

# Datendrehscheibe Client-API-How-To

## Hinweis

Die folgende Dokumentation basiert auf einer Preview-Version der Datendrehscheibe. Teile der Dokumentation können sich im Detail ändern.

## Funktionen der Client-API

Clients, wie bspw. Apps, DFI oder andere Systeme, können über die Client-API auf für Fahrgäste optimierte Funktionen und Daten zugreifen und bspw.

- alle Haltestellen mit ihrem geplanten und Ist-Fahrplan abrufen,
- alle Linien mit ihrem geplanten und Ist-Fahrplan abrufen,
- Fahrtverläufe abfragen,
- Stammdaten zu einer Linie erfragen,
- Stammdaten zu einer Haltestelle mit ihren Haltepunkten erfragen,
- Verbindungsauskunft erfragen

Die Client-API basiert auf modernen Technologien für die Integration unterschiedlichster Anwendungen über das Internet, die im weiteren Verlauf beschrieben werden.

## GraphQL als Schnittstellentechnologie

GraphQL ist eine zustandslose Abfragesprache, die es Clients ermöglicht, die genaue Struktur der benötigten Daten zu definieren. Durch diese Parametrisierung wird im Gegensatz zu REST vermieden, bei jeder Anfrage unnötig große Datenmengen zu übermitteln.

GraphQL unterstützt das Lesen (Queries), Schreiben (Mutations) und Abonnieren von Datenänderungen (Subscriptions).

Darüber hinaus bietet GraphQL mit einer Schema Definition Language (SDL) eine technisch verarbeitbare Beschreibung des Schemas an.

Der folgende Code zeigt exemplarisch die Beschreibung eines Schemas in der GraphQL SDL:

```
type Person {
  name: String
  age: Int
  friends: [Person]
}
```

Mit der SDL wird beschrieben, dass eine Person einen Namen als `String`, ein Alter als `Integer` und eine Liste von Personen als Freunde hat.

Ein Client kann nun eine Query formulieren

```
{
  person(name: "Kris") {
    age
    friends {
      name
    }
  }
}
```

```
}  
}
```

Hier wird nach der Person mit dem Namen "Kris" angefragt. Für diese Person soll das Alter und für alle Freunde der Name ausgegeben werden. Der Client erhält die Antwort als JSON-Dokument.

```
{  
  "person": {  
    "age": 43,  
    "friends": [  
      {  
        "name" : "Steve"  
      },  
      {  
        "name" : "Bill"  
      },  
      {  
        "name" : "Sarah"  
      },  
      {  
        "name" : "Laura"  
      }  
    ]  
  }  
}
```

Auf eine detaillierte Beschreibung der GraphQL wird verzichtet und auf die offizielle [Dokumentation](#) verwiesen.

Als Transportprotokoll kommen für Queries und Mutations bewährte Internet-Technologien wie HTTP und SSL und für Subscription WebSockets zum Einsatz.

Zur Authentisierung nutzt die Client-API OAuth.

## OAuth

OAuth (Open Authorization) ist ein offenes Protokoll, das eine standardisierte, sichere API-Autorisierung für Desktop-, Web- und Mobile-Anwendungen erlaubt. Die Client-API nutzt OAuth 2.0.

OAuth verwendet Tokens zur Autorisierung eines Zugriffs auf geschützte Ressourcen. Dadurch kann einem Client Zugriff auf geschützte Ressourcen gewährt werden, ohne die Zugangsdaten des Dienstes an den Client weitergeben zu müssen. Hierbei wird zwischen Access-Token und Refresh-Token unterschieden.

Um auf geschützte Daten auf dem Resource Server zuzugreifen, muss ein Access-Token vom Client als Repräsentation der Autorisierung übermittelt werden. Ein Refresh-Token kann dazu verwendet werden beim Authorization Server einen neuen Access-Token anzufragen, falls der Access-Token abgelaufen oder ungültig geworden ist. Der Refresh-Token hat ebenfalls eine zeitlich begrenzte Gültigkeit.

Es wird auf eine detaillierte Beschreibung des OAuth-2.0-Protokollflusses verzichtet und auf die offizielle [Dokumentation](#) verwiesen.

In der Datendrehscheibe kommen verschiedene Authorization Grant Types zum Einsatz. Diese werden beim On-Boarding eines Clients festgelegt. Für Apps und Drittsystem wird der client credentials Grant verwendet und für personenbezogenen Zugriff der password Grant Type.

Sobald ein Client ein Access Token erhalten hat, kann er dieses zur Bearer Authentisierung bei jedem Aufruf der Client-API senden.

## Client-Technologien

Um die Beispiele so praktisch wie möglich zu halten, müssen Annahmen über die Client-Technologie getroffen werden. Alle Beispiele werden sowohl als [Node.js](#)-Beispiele auf Basis der [Apollo-GraphQL-Client](#) - Bibliothek als auch [bash curl](#)-Beispiele bereitgestellt. Dies lässt sich auf verschiedenste Client-Technologien wie bspw. Java, C#, Swift, Go oder Python übertragen. Es wird auf die [Standard-Dokumentation](#) verwiesen, die auch einen Überblick über gängige Client-Bibliotheken für verschiedene Programmierumgebungen bietet.

## Beispiele

Im Folgenden werden typische Szenarien eines Clients dargestellt. Zu jedem Szenario wird der entsprechende Bash curl Aufruf und eine mögliche Node.JS-Implementierung aufgezeigt.

Die wenigen Beispiele können nicht die Mächtigkeit der Abfragemöglichkeiten des Schemas widerspiegeln und sollen vielmehr die Möglichkeiten der Client-API andeuten. Die angehängte Schemabeschreibung beinhaltet eine ausführliche Beschreibung aller `Types` und `Queries` der Client-API.

### Beispiel: Authenisierung

#### Voraussetzungen

Als Ergebnis des On-Boardings erhält der Client-Entwickler fünf wesentliche Informationen, die im Rahmen der Authentisierung und der Nutzung der Client-API benötigt werden. Die `OAuth2-URL`, die öffentliche `Client-Id`, das geheime `Client-Secret`, eine `Ressource-ID` und die eigentliche `Client-API-URL`.

Die folgende Übersicht liefert Beispiele für diese Werte. Die Beispiele sind nur Werte zur Verdeutlichung und stellen keine echten Werte dar.

```
OAuth2-URL=https://login.microsoftonline.com/57a0e625-d8c0-424f-8d87-8f18695c01f2/oauth2/token

CLIENT_ID=2598aca7-6aac-4dcb-8575-84c32c51ae6b

CLIENT_SECRET=7OftXwkfUXXxE9SXXXXTbvr9tKL05btLBkGTsFTggAw=

RESOURCE_ID=a9aaaaf6-89fd-4ac9-a6c1-daaa1b5637ab

CLIENT_API_URL=https://api.data-hub.rnv-online.de
```

#### Bash curl

Mit den Informationen kann über curl ein Access-Token angefordert werden.

```
curl --request POST \
  --url $OAuth2-URL \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data grant_type=client_credentials \
  --data client_id=$CLIENT_ID \
  --data client_secret=$CLIENT_SECRET \
  --data resource=$RESOURCE_ID
```

Die Antwort liefert ein JSON-Dokument, das das Access-Token im Feld `access_token` enthält.

```
{
  "token_type": "Bearer",
  "expires_in": "3599",
  "ext_expires_in": "3599",
  "expires_on": "1563040941",
  "not_before": "1563037041",
  "resource": "a9aaaaf6-89fd-4ac9-a6c1-daaa1b5637ab",
  "access_token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIngldCI6InU0T2ZOR1Bld0VCh3NIanRyYXVPY1Y4NExuWSIsImt9lFStI9dZYPCq9BKZNaWYeNMApDlFE0xRItorr0Io4HlhFPZU-NVUixumEMfLdyg90drZkl1jtN608Wjx5-dpV1jxb0Wgj18W35L1KnLPgZ71jnlNNDIVgZWgb6Y8nSxGE5y3B9WIAgaYK2HLQRtMxkEDsFumZproaXmMa8WJjvqpl7UEctG4fPuBwUKYof2bCoh3daJtP1sGRRBj0Oy75T9XBX_uv-R0g"
}
```

Im weiteren Verlauf wird angenommen, dass der Inhalt des Feldes `access_token` in einer Variable `ACCESS_TOKEN` gespeichert ist. Dies könnte bspw. durch das Parsen der Antwort mit Bash-Tools wie `jq` erfolgen.

## Node.JS

Der folgende Block stellt das Code-Gerüst dar, um eine Authentisierung mit Bibliotheken wie `apollo-client` und `apollo-link` durchzuführen. Der Übersicht halber wurde auf erweiterte Fehlerbehandlung und refresh- und Expiration-Logik verzichtet.

```
import { ApolloClient } from 'apollo-client';
import { InMemoryCache } from 'apollo-cache-inmemory';
import { HttpLink } from 'apollo-link-http';
import { ApolloLink } from 'apollo-link';
import { setContext } from 'apollo-link-context';
import gql from 'graphql-tag';
import fetch from 'node-fetch';

import { CLIENT_API_URL, OAUTH_URL, CLIENT_ID, CLIENT_SECRET, RESOURCE_ID } from
'./env';

const accessToken = async () => {

  const promise = new Promise(async (resolve, reject) => {

    const response = await fetch(OAUTH_URL, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      body:
`grant_type=client_credentials&client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&res

    });

    resolve(response.json());
  });
}
```

```

    })

    const json = await promise;
    return json["access_token"];
  }

  const httpLink = new HttpLink({
    uri: CLIENT_API_URL,
    credentials: 'same-origin',
    fetch: fetch
  });

  const authMiddleware = setContext((request) => new Promise(async (resolve, reject) =>
  {

    const token = await accessToken();
    resolve({
      headers: {
        authorization: `Bearer ${token}`,
      }
    })
  }));

  const client = new ApolloClient({
    link: ApolloLink.from([authMiddleware, httpLink]),
    cache: new InMemoryCache()
  });

```

## Suche nach Haltestellen

### Aufbau der Abfrage

Haltestellen werden über den Type `Station` abgebildet. Die Suche erfolgt über die Query `stations`, die eine Suche nach Namen und eine Suche nach Geokoordinaten ermöglicht.

Die folgende Query zeigt die `hafasID`, die `globalID` und den Namen der ersten 3 Haltestellen im Umkreis von 500 Meter um die Geokoordinaten `lat:49,483076 long:8,468409`.

```

{
  stations(first: 3 lat:49.483076 long:8.468409 distance:0.5) {
    totalCount
    elements {
      ... on Station {
        hafasID
        globalID
        longName
      }
    }
  }
}

```

Die Antwort hierfür wäre ein JSON Dokument mit folgendem Aufbau

```

{
  "data": {
    "stations": {
      "totalCount": 7,
      "elements": [
        {
          "hafasID": "2471",
          "globalID": "de:08222:2471",
          "longName": "Universität"
        },
        {
          "hafasID": "2466",
          "globalID": "de:08222:2466",
          "longName": "Tattersall"
        },
        {
          "hafasID": "2417",
          "globalID": "de:08222:2417",
          "longName": "MA Hauptbahnhof"
        }
      ]
    }
  },
  "extensions": {}
}

```

### Bash curl

```

curl $CLIENT_API_URL -X POST -H "authorization: Bearer $ACCESS_TOKEN" -H 'Content-Type: application/json' -H 'Accept: application/json' --data-binary '{"query":{"\n stations(first: 3 lat:49.483076 long:8.468409 distance:0.5) {\n   totalCount\n elements {\n   ... on Station {\n     hafasID\n     globalID\n     longName\n   }\n } }\n}'

```

### Node.JS

```

import { client } from "./client";
import gql from "graphql-tag";

client.query({
  query: gql`
    query {
      stations(first: 3 lat:49.483076 long:8.468409 distance:0.5) {
        totalCount
        elements {
          ... on Station {
            hafasID
            globalID
            longName
          }
        }
      }
    }
  `
})

```

```
    }  
  },  
}).then(result => console.log(result["data"]));
```

## Abfrage einer Haltestelle

### Aufbau der Abfrage

Eine einzelne Haltestelle wird über die Query `station` abgefragt, die als Parameter die `hafasID` benötigt. Die folgende Abfrage zeigt einige Stammdaten der Haltestelle `Universität`.

```
{  
  station(id:"2471") {  
    hafasID  
    longName  
    shortName  
    longName  
  }  
}
```

und liefert als JSON

```
{  
  "data": {  
    "station": {  
      "hafasID": "2471",  
      "longName": "Universität",  
      "shortName": "MIUN"  
    }  
  }  
}
```

### Bash curl

```
curl $CLIENT_API_URL -X POST -H "authorization: Bearer $ACCESS_TOKEN" -H 'Content-Type: application/json' -H 'Accept: application/json' --data-binary '{"query": "{\n station(id: \"2471\") {\n   hafasID\n   longName\n   shortName\n   longName\n }\n}"}'
```

### Node.JS

```
import { client } from "./client";  
import gql from "graphql-tag";  
  
client.query({  
  query: gql`  
    query {  
      station(id:"2471") {  
        hafasID  
        longName  
        shortName  
        longName  
      }  
    }  
  `
```

```

    }
    ,
  }) .then(result => console.log(result["data"]));

```

## Erstellung eines Abfahrtsmonitors

### Aufbau der Abfrage

Ein Abfahrtsmonitor kann nun über das Traversieren des Graphen aufgebaut werden. Die folgende Query zeigt die nächsten 5 Abfahrten am 4. Juni 2019 gegen 17:00 Uhr mit Soll- und Ist-Zeiten der Haltestelle MA Hauptbahnhof mit der hafasID 2417 an.

```

{
  station(id:"2417") {
    hafasID
    longName
    journeys(startTime: "2019-06-04T15:00:00Z" first: 5) {
      totalCount
      elements {
        ... on Journey {

          line {
            id
            description
          }

          stops(hafasID:"2417") {

            plannedDeparture {
              isoString
            }

            realtimeDeparture {
              isoString
            }
          }
        }
      }
    }
  }
}

```

und liefert eine Antwort

```

{
  "data": {
    "station": {
      "hafasID": "2417",
      "longName": "MA Hauptbahnhof",
      "journeys": {
        "totalCount": 52,
        "elements": [
          {

```



```
"line": {
  "id": "14-5A",
  "description": "5A"
},
"stops": [
  {
    "plannedDeparture": {
      "isoString": "2019-06-04T15:01:00.000Z"
    },
    "realtimeDeparture": null
  }
]
},
{
  "line": {
    "id": "60-60",
    "description": "60"
  },
  "stops": [
    {
      "plannedDeparture": {
        "isoString": "2019-06-04T15:01:00.000Z"
      },
      "realtimeDeparture": null
    }
  ]
},
{
  "line": {
    "id": "9-9",
    "description": "9"
  },
  "stops": [
    {
      "plannedDeparture": {
        "isoString": "2019-06-04T15:01:00.000Z"
      },
      "realtimeDeparture": {
        "isoString": "2019-06-04T15:01:00.000Z"
      }
    }
  ]
},
{
  "line": {
    "id": "8-8",
    "description": "8"
  },
  "stops": [
    {
      "plannedDeparture": {
        "isoString": null
      }
    }
  ]
}
```



```

elements {
  ... on Journey {

    line {
      id
      description
    }

    stops(hafasID:"2417") {

      plannedDeparture {
        isoString
      }

      realtimeDeparture {
        isoString
      }
    }
  }
}
}
}
}
}

}).then(result => console.log(result["data"]));

```

## Verbindungsankunft

### Aufbau der Abfrage

Die Client-API bietet neben der Beauskunftung der Linien, Haltestellen und Fahrten eine Verbindungsankunft. Die folgende Query fragt nach einer Verbindung zwischen der Universität und dem Hauptbahnhof mit einer geplanten Abfahrt um 17:15 Uhr am 17 Juni 2019.

```

{
  trips(origin:"de:08222:2471" destination:"de:08222:2417" departureTime:"2019-06-17T15:15:00Z") {
    startTime {
      isoString
    }

    endTime {
      isoString
    }

    interchanges

    legs {

      ... on InterchangeLeg {
        mode

      }
    }
  }
}

```

```
... on TimedLeg {
  board {
    point {
      ref
      name
    }
    estimatedTime {
      isoString
    }
    timetabledTime {
      isoString
    }
  }

  alight {
    point {
      ref
      name
    }

    estimatedTime {
      isoString
    }
    timetabledTime {
      isoString
    }
  }

  service {
    type
    name
    description
    destinationLabel
  }
}

... on ContinuousLeg {
  mode
}
}
}
```

und liefert als Antwort

```

{
  "data": {
    "trips": [
      {
        "startTime": {
          "isoString": "2019-06-17T15:14:00.000Z"
        },
        "endTime": {
          "isoString": "2019-06-17T15:16:00.000Z"
        },
        "interchanges": 0,
        "legs": [
          {
            "board": {
              "point": {
                "ref": "de:08222:2471:3:2",
                "name": "Mannheim, Universität"
              },
              "estimatedTime": {
                "isoString": null
              },
              "timetabledTime": {
                "isoString": "2019-06-17T15:14:00.000Z"
              }
            },
            "alight": {
              "point": {
                "ref": "de:08222:2417:3:Ri01",
                "name": "Mannheim, Hauptbahnhof"
              },
              "estimatedTime": {
                "isoString": null
              },
              "timetabledTime": {
                "isoString": "2019-06-17T15:16:00.000Z"
              }
            },
            "service": {
              "type": "TRAM",
              "name": "RNV 5",
              "description": "Weinheim - Heidelberg - Mannheim - Viernheim - Weinheim"
            },
            "destinationLabel": "Mannheim Hbf - Käfertal"
          }
        ]
      },
      {
        "startTime": {
          "isoString": "2019-06-17T15:15:00.000Z"
        },
        "endTime": {

```

```

    "isoString": "2019-06-17T15:17:00.000Z"
  },
  "interchanges": 0,
  "legs": [
    {
      "board": {
        "point": {
          "ref": "de:08222:2471:3:2",
          "name": "Mannheim, Universität"
        },
        "estimatedTime": {
          "isoString": null
        },
        "timetabledTime": {
          "isoString": "2019-06-17T15:15:00.000Z"
        }
      },
      "alight": {
        "point": {
          "ref": "de:08222:2417:3:Rio2",
          "name": "Mannheim, Hauptbahnhof"
        },
        "estimatedTime": {
          "isoString": null
        },
        "timetabledTime": {
          "isoString": "2019-06-17T15:17:00.000Z"
        }
      },
      "service": {
        "type": "BUS",
        "name": "RNV 60",
        "description": "MA Pfeifferswörth - Ulmenweg - Herzogenried -
Neckarstadt West - Hafenstraße - Schloss - Universität - MA Hauptbahnhof - MA
Oststadt",
        "destinationLabel": "Mannheim, Lanzvilla"
      }
    }
  ]
},
{
  "startTime": {
    "isoString": "2019-06-17T15:18:00.000Z"
  },
  "endTime": {
    "isoString": "2019-06-17T15:20:00.000Z"
  },
  "interchanges": 0,
  "legs": [
    {
      "board": {
        "point": {

```

```

    "ref": "de:08222:2471:3:2",
    "name": "Mannheim, Universität"
  },
  "estimatedTime": {
    "isoString": null
  },
  "timetabledTime": {
    "isoString": "2019-06-17T15:18:00.000Z"
  }
},
"alight": {
  "point": {
    "ref": "de:08222:2417:3:Ri01",
    "name": "Mannheim, Hauptbahnhof"
  },
  "estimatedTime": {
    "isoString": null
  },
  "timetabledTime": {
    "isoString": "2019-06-17T15:20:00.000Z"
  }
},
"service": {
  "type": "TRAM",
  "name": "RNV 5A",
  "description": "Abendakademie - Paradeplatz - MA Hauptbahnhof -
Universitätsklinikum - Käfertal Bf - Heddesheim",
  "destinationLabel": "Heddesheim, Bahnhof (RNV)"
}
}
]
},
{
  "startTime": {
    "isoString": "2019-06-17T15:21:00.000Z"
  },
  "endTime": {
    "isoString": "2019-06-17T15:23:00.000Z"
  },
  "interchanges": 0,
  "legs": [
    {
      "board": {
        "point": {
          "ref": "de:08222:2471:3:2",
          "name": "Mannheim, Universität"
        },
        "estimatedTime": {
          "isoString": null
        },
        "timetabledTime": {
          "isoString": "2019-06-17T15:21:00.000Z"
        }
      }
    }
  ]
}

```

```

    }
  },
  "alight": {
    "point": {
      "ref": "de:08222:2417:3:RiO2",
      "name": "Mannheim, Hauptbahnhof"
    },
    "estimatedTime": {
      "isoString": null
    },
    "timetabledTime": {
      "isoString": "2019-06-17T15:23:00.000Z"
    }
  },
  "service": {
    "type": "TRAM",
    "name": "RNV 1",
    "description": "MA Schönau - Waldhof - Paradeplatz - MA Hbf - Neckarau -
MA Rheinau Bf",
    "destinationLabel": "MA-Rheinau, Bahnhof"
  }
}
],
{
  "startTime": {
    "isoString": "2019-06-17T15:15:00.000Z"
  },
  "endTime": {
    "isoString": "2019-06-17T15:21:30.000Z"
  },
  "interchanges": 0,
  "legs": [
    {
      "mode": "WALK"
    }
  ]
}
]
}
}

```

## Bash curl

```

curl $CLIENT_API_URL -X POST -H "authorization: Bearer $ACCESS_TOKEN" -H 'Content-
Type: application/json' -H 'Accept: application/json' --data-binary '{"query":{"\n
trips(origin:"de:08222:2471\" destination:"de:08222:2417\" departureTime:"2019-06-
17T15:15:00Z\") {\n  startTime {\n    isoString\n  }\n  \n  endTime {\n
isoString\n  }\n  \n  interchanges\n  \n  legs {\n    \n    ... on
InterchangeLeg {\n      mode\n      \n    }\n    \n    ... on TimedLeg {\n
board {\n      point {\n        ref\n        name\n      }\n
estimatedTime {\n      isoString\n      }\n      timetabledTime {\n

```



```

isoString\n          }\n          }\n          \n          alight {\n          point {\n
ref\n              name\n          }\n          \n          estimatedTime {\n
isoString\n          }\n          timetabledTime {\n          isoString\n
}\n          \n          \n          }\n          \n          service {\n          type\n
name\n          description\n          destinationLabel\n          }\n          \n
\n          }\n          \n          ... on ContinuousLeg {\n          mode\n          }\n          }\n
}\n\n}"'

```

## Node.JS

```

import { client } from "./client";
import gql from "graphql-tag";

client.query({
  query: gql`
    query {
      {
        trips(origin:"de:08222:2471" destination:"de:08222:2417"
departureTime:"2019-06-17T15:15:00Z") {
          startTime {
            isoString
          }

          endTime {
            isoString
          }

          interchanges

          legs {

            ... on InterchangeLeg {
              mode
            }

            ... on TimedLeg {
              board {
                point {
                  ref
                  name
                }
              }
              estimatedTime {
                isoString
              }
              timetabledTime {
                isoString
              }
            }

            alight {
              point {
                ref

```

```

        name
      }

      estimatedTime {
        isoString
      }
      timetabledTime {
        isoString
      }
    }

    service {
      type
      name
      description
      destinationLabel
    }
  }

  ... on ContinuousLeg {
    mode
  }
}
}
}
}
}

}).then(result => console.log(result["data"]));

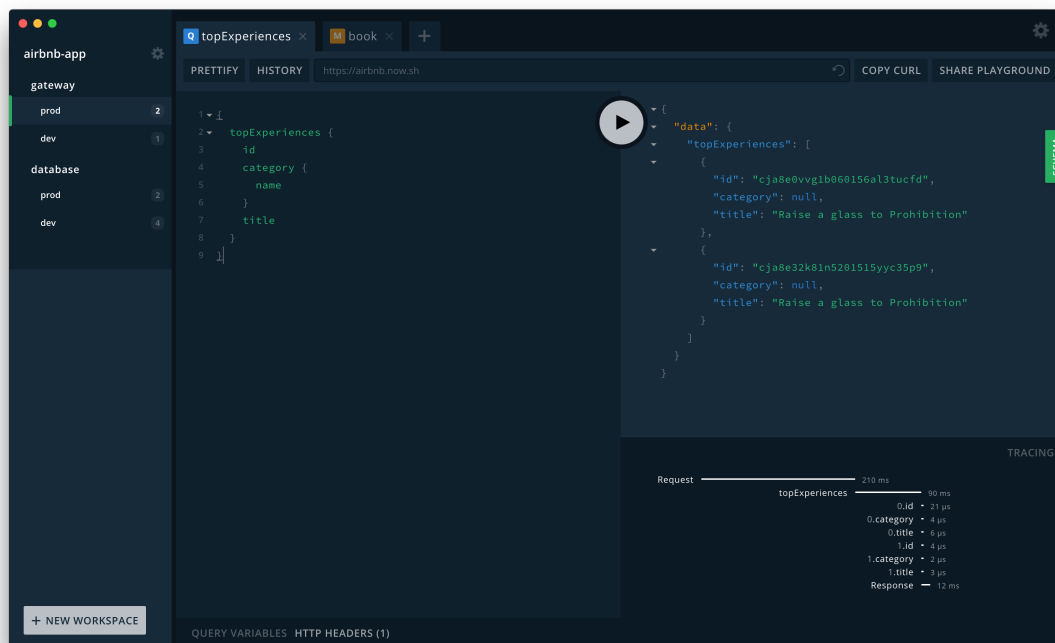
```

## Entwicklungshinweise

Im Folgenden werden Best practices und Konzepte vorgestellt, die die Entwicklung mit der Client-API wesentlich vereinfachen könnten.

### Playground

Tools wie der [GraphQL Playground](#) ermöglichen ein schnelles Prototyping und Erforschen einer GraphQL-API. Der Playground bietet zudem den Zugriff auf das integrierte Schema und die integrierte Schema-Dokumentation.



Das Access-Token könnte bspw. über curl erfragt und als HTTP-Header hinterlegt werden. So könnten Client-Entwickler die Client-API kennenlernen und die Abfragen für ihre Anwendungsfälle vorab aufbauen.

## Verschachtelung

Eine GraphQL-API ermöglicht eine beliebige Verschachtelung. So fragt die folgende Query zunächst für die Haltestelle mit der hafasID 2471 den ersten Mast und den Mast wieder nach der Haltestelle, in diesem Fall wieder die 2471, die wieder den ersten Mast abfragt. Diese Abfrage könnte beliebig tief verschachtelt werden.

```
{
  station(id:"2471") {
    hafasID
    poles(first: 1) {
      elements {
        ... on Pole {
          station {
            hafasID
            poles(first: 1) {
              elements {
                ... on Pole {
                  station {
                    hafasID
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
    }
  }
}
```

Die Datendrehscheibe versucht durch intelligentes Caching die Last auf ein Minimum zu reduzieren. Zudem zeichnet die Client-API pro Aufruf einen so genannten Request-Charge auf und bricht zu tief verschachtelte Abfragen ab, sofern diese die Charge-Grenze überschreiten. Falls ein Client wider Erwarten äußerst tief verschachtelte Abfragen benötigt, kann die Charge Grenze im Rahmen des On-Boardings individuell pro Client erweitert werden.

## Pagination

Die Client-API nutzt bei vielen Queries ein Pagniationkonzept. Die folgende Abfrage listet bspw. alle Linien.

```
{
  lines {
    totalCount
  }
}
```

```
{
  "data": {
    "lines": {
      "totalCount": 339
    }
  }
}
```

Jede Liste enthält die Gesamtanzahl der Treffermenge. In diesem Fall 339. Möchte ein Client auf Informationen zugreifen, müssen diese über das `elements` Element abgefragt werden. Die folgende Abfrage liefert die id der Linien.

```
{
  lines {
    totalCount
    elements {
      ... on Line {
        id
      }
    }
  }
}
```

Sofern der Client nichts weiter angibt, werden nur die ersten 10 Elemente übertragen. Ein Client kann die Anzahl der übertragenen Elemente explizit angeben. In dem folgenden Beispiel werden nur die ersten 3 Linien angefragt.

```
{
  lines(first:3) {
    totalCount
    elements {
      ... on Line {
```

```
    id
  }
}
}
```

Möchte ein Client nun durch alle 339 Linien iterieren, kann er sich für eine Abfrage einen `cursor` ausgeben lassen.

```
{
  lines(first:3) {
    totalCount
    elements {
      ... on Line {
        id
      }
    }
    cursor
  }
}
```

```
{
  "data": {
    "lines": {
      "totalCount": 339,
      "elements": [
        {
          "id": "1-1"
        },
        {
          "id": "1-B"
        },
        {
          "id": "10-10"
        }
      ],
      "cursor": "10741005-10-10"
    }
  }
}
```

Den Inhalt des Cursors kann ein Client bei der nächsten Abfrage angeben, um die nächsten 3 Linien zu erfragen.

```
{
  lines(first:3 after:"10741005-10-10") {
    totalCount
    elements {
      ... on Line {
        id
      }
    }
  }
}
```

```
    cursor
  }
}
```

Mit diesem Pagniationkonzept könnte ein Client durch große Datenmengen iterieren.

## Datum

Ein herzuhebender Datentyp ist ein Zeitstempel. Dieser ist im Schema wie folgt definiert

```
"""Ein Zeitstempel"""
type Time {
  """Tag, beginnend bei 1 ... 31"""
  date: Int!
  """Monat, beginnend bei 1 ... 12"""
  month: Int!
  """Vierstellig, bspw. 2019"""
  year: Int!
  """Stunden, beginnend bei 0 ... 23"""
  h: Int!
  """Minuten, beginnend bei 0 ... 59"""
  m: Int!
  """Sekunden, beginnend bei 0 ... 59"""
  s: Int!
  """Anzahl der Millisekunden seit dem 1.1.1970T00:00:00 UTC+0"""
  x: Int
  """Anzahl der Sekunden seit dem 1.1.1970T00:00:00 UTC+0"""
  X: Int
  """UTC Offset, bspw +2 für MESZ oder +0"""
  offSet: Int
  """Repräsentation der Zeit als Zeichenkette nach ISO 8601"""
  isoString: String
}
```

Alle Zeiten werden in UTC+0 angegeben. Es obliegt dem Client, die korrekte Umrechnung in die lokale Zeitzone durchzuführen.

## Anlagen

Diese How-To wird ergänzt durch:

- Node.JS-Beispiel-Code
- GraphQL-SDL der Client-API