Klimadatastyrelsen

Brugervejledning -Transitionsguide for entitetsbaserede GraphQL-tjenester

Drift og modernisering af Datafordeleren

September 2025

Version 3.5 – 16-09-2025

Indholdsfortegnelse

1 Indledning

Dette dokument er en guide til hvordan man som Anvender kan bruge Datafordelerens entitetsbaserede GraphQL-tjenester.

Dokumentet er opdelt i tre dele:

1) Første del af dokumentet beskriver hvad entitetsbaserede GraphQL-tjenester er og hvordan de bruges.
2) Anden del af dokumentet beskriver hvordan autentifikation og autorisation håndteres.
3) Tredje del af dokumentet forklarer forskelle mellem GraphQL og de nuværende REST-tjenester på Datafordeleren, samt hvordan man som Anvender kan komme i gang med at anvende GraphQL-tjenester.

Dokumentet er tiltænkt følgende læsere:

1) Eksisterende anvendere af Datafordeleren der bruger den Nuværende Datafordelers REST-tjenester, som ønsker at komme i gang med at bruge de entitetsbaserede GraphQL-tjenester i stedet.
2) Nye anvendere, der vil i gang med at bruge GraphQL-tjenesterne på Datafordeleren.

1.1 Begreber

I dette dokument bruges begreberne i Tabel 1. Bemærk at begreberne også bliver yderligere forklaret i løbet af dokumentet med eksempler.

Begreb	Beskrivelse
Nuværende Datafordeler	
REST-tjenester:	REST-tjenester er en metode til at få data fra Datafordeleren. Man kan som anvender hente data ved at bruge en URL. REST-tjenester er beskrevet her https://confluence.sdfi.dk/pages/viewpage.action?pageId=17138461.
Moderniserede Datafordeler	
GraphQL-tjenester:	GraphQL-tjenester beskrives i denne vejledning.
Entitet:	En entitet er en fysisk repræsentation af et registerobjekt som findes i Grunddatamodellen. De total- og deltadownload, der kan hentes fra Datafordeleren, samt de entitetsbaserede GraphQL-tjenester indeholder hver især data bestående af en enkelt entitet.
GraphQL-skemaer:	GraphQL-skemaer er .graphql-filer der genereres på baggrund af registrenes datamodeller og er yderligere beriget med metadata fra hvert register. Skemaerne beskriver hvordan data er struktureret og hvordan data kan hentes.
Paging:	Paging er når resultater inddeles i sider således at resultater returneres én side ad gangen.

Tabel 1: Begrebsliste.

2 GraphQL på Datafordeleren

GraphQL er en funktionalitet på Datafordeleren til udstilling af data. Dette dokument beskriver hvordan anvendere kan benytte GraphQL-tjenesterne til at hente tabulære data fra registrene på Datafordeleren.

2.1 Hvad er GraphQL?

GraphQL er et moderne API-sprog, der gør det muligt for anvendere at hente data på en effektiv og fleksibel måde. GraphQL giver mulighed for at hente præcis de oplysninger, du som anvender har behov for. GraphQL-tjenesterne er derfor velegnede når du ønsker at hente mindre datamængder og ikke nødvendigvis ønsker at postprocessere data efterfølgende.

Du kan bruge GraphQL-tjenesterne ved at sende en forespørgsel, der beskriver de data du ønsker, til Datafordeleren, hvorefter serveren kun svarer med de specifikke data, der blev anmodet om. Dette kan du blandt andet gøre gennem en desktop applikation eller gennem browseren ved brug af en browser-extension.

Med GraphQL kan du definere hvilke datafelter de ønsker returneret. Dette betyder, at du som anvender kun modtager netop de oplysninger, du har brug for. Det kan dermed forbedre svartiden for tjenesterne og reducere mængden af data, som overføres.

Yderligere information om GraphQL kan findes på **graphql.org**.

Måden du bruger GraphQL-tjenesterne ved at sende en såkaldt *query* til Datafordelerens GraphQL-endepunkter. En query er anvendernes måde at definere hvilke data du ønsker at hente.

Et eksempel på en POST request mod tjenesten https://graphql.datafordeler.dk/BBR/v1 kan ses i Figur 1 herunder.

```
query {
  BBR_Bygning( # Entitet
    registreringstid: "2025-03-23T14:41:33Z" # Bitemporal point-in-time filtrering på registreringstid
    virkningstid: "2024-11-12T14:41:33Z" # Bitemporal point-in-time filtrering på virkningstid
    first: 10 # Pagination
    where: { # Standardfiltrering
      kommunekode: { eq: "0550" }
      status: { eq: "6" }
    }
  ) {
    pageInfo { # Inkluder Pagination information
      endCursor
      hasNextPage
    }
    nodes { # Inkluder Entitetens felter
      id_lokalId
      kommunekode
      virkningsaktoer
      forretningsproces
    }
  }
}
```

Figur 1: Eksempel på en GraphQL-query

GraphQL-queries fungerer således:

- **Entitet** (beskrevet i afsnit 2.2)
  - En query skal specificere hvilken entitet den omhandler.
  - Eksemplet ovenfor viser en query for entiteten BBR_Bygning fra BBR.
- **Tjeneste** (beskrevet i afsnit 2.3)
  - En query skal sendes til en GraphQL-tjeneste, der indeholder den pågældende entitet, som et GET- eller POST-request.
- **Standardfiltre** (beskrevet i afsnit 2.4)
  - En query kan indeholde standardfiltre der filtrerer data.
  - Eksemplet ovenfor indeholder to standardfiltre: lighedsoperator (eq: "0550") på kommunekode og (eq: "6") på status.
- **Geometriske filtre** (beskrevet i afsnit 2.5)
  - En query kan indeholde geometriske filtre, der filtrerer data på baggrund af geometri-felter.
- **Bitemporale filtre** (beskrevet i afsnit 2.6)
  - En query kan indeholde bitemporale filtre, der filtrerer data på baggrund af bitemporale felter.
  - Eksemplet ovenfor indeholder bitemporalt point-in-time-filter (virkningstid: "2024-11-12T14:41:33Z") på virkningstidspunktet og (registreringstid: "2025-03-23T14:41:33Z ") på registreringstidspunktet.
- **Minimumsfiltrering** (beskrevet i afsnit 2.6.2)
  - På bitemporale entiteter skal en query indeholde mindst et bitemporalt point-in-time-filter eller et standardfilter på feltet id_lokalid.
  - Eksemplet ovenfor indeholder to bitemporale point-in-time-filtre, og opfylder derfor minimumsfiltrering.
- **Paging** (beskrevet i afsnit **Fejl! Henvisningskilde ikke fundet.**)
  - En query skal definere hvilke felter fra entiteten der forespørges og hvordan resultaterne paging.
  - Eksemplet ovenfor returnerer de første 10 resultater (first: 10) for felterne (angivet i ”nodes”) id_lokalid, kommunekode, virkningsaktoer, forretningsproces samt paging-informationer (angivet i ”pageInfo”) for resultatsættet.
- **Schema** (beskrevet i afsnit 2.8)
  - Hvorvidt en query er gyldig vurderes iht. GraphQL-skemaet, der definerer hvordan data er struktureret, hvilke typer af data der kan hentes, mulige filtreringsoperationer mm.

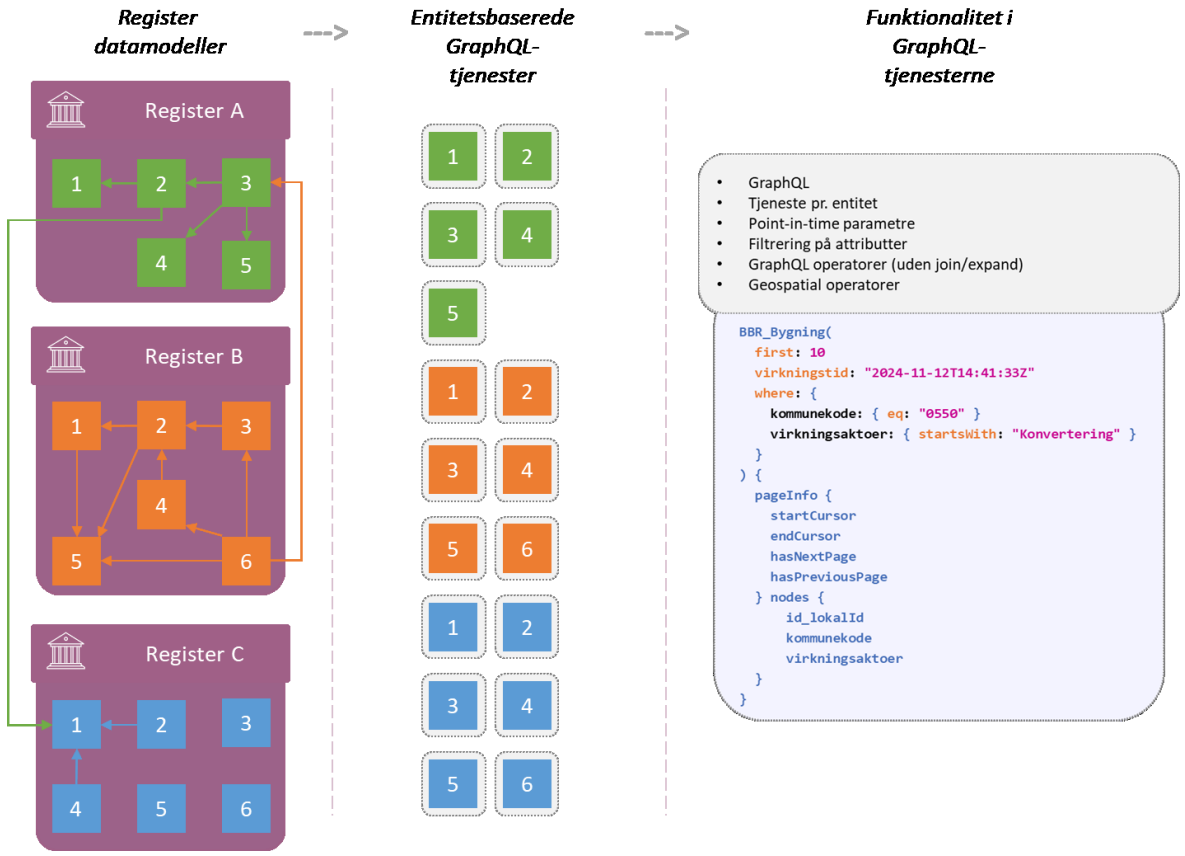
2.2 Om entitetsbaserede GraphQL-tjenester

GraphQL-tjenester tilbydes som **entitetsbaserede GraphQL-tjenester**. Entiteterne baserer sig på datamodeller fra Grunddatamodellen, jf. afsnit 1.1. En entitet kan f.eks. være en "Adresse" fra DAR, en "Bygning" fra BBR eller et "Jordstykke" fra MAT. Det at GraphQL-tjenesterne er **entitetsbaserede** betyder at et kald til en af tjenesterne returnerer data for en enkelt type af entitet. De entitetsbaserede GraphQL-tjenester leverer udelukkende tabulære data og ikke rasterdata (billeddata). Tabulære data forstås som strukturerede data der kan opbevares i tabeller, som f.eks. adresser, men ikke billeddata som f.eks. skærmkort.

Brug af API’et til entitetsbaserede GraphQL-tjenester beskrives nærmere i afsnittet 2.11.

Nedenstående Figur 2 giver et overblik over konceptet for entitetsbaserede GraphQL-tjenester. Figuren illustrerer 3 fiktive registre

- **Register A**, som har 5 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret og har også en enkelt reference til en entitet i register C udenfor registeret.
- **Register B**, som har 6 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret og har også en enkelt reference til en entitet i register A udenfor registeret.
- **Register C**, som har 6 entiteter. Registerets entiteter refererer til hinanden indbyrdes indenfor registeret.



Figur 2: Opsummering af GraphQL-funktionalitet

Figur 2 illustrerer at entitetsbaserede GraphQL-tjenester ikke udstiller relationer mellem entiteter, samt at tjenesterne ikke kan sammenstille entiteter. Den understøttede funktionalitet (se afsnit 2.1 for et overblik) uddybes i de følgende afsnit.

2.3 GraphQL-endepunkter

GraphQL-tjenesterne udstilles gennem URL'er, der følger nedenstående form:

https://graphql.datafordeler.dk/<register>/<version>

<register> er forkortelsen for det register du som anvender forsøger at hente og <version> er versionen af tjenesten for det pågældende register. Versionering er yderligere beskrevet i afsnit 2.9.

Alle registre, hvis entiteter udstilles via GraphQL-tjenesterne, er vist i Tabel 2 nedenfor.

Register	Forkortelse
Bygnings- og Boligregistret	BBR
Danmarks Administrative Geografiske Inddeling	DAGI
Danmarks Adresseregister	DAR
Danmarks Fikspunktregister	FIKSPUNKT
DHM Højdekurver	DHMHoejdekurver
DHM Oprindelse	DHMOprindelse
Danske Stednavne	DS

Det Centrale Personregister	CPR
Det Centrale Virksomhedsregister	CVR
Ejendomsbeliggenhedsregistret	EBR
Ejendomsvurdering	VUR
Ejerfortegnelsen	EJF
GeoDanmark Vektor	GEODKV
Historiske kort og data	HISTKORT
Matriklen2	MAT
Skatteforvaltningens Virksomhedsregister	SVR

Tabel 2: Oversigt over registre der udstilles via GraphQL-tjenesterne.

2.3.1 Adgangsbegrænsede registre

Nogle registre indeholder data, f.eks. personoplysninger, som kræver godkendelse for at kunne tilgås. Disse data kaldes på Datafordeleren for adgangsbegrænsede data. Adgang til adgangsbegrænsede data tildeles af registrene ved at en anvender opretter et IT-system og laver en ansøgning gennem selvbetjeningen til det specifikke register på vegne af det nyoprettede IT-system, hvorefter det pågældende register først skal godkende ansøgningen før data kan tilgås. Læs mere om **Brugeradgang på datafordeler.dk og hvordan et IT-system oprettes i** Brugeroprettelse Datafordeler Administration**.**

Tabellen nedenfor viser hvilke registre der indeholder adgangsbegrænsede data:

Register	Tjeneste version	Entiteter
EJF	v1	Alle entiteter
CPR	v1	Alle entiteter med undtagelse af CPR_AdministrativEnhed, og CPR_AdministrativEnhedType
CVR	v1	CVRPerson
SVR	v1	Alle entiteter

Tabel 3: Oversigt over registre med adgangsbegrænsede data.

Bemærk

Der er kun adgang til de fortrolige entiteter fra EJF for offentlige myndigheder.

2.3.2 Specialudviklet GraphQL-tjeneste for CVR

Entiteten CVRPerson fra CVR har et enkelt felt som indeholder adgangsbegrænsede data, mens resten af felterne ikke er adgangsbegrænsede. Datafordeleren udstiller derfor den specialudviklede entitet CVRPersonBegraenset gennem en GraphQL-tjeneste, der giver anvendere mulighed for at hente en ikke-adgangsbegrænset udgave af entiteten CVRPerson. Entiteten CVRPersonBegraenset er identisk med CVRPerson bortset fra at feltet CPRPerson er fjernet.

Filtreringsmulighederne på CVRPersonBegraenset-entiteten er de samme som tillades på CVRPerson-entiteten, bortset fra at det ikke er muligt at filtrere på CPRPerson-feltet. GraphQL-skemaet ligner det for CVRPerson-entiteten, dog uden CVRPerson-feltet.

Tjenesten er tilgængelig på følgende URL:

https://graphql.datafordeler.dk/CVR/custom/<version>

2.4 Standardfiltrering

GraphQL-tjenesterne understøtter almindelige sammenligningsoperatorer og filtreringsmuligheder (f.eks. <, >, =, in). Tabel 4 viser eksempler på datatyper og understøttede operatorer. Det skal bemærkes, at det fremgår i GraphQL-skemaerne for de enkelte registre, hvilke filtreringsmuligheder der er for hver entitet og dennes felter. Tabel 4 kan ikke anvendes som reference når der skal sammensættes queries til det enkelte register, da det afhænger af den specifikke entitet og dennes felter præcis hvilke operatorer der er understøttet. Læs mere om GraphQL-skemaer i afsnit 2.8.

Det er muligt at bruge flere filtre i den samme query, selvom der er nogle begrænsninger for, hvordan filtrering er tilladt.

Datatyper	Understøttede operatorer
Comparable (Float, Int, Long, Short)	• In • Equal to (eq) • Less than (lt) • Less than or equal to (lte) • Not less than (nlt) • Not less than or equal to (nlte) • Greater than (gt) • Greater than or equal to (gte) • Not greater than (ngt) • Not greater than or equal to (ngte)
Comparable (Date, DateTime, TimeSpan)	• Equal to (eq) • Less than (lt) • Less than or equal to (lte) • Not less than (nlt) • Not less than or equal to (nlte) • Greater than (gt) • Greater than or equal to (gte) • Not greater than (ngt) • Not greater than or equal to (ngte)
Text (String)	• In • Equal to (eq) • Starts with (startsWith)
Boolean	• Equal to (eq) • Not equal to (neq)
UUID	• Equal to (eq) • In

Tabel 4: Understøttede datatyper og operatorer

Bemærk

Det fremgår i GraphQL-skemaet for det enkelte register, hvilke filtreringsmuligheder der kan anvendes for de enkelte entiteter og deres felter.

Tabellen kan **ikke** anvendes som reference når der skal sammensættes queries til det enkelte register, da det afhænger af den specifikke entitet og dennes felter præcis hvilke operatorer der er understøttet.

2.5 Geometrisk filtrering

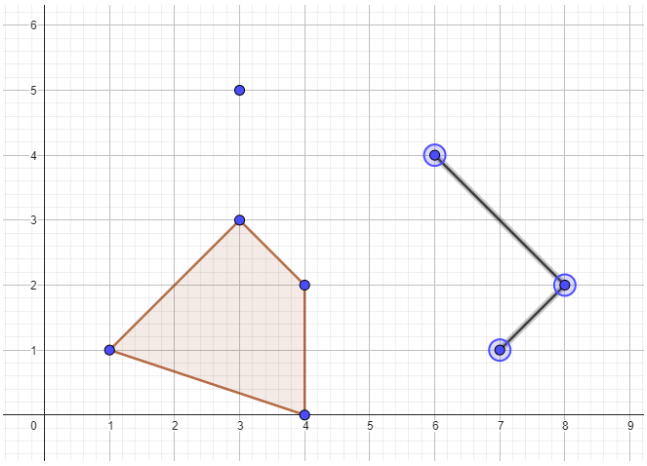
GraphQL-tjenesterne understøtter også geospatiale data og du kan derfor benytte filtre i form af WKT-strenge (Well Known Text) som gives som input til de geometriske filtre. Disse WKT'er repræsenterer geospatiale data, enten i form af en koordinattype, en geometri-type eller lignende datatyper.

De mest almindelige typer af geometriske former er punkter, linjer og polygoner. Et punkt repræsenteres af et koordinatsæt med to eller tre værdier, afhængigt af om det er et punkt i to eller tre dimensioner. En linje repræsenteres af koordinatparrene eller koordinattripletterne for de punkter, der udgør linjen, og en polygon repræsenteres af koordinatpar eller koordinattripletter, der udgør hjørnerne af polygonen. Et eksempel på WKT-formatet for hver af de tre geometrityper i 2D, er vist i Figur 3.

Point
POINT (3 5)

Line
LINESTRING (6 4, 8 2, 7 1)

Polygon
POLYGON (1 1, 3 3, 4 2, 4 0, 1 1)



Figur 3: Eksempler på 2D-geometrier

Datafordeleren understøtter en række forskellige operatorer. Alle operatorerne fremgår sammen med et link til deres PostGIS-dokumentation i tabellen nedenfor, og hver af funktionerne tager to geometriske former og returnerer en Boolean værdi. Kombinationer af spatiale operatorer understøttes også ved hjælp af tilhørende AND/OR-operatorer.

Datatyper	Understøttede operatorer	PostGIS Link
Geometri	contains(koordinatsystem, geometri)	https://postgis.net/docs/ST_Contains.html
	covers(koordinatsystem, geometri)	https://postgis.net/docs/ST_Covers.html
	coveredBy(koordinatsystem, geometri)	https://postgis.net/docs/ST_CoveredBy.html
	crosses(koordinatsystem, geometri)	https://postgis.net/docs/ST_Crosses.html
	intersects(koordinatsystem, geometri)	https://postgis.net/docs/ST_Intersects.html
	within(koordinatsystem, geometri)	https://postgis.net/docs/ST_Within.html
	overlaps(koordinatsystem, geometri)	https://postgis.net/docs/ST_Overlaps.html
	touches(koordinatsystem, geometri)	https://postgis.net/docs/ST_Touches.html

Tabel 5: Understøttede geometriske filtre og tilhørende PostGIS Link

Et eksempel på hvordan operatoren contains bruges kan ses i nedenstående figur.

```
query {
 DAR_NavngivenVej(
  registreringstid: "2024-07-11T19:12:45Z"
  where: {
   vejnavnebeliggenhed_vejnavneomraade: {
    contains: {
     crs: 25832, # koordinatsystem
     wkt: "POLYGON((
      507269.454538909 6220414.46082892,
      507305.574611149 6220539.06107812,
      507470.77494155  6220525.9010518,
      507433.33273166  6220424.33582511,
      507384.047971257 6220421.36601678,
      507269.454538909 6220414.46082892))" # geometri
    }
   }
  }
 ) {
  nodes {
   id_lokalId
  }
 }
}
```

Figur 4: Eksempel på en GraphQL-query med det spatiale operator ”contains”.

De geometriske filtre er implementeret således, at hvis man læser informationen i PostGIS-dokumentation link i Tabel 5, er geometri A altid værdien af registerdatafeltet, og geometri B altid inputgeometrien. Det vil sige, at rækkefølgen er operator(<registerdatafelt>, <input>). Hvis du ønsker den modsatte rækkefølge, bør du i stedet bruge en anden geometrisk operator, der repræsenterer den omvendte operator. En oversigt over disse kan ses i Tabel 6.

Geometrisk operator	Omvendte operator
contains	within
covers	coveredBy
coveredBy	covers
crosses	*Ingen. crosses er symmetrisk*
intersects	*Ingen. intersects er symmetrisk*
within	contains
overlaps	*Ingen. overlaps er symmetrisk*
touches	*Ingen. touches er symmetrisk*

Tabel 6: Geometriske operatorer og deres omvendte operatorer.

Syntaksen for at bruge geometriske filtre, og hvilke felter der kan filtreres geometrisk, er specificeret af GraphQL-skemaet for den pågældende tjeneste, som du ønsker at bruge.

Da geospatiale data kan have forskellige formater afhængigt af, hvilket register det drejer sig om, udfører Datafordeleren udelukkende validering på om det angivne koordinatsystem (CRS) er det samme koordinatsystem som registrets data og at den angivne WKT er valid og repræsenterer en topologisk valid geometri som specificeret i OGC SFS-specifikationen.

2.6 Bitemporal filtrering

På entiteter med bitemporale felter (forklares nærmere på Bitemporalitet | Datafordeler) er følgende to typer queries ofte anvendt:

- **Interval-queries**, hvor du ønsker data, der er bitemporalt gyldige i en bestemt tidsperiode.
- **Point-in-time-queries**, hvor du ønsker bitemporalt gyldige data pr. deres angivne tidspunkt.

Ved interval-queries har du mulighed for at specificere standardfiltre, jf. afsnit 2.4, på de bitemporale felter. Ved point-in-time-queries understøttes valgfrie argumenter for registreringstid og virkningstid, som kan angives, når du sender queries. Når disse argumenter ikke gives, returneres fuldt bitemporale data.

GraphQL-tjenesterne understøtter muligheden for at kombinere point-in-time-filtrering med enhver anden filtrering, der er tilladt, så man kan hente data, der overholder det angivne tidspunkt samt eventuelle intervalfiltre, geo-filtre eller lignende, de har angivet. Det vil sige, når point-in-time-filteret kombineres med andre filtre, vil forespørgslen ende med at være "point-in-time-filter OG (andre filtre)". Det er ikke muligt at sige point-in-time-filtreret 'ELLER' noget andet filter.

Bemærk

For at få data som er gyldigt på et givent tidspunkt, skal du sætte *både* virkningstid og registreringstid til forespørgselstidspunktet for at få data, der er gyldige på det pågældende tidspunkt.

2.6.1 Bitemporal filtrering for CVR

Alle registre der understøtter bitemporal filtrering, understøtter filtrering på både virkningstidspunkt og registreringstidspunkt bortset fra CVR. CVR er det eneste register, hvor du ikke kan specificere et registreringspunkt i din query. For CVR sættes registreringstidspunktet derimod *altid* til tidspunktet for forespørgslen, dvs. “registreringstid = NOW()”. Det er muligt at filtrere på virkningstidspunktet på sædvanlig vis.

2.6.2 Minimumsfiltrering

På entiteter med bitemporale felter kræves *minimum ét af følgende* filtre:

Filtreringstype	Filter
Bitemporal filtrering	Point-in-time filter på registreringstid
Bitemporal filtrering	Point-in-time filter på virkningstid
Standardfiltrering	Operator på feltet id_lokalid

Tabel 7: Minimumsfiltrering

2.7 Paging

GraphQL-tjenesterne kræver paging af resultater. At resultaterne *pagineres* vil sige at resultaterne inddeles i ”sider” og returneres én ad gangen. Du vil således få én side returneret ad gangen og kan efterfølgende lave et nyt kald for at få næste side for på den måde at gennemgå hele datasættet.

Paging er cursor-baseret, hvilket vil sige at der returneres en reference, en såkaldt *cursor,* til første og sidste resultat (for edges ved hvert enkelt resultat) i det returnerede datasæt ved hvert kald. Denne cursor kan derefter bruges til at hente flere sider ved at anmode om mere data fra cursorens værdi. I praksis kan du bruge cursor-værdien i en efterfølgende query, til at hente den næste side af data, startende fra den position, der er angivet ved den pågældende cursor. Paging er implementeret således at flere identiske anmodninger ved hjælp af den samme cursor vil returnere de samme data, forudsat at registrene ikke opdaterer de underliggende data i mellemtiden.

Paging er udelukkende implementeret som fremadgående paging. Det vil sige, at du kan hente den næste side af resultater fra en given cursor. Bagudgående paging understøttes ikke.

2.7.1 Paging i queries: PageInfo, nodes og edges

Paging består af tre datastrukturer: *PageInfo* samt *nodes* eller *edges*.

PageInfo indeholder metadata om den nuværende paging og består af følgende felter:

- **hasNextPage**: Dette felt angiver, om der er endnu en side med resultater.
- **hasPreviousPage**: Dette felt angiver, om der eksisterer en forrige side med resultater. Da vi ikke understøtter bagudgående paging, er dette altid sat til false.
- **startCursor**: Dette felt er en cursor, der peger på det første datapunkt i resultatdatasættet.
- **endCursor**: Dette er en cursor, der peger på det sidste datapunkt i resultatsættet. Denne cursor kan bruges af anvenderen til at hente den næste side af data.

Mens metadata kan hentes via PageInfo, kan data hentes enten som en liste af *nodes* eller en liste af *edges*. *Nodes* og *edges* adskiller sig på følgende måde:

- I resultatsættet er *nodes* en liste af datapunkter.
- I resultatsættet er *edges* en liste af datapunkter med et ekstra metadatafelt *cursor*, der matcher hvert datapunkt.

Det anbefales, at du benytter *nodes* til simpel paging, da PageInfo-metadataene er tilstrækkelige. Hvis du har brug for at paging fra en hvilken som helst position på den aktuelle side og ikke fra slutningen af den aktuelle side, skal *edges* benyttes.

Det er tilladt at specificere i GraphQL-queries, at du ønsker både *nodes* og *edges* returneret. Dette frarådes, da resultatsættet vil indeholde de samme data to gange.

2.7.2 Inputargumenter for paging

Paging har to inputargumenter, der kan specificeres i en query:

- *first*: Dette argument angiver hvor mange resultater der ønskes returneret. Dette er sidestørrelsen og skal være et ikke-negativt tal. Hvis dette argument udelades fra en query, returneres standardantallet (100) af resultater. Den maksimalt tilladte sidestørrelse er 1000, og hvis værdien af *first* er større end den maksimalt tilladte sidestørrelse returneres en fejlmeddelelse.
- *after*: Dette argument angiver cursoren til det første datapunkt i det resulterende datasæt. Hvis dette argument udelades fra query'en, returneres den første side.

Hvis du angiver ugyldige værdier for nogen af disse argumenter, vil tjenesterne returnere en fejlmeddelelse, der informerer om, at der er angivet ugyldige paging-argumenter.

2.7.3 Eksempel på paging-query

Følgende query kan sendes som en POST-request og bruges til at hente de første 5 datapunkter (første side) af BBR_Bygning-entiteten:

```
query {
 BBR_Bygning(
  first: 5
  registreringstid: "2025-03-23T14:41:33Z"
  virkningstid: "2025-03-23T14:41:33Z"
  ) {
  pageInfo {
   endCursor
   hasNextPage
  } nodes {
   id_lokalId
  }
 }
}
```

Figur 5: Eksempel på GraphQL-query med *first*.

Tjenesten returnerer følgende resultat:

```
{
 "data": {
  "BBR_Bygning": {
   "pageInfo": {
    "endCursor": "1rsNAAAAAAA=",
    "hasNextPage": true
   },
   "nodes": [
    {
     "id_lokalId": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
    },
    {
     "id_lokalId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"
    },
    {
     "id_lokalId": "cccccccc-cccc-cccc-cccc-cccccccccccc"
    },
    {
     "id_lokalId": "dddddddd-dddd-dddd-dddd-dddddddddddd"
    },
    {
     "id_lokalId": "eeeeeeee-eeee-eeee-eeee-eeeeeeeeeeee"
    }
   ]
  }
 }
}
```

Figur 6: Resultat fra query'en i Figur 5.

Fra hasNextPage i resultatet ovenfor kan man se at der er flere sider, da den er sat til true. For at hente den næste side kan anvenderen derfor bruge endCursor som *after*-værdi i den efterfølgende POST request:

```
query {
 BBR_Bygning(
  first: 5
  after: "1rsNAAAAAAA="
  registreringstid: "2025-03-23T14:41:33Z"
  virkningstid: "2025-03-23T14:41:33Z"
  ) {
  pageInfo {
   endCursor
   hasNextPage
  } nodes {
   id_lokalId
  }
 }
}
```

Figur 7: Eksempel på en GraphQL-query med *first* og *after*.

Ved iterativt at gentage dette mønster kan alt den ønskede data hentes.

2.8 GraphQL-skemaer

For alle de registre der er udstillet gennem Datafordelerens GraphQL-tjenester, kan du hente GraphQL-skemaer, der beskriver hvordan data er struktureret. Skemaerne specificerer de typer data, der kan hentes, relationerne mellem disse typer og de filtreringsoperationer, der kan understøttes for de pågældende felter. Du kan benytte skemaerne til at se hvilke queries du kan sende, og hvilke data du kan hente.

Skemaerne genereres på baggrund af registrenes datamodeller og er beriget med metadata for hvert register. Denne metadataudtrækningsproces involverer at tage relevant information fra datamodellerne og inkorporere den i skemaerne. Denne ekstra metadata giver anvendere mere omfattende indsigt i de tilgængelige data. Skemaerne findes separat for hvert register og kan hentes ved at tilgå nedenstående url:

https://graphql.datafordeler.dk/<register>/<version>/schema

<register> er forkortelsen for det register anvenderen forsøger at hente og <version> er versionen for det pågældende register. Læs mere om forkortelser i afsnit 2.3 og versionering i afsnit 2.9. Skemaet for version 1 af Skatteforvaltningens Virksomhedsregister (SVR) hentes for eksempel på følgende url:

https://graphql.datafordeler.dk/SVR/v1/schema

2.9 Versionering

I GraphQL-tjenesterne versioneres hvert register separat, og alle entiteter, der tilhører det samme register, versioneres under det samme versionsnummer som registret. Det vil sige, at DAR og BBR kan eksistere separat som DAR/v2 og BBR/v5. Men hvis BBR har en change til Bygning-entiteten i deres datamodel, vil alle entiteter i BBR blive opgraderet til BBR/v6, og BBR/v5 vil derefter efter en overgangsperiode blive udfaset.

2.10 Cost-beregning

Som en sikkerhedsforanstaltning og af performance-hensyn tilskrives hver query en cost. En cost er en beregnet talværdi der indfanger hvor kompleks og omkostningstung en given query er for Datafordeleren at processere. Cost udregnes inden en query processeres. Der håndhæves aktuelt ikke en grænse på cost før en query processeres af Datafordeleren. Det er dog en mulighed der kan finde anvendelse hvis der skulle opstå et behov. Såfremt dette behov skulle opstå, vil det blive kommunikeret inden der etableres en max cost på GraphQL tjenesterne.

Udregningen af cost er blandt andet baseret på følgende specifikationer:

- **Sidestørrelse**: Størrelse på paging-siderne der returneres. Jo større sider, jo mere data returneres, hvilket giver en højere cost.
- **Filtre**: Antallet og type af filtre der benyttes i en query påvirker cost. Jo flere filtre, jo højere cost. Derudover er geometri-filtre mere omkostningstunge og resulterer derfor i en højere cost. Se evt. afsnit 2.4, 2.5 og 2.6 for en oversigt over de forskellige typer af filtre.
- **Entiteter**: Almindelige og store entiteter har forskellige omkostninger associeret med dem. Store entiteter er mere omkostningstunge end mindre entiteter.
- **Felter**: Forskellige typer felter har forskellige omkostninger. Geometrier og sammensatte[1] felter er dyrere end simple felter.

Bemærk

Der håndhæves aktuelt ikke en grænse på cost før en query processeres af Datafordeleren.

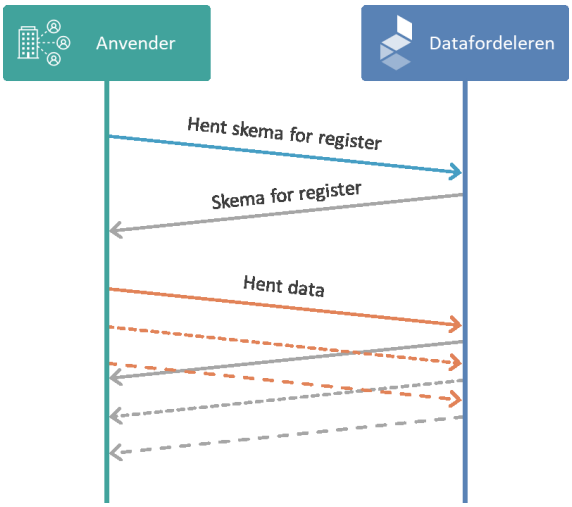
[1] Et sammensat felt er et felt der indeholder andre felter.

Det er dog en mulighed der kan finde anvendelse hvis der skulle opstå et behov.

Såfremt dette behov skulle opstå, vil det blive kommunikeret inden der etableres en max cost på GraphQL tjenesterne.

2.11 Sådan anvender du entitetsbaserede GraphQL-tjenester

GraphQL-tjenesterne på Datafordeleren tilgås via et GraphQL-API. Dette afsnit beskriver, hvordan det vil være muligt at danne et overblik over hvilke entiteter du kan tilgå, samt hvordan data hentes. GraphQL-tjenesterne kan tilgås på registerbasis og kræver at du benytter en af de tre autentifikationsmetoder beskrevet i afsnit 3.



Figur 8: Overblik over samlet anvendelsesmønster for GraphQL-tjenesterne.

Ovenstående figur viser hvordan GraphQL-API'et overordnet set anvendes. Her hentes GraphQL-skemaet først for det pågældende register ved at tilgå https://graphql.datafordeler.dk/<register>/<version>/schema, hvorefter det ønskede data hentes, ved at sende en GraphQL-query til https://graphql.datafordeler.dk/<register>/<version>.

2.11.1.1 Hent Schema

Hent Schema	
URL	https://graphql.datafordeler.dk/<register>/<version>/schema
HTTP-metode	GET
Headere i forespørgsel	Content-Type: application/json
Returværdier	• Returnerer HTTP 200 – OK, ved succes. • Returnerer HTTP 400 – Bad Request, ved angivelse af forkerte parametre. • Returnerer HTTP 404 – Not Found, hvis den angivne ressource ikke kan findes. • Returnerer HTTP 500 – Internal Server Error, hvis der er sket en ukendt fejl
Adgang	IT-system med API-nøgle, OAuth Shared Secret eller OAuth Certifikat.

2.11.1.1.1 Parametre

Navn	**Type**	**Beskrivelse**	**Obligatorisk?**
Register	String	Angiver hvilket register det pågældende skema skal hentes for. Følgende registre kan angives:	Ja

		• DAR • DAGI • CPR • BBR • DHMOprindelse • DHMHoejdekurver • MAT • EBR • FIKSPUNKT • DS • GEODKV • CVR • CVR/custom • EJF • SVR • VUR • HISTKORT	
Version	String	Angiver hvilken version af datamodellen det pågældende register skal være.	Ja

2.11.1.1.2 Returværdier

Denne tjeneste returnerer altid en fil ved HTTP 200 – OK. Filen indeholder et GraphQL-schema med metadata om GraphQL-tjenesten, der beskriver hvilke entiteter der findes for det pågældende register og hvordan data fra disse hentes.

2.11.1.1.3 Eksempler på brug af tjenesten

Hent skema for DAGI med versionsnummer v2 ved brug API-nøgle:

- https://graphql.datafordeler.dk/DAGI/v2/schema?apiKey=placeholderNoegle

2.11.1.2 Send query

Send query	
URL	https://graphql.datafordeler.dk/<register>/<version>
HTTP-metode	GET & POST
Headere i forespørgsel	For POST-requests: • Content-Type: application/json For GET- og POST-requests (valgfrit)[2]: • Accept: application/graphql-response+json
Body	For GET-requests: • Ingen. Query-parametre skal URL-encodes i overensstemmelse med GraphQL-standarden. For POST-requests: • Valid GraphQL-query.
Returværdier[3]	• Returnerer HTTP 200 – OK, ved succes. • Returnerer HTTP 400 – Bad Request, ved angivelse af forkerte parametre. • Returnerer HTTP 404 – Not Found, hvis den angivne ressource ikke kan findes. • Returnerer HTTP 500 – Internal Server Error, hvis der er sket en ukendt fejl
Adgang	• Ikke-adgangsbegrænset data: IT-system med API-nøgle, OAuth Shared Secret eller OAuth Certifikat. • Adgangsbegrænset data: IT-system med OAuth Shared Secret eller OAuth Certifikat *og* på en defineret IP-Allowlist og så skal IT-systemet have eksplicit godkendelse fra det pågældende register. Se afsnit 3 for yderligere information.

2.11.1.2.1 Parametre

Navn	Type	Beskrivelse	Obligatorisk?
Register	String	Angiver hvilket register det pågældende skema skal hentes for. Følgende registre kan angives: • DAR • DAGI • CPR • BBR • DHMOprindelse • DHMHoejdekurver • MAT • EBR • FIKSPUNKT • DS • GEODKV • CVR • CVR/custom • EJF • SVR • VUR • HISTKORT	Ja
Version	String	Angiver hvilken version af datamodellen det pågældende register skal være.	Ja

2.11.1.2.2 Returværdier

Denne tjeneste returnerer et json-response ved HTTP 200 – OK.

[2] DU *bør* inkludere Accept-headeren som specificeret her, hvis du vil opt-in til moderne GraphQL-over-HTTP. Dette er dog ikke et krav. Hvis Accept-headeren ikke inkluderes fås andre HTTP-statuskoder.

[3] Returværdierne for HTTP-statuskoderne afhænger af hvilken Accept-header der specificeres. Disse værdier er hvis Accept-headeren ikke er angivet som beskrevet i "Headere i forespørgsel".

2.11.1.2.3 Eksempler på brug af tjenesten

Hent de første 100 datapunkter fra entiteten DAR_Adresse fra version 1 af DAR inden for den angivet periode ved at sende nedenstående query som en POST-request ved brug af API-nøgle:

- https://graphql.datafordeler.dk/DAR/v1?apiKey=placeholderNoegle

```
query {
 DAR_Adresse(
  first: 100
  registreringstid: "2025-03-23T14:41:33Z"
  virkningstid: "2025-03-23T14:41:33Z"
  ) {
  pageInfo {
   endCursor
   hasNextPage
  } nodes {
   adressebetegnelse
   bygning
   id_lokalId
   registreringFra
   registreringTil
   status
   virkningFra
   virkningTil
  }
 }
}
```

Hent data der var gældende den 12/11/2024, kl. 14:41:33 med kommunekode “0550” fra entiteten BBR_Bygning fra version 1 af BBR ved at sende nedenstående query som en POST-request ved brug af API-nøgle:

- https://graphql.datafordeler.dk/BBR/v1/?apiKey=placeholderNoegle

```
query {
 BBR_Bygning(
  virkningstid: "2024-11-12T14:41:33Z"  #
  where: {
   kommunekode: { eq: "0550" }
  }
 ) {
  pageInfo {
   endCursor
   hasNextPage
  } nodes {
   husnummer
   id_lokalId
   status
  }
 }
}
```

2.11.2 Samlet gennemgang af anvendelse

Dette afsnit gennemgår et længere eksempel på, hvordan en anvender kan sammensætte flere af elementerne i denne vejledning til at hente data ud fra Datafordeleren ved at bruge de entitetsbaserede GraphQL-tjenester. Eksemplet tager udgangspunkt i at anvenderen kender til Grunddata generelt, har læst vejledningen igennem og orienteret sig omkring Datafordelerens hjemmeside, samt orienteret sig omkring den dokumentation som kan findes på Datafordelerens Dokumentation.

Eksemplet fokuserer på brugen af GraphQL på Datafordeleren og går dermed ikke i dybden angående indhentning af rettigheder til data på Datafordeleren.

2.11.2.1.1 Sammenhængende data mellem entiteterne BBR_Bygning og DAR_Husnummer

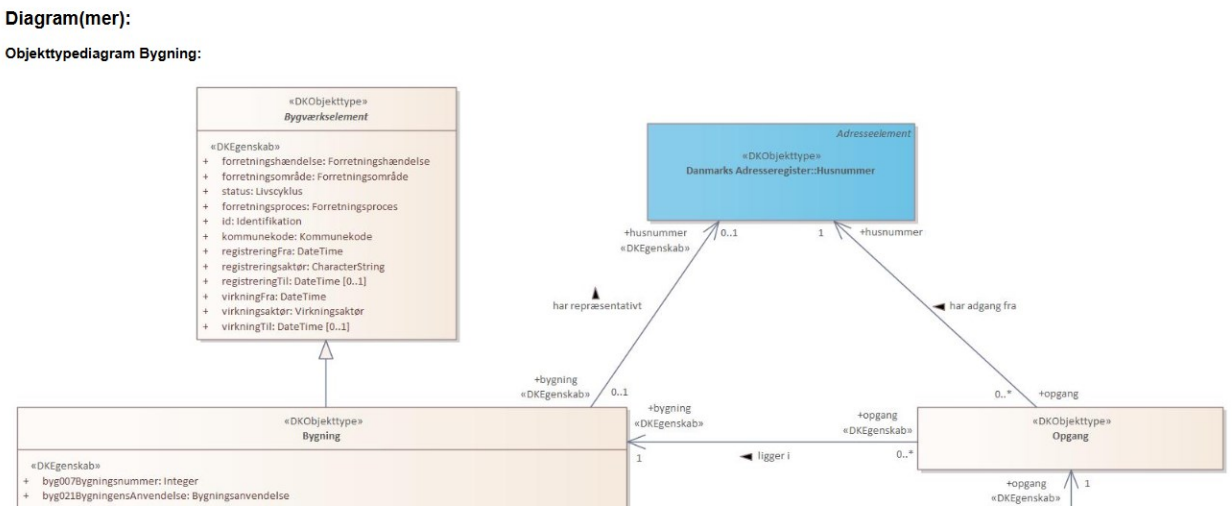
Eksemplet tager udgangspunkt i en anvender, som ønsker at få sammenhængende data mellem BBRs Bygning entitet og DARs Husnummer entitet. For at finde frem til hvilke data der er relevante for anvender, fremsøges relevante oplysninger, metadata, mm. ved de kilder som stilles til rådighed af både Datafordeleren og registrene. Følgende kilder er som udgangspunkt relevante i de fleste tilfælde:

1. Datafordelerens dokumentation: https://datafordeler.dk/vejledning/.
2. Datafordelerens dataoversigt: https://datafordeler.dk/dataoversigt/.
3. Grunddatamodellen, som beskriver registrenes data:
    a. https://grunddatamodel.datafordeler.dk/objekttypekatalog/.
    b. https://datafordeler.dk/vejledning/grunddata/datamodel/.
4. GraphQL-skemaerne, som beskriver de faktiske data der kan forespørges på, samt hvilke muligheder en anvender har når der forespørges for det enkelte register..
5. Datafordelerens dokumentation på Confluence: https://confluence.sdfi.dk/display/DML/Datafordelerens+dokumentation.

Det fremgår i Grunddatamodellen at BBR og DAR har en relation mellem deres entiteter blandt andet på https://datafordeler.dk/vejledning/grunddata/datamodel/. Det kan her ses at BBR hænger sammen med data i MAT, DAR og GeoDK.

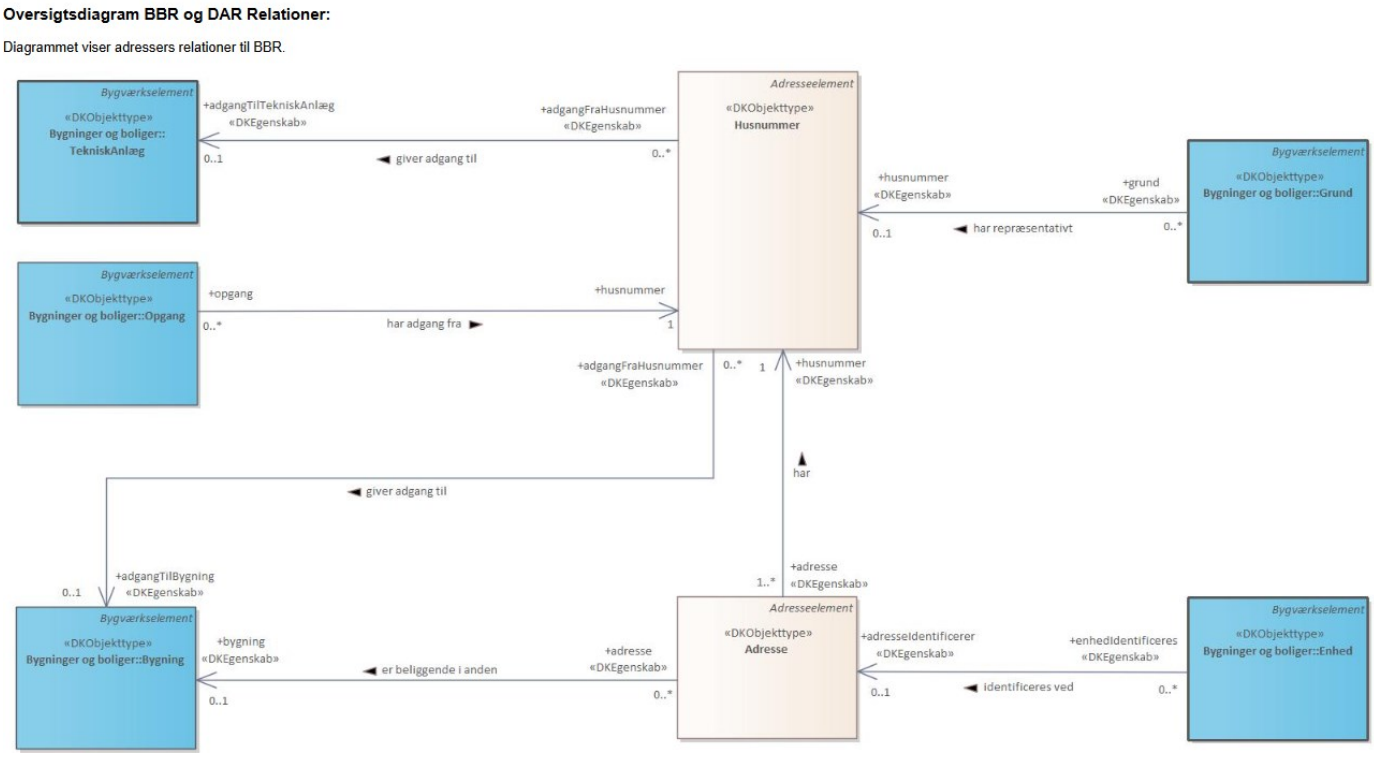
Figur 9: Relationer mellem registerdata.

Når BBR undersøges nærmere i Grunddatamodellen i Figur 10, fremgår det af registerets datamodel, at relationen findes mellem BBRs Bygning entitet og DARs Husnummer entitet.



Figur 10: Relationer mellem data i BBR.

Undersøges DARs Husnummer entitet nærmere, bekræfter registerets datamodel også at der er en relation mellem de to entiteter, som kan ses i Figur 11.



Figur 11: Relationer mellem data i DAR.

Da det nu er klart for anvender hvilke data der ønskes, samt at data mellem BBR og DAR har en relation, hentes nu de relevante GraphQL-skemaer for BBR og DAR. Dette gøres ved følgende queries:

- GET request til BBR v1 – https://graphql.datafordeler.dk/BBR/v1/schema?apiKey=placeholderNoegle. I dette skema kan BBRs Bygning entitet fremsøges ved "BBR_Bygning". Et uddrag af skemafilen kan ses nedenfor.

```
type BBR_Bygning {
  """
  Grunddatamodel info:
  type: Integer
  multiplicity: 1
  URI: https://data.gov.dk/model/profile/bbr/byg007Bygningsnummer
  prefLabel (da): Bygningsnummer
  definition (da): angiver bygningens nummer indenfor ejendommen
  legalSource: https://www.retsinformation.dk/eli/lta/2019/797
  source: https://instruks.bbr.dk/instruks/0/30
  """
  byg007Bygningsnummer: Long @cost(weight: "1")
  """
  grund: String @cost(weight: "1")
  husnummer: String @cost(weight: "1")
  id_lokalId: String! @cost(weight: "1")
}
```

- GET request til DAR v1 - https://graphql.datafordeler.dk/DAR/v1/schema?apiKey=placeholderNoegle. I dette skema kan DARs Husnummer entitet fremsøges ved "DAR_Husnummer". Et uddrag af skemafilen kan ses nedenfor.

```
type DAR_Husnummer {
  """
  Grunddatamodel info:
  type: CharacterString
  multiplicity: 0..1
  URI: https://data.gov.dk/model/profile/dar/husnummertekst
  prefLabel (da): husnummertekst
  definition (da): husnummeret til adressen inklusive evt. bogstav
  legalSource: https://www.retsinformation.dk/eli/lta/2018/271#P16
  """
  husnummertekst: String @cost(weight: "1")
  id_lokalId: String @cost(weight: "1")
}
```

Anvenderen kan med fordel her vælge at indlæse skemafilerne i et program, som kan læse GraphQL-skemafiler for at gøre det nemmere at udlede hvilke entiteter der kan forespørges på, samt hvilke felter og filtre der er understøttet for hver entitet og felt.

Det kan i skemaet ses at BBR_Bygning har et felt ved navn "husnummer", som kan bruges til at sammensætte entitetens data med et tilhørende husnummer ovre i DAR. Anvenderen ved også at Grunddataregistre som hovedregel bruger "id_lokalId" til identifikation af objekter over tid, således at det er muligt at fremfinde et objekt, sammen med eventuel historiske versioner af denne. I skemaerne fremsøger anvender de query-muligheder der er på BBR_Bygning og DAR_Husnummer.

Anvenderen kan udlede følgende fra skemaet:

- Parametrene "first", "after", "registreringstid", "virkningstid" og "where" kan angives.
- Når "where" angives, er det påkrævet at bruge henholdsvis et "BBR_BygningFilterInput" eller "DAR_HusnummerFilterInput". Det kan udledes i skemaet hvilke filtreringsmuligheder hver af disse filter inputs giver.

Anvenderen kender nu til de filtreringsmuligheder der er tilgængelige på BBR_Bygning og DAR_Husnummer. Anvenderen er nu klar til at sammensætte sine GraphQL-queries og hente relevante data ned.

Anvenderen laver nedenstående GraphQL-query på denne URL som en POST request: https://graphql.datafordeler.dk/BBR/v1?apiKey=placeholderNoegle.

```
query {
 BBR_Bygning(
  first: 5
  registreringstid: "2025-03-23T12:00:00Z"
  virkningstid: "2025-03-23T12:00:00Z"
  where: {
   kommunekode: { eq: "0461" }
  }
 ) {
   pageInfo {
   endCursor
   hasNextPage
  }
  nodes {
   kommunekode
   husnummer
   id_lokalId
  }
 }
}
```

Ovenstående GraphQL-query gør følgende:

- Henter data fra BBR_Bygning entiteten, som er angivet i BBR_Bygning.
- Henter pagings metadata som gør det muligt for anvender at page på dataen.
- Henter de første 5 resultater ved brug af “first” parameteret.
- Filtrerer på data således at det kun er data som er bitemporalt gældende d. 23/3/2025, ved at angive registreringstid og virkningstid parametrene.
- Filtrerer data således at det kun er resultater der er beliggende i Odense kommune som returneres, ved at angive “where” parameter med filter på kommunekode.

Fra GraphQL-querien fås følgende svar:

```
{
 "data": {
  "BBR_Bygning": {
   "pageInfo": {
    "endCursor": "eTBZpwAAAAA=",
    "hasNextPage": true,
   },
   "nodes": [
    {
     "kommunekode": "0461",
     "husnummer": "0a3f5089-71a3-32b8-e044-0003ba298018",
     "id_lokalId": "56a391dc-c908-48ef-ac25-00013af345f5"
    },
    {
     "kommunekode": "0461",
     "husnummer": "0a3f5089-d099-32b8-e044-0003ba298018",
     "id_lokalId": "68979e1f-194d-44c7-b4fa-0001a785ad1e"
    },
    {
     "kommunekode": "0461",
     "husnummer": "0a3f5089-d279-32b8-e044-0003ba298018",
     "id_lokalId": "7508a22e-e0e8-4392-860f-0003c5df60b9"
    },
    {
     "kommunekode": "0461",
     "husnummer": "362e6b23-5084-0f9b-e044-0003ba298018",
     "id_lokalId": "f02e1eb7-fac0-417c-a294-0004a401088b"
    },
    {
     "kommunekode": "0461",
     "husnummer": "0a3f5089-5f64-32b8-e044-0003ba298018",
     "id_lokalId": "e1ca316a-4ce8-4181-8987-0004b604ad22"
    }
   ]
  }
 }
}
```


Svaret indeholder de 3 felter der blev spurgt om i forespørgslen for hver af de 5 resultater. Det bemærkes yderligere at der i svaret også fremgår en ”endCursor”. Anvenderen bruger nu værdien på ”endCursor” i sit næste POST kald til BBR_Bygning ved nu også at angive ”after” parameter:

```
query {
 BBR_Bygning(
  first: 5
  after: "eTBZpwAAAAA="
  registreringstid: "2025-03-23T12:00:00Z"
  virkningstid: "2025-03-23T12:00:00Z"
  where: {
   kommunekode: { eq: "0461" }
  }
 ) {
   pageInfo {
   endCursor
   hasNextPage
  }
  nodes {
   kommunekode
   husnummer
   id_lokalId
  }
 }
}
```

Fra ovenstående query får anvenderen et svar tilbage der har samme struktur, dog med yderligere data. Anvenderen kan blive ved på denne måde, indtil den ønskede data er hentet. Anvenderen ved at alle resultater er hentet når feltet "hasNextPage" er sat til "false".

Nu har anvenderen alle ønskede data fra BBR_Bygning, og kan kalde DAR_Husnummer for at hente de husnumre der hænger sammen med de bygningsobjekter der blev returneret fra BBR_Bygning. Dette gør anvenderen ved at lave en GraphQL-query til DAR_Husnummer som en POST request til denne URL: https://graphql.datafordeler.dk/DAR/v1?apiKey=placeholderNoegle.

```
query {
 DAR_Husnummer(
  first: 5
  registreringstid: "2025-03-23T12:00:00Z"
  virkningstid: "2025-03-23T12:00:00Z"
  where: {
   id_lokalId: {in: ["0a3f5089-71a3-32b8-e044-0003ba298018",
                     "362e6b23-5084-0f9b-e044-0003ba298018"]}
  }
 ) {
   pageInfo {
   endCursor
   hasNextPage
  }
  nodes {
   id_lokalId
   adgangsadressebetegnelse
   status
   adgangTilBygning
  }
 }
}
```

Ovenstående GraphQL-query gør følgende:

- Henter data fra DAR_Husnummer entiteten, som er angivet i DAR_Husnummer.
- Henter pagings metadata som gør det muligt for anvender at page på dataen.

- Henter de første 5 resultater ved brug af "first" parameteret.
- Filtrerer på data således at det kun er data som er bitemporalt gældende d. 23/3/2025, ved at angive registreringstid og virkningstid parametrene. Bemærk at identiske bitemporale filtre anvendes i querien til DAR_Husnummer, som der blev anvendt til filtreringen på BBR_Bygning. På denne måde sikres det at det er sammenhængende data, som er gældende på samme tidspunkt, som returneres.
- Filtrerer data således at det kun er resultater som er relaterede til de bygninger som blev returneret i den første query til BBR_Bygning.

Svaret på ovenstående GraphQL-query ser ud som følger:


```
{
  "data": {
    "DAR_Husnummer": {
      "pageInfo": {
        "endCursor": "HPFo5QAAAAA=",
        "hasNextPage": false,
      },
      "nodes": [
        {
          "id_lokalId": "0a3f5089-71a3-32b8-e044-0003ba298018",
          "adgangsadressebetegnelse": "Lille Rugbjerg Vej 85, Anderup, 5270 Odense N",
          "status": "3",
          "adgangTilBygning": "56a391dc-c908-48ef-ac25-00013af345f5"
        },
        {
          "id_lokalId": "362e6b23-5084-0f9b-e044-0003ba298018",
          "adgangsadressebetegnelse": "Skipperkrogen 13, Stige, 5270 Odense N",
          "status": "3",
          "adgangTilBygning": "f02e1eb7-fac0-417c-a294-0004a401088b"
        }
      ]
    }
  }
}
```


Anvender har nu hentet data fra BBR_Bygning og relateret data fra DAR_Husnummer.

2.12 GraphQL Custom entiteter og tjenester

For at imødekomme specielle forretningsmæssige behov for enkelte registre udstilles der en række specialudviklede entiteter og tjenester der kan søges adgang til gennem Datafordeler Administration.

2.12.1 CPR Private Sector

Hvis du ønsker at tilgå CPR-data hos Datafordeleren via GraphQL på vegne af en privat virksomhed, skal du først have adgang til CPR Private Sector-tjenesten. Tjenesten hedder CustomPrivateSectorPerson og adgang søges på samme måde som for andre registre med beskyttet data hos Datafordeleren. Tjenesten vælges i drop-down-menuen "Vælg entiteter og tjenester" når CPR-registeret er valgt i "Register"-menuen i en almindelig "Dataadgang"-ansøgning. "Dataadgang"-ansøgninger er beskrevet yderligere i Brugeroprettelse Datafordeler Administration.

2.12.1.1 Omfang af tjenesten

Med godkendt adgang til CPR Private Sector-tjenesten, har du mulighed for at tilgå entiteten CPRCustom_PrivateSectorPerson via GraphQL. Denne specialudviklede entitet giver mulighed for at tilgå en række af de almindelige entiteter fra CPR-registret og sammenstiller disse, så du ikke selv behøver at tage stilling til hvordan data hænger sammen.

Bemærk at der er en række begrænsninger på hvilke data der kan returneres, som følge af CPR-lovgivningen. Tjenesten udstiller for eksempel udelukkende aktuelle data og der kan ikke hentes data på personer, der har status = 'nedlagt'. Derudover vil adresser, udrejse/indrejse og navn, i tilfælde hvor den pågældende person har navne- og adressebeskyttelse, ikke blive inkluderet i outputtet. Generelt gælder der de samme regler for udstilling af CPR-data til private anvendere gennem CPR Private Sector som for Datafordelerens nuværende REST-tjenester for private. Du kan læse mere om disse regler på siden *REST - private (CPR)* i Dokumentation for Datafordeleren - Det Centrale Personregister (CPR).

2.12.1.2 Opbygning af kald via tjenesten

For at bruge CPR Private Sector-tjenesten skal du give en række input der unikt identificerer en person. Dvs. du kan udelukkende hente data på én person ad gangen og denne person skal kunne identificeres unikt ud fra en række inputparametre. Tjenesten tager følgende inputparametre:

- pnr: Personnummeret på en person. Parameteren kan ikke bruges sammen med parametrene navn, adresse eller foedselsdato. Kan godt bruges i kombination med parameteren haendelse.
- navn: Navnene på en person. Hvis navn benyttes, skal du *som minimum* angive efternavn sammen med enten fornavn eller mellemnavn. Fornavn, mellemnavn og efternavn kan godt alle bruges samtidig.
- adresse: Adressen på en person. Adressen består blandt andet af daradresse, som id'et på den pågældende adresse i Danmarks Adresseregister (DAR). daradresse kan *ikke* bruges i kombination med nogen af de øvrige adresseparametre. Hvis *ikke* daradresse angives, *skal* der angives mindst én fra hver af de følgende punkter:
    - kommunekode, postnummer eller postdistrikt
    - vejkode, vejnavn eller vejadresseringsnavn
    - bygningsnummer eller husnummer
- foedselsdato: Den pågældende persons fødselsdato.
- haendelse: Hændelser for en person. Kan bruges i kombination med alle de andre parametre. Skal dog overholde følgende regler:
    - Hvis dato angives, skal både til- og fra-parametrene angives. Fra-parameteren må ikke være efter til-parameteren og hverken til- eller fra-parameteren må være i fremtiden.
    - Du har kun lov til at hente hændelser der er sket inden for de seneste 7 dage.
    - Parameteren haen kan kun tage følgende værdier: "ADRESSEOPLYSNINGER_OG_BESKYTTELSE", "CPRADRESSE", "KONTAKTADRESSE", "NAVN_OG_BESKYTTELSE", "UDREJSEINDREJSE", "VAERGEMAAL"
    - Parameteren haenkode kan kun tage følgende værdier: "A01", "A02", "A03", "A04", "A05", "A06", "A08", "A09", "A10", "A13", "A14", "A15", "A16", "A17", "A19", "A20", "A21", "A22", "A23", "A24", "A25", "A33", "A34", "A39", "A40", "A42", "A45", "A47", "A73", "A74", "K74", "P02", "P03", "P10", "P11", "P30", "P31", "P42", "P43", "P45"

Du kan identificere en person på en af tre følgende måder:

- Personnummer: Du kan fremsøge en person ved brug af personnummer (pnr). Hvis inputparameteren pnr kan ikke bruges sammen med andre inputparametre.
- Navn og adresse: Du kan fremsøge en person ved at kombinere navn med adresse
- Navn og fødselsdato: Du kan fremsøge en person ved at kombinere navn og fødselsdato.

Udover brug af inputparametre, kalder du CPR Private Sector på nøjagtig samme måde som de almindelige GraphQL-tjenester. Hvis du for eksempel ønsker at hente adresseoplysninger for en bestemt person, kan du sende nedenstående GraphQL-query som en POST-request til https://graphql.datafordeler.dk/CPR/custom/PrivateSector/v1:

```
query {
  CPRCustom_PrivateSectorPerson(
    input: {
      navn: { fornavne: "John", efternavn: "Doe" },
      adresse: {
        postnr: 1550,
        vejnavn: "Rådhuspladsen",
        husnr: "1"
      }
    }
  ) {
    navn {
      fornavne
      mellemnavn
      efternavn
    }
    adresseoplysninger {
      cprAdresse {
        daradresse
        bygningsnummer
        bynavn
        cprkommunekode
        cprvejkode
        etage
        husnummer
        postdistrikt
        postnummer
        sidedoer
        vejadresseringsnavn
        vejnavn
      }
      status
      virkningfra
      virkningfrausikkerhedsmarkering
    }
    beskyttelser {
      beskyttelsestype
      status
      virkningfra
      virkningtil
    }
  }
}
```

Bemærk at beskyttelser-feltet *altid* skal inkluderes i en query og alle kald til CPR Private Sector-tjenesten uden beskyttelser inkluderet i query'en vil blive afvist og intet data returneret.

2.12.2 CPR Public Sector

Hvis du ønsker at tilgå CPR-data hos Datafordeleren via GraphQL på vegne af en offentlig myndighed, skal du først have adgang til CPR Public Sector-tjenesten. Tjenesten hedder CustomPublicSectorPerson og adgang søges på samme måde som for andre registre med beskyttet data hos Datafordeleren. Tjenesten vælges blot i drop-down-menuen "Vælg entiteter og tjenester" når CPR-registeret er valgt i "Register"-menuen i en"Dataadgang"-ansøgning. "Dataadgang"-ansøgninger er beskrevet yderligere i Brugeroprettelse Datafordeler Administration.

2.12.2.1 Omfang af tjenesten

Med godkendt adgang til CPR Public Sector-tjenesten, har du mulighed for at tilgå entiteten CPRCustom_PublicSectorPerson via GraphQL. Denne specialudviklede entitet giver mulighed for at tilgå en række af de almindelige entiteter fra CPR-registret og sammenstille disse, så du ikke selv behøver at tage stilling til hvordan data hænger sammen.

Bemærk at du med denne tjeneste har mulighed for at hente alle data som CPR har tilgængeligt og der derfor er et dataminimeringsansvar, som ligger hos anvenderen, i kun at hente den data der er nødvendigt for at udføre sin funktion i den tilknyttede myndighed.

2.12.2.2 Opbygning af kald via tjenesten

Ligesom i CPR Private Sector-tjenesten kan CPR Public Sector tage en række inputparametre. Hvilke inputparametre og datafiltreringer og hvordan disse benyttes, fremgår af det tilhørende GraphQL-skema. I modsætning til den private tjeneste, er der ingen af parametrene der er obligatoriske, da du behøver ikke unikt at kunne identificere en person for at fremsøge informationer om vedkommende og tjenesten kan derfor returnere flere personer samtidig. Du kan for eksempel fremsøge adresseoplysninger for flere personer der hedder "John Doe" samtidig ved at sende nedenstående GraphQL-query, som en POST-request til https://graphql.datafordeler.dk/CPR/custom/PublicSector/v1:

```
query {
  CPRCustom PublicSectorPerson(
    first: 3
    input: { navn: { fornavne: { eq: "John" }, efternavn: { eq: "Doe" } } }
  ) {
    nodes {
      id
      navne {
        fornavne
        mellemnavn
        efternavn
      }
      adresseoplysninger {
        cprAdresse {
          daradresse
          bygningsnummer
          bynavn
          cprkommunekode
          cprvejkode
          etage
          husnummer
          postdistrikt
          postnummer
          sidedoer
          vejadresseringsnavn
          vejnavn
        }
        status
        virkningfra
        virkningfrausikkerhedsmarkering
      }
      beskyttelser {
        beskyttelsestype
        status
        virkningfra
        virkningtil
      }
    }
  }
}
```

Du kan fremsøge flere personer samtidig, understøtter CPR Public Sector paging af resultaterne, så du ved almindelige query-parametre (se afsnit 2.7 om paging) kan specificere hvor mange personer du ønsker returneret ved at givet kald.

Bemærk at beskyttelser-feltet, ligesom for CPR Private Sector-tjenesten, *altid* skal inkluderes i en query og alle kald til tjenesten uden beskyttelser inkluderet i query'en vil blive afvist og intet data returneret.

Noget nyt sammenlignet med de tidligere CPR-tjenester er, at det nu er muligt at filtrere på "Hændelse" og "Hændelseskoder" for den offentlige sektor. Der er introduceret fire nye parametre: haendelsekode.haendelsekode, haendelsekode.virkningfra, haendelse.forretningshaendelse og haendelse.afleveringdato. En komplet liste over hvilke hændelseskoder og forretningshændelser der findes kan ses på CPRhændelser til Datafordeleren.

2.12.3 EJF

Der er lavet specialudviklede entiteter for EJF, for at imødekomme behov fra anvendere som har behov for EJF-data men ikke har hjemmel til at se de dertilhørende CPR-numre. Disse entiteter er dog ikke inkluderede i de entitets-baserede EJF-tjenester, og er kun tilgængelige i Fleksibel Opslagslogik. Detaljer omkring dette kan derfor læses i Brugervejledning – Transitionsguide for Fleksible Opslagslogik.

2.12.4 CVR Custom

Der er lavet en specialudviklet entitet for CVRs Person-entitet, navngivet CVRPersonBegraenset, for de anvendere som har brug for adgang til denne data, men som ikke har hjemmel til at se personens CPR-nummer.

CVRPersonBegraenset entiteten har sin helt egen tjeneste og tilgås på url'en https://graphql.datafordeler.dk/CVR/custom/v1. Tjenesten kaldes på præcis samme måde som du ville kalde CVRs standard entitets-baserede tjeneste, og har sit eget skema som kan bruges i en GraphQL API Client til at udforme valide forespørgsler.

3 Autentifikation & autorisation for GraphQL-tjenester

Når en anvender ønsker at benytte GraphQL-tjenester kræver det at anvenderen er autentificeret. Datafordeleren har en række forskellige **autentifikationsmetoder**, som skal benyttes afhængigt af om man forsøger at tilgå frit tilgængeligt eller adgangsbegrænset data:

1) IT-system med **API-nøgle**: En API-nøgle er en privat nøgle, som bruges til at autentificere mod et API. Denne nøgle bruges som en del af godkendelsesprocessen, hvor en klient får adgang til et API uden at afsløre brugerens loginoplysninger. Når en API-nøgle er oprettet, bruges API-nøglen som et query-parameter for at tilgå GraphQL-tjenesterne. Bemærk at API-nøgler **ikke** kan bruges til at tilgå adgangsbegrænsede data. En API-nøgle bruges som query-parameter ved at indsætte den i selve URL'en således: https://graphql.datafordeler.dk/<Register>/<Version>?apiKey=placeholderNoegle

2) IT-system med **OAuth Shared Secret**: En OAuth Shared Secret er en privat nøgle, som deles mellem en klient (f.eks. en applikation) og udbyderen (f.eks. en service som Datafordeleren). Denne nøgle bruges som en del af godkendelsesprocessen, hvor en klient får adgang til ressourcer på vegne af en bruger uden at afsløre brugerens loginoplysninger. For at blive autentificeret ved hjælp af OAuth Shared Secret skal du som anvender først oprette en Shared Secret i selvbetjeningsportalen. Når du har oprettet en Shared Secret sendes denne, samt Client ID og grant type til https://auth.datafordeler.dk/realms/distribution/protocol/openid-connect/token i en POST-request hvorefter man vil modtage et Access Token. En mere detaljeret beskrivelse af hvordan man får et Access Token kan findes her. Det resulterende token er et Bearer Token der kan bruges til autentifikation ved efterfølgende kald til datafordeleren. Bemærk at dette token kun er validt i 60 minutter efter udstedelsen.

3) IT-system med **OAuth Certifikat**: Klientautentificering med certifikater i OAuth er en sikkerhedsmetode, hvor en klient (f.eks. en applikation) bruger et digitalt certifikat til at bevise sin identitet over for autorisationsserveren. Processen involverer, at klienten præsenterer sit certifikat under TLS-håndtrykket, hvorefter autorisationsserveren validerer det. Denne metode er særligt velegnet til miljøer med høje sikkerhedskrav eller ved håndtering af følsomme data. OAuth Certifikater kan bruges i stedet for Oauth Shared Secrets og sættes på kaldet som en client assertion. På samme måde som OAuth Shared Secret, skal certifikatet bruges til at indhente et Access Token. Dette gøres ved at Client ID, samt certifikat i en POST-request til https://auth-oces.datafordeler.dk/realms/distribution/protocol/openid-connect/token. En mere detaljeret beskrivelse af hvordan Access Token indhentes via certifikat kan findes her. Bemærk at dette token kun er validt i 60 minutter efter udstedelsen.

Bemærk at det *ikke* er muligt at benytte tjenestebruger til autentifikation for at hente data via GraphQL-tjenesterne.

3.1 Access tokens og Shared Secrets

I OAuth er et Access Token en hemmelig nøgle som er med til at identificere en anvender hos Datafordeleren, uden at kende til brugernavn eller et password. Når du laver en query med et Access Token i en Authorization header med et Bearer Token, kan Datafordeleren finde ud af hvilke rettigheder og roller som du har. En query bliver derfor afvist hvis dit Access Token mangler. Tabel 8 indeholder en beskrivelse af de oplysninger der skal angives for at få udstedt et Access Token.

Client credentials	Beskrivelse	Eksempel på værdi
Client_id	Her skal angives den værdi som fremgår i "Client ID"-kolonnen i overblikket over OAuth Shared Secret.	00001111-aaaa-2222-bbbb-3333cccc4444
Client_secret	Her skal angives værdien som blev vist da Shared Secret blev oprettet i Datafordeler Administration.	qWgdYAmab0YSkuL1qKv5bPX
Grant_type	Skal altid sættes til "client_credentials".	client_credentials

Tabel 8: Overblik over client credentials som skal angives for at få udstedt et access token.

Når du har modtaget et Access Token, skal det angives i en Authorization header i kaldet til tjenesterne. Tabel 9 beskriver hvorledes Access Token angives I kaldet.

HTTP header	Beskrivelse	Eksempel på værdi
Authorization	Her skal angives det Access Token som blev sendt tilbage ved kald til https://auth.datafordeler.dk/realms/distribution/protocol/openid-connect/token. Bemærk at eksemplet er forkortet og at det faktiske Access Token er betydeligt længere.	Bearer eyJhbGciOiJSUzI1NiIsI….

Tabel 9: Overblik over headers som skal angives for at bruge Access Token ved kald til GraphQL-tjenesterne.

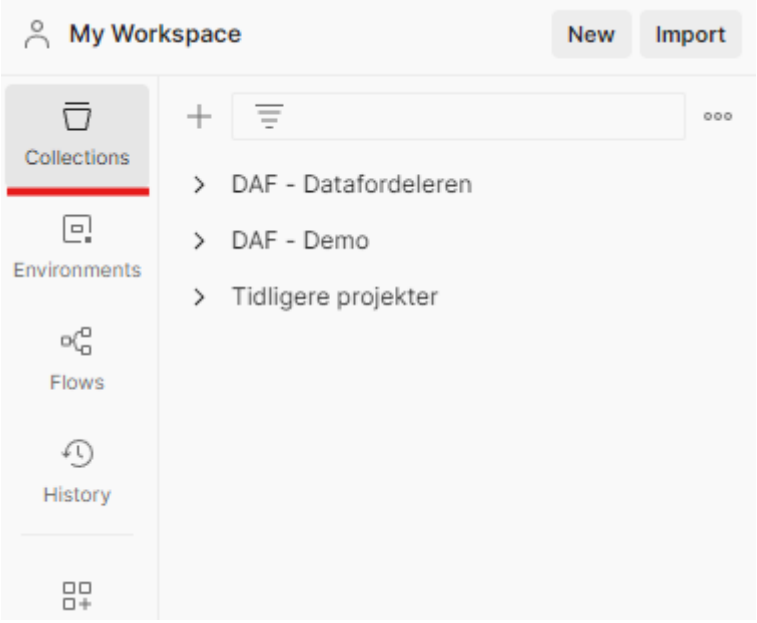
3.2 Eksempel på OAuth opsætning i en GraphQL klient

3.2.1 Opsætning med OAuth Shared Secret

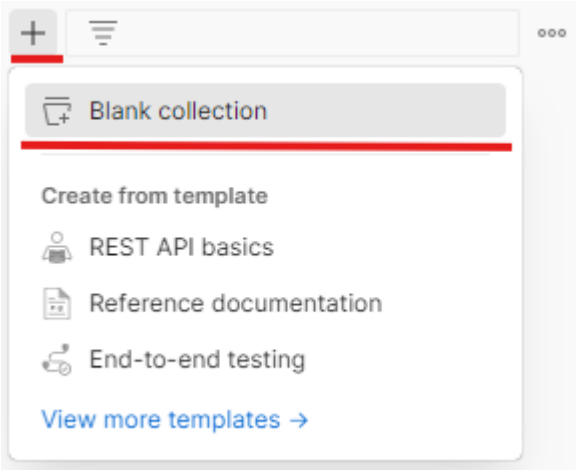
Hvis du som anvender ønsker at bruge Client secret og Client Id til at sende en query til Datafordelerens GraphQL tjeneste kan du med fordel bruge et program som f.eks. Postman, som understøtter både GraphQL-queries og OAuth-opsætning.

Når OAuth credentials er oprettet i portalen, kan du gå i gang med OAuth-opsætningen i Postman. Her er det vigtigt at du noterer dit Client ID og Client secret da disse værdier skal bruges til at hente et Access Token igennem Datafordeleren som blandt andet giver adgang til GraphQL-tjenesten.

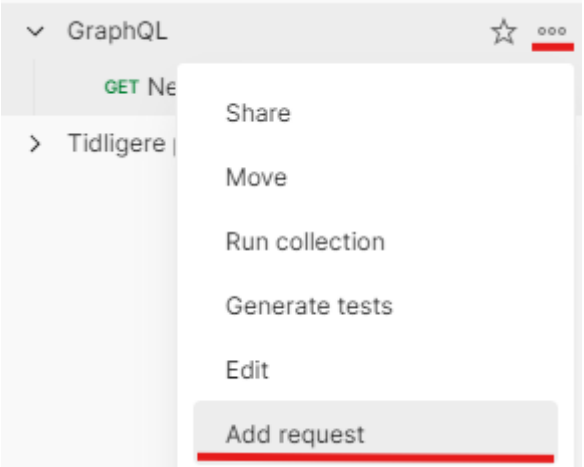
1) Åben Postman og klik på ”Collections” i venstre side. Collections er en folder/mappe med et bestemt navn, hvor du kan samle alle dine queries til Datafordeleren og eventuelt navngive dem efter behov.



For at oprette en collection trykker du på ”+” ikonet og vælger ”Blank collection”. Derefter kan du i toppen navngive din collection.



2) Når en collection er oprettet, kan du oprette en request ved at højreklikke på din collection eller trykke på de 3 prikker og derefter vælge ”Add request”.



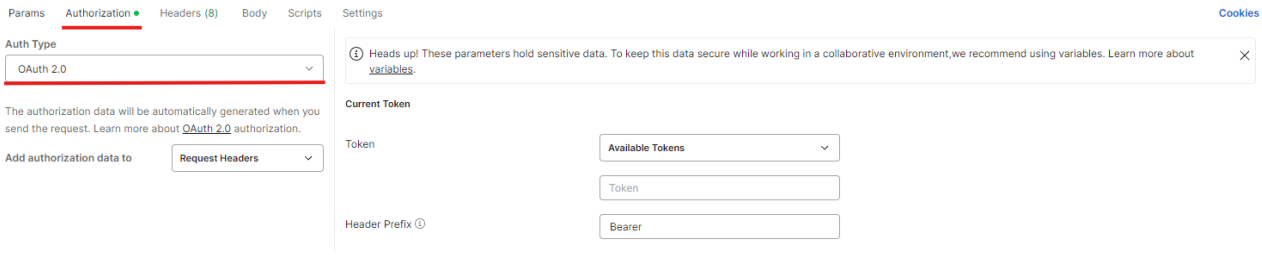
Her kan du også navngive din request efter behov.

3) Når requesten er oprettet kan du i HTTP metoden vælge GET, hvis du vil hente et GraphQL-skema, eller POST hvis du vil sende en query.

   Dette eksempel tager udgangspunkt i en POST request med OAuth-credentials til BBR.

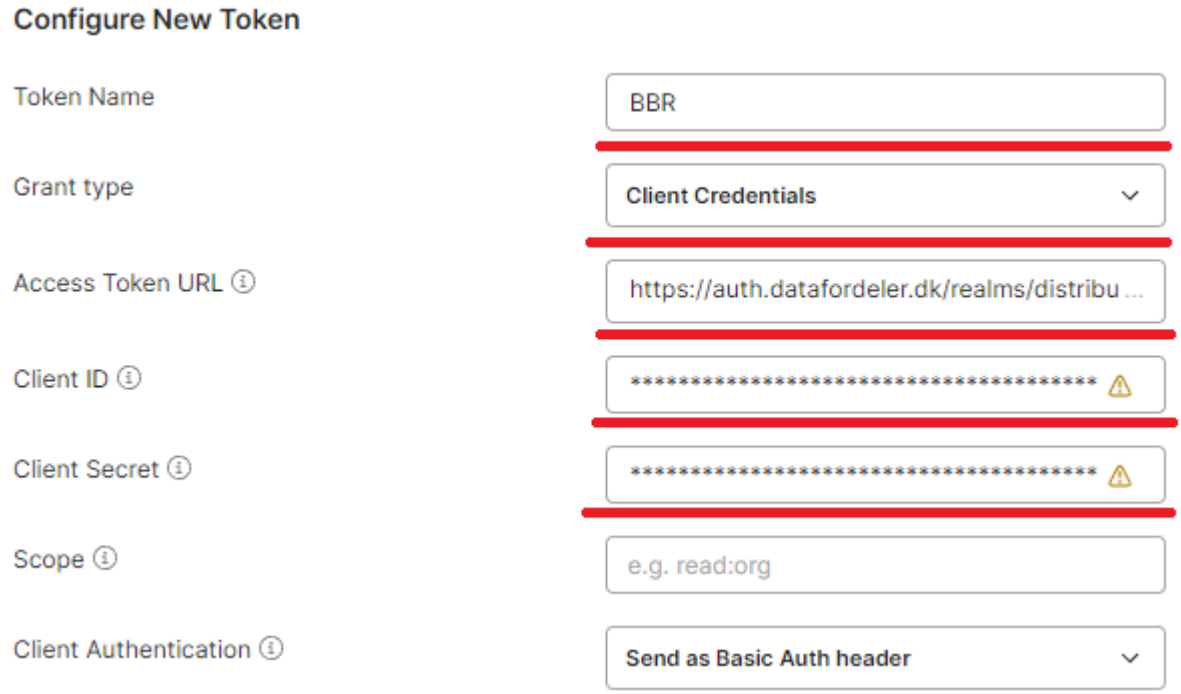
   I feltet til URL'en kan du indsætte https://graphql.datafordeler.dk/BBR/v1 for at kunne lave queries til BBR-version 1.

4) Vælg requesten i venstre side (hvis den ikke allerede er aktiv som i det forrige trin) og klik på "Authorization" for at skifte til en anden fane. Her skal du vælge "OAuth 2.0" i "Auth Type" menuen.



   Postman skulle gerne skifte indholdet når du har valgt en ny auth type.

5) Du skal herefter scrolle ned i den aktive "Authorization" fane indtil du kan se en "Configure New Token" overskriften.



   Under overskriften vil der være en række felter som skal udfyldes som beskrevet herunder.

   a. I "Token Name" kan du give din access token et passende navn.

   b. I "Grant type" menuen skal du vælge "Client Credentials". Her vil Postman ændre felterne under med dem som relevant for din konfiguration af OAuth.

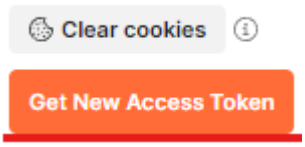
c. I ”Access Token URL” skal du indsætte følgende link - https://auth.datafordeler.dk/realms/distribution/protocol/openid-connect/token

d. I ”Client ID” skal du indsætte det Client ID som du har modtaget efter at have oprettet det i OAuth sektionen på Datafordeler Administration.

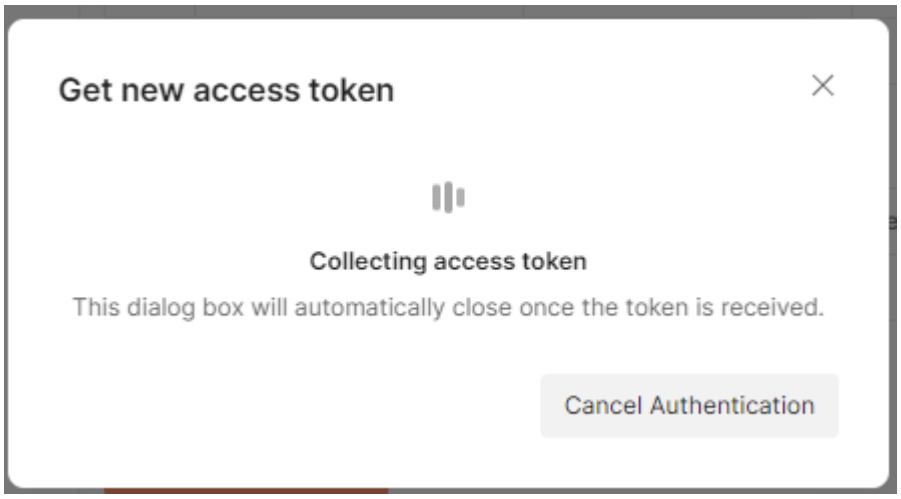
e. I ”Client Secret” skal du indsætte den Client Secret som du har modtaget efter at have oprettet det i OAuth sektionen på Datafordeler Administration.

f. Du kan ignorere ”Scope” feltet, men skal verificere at ”Client Authentication” feltet er sat til ”Send as Basic Auth header”.

6) Efter du har udfyldt de nødvendige felter, kan du hente deres Access Token ved at rulle ned indtil du ser en orange knap der hedder ”Get New Access Token” og klikke på den.

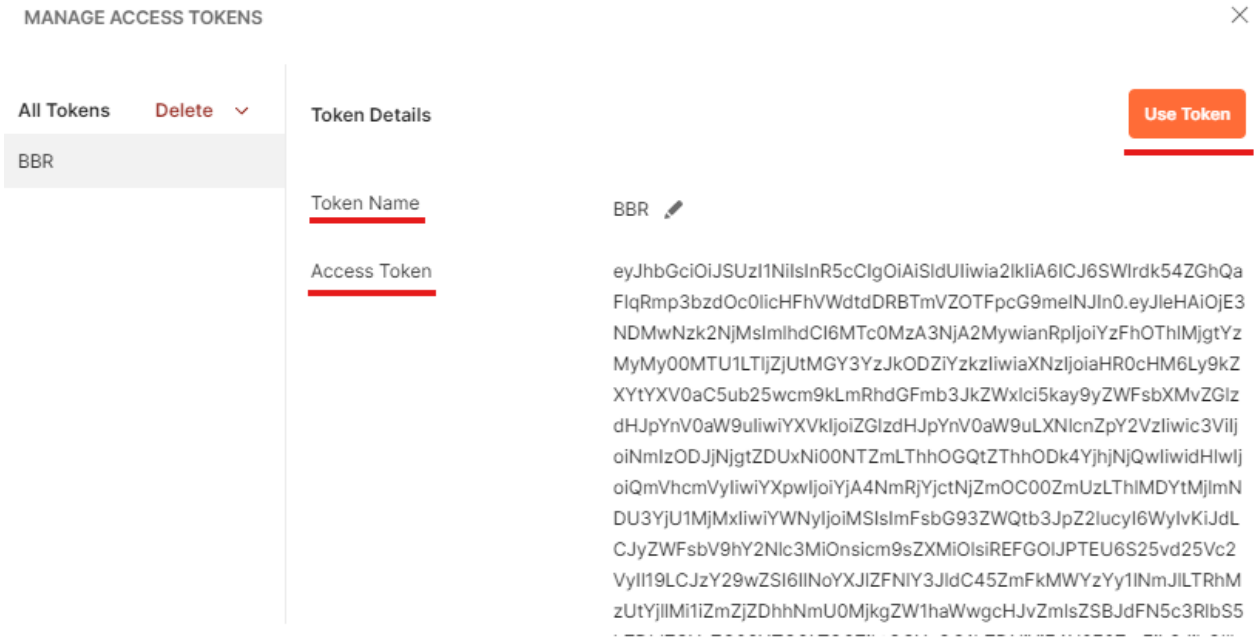


Derefter vil Postman åbne et vindue som indikerer at applikationen er gået i gang med at hente Access Token fra Datafordeleren.



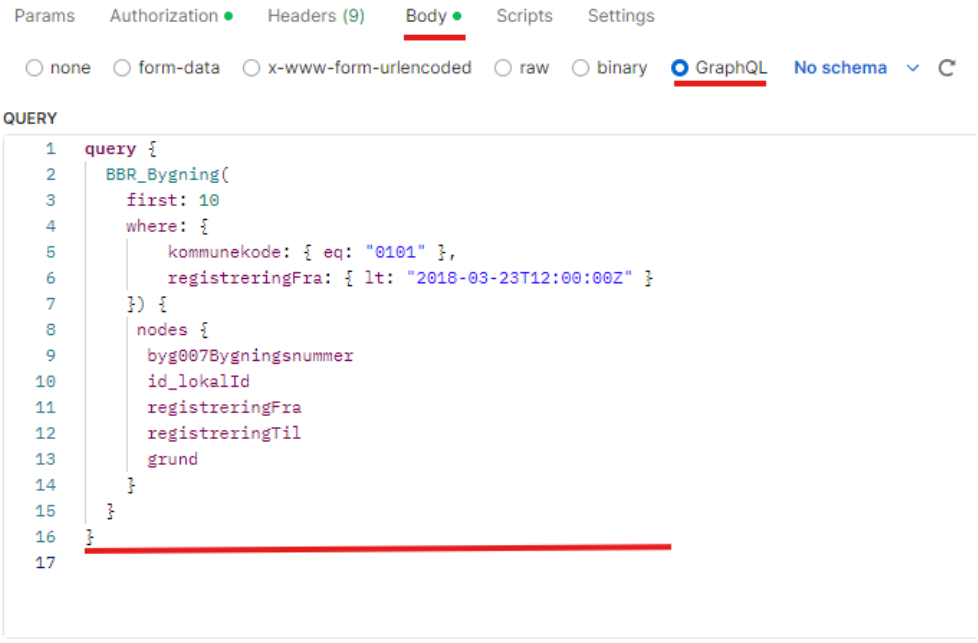
Når Access Token er hentet, vil Postman opdatere vinduet med et grønt flueben og med teksten ”Authentication complete”. Herefter kan du vente 5 sekunder eller trykke på ”Proceed” knappen i nederste højre hjørne. Herefter vil Postman åbne et nyt vindue.

7) Når Postman har åbnet det næste vindue, vil Access token og dets navn stå under ”Token Details”.



Du kan nu trykke på ”Use Token” til at bruge den til at lave queries i Postman eller kopierer access token til anden applikation.

8) Hvis du trykker på ”Use Token” vil Postman automatisk udfylde det i den request som du oprettede tidligere. Herefter kan du lukke vinduet og klikke på ”Body” fanen, efterfuldt ved at klikke på GraphQL muligheden længst til højre. Til sidst kan du skrive en GraphQL-query til BBR og indsætte det i ”Query” tekstfeltet.



9) Når du er klar til at indsende din query kan du trykke på ”Send” oppe i højre hjørne.

Herefter vil svaret fra GraphQL tjenesten blive vist i bunden.

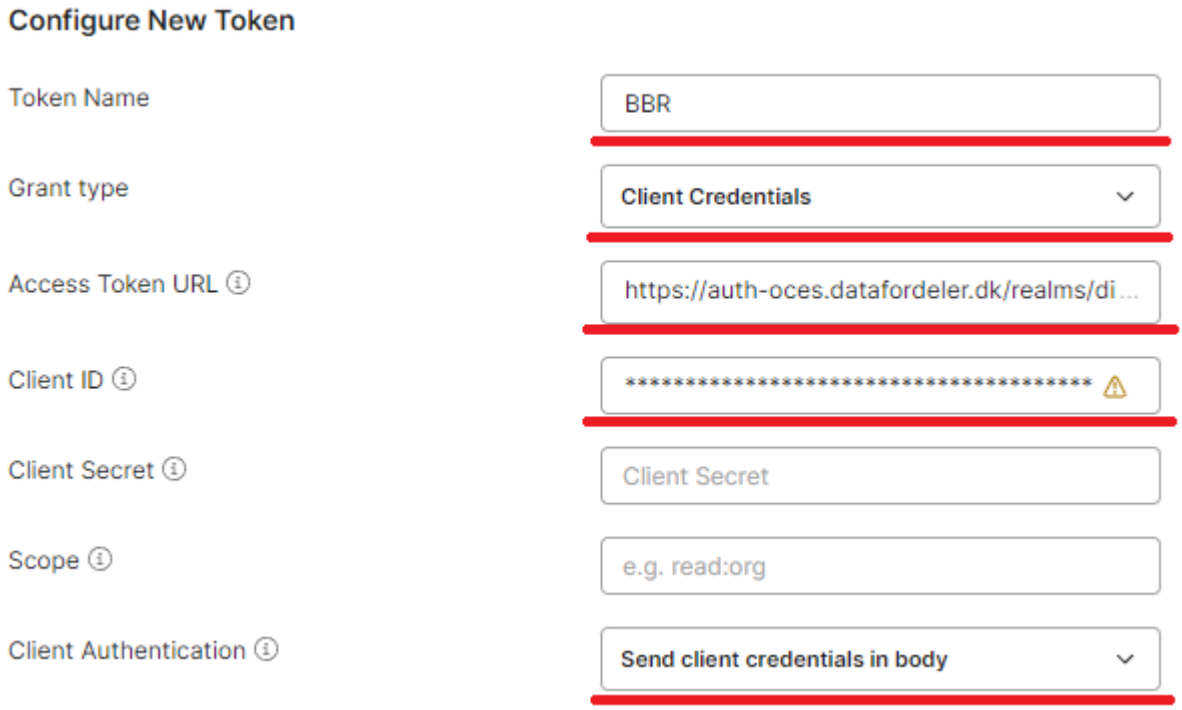
10) Du er nu klar til at indsende queries til Datafordeleren som bruger OAuth 2.0 Client Id og Client Secret til at autentificere til GraphQL tjenesterne.

3.2.2 Opsætning med OAuth certifikat

Såfremt du skal sætte Postman op med OAuth certifikat, skal du sætte OAuth 2.0-indstillinger op, samt vedhæfte et certifikat i Postman. Fremgangsmåden er næsten den samme med Shared Secret, dog med få forskelle. Fremgangsmåden er som følger:

1) Skridt 1-4 følges, som de er beskrevet ovenfor under eksemplet med Shared Secret .

2) Herefter skal OAuth 2.0-indstillinger opsættes for certifikatet. Gå ind under ”Authorization” i requesten, og scroll ned til ”Configure New Token”.



Under overskriften vil der være en række felter som skal udfyldes som beskrevet herunder.

a. I ”Token Name” kan du give din access token et passende navn.

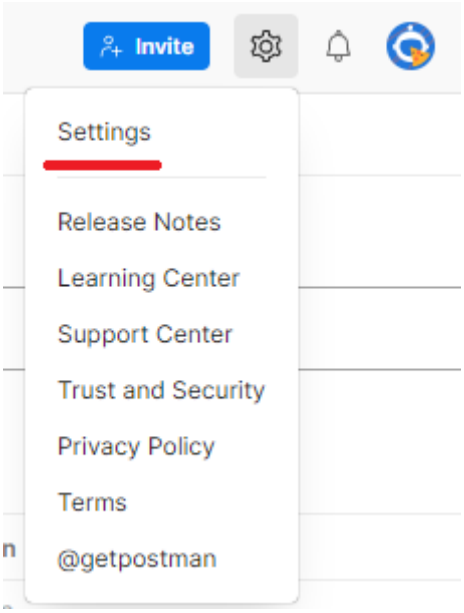
b. I ”Grant type” menuen skal du vælge ”Client Credentials”. Her vil Postman ændre felterne under med dem som relevant for din konfiguration af OAuth.

c. I ”Access Token URL” skal du indsætte følgende link - https://auth-oces.datafordeler.dk/realms/distribution/protocol/openid-connect/token.

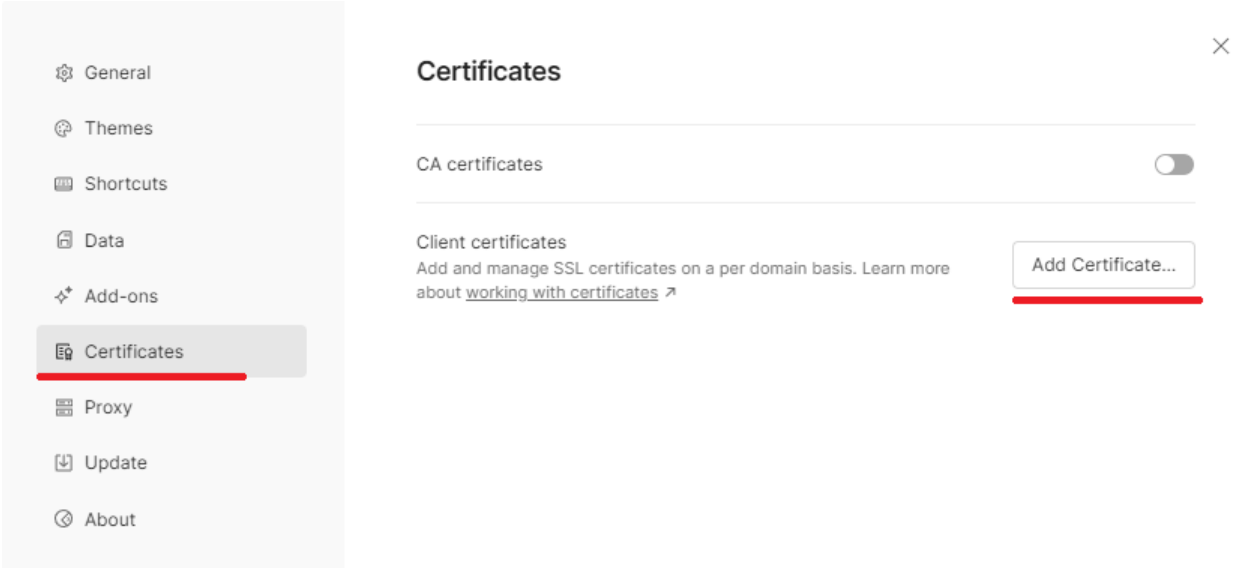
d. I ”Client ID” skal du indsætte det Client ID som du har modtaget efter at have oprettet det i OAuth sektionen på Portalen.

e. Du kan ignorere ”Scope” feltet, men skal verificere at ”Client Authentication” feltet er sat til ”Send client credentials in body”.

3) Vælge ”Settings” oppe i højre hjørne:



4) Vælg herefter ”Add Certificate”.



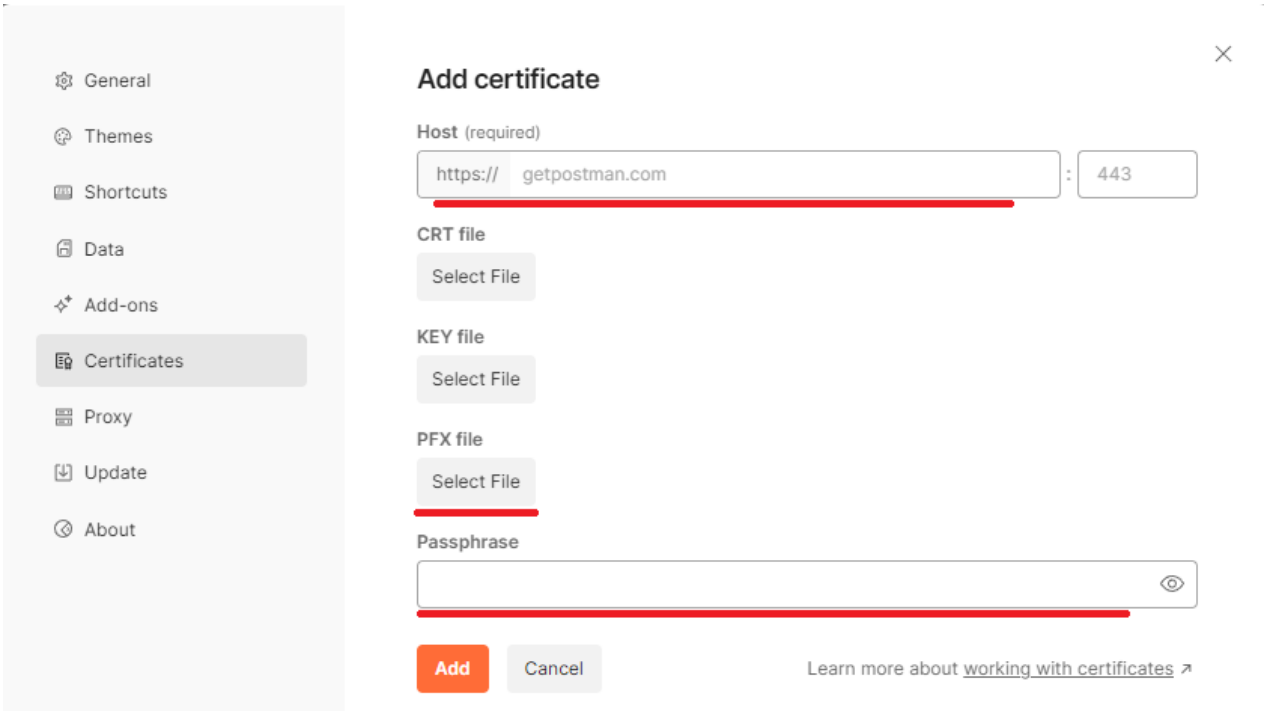
5) Angiv herefter ”host” og certifikatet.

a. Host skal være https://auth-oces.datafordeler.dk.

b. PFX file skal pege på certifikatet som indeholder den private nøgle. Det er vigtigt at dette certifikat stemmer overens med det certifikat som blev uploadet inde i Datafordeler Administration.

c. Passphrase skal stemme overens med password til certifikatet.

d. Bemærk at det kan være nødvendigt at installere ”Den Danske Stat” rodcertifikatet på anvenders maskine for at maskinen stoler på det certifikat der er vedhæftet inde i Postman.



6) Punkt 6-10 følges, som de er beskrevet ovenfor under eksemplet med Shared Secret.

4 Transitionsguide

Denne side beskriver forskelle mellem Datafordelerens nuværende REST-tjenester og GraphQL-tjenesterne, samt hvordan du som anvender af REST-tjenester, kan komme i gang med at bruge GraphQL-tjenesterne.

Det skal bemærkes at GraphQL-tjenester og REST-tjenester driftes parallelt i en periode, så anvendere af Datafordeleren har mulighed for at skifte fra REST-tjenester til GraphQL-tjenester.

Bemærk at de nuværende REST-tjenester understøtter joins mellem forskellige entiteter. Det gør de entitetsbaserede GraphQL-tjenester *ikke*. For at efterligne en af de nuværende REST-tjenester der sammenstiller 4 entiteter i ét kald, skal man, ved brug af GraphQL, lave fire kald – ét for hver entitet – for derefter selv at efterprocessere og sammenstille resultaterne.

4.1 Indhold i returdata

De nuværende REST-tjenester på Datafordeleren giver anvendere mulighed for at hente en delmængde af en entitets datapunkter givet en bestemt URL. Du kan læse mere om de nuværende **REST-tjenester** på **datafordeler.dk**.

GraphQL-tjenesterne giver anvendere mulighed for at specificere nøjagtig hvilke data de har brug for og derved udelade de overflødige data. Det er det samme data der udstilles ved de forskellige tjenester, men forskellen er at GraphQL-tjenesterne giver anvendere mulighed for at hente en delmængde af de data der udstilles ved REST-tjenester.

GraphQL-tjenesterne er derfor velegnede til at hente begrænsede datamængder uden efterfølgende at skulle processere og smide unødig data væk.

4.2 Anvendereksempel på transition fra de nuværende REST-tjenester til GraphQL-tjenester

Dette eksempel tager udgangspunkt i en anvender som benytter entiteten BBR_Bygning i Frederiksberg kommune (kommunekode: "0147") og derfor bruger REST-tjenester.

4.2.1 Situationsbeskrivelse

I dette eksempel laver anvenderen et kald til de nuværende REST-tjenester for at hente følgende oplysninger om alle bygninger i Frederiksberg Kommune:

- Husnummer
- byg007Bygningsnummer
- byg024AntalLejligheder MedKoekken
- byg025AntalLejligheder UdenKoekken
- byg030Vandforsyning

Anvenderen benytter følgende parametre (bemærk at registrerings- og virkningstidspunkt sættes til NOW() hvis ikke angivet):

- Kommunekode = 0147
- PageSize = 1000
- Page = 1

Anvenderen bruger en tjenestebruger med brugernavn og password til at tilgå REST-tjenesten på URL'en:

- https://services.datafordeler.dk/BBR/BBRPublic/1/rest/bygning?username=xxxx&password=yyyy&kommunekode=0147&pagesize=1000&page=1

For at hente de resterende data øges page med én og tjenesten kaldes igen indtil alt data er hentet. Dette resulterer i data som indeholder alle informationer fra entiteten BBR_Bygning og skal derfor efterprocesseres for at fjerne overflødigt data.

4.2.2 Transition

1. Anvender tilgår følgende tjeneste med sin API-nøgle for at hente GraphQL-skemaet for BBR:
   - https://graphql.datafordeler.dk/BBR/v1/schema?apiKey=placeholderNoegle
2. Da anvender nu ved hvilke queries der kan sendes for BBR, sender anvender en POST request til URL'en nedenfor for at hente de første 1.000 datapunkter fra BBR_Bygning for Frederiksberg Kommune med de ønskede oplysninger:
   - https://graphql.datafordeler.dk/BBR/v1?apiKey=placeholderNoegle

```
query {
 BBR_Bygning(
  first: 1000
  virkningstid: "2024-12-12T12:00:00Z" # Dags dato
  registreringstid: "2024-12-12T12:00:00Z" # Dags dato
  where: {
   kommunekode: { eq: "0147" }
  }
 ) {
  pageInfo {
   endCursor
   hasNextPage
  } nodes {
   husnummer
   byg007Bygningsnummer
   byg024AntalLejligheder MedKoekken
   byg025AntalLejligheder UdenKoekken
   byg030Vandforsyning
  }
 }
```

```
}
```

3. Anvender modtager nu de første 1000 datapunkter samt paging-information i json-format som vist nedenfor:

```
{
 "data": {
  "BBR_Bygning": {
   "pageInfo": {
    "endCursor": "1rsNAAAAAAA=",
    "hasNextPage": true
   },
   "nodes": [
    {
     ... # Data
    }
   ]
  }
 }
}
```

4. Anvender kan nu hente de resterende data ved iterativt at sende queries, med den resulterende endCursor-værdi fra det forrige svar, så længe det resulterende JSON-svar har hasNextPage=true.

```
query {
 BBR_Bygning(
  first: 1000
  after: "1rsNAAAAAAA=" # endCursor fra forrige svar
  virkningstid: "2024-12-12T12:00:00Z" # Dags dato
  registreringstid: "2024-12-12T12:00:00Z" # Dags dato
  where: {
   kommunekode: { eq: "0147" }
  }
 ) {
  pageInfo {
   ... # Pagination information
  } nodes {
   ... # Felter
  }
 }
}
```

5. Anvender er nu i besiddelse af alle datapunkter fra BBR_Bygning entiteten for Frederiksberg Kommune med de ønskede informationer. Data behøver ikke efterfølgende at blive efterprocesseret.