

# GraphQL på Datafordelren

Martin Jensen, Klimadatastyrelsen

Margit Kildevang, Klimadatastyrelsen

Morten Fuglsang, Septima

## Hvorfor dette seminar ?



### Nuværende Datafordeler

- ÷ Nuværende REST-tjenester med forskellig logik og struktur fra register til register
- ÷ Komplekst for registre at specificere REST-tjenester via DLS – og høj time-to-market for implementering heraf
- ÷ Meget manuelt arbejde ifm. implementering af nye og ændrede REST-tjenester
- ÷ Mange data, der ellers er på Datafordeleren, er ikke udstillet via REST-tjenester
- ÷ Udstillingens output skemaer følger ikke altid grunddatamodellen



### Moderniserede Datafordeler

- + GraphQL-udstilling for alle entiteter, som anvendere har adgang til
- + Samme logik og struktur for opslag af data, uanset register
- + Ingen register-specifikation (DLS) af udstillings-skemaer, tjenesteparametre eller tjenestelogik
- + Autogenerering af implementering af nye og ændrede GraphQL-tjenester, med minimalt manuelt arbejde
- + Anvendere kan nøjes med at hente de data der er brug for
- ! Anvendere skal (som hidtil) forholde til sammenhæng mellem data ud fra grunddatamodellen

## Dagens program

13.00 - 13.30 : Introduktion og status Martin Jensen KDS

13.30 - 13.50 : Introduktion til GraphQL

13.50 - 14.00 : Pause

14.00 - 14.45 : Hands on eksempler på GraphQL forespørgsler

14.45 - 15.00 : Spørgsmål, opsamling og afslutning

# Introduktion og status

# Introduktion til GraphQL

# Hvorfor GraphQL

## Før GraphQL: REST dominerede API-verdenen

Udfordringer med REST:

- Overfetching og underfetching af data
- Mange API-kald for at hente relaterede data
- Vanskeligt at tilpasse data til forskellige klienter (web, mobil)

# Fødselen af GraphQL hos Facebook

- Skabt af Facebook i 2012 som intern løsning
- Bruges først i deres mobile app til nyhedsfeedet
- Offentliggjort som open source i 2015
- Formål: Effektiv og fleksibel datatilgængelighed med ét enkelt kald

# GraphQL i dag

- Bred adoption hos store virksomheder (GitHub, Shopify, Twitter)
- Bruges til både frontend og backend integrationer
- Aktivt community og voksende økosystem (Apollo, Relay, Hasura osv.)

Et paradigmeskifte: fra ressourcer til grafer

# Hvad er en graf ?

En **graf** består af:

- **Noder (nodes):** Objekter som fx Bruger , Adresse , Postnummer
- **Kanter (edges):** Relationer mellem objekterne (fx en bruger har en adresse)
- **Felter (fields):** Informationen på hver node

En GraphQL-query følger forbindelserne i grafen

Du spørger på tværs af datarelationer – fleksibelt

# Hvad er en query ?

Query: Nøgleordet for en læseoperation

Vi beder om et user-objekt med id =  
"123"

Vi specificerer præcis hvilke felter vi vil  
have: id, name, email

```
query {
  user(id: "123") {
    id
    name
    email
  }
}
```

# Hvordan sender man en forespørgsel ?

GraphQL bruger som regel HTTP POST (kan også være GET)

Body'en indeholder query som tekst (JSON)

```
POST /graphql
```

```
Content-Type: application/json
```

```
{  
  "query": "query { user(id: \"123\") { name email } }"  
}
```

# Applikationer

Man kan i praksis godt bygge en **GET** url, men der skal url-encodes en masse ting.

I praksis vil man ofte benytte en applikation der kan lave **POST** forespørgsler:

- Postman / Insomnia / Altair
- Apollo Client
- curl
- GraphQL Playground / GraphiQL

# Response

Response indeholder kun de felter der blev forespurgt

Ingen overflødig data – ingen status, links, osv.

Altid under data-nøglen (fejl kommer i errors)

```
{  
  "data": {  
    "user": {  
      "name": "Alice",  
      "email": "alice@example.com"  
    }  
  }  
}
```

# Mutations ?

Bruges til at ændre data (oprette, opdatere, slette)

Ligner en query, men påvirker serverens tilstand

Serveren returnerer typisk det opdaterede objekt

```
mutation {
  updateUser(id: "123", input: {
    email: "new@example.com"
  }) {
    id
    name
    email
  }
}
```

The diagram illustrates the comparison between REST API and GraphQL. It features two main sections separated by a large blue 'VS' symbol.

**REST API Section:**

- REST REQUEST:** GET `https://sample.com/person/1`
- REST JSON Response:**

```
{  
  "firstName": "John",  
  "middleName": "Andrew",  
  "lastName": "Smith",  
  "email": "jas1992@gmail.com",  
  "relationshipStatus": "single"  
}
```

**GraphQL Section:**

- GraphQL QUERY:**

```
{  
  person {  
    firstName  
    lastName  
  }  
}
```
- GRAPHQL JSON Response:**

```
{  
  "data": {  
    "person": {  
      "firstName": "John",  
      "lastName": "Smith",  
    }  
  }  
}
```

# GraphQL schemas

Et **GraphQL-schema** definerer de typer og felter, du kan spørge om i API'et.

Schemaet fungerer som:

- En **kontrakt** mellem klient og server
- En **struktur** over alle data og deres relationer
- En **validering** af dine queries

Et schema består af:

- `type` – definerer objekttyper (f.eks. `Adresse`, `Vejstykke`)
- `Query` – definerer tilgængelige læseoperationer
- `Mutation` – definerer skrive-/opdateringsoperationer (hvis tilladt)

Det hele er stærkt typet og dokumenteret...

**type Adresse** = en type med felterne id og adgangsadresse

'ID! = et obligatorisk ID-felt (udråbstegn = required)

**adresse(id: ID!): Adresse** = query, der returnerer én adresse

**adgangsadresser(...)** = query, der returnerer en liste [...]

```
type Adresse {  
    id: ID!  
    adgangsadresse: Adgangsadresse  
}  
  
type Adgangsadresse {  
    vejstykke: Vejstykke  
    husnr: String  
    postnummer: Postnummer  
}  
  
type Query {  
    adresse(id: ID!): Adresse  
    adgangsadresser(postnr: String): [Adgangsadresse]  
}
```

# Eksempel fra Datafordeleren

```
query {
  BBR_Bygning( # Entitet
    where: { # Standardfiltre
      kommunekode: { eq: "0550" }
      virkningsaktoer: { startsWith: "Konvertering" }
    }
    virkningstid: "2024-11-12T14:41:33Z" # Bitemporal filter
    first: 10 # Paging
  ) {
    pageInfo { # Paginginformation
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
    nodes {
      id_lokalId
      kommunekode
      virkningsaktoer
    }
  }
}
```

- **Entitet:** Hver query skal specificere hvilken entitet den omhandler. Eksemplet ovenfor viser en query for entiteten BBR\_Bygning fra BBR og specifikt kun for 3 kolonner (angivet i "nodes"): id\_lokalId, kommunekode og virkningsaktoer.
- **Endepunkt:** En query sendes til et GraphQL-endepunkt som et GET- eller POSTrequest.
- **Standardfiltre:** En query kan indeholde standardfiltre der filtrerer data. Query'en ovenfor indeholder to standardfiltre: et lighedsfilter (eq: "0550") på kommunekode-kolonnen og et tekstfilter (startsWith: "Konvertering") på virkningsaktoer-kolonnen.

- **Geometriske filtre:** En query kan indeholde geometriske filtre, der filterer data på baggrund af geometri.
- **Bitemporale filtre:** En query kan også indeholde bitemporale filtre. Query'en oven for indeholder et bitemporalt point-in-time-filter (virkningstid: "2024-11-12T14:41:33Z") på virkningstidspunktet.
- **Paging :** En query kan også definere hvordan data pagineres. Query'en oven for returnerer de første 10 rækker (first: 10) givet filtreringen samt paginginformation for resultatsættet (angivet i "pageInfo").

# Hvordan får man adgang ?

- Det er nødvendigt at etablere en ny type adgang - det er ikke længere nok med en webbruger og en tjenestebruger.
- Hvis man skal have adgang til frie data, er en token nok, men den skal oprettes

## Administration

Log på Datafordelerens Administration og opret de IT-systemer, der skal have adgang til at hente data via Fildownload (HTTPS) og GraphQL.

[DATAFORDELER ADMINISTRATION](#)

Bemærk at hvert testmiljø har egen Administration.

TEST 03

TEST 04

TEST 06

**Bemærk:** Du skal ansøge i Administration for at få adgang til beskyttede data. Under hvert IT-system kan du oprette API-keys eller OAuth Shared Secrets, som du skal indsætte i URL for at hente data via HTTPS. Du kan også oprette og administrere andre anvendere, der skal have adgang til IT-systemer eller Administration.

# **Velkommen til Datafordeler**

## **Administrationen**

For at få adgang til data fra Datafordeleren skal du oprette dig som anvender med enten e-mail eller MitID. Derefter kan du logge ind i Datafordelerens Administration og oprette de IT-systemer, der skal have adgang til at hente data. Under hvert IT-system kan du oprette API-keys eller OAuth Shared Secrets, som du skal indsætte i URL for at hente data via HTTPS. Du kan også oprette og administrere andre anvendere, der skal have adgang til IT-systemer eller Administration. Du skal ansøge i Administration for at få adgang til beskyttede data. Bemærk at hvert testmiljø også har egen Administration.

IT-systemer

## IT-systemer

Oversigt over oprettede IT-systemer med information om status og din rettighed til IT-systemet. Vælg et IT-system.

Navn	Organisation	Oprettet	Rettighed	Status
<u>MortenTest</u>	Privat bruger	03-06-2025	Ejer	Aktiv

Opret

IT-systemer / IT-system

## Opret API-Key

Navn\*:

MortensTestKey

Opret

## Opret API-Key

Navn\*:

Opret

### API-Key er oprettet

API-Key:

doapzzSj91v

Kopier

Kopier ovenstående API-Key. Den bliver kun vist en gang, og der kan gå op til 15 minutter før API-Key er aktiv.

Tilbage til IT-system

## MortenTest

[API Keys](#)

[Shared Secrets](#)

[Certifikater](#)

[IP Adresser](#)

[Deling af IT-System](#)

### API-Keys

Opret API-Key for at give IT-systemet adgang til tjenester med frie data. Kopier og brug API-Key i URL til tjenesterne.

Navn	Status	Udløbsdato
MortensTestKey	Aktiv	03-06-2027

Bevæbnet med en token, er vi nu klar til  
at begynde at spørge...

# Lad os hente nogle schemaer

`graphql.datafordeler.dk/DAR/v1/schema?apiKey=API-KEY`

`graphql.datafordeler.dk/BBR/v1/schema?apiKey=API-KEY`

`graphql.datafordeler.dk/MAT/v1/schema?apiKey=API-KEY`

Vi starter med at kigge nærmere på DAR schemaet...

# Elementer

- Metadata
- Query types
- Filter inputs
- Spatiale filtre

```
schema {
    query: Query
}

interface SpatialInterfaceType {
    "The OGC geometry type of the geometry."
    type: SpatialGeometryNameEnumType!
    "The coordinate reference system of the geometry."
    crs: Int!
    "The dimensions used by the geometry."
    dimension: SpatialDimensionEnumType!
    "The WKT representation of the geometry"
    wkt: String
}

"""
Grunddatamodel info:
name: Adresse
comment (da): <memo>#NOTES#en sammensat betegnelse, som udpeger o
definition (da): <memo>#NOTES#en struktureret betegnelse som angi
example (da): <memo>#NOTES#adressebetegnelse: Gimles Allé 14, 230
historikmodel: bitemporalitet#NOTES#Values: registreringshistorik
legalSource: https://www.retsinformation.dk/eli/lta/2018/271
prefLabel (da): Adresse
URI: https://data.gov.dk/model/profile/dar/Adresse
EAID: EAID_6043F4D9_C4CA_44d0_BE88_027166A8B008
"""
type DAR_Adresse {
    """
    Grunddatamodel info:
    type: CharacterString
    multiplicity: 1
    URI: https://data.gov.dk/model/profile/dar/adressebetegnelse
}
```

# Query types

```
"Query type for DAR - Danmarks Adresseregister"
type Query {
    DAR_Adressepunkt("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list tha
    DAR_Adresse("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list that co
    DAR_Husnummer("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list that c
    DAR_NavngivenVejKommunedel("Returns the first _n_ elements from the list." first: Int "Returns the elements in th
    DAR_NavngivenVejPostnummer("Returns the first _n_ elements from the list." first: Int "Returns the elements in th
    DAR_NavngivenVejSupplerendeBynavn("Returns the first _n_ elements from the list." first: Int "Returns the element
    DAR_NavngivenVej("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list tha
    DAR_Postnummer("Returns the first _n_ elements from the list." first: Int "Returns the elements in the list that
    DAR_SupplerendeBynavn("Returns the first _n_ elements from the list." first: Int "Returns the elements in the li
}
```

# Filter inputs

```
input DAR_AdresseFilterInput {  
    and: [DAR_AdresseFilterInput!]  
    adressebetegnelse: DafSearchableStringOperationFilterInput  
    bygning: DafStringOperationFilterInput  
    datafordelerOpdateringstid: DafDateTimeOperationFilterInput  
    doerbetegnelse: DafStringOperationFilterInput  
    doerpunkt: DafStringOperationFilterInput  
    etagebetegnelse: DafStringOperationFilterInput  
    husnummer: DafStringOperationFilterInput  
    id_lokalId: DafStringOperationFilterInput  
    registreringFra: DafDateTimeOperationFilterInput  
    registreringsaktoer: DafStringOperationFilterInput  
    registreringTil: DafDateTimeOperationFilterInput  
    status: DafStringOperationFilterInput  
    virkningFra: DafDateTimeOperationFilterInput  
    virkningsaktoer: DafStringOperationFilterInput  
    virkningTil: DafDateTimeOperationFilterInput  
}
```

# Spatial Filters

```
input SpatialFilterInput {  
    and: [SpatialFilterInput!]  
    or: [SpatialFilterInput!]  
    contains: SpatialContainsOperationFilterInput @cost(weight: "5000")  
    covers: SpatialCoversOperationFilterInput @cost(weight: "5000")  
    crosses: SpatialCrossesOperationFilterInput @cost(weight: "5000")  
    intersects: SpatialIntersectsOperationFilterInput @cost(weight: "5000")  
    within: SpatialWithinOperationFilterInput @cost(weight: "5000")  
    overlaps: SpatialOverlapsOperationFilterInput @cost(weight: "5000")  
    touches: SpatialTouchesOperationFilterInput @cost(weight: "5000")  
    coveredBy: SpatialCoveredByOperationFilterInput @cost(weight: "5000")  
}
```

# Metadata

```
type DAR_Adresse {  
    """  
    Grunddatamodel info:  
    type: CharacterString  
    multiplicity: 1  
    URI: https://data.gov.dk/model/profile/dar/adressebetegnelse  
    prefLabel (da): adressebetegnelse  
    definition (da): en læsbar struktur, som identifierer adressen fuldstændigt  
    legalSource: https://www.retsinformation.dk/eli/lta/2018/271#P12  
    """  
    adressebetegnelse: String @cost(weight: "1")  
    bygning: String @cost(weight: "1")  
    datafordelerOpdateringstid: DafDateTime @cost(weight: "1")  
    """  
    Grunddatamodel info:  
    type: CharacterString  
    multiplicity: 0..1  
    URI: https://data.gov.dk/model/profile/dar/dørbetegnelse  
    prefLabel (da): dørbetegnelse  
    definition (da): betegnelse, som angiver den adgangsdør e.l. som adressen identifierer  
    legalSource: https://www.retsinformation.dk/eli/lta/2018/271#P22 || https://www.retsinformation.dk/eli/lta/2018/271#P23  
    """
```

## Hvordan sender man en POST forespørgsel ?

Det er kompliceret at lave en GET - det er bedre at sende et POST request afsted.

- Dette kan gøres med en række forskellige stykker software - Jeg bruger Postman til mine eksempler:

[www.postman.com/](http://www.postman.com/)

# Basis request

Endpoint : graphql.datafordeler.dk/DAR/v1?apiKey=API-KEY

```
query {
  DAR_Adresse(
    registreringstid: "2025-06-04T00:00:00Z"
    virkningstid: "2025-06-04T00:00:00Z"
    first: 10
  ) {
    nodes {
      adressebetegnelse
      id_lokalId
      husnummer
      doerbetegnelse
      etagebetegnelse
      status
      registreringFra
      virkningFra
    }
  }
}
```

# GEO FORUM

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Collections, Environments, Flows, and History. The main workspace is titled "My Workspace". A search bar at the top right contains the placeholder "Search Postman" and keyboard shortcuts "Ctrl K". The top navigation bar has links for "Home", "Workspaces", and "API Network". A central panel displays a "GET Get DAR Schema" request to the "GraphQL Datafordeleren / Get DAR Schema" endpoint. The request URL is <https://graphql.datafordeler.dk/BBR/v1/schema?apiKey=doapzzSj91vskMzjkCPaiE88nZ0lpSKsqUVztJmt991GNsSSxzcpAiyxB1avf...>. The "Params" tab is selected, showing two parameters: "apiKey" with value "doapzzSj91vskMzjkCPaiE88nZ0lpSKsqUVztJmt991GNsSSxzcpAiyxB1avf...". Below the table are sections for "Cookies", "Query Params", and "Body". The "Body" section shows the response status as "200 OK", time taken as "5.96 s", and size as "149.3 KB". The response content is displayed in "Raw" format, showing a schema definition:

```
schema {
  query: Query
}

interface SpatialInterfaceType {
  "The OGC geometry type of the geometry."
  type: SpatialGeometryNameEnumType!
  "The coordinate reference system of the geometry."
  crs: Int!
  "The dimensions used by the geometry."
  dimension: SpatialDimensionEnumType!
  "The WKT representation of the geometry"
  wkt: String
```

HTTP GraphQL Datafordeleren / DAR adresse

POST <https://graphql.datafordeler.dk/DAR/v1?apiKey=doapzzSj91vskMzjkCPaiE88nZ0lpSKsqUVztJmt991GNsSSxzcpAiyxB1avBclmza5l...> Send

Params • Authorization Headers (9) Body • Scripts Settings Cookies

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL Auto Fetch C ⚠

**QUERY**

```
1 query {  
2   DAR_Adresse(  
3     registreringstid: "2025-06-04T00:00:00Z"  
4     virkningstid: "2025-06-04T00:00:00Z"  
5     first: 10  
6   ) {  
7     nodes {  
8       adressebetegnelse  
9       id_lokalId  
10      husnummer  
11      doerbetegnelse  
12      etagebetegnelse  
13      status  
14      registreringFra  
15      virkningFra  
16    }  
17  }  
18 }
```

**GRAPHQL VARIABLES** ⓘ

1	
---	--

Body Cookies Headers (6) Test Results | ⏱

200 OK • 6.70 s • 3.39 KB • ⓘ | e.g. Save Response ⋮

{ } JSON ▾ ▷ Preview ⚡ Visualize | ↻

```
1 {  
2     "data": {  
3         "DAR_Adresse": {  
4             "nodes": [  
5                 {  
6                     "adressebetegnelse": "Østre Stationsvej 2F, 6. 211, 5000 Odense C",  
7                     "id_lokalId": "0000090e-e9f3-4ffe-a0a5-2852666d158c",  
8                     "husnummer": "405bec91-1609-46de-89e0-94e646a898d2",  
9                     "doerbetegnelse": "211",  
10                    "etagebetegnelse": "6",  
11                    "status": "4",  
12                    "registreringFra": "2021-06-21T07:54:04.191816Z",  
13                    "virkningFra": "2021-06-21T07:54:04.191816Z"  
14                },  
15                {  
16                    "adressebetegnelse": "Vestergade 41A, 2. tv, 4850 Stubbekøbing",  
17                    "id_lokalId": "000021c5-e9ee-411d-b2d8-ec9161780ccd",  
18                    "husnummer": "0a3f5086-e9cd-32b8-e044-0003ba298018",  
19                    "doerbetegnelse": "tv",  
20                }  
21            ]  
22        }  
23    }  
24}
```

# Hands on eksempler på GraphQL forespørgsler

Alle ressourcer vi har brugt i dag, kan hentes på  
Github

[Github.com/MFuglsang/graphql\\_datafordeleren](https://github.com/MFuglsang/graphql_datafordeleren)

Her er bla. en pdf med en lang række eksempler - langt flere end vi har vist her...

# Spørgsmål, opsamling og afslutning

Tak for i dag...