren, samt behandle hendelser og input fra brukeren. Klienten har en full dupleks kommunikasjon med tjeneren ved hjelp av SignalR.
**AIS-Tjener**: Tjeneren lytter på en AIS-tjener som kringkaster AIS-meldinger fra fartøyene utenfor Norges kyst. Det er altså en enveiskommunikasjon fra AIS-tjeneren til ASP.NET Core tjeneren.
**MADAPI**: Maskinlæringsmodellen MADART, som som tjeneren har toveiskommunikasjon med, har et API som tjeneren når gjennom REST-endepunkter. Tjeneren sender inn data hentet fra AIS-meldingene til fartøy, som MADART bruker til å klassifisere fartøy, beregne avvik og farer for grunnstøtinger.
**Database**: For å lagre data er ASP.NET Core-tjeneren koblet opp mot en PostgreSQL-database. Kommunikasjonen går igjennom ORM-klasser, som er bygget på Entity Framework Core. Entity Framework Core brukes til å oversette C#-objekter til og fra strukturen i databasen, og omvendt.

# 3. Prosjektstruktur

Dette kapittelet viser hvordan fil- og katalogstrukturen er organisert. Produktet består av sju delprosjekt, der alle av delprosjektene har ansvar for hver sine oppgaver i produktet.




*Figur 3-1: Prosjektstrukturen med de viktigste prosjekter og mapper.*

Produktet følger Separation of Concerns-prinsippet, hvor hvert delprosjekt har et spesifikt ansvarsområde. Dette holder orden og oversikt i systemet på en logisk måte, og gjør det enkelt å bytte ut komponenter og funksjonalitet.

## 3.1 AIS

Dette delprosjektet har som hensikt å håndtere AIS meldinger fra AIS-tjeneren. Delprosjektet inneholder to klasser: AISListener.cs og AISParser.cs, den førstnevnte lytter på en strøm med AIS meldinger fra Kystverket, og leser av disse meldingene. AISParser tolker meldingene slik at vi kan forstå informasjonen om fartøyet.

## 3.2 Common

Dette delprosjektet inneholder klasser som alle kan bruke, slik at vi unngår sirkulære avhengigheter. I vårt produkt så har vi VesselStatesExtensions.cs, som tilbyr metoder for å omformatere MessageType123-objekter til VesselState-objekter, VesselState-objekter til CurrentVesselState-objekter.

## 3.3 DataAccess

Dette delprosjektet har ansvaret for kommunikasjonen mellom serveren og databasen. Delprosjektet oppretter kommunikasjon til databasen og inneholder Repositories tilhørende de forskjellige entitene i databasen som sender ulike spørringer til databasen. Entity Framework Core blir brukt for å gi en sammenheng mellom entitetene i databasen og tilhørende C#-objekter.

## 3.4 MadAPI

Dette delprosjektet sitt ansvar er kommunikasjonen med API-et til MADART. API-et tilbyr flere REST-endepunkter som delprosjektet sender POST-forespørsler til. POST-forespørslene blir sendt og mottatt på JSON format, for å serialisere et C#-objekt til et JSON-objekt bruker vi biblioteket Newtonsoft.Json.

## 3.4 SignalR

Dette delprosjektet inneholder klasser for de objektene vi sender fra server til klient med SignalR. Disse klassene er CurrentVesselState og Warning, og de inneholder alt som er nyttig å sende til klienten.

## 3.5 Tests

Dette er test-prosjektet vårt, her blir det testet logiske og viktige metoder vi bruker. Det er og tester opp mot MADART slik at vi fort legger merke til situasjoner der koden ikke fungerer som forventet. Testene er enhetstester og integrasjonstester der vi sjekker om metodene returnerer antatt resultat.

## 3.5 WebApp

Dette er nettjenesten som knytter sammen alle delprosjektene, og står for logikken om hvordan vi håndterer hver AIS-melding. Denne logikken følger aktivitetsdiagrammet i vedlegg B, *Kravdokumentasjon*, kapittel 4. For et mer detaljert bilde av logikken se kapittel 5, Sekvensdiagram.

Delprosjektet har også ansvar for kommunikasjonen med klientene, og har logikken, og dataen til klienten.

I tabellen under finnes det en kort beskrivelse av de øverste mappene i WebApp:

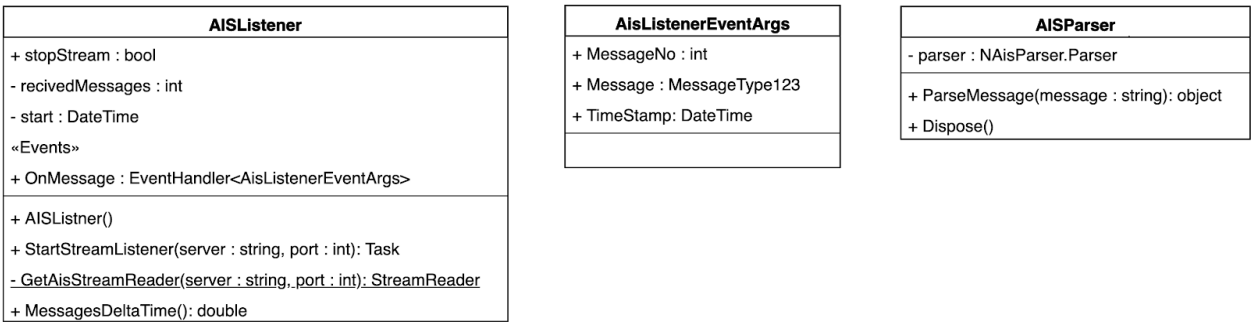
| Mappenavn | Beskrivelse |
|---|---|
| AIS | Klasser som håndterer mottak av AIS-meldinger. |
| Collections | Klasser som håndterer tilstandene til fartøy og advarsler |
| Handlers | Kontrollere som håndterer input fra klienten. |
| Hubs | Klasser som definerer kommunikasjonen med klienten gjennom SignalR |
| Pages | Inneholder prosjektets nettsider. |
| wwwroot | Inneholder logikken og dataen til klienten |

*Figur 3-2: Beskrivelse av viktige mapper i prosjektet: WebApp.*

# 4. Klassediagram

## 4.1 Madart.Operational.UI.AIS

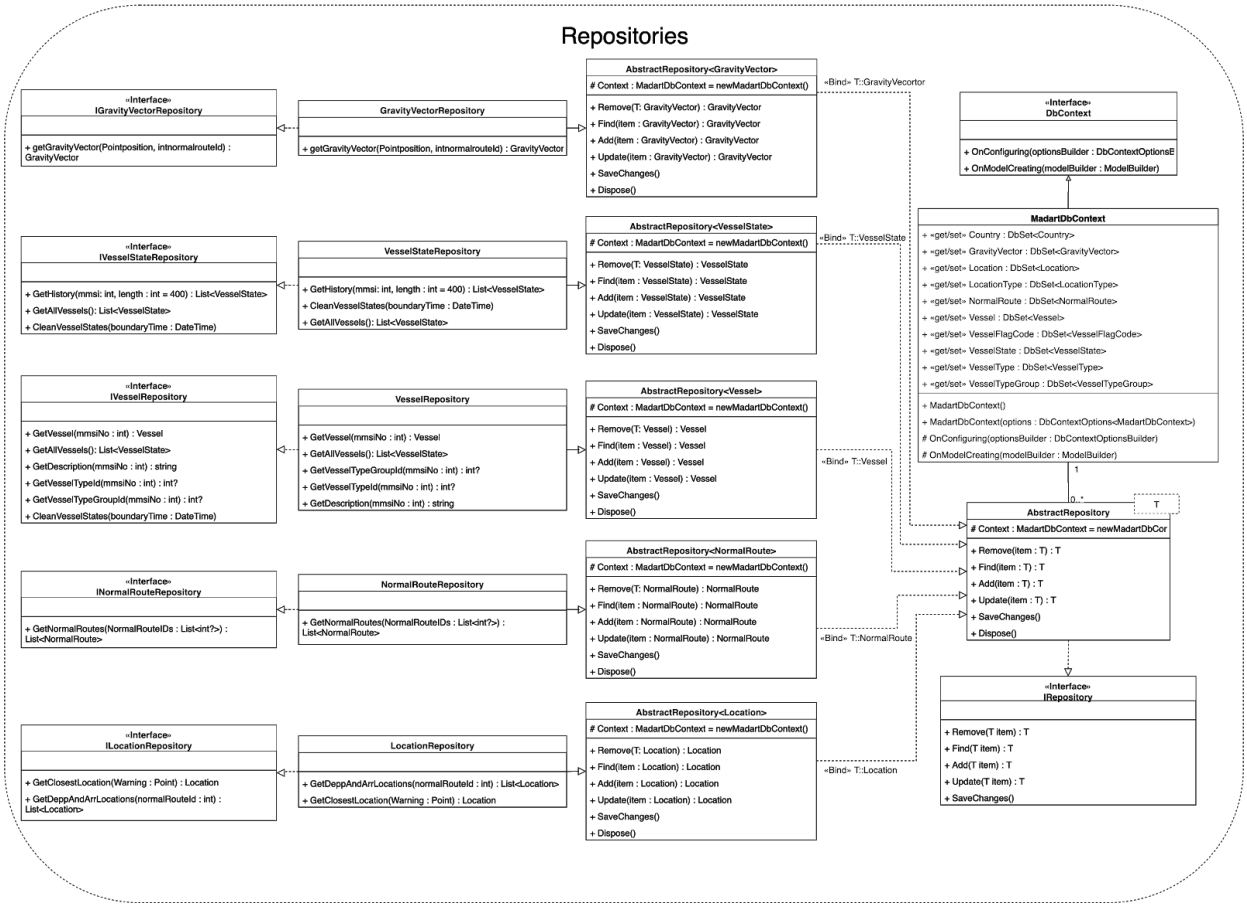
Figuren under viser klassediagram for AIS-prosjektet.




*Figur 4-1: Klassediagram for prosjektet: Madart.Operational.UI.AIS.*

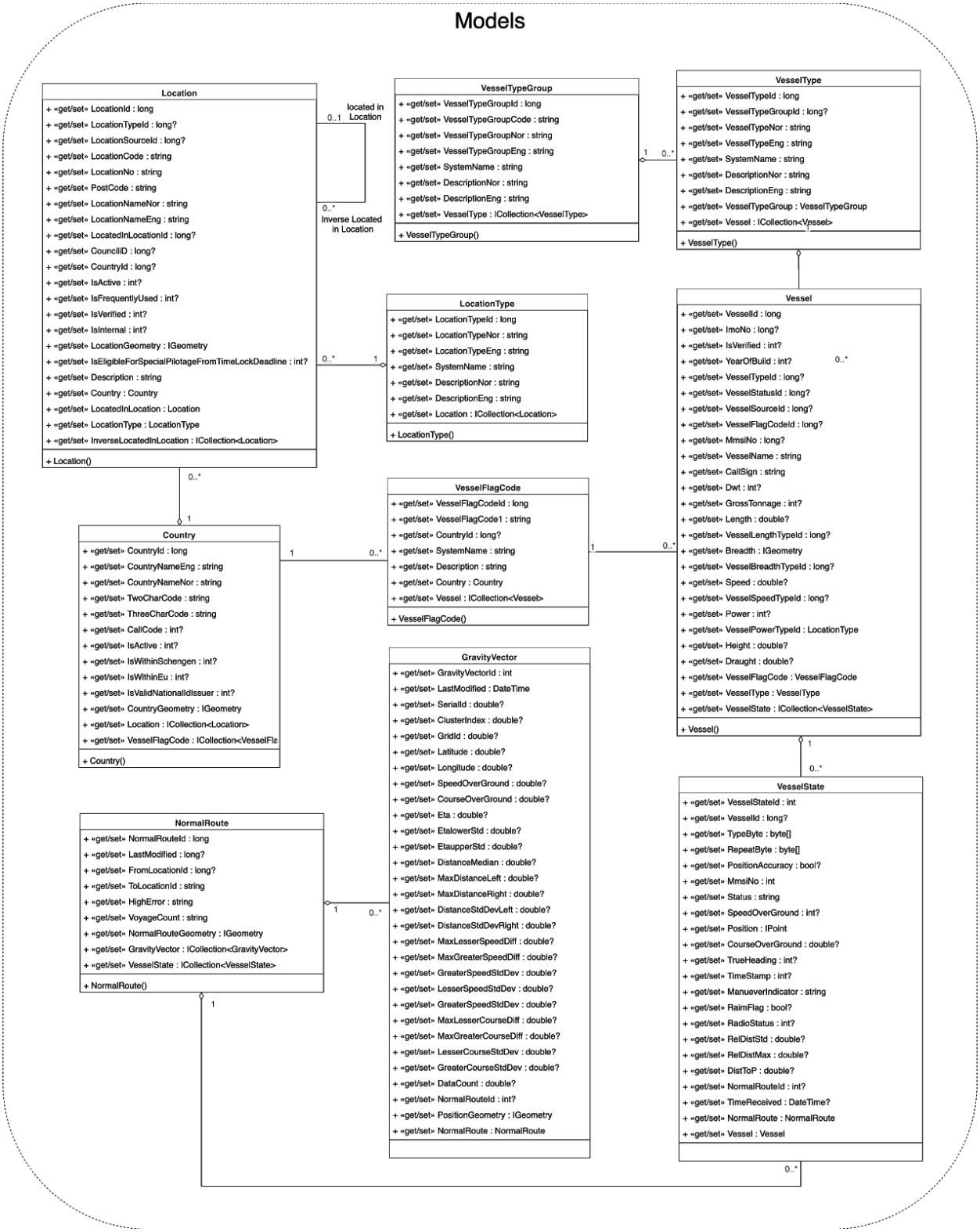
## 4.2 Madart.Operational.UI.DataAccess

Figuren under viser klassediagram for mappen “Repositories” i prosjektet DataAccess. For ordens skyld så er relasjonen mellom MadartDBContext og dens attributter ikke vist.




*Figur 4-2: Klassediagram for klasser i prosjektet Madart.Operational.UI.DataAccess, under Repositories.*

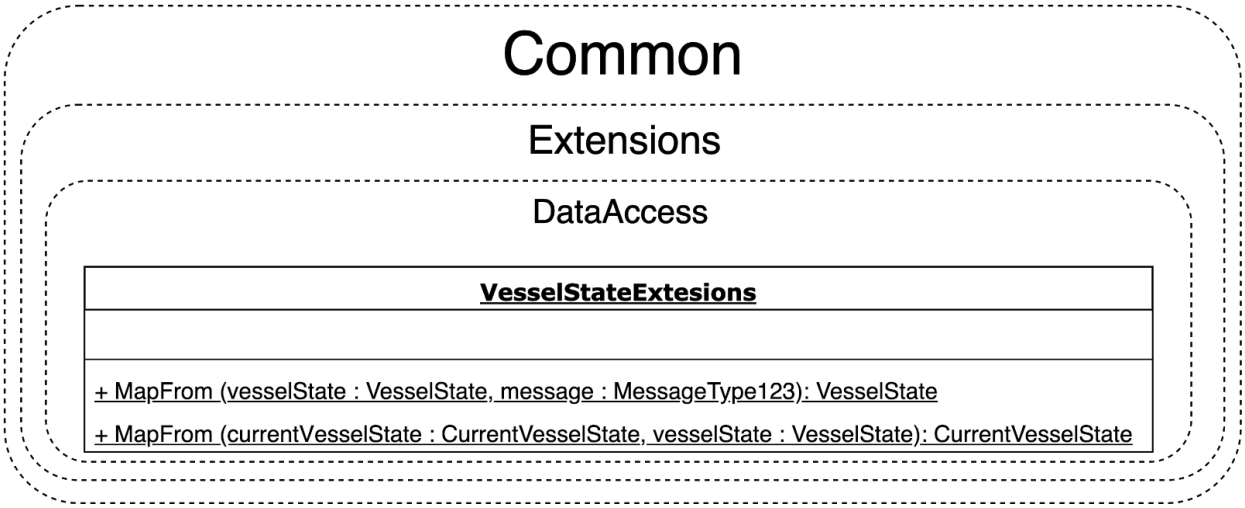
Figuren under viser klassediagram for mappen “Models” i prosjektet DataAccess. For ordens skyld så vises ikke relasjoner med klasser utenfor mappen.




*Figur 4-3: Klassediagram for klasser i prosjektet Madart.Operational.UI.DataAccess, under Models.*

## 4.3 Madart.Operational.UI.Common

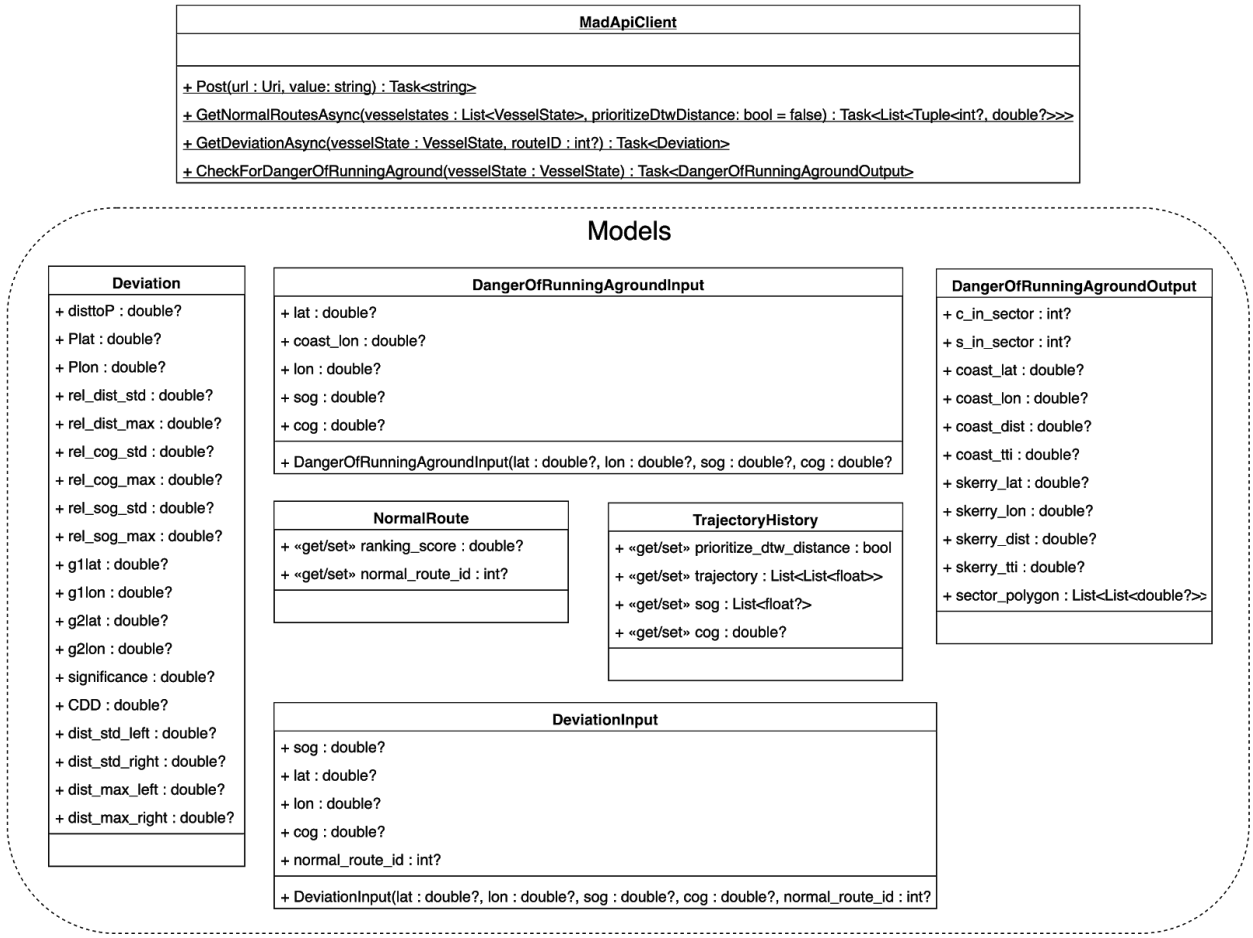
Figur under viser klassediagram for klasser i prosjektet Common.




*Figur 4-4: Klassediagram for klasser i prosjektet Madart.Operational.UI.Common.*

## 4.4 Madart.Operational.UI.MadAPI

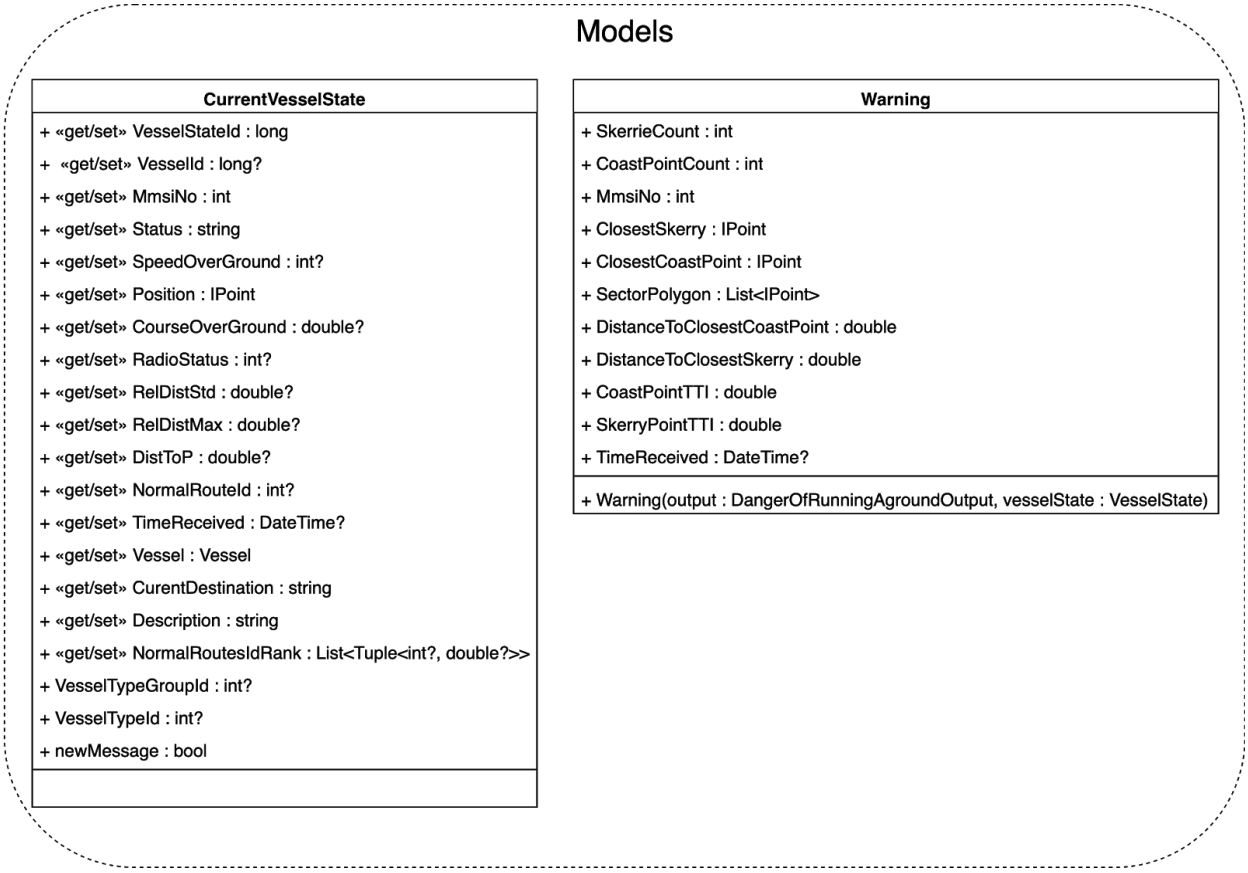
Figur under viser klassediagrammet for prosjektet MadAPI.




*Figur 4-5: Klassediagram for klasser i prosjektet Madart.Operational.UI.MadAPI.*

## 4.5 Madart.Operational.UI.SignalR

Figuren under viser klassediagrammet for SignalR. For ordens skyld så vises ikke relasjoner med klasser utenfor prosjektet.



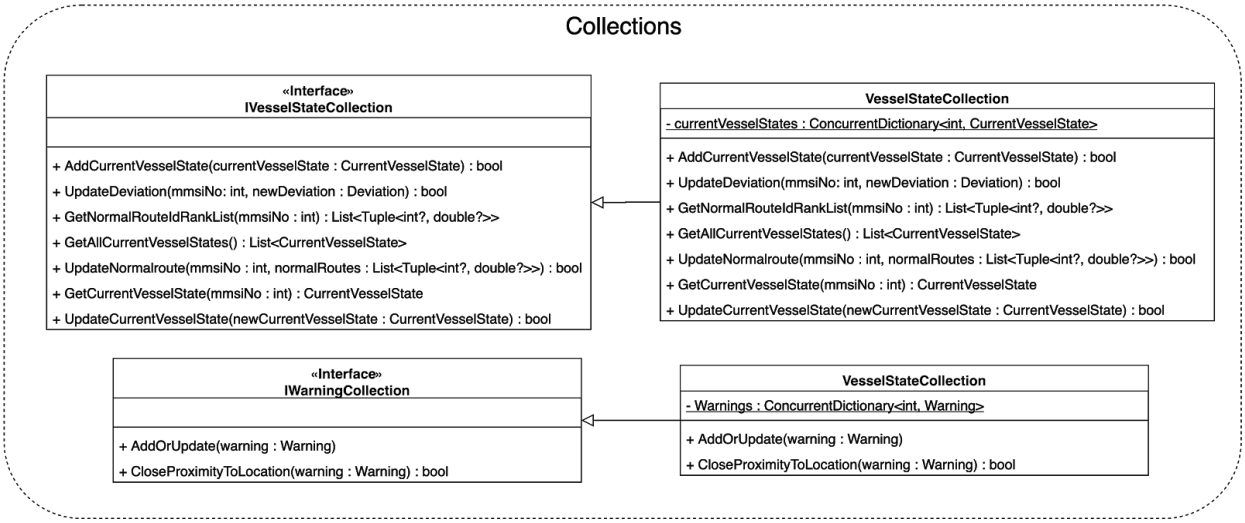

*Figur 4-6: Klassediagram for klasser i prosjektet Madart.Operational.UI.SignalR.*

## 4.6 Madart.Operational.UI.WebApp

Figuren under viser klassediagram for mappen “AIS” i prosjektet WebApp. For ordens skyld, så er ikke relasjonen med klasser utenfor mappen vist.

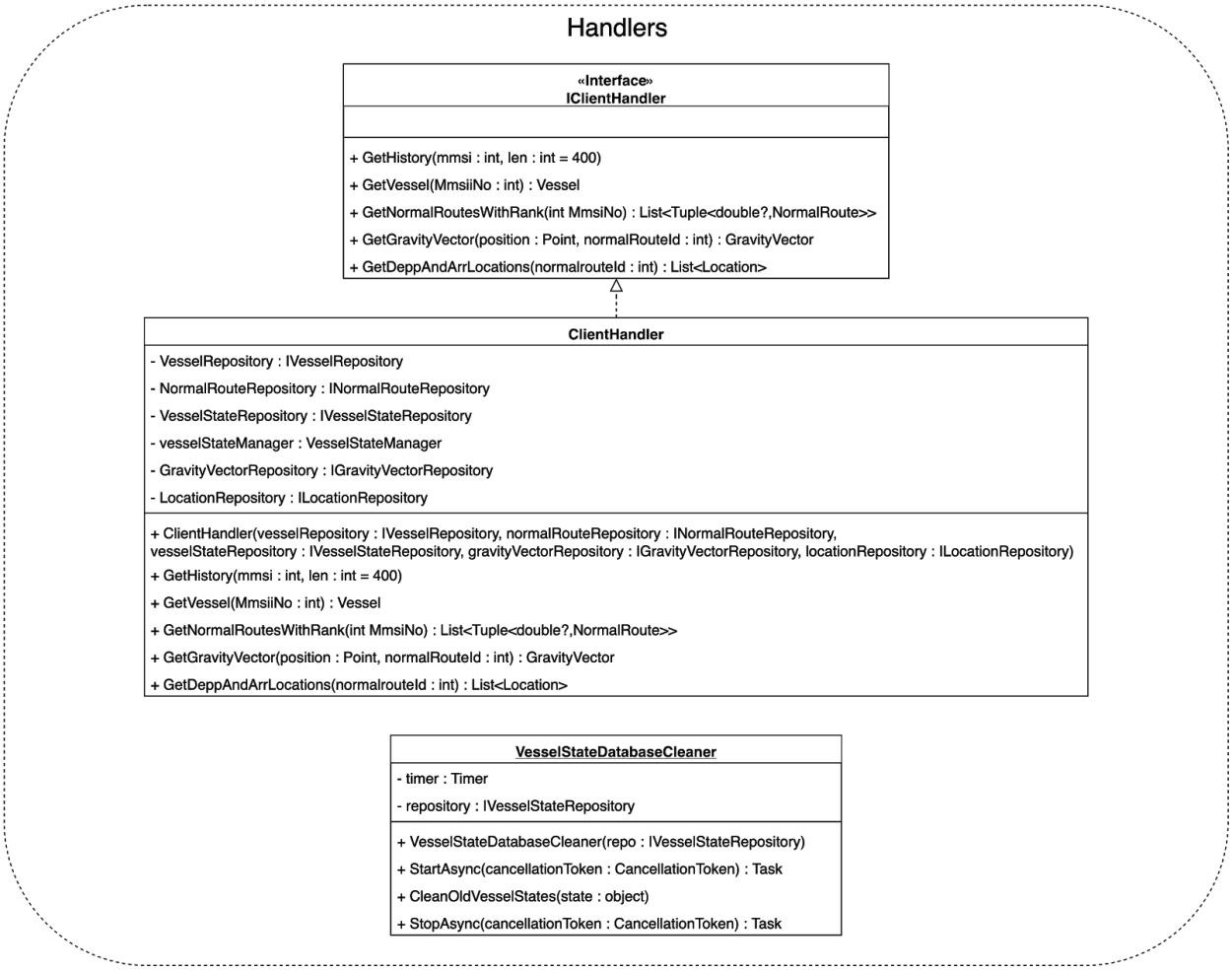

*Figur 4-7: Klassediagram for klasser i prosjektet Madart.Operational.UI.WebApp, under AIS.*

Figuren under viser klassediagram for mappen “Collections” i prosjektet WebApp. For ordens skyld, så er ikke relasjoner med klasser utenfor mappen vist.



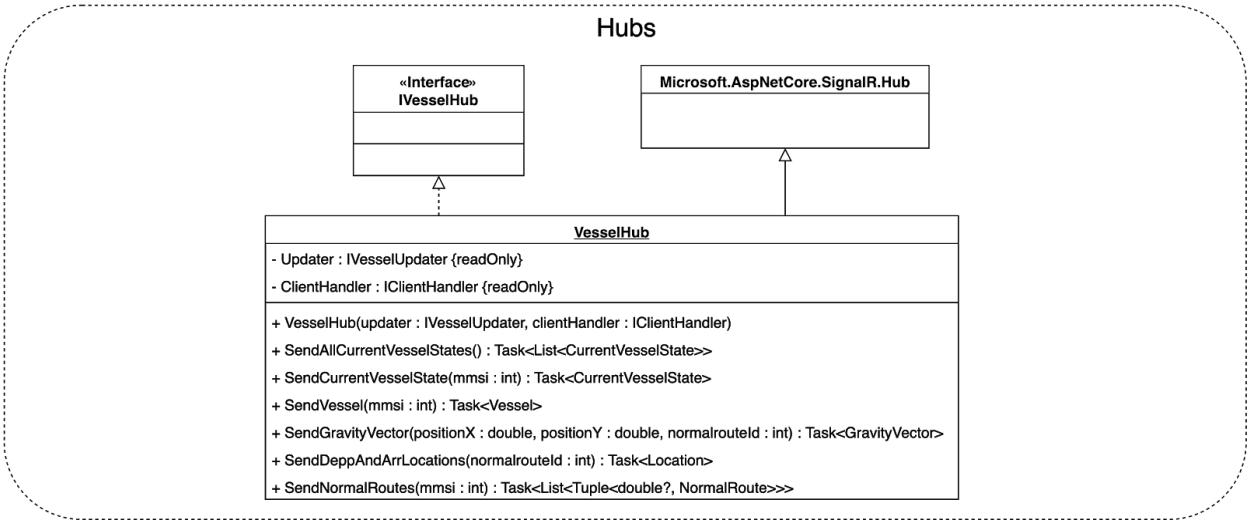

*Figur 4-8: Klassediagram for klasser i prosjektet Madart.Operational.UI.WebApp, under Collections.*

Figuren under viser klassediagram for mappen “Handlers” i prosjektet WebApp. For ordens skyld, så er ikke relasjoner med klasser utenfor mappen vist.



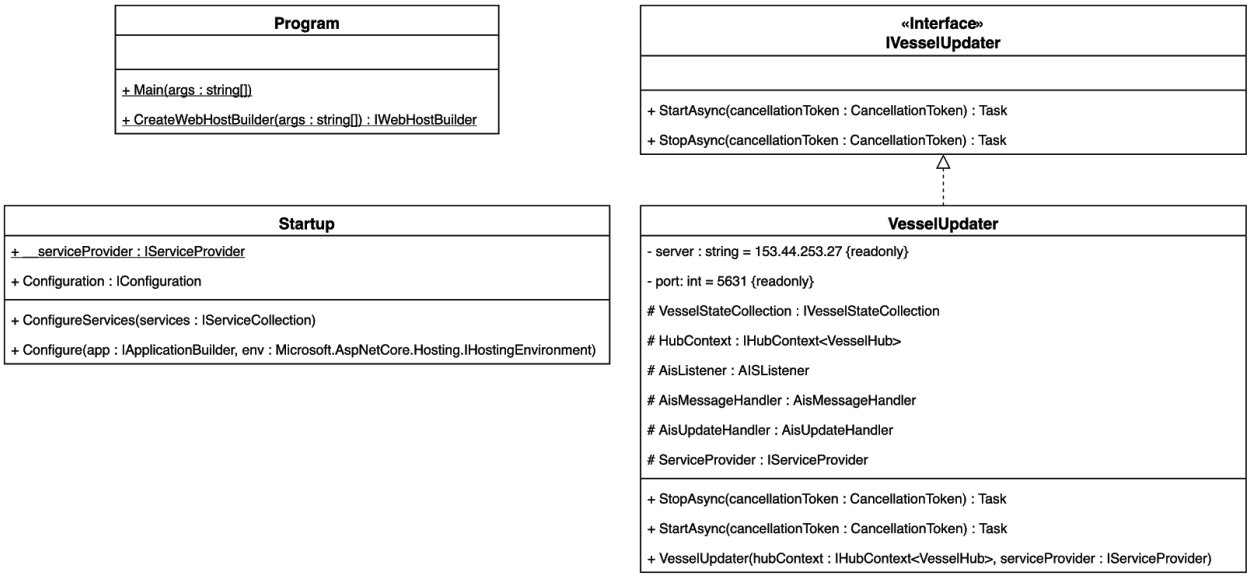

*Figur 4-9: Klassediagram for klasser i prosjektet Madart.Operational.UI.WebApp, under Handlers.*

Figuren under viser klassediagram for mappen “Hubs” i prosjektet WebApp. For ordens skyld, så er ikke relasjoner med klasser utenfor mappen vist.




*Figur 4-10: Klassediagram for klasser i prosjektet Madart.Operational.UI.WebApp, under Hubs.*

Figuren under viser klassediagram for resterende klasser i prosjektet WebApp. For ordens skyld, så er ikke relasjoner med klasser utenfor prosjektet eller på lavere nivåer vist.

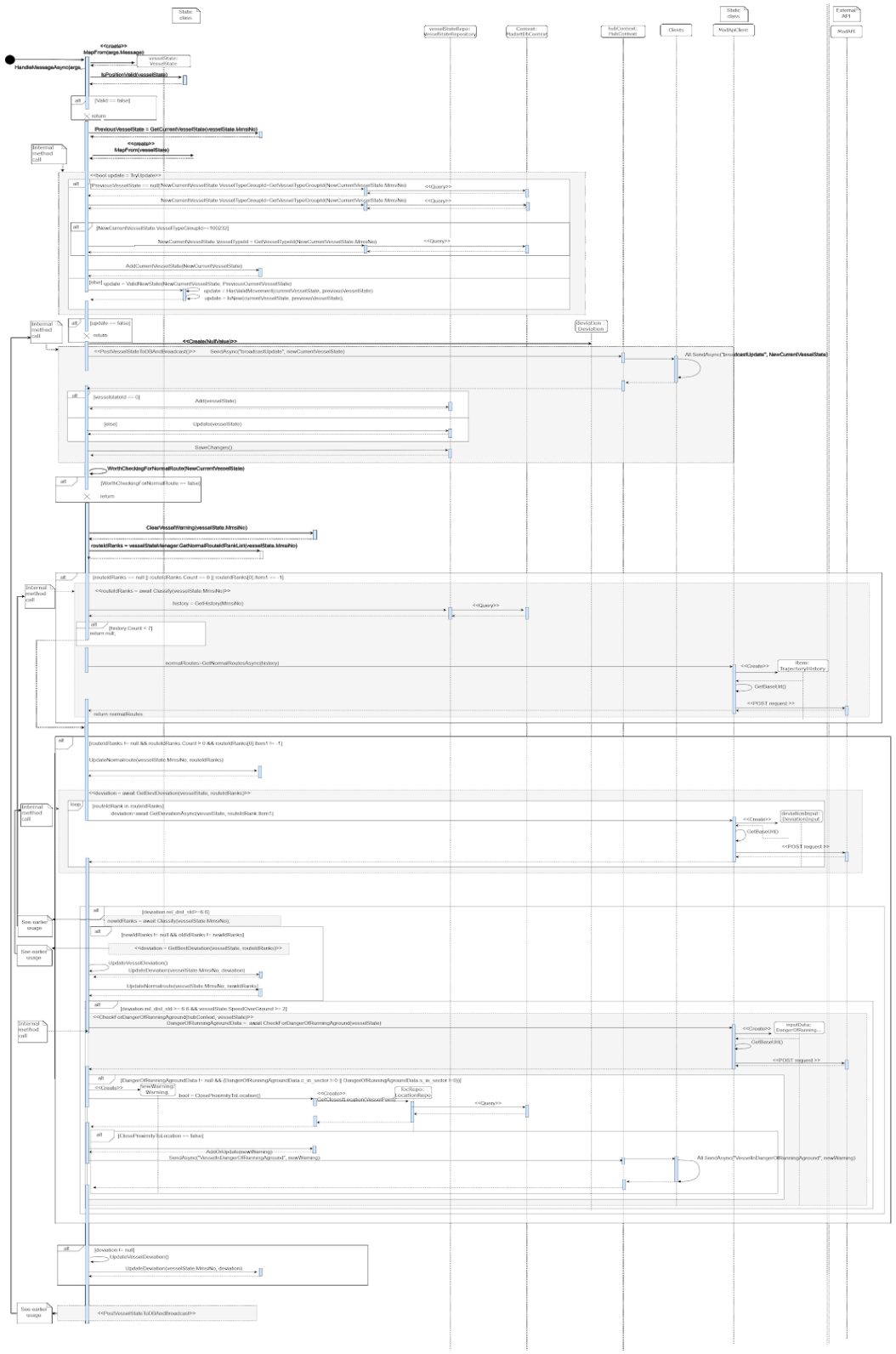



*Figur 4-11: Klassediagram klasser i prosjektet Madart.Operational.UI.WebApp.*

# 5. Sekvensdiagram

Dette kapittelet tar for seg i detalj hvordan AIS-meldinger håndteres av tjeneren. Det anbefales å sette seg inn i aktivitetsdiagrammet til samme prosess i forkant. Dette finnes i vedlegg B, *Kravdokumentasjon*, kapittel 4

Etter at meldingene er dekodet i AIS-prosjektet, håndteres de av tjeneren gjennom metoden HandleMessage, i klassen AISMessageHandler. Sekvensdiagrammet under, viser hvordan meldingen behandles. Merk at sekvensdiagrammet er så omfattende at det er nærmest uleselig i et tekstdokument. Det legges ved en egen pdf-fil i innleveringen, for å lettere kunne navigere og lese diagrammet. Det er laget visse enkle hjelpemetoder for å gjøre diagrammet mer leselig. Disse er markert med en grå boks i sekvensdiagrammet. Visse forenklinger er gjort for å gjøre diagrammet lesbart og nyttig for videre utvikling.

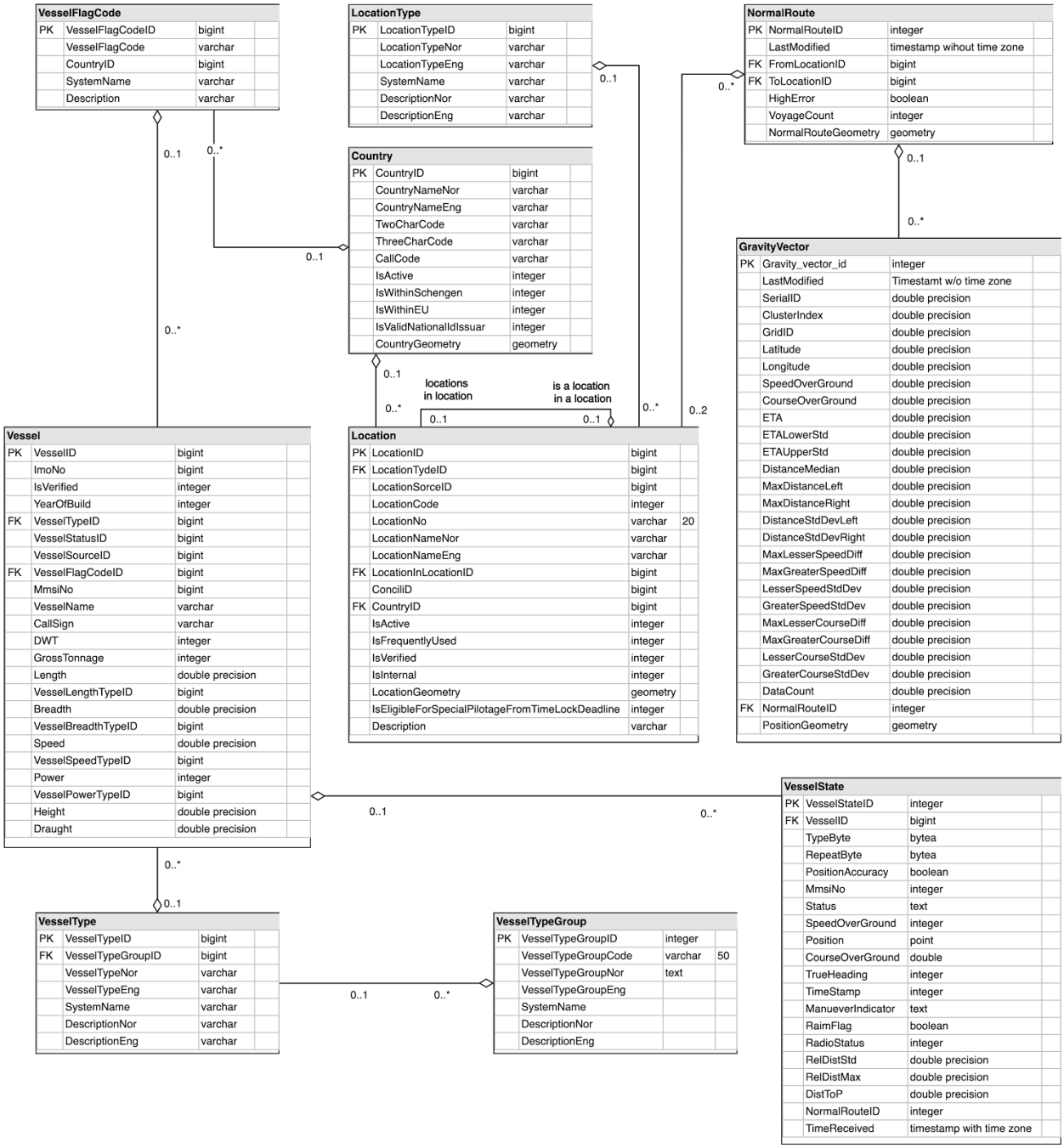


*Figur 5-1: Sekvensdiagram for metoden HandleMessage i klassen Madart.OperationalUI.WebApp.Ais.AisMessageHandler.*

Sekvensdiagrammet viser livsløpet til metoden HandleMessage. HandleMessage er en omfattende metode med flere alternative utfall og hendelsesforløp. HandleMessage er en Task, som betyr at metoden utføres som en tråd. Metoden tar inn et objekt av klassen AisMessageEventArgs som inneholder et objekt av klassen MessageType123. Dette objektet inneholder en posisjonsrapport til et fartøy, som metoden forholder seg til. Først vil metoden prøve å validere den aktuelle meldingen. Dette gjøres ved å sjekke at posisjonen er logisk, og dersom serveren har historikk på det aktuelle fartøyet vil metoden sjekke at bevegelsesomfanget er gyldig. Om meldingen er ugyldig forkastes den. Dersom meldingen er gyldig oppdaterer vi en intern klasse som holder orden på en datastruktur over CurrentVesselState-objekter (vesselStateCollection). CurrentVesselState-objekter har fartøysdata i forrige registrerte posisjon. I tillegg så oppdateres databasen, og fartøysdataen sendes til alle klienter.
Deretter vil metoden gå opp mot det eksterne API-et til MADART for å sjekke om det er betydelig avvik mellom fartøyet og tidligere klassifisering. Dersom avviket er stort på alle normalruter, eller fartøyet ikke allerede er klassifisert, prøver metoden å klassifisere seilaset på nytt ved å hente historikk fra databasen. Dersom klassifisering ikke ga bedre resultat på avvik, sjekker vi om fartøyet er i potensiell fare for grunnstøting. Ved fare for grunnstøting vil klientene varsles med et Warning-objekt.
Uavhengig av hvilke betingelser som slår ut med tanke på avvik og varsling, vil vi oppdatere vesselStateCollection, database og klienter med data om avvik og normalruter.

# 6. Databasemodell

Når produktet installeres, vil en database bli laget og fylt med data. Denne dataen er fra Kystverket, og brukes av tjeneren som oppslagsverk. I tillegg lagrer tjeneren AIS-meldinger og oppdaterer enkelte tabeller under kjøretiden.
Figuren under viser databasemodellen for produktet:




*Figur 6-1: Databasemodell av databasen til produktet.*

Tabellene LocationType, Location, Country, NormalRoute og GravityVector brukes som oppsalgsverk. Gjennom MADART blir fartøy klassifisert til normalruter. Med tabellen NormalRoute kan vi gjøre oppslag på de aktuelle normalrutene, tilhørende Gravity Vector-er i tabellen GravityVector, og se hva som er avgangs- og ankomsthavn gjennom tabellen Location. Tabellen Vessel brukes av programmet for gjøre oppslag om fartøyet. Gjennom VesselFlagCode-tabellen kan vi finne informasjon om fartøyets opphavsland. For å finne hva slags type fartøy det er, og hvilken gruppe fartøystyper fartøyet tilhører, gjør programmet oppslag i VesselType- og VesselTypeGroup-tabellene. Oppslag på fartøystype og hvilken gruppe fartøystype et fartøy tilhører, gjør det mulig å filtrere vekk, og ignorere fartøy som det ikke er ønskelig å klassifisere.

VesselState-tabellen brukes for å lagre AIS-meldinger av typen 1, 2 og 3. Disse AIS-meldingene brukes for å lage et bilde av hvor et fartøy har beveget seg, og klassifisere fartøyet.
AIS-meldinger av typen 5 fører til en oppdatering på det aktuelle fartøyet i Vessel-tabellen.

# 7. Kommunikasjon og grensesnitt

Tjeneren kommuniserer med mange komponenter. Dette kapittelet er ment til å gi en rask innføring i denne kommunikasjonen, hvilke definerte endepunkter som allerede er utviklet, og hvordan disse kan videreutvikles i vårt produkt.

## 7.1 SignalR

Tjeneren i MADART Operational UI tilbyr ulike tjenester til klienten. Kommunikasjonen mellom disse partene foregår ved bruk av ASP.NET-biblioteket SignalR. Klienten oppretter en forbindelse med en hub på tjeneren, og spør om data ved å kalle på en metode til forbindelsen, som returnerer et Promise. Tjeneren kan deretter sende data til en eller flere klienter gjennom en HubContext.

For å bruke SignalR, følger man Microsoft sin guide for ASP.NET Core SignalR JavaScript client. SignalR-pakken til prosjektet ligger i prosjektet MadartOperatinal.UI.WebApp under wwwroot/js/lib/aspnet/signalr. Vi kan lage og starte en forbindelse med hub-en VesselHub ved å skrive denne javascriptkoden:

```
connection = new signalR.HubConnectionBuilder().withUrl("/vesselHub").build();
```

For å kalle på metoder til hub-en fra klienten så kaller vi i på HubConnection sin metode: 'invoke'. Den tar to argumenter: navnet på hub-metoden, og argumentene som er definert i huben:

```
connection.invoke("SendVessel", mmsiNo).catch(err => console.log(err));
```

Denne metoden returnerer et JavaScript Promise. Promise-et løses med returverdien, dersom det eksisterer. Tabellen under viser følgende metoder som kan kalles på fra klienten mot VesselHub.

| Metodenavn | Argumenter | Beskrivelse |
|---|---|---|
| SendHistory | mmsi : int | Tjeneren sender historikken til et gitt fartøy. |
| SendAllCurrentVesselStates | | Tjeneren sender alle CurrentVesselState-objekter på serveren. |
| SendAllCurrentWarnings | | Tjeneren sender alle Warning-objects på serveren. |

| | | |
|---|---|---|
| SendCurrentVesselState | mmsi : int | Tjeneren sender CurrentVesselState-objektet til et gitt fartøy. |
| SendVessel | mmsi : int | Tjeneren sender fartøysinfomasjon for et gitt fartøy. |
| SendGravityVector | postitionX : double, postitionY : double, normalrouteId : int | Tjeneren sender GravityVector-objektet gitt posisjon og normalrute |
| SendDeppAndArrLocations | normalrouteId : int | Tjeneren sender Location-objektet til avgangshavnen, og ankomsthavnen. |
| SendNormalRoutes | mmsi : int | Tjeneren sender NormalRoute-objektene som er tilknyttet et fartøy. |

*Figur 7-1: Metoder som klienten kan kalle på mot VesselHub.*

For å lage nye endepunkter, må den bli deklarert som en metode i klassen VesselHub på tjeneren. Det er anbefalt å bruke en asynkron metode for å oppnå god ytelse. Eksempel:

```
public async Task<List<CurrentVesselState>> sendAllCurrentVesselStates()
{
  ...
}
```

Det er to måter å returnere verdien til metoden på. Den ene måten for å returnere verdien er ved å bruke det reserverte ordet '`return`', eller så kan tjeneren sende data til klienten gjennom å bruke namespace-et: Microsoft.AspNetCore.SignalR, og kalle på metoden `Clients.Caller.SendAsync`. Denne metoden tar to argumenter: navnet på klient-metoden, og argumenter som hub-en gir. Eksempel:

```
Clients.Caller.SendAsync("vesselHistory", history);
```

Det er også mulig for å tjeneren å sende ut data til alle klienter som er koblet til en hub på liknende måte, men i stedet for '`Caller`', så benytter man seg av '`All`'.

```
hubContext.Clients.All.SendAsync("broadcastUpdate", currentVesselState);
```

Følgende metoder kan hub-en VesselHub kalle på:

| Metodenavn | Argumenter | Beskrivelse |
|---|---|---|
| broadcastUpdate | currentVesselState : CurrentVesselState | Tjeneren sender et CurrentVesselState-objekt |
| VesselInDangerOfRunningAground | newWarning : Warning | Tjeneren sender et Warning-objekt |

*Figur 7-2: Metoder VesselVub kan kalle på.*

For å lage nye endepunkter til klienten, så bruker man HubConnection sin metode: 'on'.

```
connection.on("broadcastUpdate", function (data) { receiveVesselState(data); });
```

## 7.2 AISListener

Tjeneren lytter på en AIS-strøm gjennom "`AISListener`". "`AISListener`", sender data fra AIS-strømmen gjennom .NET sin Event håndtering. "`AISListener`" fungerer som event sender, og "`AISMessageHandler`" fungerer som event handler.

For å lytte på en AIS-strøm, så er det bare å oppgi ip-adressen og portnummeret til strømmetjeneren: "`AISListener.StartStreamListener(string server, int port)`", i "`VesselUpdater`", under metoden: "`VesselUpdater.StartAsync()`".

"`AISListener`", har to Event-er: "`OnMessage`", og "`OnUpdate`". "`AISListener`" sender en AIS-melding til sine abonnenter, ved å lage en "`AisMessageEventArgs`", som den sender ved å kalle på metoden "`OnMessage.Invoke()`", eller "`OnUpdate.Invoke()`".

"`AISMessageHandler`", abonnerer på "`AISListener.OnMessage`", og "`AISListener.OnUpdate`", under metoden til "`VesselUpdater`": "`StartAsync`":

```
AisListener.OnMessage += async (s, args) =>
      {
        ...
      };
```

Meldinger av typen "`MessageType123`" skal kalle på "`AISMessageHandler`", sin metode: "`HandleMessageAsync`", og "`MessageType5`", skal kalle på "`AISUpdateHandler`" sin metode: "`HandleUpdateAsync`".

## 7.3 MadAPI

ASP.NET Core-tjeneren kommuniserer med et API som er bygget på maskinlæringsmodellen, MADART. Kommunikasjonen skjer gjennom REST, hvor tjeneren til produktet er klienten, og MADART er serveren.

Tabellen under viser en liste av alle ressursene med oversikt over endepunktene, beskrivelse av dem og formatet på meldingen skal sendes og mottas på.

| Endepunkt | Beskrivelse | Format |
|---|---|---|
| api/get_routes | Tjenesten tar inn en liste av posisjonshistorikk og bruker maskinlæring til å finne normalruter til fartøyet. Det blir returnert normalruter dersom API-et finner noen som passer, eller “-1” om fartøyet ikke kan bli klassifisert til noen ruter. | application/json |
| api/get_deviations | Tjenesten tar inn informasjon om posisjonen til fartøyet samt normalruten fartøyet er klassifisert til. Det blir returnert data om hvordan fartøyet avviker fra normalen. | application/json |
| api/get_sector | Tjenesten tar inn informasjon om posisjonen til fartøyet og størrelse på sektor. Tjenesten sjekker om det er farlige skjær eller grunne i sektoren foran fartøyet. Det blir returnert antall treffpunkt for skjær og grunne innenfor sektoren, samt posisjonen og sekunder til grunnstøting for det nærmeste punktet. Dersom det ikke blir funnet noen farer innenfor sektoren blir bare antallet lik 0, samt posisjonen og tiden blir lik “null”. | application/json |

*Figur 7-3: REST-endepunktene til API-et, med beskrivelse og format.*

# 8. Sikkerhet

MADART Operational UI har sikkerhet i form av en kryptert forbindelse mellom klient og tjener. Sertifikatet til tjeneren er generert ved hjelp av Nuget-pakken: dotnet-dev-certs 2.2. Applikasjonen er beskyttet mot Cross-Site Scripting og SQL-Injections, applikasjonen følger Microsoft sine retningslinjer for hvordan å beskytte ASP.NET Core applikasjoner mot Cross-Site Scripting [1].

Applikasjonen er beskyttet mot SQL-injections, gjennom ORM-rammeverket Entity Framework Core, hvor vi ikke sender SQL-spørringer i råformat, men gjennom å gi parametere. Dette er i tråd med Microsoft sine retningslinjer for hvordan en skal unngå SQL-injections i EF Core [2]. Slik som dagens løsning er utformet er det ingen plasser en bruker kan skrive inn informasjon. Siden det ikke er noen input-felt på nettsiden, så kan ikke en klient sende kall med egendefinert input til tjeneren.

# 9. Installasjon og kjøring

Denne installasjonsguiden viser hvordan prosjektet installeres, kompileres og kjøres på en lokal maskin. Denne installasjonsguiden er ikke ment som en veiledning til hvordan prosjektet kan bli utrullet, men heller hvordan den kan installeres for videreutvikling.

## 9.1 Forutsetninger

En forutsetning for at prosjektet og testene skal kjøre som ment i denne installasjonsguiden, er at maskinen står inne på nettverket til Norconsult Informasjonssystemer, slik at den får kontakt med API-et.

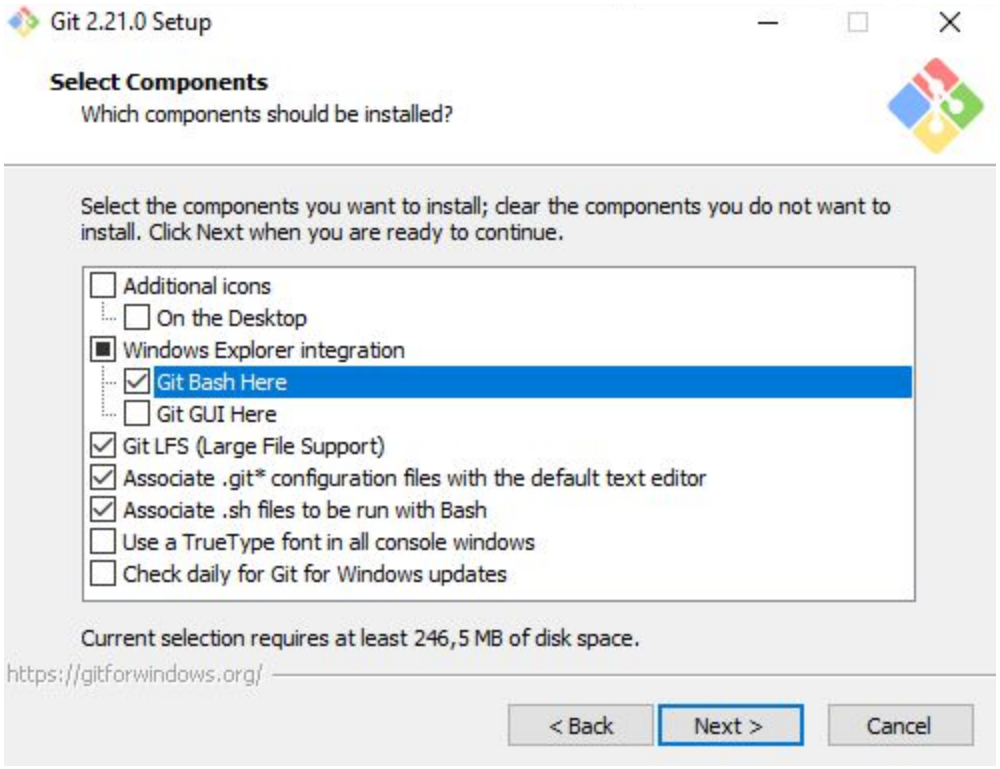
En annen forutsetning er tilgang til repo-et på github. Ta kontakt med Norconsult Informasjonssystemer, for å be om tilgang til git repo-et.

## 9.2 Windows 10

Programmer nødvendige for å kjøre prosjektet er listet i tabellen under.

| Program | Versjon |
|---|---|
| Git | 2.21.0 |
| PostgreSQL 10 | 10.7 |
| PostGIS 2.5 | 2.5.2 |
| .NET Core SDK 2.2 | 2.2.203 |

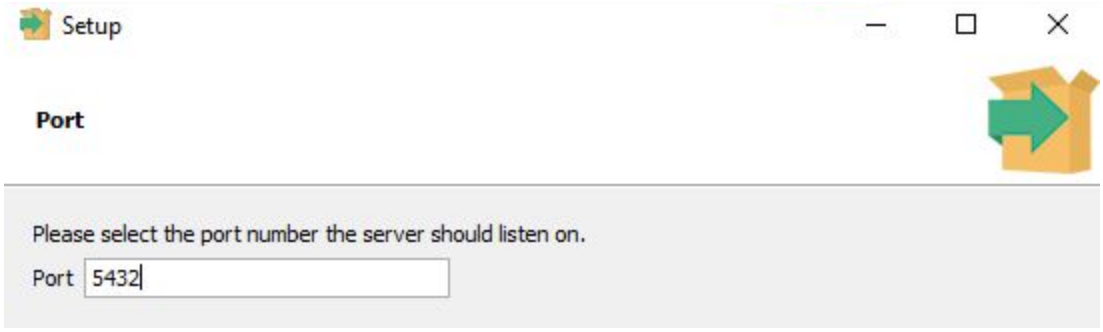
1. Installer git. Følg installasjonsveiledningen til Git: https://git-scm.com/download/win
    1. Huk av for å installere Git Bash:




Installer PostgreSQL. Følg installasjonsveiledningen til PostgreSQL Global Development Group: https://www.postgresql.org/download/windows/

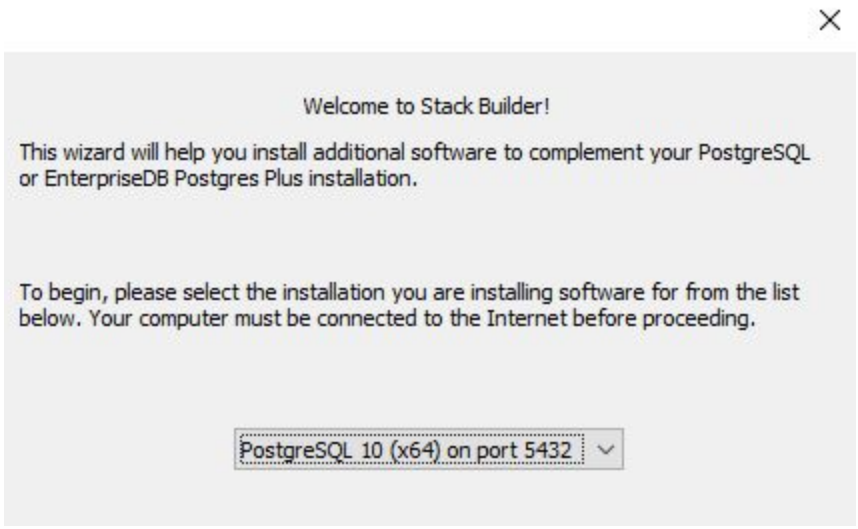
2. Huk av for å installere PostgreSQL Server, pgAdmin 4, StackBuilder, og Command Line Tools:

3. Sett serveren til å lytte på port 5432:

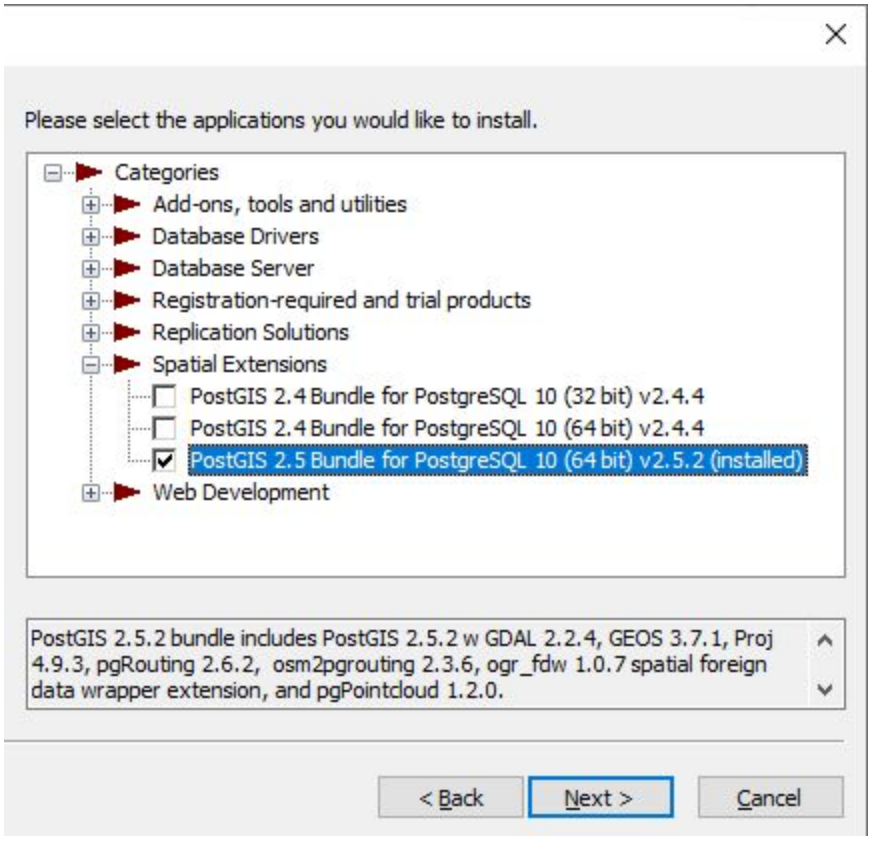



Installer PostGIS for PostgreSQL-serveren.

1. Kjør StackBuilder og veld postgreSQL-serveren:



2. Velg: Spatial Extensions → PostGIS 2.5 Bundle for PostgreSQL 10 (64bit) v2.5.2.




Lag og fyll databasen med data:

1. Last ned script for å fylle databasen med data: https://drive.google.com/file/d/1b1i2ezNvWjxfdGvRDi1ghLVBzwl56QlP/view
2. Pakk ut den komprimerte mappen, og kjør 'DropCreateMadartDatabase.bat'. Dette kan ta litt tid.

Installer .NET Core: Følg installasjonsveiledningen til Microsoft:
https://dotnet.microsoft.com/download
Klone repo-et og bygg løsningen:

```
git clone https://github.com/Fundator/Madart.OperationalUI.git
```

Kjør løsningen:

```
cd Madart.OperationalUI/Madart.OperationalUI.WebApp/
dotnet run
```

Åpne løsningen i nettleseren med url-en: https://localhost:5001/

# 10. Avinstallering

Denne veiledningen viser hvordan du kan avinstallere applikasjonen og slette databasen prosjektet bruker. Veiledningen forutsetter at du har installert prosjektet, som vist i forrige kapittel.

## 10.1 Slette prosjektmappen

For å slette mappen til prosjektet og alt innhold i mappen følg punktene under

1. Start opp Windows cmd.
2. Gå til mappen prosjektet ligger i ved hjelp av kommandoen "`cd`".
   Eksempel: "`cd \Users\bruker\Documents\Github`".
3. Bruk deretter kommandoen: "`rmdir /S Madart.OperationalUI`".
4. Du vil nå få et spørsmål om du vil slette mappen, dersom du vil slette mappen tast inn "y". Mappen og alt innholdet vil nå bli fjernet.

## 10.2 Slette databasen

Følg trinnene under for å slette databasen MADART1

1. Last ned script for å slette databasen(DropMADART1db.bat), https://drive.google.com/file/d/1xvwuhtnbhvulyrnyf3P8tMX6-RogPLvN/view?usp=sharing
2. Kjør 'DropMADART1db.bat', og skriv inn passordet til postgres, tast inn "y", og skriv inn passordet igjen. Databasen vil nå bli slettet.

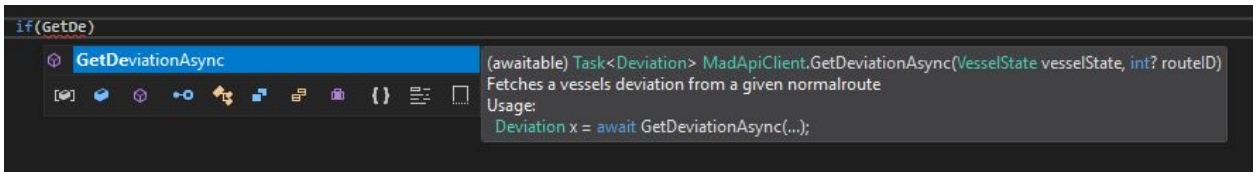
# 11. Dokumentasjon av kildekode

I kildekoden er det kommentert ved sentrale logiske uttrykk og ved bruk av sentrale metoder. De mest sentrale metodene er dokumentert ved implementasjon. Denne metoden for dokumentering følger Microsoft sin dokumentasjonsstandard [3].

```
/// <summary>
/// Fetches a vessels deviation from a given normalroute
/// </summary>
/// <param name="vesselState">The current vesselstate, containing location of the state</param>
/// <param name="routeID"> ID of the route the method fetches deviation from</param>
/// <returns> A deviation object containing data about the deviation. </returns>
public static async Task<Deviation> GetDeviationAsync(VesselState vesselState, int? routeID)
{
```

*Figur 10-1: Eksempel på dokumentering av metode.*

Alle metodene som er dokumentert i prosjektet har et lite sammendrag om hva metoden gjør, hva inputen er, og hva som blir returnert. Denne typen dokumentering er vist i figuren over.




*Figur 10-2: Eksempel på hvordan dokumentasjon av metode blir brukt*

Denne dokumenteringen gjør at når en programmerer opp mot metoden vil en få opp kort informasjon om metoden, dette er vist i figuren over.

# 12. Testing

Testene som er laget for prosjektet er NUnit-tester. Det blir laget tester på all funksjonalitet som vi mener det er nødvendig å teste, slik at prosjektet kjører og opererer som tenkt. Når vi refaktorerer koden vår vil testene si ifra om resultatene til metodene har blitt endret og vi kan sjekke om alt fungerer som tenkt.

## 12.1 Kjøre tester i Visual Studio

Dersom en kun vil kjøre testene, kan en i Visual Studio gå til Test -> Run -> All Tests. For en fin visualisering over testene kan en åpne Test Explorer ved å gå til Test -> Windows -> Test Explorer. Her vil alle passerte tester vises som grønne, testene som feilet som røde, og testene som ennå ikke har blitt testet som blå.

## 12.2 Kjøre tester fra konsolen

For å kjøre testene i konsolen så kan dotnet test-kommandoen benyttes.
Åpne terminalen og naviger til rotmappent til løsningen, og skriv:

```
dotnet test
```

# 13. Referanser


1. Rick-Anderson. Prevent Cross-Site Scripting (XSS) in ASP.NET Core [Internett]. [sitert 22. april 2019]. Tilgjengelig på: https://docs.microsoft.com/en-us/aspnet/core/security/cross-site-scripting

2. Miller R, Vega D, Martz S, Barbast M. Raw SQL Queries - EF Core [Internett]. Micorsoft. 2016 [sitert 22. april 2019]. Tilgjengelig på: https://docs.microsoft.com/en-us/ef/core/querying/raw-sql

3. Wagner B, Gens D, Potapenko N. Documenting your code with XML comments [Internett]. Microsoft. 2017 [sitert 16. mai 2019]. Tilgjengelig på: https://docs.microsoft.com/en-us/dotnet/csharp/codedoc