Datendrehscheibe Client-API-How-To

Hinweis

> *Die folgende Dokumentation basiert auf einer Beta-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 Subscriptions 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. Letztere können nur auf Linux ausgeführt werden. Neben curl, wird dabei auch das Kommandozeilentool jq benötigt. Das Vorgehen lässt sich aber auf verschiedenste Client-Technologien wie bspw. Java, C#, Swift, Go oder Python übertragen. Es wird auf die Standard-Dokumentation erwiesen, 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: Authentisierung

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. Sobald die API-Key Datei vorhanden ist, kann diese mit dem folgendem Skript verwendet werden ums sich einen Access Token über die OAuth-Schnittstelle zu holen.

```
#!/bin/bash

tenantID=$(awk '{if ( $0 ~ /^@tenantID=/ ) {print $0}}' graphql.http | cut -d'=' -
f2)
OAUTH_URL=https://login.microsoftonline.com/${tenantID}/oauth2/token
CLIENT_ID=$(awk '{if ( $0 ~ /^@clientID=/ ) {print $0}}' graphql.http | cut -d'=' -
f2)
CLIENT_SECRET=$(awk '{if ( $0 ~ /^@clientSecret=/ ) {print $0}}' graphql.http | cut
-d'=' -f2)
RESOURCE_ID=$(awk '{if ( $0 ~ /^@resource=/ ) {print $0}}' graphql.http | cut -d'='
-f2)
CLIENT_API_URL=https://graphql-sandbox-dds.rnv-online.de/

echo "OAUTH_URL = $OAUTH_URL"
echo "CLIENT_ID = $CLIENT_ID"
```

```
echo "CLIENT_SECRET = $CLIENT_SECRET"
echo "RESOURCE_ID = $RESOURCE_ID"
echo "CLIENT_API_URL = $CLIENT_API_URL"
```

Bash curl

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

```
ACCESS_TOKEN=$(curl $OAUTH_URL -X POST \
                          -H "Content-Type: application/x-www-form-urlencoded" \
                              -d grant_type=client_credentials \
                              -d client_id=$CLIENT_ID \
                              -d client_secret=$CLIENT_SECRET \
                      -d resource=$RESOURCE_ID)

ACCESS_TOKEN=$(echo ${ACCESS_TOKEN} | sed -s 's/,/ /g' | cut -d' ' -f7 | cut -d':' -
f2 | cut -d'}' -f1 | sed -s 's/"//g')
echo
echo "ACCESS_TOKEN \"${ACCESS_TOKEN}\""
echo
```

Die Antwort das `ACCESS_TOKEN` .

```
ACCESS_TOKEN "eyJ0eXAiOiJKV1QiLCJh ... GRRBj0Oy75T9XBX_uv-R0g"
```

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. Es wird davon ausgegangen, dass die Variablen `CLIENT_ID` , `CLIENT_SECRET` und `RESOURCE_ID` in einer `env.js` festgelegt wurden. Diese können der `graphql.http` entnommen werden. Außerdem sollten dort die Variablen `CLIENT_API_URL` und `OAUTH_URL` gesetzt werden. Deren Zusammensetzung erfolgt hier analog zum bash Script und kann diesem entnommen werden.

```
import pkg from '@apollo/client/core/core.cjs';
const { ApolloClient, InMemoryCache, ApolloLink, HttpLink } = pkg;
import { setContext } from '@apollo/client/link/context/context.cjs';
import fetch from 'node-fetch';
import {
  CLIENT_API_URL,
  OAUTH_URL,
  CLIENT_ID,
  CLIENT_SECRET,
  RESOURCE_ID,
} from './env.js';

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}&resource=${RESOURCE_ID}`,
    });

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

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

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

const authMiddleware = setContext(
  request =>
    new Promise(async (resolve, reject) => {
      const token = await accessToken();

      resolve({
        headers: {
          authorization: `Bearer ${token}`,
        },
      });
    })
);
export 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": "Mannheim Hauptbahnhof"
        }
      ]
    }
  }
}
```


Bash curl

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

Node.JS

```
import { client } from './client.js';
import gql from 'graphql-tag';

const result = 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(JSON.stringify(result['data'], null, 2)));
```

Abfrage einer Haltestelle

Aufbau der Abfrage

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

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

und liefert als JSON

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

Bash curl

```
curl $CLIENT_API_URL -X POST -H "Authorization: Bearer ${ACCESS_TOKEN}" -H "Content-Type: application/json" -H "Accept: application/json" -d '{"query": "{ station(id: \"2471\") { hafasID longName shortName name } }" }' | jq .
```

Node.JS

```
import { client } from './client.js';
import gql from 'graphql-tag';

client
  .query({
    query: gql`
      query {
        station(id: "2471") {
          hafasID
          longName
          shortName
          name
        }
      }
    `,
  })
  .then(result => console.log(JSON.stringify(result['data'], null, 2)));
```

Erstellung eines Abfahrtsmonitors

Aufbau der Abfrage

Ein Abfahrtsmonitor kann nun über das Traversieren des Graphen aufgebaut werden. Die folgende `Query` zeigt die nächsten 2 Abfahrten am 26.02.24 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: "2024-02-26T17:00:00Z", first: 2) {
      totalCount
      elements {
        ... on Journey {
          line {
            id
          }

          stops(onlyHafasID: "2417") {
            plannedDeparture {
              isoString
            }

            realtimeDeparture {
```

```
              isoString
            }
          }
        }
      }
    }
  }
}
```

und liefert eine Antwort


```
{
  "data": {
    "station": {
      "hafasID": "2417",
      "longName": "Mannheim Hauptbahnhof",
      "journeys": {
        "totalCount": 40,
        "elements": [
          {
            "line": {
              "id": "8-8"
            },
            "stops": [
              {
                "plannedDeparture": {
                  "isoString": null
                },
                "realtimeDeparture": {
                  "isoString": null
                }
              }
            ]
          },
          {
            "line": {
              "id": "14-5A"
            },
            "stops": [
              {
                "plannedDeparture": {
                  "isoString": "2024-02-26T17:01:00.000Z"
                },
                "realtimeDeparture": {
                  "isoString": "2024-02-26T17:03:20.000Z"
                }
              }
            ]
          }
        ]
      }
```

```
    }
  }
}
```

Bash curl

```
QUERY="{\"query\": \"{ station(id: \\\"2417\\\") { hafasID longName
journeys(startTime: \\\"2024-02-26T17:00:00Z\\\" first: 2) { totalCount elements {
... on Journey { line { id } stops(onlyHafasID: \\\"2417\\\") { plannedDeparture {
isoString } realtimeDeparture { isoString } } } } } } }\" }"

curl $CLIENT_API_URL -X POST -H "Authorization: Bearer ${ACCESS_TOKEN}" -H "Content-
Type: application/json" -H "Accept: application/json" -d "${QUERY}" | jq .
```

Node.JS

```
import { client } from './client.js';
import gql from 'graphql-tag';

client
  .query({
    query: gql`
      query {
        station(id: "2417") {
          hafasID
          longName
          journeys(startTime: "2024-02-26T17:00:00Z", first: 2) {
            totalCount
            elements {
              ... on Journey {
                line {
                  id
                }

                stops(onlyHafasID: "2417") {
                  plannedDeparture {
                    isoString
                  }

                  realtimeDeparture {
                    isoString
                  }
                }
              }
            }
          }
        }
      }
    }
    `,
```

```
  })
  .then(result => console.log(JSON.stringify(result['data'], null, 2)));
```

Abfrage der Auslastungsinformation (über Abfahrtsmonitor)

Aufbau der Abfrage

Die Auslastungsinformation einer Fahrt kann über einen Abfahrtsmonitor abgefragt werden. Die folgende `Query` zeigt die nächsten 3 Abfahrten an der Haltestelle `MA Hauptbahnhof` mit der hafasID `2417` mit den jeweiligen statistischen oder Echtzeit-Besetztgradzahlen, die prozentuale Besetzung im Fahrzeug sowie die Auslastungsstufe an.

```
{
  station(id: "2417") {
    hafasID
    longName
    journeys(startTime: "2024-02-26T17:00:00Z", first: 3) {
      totalCount
      elements {
        ... on Journey {
          id
          line {
            id
          }
          loadsForecastType

          loads(onlyHafasID: "2417") {
            realtime
            forecast
            adjusted
            loadType
            ratio
          }

          stops(only: "2417") {
            plannedDeparture {
              isoString
            }

            realtimeDeparture {
              isoString
            }
          }
        }
      }
    }
  }
}
```

und liefert als Antwort

```
{
  "data": {
    "station": {
      "hafasID": "2417",
      "longName": "Mannheim Hauptbahnhof",
      "journeys": {
        "totalCount": 40,
        "elements": [
          {
            "id": "57-830-01023-1",
            "line": {
              "id": "8-8"
            },
            "loadsForecastType": "PARTIALLY_REALTIME",
            "loads": [
              {
                "realtime": 0,
                "forecast": 0,
                "adjusted": 0,
                "loadType": "I",
                "ratio": 0
              }
            ],
            "stops": [
              {
                "plannedDeparture": {
                  "isoString": null
                },
                "realtimeDeparture": {
                  "isoString": null
                }
              }
            ]
          },
          {
            "id": "57-475455-01023-1",
            "line": {
              "id": "14-5A"
            },
            "loadsForecastType": "REALTIME",
            "loads": [
              {
                "realtime": 97,
                "forecast": 44.50958284461058,
                "adjusted": 97,
                "loadType": "III",
                "ratio": 0.6423841059602649
              }
            ],
            "stops": [
              {
```

```
              "plannedDeparture": {
                "isoString": "2024-02-26T17:01:00.000Z"
              },
              "realtimeDeparture": {
                "isoString": "2024-02-26T17:03:20.000Z"
              }
            }
          ]
        },
        {
          "id": "57-10616-01021-1",
          "line": {
            "id": "60-60"
          },
          "loadsForecastType": "PARTIALLY_REALTIME",
          "loads": [
            {
              "realtime": null,
              "forecast": 6.661827390354489,
              "adjusted": 16,
              "loadType": "I",
              "ratio": 0.2
            }
          ],
          "stops": [
            {
              "plannedDeparture": {
                "isoString": "2024-02-26T17:02:00.000Z"
              },
              "realtimeDeparture": {
                "isoString": "2024-02-26T17:02:00.000Z"
              }
            }
          ]
        }
      ]
    }
  }
}
```


Weitere Hinweise zu den einzelnen Attributen:

- `loadsForecastType` – gibt an, ob es sich um Statistik, Echtzeit oder teilweise Echtzeit handelt
- `forecast` – statistische Prognose der Anzahl an Personen für diese Fahrt, ermittelt durch statistischen Prognose-Algorithmus
- `realtime` – die tatsächliche Anzahl an Personen, wenn Echtzeitdaten der Sensoren im Fahrzeug vorliegen
- `adjusted` – im Falle von statistischen Fahrten gerundeter Wert von forecast. Im Falle von Echtzeitdaten, durch Echtzeitdaten angepasster Wert
- `loadType` – Auslastungsstufe der Fahrt, berechnet aus dem Attribut ratio

- `ratio` – prozentualer Anteil an Personen im Fahrzeug im Verhältnis zur Kapazität

Bash curl

```
QUERY="{\"query\": \"{station(id: \\\"2417\\\") {hafasID longName
journeys(startTime: \\\"2024-02-26T17:00:00Z\\\", first: 3) {totalCount elements
{... on Journey {id line {id} loadsForecastType loads(onlyHafasID: \\\"2417\\\")
{realtime forecast adjusted loadType ratio} stops(only: \\\"2417\\\")
{plannedDeparture {isoString} realtimeDeparture {isoString}}}}}}\" }"

curl $CLIENT_API_URL -X POST -H "Authorization: Bearer ${ACCESS_TOKEN}" -H "Content-
Type: application/json" -H "Accept: application/json" -d "${QUERY}" | jq .
```

Node.JS

```
import { client } from './client.js';
import gql from 'graphql-tag';

client
  .query({
    query: gql`
      query {
        station(id: "2417") {
          hafasID
          longName
          journeys(startTime: "2024-02-26T17:00:00Z", first: 3) {
            totalCount
            elements {
              ... on Journey {
                id
                line {
                  id
                }
                loadsForecastType

                loads(onlyHafasID: "2417") {
                  realtime
                  forecast
                  adjusted
                  loadType
                  ratio
                }

                stops(only: "2417") {
                  plannedDeparture {
                    isoString
                  }

                  realtimeDeparture {
                    isoString
                  }
```

```
              }
            }
          }
        }
      }
    }
  `,
})
.then(result => console.log(JSON.stringify(result['data'], null, 2)));
```

Verbindungsauskunft

Aufbau der Abfrage

Die Client-API bietet neben der Auskunft zu Linien, Haltestellen und Fahrten eine Verbindungsauskunft. Die folgende Query fragt nach einer Verbindung zwischen der Universität und dem Hauptbahnhof mit einer geplanten Abfahrt um 17:00 Uhr am 26.02.24.

```
{
  trips(
    originGlobalID: "de:08222:2471"
    destinationGlobalID: "de:08222:2417"
    departureTime: "2024-02-26T17:00:00Z"
  )
  {
    startTime {
      isoString
    }
    endTime {
      isoString
    }

    interchanges

    legs {
      ... on InterchangeLeg {
        mode
      }

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

```
          }
        }

        alight {
          point {
            ... on StopPoint {
              ref
              stopPointName
            }
          }

          estimatedTime {
            isoString
          }
          timetabledTime {
            isoString
          }
        }

        service {
          type
          name
          description
          destinationLabel
        }
      }

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

und liefert als Antwort


```
{
  "data": {
    "trips": [
      {
        "startTime": {
          "isoString": "2024-02-26T17:01:00.000Z"
        },
        "endTime": {
          "isoString": "2024-02-26T17:03:00.000Z"
        },
        "interchanges": 0,
        "legs": [
          {
            "board": {
              "point": {
```

```
              "ref": "de:08222:2471:3:2",
              "stopPointName": "Mannheim, Universität"
            },
            "estimatedTime": {
              "isoString": null
            },
            "timetabledTime": {
              "isoString": "2024-02-26T17:01:00.000Z"
            }
          },
          "alight": {
            "point": {
              "ref": "de:08222:2417:3:RiO2",
              "stopPointName": "Mannheim, Hauptbahnhof"
            },
            "estimatedTime": {
              "isoString": null
            },
            "timetabledTime": {
              "isoString": "2024-02-26T17:03:00.000Z"
            }
          },
          "service": {
            "type": "STRASSENBAHN",
            "name": "RNV 1",
            "description": "MA Schönau - Waldhof - Paradeplatz - MA Hbf - Neckarau
- MA Rheinau Bf",
            "destinationLabel": "MA-Rheinau, Bahnhof"
          }
        }
      ]
    },
    {
      "startTime": {
        "isoString": "2024-02-26T17:03:00.000Z"
      },
      "endTime": {
        "isoString": "2024-02-26T17:05:00.000Z"
      },
      "interchanges": 0,
      "legs": [
        {
          "board": {
            "point": {
              "ref": "de:08222:2471:3:2",
              "stopPointName": "Mannheim, Universität"
            },
            "estimatedTime": {
              "isoString": null
            },
            "timetabledTime": {
              "isoString": "2024-02-26T17:03:00.000Z"
```

```
            }
          },
          "alight": {
            "point": {
              "ref": "de:08222:2417:3:RiO1",
              "stopPointName": "Mannheim, Hauptbahnhof"
            },
            "estimatedTime": {
              "isoString": null
            },
            "timetabledTime": {
              "isoString": "2024-02-26T17:05:00.000Z"
            }
          },
          "service": {
            "type": "STRASSENBAHN",
            "name": "RNV 4",
            "description": "Bad Dürkheim - LU Oggersheim - LU Hbf - Berliner Platz
- MA Hbf - Wasserturm - Paradeplatz - Bonifatiuskirche - Hessische Str. -
Waldfriedhof",
            "destinationLabel": "Waldhof, Waldfriedhof"
          }
        }
      ]
    },
    {
      "startTime": {
        "isoString": "2024-02-26T17:04:00.000Z"
      },
      "endTime": {
        "isoString": "2024-02-26T17:06:00.000Z"
      },
      "interchanges": 0,
      "legs": [
        {
          "board": {
            "point": {
              "ref": "de:08222:2471:3:2",
              "stopPointName": "Mannheim, Universität"
            },
            "estimatedTime": {
              "isoString": null
            },
            "timetabledTime": {
              "isoString": "2024-02-26T17:04:00.000Z"
            }
          },
          "alight": {
            "point": {
              "ref": "de:08222:2417:3:RiO2",
              "stopPointName": "Mannheim, Hauptbahnhof"
            },
```

```
              "estimatedTime": {
                "isoString": null
              },
              "timetabledTime": {
                "isoString": "2024-02-26T17:06:00.000Z"
              }
            },
            "service": {
              "type": "STRASSENBAHN",
              "name": "RNV 6",
              "description": "LU Rheingönheim - Am Schwanen - LU Berliner Platz - MA
Neuostheim",
              "destinationLabel": "Neuhermsheim, SAP Arena S-Bf. (RNV)"
            }
          }
        ]
      },
      {
        "startTime": {
          "isoString": "2024-02-26T17:00:00.000Z"
        },
        "endTime": {
          "isoString": "2024-02-26T17:07:36.000Z"
        },
        "interchanges": 0,
        "legs": [
          {
            "mode": "WALK"
          }
        ]
      }
    ]
  }
}
```


Bash curl

```
QUERY="{\"query\": \"{ trips(originGlobalID: \\\"de:08222:2471\\\"
destinationGlobalID: \\\"de:08222:2417\\\" departureTime: \\\"2024-02-
26T17:00:00Z\\\" ) { startTime { isoString } endTime { isoString } interchanges legs
{ ... on InterchangeLeg { mode } ... on TimedLeg { board { point { ... on StopPoint
{ ref stopPointName } } estimatedTime { isoString } timetabledTime { isoString } }
alight { point { ... on StopPoint { ref stopPointName } } estimatedTime { isoString
} timetabledTime { isoString } } service { type name description destinationLabel }
} ... on ContinuousLeg { mode } } } }\" }"
curl $CLIENT_API_URL -X POST -H "Authorization: Bearer ${ACCESS_TOKEN}" -H "Content-
Type: application/json" -H "Accept: application/json" -d "${QUERY}" | jq .
```

Node.JS

```
import { client } from './client.js';
import gql from 'graphql-tag';

client
  .query({
    query: gql`
      query {
        trips(
          originGlobalID: "de:08222:2471"
          destinationGlobalID: "de:08222:2417"
          departureTime: "2024-02-26T17:00:00Z"
        ) {
          startTime {
            isoString
          }

          endTime {
            isoString
          }

          interchanges

          legs {
            ... on InterchangeLeg {
              mode
            }

            ... on TimedLeg {
              board {
                point {
                  ... on StopPoint {
                    ref
                    stopPointName
                  }
                }
                estimatedTime {
                  isoString
                }
                timetabledTime {
                  isoString
                }
              }

              alight {
                point {
                  ... on StopPoint {
                    ref
                    stopPointName
                  }
                }
```

```
              estimatedTime {
                isoString
              }
              timetabledTime {
                isoString
              }
            }

            service {
              type
              name
              description
              destinationLabel
            }
          }

          ... on ContinuousLeg {
            mode
          }
        }
      }
    }
  `,
})
.then(result => console.log(JSON.stringify(result['data'], null, 2)));
```

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. Es gibt darüber hinaus weitere Tools, wie bspw. Insomnia, die eine ausgereifte GraphQL-Unterstützung anbieten.

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": 342
    }
  }
}
```

Jede Liste enthält die Gesamtanzahl der Treffer. In diesem Fall 342. 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 342 Linien iterieren, kann er sich für eine Abfrage einen `cursor` ausgeben lassen.

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

```
{
  "data": {
    "lines": {
      "totalCount": 342,
      "elements": [
        {
          "id": "1-1"
        },
```

```
        {
          "id": "1-E"
        },
        {
          "id": "10-10"
        }
      ],
      "cursor": "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:"10-10") {
    totalCount
    elements {
      ... on Line {
        id
      }
    }
    cursor
  }
}
```

Mit diesem Pagniationkonzept könnte ein Client durch große Datenmengen iterieren.

Datum

Ein hervorzuhebender 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.