ridesharing.api

ridesharing.api 0.1

Spezifikation einer einheitlichen Schnittstelle für Mitfahr-Portale.

Version dev 01.01.2019

Inhaltsverzeichnis

1 Einleitung

Der Mitfahrmarkt in Deutschland ist stark fragmentiert und hierdurch für die Nutzer der vielfältigen Vermittlungsangebote nicht ausreichend transparent. Fahrer und Mitfahrer finden zum Teil nicht oder nur mit großem Zeitaufwand zusammen. Dieses Problem wollen wir mit einem offenen Datenstandard und der Perspektive auf ein gemeinsames Meta-Such-Portal angehen.

1.1 Zielsetzung

Dass Mobilität durch eine bessere Vernetzung verschiedener Verkehrsmittel schneller, komfortabler und umweltfreundlicher werden kann ist mittlerweile allgemeiner Konsens. Doch an dem „wie“ scheiden sich oft die Geister. Eines der Kernprobleme ist, dass fast jeder Anbieter von Mobilitätslösungen in Deutschland einen anderen Datenstandard nutzt.

Dies ist auch imbeim Mitfahrmarkt nicht anders. Dabei wäre gerade der Mitfahrmarkt besonders auf eine Vernetzung angewiesen, wie ein früheres Forschungsprojekt und die darauf folgenden Entwicklungen des Mitfahrmarkts in den letzten Jahren verdeutlicht haben. Die große Anzahl an Vermittlungsangeboten erhöht den Aufwand für das Zusammenfinden und vermindert mögliche Vermittlungserfolge. Ein Metaportal könnte dieses roblem lösen, doch dies ist nur möglich, wenn die Daten mindestens strukturiert, im besten Fall in einem einheitlichen Datenstandard vorliegen.

Vergleichbar ist die Situation mit der menschlichen Kommunikation: Möchte man komplexe Informationen austauschen, so ist es zwangsweise erforderlich, dass man die Sprache des Anderen beherrscht. Ansonsten ist Kommunikation nur mit hohem Aufwand und dem Verlust vieler Details möglich. Genauso verhält es sich in der Mobilität: Eine gut funktionierende Vernetzung gibt es nur mit gemeinsamer Sprache, also einem gemeinsamen Datenstandard.

Wichtig ist, dass es sich dabei um einen offenen Datenstandard handelt, der ohne Lizenzgebühren einsehbar und nutzbar ist. Die aktuelle unbefriedigende Situation ist unter anderem auf geschlossene Standards zurückzuführen – ein Datenaustausch zwischen den Plattformen findet zur Zeit nahezu nicht statt. Das Internet selbst sollte dabei Vorbild sein: Nur durch die vielen offenen Standards ist die Vernetzung möglich geworden. Und auch hier wieder gilt die Analogie zur menschlichen Sprache: Die Idee, für jede Nutzung eines deutschen Wortes eine Lizenzgebühr zu verlangen, würde zu Recht als völlig absurd abgetan werden.

1.2 Nutzungsszenarien

Ein Datenstandard für den Mitfahrmarkt hat zahlreiche Anwendungen. Einige davon sollen hier exemplarisch vorgestellt werden.

1.3 Marktumfeld

1.3.1 Metaportal

Aktuell verlieren sich Fahrt-Anbieter und Mitfahrer von Mitfahrgelegenheiten auf den über 50 Mitfahrportalen, Apps u.v.m.. Darüber hinaus gibt es noch zahlreiche interne und / oder analoge Systeme wie Mitfahrbänke.

Es bräuchte also eine spezielle Suchmaschine, welche Zugriff auf alle Daten aller Mitfahrbörsen hat und so Nutzer schnell über alle Plattformen hinweg Fahrten suchen können. Diese Suchmaschine wäre das schon mehrfach zuvor angesprochene Meta-Portal.

Ein Meta-Portal würde von möglichst vielen Mitfahr-Plattformen Daten aggregieren und zusammenfügen. Es hätte lediglich Fahrt-Informationen und wäre so eine rein anonyme Suchmaschine: Die Kontaktaufnahme würde weiterhin auf der Plattform geschehen, in welcher der Fahrer die Fahrt eingestellt hat.

Darüber hinaus könnte das Metaportal die aggregierten Daten normalisieren, automatisiert veredeln und wiederum via ridesharing.api zur Verfügung stellen. Dies würde den Aufwand einer Vernetzung zwischen den Portalen reduzieren, da Fehler einmal auf der Metaplattform statt bei jedem Datenimport einer Mitfahrbörse ausgebügelt werden können. Außerdem könnte das Meta-Portal dieselben Daten auch über moderne Schnittstellen wie z.B. Websockets bereitstellen, selbst wenn das ursprüngliche Portal dies nicht kann.

1.3.2 Multimodalität

1.3.3 Export von Profildaten

Da es zahlreiche Ridesharing-Plattformen gibt, sollte es eine Möglichkeit geben, dass Nutzer ihre persönlichen Einstellngen und Präferenzen auf einer Plattform exportieren und auf einer anderen importieren können. Mit der DSGVO ist dieses eh wünschenswerte Feature verpflichtend geworden.

Vielfach scheitert dieses Anliegen aber noch an dem Datenformat, da man aktuell eher CSVs und Textdateien statt strukturierte Daten bekommt. Dies kann die ridesharing.api ändern, indem es ein JSON-Datenmodell definiert und so die Plattformen in weiten Teilen untereinander kompatibel macht.

Praktisch sähe das dann so aus, dass ein Nutzer beim Export seiner Nutzerdaten eine einzige JSON-Datei herunterladen könnte. Wenn die Ziel- Plattform das unterstützt, könnte der Nutzer die Datei dann auf der Ziel- Plattform hochladen und hätte so ganz einfach sämtliche Einstellungen übertragen.

1.4 Bestehende Schnittellen

Es gibt bereits eine Reihe an Datenstandards, die sich im Umfeld des Ridesharing-Marktes bewegen. Grob lassen sich diese in zwei Kategorien einsortieren: Spezielle Ride-Sharing-Schnittstellen, welche die besonderen Bedürfnisse des Ridesharings berücksichtigen, und allgemeine Mobilitäts-Daten-Standards, welche durch eine höhere Abstraktion Kompatibilität zu anderen Mobilitätsformen herstellen. Eine Auswahl:

- Dycapo ist das Ergebnis einer Bachelor-Arbeit, welche ein abstraktes Datenmodell auf JSON-Basis definiert und einen vollwertigen Server mit vielen Aspekten des Meta-M-Mitfahrportals entwickelt hat. Die Dokumentation und der Code sind öffentlich auf Github einsehbar.
- Das oben erwähnte Forschungsprojekt hat ein sehr umfangreiches XML-Datenmodell spezifiziert, welches viele wichtige Aspekte anspricht, aber mittlerweile aufgrund seines Alters von ca. 10 Jahren einer Aktualisierung bedarf .
- Eine Reihe an plattform-spezifischen Schnittstellen bilden ebenfalls Mitfahrgelegenheiten ab. Diese werden hier aus Neutralitätsgründen nicht aufgeführt. Umso mehr wird eingeladen, diese auf Github zu sammeln und zu diskutieren .
- Die abstrakten Datenstandards GTFS und GTFS-RT bieten eine gute und realistische Möglichkeit, Ridesharing-Daten mit weiteren Mobilitätsdaten wie zum Beispiel dem ÖPNV zu verknüpfen. Außerdem ist GTFS ein Standard-Import-Format für Open-Source-Routing-Engines.

- Es sollte darauf geachtet werden, Kompatibilität zum zukünftigen europäischen Datenstandard für Nahverkehrs-Daten NeTEx zu gewährleisten, da abzusehen ist, dass in diesem Umfeld Daten und Software entstehen wird, welche auch für den Mitfahr-Markt von Interesse sind.

1.5 Nomenklatur

1.6 Datenschutz

1.7 Autoren

2 Prinzipien und Funktionen der Schnittstelle

2.1 Designprinzipien

2.1.1 Aufbauen auf gängiger Praxis

Grundlage für die Erarbeitung der ridesharing.api-Spezifikation in der vorliegenden Version ist eine Analyse von aktuell (2018 - 2019) in Deutschland etablierten Ridesharing-Angeboten. Erklärtes Ziel für diese erste Version ist es, mit möglichst geringem Entwicklungsaufwand auf Seite der Plattformanbieter. Für die ridesharing.api-Spezifikation wurde sozusagen ein Datenmodell als "gemeinsamer Nenner" auf Basis der gängigen Praxis konstruiert.

2.1.2 Verbesserung gegenüber dem Status Quo wo möglich

Dort, wo es dem Ziel der einfachen Implementierbarkeit und der einfachen Migration nicht im Weg steht, erlauben sich die Autoren dieser Spezifikation, auch Funktionen aufzunehmen, die noch nicht als gängige Praxis im Bereich der Ridesharing-Plattformen bezeichnet werden können oder welche nur von einzelnen Systemen unterstützt werden. Solche Funktionen sind dann so integriert, dass sie nicht als zwingende Anforderung gelten.

Als Beispiel wäre die Fähigkeit zu Websocket-basierten Live-Updates zu nennen. Diese sind nicht verpflichtend, sind aber eine sinnvolle Erweiterung, die mit demselben Datenmodel realsierbar sind.

2.1.3 Selbstbeschreibungsfähigkeit

Ausgaben des Servers sollten so beschaffen sein, dass sie für menschliche Nutzerinnen weitgehend selbsterklärend sein können. Dies betrifft besonders die Benennung von Objekten und Objekteigenschaften.

Um den Kreis der Entwicklerinnen und Entwickler, die mit einer ridesharing.api arbeiten können, nicht unnötig einzuschränken, wird hierbei grundsätzlich und soweit sinnvoll auf englischsprachige Begrifflichkeiten gesetzt.

2.1.4 Erweiterbarkeit

Implementierer sollen in der Lage sein, über eine ridesharing.api-konforme Schnittstelle auch solche Informationen auszugeben, die nicht im Rahmen des ridesharing.api-Schemas abgebildet

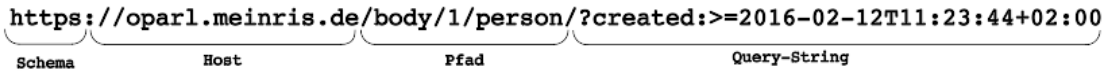

Abbildung 1: Aufbau einer URL

werden können. Dies bedeutet zum einen, dass ein System Objekttypen unterstützen und ausliefern darf, die nicht (oder noch nicht) im ridesharing.api-Schema beschrieben sind. Das bedeutet auch, dass Objekttypen so um eigene Eigenschaften erweitert werden können, die nicht im ridesharing.api Schema beschrieben sind.

Ein weiterer Aspekt betrifft die Abwärtskompatibilität, also die Kompatibilität von ridesharing.api-Clients mit zukünftigen Schnittstellen. So können beispielsweise zukünftige Erweiterungen des ridesharing.api-Schemas, etwa um neue Objekttypen, genauso durchgeführt werden, wie die Erweiterungen um herstellerspezifische Objekttypen. Ein Client muss diese Anteile nicht auswerten, sofern sie nicht für die Aufgabe des Clients relevant sind. Es bedeutet im Umkehrschluss allerdings auch, dass ein Client nicht fehlschlagen darf, falls derartige Erweiterungen vorhanden sind.

2.1.5 Browseability/Verlinkung

Klassische Webservice-Schnittstellen erfordern von den Entwicklern vollständige Kenntnis der angebotenen Einstiegspunkte und Zugriffsmethoden, gepaart mit sämtlichen unterstützten URL-Parametern, um den vollen Funktionsumfang der Schnittstelle ausschöpfen zu können.

Ridesharing-Angebote sind weitgehend in Form von Graphen aufgebaut. Das bedeutet, dass Objekte häufig mit einer Vielzahl anderer Objekte verknüpft sind. So hat eine Fahrt mehrere Stops, an denen Personen in Form einer Participation ein- oder aussteigen. Gleichzeitig können Personen Autos besitzen, die wiederum bei mehreren angebotenen Fahrten eingesetzt werden.

Eine ridesharing.api-Schnittstelle gibt jedem einzelnen Objekt eine eindeutige Adresse, eine URL. Somit kann die Schnittstelle den Verweis von einem Objekt, beispielsweise einem Gremium, auf ein anderes Objekt, etwa ein Mitglied des Gremiums, dadurch ausgeben, dass im Kontext des Gremiums die URL des Mitglieds ausgeben wird. Der Client kann somit ausgehend von einem bestimmten Objekt die zugehörigen Objekte im System finden, indem er einfach den angebotenen URLs folgt. Dieses Prinzip wird auch “Follow Your Nose”[1] genannt.

2.2 Zukunftssicherheit

Sollte in Zukunft eine zu ridesharing.api 1.0 inkompatible Version 2.0 erscheinen, kann ein Server beide Versionen gleichzeitig unterstützen, um mit ridesharing.api 1.0 Clients kompatibel zu bleiben. Dazu muss der Server die ridesharing.api 2.0-Schnittstelle unter einer eigenen URL parallel zur bestehenden ridesharing.api 1.0-Schnittstelle anbieten, siehe Kapitel System.

2.3 URLs

Den URLs (für *Uniform Resource Locators*) kommt eine besondere Bedeutung zu und es werden deshalb eine Reihe von Anforderungen an deren Aufbau und Eigenschaften gestellt. Die allgemeine Funktionsweise von URLs ist in RFC 3986 beschrieben[2].

[1] http://patterns.dataincubator.org/book/follow-your-nose.html
[2] RFC 3986: http://tools.ietf.org/html/rfc3986

Grundsätzlich **müssen** alle Zugriffe zustandslos erfolgen können, also ohne Sessioninformationen wie Cookies. Das bedeutet, dass alle Informationen, die zum Abrufen eines Objekts nötig sind, in der URL vorhanden sein müssen.

2.3.1 URL-Kanonisierung

Um Objekte eindeutig identifizieren zu können ist es notwendig, dass ein Server für ein Objekt genau eine unveränderliche URL benutzt. Diese Festlegung auf genaue eine eindeutige URL wird Kanonisierung genannt. Ein Server **muss** deshalb für jedes seiner Objekte eine kanonische URL bestimmen können.

Es wird empfohlen keine IP-Adressen in URLs zu benutzen, sondern einen mit Bedacht gewählten Hostnamen einzusetzen. Das ist vor allem im Hinblick auf die Langlebigkeit der URLs wichtig.

Um die Kanonisierung zu gewährleisten **sollten** ridesharing.api-Server so konfiguriert werden, dass sie nur über eine bestimmte Domain erreichbar sind. ridesharing.api-Server **sollten** dagegen möglichst **nicht** nur über eine IP-Addresse sowieso möglichst auch **nicht** über weitere, nicht kanonische URLs erreichbar sein.

Wenn ein Server auch durch eine nicht-kanonische URL erreichbar ist, dann **sollte** eine entsprechende HTTP-Anfrage mit einer Weiterleitung auf die entsprechende kanonische URL und HTTP-Status-Code 301 beantwortet werden. Zur Überprüfung kann z.B. der `Host`-Header einer HTTP-Anfrage verwendet werden.

Beim Pfad-Bestandteil der URL **müssen** Server-Implementierer darüber hinaus beachten, dass zur kanonischen Schreibweise auch die Groß- und Kleinschreibung, die Anzahl von Schrägstrichen als Pfad-Trennzeichen und die Anzahl von führenden Nullen vor numerischen URL-Bestandteilen gehört.

Die Kanonisierung umfasst auch den Query-String-Bestandteil der URL. Wie auch beim Pfad gilt, dass für jeden Parameter und jeden Wert im Query-String genau eine kanonische Schreibweise gelten **muss**.

Darüber hinaus **sollte** der Server-Implementierer darauf achten, Query-String-Parameter immer nach demselben Prinzip zu sortieren. Als Beispiel: Die beiden URLs

```
https://ridesharing.example.org/stops?person=1&trip=2
https://ridesharing.example.org/stops?trip=2&person=1
```

unterscheiden sich lediglich in der Reihenfolge der Query-String-Parameter. Da sie jedoch nicht identisch sind, könnten Clients annehmen, dass beide URLs verschiedene Objekte repräsentieren.

Clients **sollen** die vom Server gelieferten URLs bei Anzeige, Speicherung und Weiterverarbeitung nicht verändern.

2.3.2 HTTP und HTTPS

Der Einsatz des verschlüsselten HTTPS wird empfohlen. Bei Verwendung von HTTPS wird allen URLs “https://” voran gestellt, ansonsten beginnen URLs mit “http://”.

Aus Gründen der URL-Kanonisierung ist es **zwingend** notwendig, dass ein Server-Betreiber sich entweder für HTTP oder für HTTPS entscheidet. Es jedoch möglich, eine Weiterleitung (HTTP Status-Code 301) einzurichten. Eine Weiterleitung von HTTPS auf HTTP wird **nicht empfohlen**.

2.3.3 Langlebigkeit

Weiterhin sollen URLs langlebig sein, sodass sie möglichst lange zur Abfrage des dazugehörigen Objekts verwendet werden können.

In URLs **sollten** deshalb nur Eigenschaften des Objekts aufgenommen werden, die nicht verändert werden. Ändert sich beispielsweise die Kennung einer Drucksache im Verlauf ihrer Existenz, dann scheidet sie für die Bildung der URL aus.

Des Weiteren sollen Eigenschaften der Implementierung nicht sichtbar sein. Ist ein ridesharing.api-Server beispielsweise in PHP geschrieben, **sollte** dies **nicht** dazu führen, dass im Pfad ein Bestandteil wie "ridesharing.php/" erscheint.

Weitere Empfehlungen für langlebige URLs liefern Tim Berners-Lee[3] sowie die Europäische Kommission[4].

2.4 JSON-Ausgabe

Ein ridesharing.api-Server **muss** Objekte in Form von JSON ausgeben. Die Abkürzung JSON steht für "JavaScript Object Notation". Das JSON-Format ist in RFC 7159[5] beschrieben.

Sämtliche JSON-Ausgabe **muss** in UTF-8 ohne Byte Order Mark (BOM) geschehen. Dies entspricht RFC 7159 Section 8.1[6]. Gemäß RFC 7159 Section 7[7] **darf** UTF-8 String-Escaping verwendet werden. XML-/HTML-String-Escaping **darf nicht** verwendet werden.

Eine Syntaxübersicht und weitere Implementierungshinweise finden sich auf json.org.

Es ist gestattet, weitere zur JSON-Ausgabe semantisch identische Formate[8] anzubieten. Da diese jedoch nicht Bestandteil der Spezifikation sind, **sollten** sich Clients nicht auf deren Vorhandensein verlassen.

2.4.1 In der ridesharing.api verwendete Datentypen

In ridesharing.api werden alle in JSON definierten Dateitypen verwendet:

object: Objects entsprechen der Definition des Objects in RFC 7159 Section 4
array: Arrays entsprechen der Definition des Arrays in RFC 7159 Section 5
integer: Integers entsprechen der Definition des Integer-Parts der Number aus RFC 7159 Section 6
boolean: Booleans entsprechen der Definition von Boolean in RFC 7159 Section 3
string: Strings entsprechen der Definition der Unicode-Strings aus RFC 7159 Section 7

In der ridesharing.api werden verschiedene String-Typen verwendet. Wenn von diesen Typen gesprochen wird, so wird automatisch ein JSON-String vorausgesetzt:

url: Eine URL ist ein String, der entsprechend des URL-Kapitels formatiert wurde.

[3] Berners-Lee, Tim: Cool URIs don't change. http://www.w3.org/Provider/Style/URI.html
[4] Study on persistent URIs, with identification of best practices and recommendations on the topic for the MSs and the EC. (PDF) https://joinup.ec.europa.eu/sites/default/files/D7.1.3%20-%20Study%20on%20persistent%20URIs.pdf
[5] RFC 7159: https://tools.ietf.org/html/rfc7159
[6] RFC 7159 Section 8.1
[7] RFC 7159 Section 7
[8] Zu semantisch identischen Formaten zählen u.a.: YAML, MessagePack, etc.

url (Object): Eine URL mit in Klammern angehängtem Objektname beschreibt eine URL auf eben diesen Objekttypus.

date: Entspricht einem Datum ohne Uhrzeit und ohne Zeitzone, wie sie im folgenden Abschnitt beschrieben werden.

date-time: Entspricht einem Datum und einer Uhrzeit mit Zeitzone, wie sie im folgenden Abschnitt beschrieben werden.

2.4.2 Datums- und Zeitangaben

Für Datums- und Zeitangaben wird eine Spezielisierung der in ISO 8601 beschriebenen Formate verwendet. Ein Datum (date) **muss** muss die Form `yyyy-mm-dd` besitzen und ein Zeitpunkt (date-time) **muss** in der Form `yyyy-mm-ddThh:mm:ss±hh:mm` angegeben werden.

Beispiel für ein Datum: `1969-07-21`

Beispiel für einen Zeitpunkt: `1969-07-21T02:56:00+00:00`

2.4.3 `null`-Werte und leere Listen

JSON erlaubt es grundsätzlich, Eigenschaften mit dem Wert `null` zu versehen. Eigenschaften **sollten** nicht mit dem Wert `null` ausgegeben werden, wenn zu einer Eigenschaft keine Daten vorliegen. Obligatorische Eigenschaften **dürfen nicht** den Wert `null` haben.

Im Fall von Arrays erlaubt JSON grundsätzlich die Ausgabe von `[]` für leere Arrays. Wie bei `null` wird auch hier **empfohlen**, auf die Ausgabe einer Eigenschaft mit dem Wert `[]` zu verzichten, wenn zu einer Eigenschaft keine Daten vorliegen. Bei obligatorischen Eigenschaften **muss** jedoch eine leere Liste ausgegeben werden.

Bei nicht obligatorischen Eigenschaften sollte gleichermaßen auf die Ausgabe eines leeren Strings verzichtet werden.

2.5 Objektlisten und Paginierung

Oft wird für ein Attribut kein Wert ausgegeben, sondern ein anderes Objekt oder eine Liste von Objekten. Dabei kann eine Referenz auf das Objekt bzw. die Objektliste angegeben werden, oder das Objekt bzw. die Objektlist wird intern ausgegeben. Beide Verfahren sollen im Folgenden erklärt werden. Zu beachten ist, dass für jedes Listenattribut festgelegt ist, welches dieser Verfahren jeweils zu verwenden ist. Diese Information ist den Schemadefinitionen zu entnehmen.

2.5.1 Referenzierung von Objekten via URL

Bei der Referenzierung einzelner Objekte wird eine URL angegeben, welche auf das entsprechende Objekt verweist. Der Typ ist hierbei ein `string (url: Objekt-ID)`. Ein Beispiel hierfür ist `in`:

```
{
}
```

Es kann auch eine Liste von Referenzen ausgegeben werden. Der Typ ist in diese Fall `array of string (url: Objekt-ID)`.

Ein Beispiel hierfür ist `in`:

```
{
}
```

2.5.2 Interne Ausgabe von Objekten

Objekte können auch intern ausgegeben werden. Dabei wird das gesamte Objekt als Wert eines Attributs angegeben. Ein Beispiel für ein internes Objekt ist `trip` in `ridesharing-api:System`:

Ebenso kann eine Liste von Objekten intern ausgegeben werden. Hier das Beispiel des Attributes `stop` in `ridesharing-api:Trip`.

```
{
}
```

Bei der internen Ausgabe von Objekten **darf** der Server keine gelöschten Objekte ausgeben.

2.5.3 Externe Objektlisten

Es können auch Referenzen zu sogenannten externen Objektlisten angegeben werden. Die externe Liste enthält dann die betreffenden Objekte in Form einer Listenausgabe. Ein Beispiel dafür ist `trips` in `ridesharing-api:System`.

`ridesharing-api:System`:

```
{
}
```

Die externe Objektliste:

```
{
    "data": [

    ],
    ...
}
```

2.5.4 Paginierung

Für externe Objektlisten ist eine Aufteilung sogenannte *Listenseiten* vorgesehen, wobei jede Listenseite eine eigene URL erhält. Das dient dazu, die bei der jeweiligen Anfrage übertragenen Datenmengen und Antwortzeiten zu begrenzen.

Die Entscheidung, ob eine externe Objektliste mit Paginierung ausgegeben wird, liegt allein beim Server. Bei Listen mit mehr als 100 Einträgen wird dies **empfohlen**.

Ein Server **muss** für eine stabile Sortierung von Listeneinträgen sorgen. Das heißt, dass die Sortierung der Einträge einem konstanten Prinzip folgt und sich nicht von Abfrage zu Abfrage ändert. Das kann z.B. durch die Sortierung von Objekten nach einer eindeutigen und unveränderlichen ID erreicht werden.

Jede Listenseite **muss** die Attribute folgenden Attribute enthalten:

- **data** (Array der intern ausgegebenen Objekte)
- **pagination** (Object)
- **links** (Object)

Für `pagination` sind die folgenden Attribute festgelegt, die alle **optional** sind:

- `totalElements`: Gibt die Gesamtanzahl der Objekte in der Liste an. Diese Zahl kann sich unter Umständen bis zum Aufruf der nächsten Listenseiten ändern.
- `elementsPerPage`: Gibt die Anzahl der Objekte pro Listenseite an. Dieser Wert muss auf allen Listenseiten bis auf die letzte gleich sein.
- `currentPage`: Gibt die aktuelle Seitenzahl in der Liste an.
- `totalPages`: Gibt die Gesamtanzahl der Seiten in der Liste an.

Für `links` sind folgende Attribute festgelegt, die bis auf `next` alle **optional** sind:

- `first`: URL der ersten Listenseite
- `prev`: URL der vorherigen Listenseite
- `self`: Die kanonische URL dieser Listenseite
- `next`: URL der nächsten Listen. Für alle Seiten bis auf die letzte ist die Angabe dieser URL **zwingend**.
- `last`: URL der letzten Listenseite

```
{
    "data": [
        {...},
        {...},
        ...
    ],
    "pagination": {
        "totalElements": 50000,
        "elementsPerPage": 100,
        "currentPage": 3,
        "totalPages":500
    },
    "links": {
        "first": "https://ridesharing.example.org/trips/",
        "prev": "https://ridesharing.example.org/trips/?page=2",
        "self": "https://ridesharing.example.org/trips/?page=3",
        "next": "https://ridesharing.example.org/trips/?page=4",
        "last": "https://ridesharing.example.org/trips/?page=500",
    }
}
```

2.5.5 Filter

Externe Objektlisten können mit den URL-Parametern `created_since`, `created_until`, `modified_since` und `modified_until` eingeschränkt werden. Diese Parameter beziehen sich auf die entsprechenden Attribute der jeweiligen Objekte, wobei reservierte Zeichen URL-Kodiert werden müssen. Ein Server muss diese Parameter bei allen externen Objektlisten unterstützen.

Die Filter werden vom Client benutzt, indem die gewünschten URL-Parameter an die URL der ersten Listenseite angehängt werden. Bei allen weiteren Seiten, genauer gesagt bei den Werten von `links`, **muss** der Server sicherzustellen, dass die verwendeten Filter erhalten bleiben.

Ein Server **muss** für den im nächsten Abschnitt beschrieben Aktualisierungsmechanismus auch die den Filtern entsprechenden gelöschten Objekte ausgeben, wenn der Parameter `modified_since` gesetzt ist. Wenn `modified_since` nicht gesetzt ist, dann **dürfen** die gelöschten Objekte **nicht** ausgegeben werden. Dadurch kann sich ein Client effizient darüber informieren, welche der Objekte in seinem lokalen Bestand gelöscht wurden.

Lautet die URL für eine Liste von Drucksachen wie folgt:

```
https://ridesharing.example.org/trips/
```

kann der Client die folgende URL bilden, um die Ausgabe der Liste auf Drucksachen einzuschränken, die seit dem 1. Januar 2014 veröffentlicht wurden:

```
https://ridesharing.example.org/trips/?created_since=2014-01-01T00%3A00%3A00%2B01%3A00
```

Mehrere Parameter können auch gemeinsam verwendet werden. So kann man z.B. eine Einschränkung vom 1.1.2014 bis zum 31.1.2014 vornehmen:

```
https://ridesharing.example.org/trips/?created_since=2014-01-01T00%3A00%3A00%2B01%3A00&created_un
```

Die genannten URL-Parameter erwarten grundsätzlich eine vollständige `date-time`-Angabe.

Des Weiteren kann ein Client die Anzahl der Objekte pro Listenseite durch den URL-Parameter `limit` begrenzen, der sich auf das gleichnamige Attribut bezieht. Ein Client **darf nicht** erwarten, dass sich ein Server an seine `limit`-Anfrage hält.

2.5.6 Der Aktualisierungsmechanismus

Der Hauptnutzen der Filter ist die Möglichkeit, einen lokalen Datenbestand inkrementell zu aktualisieren.

Ein Client könnte z.B. am 1.1.2014 um 2:00 Uhr deutscher Zeit die Liste aller Drucksachen herunterladen und in einer Datenbank speichern.

```
https://ridesharing.example.org/trips/
```

Um den Datenbestand am nächsten Tag zu aktualisieren, ruft der Client dieselbe URL auf, diesmal jedoch mit dem Parameter `modified_since` mit dem Wert `2014-01-01T02:00:00+01:00` und mit `omit_internal`.

```
https://ridesharing.example.org/trips/?modified_since=2014-01-01T02%3A00%3A00%2B01%3A00&omit_inte
```

Diese Liste ist in der Regel deutlich kürzer als die Liste aller Objekte, sodass die Aktualisierung bedeutend schneller ist als der erste Abruf. Der Client muss außerdem nur noch eine deutlich kleinere Menge an Objekten in die Datenbank einfügen, aktualisieren oder löschen, um den gleichen Datenstand wie der Server zu haben.

2.6 Cross-Origin Resource Sharing (CORS)

Wenn Webbrowser mittels Skript auf JSON-Ressourcen zugreifen sollen unterliegen diese Zugriffe üblicherweise einer *Same-Origin-Policy* (SOP). Das heißt, eine Anfrage ist nur an den Server zulässig, der auch das initiierende Skript ausgeliefert hat. Anfragen an andere Server werden vom Browser blockiert. Diese Einschränkung dient im Allgemeinen der Sicherheit von Webbrowsern.[9]

Um die Daten von ridesharing.api-Servern auch im Kontext von Webanwendungen flexibel nutzen zu können, ist die Überwindung der SOP nötig. Hierzu dient *Cross-Origin Resource Sharing* (CORS)[10]. Mittels CORS kann ein Server mitteilen, dass bestimmte von ihm ausgelieferte Ressourcen auch innerhalb von Webapplikationen genutzt werden dürfen, die nicht vom selben Server ausgeliefert werden. Technisch wird dies durch Ausgabe zusätzlicher HTTP-Header erreicht.

ridesharing.api-Server **müssen** für jegliche Anfrage, die mit der Ausgabe von JSON-Daten beantwortet wird (das sind alle Anfragen außer Dateizugriffe) den folgenden HTTP-Antwort-Header senden:

```
Access-Control-Allow-Origin: *
```

Der HTTP-Antwort-Header `Access-Control-Allow-Methods` **sollte** darüber hinaus **nicht** gesetzt sein, oder **muss** die Methode `GET` beinhalten.

Entwicklerinnen von Webanwendungen sollten sich darüber bewusst sein, dass durch die direkte Einbindung von Skripten Dritter in ihre Anwendungen mögliche Sicherheitsrisiken entstehen. Für den Fall, dass ein ridesharing.api-Server, etwa in Folge einer Manipulation, Schadcode ausliefert, könnte dieser unmittelbar von Skripten im Browser ausgeführt werden.

2.7 Gelöschte Objekte

In der ridesharing.api **dürfen** Objekte **nicht** einfach gelöscht werden, sodass unter der betreffenden URL kein gültiges Objekt ausgeliefert wird. Stattdessen wird ein sogenanntes *soft delete* verwendet.

Hintergrund ist, dass ridesharing.api-Clients bei der Aktualisierung ihres Datenbestandes, z.B. mit den Filtern `modified_since` bzw. `created_since`, erfahren können müssen, welche Objekte gelöscht wurden.

Dies wird durch die folgenden Regeln gewährleistet.

Wenn ein Objekt gelöscht wird,

- **muss** das Objekt das zusätzliche Attribut `deleted` mit dem Wert `true` bekommen
- **muss** das Attribut `modified` auf den Zeitpunkt der Löschung setzen
- **müssen** die Attribute `id`, `type` und `created` erhalten bleiben
- **dürfen** alle weiteren Attribute entfernt werden

Als HTTP-Statuscode **muss** weiterhin 200 verwendet werden.

[9] vgl. Wikipedia: Same-Origin-Policy https://de.wikipedia.org/wiki/Same-Origin-Policy
[10] Cross Origin Resource Sharing - W3C Recommendation 16. Januar 2014: http://www.w3.org/TR/cors/

2.8 Ausnahmebehandlung

Wenn ein Server eine Anfrage nicht bearbeiten kann, z.B. weil die URL ungültig ist oder das angefragte Objekt nicht existiert, dann **sollte** er mit dem entsprechenden HTTP-Statuscode antworten.

Ein Server **sollte** in diesem Fall ein Objekt ausgeben, das die folgenden 3 Attribute enthält:

- `type`: Enthält als Wert die URL `https://ridesharing-api.org/1.0/Error`
- `message`: Eine Fehlermeldung, die zur Anzeige für einen Nutzer gedacht ist. Die Fehlermeldung sollte deshalb in der Sprache der durch die Schnittstelle ausgelieferten Inhalte verfasst sein
- `debug`: Zusätzliche Informationen über den Fehler

Wenn ein Server ein solches Objekt ausgibt, dann **muss** er dazu einen HTTP-Statuscode senden, der einen Fehler anzeigt.

Ein Client **darf nicht** voraussetzen, dass er im Fall eines Fehlers verwertbare Informationen wie das oben beschriebene Fehlerobjekt erhält.

2.9 ridesharing.api Endpunkt

Als ridesharing.api Endpunkt bzw. Einsprungspunkt zur Schnittstelle wird ein `ridesharing.api:System` Objekt genutzt. Falls auf einem HTTP-Host mehrere ridesharing.api-Schnittstellen oder mehrere ridesharing.api Versionen parallel installiert sind, **müssen** diese eindeutige und voneinander unabhängige ridesharing.api-Endpunkte anbieten. Es ist allerdings möglich, eine Liste von `ridesharing.api:System`-Objekten auszugeben, die z.B. auf verschiedene ridesharing.api-Versionen einer Schnittstelle verweisen.

3 Schema

Dieses Kapitel beschreibt das Schema der ridesharing.api. Das Schema definiert die Objekttypen und ihre Eigenschaften. Darüber hinaus ist im Schema auch festgelegt, in welcher Beziehung verschiedene Objekttypen zu einander stehen.

3.1 Die Objekte

Die ridesharing.api nutzt folgenden Objekte:

- ridesharing.api:System
- ridesharing.api:Trip
- ridesharing.api:Person
- ridesharing.api:Car
- ridesharing.api:Participation
- ridesharing.api:Stop
- ridesharing.api:Location
- ridesharing.api:Preferences

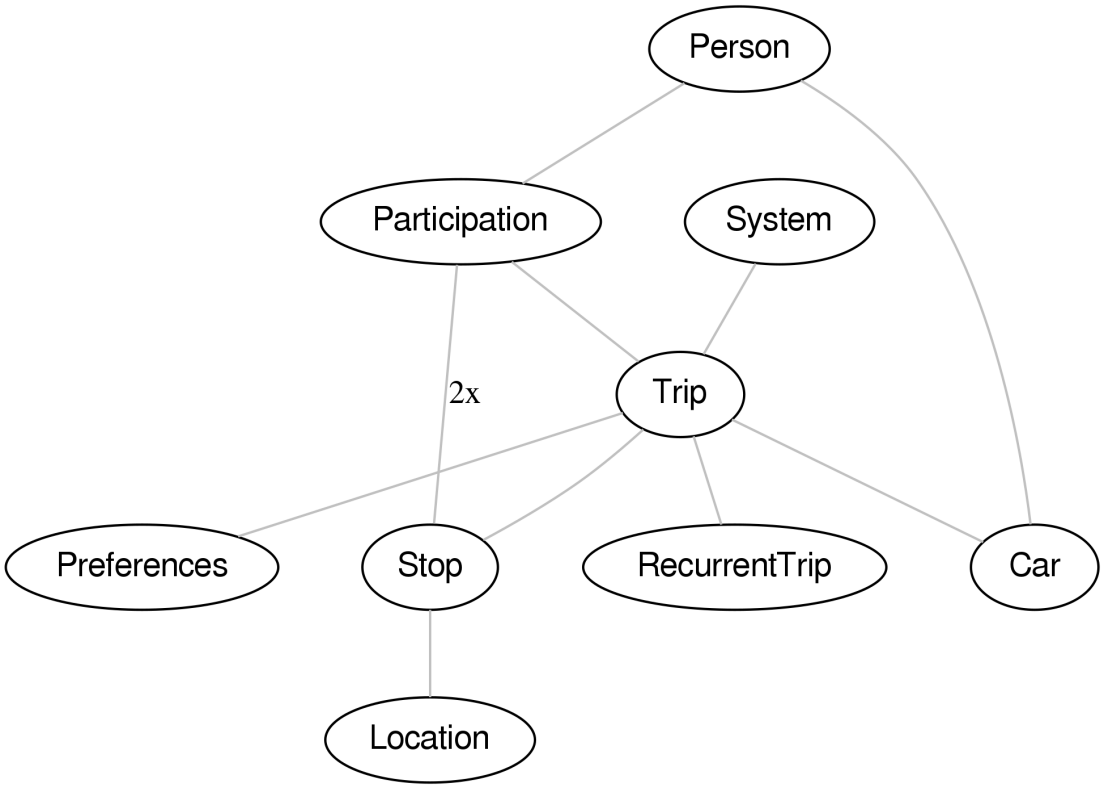

Abbildung 2: ridesharing.api Objekttypen: Ein Überblick. Die Zahl an den Verbindungslinien entspricht der Anzahl der Attribute, die eine oder mehrere Verknüpfungen herstellen.

Grundsätzlich muss jedes Objekt unter seiner ID abrufbar sein - auch dann, wenn das Objekt in anderen Objekten intern ausgegeben wird. Bei der internen Ausgabe wird beim internen Objekt auf die Rückreferenz auf das Elternobjekt verzichtet.

Als Beispiel hier eine Ausgabe von `ridesharing-api:Trip`, in welchem ein `ridesharing-api:Stop` enthalten ist:

```
{
}
```

Das enthaltene `ridesharing-api:Stop` muss auch einzeln abgerufen werden können. Dabei kommt dann das Eltern-Objekt als zusätzliches Attribut hinzu.:

```
{
}
```

Das zusätzliche Attribut ist ein Array, da es auch möglich ist, dass Fahrzeuge von mehreren Hauptobjekten aus genutzt werden. Das kann z.B. bei `ridesharing-api:Car` vorkommen:

```
{
}
```

3.2 Übergreifende Aspekte

3.2.1 Vollständigkeit

Alle regulär öffentlich abrufbaren Informationen **sollten** auch in der ridesharing.api ausgegeben werden, solange dies nicht den Datenschutzbestimmungen widerspricht. Daher sind sämtliche Felder im Schema als **empfohlen** zu behandeln, wenn nicht explizit etwas anderes angegeben wurde.

3.2.2 Herstellerspezifische Erweiterungen

In der ridesharing.api können zusätzliche, herstellerspezifische Eigenschaften hinzugefügt werden. Dazu wird diesen Eigenschaften ein Herstellerprefix vorangestellt. So könnte man z.B. `ridesharing-api:Location` um einen Point of Interest erweitern:

```
"BeispielHersteller:pointOfInterest": "Altes Stadttor",
```

3.2.3 URL-Pfade in den Beispielen

ridesharing.api-Clients wissen nichts vom Aufbau von Pfaden innerhalb von URLs, müssen dies nicht wissen, und es gibt deshalb in der ridesharing.api-Spezifikation keine Festlegungen dazu. Die in den Beispielen verwendeten URLs zeigen einen möglichen Weg zur Umsetzungen der Empfehlungen in URLs.

3.3 Eigenschaften mit Verwendung in mehreren Objekttypen

3.3.1 `id`

Die Eigenschaft `id` enthält den eindeutigen Bezeichner des Objekts, nämlich seine URL. Dies ist ein **zwingendes** Merkmal für jedes Objekt.

3.3.2 `type`

Objekttypenangabe des Objekts, **zwingend** für jedes Objekt. Der Wert ist eine Namespace-URL. Für die ridesharing.api-Objekttypen sind die folgenden URLs definiert:

Typ (kurz)	Namespace-URL
`ridesharing-api:Car`	https://schema.ridesharing-api.org/1.0/Car
`ridesharing-api:Location`	https://schema.ridesharing-api.org/1.0/Location
`ridesharing-api:Participation`	https://schema.ridesharing-api.org/1.0/Participation
`ridesharing-api:Person`	https://schema.ridesharing-api.org/1.0/Person
`ridesharing-api:Preferences`	https://schema.ridesharing-api.org/1.0/Preferences
`ridesharing-api:RecurrentTrip`	https://schema.ridesharing-api.org/1.0/RecurrentTrip
`ridesharing-api:Stop`	https://schema.ridesharing-api.org/1.0/Stop
`ridesharing-api:System`	https://schema.ridesharing-api.org/1.0/System
`ridesharing-api:Trip`	https://schema.ridesharing-api.org/1.0/Trip

3.3.3 `created`

Datum und Uhrzeit der Erstellung des jeweiligen Objekts.

Diese Eigenschaft **muss** in allen Objekttypen angegeben werden.

3.3.4 `modified`

Diese Eigenschaft kennzeichnet stets Datum und Uhrzeit der letzten Änderung des jeweiligen Objekts.

Diese Eigenschaft **muss** - genau wie `created` - in allen Objekttypen angegeben werden.

Es ist **zwingend**, dass bei jeder Änderung eines Objekts der Wert dieses Attributs auf die zu diesem Zeitpunkt aktuelle Uhrzeit gesetzt wird, da ein Client in der Regel seinen Datenbestand nur auf Basis dieses Attributs verlustfrei aktualisieren kann.

3.3.5 `deleted`

Falls das Objekt gelöscht wurde, muss dieses gemäß Kapitel 2.8 das Attribut `deleted: true` bekommen.

3.4 System

Ein ridesharing-api:System-Objekt repräsentiert eine ridesharing.api-Schnittstelle für eine bestimmte ridesharing.api-Version. Es ist außerdem der Startpunkt für Clients beim Zugriff auf einen Server.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	**ZWINGEND** Zeitpunkt der Erstellung.
`modified`	date-time	**ZWINGEND** Zeitpunkt der letzten Änderung.
`ridesharingApiVersion`	string	**ZWINGEND** ridesharing.api Version
`otherRidesharingApiVersions`	array of urls (System)	Andere ridesharing.api Versionen
`license`	url	Lizenz, unter der durch diese API abrufbaren Daten stehen.
`name`	string	Nutzerfreundlicher Name für das System, mit dessen Hilfe Nutzerinnen und Nutzer das System erkennen und von anderen unterscheiden können.
`contactEmail`	string	E-Mail-Adresse für Anfragen zur ridesharing.api-API. Die Angabe einer E-Mail-Adresse dient sowohl NutzerInnen wie auch Entwicklerinnen von Clients zur Kontaktaufnahme mit dem Betreiber.
`contactName`	string	Name der Ansprechpartnerin bzw. des Ansprechpartners oder der Abteilung, die über die in contactEmail angegebene Adresse erreicht werden kann.
`website`	url	URL der Website des Mitfahr-Portals

3.5 Route

Eine Route, zu der eine oder mehrere Fahrten gehören.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`website`	url	URL der Route auf dem Mitfahr-Portals (Deeplink)
`system`	url (System)	System
`owner`	url (Person)	Besitzer der Fahrt. Wert darf nur bei personenbesogenen Exporten ausgegeben werden.
`trip`	url (Trip)	Die zu der Route gehörenden Fahrten.
`boardingMinimum`	float	Mindestanteil der Fahrt, die ein Mitfahrender dabei sein muss.
`boardingAllowedFrom`	float	Anteil der Fahrt, bis zu dem Aussteigen nicht erlaubt ist.
`boardingAllowedTill`	float	Anteil der Fahrt, ab dem kein Zusteigen mehr erlaubt ist.
`seats`	integer	Anzahl der Sitze
`nonsmoking`	boolean	Nichtraucher-Fahrt.
`bike`	boolean	Fahrrad-Mitnahme.

Name	Typ	Beschreibung
ageFrom	integer	Mindestalter der MitfahrerInnen, numerischer Wert.
ageTill	integer	Maximales Alter der MitfahrerInnen, numerischer Wert.
gender	string	Geschlecht der MitfahrerInnen
talkingLevel	float	Das Level an Gesprächigkeit.

3.6 Trip

Das Objekt beschreibt eine abstrakte Fahrt, welche sich wöchentlich wiederholen kann.

Name	Typ	Beschreibung
id	url	
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
published	date-time	Zeitpunkt der Veröffentlichung.
expired	date-time	Zeitpunkt, ab welchem das Angebot nicht mehr gültig ist.
active	boolean	Status, ob das Angebot weiterhin angeboten wird.
website	url	Öffentlicher Link auf das Angebot (Deeplink).
route	url (RecurrentTrip)	Wiederkehrende Fahrt (falls vorhanden)
car	url (Car)	Fahrzeug
stop	array of url (Stop)	Haltepunkte
backTrip	url (Trip)	Der in die exakt entgegengesetzte Richtung laufende Trip, also die Rückfahrt.
singleTrip	array of url (SingleTrip)	Die Instanzierungen des abstrakten Trips.
seats	integer	Anzahl der Sitze
boardingMinimum	float	Mindestanteil der Fahrt, die ein Mitfahrender dabei sein muss.
boardingAllowedFrom	float	Anteil der Fahrt, bis zu dem Aussteigen nicht erlaubt ist.
boardingAllowedTill	float	Anteil der Fahrt, ab dem kein Zusteigen mehr erlaubt ist.
nonsmoking	boolean	Nichtraucher-Fahrt.
bike	boolean	Fahrrad-Mitnahme.
ageFrom	integer	Mindestalter der MitfahrerInnen, numerischer Wert.
ageTill	integer	Maximales Alter der MitfahrerInnen, numerischer Wert.
gender	string	Geschlecht der MitfahrerInnen

3.7 RecurrentTrip

Das Objekt beschreibt eine wiederkehrende Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`start`	date	Beginn der Gültigkeit der angegebenen Wochentage
`end`	date	Ende der Gültigkeit der angegebenen Wochentage
`weekday`	array of integer	Tage, an denen die Fahrt angeboten wird, als Liste an ISO 8601 Wochentag-Nummern.
`trip`	url (Trip)	Fahrt
`calendarException`	array of url (CalendarException)	Ausnahmen zu den definierten Tagen

3.8 RecurrentTrip

Das Objekt beschreibt eine wiederkehrende Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`date`	date	Ausnahme-Datum zum bestehenden Calendar
`end`	date	Ende der Gültigkeit der angegebenen Wochentage
`reason`	string	Grund führ die Ausnahme
`trip`	url (Calendar)	Calendar

3.9 Stop

Das Objekt beschreibt einen geplanten, noch nicht instanzierten Haltepunkte einer Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`moment`	date-time	**ZWINGEND** Geplante Zeitpunkt des Stops.
`momentInaccuracy`	integer	Geplante Ungenauigkeit des Stop-Zeitpunktes in Sekunden
`trip`	url (Trip)	Mitfahrgelegenheit, zu welchem der Stop gehört.
`location`	url (Location)	Ort.
`singleStop`	array of url (Stop)	Instanzierter Stop

Name	Typ	Beschreibung
`participationStart`	array of url (Participation)	Einsteigende Teilnahmen
`participationStop`	array of url (Participation)	Aussteigende Teilnahmen

3.10 Location

Das Objekt beschreibt einen geplanten, noch nicht instanzierten Ort.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`name`	string	**ZWINGEND** Name des Ortes.
`streetAddress`	string	Straße und Hausnummer des Ortes.
`postalCode`	string	Postleitzahl des Ortes.
`locality`	string	Stadt oder Ort des Ortes.
`subLocality`	string	Untergeordnete Ortsangabe der Anschrift, z.B. Stadtbezirk, Ortsteil oder Dorf.
`geojson`	object	Geodaten-Repräsentation des Orts. Der Wert dieser Eigenschaft muss der Spezifikation von GeoJSON entsprechen, d.h. es muss ein vollständiges Feature-Objekt ausgegeben werden.
`singleLocation`	array of url (SingleLocation)	Instanzierte Location
`stop`	array of url (Stop)	Haltepunkte

3.11 SingleTrip

Das Objekt beschreibt eine instanzierte, also eine real zu einem Zeitpunkt stattfindende Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`published`	date-time	Zeitpunkt der Veröffentlichung.
`expired`	date-time	Zeitpunkt, ab welchem das Angebot nicht mehr gültig ist.
`active`	boolean	Status, ob das Angebot weiterhin angeboten wird.
`url`	url	**ZWINGEND** Öffentlicher Link auf das Angebot.
`trip`	url (RecurrentTrip)	Abstrakter Trip
`singleStop`	array of url (Stop)	Haltepunkte

Name	Typ	Beschreibung
`participation`	array of url (Participation)	Teilnahmen

3.12 Stop

Das Objekt beschreibt die Haltepunkte einer Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`moment`	date-time	**ZWINGEND** Erwarteter Zeitpunkt des Stops.
`momentInaccuracy`	integer	Erwartete Ungenauigkeit des Stop-Zeitpunktes in Sekunden
`singleTrip`	url (SingleTrip)	Instanzierter Trip, zu welchem der Stop gehört.
`singleLocation`	url (SingleLocation)	Ort.
`stop`	url (Stop)	Noch nicht instanzierter Stop, von dem der SingleStop abgeleitet wurde.
`participationStart`	array of url (Participation)	Einsteigende Teilnahmen
`participationStop`	array of url (Participation)	Aussteigende Teilnahmen

3.13 SingleLocation

Das Objekt beschreibt einen instanzierten Ort.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`name`	string	**ZWINGEND** Name des Ortes.
`streetAddress`	string	Straße und Hausnummer des Ortes.
`postalCode`	string	Postleitzahl des Ortes.
`locality`	string	Stadt oder Ort des Ortes.
`subLocality`	string	Untergeordnete Ortsangabe der Anschrift, z.B. Stadtbezirk, Ortsteil oder Dorf.
`geojson`	object	Geodaten-Repräsentation des Orts. Der Wert dieser Eigenschaft muss der Spezifikation von GeoJSON entsprechen, d.h. es muss ein vollständiges Feature-Objekt ausgegeben werden.
`stop`	array of url (SingleStop)	Instanzierte Haltepunkte

Name	Typ	Beschreibung
`location`	url (Location)	Ursprüngliche noch nicht instanzierte Location, von der die SingleLocation abgeleitet wurde.

3.14 Person

Eine anonymisierte Person.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`name`	string	Der vollständige Name der Person mit akademischem Grad und dem gebräuchlichen Vornamen, wie er zur Anzeige durch den Client genutzt werden kann.
`affix`	string	Namenszusatz (z.B. jun. oder MdL.)
`title`	array of string	Akademische Titel
`givenName`	string	Vorname bzw. Taufname.
`familyName`	string	Familienname bzw. Nachname.
`gender`	string	Geschlecht. Vorgegebene Werte sind female und male, weitere werden durch die durchgehend klein geschriebene englische Bezeichnung angegeben. Für den Fall, dass das Geschlecht der Person unbekannt ist, sollte die Eigenschaft nicht ausgegeben werden.
`route`	array of url (Route)	Routen, die die Person anbietet.
`car`	array of url (Car)	Fahrzeuge
`participation`	array of url (Participation)	Teilnahmen
`personContact`	array of url (PersonContact)	Kontaktmöglichkeiten zur Person
`preferences`	url (Preferences)	Persönliche Präferenzen gegenüber Mitfahrern.

3.15 PersonContact

Eine anonymisierte PersonContact.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.

Name	Typ	Beschreibung
`contactType`	string	Art der Kontaktmöglichkeit. Vorgegebene Werte sind email, phone, fax, mobile, website. Telefonnummern sollten immer mit internationaler Vorwahl ausgegeben werden (z.B. für Deutsche Nummern also +49123456780 oder 00491234567890).
`contactIdentifier`	string	Kontaktmöglichkeit, also z.B. die E-Mail oder die Handynummer.
`person`	string	Person, zu dem der Kontakt gehört

3.16 Participation

Das Objekt beschreibt die Teilnahme an einer Fahrt.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`role`	string	**ZWINGEND** Rolle des Mitfahrenden. Mögliche Werte sind driver und passenger.
`status`	string	**ZWINGEND** Status der Teilnahme an einer Fahrt. Mögliche Werte sind driver, passenger, requested und rejected.
`start`	url (Stop)	Haltepunkt, an welchem die Person zusteigt.
`stop`	url (Stop)	Haltepunkt, an welchem die Person aussteigt.
`trip`	url (Trip)	Fahrt.
`person`	url (Person)	Anonymisierte Person.

3.17 Preferences

Das Objekt beschreibt die Präferenzen gegenüber Mitfahrern.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`nonsmoking`	boolean	Nichtraucher-Fahrt.
`gender`	string	Gender der MitfahrerInnen. Es SOLLTEN die Begriffe female fü weiblich und male für männlich verwendet werden. Andere Geschlechter SOLLTEN klein geschrieben und in englisch beschrieben werden.
`age`	string	Alter der MitfahrerInnen, Freitext.

Name	Typ	Beschreibung
`ageFrom`	integer	Mindestalter der MitfahrerInnen, numerischer Wert.
`ageTo`	integer	Maximales Alter der MitfahrerInnen, numerischer Wert.
`person`	url (Person)	Person, dem die Einstellungen gehören

3.18 Car

Das Objekt beschreibt das Fahrzeug.

Name	Typ	Beschreibung
`id`	url	
`type`	string	
`created`	date-time	Zeitpunkt der Erstellung.
`modified`	date-time	Zeitpunkt der letzten Änderung.
`carClass`	string	Art des Fahrzeugs. Hierbei SOLLTE die aus Buchstaben bestehende Klassifizierung verwendet werden, welche in der Verordnung 1400/2002 der EU-Komission beschrieben wird.
`capacity`	integer	Anzahl der Sitze.
`vacancy`	integer	Anzahl der freien Sitze.
`color`	string	Farbe des Fahrzeugs. Der Wert SOLLTE klein geschrieben und in englisch angegeben werden.
`year`	integer	Baujahr des Fahrzeugs.
`manufacturer`	string	Hersteller des Fahrzeugs.
`model`	string	Modell des Fahrzeugs.
`licencePlate`	string	Nummernschild des Fahrzeugs.
`trip`	url (Trip)	Mitfahrgelegenheit, zu welchem der Stop gehört.
`owner`	url (Person)	Besitzer des Fahrzeugs.