MDD NRW Schnittstellenbeschreibung

v1.6

Januar 2026

Contents

1 Autorisierung **2**

2 GraphQL Endpunkt **4**
2.1 Schema . . . 4
2.2 Komplexe Attribute . . . 5
2.3 Queries . . . 6
2.3.1 Aufbau des Query types . . . 7
2.3.2 Query Argumente . . . 9
2.3.3 Filtern von Queries . . . 10
2.3.4 Abfragen gebündelter Daten . . . 11
2.3.5 Geofilter . . . 13
2.3.6 Datenkatalog Meta-Daten . . . 15
2.3.7 Postman Query Baum . . . 17
2.3.8 GZIP Kompression . . . 19

3 GBFS Endpunkt **20**
3.1 Liste der unterstützten GBFS Versionen . . . 20
3.2 Gebündelte GBFS Feeds . . . 20

4 GTFS Endpunkt **21**

5 Datex-II Endpunkt **22**

6 Geoserver **23**
6.1 Authentifizierung . . . 23
6.2 Endpunkte . . . 24
6.3 Beispiel: Hinzufügen des WFS Feeds in QGIS . . . 26

7 Direkte Schnittstellen **28**
7.1 DELFI Landeshintergrundsystem . . . 28
7.2 goFLUX Mobility GmbH . . . 29

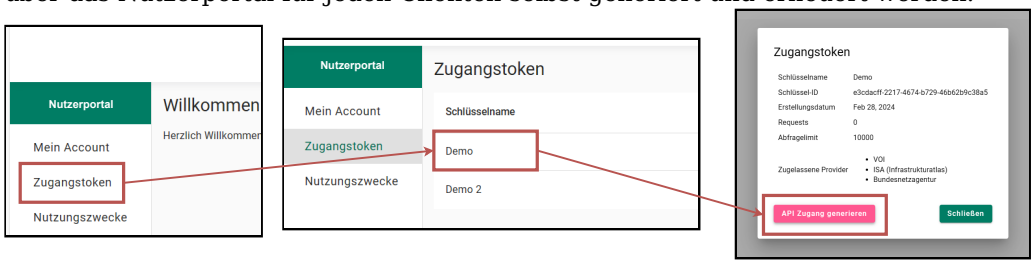
1 Autorisierung

In der MDD NRW sind jedem Nutzer ein oder mehrere Clienten zugeordnet. Clienten stellen voneinander unabhängige Zugänge zu den Daten in der MDD NRW dar. Jeder Client verfügt über einen Zugangstoken (in Form eines API-Keys), mit dem die Daten der MDD NRW abgefragt werden können. An den Clienten sind auch Einschränkungen, wie das Request Limit und Zugriffsrechte, gebunden. Die Anzahl der Clienten und die jeweiligen Zugriffsrechte werden **nicht** vom User selbst verwaltet.
Für eine Erweiterung der Zugriffsrechte oder Erhöhung des Request Limits muss der Support unter mdd-nrw@gorheinland.com kontaktiert werden.

!Wichtig!

Jedem Clienten sind eine oder mehrere Funktionen zugeordnet, aus denen sich die ebenfalls im Nutzerportal einsehbaren Nutzungszwecke ableiten. Es ist die Pflicht jedes Nutzers der MDD NRW diese Nutzungszwecke einzuhalten.

Die Autorisierung erfolgt mittels API-Key. API-Keys können von registrierten Nutzern über das Nutzerportal für jeden Clienten selbst generiert und erneuert werden.



Zur Autorisierung gegenüber einer der Schnittstellen der MDD NRW muss der API-Key im Header mit dem **key** x-api-key mitgegeben werden.

Kontakt: mdd-nrw@gorheinland.com

Beispiel:
Abfrage des GBFS Auto-discovery Files der Version 2.3 des Providers mit ID 1 mit dem API-Key <api-key>:

```
curl --location
'https://mdd.gorheinland.com/gbfs-output/1/2.3/gbfs.json' \
--header 'x-api-key: <api-key>'
```

Hinweis

API-Keys werden nicht zentral in der MDD NRW gespeichert. Sie werden beim generieren einmalig dem Nutzer angezeigt und müssen vom Nutzer eigenständig gesichert werden. Ist der aktuelle API-Key verloren, kann dieser nicht wiederhergestellt werden und der Nutzer muss sich im Nutzerportal der MDD NRW einen neuen API-Key generieren.

Kontakt: mdd-nrw@gorheinland.com

2 GraphQL Endpunkt

GraphQL ist eine Abfragesprache, die es erlaubt mittels einer query nur diejenigen Daten abzufragen, die benötigt werden. Das GraphQL-Schema (siehe Kapitel Schema) definiert, welche Abfragen an die Schnittstelle grundlegend möglich sind. Die genaue Erstellung der query ist dem Nutzer überlassen.

In GraphQL gibt es zwei Arten von **Attributen**, primitiv und komplex. Ein Attribut ist **primitiv**, wenn es selber keine Attribute hat. Primitive Attribute haben primitive Datentypen wie

- String
- BigInteger
- Boolean
- Float
- DateTime
- ...

Komplexe Attribute werden im Schema beschrieben und bestehen selbst aus primitiven und komplexen Attributen. Durch das Verschachteln komplexer Attribute werden Verknüpfungen zwischen Daten hergestellt, welche flexible Abfragen über mehrere Attribute hinweg erlauben. Weitere Informationen zum Abfragen komplexer Attribute finden sich im Kapitel Queries.

2.1 Schema

Das GraphQL-Schema kann über einen **GET**-Request an folgende URL abfragt werden:

```
https://mdd.gorheinland.com/graphql/schema.graphql
```

Die Response ist ein GraphQL-Schema. Das GraphQL-Schema beschreibt, welche Abfragen über die MDD NRW GraphQL-Schnittstelle möglich sind. Es enthält folgende grundlegenden Keywords:

- type: Komplexe Datentypen
- enum: Auflistungen
- input: Argumente

Kontakt: mdd-nrw@gorheinland.com

Ein type definiert komplexe Datentypen. Komplexe Datentypen sind in GraphQL vielseitig einsetzbar. Neben abfragbaren Daten werden hierüber auch die Funktionen zum Abfragen der Daten definiert, siehe Kapitel Queries. Mehr zu komplexen Datentypen zum Abfragen von Daten findet sich in Kapitel Komplexe Attribute.
Ein enum ist eine vordefinierte Liste von Werten. Hier z.B. FeatureType in gekürzter Darstellung:

```
enum FeatureType {
  ACCESSIBLE_PARTIAL_PAVED_SURFACE
  ACCESSIBLE_PAVED_SURFACE
  ACCESSIBLE_UNPAVED_SURFACE
  ACCESS_AT_GROUND_LEVEL
  ACCESS_CURB_STEP
  ACCESS_DOOR
...
}
```

Das Keyword input weist darauf hin, dass dieser Typ als Input für Queries verwendet wird, wie beispielsweise **Filter**. Der Input-Typ ParkingFacilityFilter wird zum Beispiel im query listParkingFacilities verwendet. Mehr zum Thema Filtern findet sich in Kapitel Filtern von Queries.

Hinweis

Die hier gezeigten Queries dienen nur zur Veranschaulichung und basieren nicht unbedingt auf der aktuellsten Version des GraphQL-Schemas.
Wir empfehlen immer das aktuelle GraphQL-Schema über die Schnittstelle abzufragen und auf dessen Basis Queries zu bauen.

2.2 Komplexe Attribute

Ein komplexes Attribut wird mit dem Keyword type initialisiert. Alle verfügbaren komplexen Attribute sind im GraphQL-Schema beschrieben und sind wie folgt aufgebaut:

```
"DESCRIPTION"
type TYPENAME {
  FIELD_NAME : TYPE
  OTHER_FIELD_NAME: DATA_TYPE
  ...
}
```

TYPENAME ist der Name des komplexen Attributs. Die Attribute hinter den Feldnamen zeigen an, welche Attribute in einer query abgefragt werden können.
Es ist also möglich anhand des Schemas auszulesen, welche Informationen über einen im GraphQL-Schema definierten type abgefragt werden können und mit welchen anderen Datentypen die Informationen verknüpft sind.
Beispiel: Definition eines komplexen Datentypen
Der type ParkingFacility ist wie folgt im Schema definiert:

Kontakt: mdd-nrw@gorheinland.com

```
"parking facility, for example a bikebox"
type ParkingFacility {
  address: String
  available: Boolean
  boxes: [Box]
  capacities: [Capacity]
  chargingStations: [ChargingStation]
  description: String
  "Id given by provider"
  externalId: String!
  externalType: String
  features: [ParkingFacilityFeature]
  feeDescription: String
  id: BigInteger
  "ISO-8601"
  lastUpdated: DateTime
  "latitude of current location of the parking facility"
  lat: Float
  "longitude of current location of the parking facility"
  lon: Float
  lowestPriceInCents: Int
  name: String!
  nextPublicTransportLocation: Location
  openingTimesDescription: String
  parkingType: ParkingType
  postCode: String
  provider: Provider!
  tariffs: [Tariff]
  "Get entry/entries for a certain key/s"
  urls(key: [UrlKey]): [Entry_UrlKey_Url]
}
```

Es ist möglich, im komplexen Attribut ParkingFacility direkt primitive Attribute wie lat, lon und id abzufragen. Bei komplexen Attributen wie "Box" muss in der query angegeben werden, welche Attribute von "Box" enthalten sein sollen.

Hinweis

Wenn ein Datenfeld in der type Definition mit eckigen Klammern "[]" umschlossen ist, wird bei Abfragen des Datenfeldes eine Liste zurückgegeben. Beispielsweise gibt "boxes" eine Liste von komplexen Datentypen der Entität "Box" zurück.

2.3 Queries

Um die GraphQL Schnittstelle anzusprechen, sendet man einen **POST Request** an:

```
https://mdd.gorheinland.com/graphql
```

Eine query bestimmt, welche Attribute abgefragt werden. Wenn ein Feld selbst ein komplexes Attribut zurück gibt, muss festlegt werden, welche Attribute man von diesem Attribut benötigt, solange bis alle Anfrage-Äste auf einem primitiven Attribut enden. Dieser Baum, den man in einem Query aufspannt, muss also **immer auf einem primitiven Attribut enden**. Es können alle Attribute, welche im type Query

Kontakt: mdd-nrw@gorheinland.com

aufgelistet sind, abgefragt werden.

Die query selbst wird im **body** des Requests mitgeschickt. Dabei können den Queries Argumente übergeben werden und der Rückgabetyp ist ebenfalls im Schema angegeben, wie in Kapitel Komplexe Attribute beschrieben.

2.3.1 Aufbau des Query types

Welche Anfragen an die Schnittstelle möglich sind, ist ebenfalls im **GraphQL-Schema** als type definiert.

```
"Query root"
type Query {
  VehicleTypeTypeById(id: BigInteger!): VehicleType
  alertById(id: BigInteger!): Alert
  calendarById(id: BigInteger!): Calendar
  geofencingZoneById(id: BigInteger!): GeofencingZone
  "get Rides by goFlux"
  getRides(arrivalLat: Float!, arrivalLng: Float!, arrivalRadius: Int!, day: Int!, departureLat: Float
       !, departureLng: Float!, departureRadius: Int!, hour: Int!, minute: Int!, month: Int!,
       timeDeltaSeconds: Int!, year: Int!): String
  listAlerts(limit: Int, offset: Int, providerId: BigInteger): [Alert]
  listCalendars(limit: Int, offset: Int, providerId: BigInteger): [Calendar]
  listGeofencingZones(limit: Int, offset: Int, providerId: BigInteger): [GeofencingZone]
  listLocations(limit: Int, offset: Int, providerId: BigInteger): [Location]
  listParkingFacilities(filter: ParkingFacilityFilter, limit: Int, offset: Int, order: Order, orderBy:
        ParkingFacilityOrderBy): [ParkingFacility]
  listPricingPlans(limit: Int, offset: Int, providerId: BigInteger): [PricingPlan]
  listProviders(limit: Int, offset: Int): [Provider]
  listRawFeedData(limit: Int, offset: Int, providerId: BigInteger): [RawFeedData]
  listRegions(limit: Int, offset: Int, providerId: BigInteger): [Region]
  listRentalHours(limit: Int, offset: Int, providerId: BigInteger): [RentalHour]
  listSharingStations(limit: Int, offset: Int, providerId: BigInteger): [SharingStation]
  listSystemInformation(limit: Int, offset: Int, providerId: BigInteger): [SystemInformation]
  listVehicleTypeTypes(limit: Int, offset: Int, providerId: BigInteger): [VehicleType]
  listVehicles(limit: Int, offset: Int, providerId: BigInteger): [Vehicle]
  locationById(id: BigInteger!): Location
  parkingFacilityById(id: BigInteger!): ParkingFacility
  pricingPlanById(id: BigInteger!): PricingPlan
  providerById(id: BigInteger!): Provider
  rawFeedDataById(id: BigInteger!): RawFeedData
  regionById(id: BigInteger!): Region
  rentalHourById(id: BigInteger!): RentalHour
  sharingStationById(id: BigInteger!): SharingStation
  systemInformationById(id: BigInteger!): SystemInformation
  vehicleById(id: BigInteger!): Vehicle
  "Welcome"
  welcome(name: String = "World"): String
}
```


Kontakt: mdd-nrw@gorheinland.com

Alle im type Query definierten Funktionen lassen sich für Abfragen der Daten in der MDD NRW nutzen. Die Query

```
query ListProviders {
    listProviders {
        name
        id
    }
}
```

beispielsweise gibt Namen und ID aller Provider zurück, auf die man Zugriffsrechte hat.
Grundlegend lassen sich die so zur Verfügung gestellten Endpunkte in zwei Arten unterteilen.

`listEntity,`

welche eine Liste der angefragten Entität zurückgeben und

`entityById,`

welche die der ID zugehörige Entität zurückgeben.
Die query entityById benötigt dabei immer den Parameter "id". Zu dieser "id" muss eine Entität in der Datenbank existieren, damit die Abfrage funktioniert.

Beispiel: entityById Query

```
query {
  ParkingFacilityById(id: 1) {
    address
    available
    boxes {
      capacity
    }
  }
}
```

Diese Abfrage gibt alle Entitäten des types ParkingFacility zurück, welche dem provider mit ID "1" zugeordnet sind.
Analog gibt die folgende Abfrage die gleichen Informationen für Entitäten des types ParkingFacility zurück, allerdings anstatt nur die Entitäten eines Providers eine Liste aller Entitäten, auf die man Zugriffsrechte hat.

```
query {
  listParkingFacility {
    address
    available
    boxes {
      capacity
    }
  }
}
```


Kontakt: mdd-nrw@gorheinland.com

Hinweis

Es ist auch möglich, bei Abfragen der Art listEntity die Ergebnisse zu verfeinern. Mehr dazu findet sich im Kapitel Filtern von Queries.

2.3.2 Query Argumente

Einer Query können **Argumente** übergeben werden, um die Response weiter zu verfeinern. Die verfügbaren Argumente können in den im type Query definierten Funktionen eingesehen werden. **Beispiel:** "listParkingFacilities" ist im Schema definiert als:

```
listParkingFacilities(filter: ParkingFacilityFilter,
                      limit: Int,
                      offset: Int,
                      order: Order,
                      orderBy: ParkingFacilityOrderBy)
                      : [ParkingFacility]
```

Die verfügbaren Argumente sind somit

- filter: Komplexe Filter, siehe Kapitel Filtern von Queries
- limit (Int): Maximale Anzahl zurückgegebener Entitäten. Es wird empfohlen bei jeder Abfrage ein Limit zu setzen, da das Abfragen von zu vielen Entitäten auf einmal sehr lange dauern kann. Das ist insbesondere beim Abfragen von GTFS-Daten über GraphQL relevant.
- offset (Int): Anzahl von Entitäten die in der Response übersprungen werden. Zusammen mit limit für **Pagination** geeignet.
- order: **ASC** für aufsteigend, **DESC** für absteigend.
- orderBy: Bestimmt, nach welchem Feld sortiert werden soll. Die verfügbaren Felder sind als enum der zugehörigen "typeOrderBy" im GraphQL-Schema definiert.

Beispiel: Query mit Argumenten

```
query {
  listParkingFacilities(limit: 3, offset: 2, orderBy: parkingType, order: ASC) {
    id
    name
    parkingType
    lastUpdated
    provider {
      id
    }
  }
}
```


Kontakt: mdd-nrw@gorheinland.com

2.3.3 Filtern von Queries

Filter arbeiten auf den in type Query definierten Funktionen für die Datenabfrage. Im Beispiel von "listParkingFacilities" ist als Filter der ParkingFacilityFilter definiert. Filter sind, wie in Kapitel Schema beschrieben, im GraphQL-Schema als input definiert. Der ParkingFacilityFilter sieht wie folgt aus:

```
input ParkingFacilityFilter {
  address: [StringFieldFilter]
  available: EqualityFieldFilter_Boolean
  coordinate: LocationFieldFilter
  description: [StringFieldFilter]
  externalId: EqualityFieldFilter_BigInteger
  externalType: [StringFieldFilter]
  lastUpdated: [DateFieldFilter]
  lowestPriceInCents: [NumberFieldFilter_Int]
  name: [StringFieldFilter]
  nextPublicTransportLocationId: EqualityFieldFilter_BigInteger
  nextPublicTransportLocationName: [StringFieldFilter]
  parkingType: EqualityFieldFilter_ParkingType
  postCode: [StringFieldFilter]
  providerId: EqualityFieldFilter
  providerName: [StringFieldFilter]
}
```

Im ParkingFacilityFilter sind für die meisten primitiven Attribute vom type ParkingFacility je nach Datentyp **Feld-Filter** definiert. Darüber hinaus gibt es einen Filter für ID und Name des Providers, also für primitive Attribute des komplexen Attributs Provider. Die Feld-Filter sind ebenfalls im GraphQL-Schema aufgelistet. Um nach providerId zu filtern, wird hier der Feld-Filter ProviderIdFilter genutzt. Dieser ist definiert als:

```
input ProviderIdFilter {
  operator: ContainedOperator
  values: [BigInteger]
}
```

Feld-Filter bestehen immer aus einem value und je nach Filter über einen operator. Wenn es einen Operator gibt, ist der ebenfalls im GraphQL-Schema als enum definiert. Operatoren geben an, wie die Einträge in "value" zum Filtern genutzt werden sollen. Hier wird der ContainedOperator verwendet, dieser ist definiert als:

```
enum ContainedOperator {
  containedIn
  notContainedIn
}
```


Kontakt: mdd-nrw@gorheinland.com

Um hier also nur ParkingFacilities abzufragen, die den Providern mit ID "1", "2" und "3" zugeordnet sind, muss der Filter wie folgt gesetzt werden:

```
query {
  listParkingFacilities(
    filter: { providerId: { values: [1, 2, 3], operator: containedIn}}
  ) {
    id
    name
    parkingType
    lastUpdated
    provider {
      id
    }
  }
}
```

Es können beliebig viele Filter unter dem filter Keyword gesetzt und kombiniert werden. Multiple Filter werden mit Kommas getrennt und sind logisch als "und" verknüpft.

```
query {
  listParkingFacilities(
    filter: {
      providerId: { values: [1, 2, 3], operator: containedIn},
      lastUpdated: { value: "2024-01-17T18:00:00.00Z", operator: after}
    }
  ) {
    id
    name
    parkingType
    lastUpdated
    provider {
      id
    }
  }
}
```

Diese Abfrage gibt also nur Entitäten zurück, welche providerId "1", "2" oder "3" haben **und** nach dem 17.01.2024 00:00 Uhr zuletzt aktualisiert wurden.
Geofilter sind besondere Filter, die in Kapitel Geofilter beschrieben werden.

2.3.4 Abfragen gebündelter Daten

GraphQL erlaubt durch das Setzen der entsprechenden Filter das Abfragen von gebündelten Daten. Mit gebündelten Daten sind in diesem Kontext die Daten mehrerer Provider gemeint, welche im Datenmodell der MDD NRW harmonisiert wurden.

Beispiel: Abfrage aller Fahrzeuge die zu den Providern mit den IDs 1, 2 und 3 gehören:

Kontakt: mdd-nrw@gorheinland.com

```
query ListVehicles {
  listVehicles(
    filter: {
      providerId: { values: [1, 2, 3] }, operator: containedIn}
  ) {
    id
    lastReported
    lastUpdated
    lat
    lon
    status
  }
}
```

Der Provider-Filter erlaubt es außerdem, Nutzern der MDD NRW die gebündelten Daten aller Zonen eines Providers abzufragen. Wenn zum Beispiel der Provider mit ID 1 Zonen mit den IDs 11, 12 und 13 hat, sind folgende Abfragen identisch:

```
query ListVehicles {
  listVehicles(
    filter: {
      providerId: { values: [1] }, operator: containedIn}
  ) {
    id
    lastReported
    lastUpdated
    lat
    lon
    status
  }
}
```

und

```
query ListVehicles {
  listVehicles(
    filter: {
      providerId: { values: [11, 12, 13] }, operator: containedIn}
  ) {
    id
    lastReported
    lastUpdated
    lat
    lon
    status
  }
}
```

Im Datenmodell der MDD NRW wird nicht zwischen Providern und Provider-Zonen unterschieden. Des Weiteren verfügt nicht jeder Provider über Zonen. Die Zugehörigkeit von Provider-Zonen zu Providern kann im Datenkatalog der MDD NRW oder dem Feld parentProvider in der listProviders und providerbyID Queries entnommen werden. Welche Provider über Zonen verfügen und welche nicht, kann im Datenkatalog der MDD NRW eingesehen werden.

Kontakt: mdd-nrw@gorheinland.com

2.3.5 Geofilter

Ortsbezogene Daten der MDD NRW können mittels **Umkreisfilter**, **Polygonfilter** und **Keyword** nach Kommune gefiltert werden.
Umkreisfilter erlauben es, Entitäten innerhalb eines nutzerdefinierten Kreises abzufragen. Wenn ein Attribut als Feld-Filter "LocationFieldFilter"

```
input LocationFieldFilter {
  lat: Float
  lon: Float
  rangeInMeter: Float
}
```

hat, ist es möglich durch übergeben eines Mittelpunkts, in der Form von longitude und latitude Koordinaten und eines Radius in Metern nur jene Entitäten zu erhalten, die innerhalb des dadurch entstehenden Kreises liegen.
Beispiel: Umkreisfilter mit Radius 15km

```
query {
  listParkingFacilities(
    filter: {
      coordinateInRadius: { lat: 50.786075564, lon: 6.101432940, rangeInMeter: 15000 }
    }
  ) {
    id
    name
    lat
    lon
    provider {
      id
    }
  }
}
```

Der Polygonfilter erlaubt es, eine Liste von Geokoordinaten zu übergeben. Es werden nur Entitäten zurückgegeben, die sich innerhalb des so definierten Polygons befinden. Das gegebene Polygon muss dabei zusammenhängend und geschlossen sein, das heißt der erste Punkt entspricht dem Letzten. Um mehrere, nicht überlappende Polygone abzufragen, müssen multiple Abfragen gestellt werden.
Beispiel: Polygonfilter

```
query {
  listParkingFacilities(
    filter: {
      coordinateInPolygon: {coordinates :  [{lat: 50.781547, lon: 6.087198},
                                            {lat: 50.790207, lon: 6.057198},
                                            {lat: 50.774207, lon: 6.057198},
                                            {lat: 50.781547, lon: 6.087198}]}
    }
  ) {
    id
    name
    lat
    lon
    provider {
      id
    }
```


Kontakt: mdd-nrw@gorheinland.com

```
  }
}
```

Der Filter nach Keywords für Kommunen (siehe Liste) ist ein Polygonfilter, der bereits die Fläche der jeweiligen Kommune (auf Kreisebene bzw. Ebene einer kreisfreien Stadt) berücksichtigt.

Beispiel: Filter nach Keywords für Kommunen

```
query {
  listParkingFacilities(
    filter: {
      providerId: { operator: containedIn, values: 45 }
            coordinateInRegion: { namedRegion: Duesseldorf }
    }
  ) {
    id
    name
    lat
    lon
    provider {
      id
    }
  }
}
```

Die Keywords AVV, NWL, VRR, VRS sowie go.Rheinland und Nordrhein-Westfalen sind gültig, ebenso für die Kommunen auf der folgenden Seite (zugeordnet nach Verbundraum):

Kontakt: mdd-nrw@gorheinland.com

AVV	NWL	VRR	VRS
Düren	Bielefeld	Bochum	Bonn
Heinsberg	Borken	Bottrop	Euskirchen
Städteregion Aachen	Coesfeld	Dortmund	Köln
	Gütersloh	Duisburg	Leverkusen
	Hamm	Düsseldorf	Oberbergischer Kreis
	Herford	Ennepe-Ruhr-Kreis	Rhein-Erft-Kreis
	Hochsauerlandkreis	Essen	Rheinisch-Bergischer Kreis
	Höxter	Gelsenkirchen	Rhein-Sieg-Kreis
	Lippe	Hagen	
	Märkischer Kreis	Herne	
	Minden-Lübbecke	Kleve	
	Münster	Mettmann	
	Olpe	Mönchengladbach	
	Paderborn	Mülheim a.d. Ruhr	
	Siegen-Wittgenstein	Oberhausen	
	Soest	Recklinghausen	
	Steinfurt	Rhein-Kreis-Neuss	
	Unna	Remscheid	
	Warendorf	Solingen	
		Wesel	
		Wuppertal	
		Viersen	

2.3.6 Datenkatalog Meta-Daten

Die Synchronisation zwischen Datenkatalog und MDD NRW erlaubt es, die dort gesetzten Meta-Daten als Filter für GraphQL zu verwenden.

Die verfügbaren Meta-Daten für alle verfügbaren Provider sind über die folgende Abfrage verfügbar:

```
query {
    listMetakeywords {
        keyword
        type
        value
        id
    }
}
```


Kontakt: mdd-nrw@gorheinland.com

Die Datenkatalog Meta-Daten setzen sich aus drei Feldern zusammen: Typ, Schlüsselwort und Wert.

Typ:
Der Typ des Meta-Datum entspricht den verschiedenen Schlüsselwort - Typen des Datenkatalogs. Die möglichen Werte sind als ENUMS definiert und müssen ohne Anführungszeichen (") übergeben werden. Die möglichen Typen sind:

- TAG
- GROUP
- FORMAT
- LICENSE
- CUSTOM_FIELD

Schlüsselwort:
Dies ist nur notwendig bei Abfrage von Meta-Daten des Typs "CUSTOM_FIELD". Benutzerdefinierte Felder bieten mit dem Schlüsselwort eine weitere Ebene in der Hierarchie bei der Abfrage, um verschiedene benutzerdefinierte Felder voneinander abtrennen zu können. Das Schlüsselwort entspricht bei allen Typen außer "CUSTOM_FIELD" dem Wert des Meta-Datums.

Wert:
Dieses Feld beinhaltet die Ausprägung des Meta-Datums. **Beispiel:** „Bikesharing“.

Die Meta-Daten können beim Filtern der Provider angewendet werden, in dem eine Liste von Typ, Schlüsselwort und Wert - Kombinationen in den metaKeyword Filter gesetzt werden. Das Filtern einer Kombination dieser Felder ist mit einem logischen **„UND“** verknüpft und liefert alle Datenlieferanten, auf die diese Kombination zutrifft.
Beispiel:
Schema für eine Abfrage von providern mit Meta-Daten Filter.

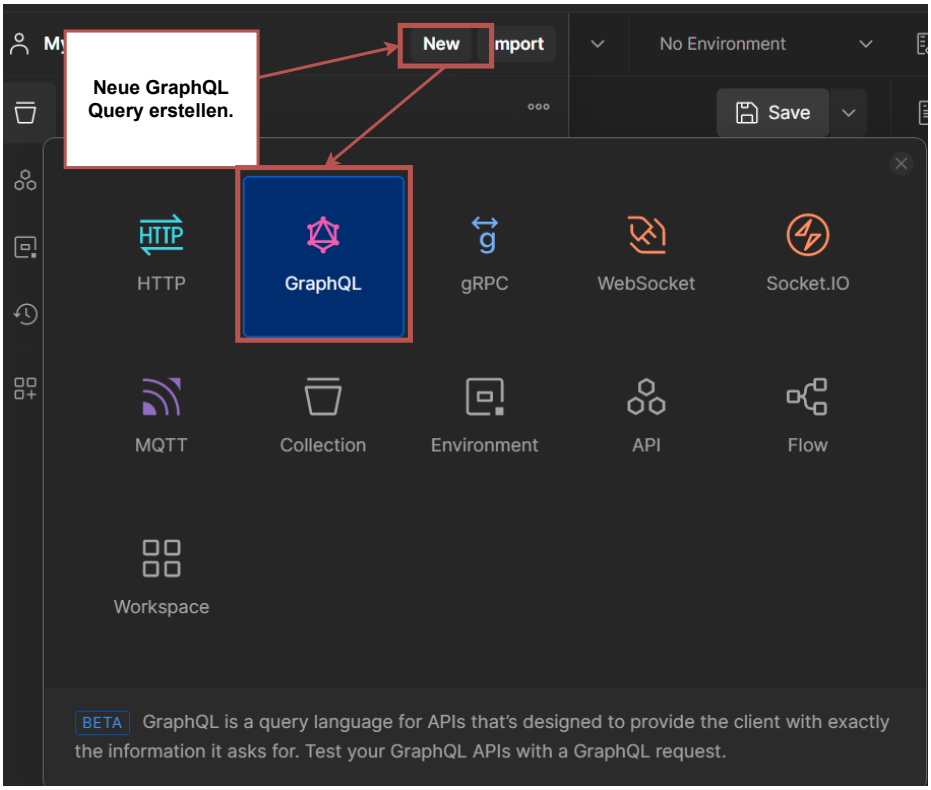
```
query {
  listProviders(
    filter: {
      metaKeyword: [
        { type: <Typ> }
        { keyword: "<Schluesselwort>" }
        { value: "<Wert>" }
      ]
    }
  ) {
    id
    name
    metakeywords {
      keyword
```


Kontakt: mdd-nrw@gorheinland.com

```
      type
      value
    }
  }
}
```

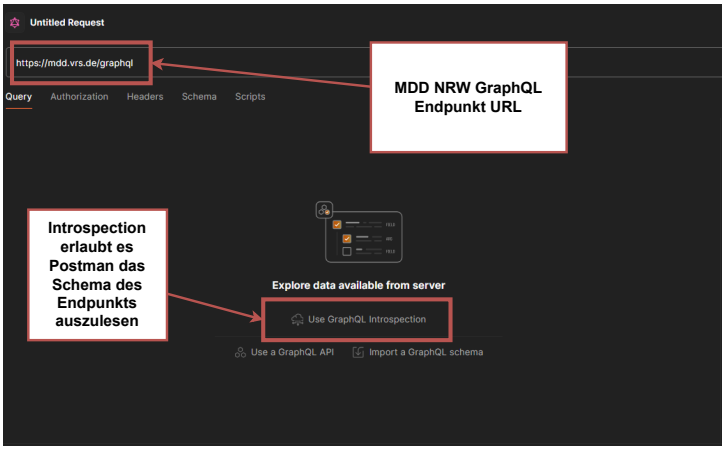
2.3.7 Postman Query Baum

Zum einfachen Erstellen von Queries empfehlen wir die Nutzung von Postman. Über die GraphQL Introspection ist es möglich, zum Schema passende Queries direkt in Postman zu erstellen.
Dazu wird zunächst eine neuer GraphQL Request in einem Postman Workspace erstellt.

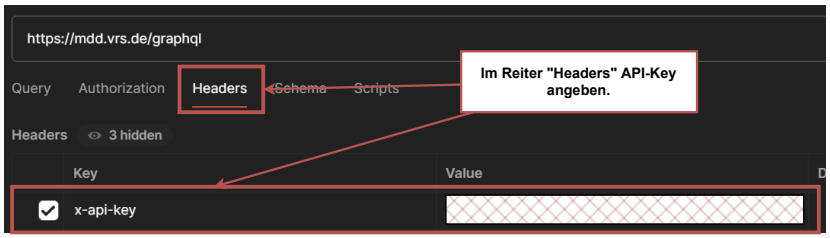


Als URL wird der GraphQL Endpunkt der MDD NRW angegeben und die Introspection gestartet.

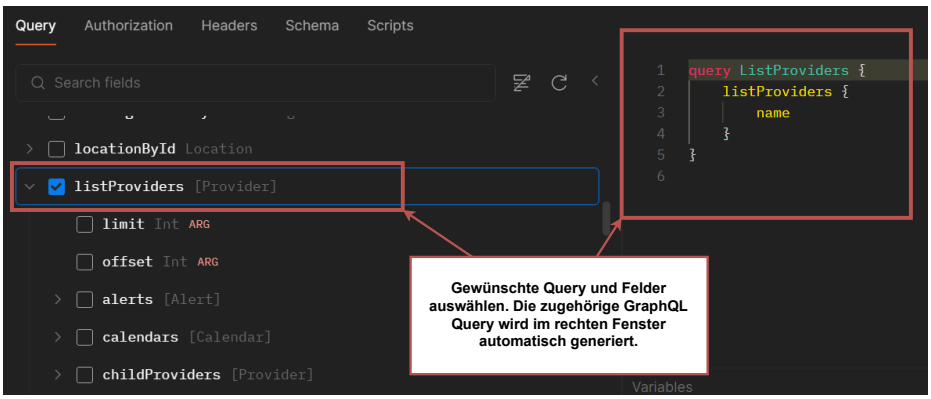
Kontakt: mdd-nrw@gorheinland.com



Für die Autorisierung muss im Header über den Key x-api-key ein gültiger API-Key übergeben werden, vgl. Kapitel Autorisierung.

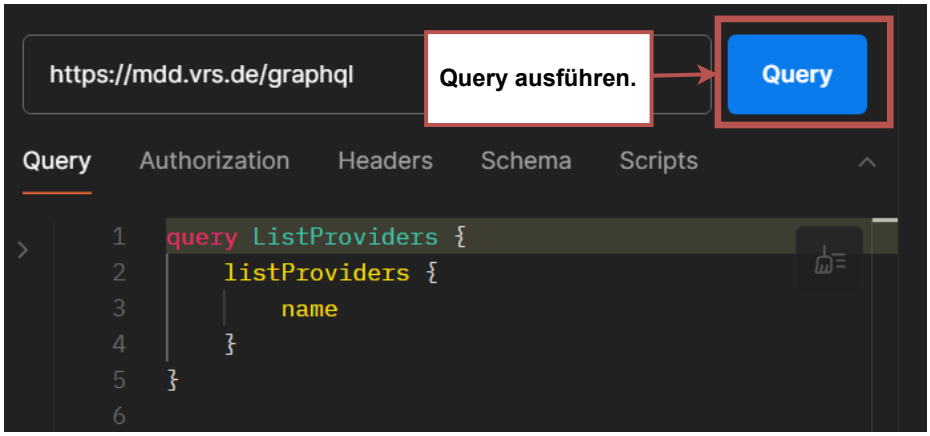


Postman erkennt nun automatisch die in Abschnitt Queries beschriebenen Endpunkte und erlaubt durch eine klickbare Oberfläche gültige Queries zu generieren.



Die Query kann nun direkt in Postman ausgeführt oder anderweitig verwendet werden.

Kontakt: mdd-nrw@gorheinland.com



Postman unterstützt auch Pagination und Filter über diese Oberfläche, allerdings werden Listenfilter aktuell noch nicht unterstützt und müssen vom Nutzer manuell zur Query hinzugefügt werden.

2.3.8 GZIP Kompression

Es ist möglich, die Größe des Outputs einer GraphQL Abfrage zu reduzieren, indem bei der Abfrage GZIP-Kompression verwendet wird.
Dafür müssen im Header der Anfrage folgende Parameter mitgegeben werden:

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

und

```
Accept-Encoding: gzip
```


Kontakt: mdd-nrw@gorheinland.com

3 GBFS Endpunkt

Der GBFS Endpunkt kann per **GET**-Request nach folgendem Schema abgerufen werden:

```
https://mdd.gorheinland.com/gbfs-output/<provider_id>/<version>/<
    endpunkt>
```

Als <provider_id> wird die ID eines zulässigen Providers eingesetzt. In diesem Beispiel arbeiten wir mit dem erfunden Provider "ScooterSharer" mit ID "1". Das Keyword <version> bestimmt die Version des GBFS Feeds, der abgefragt werden soll und als <endpunkt> stehen alle der gewählten Version üblichen Endpunkte nach GBFS-Spezifikation zur Verfügung. Über den gbfs.json Endpunkt erhält man somit auch eine Übersicht über die verfügbaren GBFS-Endpunkte des jeweiligen Datenlieferanten in der MDD NRW.

Beispiel: Abfrage des Auto-discovery Files des GBFS-Feeds der Version 2.3 für den Provider mit ID 1:

```
https://mdd.gorheinland.com/gbfs-output/1/2.3/gbfs.json
```

3.1 Liste der unterstützten GBFS Versionen

Zum aktuellen Zeitpunkt unterstützt die MDD NRW GBFS Feeds folgender Versionen:

- Version 2.3
- Version 3.0

3.2 Gebündelte GBFS Feeds

Gebündelte Feeds sammeln die Daten aller Zonen der zugehörigen Datenlieferanten. Gebündelte Feeds können, falls verfügbar, über die ID des Anbieters wie von einer Zone abgefragt werden.
Hat zum Beispiel der Anbieter mit ID 1 die Zonen mit IDs 11 und 12, so kann der gebündelte Feed mit allen Daten der Zonen 11 und 12 wie folgt abgefragt werden:

```
https://mdd.gorheinland.com/gbfs-output/1/2.3/gbfs.json
```

Um einzigartige GBFS-IDs zu gewährleisten, werden die originalen IDs mit dem Namen der jeweiligen Zone verschnitten. Das verwendete Muster ist:

```
<system_information name der Zone>_:_<original ID des objects>
```


Kontakt: mdd-nrw@gorheinland.com

4 GTFS Endpunkt

Der GTFS Endpunkt kann per **GET**-Request nach folgendem Schema abgerufen werden:

```
https://mdd.gorheinland.com/gtfs-output/<provider_id>/
```

Als <provider_id> wird die ID eines zulässigen Providers eingesetzt. In diesem Beispiel arbeiten wir mit dem erfunden Provider "Fahrplandaten" mit ID "2". Der Call gibt ein ZIP-Ordner mit den verfügbaren Dateien nach GTFS-Standard zurück.
Beispiel: Abfrage des GTFS-Feeds den Provider mit ID 2:

```
https://mdd.gorheinland.com/gtfs-output/2/
```

Hinweis

Der GTFS-Feed für die Landesweiten Fahrplandaten NRW wird aufgrund seiner Größe nur über den direkten GTFS Endpunkt vollständig zur Verfügung gestellt. Über die Graph-QL Schnittstelle erhält man nur einen reduzierten Feed **ohne** Haltestelleninformationen.

Kontakt: mdd-nrw@gorheinland.com

5 Datex-II Endpunkt

Datex-II ist der europäische Standard für den Austausch von Verkehrsdaten.
Die MDD NRW stellt für geeignete Anbieter dynamische und statische Datex-II-Feeds zur Verfügung.
Das URL-Schema für die Abfrage der statischen Feeds ist:

```
https://mdd.gorheinland.com//datex2-output/<version>/<provider_id>/
```

Die URL für dynamische Feeds ist:

```
https://mdd.gorheinland.com//datex2-output/<version>/<provider_id>/
    status
```

Welche Anbieter welche Feeds unterstützt, kann den Factsheets entnommen werden.
Zum aktuellen Zeitpunkt unterstützt die MDD NRW Datex-II in der Version 3.1.
Link zur offiziellen Dokumentation.

Das Archiv für das verwendete Datex-II Schema kann hier heruntergeladen werden.

Beispiel: Abfrage des statischen Datex-II Feeds der Version 3.1 für den Provider mit ID 1:

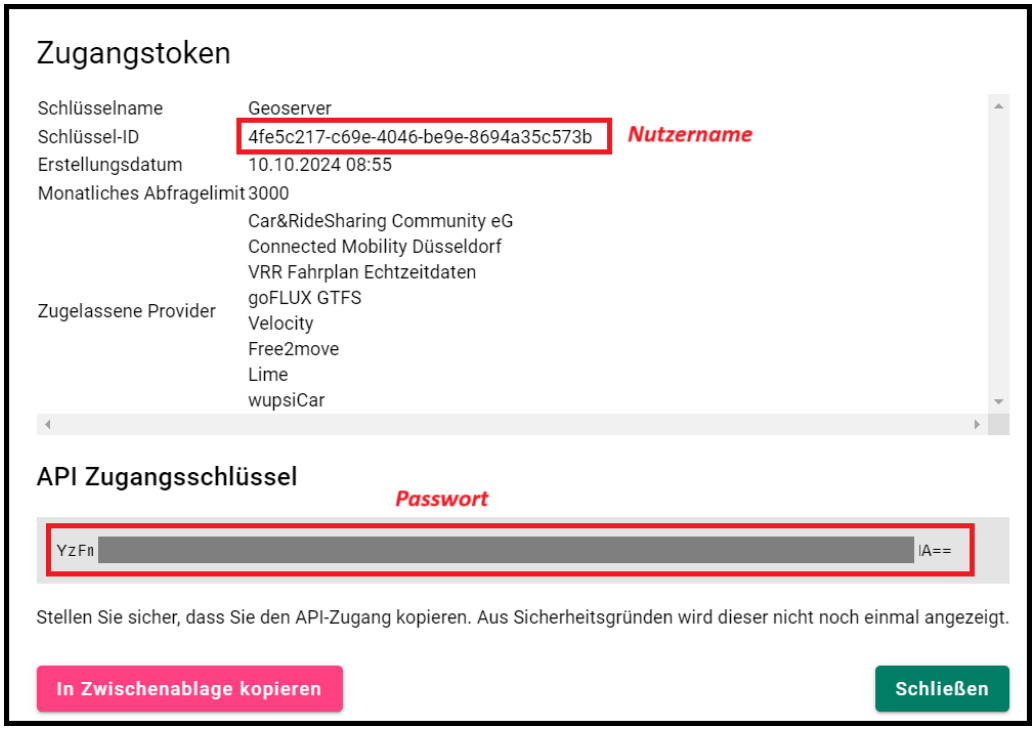
```
https://mdd.gorheinland.com/datex2-output/3.1/1/
```


Kontakt: mdd-nrw@gorheinland.com

6 Geoserver

6.1 Authentifizierung

Der Zugang zum Geoserver (mdd.gorheinland.com/geoserver) erfolgt anders als bei anderen Endpunkten über Basic Auth mit Nutzername und Passwort. Nutzername und Passwort setzen sich aus der Schlüssel-ID und dem API-Token eines Clienten zusammen.



Kontakt: mdd-nrw@gorheinland.com

Die Zugangsdaten sind in dieser Maske einzutragen:

Hinweis

Die Schlüssel-ID eines Clienten ist fest und ändert sich **nicht**, wenn ein neues API-Token generiert wird. In diesem Fall muss für den Zugang zum Geoserver also nur das Passwort ausgetauscht werden.

6.2 Endpunkte

Die MDD NRW bietet Karten-Endpunkte über einen Geoserver an.
Die unterstützten Formate sind:

Endpunkt	Versionen				
WMS	1.3.0	1.1.1			
WMTS	1.1.1				
TMS	1.0.0				
WMS-C	1.1.1				
WFS	2.0.0	1.1.0	1.0.0		
WCS	2.0.1	1.1.1	1.1.0	1.1	1.0.0

Jedes Format verfügt dabei über einen eigenen Endpunkt:

WMS

```
https://mdd.gorheinland.com/geoserver/ows?service=WMS&version=<version
    >&request=GetCapabilities
```

WMTS

```
https://mdd.gorheinland.com/geoserver/gwc/service/wmts?service=WMTS&
    version=<version>&request=GetCapabilities
```


Kontakt: mdd-nrw@gorheinland.com

TMS

```
https://mdd.gorheinland.com/geoserver/gwc/service/tms/<version>
```

WMS-C

```
https://mdd.gorheinland.com/geoserver/gwc/service/wms?service=WMS&
    version=<version>&request=GetCapabilities&tiled=true
```

WFS

```
https://mdd.gorheinland.com/geoserver/ows?service=WFS&version=<version
    >&request=GetCapabilities
```

WCS

```
https://mdd.gorheinland.com/geoserver/ows?service=WCS&version=<version
    >&request=GetCapabilities
```

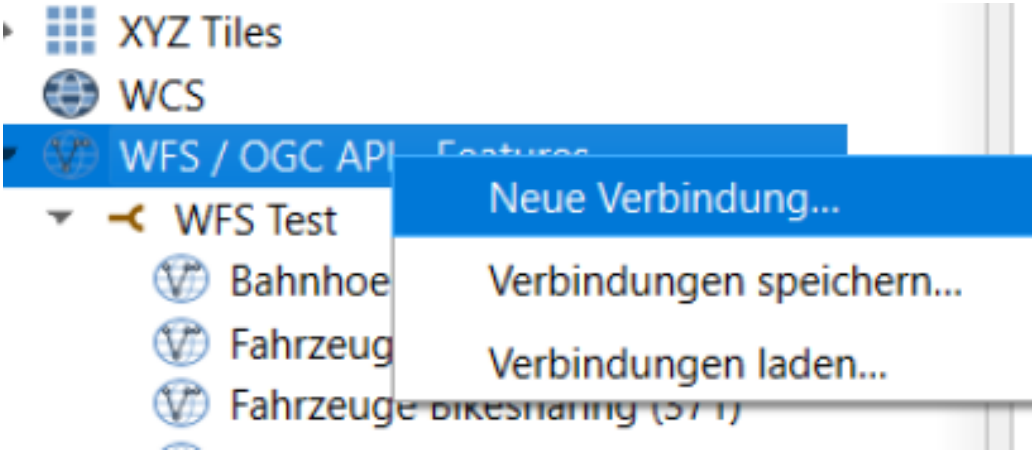
Beispiel: Abfrage des WCS Services mit Version 2.0.0

```
curl --location 'https://mdd.gorheinland.com/geoserver/ows?service=WCS&
    version=2.0.0&request=GetCapabilities' \
--header 'Authorization: <Base64Encode{Nutzername:Passwort}>'
```

6.3 Beispiel: Hinzufügen des WFS Feeds in QGIS

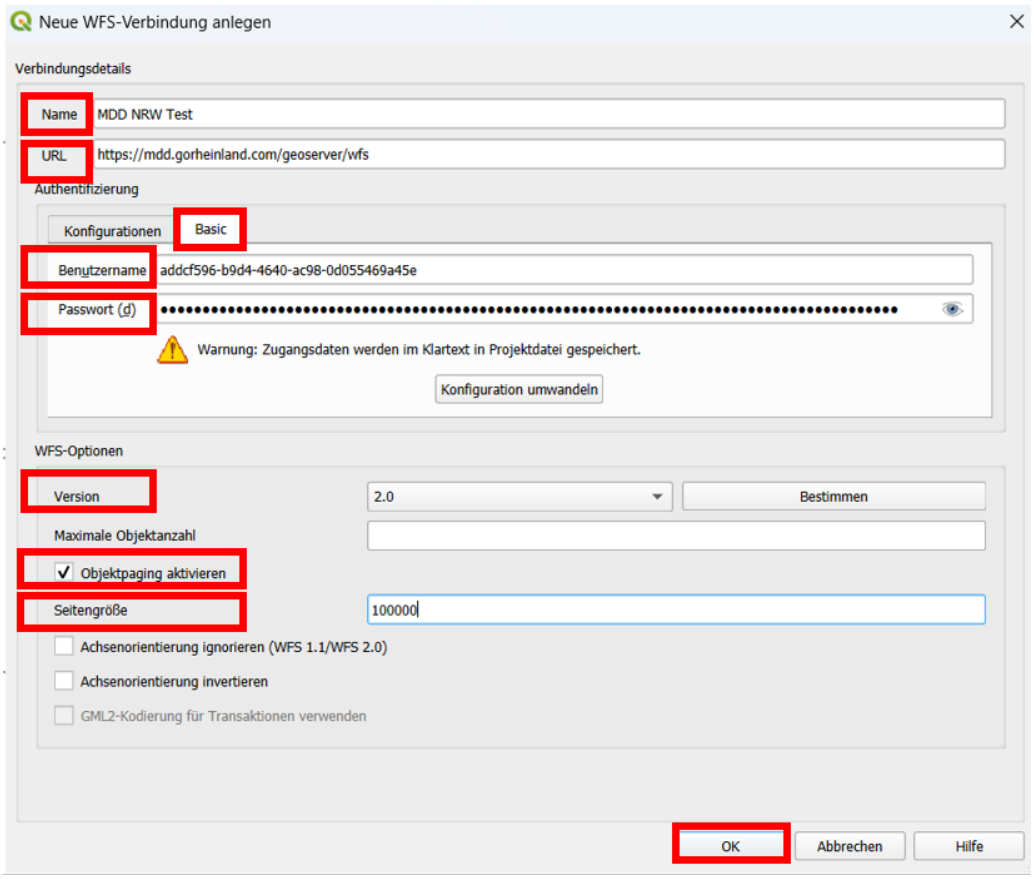
Im Folgenden werden Beispiele für das Hinzufügen des WFS- Feeds in QGIS gegeben, da dieses eins der gängigen Tools zur Einbettung von Geodaten darstellt. Es ist zu beachten, dass sich die Oberfläche ändern kann. Die Screenshots dienen lediglich der Veranschaulichung.
In QGIS müssen folgende Schritte umgesetzt werden, um den WFS-Feed hinzuzufügen:

- im linken Reiter per Rechtsklick auf WFS / OGC API Feature "Neue Verbindung auswählen" (siehe Screenshot)



- im Pop-Up unter Verbindungsdetails
  - Name vergeben
  - URL für WFS-Feed eingeben
- unter Authentifizierung "Basic" auswählen
  - Benutzername eingeben
  - Passwort eingeben
- unter WFS-Optionen
  - Version 2.0 auswählen
  - Objektpaging anhaken
  - Seitengröße auf 10000 einstellen
  - ok klicken (siehe Screenshot nächste Seite)

Kontakt: mdd-nrw@gorheinland.com



Danach werden die einzelnen Layer (je nach zugelassenen Rechten variieren diese) geladen.

7 Direkte Schnittstellen

Datenlieferanten, deren Daten nicht in der MDD NRW vorgehalten werden, aber über die MDD NRW abfragbar sind, werden als direkte Schnittstellen zur Verfügung gestellt.
Diese Schnittstellen sind über die MDD NRW abfragbar, indem der Request über die MDD NRW direkt an den betreffenden Service weiter geleitet wird.
Zum jetzigen Zeitpunkt ist die direkte Abfrage von Daten aus dem DELFI Landeshintergrundsystem und goFLUX Mobility GmbH möglich.

7.1 DELFI Landeshintergrundsystem

Das DELFI Landeshintergrundsystem verwendet als Schnittstelle die vom VDV entwickelte VDV 431 TRIAS Schnittstelle. Die Dokumentation kann auf dieser Website eingesehen werden: Informationen zu TRIAS.
Abfragen werden über den Endpunkt

```
POST https://mdd.gorheinland.com/delfi
```

unter Angabe des MDD NRW API-Key gestellt, siehe Kapitel Autorisierung. Der Content-Type des Requests ist "application/xml".
Der Inhalt des Requests wird über den Request-Body abgefragt, dieser muss dem TRIAS-Format entsprechen.
Beispiel:
Schematischer Request-Body für eine DELFI Abfrage:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Trias version="1.2" xmlns="http://www.vdv.de/trias" xmlns:siri="http://www.siri.org.uk/siri">
    <ServiceRequest>
        <siri:RequestTimestamp>YYYY-MM-DDTHH:mm:SS</siri:RequestTimestamp>
        <siri:RequestorRef>xxx</siri:RequestorRef>
        <RequestPayload>
            <StopEventRequest>
                <Location>
                    <LocationRef>
                        <StopPointRef>STOP_POINT_REF</StopPointRef>
                    </LocationRef>
                    <DepArrTime>YYYY-MM-DDTHH:mm:SS</DepArrTime>
                </Location>
                <Params>
                    <NumberOfResults>10</NumberOfResults>
                    <StopEventType>arrival</StopEventType>
                    <IncludePreviousCalls>true</IncludePreviousCalls>
                    <IncludeOnwardCalls>true</IncludeOnwardCalls>
                    <IncludeRealtimeData>true</IncludeRealtimeData>
                </Params>
            </StopEventRequest>
        </RequestPayload>
    </ServiceRequest>
</Trias>
```


Kontakt: mdd-nrw@gorheinland.com

7.2 goFLUX Mobility GmbH

Abfragen werden über den Endpunkt

```
POST https://mdd.gorheinland.com/goflux
```

unter Angabe des MDD NRW API-Keys gestellt, siehe Kapitel Autorisierung. Der Content-Type des Requests ist "application/json".
Der Inhalt des Requests wird über den Request-Body abgefragt, dieser muss dem goFLUX-Format entsprechen. Die Schnittstellenbeschreibung kann auf dieser Website eingesehen werden: Dokumentation goFLUX.

Beispiel
Schematischer Request-Body für eine goFLUX Abfrage:

```
{
    "date": {unix_timestamp[s]}, # Beispiel: 1721137180
    "timeDelta": 7200,
    "departureLat": 48.800534,
    "departureLng": 2.295088,
    "arrivalLat": 48.765463,
    "arrivalLng": 2.072499,
    "departureRadius": 10,
    "arrivalRadius": 10,
    "expectedDepartureDateTime": "{{request_timestamp}}"
}
```


Kontakt: mdd-nrw@gorheinland.com