Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap
Institutt for datateknologi, elektroteknologi og realfag

Automatisert modellering av integrasjoner

Automatic modeling of integrations

Systemdokumentasjon

Versjon <3.1>

REVISJONSHISTORIE

Dato	Versjon	Beskrivelse	Forfatter
14.04.2023	<1.0>	Innledning, arkitektur, klassediagram, sikkerhet	Sindre Holtan Mathias Pham
03.05.2023	<2.0>	Arkitektur, klassediagram og server-tjenester	Mathias Pham Sindre Holtan
14.05.2023	<3.0>	Installasjon og kjøring, dokumentasjon av kildekode, Kontinuerlig integrasjon og testing	Mathias Pham Sindre Holtan
21.05.2023	<3.1>	Oppdatert prosjektstruktur	Mathias Pham Sindre Holtan

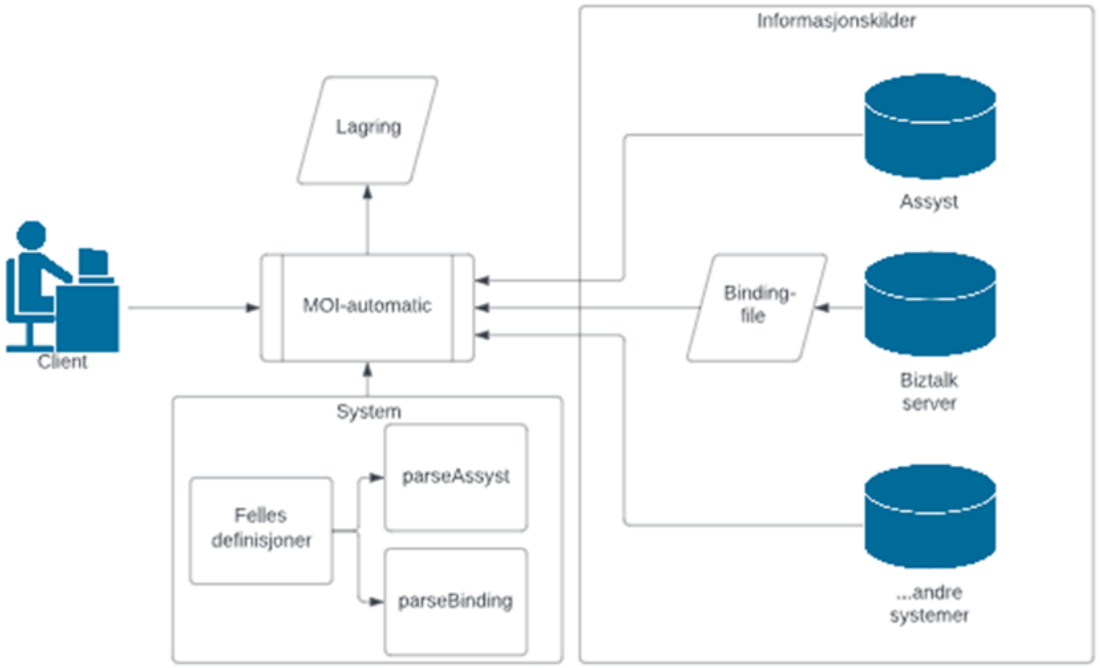
INNHOLDSFORTEGNELSE

1 INNLEDNING

Dette dokumentet inneholder systemdokumentasjon for programmet MOI-AUTOMATIC. Hensikten med dokumentet er å beskrive systemets design og funksjonalitet på en måte som gir tilstrekkelig informasjon for videreutvikling av programmet. Dokumentet inkluderer en arkitekturskisse for systemet, en beskrivelse av fil- og katalogstrukturen for prosjektet, sikkerhetstiltakene tilrettelagt for programmet, og en kort forklaring av server-tjenester som brukes i løsningen.

I tillegg til dette inneholder dokumentet instruksjoner for installasjon og kjøring av applikasjonen, dokumentasjon av kildekoden og en beskrivelse av hvordan systemet blir testet.

2 ARKITEKTUR



Figur 2.1: Arkitekturskisse av systemet

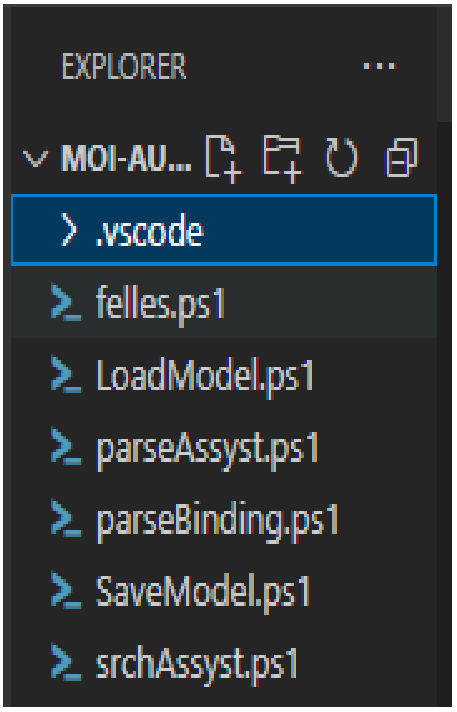
I Figur 2.1 vises en overordnet skisse av løsningens arkitektur og flyten i programmet.

I MOI-Automatic har en klient muligheten til å hente informasjon fra applikasjoner og integrasjoner, inkludert avhengigheter og relasjoner, fra ulike informasjonskilder. Disse kildene inkluderer Assyst (Helse Vest IKT sin CMDB), BizTalk Server hvor det hentes ut binding-files og andre systemer.

I senter av systemet finner vi en felles modul kalt "*Felles definisjoner*" som er tilknyttet både parseAssyst og parseBinding. Disse parse-skriptene blir kjørt under innhenting av data, og resultatene blir lagret i en array ved hjelp av sesjonsvariabler. Ved hjelp av felles modulen, blir det opprettet en modell som brukes av både parseAssyst og parseBinding. Dette muliggjør kobling av informasjonen mellom disse skriptene, da de opererer på samme modell. På denne måten kan data og funksjonalitet deles og samordnes på en effektiv måte.

Programmet tar i bruk sesjonsvariabler, men siden disse blir kjørt i minnet, risikerer dataen å bli tapt ved restart av sesjonen. Derfor har programmet en lagringsmetode for å sikre at dataene blir lagret og ikke går tapt.

3 PROSJEKTSTRUKTUR

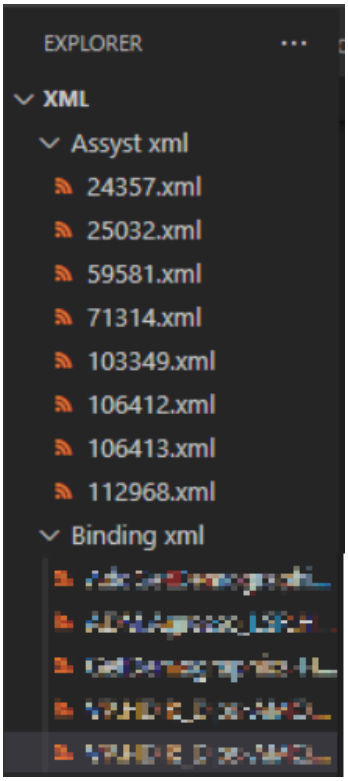


Figur 3.1: Prosjektkatalog

Figur 3.1 viser til en overordnet filstruktur for prosjektet som i utgangspunktet inneholder PowerShell skripting i Visual Studio Code.

felles.ps1 - fellesmodul for å opprette objekter, finne objekter og knytte relasjoner.
parseAssyst.ps1 - parser en fil fra Assyst og kobler denne til modellen (*felles*).
parseBinding.ps1 - parser en fil fra en BizTalk Binding file og kobler denne til modellen (*felles*).
LoadModel.ps1 - Henter inn en lagret versjon av sesjonsvariabel
SaveModel.ps1 - Lagrer modellen (sesjonsvariabel) til fil.
srchAssyst.ps1 - søk på navn i Assyst etter entiteter/objekter.

Data i XML

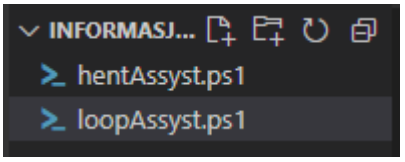


Figur 3.2: data-mappe

Figur 3.2 viser mappestrukturen for de innhentede dataene fra Assyst og BizTalk Binding-filer. Dataen blir først lagret i en XML-mappe, før den blir delt inn i to separate mapper for Assyst og BizTalk Binding-filer.

Noen av dataene har blitt brukt i prosjektet
både til analyse av data og for å koble relasjoner mellom systemene.

Informasjonsinnhenting



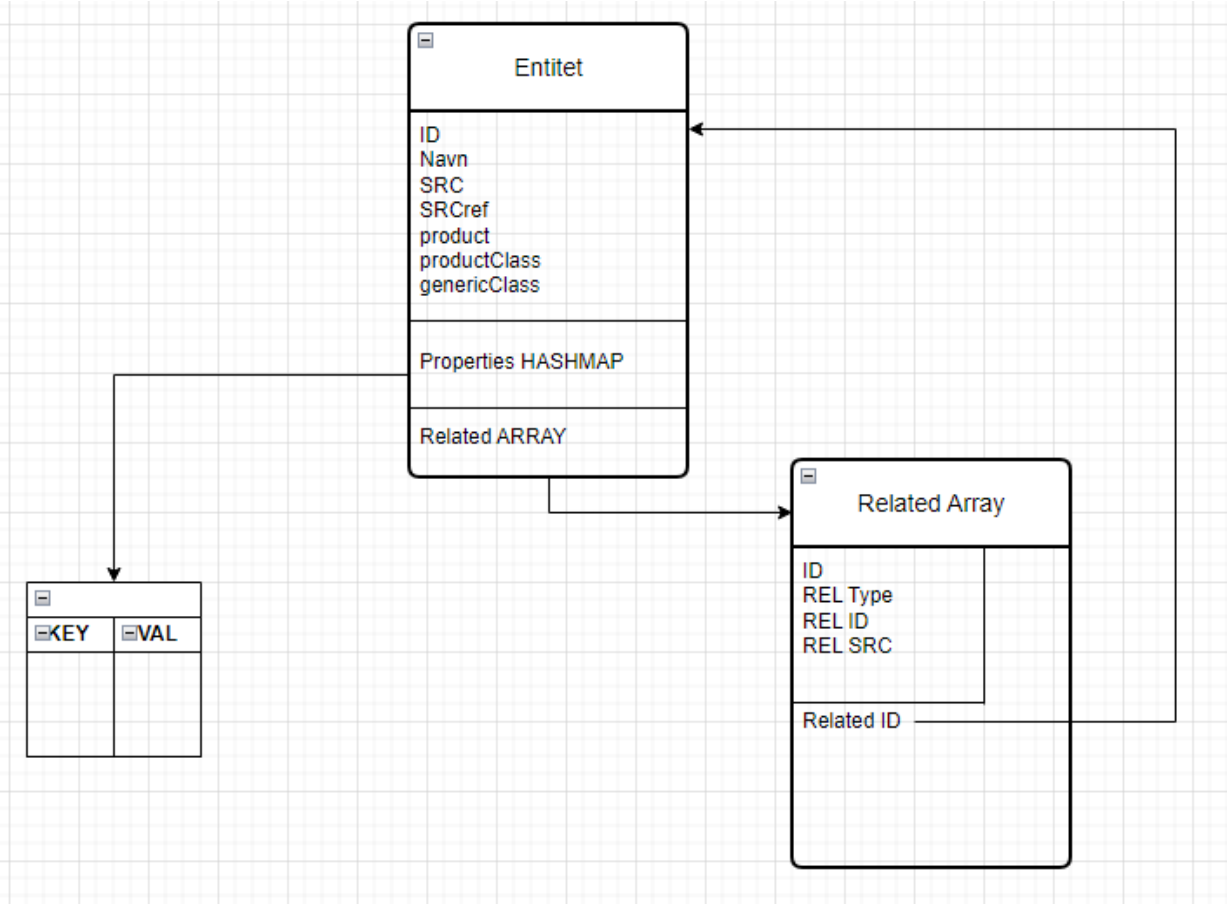
Figur 3.3: Prosjektkatalog

Figur 3.3 viser til en overordnet filstruktur for støtte skript som gruppen lagde for å hente ut XML-filer fra Assyst.

hentAssyst.ps1 - Henter ned info fra Assyst basert på en ID for å så lagre filen. Lagrer så filen i XML-format basert på ID ved den innhentede filen.

LoopAssyst.ps1 - Lagrer all related.ID i en array. Deretter kjøres **hentAssyst.ps1** i loop mot hver ID som ligger i arrayet.

4 KLASSEDIAGRAM



Figur 4.1: Klassediagram av MOI-Automatic

En entitet eller et objekt kan representere mange forskjellige ting, fra servere til applikasjoner. Hvis det finnes en relasjon til en annen entitet, vil den være koblet via et "Related Array", hvor "Related ID" knyttes til ID-en til den relaterte entiteten. Egenskaper/Properties blir lagret i en HASHMAP, som inneholder både en nøkkel (key) og en verdi (value).

5 Server-tjenester

Selve programmet MOI-AUTOMATIC bruker ikke server-tjenester, men det blir brukt et skript ("HentAssyst.ps1") for å hente inn informasjon fra en REST-tjeneste i Assyst. Applikasjoner og integrasjoner blir lagt inn i Assyst av systemforvaltere fra Helse Vest IKT og denne informasjonen om applikasjoner (og andre enheter) blir hentet inn fra Assyst via skriptet.

PowerShell er et verktøy som ble brukt til å hente inn denne informasjonen ved bruk av kommandoen **".\hentAssyst.ps1 103349"** vist i figur 6.1**.** Skriptet henter inn informasjon fra testmiljøet "assyst.ihelse.kurs" og oppretter en XML-fil som inneholder informasjonen om den aktuelle applikasjonen.

```
5
6   Nøkkel her er 103349 (Biztalk stage)
7   Eks:
8   .\hentAssyst.ps1 103349
9
10  #>
11  Param(
12      [Parameter(Mandatory=$true,HelpMessage="Trenger kode for systemID")][string] $systemID
13  )
14
15  $resp = $null
16  $urlSource = "https://assyst....................."+$systemID+"?fields=product,product.productClass.genericClass,product.GenericClass,relations.relate
17  Write-Host "URL          : "$urlSource
18  try {
19      $resp = Invoke-WebRequest -UseDefaultCredential -URI $urlSource
20  }
21  catch {
22      Write-Host $_
23  }
24  if(200 -eq $resp.StatusCode) {
25      $outFilename = "C:\Users\sinhol\OneDrive - Helse Vest\Dokumenter\Assyst\xml files\"+$systemID+".xml"
26      Write-Host "Skriver til "$outFilename
27      Out-File -FilePath $outFilename -Force -InputObject $resp.Content
28  } else {
29      Write-Host "Statuscode"$resp.StatusCode -ForegroundColor Red
30      $resp
31  }
32
```

Figur 5.1 utklipp av skriptet: hentAssyst.ps1 som bruker REST-tjenesten for å hente informasjon

6 SIKKERHET

Programmet har ikke har noen sikkerhetstiltak som er direkte implementert, men krever at en bruker er logget på en klientmaskin som er koblet til Helse Vest intranett for å kunne kjøre programmene. For å kjøre skriptene kreves det også at brukeren har høye nok rettigheter, på grunn av restriksjoner for kjøring av skript på generell basis. For å få tilgang til Assyst må brukeren være tilknyttet i Active Directory (AD) og ha tilgang til nettverksressurser som Assyst benytter seg av. Dette er et tilleggskrav for å kunne bruke programmet.

På grunn av restriksjoner og nødvendige rettigheter hos Helse Vest IKT, vil det ikke bli mulig for en bruker uten tilganger hos Helse Vest IKT til å kjøre skriptene **hentAssyst.ps1**, **srchAssyst** og **loopAssyst.ps1** (vist i figur 3.3 i kapittel 3). Disse restriksjonene fører også til at de originale XML-filer fra Assyst og Biztalk binding-files ikke blir tilgjengelige da de inneholder sensitiv informasjon om Helse Vest IKT sine systemer.

7 INSTALLASJON OG KJØRING

Veiledning for å installere nødvendige program for kjøring av programmet på egen maskin. For å kjøre programmet blir brukeren nødt til å ha en av programmene gitt under for kjøring av skriptene i programmet. Gruppen anbefaler Visual Studio Code for dette da dette er svært brukervennlig, men det er også mulig å bare bruke PowerShell. Nedenfor blir stegene vist i rekkefølge.

1) Visual Studio Code

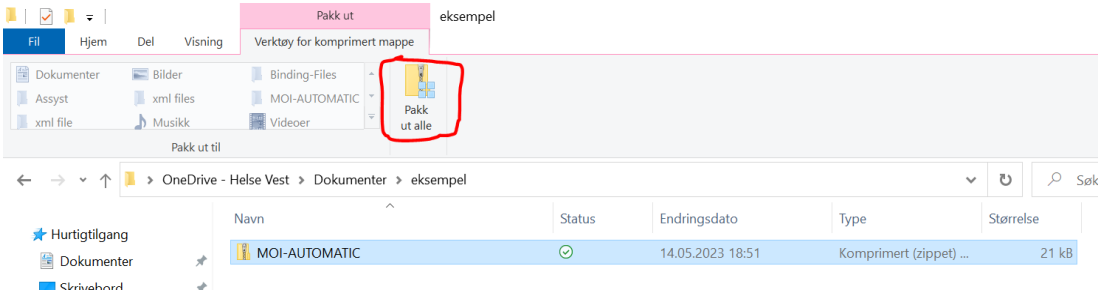
- Visual Studio Code er en populær teksteditor og utviklingsmiljø som brukes til å skrive og redigere kode.
- Installasjonslink: https://code.visualstudio.com/download

2) Valgfri: 7zip

- 7-Zip er et program som brukes til å komprimere og dekomprimere filer.
- Installasjonslink: https://www.7-zip.org/

3) Utpakking av zip-filer

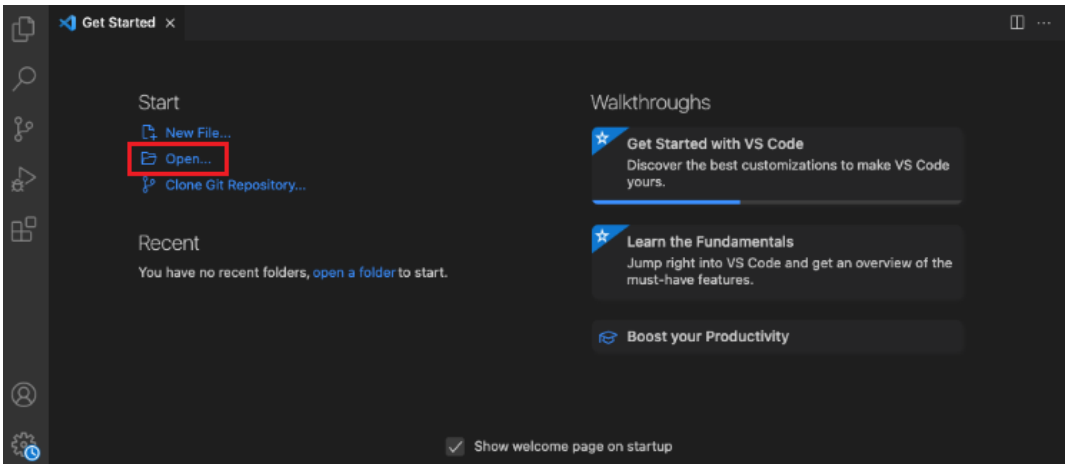
- Hvis du ikke har et utpakknings-vektøy tilgjengelig, anbefales det å laste ned et verktøy som 7zip.
- Pakk ut zip-filene til en ønsket lokasjon på din pc.



Figur 7.1 Pakker ut zip-filen

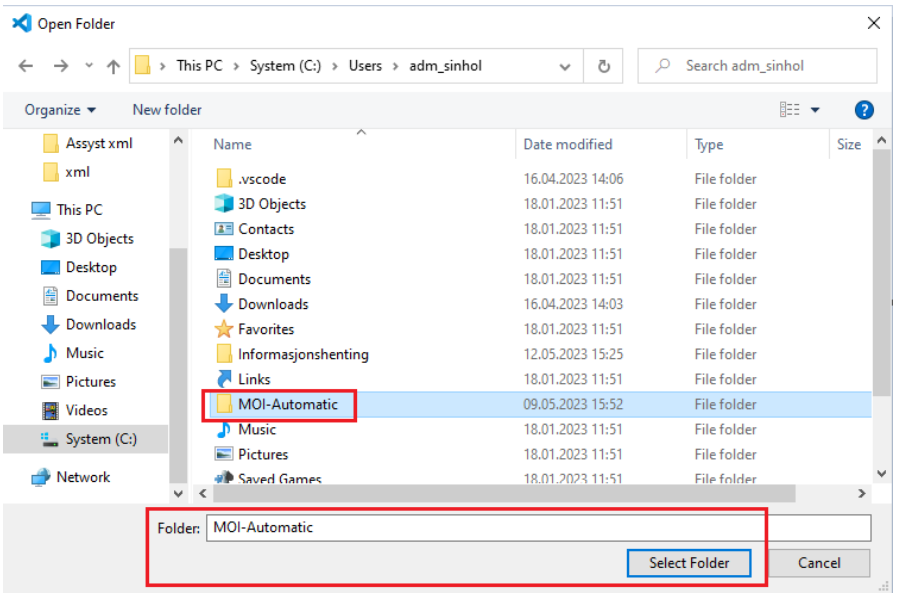
4) Hent skriptene til Visual Studio Code

- I et tomt Visual Studio Code får man opp en startskjerm. Her velger brukeren “Open Folder…”



Figur 7.2: Åpne mappen i Visual Studio Code

- Her vil det dukke opp et vindu, og her velger du lokasjonen hvor du lagret MOI-AUTOMATIC mappen i steg 3.



Figur 7.3: Velg lokasjonen for MOI-Automatic mappen

7.1 Kjøring av programmet

Som nevnt tidligere, vil tre skript ikke fungere (loopAssyst, hentAssyst og srchAssyst) på grunn av restriksjoner. Gruppen har laget 2 anonymiserte filer (1 Assyst-fil og 1 Binding-file) som en ekstern bruker kan bruke til å hente inn informasjon med skriptene i MOI-AUTOMATIC.

Set-ExecutionPolicy Bypass -Scope Process -Force
Det kan være nødvendig å kjøre denne kommandoen for at programmet skal fungere under kjøring av skript i PowerShell (Visual studio code bruker PowerShell terminal under kjøring). Denne kommandoen kan omgå rettighetene for den gjeldende prosessen eller sesjonen og lar skript bli utført uten restriksjoner (Microsoft, u.å).

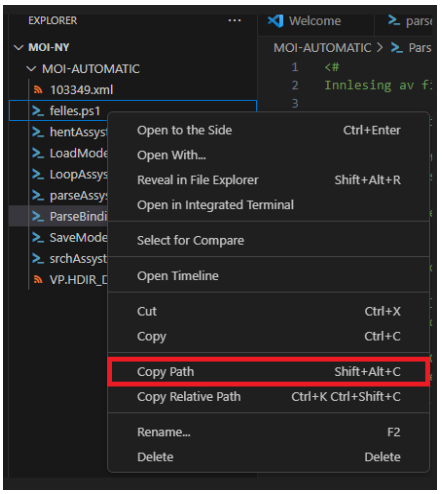
Kommandoen "Set-ExecutionPolicy Bypass -Scope Process -Force" tillater midlertidig kjøring av skript uten begrensninger i PowerShell, og sikrer problemfri utførelse av programmet.
Da dette er en midlertidig kjøring, vil ‘’policyen” bare bli endret i økten. Dette vil si at om det blir laget en ny økt, så må kommandoen kjøres på nytt.

NB! Bruken av denne kommandoen kan potensielt utgjøre en midlertidig sikkerhetsrisiko (inntil økten er avsluttet). Dette skyldes at kommandoen tillater kjøring av alle skript, også de som er lastet ned fra internett eller mottatt fra ukjente kilder, uten noen form for advarsel eller bekreftelse. For å sikre en trygg praksis, bør man være forsiktig og kun benytte denne muligheten når det er absolutt nødvendig. I tillegg er det viktig å kontrollere at skriptet som kjøres, stammer fra pålitelige kilder.

Tilpasse skriptene til ditt system:

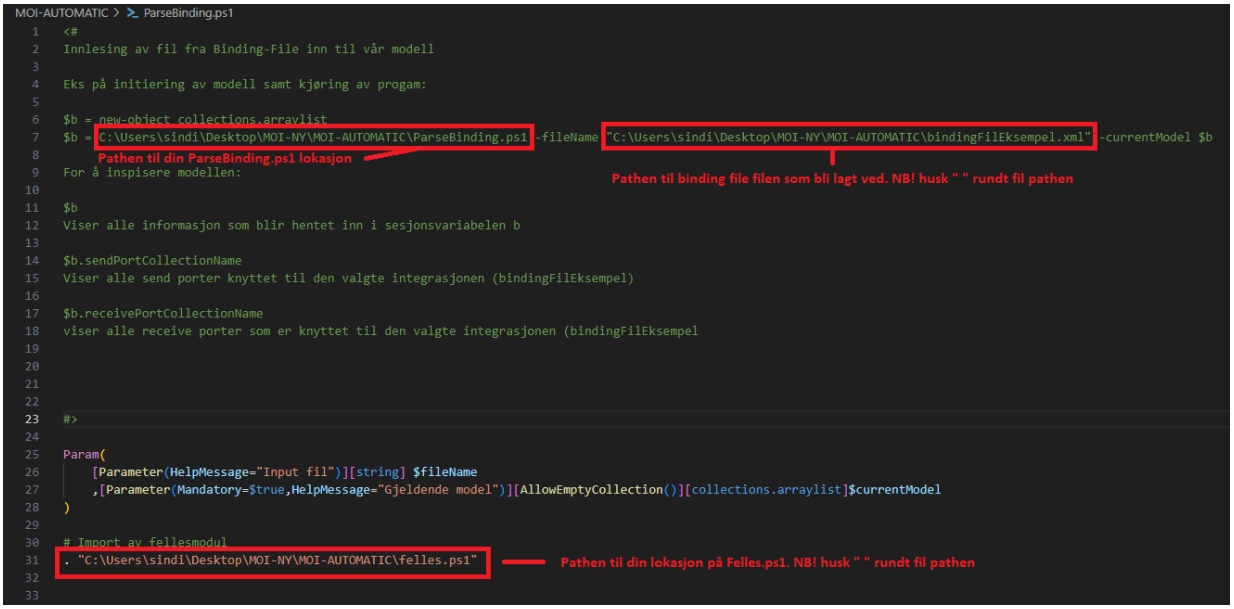
Før brukeren kan kjøre skriptene, er det viktig å først endre enkelte linjer på skriptene, slik at de vil være utførbare for brukeren. Under vises bilder med tekst som forklarer hvordan en bruker kan endre skriptene slik at de vil være kjørbare.

- En enkel måte å kopiere lokasjonen til filene er å gjøre som dette eksempelet: Høyreklikk på ønsket skript/fil hvor du ønsker å kopiere lokasjonen (felles.ps1 er vist i figur 7.4) og trykk på "Copy Path".

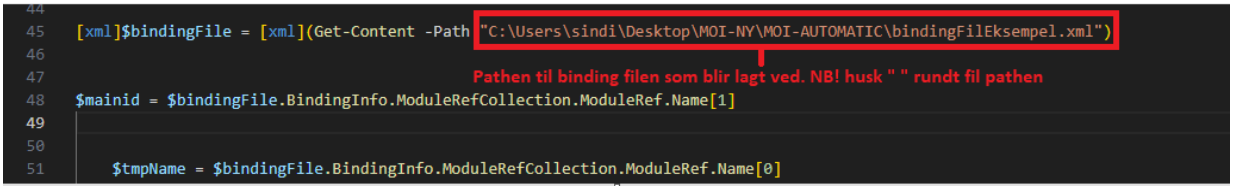


Figur 7.4: Copy Path til filen

- I ***parseBinding*-skriptet** må du endre stien fire steder. Bildene under viser hvor stien skal endres. Husk anførselstegn rundt fil-pathen.



Figur 7.5: Bildet viser hvor man skal endre stien i parseBinding



Figur 7.6: Ekstra linje i parseBinding hvor stien må endres.

- I ***parseAssyst*** blir det slikt:

Figur 7.7: Viser til hvor stien skal endres i parseAssyst.

Kjøring av skriptene:

- For å få terminal opp: Trykk på "Terminal", deretter "New Terminal" i Visual Studio Code (ligger øverst i programmet).
- I terminalen kan kommandoene bli brukt.
- Den tomme tabellen må alltid opprettes først:
  $a =new-object collections.arraylist
- Deretter kan du lime inn den andre linjen rett under som innhenter informasjonen til sesjonsvariabelen $a (det er markert med rødt, der du må endre til dine lokasjoner):
  $a =DIN LOKASJON FOR PARSEASSYST.PS1 -fileName "DIN LOKASJON FOR ASSYSTFILEN" -currentModel $a
- I skriptfilene (*parseAssyst og parseBinding*) finner du instruksjoner om hvilke kommandoer du kan bruke øverst i form av kommentarsetninger.

8 DOKUMENTASJON AV KILDEKODE

Dokumentasjon av kildekode spiller en viktig rolle i å forbedre lesbarheten og forståelsen av koden. Ved å inkludere relevante kommentarer, kan det bidra til å forklare komplekse deler av koden, logiske beslutninger og eventuelle avhengigheter mellom forskjellige deler av koden. Gruppen har inkludert kommentarer i koden i form av små kommentarer etterfulgt av "#". Gjennom kommentarer i kildekoden kan gruppen forklare hensikten med ulike variabler, konstanter og metodekall. "<#" blir brukt for større kommentarer, som f.eks instruksjon for kjøring av skriptene, som står i starten av filene.

```
function createObject() {
    $object = New-Object -TypeName PSObject

    Add-Member -InputObject $object -MemberType NoteProperty -Name GenID -Value "" # Generisk ID for alle typer referanser. Enkelte systemer krever 'id-' i begynnelsen.
    $object.GenID = "id-"+[guid]::NewGuid()
    Add-Member -InputObject $object -MemberType NoteProperty -Name id -Value ""  # Identifikator for artifakt. (Assyst item.shortcode) NB! kan være case sensitiv
    Add-Member -InputObject $object -MemberType NoteProperty -Name name -Value ""  # Navn på artifaktet - eks Assyst item.name

    Add-Member -InputObject $object -MemberType NoteProperty -Name SRC     -Value ""  # Kilde/system for dette artifaktet (hvor vi hentet informasjonen)
    Add-Member -InputObject $object -MemberType NoteProperty -Name SRCref  -Value ""  # Kildereferanse for dette artifaktet

    # Har denne blir sjekket for avhengigheter?  Hvis ikke, kan dette være 'en løs tråd'.
    # Eks. kan man lage et objekt basert på info fra Assyst, men det er ikke sikkert vi har sjekket alle avhengigheter (som vi snakket litt om)
    Add-Member -InputObject $object -MemberType NoteProperty -Name dependencyDiscovery -Value $false

    # Produkthierarki (har ikke tatt med id'er - om dere skulle trenge det) - her har vi det meste fra Assyst.
    Add-Member -InputObject $object -MemberType NoteProperty -Name product         -Value $null  # product.shortCode
    Add-Member -InputObject $object -MemberType NoteProperty -Name productClass    -Value $null  # product.productClass.shortCode
    Add-Member -InputObject $object -MemberType NoteProperty -Name genericClass    -Value $null  # product.productClass.genericClass.shortCode

    Add-Member -InputObject $object -MemberType NoteProperty -Name Properties      -Value @{}  # Hasmap of properties - test/prod/stage...
    Add-Member -InputObject $object -MemberType NoteProperty -Name Related -Value @() # Array of related objects
    $object.Related = @()  # Array av relasjoner fra dette objektet.
    return $object
}
```

Figur 8.1: Kommentarsetninger som beskriver objektene

```
felles.ps1   103349.xml   parseAssyst.ps1   SaveModel.ps1   HentAs
parseAssyst.ps1
3
4   Eks på initiering av modell samt kjøring av progam:
5
6   $a = new-object collections.arraylist
7
8   $a = .\parseAssyst.ps1 -fileName ..\xml\103349.xml -currentModel $a
9
10  For å inspisere modellen:
11  $a.Count
12  49 er antall objekter i modellen.
13
14  $a[0]
15  viser objekt 0 i arraylist.
16
17  $a[0].Related
18  viser alle relasjoner fra dette objektet
19
```

Figur 8.2: Kommentarsetninger som viser til instruksjoner

9 REFERANSER

Intern dokumentasjon om Helse Vest IKT sine systemer.

- Microsoft (Uten år) *Set-ExecutionPolicy.* Tilgjengelig fra: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy?view=powershell-7.3 (Hentet 10.05.2023)