

ridesharing.api

ridesharing.api 0.1

Spezifikation einer einheitlichen Schnittstelle für Mitfahr-Portale.

Version dev 01.01.2019



Inhaltsverzeichnis

1	Einleitung	3
1.1	Zielsetzung	3
1.2	Nutzungsszenarien	3
1.3	Marktumfeld	3
1.4	Bestehende Standards	5
1.5	Nomenklatur	5
1.6	Datenschutz	7
1.7	Autoren	7
2	Prinzipien und Funktionen der Schnittstelle	7
2.1	Designprinzipien	7
2.2	Zukunftssicherheit	8
2.3	URLs	9
2.4	JSON-Ausgabe	10
2.5	Objektlisten und Paginierung	11
2.6	Cross-Origin Resource Sharing (CORS)	16
2.7	Gelöschte Objekte	17
2.8	Ausnahmebehandlung	17
2.9	ridesharing.api Endpunkt	17
2.10	Instanziierung	18
3	Schema	18
3.1	Die Objekte	18
3.2	Übergreifende Aspekte	21
3.3	Eigenschaften mit Verwendung in mehreren Objekttypen	22
3.4	System	23
3.5	Route	23
3.6	Trip	24
3.7	Calendar	26
3.8	CalendarException	26
3.9	Stop	26
3.10	Location	27
3.11	SingleTrip	27
3.12	SingleStop	29
3.13	SingleLocation	29

3.14 Person	30
3.15 PersonContact	30
3.16 Participation	31
3.17 Preferences	31
3.18 Car	32
4 Sucherweiterung	32
4.1 Suchanfrage	33
4.2 Suchantwort	34

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 Problem 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

Mit über 50 Mitfahrbörsen und Mitfahrapps gibt es eine Vielzahl an Angeboten. Unter den Plattformen gibt es nur in Einzelfällen einen Datenaustausch. Da die Anzahl der Fahrerinnen und Fahrer eher begrenzt ist, gibt es viele recht leere Mitfahrbörsen. Mitfahrerinnen und Mitfahrer haben daher häufig das Problem, dass Suchen nach Mitfahrgelegenheiten sehr mühsam sind, da man etliche Börsen durchgehen muss, um etwas zu finden.

FahrerInnen und MitfahrerInnen verlieren sich schlicht auf der Vielzahl nicht vernetzter Angebote, und dies sorgt für ein Schrumpfen des gesamten Marktes.

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

Ridesharing funktioniert dann ganz besonders gut, wenn man es mit dem öffentlichen Nahverkehr verbindet. Denn gerade, wenn MitfahrerInnen zwischendurch dazusteigen wollen, hängt vom Wahl des Treffpunktes die Effizienz der gesamten Fahrt ab. Bei Langstrecken-Fahrten lohnt es sich oft, einen Treffpunkt außerhalb der Innenstadt zu wählen. Und auch bei Pendelfahrten lohnt es sich, den Mitfahrenden nicht unbedingt von zu Hause aus abzuholen.

Moderne Routing-Engines bringen alles mit, um Mitfahrgelegenheiten und ÖPNV optimal zu verknüpfen. Um jedoch zu funktionieren, brauchen Routing-Engines standardisierte Rohdaten. Viele Routing-Engines funktionieren mit GTFS sowie GTFS-RT, so dass ein Datenstandard für Mitfahrgelegenheiten bestmöglich in GTFS umformbar sein sollte.

1.3.3 Export von Profildaten

Da es zahlreiche Ridesharing-Plattformen gibt, sollte es eine Möglichkeit geben, dass Nutzer ihre persönlichen Einstellungen 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 Standards

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 Ridesharing 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 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

1.5.1 Zwingende, empfohlene und optionale Anforderungen

Diese Spezifikation nutzt **müssen**, **können** und **sollten** in einer eindeutig definierten Art und Weise. Diese ist angelehnt an die Definitionen der Begriffe MUST, SHOULD und MAY (bzw. MUST NOT, SHOULD NOT und MAY NOT) aus RFC2119.¹

Die Bedeutung im Einzelnen:

müssen/muss bzw. zwingend: Die Erfüllung einer so gekennzeichneten Anforderung ist zwingend erforderlich.

Die Entsprechung in RFC2119 lautet “MUST”, “REQUIRED” oder “SHALL”.

nicht dürfen/darf nicht: Dieses Stichwort kennzeichnet ein absolutes Verbot.

Die Entsprechung in RFC2119 lautet “MUST NOT” oder “SHALL NOT”.

sollten/sollte bzw. empfohlen: Mit dem Wort **sollten** bzw. **sollte** sind empfohlene Anforderungen gekennzeichnet, die von jeder Implementierung erfüllt werden sollten. Eine Nichterfüllung ist als Nachteil zu verstehen, beispielsweise weil die Nutzerfreundlichkeit dadurch Einbußen erleidet, und sollte daher sorgfältig abgewogen werden.

Die Entsprechung in RFC2119 lautet “SHOULD” oder “RECOMMENDED”.

¹RFC2119 <http://tools.ietf.org/html/rfc2119>

sollten nicht/sollte nicht bzw. nicht empfohlen: Diese Formulierung wird verwendet, wenn unter gewissen Umständen Gründe existieren können, die ein bestimmtes Verhalten akzeptabel oder sogar nützlich erscheinen lassen, jedoch die Auswirkung des Verhaltens vor einer entsprechenden Implementierung verstanden und abgewogen werden sollten.

Die Entsprechung in RFC2119 lautet “SHOULD NOT” oder “NOT RECOMMENDED”.

dürfen/darf bzw. optional: Mit dem Wort **dürfen** bzw. **darf** oder **optional** sind optionale Bestandteile gekennzeichnet. Ein Anbieter könnte sich entscheiden, den entsprechenden Bestandteil aufgrund besonderer Kundenanforderungen zu unterstützen, während andere diesen Bestandteil ignorieren könnten. Implementierer von Clients oder Servern **dürfen** in solchen Fällen **nicht** davon ausgehen, dass der jeweilige Kommunikationspartner den entsprechenden, optionalen Anteil unterstützt.

Die Entsprechung in RFC2119 lautet “MAY”.

1.5.2 Geschlechterspezifische Begrifflichkeiten

Um bei Begriffen wie Nutzer, Anwender, Betreiber etc. die sonst übliche Dominanz der männlichen Variante zu vermeiden, werden in diesem Dokument männliche und weibliche Varianten gemischt. Gemeint sind in allen Fällen Personen jeglichen Geschlechts.

1.5.3 Codebeispiele

Die in diesem Dokument aufgeführten Codebeispiele dienen der Veranschaulichung der beschriebenen Prinzipien. Es handelt sich um frei erfundene Daten.

Codebeispiele erheben insbesondere bei JSON-Code nicht den Anspruch auf syntaktische Korrektheit und Vollständigkeit. Dementsprechend können in Codebeispielen Auslassungen vorkommen, die mit ... gekennzeichnet werden.

1.5.4 Namespace-Präfixe für Objekt- und Datentypen

Bei der Erwähnung von Objekttypen, die in dieser Spezifikation beschrieben werden, wird in der Regel ein Präfix `ridesharing-api:` vor den Namen gesetzt, z. B. “`ridesharing-api:Trip`”. Damit soll verdeutlicht werden, dass der Objekttyp innerhalb der `ridesharing-api`-Spezifikation gemeint ist.

Das Präfix `ridesharing-api:` steht hierbei für die folgende Namespace-URL:

```
https://schema.ridesharing-api.org/1.1/
```

Dadurch kann eine Typenangabe wie `ridesharing-api:Trip` eindeutig in die folgende URL übersetzt werden:

```
https://schema.ridesharing-api.org/1.1/Trip
```

1.6 Datenschutz

Bei all den besprochenen Daten geht es zunächst um nicht personenbezogene Daten, welche bestenfalls gefahrenlos geteilt werden können. Konkret bedeutet das, dass alle direkt personenbezogenen Daten wie E-Mail-Kontakt, postalische Adresse oder Handynummer auf dem Server des Anbieters verbleiben. Lediglich Informationen über die Fahrt selbst dürfen freigegeben und weiterverwendet werden. Die Kontaktaufnahme eines Interessenten muss daher auf dem Server des ursprünglichen Anbieters geschehen.

Indem man die Daten verknüpft, sind jedoch trotzdem Rückschlüsse auf den Fahrtenanbieter möglich. Daher sollte mindestens die Einverständnis des Fahrtenanbieters abgefragt werden.

Denn eines ist klar: Datenschutz hat allerhöchste Priorität. Bei dem Protokoll soll es strukturell unmöglich sein, aus Versehen personenbezogene Daten zu veröffentlichen. Daher muss klar zwischen öffentlichen und privaten Daten unterschieden werden: Erstere möchte man nutzen, zweiteere schützen.

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 Datenmodell realisierbar 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.

`https://oparl.meinris.de/body/1/person/?created:>=2016-02-12T11:23:44+02:00`

Schema Host Pfad Query-String

Abbildung 1: Aufbau einer URL

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 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"² 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 beschrieben³.

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 genau 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-Adresse sowie 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<http://patterns.dataincubator.org/book/follow-your-nose.html>

³RFC 3986: <http://tools.ietf.org/html/rfc3986>

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-Lee⁴ sowie die Europäische Kommission⁵.

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 7159⁶ beschrieben.

Sämtliche JSON-Ausgabe **muss** in UTF-8 ohne Byte Order Mark (BOM) geschehen. Dies entspricht RFC 7159 Section 8.1⁷. Gemäß RFC 7159 Section 7⁸ **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 Formate⁹ 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 Datentypen verwendet:

object: Objects entsprechen der Definition des Objects in RFC 7159 Section 4

⁴Berners-Lee, Tim: Cool URIs don’t change. <http://www.w3.org/Provider/Style/URI.html>

⁵Study 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>

⁶RFC 7159: <https://tools.ietf.org/html/rfc7159>

⁷RFC 7159 Section 8.1

⁸RFC 7159 Section 7

⁹Zu semantisch identischen Formaten zählen u.a.: YAML, MessagePack, etc.

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.
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 Spezialisierung der in ISO 8601 beschriebenen Formate verwendet. Ein Datum (`date`) **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 `location` in `Stop`:

```
{
  "id": "https://ridesharing-api.example-ridesharing.de/stop/1"
  "type": "https://schema.ridesharing-api.org/1.0/Stop",
  "location": "https://ridesharing-api.example-ridesharing.de/location/1",
  [...]
}
```

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

Ein Beispiel hierfür ist `stop` in `Trip`:

```
{
  "id": "https://ridesharing-api.example-ridesharing.de/trip/1"
  "type": "https://schema.ridesharing-api.org/1.0/Trip",
  "stop": [
    "https://ridesharing-api.example-ridesharing.de/stop/1",
    "https://ridesharing-api.example-ridesharing.de/stop/2",
    [...],
  ],
  [...]
}
```

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 `location` in `ridesharing-api:Stop`:

```
{
  "id": "https://ridesharing-api.example-ridesharing.de/stop/1"
  "type": "https://schema.ridesharing-api.org/1.0/Stop",
  "location": {
    "id": "https://ridesharing-api.example-ridesharing.de/location/1"
    "type": "https://schema.ridesharing-api.org/1.0/Location",
    [...]
  },
  [...]
}
```

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

```
{
  "id": "https://ridesharing-api.example-ridesharing.de/trip/1"
```

```

"type": "https://schema.ridesharing-api.org/1.0/Trip",
"stop": [
  {
    "id": "https://ridesharing-api.example-ridesharing.de/stop/1"
    "type": "https://schema.ridesharing-api.org/1.0/Stop",
  },
  {
    "id": "https://ridesharing-api.example-ridesharing.de/stop/2"
    "type": "https://schema.ridesharing-api.org/1.0/Stop",
  },
  [...]
]
}

```

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 `route` in `ridesharing-api:System`.

`ridesharing-api:System`:

```

{
  "id": "https://ridesharing-api.example-ridesharing.de/system"
  "type": "https://schema.ridesharing-api.org/1.0/System",
  "route": "https://ridesharing-api.example-ridesharing.de/routes"
}

```

Die externe Objektliste:

```

{
  "data": [
    {
      "id": "https://ridesharing-api.example-ridesharing.de/route/1",
      "type": "https://schema.ridesharing-api.org/1.0/System",
      [...]
    },
    {
      "id": "https://ridesharing-api.example-ridesharing.de/route/2",
      "type": "https://schema.ridesharing-api.org/1.0/System",
      [...]
    },
    [...]
  ],
  ...
}

```

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 beschriebenen 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`.

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

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 initiiierende Skript ausgeliefert hat. Anfragen an andere Server werden vom Browser blockiert. Diese Einschränkung dient im Allgemeinen der Sicherheit von Webbrowsern.¹⁰

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)¹¹. 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.

¹⁰vgl. Wikipedia: Same-Origin-Policy <https://de.wikipedia.org/wiki/Same-Origin-Policy>

¹¹Cross Origin Resource Sharing - W3C Recommendation 16. Januar 2014: <http://www.w3.org/TR/cors/>

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.

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.

2.10 Instanziierung

Ein wichtiger Aspekt im Datenstandard ist die Instanziierung. Damit ist gemeint, dass auch Mitfahrbörsen ganz analog zum ÖPNV einen Soll- und einen Ist-Fahrplan haben.

Die allermeisten Börsen haben zur Zeit nur einen Soll-Fahrplan, den Börsen ist meistens nicht bekannt, ob eine konkrete Fahrt wirklich stattfindet. Dafür hat ein relevanter Teil der Mitfahrangebote eine Art Fahrplan: die Fahrten finden nicht nur ein mal statt, sondern regelmäßig. Im Falle einer regionalen Fahrt ist dies zum Beispiel eine tägliche Fahrt zur Arbeit, im Falle einer überregionalen Fahrt Wochenend-Pendler, die jedes Wochenende an einen anderen Ort fahren.

Die `ridesharing.api` bildet die Instanziierung über die `Single*`-Objekte ab. `Trip`, `Stop` und `Location` repräsentieren den Soll-Fahrplan und haben daher keine Festlegung auf ein Datum, sondern nur auf eine Zeit und einen Rythmus. `SingleTrip`, `SingleStop` und `SingleLocation` dagegen repräsentieren konkrete Fahrten und haben daher ein exaktes Datum sowie exakte Uhrzeiten.

Durch die Verlinkung via `ids` können Update der instanziierten Datensätze auch einzeln vorgenommen werden, so dass die Serverlast gering gehalten werden kann.

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:Route`
- `ridesharing.api:Trip`
- `ridesharing.api:Calendar`
- `ridesharing.api:CalendarException`
- `ridesharing.api:Stop`
- `ridesharing.api:Location`
- `ridesharing.api:SingleTrip`
- `ridesharing.api:SingleStop`
- `ridesharing.api:SingleLocation`
- `ridesharing.api:Person`
- `ridesharing.api:PersonContact`
- `ridesharing.api:Participation`
- `ridesharing.api:Preferences`
- `ridesharing.api:Car`

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:

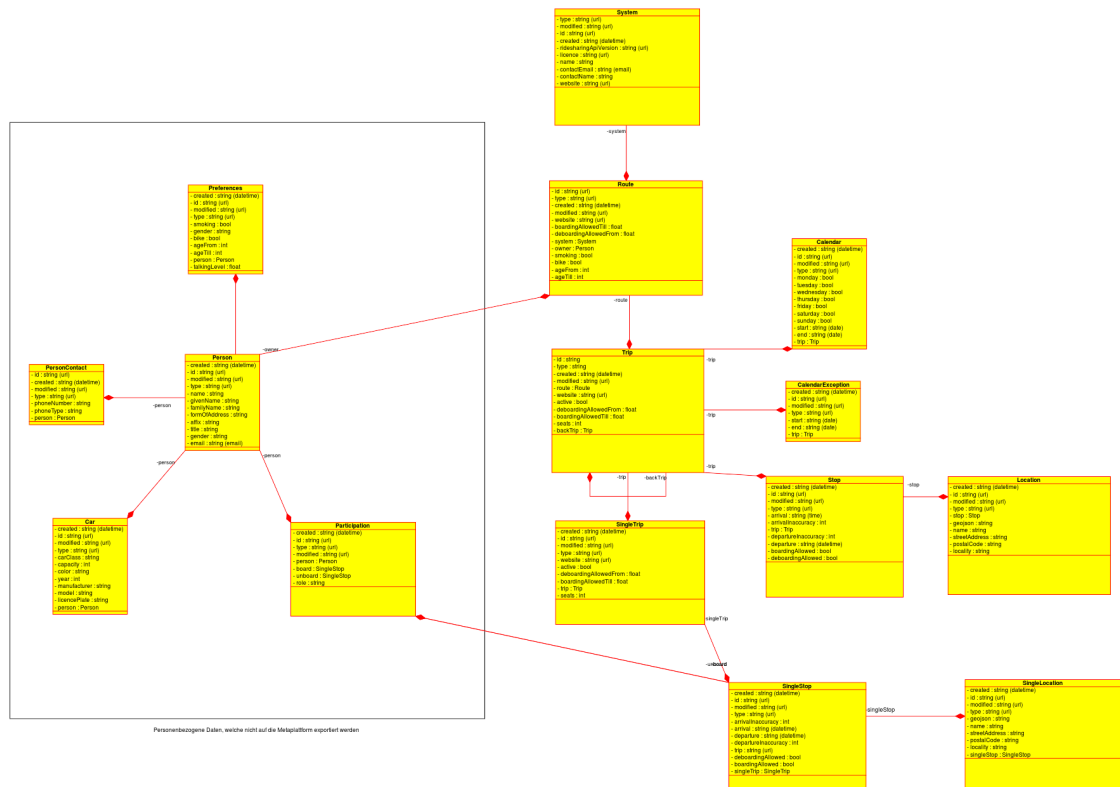


Abbildung 2: ridesharing.api Objekttypen: Ein Überblick. Die Zahl an den Verbindungslinien entspricht der Anzahl der Attribute, die eine oder mehrere Verknüpfungen herstellen.

```

{
  "id": "https://api.meine-mitfahrboerse.de/trip/123",
  "type": "https://schema.ridesharing-api.org/1.0/Trip",
  "created": "2019-02-07T18:28:18",
  "modified": "2019-03-14T15:09:26",
  "active": true,
  "seats": 3,
  "website": "https://meine-mitfahrboerse.de/trip/123",
  "route": "https://api.meine-mitfahrboerse.de/route/456",
  "stop": [
    {
      "id": "https://api.meine-mitfahrboerse.de/stop/789",
      "type": "https://schema.ridesharing-api.org/1.0/Stop",
      "created": "2019-02-07T18:28:18",
      "modified": "2019-03-14T15:09:26",
      "arrival": "10:00:00",
      "departure": "10:10:00",
      "location": {
        "id": "https://api.meine-mitfahrboerse.de/stop/012",
        "type": "https://schema.ridesharing-api.org/1.0/Location",
        "name": "Lyonesse Bahnhof"
      }
    },
    {
      "id": "https://api.meine-mitfahrboerse.de/stop/345",
      "type": "https://schema.ridesharing-api.org/1.0/Stop",
      "created": "2019-02-07T18:28:18",
      "modified": "2019-03-14T15:09:26",
      "arrival": "12:00:00",
      "departure": "12:10:00",
      "location": {
        "id": "https://api.meine-mitfahrboerse.de/stop/678",
        "type": "https://schema.ridesharing-api.org/1.0/Location",
        "name": "Atlantis Hafen"
      }
    },
    [...]
  ]
}

```

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

```

{
  "id": "https://api.meine-mitfahrboerse.de/stop/789",
  "type": "https://schema.ridesharing-api.org/1.0/Stop",
  "created": "2019-02-07T18:28:18",
  "modified": "2019-03-14T15:09:26",
  "arrival": "10:00:00",
  "departure": "10:10:00",
  "trip": "https://api.meine-mitfahrboerse.de/trip/123",
  "location": {

```

```

    "id": "https://api.meine-mitfahrboerse.de/stop/012",
    "type": "https://schema.ridesharing-api.org/1.0/Location",
    "name": "Lyonesse Bahnhof"
  }
}

```

Die im ersten Beispiel gezeigte Liste kann auch als Liste an URLs geschrieben werden:

```

{
  "id": "https://api.meine-mitfahrboerse.de/trip/123",
  "type": "https://schema.ridesharing-api.org/1.0/Trip",
  "created": "2019-02-07T18:28:18",
  "modified": "2019-03-14T15:09:26",
  "active": true,
  "seats": 3,
  "website": "https://meine-mitfahrboerse.de/trip/123",
  "route": "https://api.meine-mitfahrboerse.de/route/456",
  "stop": [
    "https://api.meine-mitfahrboerse.de/stop/789",
    "https://api.meine-mitfahrboerse.de/stop/345",
    [...]
  ]
}

```

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
<code>ridesharing-api:Calendar</code>	https://schema.ridesharing-api.org/1.0/Calendar
<code>ridesharing-api:CalendarException</code>	https://schema.ridesharing-api.org/1.0/CalendarException
<code>ridesharing-api:Car</code>	https://schema.ridesharing-api.org/1.0/Car
<code>ridesharing-api:Location</code>	https://schema.ridesharing-api.org/1.0/Location
<code>ridesharing-api:Participation</code>	https://schema.ridesharing-api.org/1.0/Participation
<code>ridesharing-api:Person</code>	https://schema.ridesharing-api.org/1.0/Person
<code>ridesharing-api:PersonContact</code>	https://schema.ridesharing-api.org/1.0/PersonContact
<code>ridesharing-api:Preferences</code>	https://schema.ridesharing-api.org/1.0/Preferences
<code>ridesharing-api:Route</code>	https://schema.ridesharing-api.org/1.0/Route
<code>ridesharing-api:SingleLocation</code>	https://schema.ridesharing-api.org/1.0/SingleLocation
<code>ridesharing-api:SingleStop</code>	https://schema.ridesharing-api.org/1.0/SingleStop
<code>ridesharing-api:SingleTrip</code>	https://schema.ridesharing-api.org/1.0/SingleTrip
<code>ridesharing-api:Stop</code>	https://schema.ridesharing-api.org/1.0/Stop
<code>ridesharing-api:System</code>	https://schema.ridesharing-api.org/1.0/System
<code>ridesharing-api:Trip</code>	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
<code>id</code>	url	
<code>type</code>	string	
<code>created</code>	date-time	ZWINGEND Zeitpunkt der Erstellung.
<code>modified</code>	date-time	ZWINGEND Zeitpunkt der letzten Änderung.
<code>ridesharingApiVersion</code>	string	ZWINGEND <code>ridesharing.api</code> Version
<code>otherRidesharingApiVersions</code>	array of url (System)	Andere <code>ridesharing.api</code> Versionen
<code>route</code>	array of url (Route)	Die zu dem System gehörenden Routen. es wird auf eine paginierte externe Objektliste verlinkt.
<code>license</code>	url	Lizenz, unter der durch diese API abrufbaren Daten stehen.
<code>name</code>	string	Nutzerfreundlicher Name für das System, mit dessen Hilfe Nutzerinnen und Nutzer das System erkennen und von anderen unterscheiden können.
<code>contactName</code>	string	Name der Ansprechpartnerin bzw. des Ansprechpartners oder der Abteilung, die über die in <code>contactEmail</code> angegebene Adresse erreicht werden kann.
<code>contactEmail</code>	string	E-Mail-Adresse für Anfragen zur <code>ridesharing.api</code> -API. Die Angabe einer E-Mail-Adresse dient sowohl NutzerInnen wie auch Entwicklerinnen von Clients zur Kontaktaufnahme mit dem Betreiber.
<code>website</code>	url	URL der Website des Mitfahr-Portals

3.5 Route

Eine Route, zu der eine oder mehrere Fahrten gehören.

Name	Typ	Beschreibung
<code>id</code>	url	
<code>type</code>	string	
<code>created</code>	date-time	Zeitpunkt der Erstellung.
<code>modified</code>	date-time	Zeitpunkt der letzten Änderung.
<code>system</code>	url (System)	System

Name	Typ	Beschreibung
owner	url (Person)	Besitzer der Fahrt. Wert darf nur bei personenbesogenen Exporten ausgegeben werden.
trip	array of url (Trip)	Die zu der Route gehörenden Fahrten.
active	boolean	Zeigt an, ob die Fahrt aktuell aktiv ist.
published	date-time	Zeitpunkt der Veröffentlichung.
expired	date-time	Zeitpunkt, ab welchem das Angebot nicht mehr gültig ist.
boardingMinimum	float	Mindestanteil der Fahrt, die ein Mitfahrender dabei sein muss.
boardingAllowedTill	float	Anteil der Fahrt, ab dem kein Zusteigen mehr erlaubt ist.
deboardingAllowedFrom	float	Anteil der Fahrt, bis zu dem Aussteigen nicht erlaubt ist.
maxDetourTime	integer	Maximale Anzahl an Minuten, die der / die FahrerIn zum Einsammeln der Mitfarherer zusätzlich fahren würde.
maxDetourDistance	integer	Maximale Anzahl an Kilometern, die der / die FahrerIn zum Einsammeln der MitfahrererInnen zusätzlich fahren würde.
seats	integer	Anzahl der Sitze
nonsmoking	boolean	Nichtraucher-Fahrt.
bike	integer	Anzahl an Fahrrädern, welche mitgenommen werden können.
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.
website	url	URL der Route auf dem Mitfahr-Portals (Deeplink)

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.
route	url (Route)	Abstrakte Route, mit der eine oder mehrere Fahrten zusammengefasst werden.
car	url (Car)	Fahrzeug
backTrip	url (Trip)	Der in die exakt entgegengesetzte Richtung laufende Trip, also die Rückfahrt.

Name	Typ	Beschreibung
<code>relatedTrip</code>	array of url (Trip)	Trips, welche in irgendeiner anderen Form mit dem bestehenden Trip in Zusammenhang stehen (z.B. Weiterfahrten).
<code>stop</code>	array of url (Stop)	Haltepunkte
<code>singleTrip</code>	array of url (SingleTrip)	Die Instanzierungen des abstrakten Trips.
<code>active</code>	boolean	Zeigt an, ob die Fahrt aktuell aktiv ist.
<code>published</code>	date-time	Zeitpunkt der Veröffentlichung.
<code>expired</code>	date-time	Zeitpunkt, ab welchem das Angebot nicht mehr gültig ist.
<code>maxDetourTime</code>	integer	Maximale Anzahl an Minuten, die der / die FahrerIn zum Einsammeln der MitfahrerIn zusätzlich fahren würde. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>maxDetourDistance</code>	integer	Maximale Anzahl an Kilometern, die der / die FahrerIn zum Einsammeln der MitfahrerInInnen zusätzlich fahren würde. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>seats</code>	integer	Anzahl der für Mitfahrer verfügbaren Sitze. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>boardingMinimum</code>	float	Mindestanteil der Fahrt, die ein Mitfahrender dabei sein muss. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>boardingAllowedFrom</code>	float	Anteil der Fahrt, bis zu dem Aussteigen nicht erlaubt ist. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>boardingAllowedTill</code>	float	Anteil der Fahrt, ab dem kein Zusteigen mehr erlaubt ist. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>nonsmoking</code>	boolean	Nichtraucher-Fahrt. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>bike</code>	integer	Anzahl an Fahrrädern, welche mitgenommen werden können. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>ageFrom</code>	integer	Mindestalter der MitfahrerInnen, numerischer Wert. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>ageTill</code>	integer	Maximales Alter der MitfahrerInnen, numerischer Wert. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.

Name	Typ	Beschreibung
<code>gender</code>	string	Geschlecht der MitfahrerInnen. Wird nur ausgegeben, wenn der Wert sich vom Wert in Route unterscheidet.
<code>website</code>	url	Öffentlicher Link auf das Angebot (Deeplink).

3.7 Calendar

Das Objekt beschreibt eine Art Fahrplan, an denen die Mitfahrgelegenheit fährt.

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

3.8 CalendarException

Das Objekt beschreibt Ausnahmen zu den im Calendar beschriebenen Tagen.

Name	Typ	Beschreibung
<code>id</code>	url	
<code>type</code>	string	
<code>created</code>	date-time	Zeitpunkt der Erstellung.
<code>modified</code>	date-time	Zeitpunkt der letzten Änderung.
<code>calendar</code>	url (Calendar)	Calendar
<code>date</code>	date	Ausnahme-Datum zum bestehenden Calendar
<code>reason</code>	string	Grund für die Ausnahme

3.9 Stop

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

Name	Typ	Beschreibung
<code>id</code>	url	

Name	Typ	Beschreibung
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
trip	url (Trip)	Mitfahrgelegenheit, zu welchem der Stop gehört.
location	url (Location)	Ort.
singleStop	array of url (Stop)	Instanziertes Stop
arrival	date-time	Geplante Ankunftszeit am Stop.
arrivalInaccuracy	integer	Geplante Ungenauigkeit der Ankunft in Sekunden
departure	date-time	Geplante Abfahrtszeit am Stop.
departureInaccuracy	integer	Geplante Ungenauigkeit der Abfahrt in Sekunden
boardingAllowed	boolean	Status, ob ein Einsteigen erlaubt ist.
deboardingAllowed	boolean	Status, ob ein Aussteigen erlaubt ist.

3.10 Location

Das Objekt beschreibt einen geplanten, noch nicht instanziierten Ort.

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

3.11 SingleTrip

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

Name	Typ	Beschreibung
id	url	
type	string	

Name	Typ	Beschreibung
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
trip	url (RecurrentTrip)	Abstrakter Trip
car	url (Car)	Fahrzeug
singleStop	array of url (Stop)	Haltepunkte
participation	array of url (Participation)	Teilnahmen
cancelled	boolean	Wenn der instanzierte Trip ausfällt, darf er nicht einfach gelöscht werden, sondern bekommt das Attribut cancelled.
maxDetourTime	integer	Maximale Anzahl an Minuten, die der / die FahrerIn zum Einsammeln der MitfahrerIn zusätzlich fahren würde. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
maxDetourDistance	integer	Maximale Anzahl an Kilometern, die der / die FahrerIn zum Einsammeln der MitfahrerInInnen zusätzlich fahren würde. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
seats	integer	Anzahl der aktuell freien Sitze.
boardingMinimum	float	Mindestanteil der Fahrt, die ein Mitfahrender dabei sein muss. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
boardingAllowedFrom	float	Anteil der Fahrt, bis zu dem Aussteigen nicht erlaubt ist. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
boardingAllowedTill	float	Anteil der Fahrt, ab dem kein Zusteigen mehr erlaubt ist. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
nonsmoking	boolean	Nichtraucher-Fahrt. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
bike	integer	Anzahl an Fahrrädern, welche mitgenommen werden können. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
ageFrom	integer	Mindestalter der MitfahrerInnen, numerischer Wert. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
ageTill	integer	Maximales Alter der MitfahrerInnen, numerischer Wert. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.
gender	string	Geschlecht der MitfahrerInnen. Wird nur ausgegeben, wenn der Wert sich vom Wert in Trip unterscheidet.

Name	Typ	Beschreibung
website	url	Öffentlicher Link auf das Angebot (Deeplink).

3.12 SingleStop

Das Objekt beschreibt einen instanziierten Haltepunkte einer Fahrt.

Name	Typ	Beschreibung
id	url	
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
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
arrival	date-time	Geplante Ankunftszeit am Stop.
arrivalInaccuracy	integer	Geplante Ungenauigkeit der Ankunft in Sekunden
departure	date-time	Geplante Abfahrtszeit am Stop.
departureInaccuracy	integer	Geplante Ungenauigkeit der Abfahrt in Sekunden
boardingAllowed	boolean	Status, ob ein Einsteigen erlaubt ist.
deboardingAllowed	boolean	Status, ob ein Aussteigen erlaubt ist.

3.13 SingleLocation

Das Objekt beschreibt einen instanziierten Ort.

Name	Typ	Beschreibung
id	url	
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
stop	array of url (SingleStop)	Instanzierter Haltepunkte
location	url (Location)	Ursprüngliche noch nicht instanziierte Location, von der die SingleLocation abgeleitet wurde.
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.

Name	Typ	Beschreibung
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.

3.14 Person

Eine Person.

Name	Typ	Beschreibung
id	url	
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
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	array of url (Preferences)	Persönliche Präferenzen gegenüber Mitfahrern.
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.
formOfAddress	string	Anrede.
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.

3.15 PersonContact

Eine Kontaktmöglichkeit zu einer Person.

Name	Typ	Beschreibung
id	url	
type	string	
created	date-time	Zeitpunkt der Erstellung.
modified	date-time	Zeitpunkt der letzten Änderung.
person	string	Person, zu dem der Kontakt gehört
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).
contactIdentifizierstring		Kontaktmöglichkeit, also z.B. die E-Mail oder die Handynummer.

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.
board	url (SingleStop)	Haltepunkt, an welchem die Person zusteigt.
deboard	url (SingleStop)	Haltepunkt, an welchem die Person aussteigt.
person	url (Person)	Person.
role	string	ZWINGEND Rolle des Mitfahrenden. Mögliche Werte sind driver und passenger.
status	string	ZWINGEND Status der Teilnahme an einer Fahrt. Mögliche Werte sind attending, requested und rejected.

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.
person	url (Person)	Person, dem die Einstellungen gehören
nonsmoking	boolean	Nichtraucher-Fahrt.

Name	Typ	Beschreibung
<code>gender</code>	string	Gender der MitfahrerInnen. Es SOLLTEN die Begriffe female für weiblich und male für männlich verwendet werden. Andere Geschlechter SOLLTEN klein geschrieben und in englisch beschrieben werden.
<code>bike</code>	boolean	Fahrradmitnahme erwünscht
<code>ageFrom</code>	integer	Mindestalter der MitfahrerInnen, numerischer Wert.
<code>ageTo</code>	integer	Maximales Alter der MitfahrerInnen, numerischer Wert.
<code>talkingLevel</code>	float	Das Level an Gesprächigkeit.

3.18 Car

Das Objekt beschreibt das Fahrzeug.

Name	Typ	Beschreibung
<code>id</code>	url	
<code>type</code>	string	
<code>created</code>	date-time	Zeitpunkt der Erstellung.
<code>modified</code>	date-time	Zeitpunkt der letzten Änderung.
<code>trip</code>	array of url (Trip)	Mitfahrgelegenheiten, in welchem das Fahrzeug eingesetzt wird.
<code>singleTrip</code>	array of url (SingleTrip)	Instanzierte Mitfahrgelegenheiten, in welchem das Fahrzeug eingesetzt wird.
<code>owner</code>	url (Person)	Besitzer des Fahrzeugs.
<code>carClass</code>	string	Art des Fahrzeugs. Hierbei SOLLTE die aus Buchstaben bestehende Klassifizierung verwendet werden, welche in der Verordnung 1400/2002 der EU-Kommission beschrieben wird.
<code>capacity</code>	integer	Anzahl der Sitze inkl. Fahrersitz.
<code>color</code>	string	Farbe des Fahrzeugs. Der Wert SOLLTE klein geschrieben und in englisch angegeben werden.
<code>year</code>	integer	Baujahr des Fahrzeugs.
<code>manufacturer</code>	string	Hersteller des Fahrzeugs.
<code>model</code>	string	Modell des Fahrzeugs.
<code>licencePlate</code>	string	Nummernschild des Fahrzeugs.
<code>vin</code>	string	Vehicle Identification Number.
<code>luggageSuitcaseCapacity</code>	integer	Gepäckkapazität, in großen Koffern gemessen.

4 Sucherweiterung

Die `ridesharing.api` ist konzipiert, um volle Datenbestände abzurufen und aktuell zu halten. Hintergrund ist, dass nur mit dem Vorliegen aller Daten wünschenswerte Features wie multimoda-

les Routing umgesetzt werden können. Daher wird empfohlen, immer diesen Weg bereitzustellen. Da zum Beispiel aber bei Browserabfragen lediglich ein nutzerspezifisches Subset an Informationen gebraucht wird, soll im Folgenden skizziert werden, wie man leicht auf dem `ridesharing.api` Datenmodell Abfragen erstellen kann.

4.1 Suchanfrage

In den allermeisten Fällen wird nach einem `SingleTrip` gesucht, da üblicherweise eine spezifische Fahrt von A nach B gesucht wird. Darüber hinaus wird im Fall von regelmäßigen Fahrten ein abstrakterer `Trip` gesucht. Die Suchanfragen unterscheiden sich wie bei `SingleTrip` vs. `Trip` vor allem darin, dass beim `SingleTrip` eine konkrete Abfahrtszeit gesucht wird, beim `Trip` dagegen aber ein sich wiederholendes Muster wie z.B. "jeden Wochentag 7:00 Uhr".

Unsere Such-API muss also für beide Basis-Objekte funktionieren, was angesichts der Ähnlichkeit beider Objekte recht einfach ist.

Um eine Suche zu starten, muss ein mit den Wünschen des Anfragenden ausgefüllter `Trip` bzw. `SingleTrip` an den Server gesendet werden. Relevant bei dieser Anfrage sind zudem die `Stops` bzw. `SingleStops` und ihre dazugehörigen `Location` bzw. `SingleLocation`.

Um alles in einen Request zu verpacken, wird die in Kapitel 2.5.3 dokumentierte interne Ausgabe von Objekten verwendet. Eine so aufgebaute Suchanfrage sähe zum Beispiel also wie folgt aus:

```
{
  "type": "https://schema.ridesharing-api.org/1.0/SingleTrip",
  "nonsmoking": true,
  "singleStop": [
    {
      "type": "https://schema.ridesharing-api.org/1.0/Stop",
      "departure": "2019-03-14T15:09:26",
      "singleLocation": {
        "streetAddress": "Am Geldspeicher 10",
        "postalCode": "12354",
        "locality": "Entenhausen"
      }
    },
    {
      "type": "https://schema.ridesharing-api.org/1.0/Stop",
      "singleLocation": {
        "streetAddress": "Am großen Pilz 5",
        "postalCode": "54321",
        "locality": "Schlumpfhausen"
      }
    }
  ]
}
```

Ebenfalls möglich ist eine Suche über Längen- und Breitengrad:

```
{
  "type": "https://schema.ridesharing-api.org/1.0/SingleTrip",
  "nonsmoking": true,
```

```

"singleStop": [
  {
    "type": "https://schema.ridesharing-api.org/1.0/Stop",
    "departure": "2019-03-14T15:09:26",
    "singleLocation": {
      "geojson": {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [49, 09]
        }
      }
    }
  },
  {
    "type": "https://schema.ridesharing-api.org/1.0/Stop",
    "singleLocation": {
      "geojson": {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [50, 10]
        }
      }
    }
  }
]
}

```

Dieselbe Art der Anfrage ist auch auf `Trip` möglich und lässt sich beliebig durch Zusätze a la `Calendar` zur Definition von regelmäßigen Fahrten erweitern.

4.2 Suchantwort

Eine Antwort auf eine Suchanfrage findet ebenfalls als `Trip` bzw `SingleTrip` mit intern ausgegebenen Objekten statt. Da die Ausgabe zu 100% der normalen `ridesharing.api` Ausgabe entspricht, verzichten wir hier auf ein Beispiel.

Wenn mit der Suchabfrage mehrere Server abgefragt werden, so wird es zwangsweise zu sehr unterschiedlichen Ergebnissen kommen: jeder Dienst wird eine andere Form der Georeferenzierung und der Priorisierung haben. Diese zwangsweise auftretende Ungenauigkeit sollte beim Entwickeln berücksichtigt werden und als Empfehlung zum vollständigen Bereitstellen der Daten verstanden werden: nur so lassen sich konsistente Suchergebnisse erzeugen.