# Projekt: adit
# REST-API Definition

Oliver Dias odiaslal@hsr.ch, Fabian Hauser fhauser@hsr.ch, Murièle Trentini mtrentin@hsr.ch,
Nico Vinzens nvinzens@hsr.ch, Michael Wieland mwieland@hsr.ch

# Änderungsgeschichte

| Datum | Version | Änderung | Autor |
|---|---|---|---|
| 31.03.2017 | 1.0 | Erstellen des Dokuments | dia |
| 02.04.2017 | 1.1 | Review Dokument | hau |
| 06.04.2017 | 1.2 | Überarbeitung HTTP-Statuscodes | vin |
| 24.04.2017 | 1.3 | Neue API Routen nachgetragen | Wie |
| 23.05.2017 | 1.4 | Hinzufügen JWT | vin |

# Inhalt

# 1. Einführung

## 1.1 Zweck

Dieses Dokument definiert die REST-Schnittstelle, welche das Backend der adit Applikation bietet und vom Frontend benutzt wird. Es soll ersichtlich sein, mit welchen Anfragen was für Objekte erhalten werden können.

## 1.2 Gültigkeitsbereich

Der Gültigkeitsbereich beschränkt sich auf die Projektdauer des Modul Engineering Projekt FS17. Das Dokument wird HSR Intern verwendet.

# 1. API Definition

## 1.1 Generelles

### 1.1.1 Allgemeines Schema

Die REST-API folgt dem Schema, dass auf der ersten Stufe eines GET-Requests auf das Objekt zugegriffen werden kann und auf der zweiten Stufe auf das dazugehörige Attribut. z.B:

```
/advertisement/<advertisementID>
```

Ist ein Request gültig, so wird immer der HTTP-Code 200 sowie ein JSON-File zurückgeliefert. Falls ein Objekt nicht gefunden werden kann, so liefert das Backend einen HTTP-Code 404 (Not Found) zurück. Bei unerlaubtem Zugriff wird ein HTTP-Code 403 (Forbidden) zurückgesendet, und falls der Request vom Backend nicht interpretiert werden kann wird der HTTP-Code 400 (Bad Request) zurückgeschickt.

### 1.1.2 Relationen

Relationen müssen client-seitig gelöst werden. Es ist möglich, z.B. alle Advertisements von User X mit der ID „XID“ mittels folgendem GET-Request abzurufen:

```
/advertisements/?userid=XID
```

Anderes Beispiel: Man sucht alle Medien (Fotos) eines Advertisements mit der ID AdvID mit dem folgenden Request:

```
/media/?advertisementid=AdvID
```

### 1.1.3 Query-Abfragen

Um eine Suchfunktion zu unterstützen, ist es möglich, Query-Abfragen zu machen. Es ist Sache des Clients, die Abfragen so zu formulieren, dass nach den entsprechenden Feldern gesucht wird. Die Suche unterstützt case-insensitive Substrings.

```
/advertisements/?description=informatik+betriebssysteme&title=informatik&tag[]=1&tag[]=2
```

## 1.2 Authentifizierung

Beispiel eines User Objektes:

### 1.1.1 Credential

2. Credential.json

```
{
 “email”: “mmuster@hsr.ch”,
 “plaintextPassword”: “supersecure”
}
```

### 1.1.2 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /authenticate | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | - | |
| **Request Body** | {Credential.json} | |
| **Bezeichnung** | Authentifiziert einen Benutzer gegenüber der REST API. Ist die Authentifizierung erfolgreich, wird ein JWT Token ausgestellt. | |
| **HTTP-Response** | 200 OK | - User existiert und Passwort ist Korrekt |
| | 401 Unauthorized | - User hat keine Berechtigung / Passwort ist falsch |

### 1.1.3 JWT (JSON Web Token)

Nach der Authentifizierung wird ein JWT zurückgeliefert das folgende Form hat:

| | |
|---|---|
| **Base-64 codiert** | eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBoc3IuY2giLCJwZXJtaXNzaW9ucyI6WyJlZGl0X3JvbGUiLCJzdXBlcnZpc29yX3Blcm1pc3Npb24iLCJhZG1pbmlzdHJhdG9yX3Blcm1pc3Npb24iLCJlZGl0X2NhdGVnb3JpZXMiLCJiYXNpY19wZXJtaXNzaW9uIiwicmV2aWV3X2FkdmVydGlzZW1lbnRzIiwiZWRpdF9pc0FjdGl2ZSJdLCJpc3MiOiJhZGl0IiwiZXhwIjoxNDk1Njk5MzY5fQ.RSCRdBWO3yO2cmz2vHFR0EtDTw2z7Rv3WeM-qD2Icrw |
| **Header:** | {<br>"alg": "HS256"<br>} |
| **Payload: Data** | {<br>"sub": "admin@hsr.ch",<br>"permissions": [<br>"edit_role",<br>"supervisor_permission",<br>"administrator_permission",<br>"edit_categories",<br>"basic_permission",<br>"review_advertisements",<br>"edit_isActive"<br>],<br>"iss": "adit",<br>"exp": 1495699369<br>} |

| | |
|---|---|
| **alg** | Signieralgorithmus |
| **sub** | E-Mail Adresse des Users |
| **permissions** | Berechtigungen des Users |
| **iss** | Aussteller des Tokens (die Website adit) |
| **exp** | Expiration-Date |
| **[signature]** | Signatur des Tokens |

## 2.1 User

Beispiel eines User Objektes:

`User.json`

```
{
 “userid”: “163”,
 “username”: “mmuster”,
 “email”: “mmuster@hsr.ch”,
 “plaintextPassword”: “supersecure”,
 “passwordHash”: “qi8H8R7OM4xMUNMPuRAZxlY”,
 “jwtToken”:“eyJhbGciOiJIUz.W4iOnRydWV9.TJVA95OrM7E2cBab30R”,
 “isPrivate”: true,
 “wantsNotifications”: “true”,
 „isActive”: true,
 “created”: “2017-03-31T07:47+01:00”,
 “updated”: “2017-03-31T10:47+01:00”,
 „roleId“: „2“
}
```

### 2.1.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /user/<userid> | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach einem User mit der userid <userid>. | |
| **HTTP-Response** | `200 OK`<br>`{ User.json }` | `- User existiert` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |
| | `404 Not found` | `- User existiert nicht` |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /users/ | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | ? conversationUserId=<userId> | |
| **Bezeichnung** | Gibt alle Benutzer zu einer Conversation zurück. Ist kein Parameter gestzt werden alle Benutzer retourniert. | |
| **HTTP-Response** | `200 OK` | `- Existieren keine User, wird ein leeres Array geschickt` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |

### 2.1.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /user | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Erstellt einen neuen Benutzer. | |
| **Objekt in Request-Body** | `{ NewUser.json }` | |
| **HTTP-Response** | `200 OK` | `- User konnte erstellt werden` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |
| | `409 Conflict` | `- Eines der Felder (z.B. Email) existiert bereits`<br>`- Das Objekt ist ungültig (z.B. Null-Field auf not null Attribut)` |

### 2.1.3 PUT-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /user/<userid> | |
| **HTTP-Methode** | PUT | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Updated einen User mit der userid <userid>. | |
| **Objekt in Request-Body** | `{ UpdatedUser.json }` | |
| **HTTP-Response** | `200 OK` | `- User existiert` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |
| | `409 Conflict` | `- Eines der Felder (z.B. Email) existiert bereits`<br>`- Das Objekt ist ungültig` |

### 1.2.1 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /user/<userid> | |
| **HTTP-Methode** | DELETE | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Löscht einen User mit der userid <userid>. | |
| **Objekt in Request-Body** | - | |
| **HTTP-Response** | `200 OK` | `- User konnte gelöscht werden` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |
| | `404 Not Found` | `- User existiert nicht` |
| | `409 Conflict` | `- Fehler, weil zu löschendes Objekt noch in anderer Tabelle benötigt wird` |

## 2.2 UserLog

Beispiel eines User Log Objektes:

UserLog.json

```
{
 „id“: „1“,
 „ip“: „152.96.210.21“,
 „action“: „Add Advertisement ‚Betriebssysteme‘“,
 “created”: “2017-03-31T07:47+01:00”,
 “userid”: “163”
}
```

### 2.2.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /userLog/<userLogId> | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach dem UserLog mit der Id <userLogId> | |
| **HTTP-Response** | 200 OK<br>{ UserLog.json } | - Userlog existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not Found | - Userlog existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /userLogs/ | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | ?userid=163&ip=152.96.210.21 | |
| **Bezeichnung** | Fragt nach allen UserLogs des Users mit der id 163 die von einem Rechner mit der IP 152.96.210.21 erfasst wurden. | |
| **HTTP-Response** | 200 OK<br>{ UserLog1.json },<br>{ UserLog2.json } | - Existieren keine Userlogs, wird ein leeres Array geschickt |
| | 403 Forbidden | - User hat keine Berechtigung |

### 2.2.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /userLog | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Erfasst einen neuen Userlog. | |
| **HTTP-Response** | 200 OK<br>{ NewUserLog.json } | - Userlog konnte erstellt werden |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Userlog existiert bereits<br>- Das Objekt ist ungültig |

## 2.3 Role

Beispiel eines Role Objektes:
Role.json

```
{
  „id“: „2“,
  „name“: „supervisor“,
}
```

### 2.3.1 GET-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /role/<roleId> | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach der Rolle mit der id <roleId> | |
| **HTTP-Response** | 200 OK<br>{ Role.json } | - Rolle existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Rolle existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /roles/ | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach allen Rollen | |
| **HTTP-Response** | 200 OK | - Rollen existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Keine Rollen vorhanden |

### 2.3.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /role | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | { Role.json } | |
| **Bezeichnung** | Erstellt eine neue Rolle | |
| **HTTP-Response** | 200 OK<br>{ Role.json } | - Rolle wurde erstellt |
| | 403 Forbidden | - User hat keine Berechtigung |

### 2.3.3 UPDATE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /role/<roleId> | |
| **HTTP-Methode** | PUT | |
| **Parameter/Query** | { Role.json } | |
| **Bezeichnung** | Updated die Rolle mit der id <roleId> | |
| **HTTP-Response** | 200 OK<br>{ Role.json } | - Rolle existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Rolle existiert nicht |

### 2.3.4 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /role/<roleId> | |
| **HTTP-Methode** | DELETE | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Löscht die Rolle mit der id <roleId> | |
| **HTTP-Response** | 200 OK<br>{ Role.json } | - Rolle existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Rolle existiert nicht |

## 2.4 Permission

Beispiel eines Permission Objektes:
`Permission.json`

```
{
  „id“: „2“,
  „name“: „Create_Review“,
}
```

### 2.4.1 GET-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /permission/<permissionId> | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach der Berechtigung mit der id < permissionId > | |
| **HTTP-Response** | 200 OK | - Berechtigung existiert |
| | 403 Forbidden | - Berechtigung hat keine Berechtigung |
| | 404 Not found | - Berechtigung existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /permissions/ | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach allen Permission | |
| **HTTP-Response** | 200 OK | - Berechtigung existiert |
| | 403 Forbidden | - Berechtigung hat keine Berechtigung |
| | 404 Not found | - Berechtigung existiert nicht |

### 2.4.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /permission | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | { Permission.json } | |
| **Bezeichnung** | Erstellt eine neue Berechtigung | |
| **HTTP-Response** | 200 OK | - Berechtigung wurde erstellt |
| | 403 Forbidden | - Berechtigung hat keine Berechtigung |

### 2.4.3 UPDATE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /permission/<permissionId> | |
| **HTTP-Methode** | PUT | |
| **Parameter/Query** | { Permission.json } | |
| **Bezeichnung** | Updated die Berechtigung mit der id <permissionId> | |
| **HTTP-Response** | 200 OK<br>{ Role.json } | - Berechtigung existiert |
| | 403 Forbidden | - Berechtigung hat keine Berechtigung |
| | `404 Not found` | – Berechtigung `existiert nicht` |

### 2.4.4 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /permission/<permissionId> | |
| **HTTP-Methode** | DELETE | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Löscht die Berechtigung mit der id <permissionId> | |
| **HTTP-Response** | 200 OK | - Berechtigung existiert |
| | 403 Forbidden | - Berechtigung hat keine Berechtigung |
| | 404 Not found | - Berechtigung existiert nicht |

## 2.5 Message

Beispiel eines Message Objektes:

Message.json

```
{
 "id": "30",
 "text": "Hallo Bastian, die Messages scheinen ja zu funktionieren!",
 "created": "2017-03-31T07:47+01:00",
 „messageState“: „unread“,
 „senderUserId:" "163",
 "recipientUserId": "164"
}
```

### 2.5.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /message/<messageId> | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Fragt nach der Message mit der ID <messageId> | |
| **HTTP-Response** | 200 OK<br>{ Message.json } | - Message existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Message existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /messages/ | |
| **HTTP-Methode** | GET | |
| **Parameter/Query** | ?userId[]=<userId1> | |
| **Bezeichnung** | Fragt nach allen Messages, bei denen der User als Sender ODER Empfänger eingetragen ist. (Es können mehrere UserId's angegeben werden). Wird kein Paramter gesetzt, werden alle Meldungen retourniert. | |
| **HTTP-Response** | 200 OK | - Existieren keine Messages, wird ein leeres Array geschickt |
| | 403 Forbidden | - User hat keine Berechtigung |

### 2.5.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /message | |
| **HTTP-Methode** | POST | |
| **Parameter/Query** | - | |
| **Bezeichnung** | Erstellt eine neue Nachricht | |
| **Objekt in Request-Body** | `{ NewMessage.json }` | |
| **HTTP-Response** | `200 OK` | `- Message konnte erstellt werden` |
| | `403 Forbidden` | `- User hat keine Berechtigung` |
| | `409` | `- Das Objekt ist ungültig` |

## 2.6 Advertisement

Beispiel eines Advertisement Objektes:

Advertisement.json

```
{
 “id”: “215”,
 “title”: “Betriebssysteme”,
 “price”: 2000,
 “description”: “Bsys2, Eduard Glatz”,
 “advertisementState”: “active”,
 “categoryId”: “15”,
 “created”: “2017-03-31T07:47+01:00”,
 “updated”: “2017-03-31T10:47+01:00”,
 “tags”: [
    { "id": 1, "name": “Informatik”},
    { "id": 2, "name": “Buch”}
  ]
}
```

### 2.6.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /advertisement/<advertisementId> | |
| **Methode** | GET | |
| **Parameter** | - | |
| **Bezeichnung** | Rückgabe des Advertisement Objektes, welches <advertisementId> hat. | |
| **HTTP-Response** | 200 OK<br>{ Advertisement.json } | - Advertisement existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Advertisement existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /advertisements/ | |
| **Methode** | GET | |
| **Parameter** | ?advertisementState[]=<advertisementState><br>&tagId[]=<tagid><br>&categoryId[]=<categoryId><br>&description=<searchterm><br>&title=<searchterm><br>&userid=<UID> | |
| **Bezeichnung** | Gefilterte Rückgabe. Alle Parameter sind optional. Sind keine Parameter gesetzt, werden alle Inserate retourniert. | |
| **HTTP-Response** | 200 OK | - Existieren keine Advertisements, wird ein leeres Array geschickt |
| | 403 Forbidden | - User hat keine Berechtigung |

### 2.6.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /advertisement | |
| **Methode** | POST | |
| **Parameter** | - | |
| **Bezeichnung** | Erstellen eines neuen Advertisement Objektes. | |
| **Objekt in Request-Body** | { NewAdvertisement.json } | |
| **HTTP-Response** | 200 OK | - Advertisement konnte erstellt werden |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Das Objekt ist ungültig |

### 2.6.3 PUT-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /advertisement/<advertisementid> | |
| **Methode** | PUT | |
| **Parameter** | - | |
| **Bezeichnung** | Updaten eines vorhandenen Advertisement Objektes mit ID advertisementId. | |
| **Objekt in Request-Body** | { UpdatedAdvertisement.json } | |
| **HTTP-Responses** | 200 OK | - Advertisement existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Das Objekt ist ungültig |

### 2.6.4 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /advertisement/<advertisementid> | |
| **Methode** | DELETE | |
| **Parameter** | - | |
| **Bezeichnung** | Löscht das Advertisement mit der Id <advertisementId> | |
| **Objekt in Request-Body** | - | |
| **HTTP-Responses** | 200 OK | - Advertisement wurde gelöscht |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Advertisement existiert nicht |

## 2.7 Category

Beispiel eines Category Objektes:

`Category.json`

```
{
 “id”: “15”,
 “name”: “Buecher”,
 “parentCategoryId: “10”
}
```

### 2.7.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /category/<categoryId> | |
| **Methode** | GET | |
| **Parameter** | - | |
| **Bezeichnung** | Rückgabe des Kategorie Objektes mit der KategorieId <categoryId> | |
| **HTTP-Response** | 200 OK<br>{ Category.json } | - Category existiert |
| | 404 Not found | - Category existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /categories/ | |
| **Methode** | GET | |
| **Parameter** | ?name=<categoryName> | |
| **Bezeichnung** | Suche alle Kategorien die den Filterkriterien entsprechen. Sind keine Parameter gesetzt, werden alle Kategorien retourniert. | |
| **HTTP-Response** | 200 OK | - Existieren keine Categories, wird ein leeres Array geschickt |

### 2.7.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /category | |
| **Methode** | POST | |
| **Parameter** | - | |
| **Bezeichnung** | Erstelle neue Kategorie | |
| **Objekt in Request-Body** | { NewCategory.json } | |
| **HTTP-Response** | 200 OK | - Category wurde erstellt |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Category existiert bereits |

### 2.7.3 PUT-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /category/<categoryId> | |
| **Methode** | PUT | |
| **Parameter** | - | |
| **Bezeichnung** | Update bestehende Kategorie | |
| **Objekt in Request-Body** | { UpdatedCategory.json } | |
| **HTTP-Response** | 200 OK | - Category existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Das Objekt ist ungültig |

### 2.7.4 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /category/<categoryId> | |
| **Methode** | DELETE | |
| **Parameter** | - | |
| **Bezeichnung** | Lösche bestehende Kategorie mit der Kategorie ID <categoryId> | |
| **Objekt in Request-Body** | - | |
| **HTTP-Response** | 200 OK | - Category wurde gelöscht |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Category existiert nicht |
| | 409 Conflict | - Fehler, weil zu löschendes Feld noch in anderer Tabelle benötigt wird |

## 2.8 Subscription

Beispiel eines Subscription Objektes:

`Subscription.json`

```
{
 “id”: “2”,
 “interval”: “300”,
 “lastUpdated”: “2017-03-31T10:47+01:00”,
 “categoryId”: “15”,
 “userId”: “163”
}
```

### 2.8.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /subscription/<subscriptionId> | |
| **Methode** | GET | |
| **Parameter** | <subscriptionId> | |
| **Bezeichnung** | Hole die Subscription mit der Id <subscriptionId> | |
| **HTTP-Response** | 200 OK | - Existieren keine Subscriptions, wird ein leeres Array geschickt |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Subscription existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /subscriptions/ | |
| **Methode** | GET | |
| **Parameter** | ?categoryId=<categoryId> | |
| **Bezeichnung** | Gefilterte Rückgabe. Alle Parameter sind optional. | |
| **HTTP-Response** | 200 OK | - Existieren keine Subscriptions, wird ein leeres Array geschickt |
| | 403 Forbidden | - User hat keine Berechtigung |

### 2.8.2 PUT-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /subscription/<subscriptionId> | |
| **Methode** | PUT | |
| **Parameter** | <subscriptionId> | |
| **Bezeichnung** | Anpassen einer neuen Subscription | |
| **Objekt in Request-Body** | { UpdatedSubscription.json } | |
| **HTTP-Response** | 200 OK | - Subscription existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Das Objekt ist ungültig |

### 2.8.3 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /subscription | |
| **Methode** | POST | |
| **Parameter** | <subscriptionId> | |
| **Bezeichnung** | Erstellen einer neuen Subscription | |
| **Objekt in Request-Body** | { NewSubscription.json } | |
| **HTTP-Response** | 200 OK | - Subscription wurde erstellt |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Subscription existiert bereits<br>- Das Objekt ist ungültig |

### 2.8.4 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /subscription/<subscriptionId> | |
| **Methode** | DELETE | |
| **Parameter** | <subscriptionId> | |
| **Bezeichnung** | Löscht die Subscription mit der Id <subscriptionId> | |
| **Objekt in Request-Body** | - | |
| **HTTP-Response** | 200 OK | - Subscription wurde gelöscht |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Subscription existiert nicht |

## 2.9 Tag

Beispiel eines Tag Objektes:

Tag.json

```
{
 “id”: “2”,
 “name”: “Unterhaltenslektüre”
}
```

### 2.9.1 GET-Methoden

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /tag/<tagId> | |
| **Methode** | GET | |
| **Parameter** | - | |
| **Bezeichnung** | Rückgabe des Tags mit der Id <tagId> | |
| **HTTP-Response** | 200 OK | - Tag existiert |
| | 404 Not found | - Tag existiert nicht |

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /tags/ | |
| **Methode** | GET | |
| **Parameter** | ?name=<tagName> | |
| **Bezeichnung** | Gefilterte Rückgabe. Alle Parameter sind optional) | |
| **HTTP-Response** | 200 OK | - Existieren keine Tags, wird ein leeres Array geschickt |

### 2.9.2 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /tags/ | |
| **Methode** | POST | |
| **Parameter** | - | |
| **Bezeichnung** | Erstellt mehrere neue Tags | |
| **Objekt im Request-Body** | { NewTags[].json } | |
| **HTTP-Response** | 200 OK | - Tag wurde erstellt |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 409 Conflict | - Das Objekt ist ungültig |

### 2.9.3 DELETE-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /tag/<tagId> | |
| **Methode** | DELETE | |
| **Parameter** | - | |
| **Bezeichnung** | Löschen eines Tags mit der ID <tagId> | |
| **Objekt im Request-Body** | - | |
| **HTTP-Response** | 200 OK | - Tag existiert |
| | 403 Forbidden | - User hat keine Berechtigung |
| | 404 Not found | - Tag existiert nicht |

## 2.10 Media

Beispiel eines Media Objektes:

Media.json

```
{
  “id”: “2”,
  “filename”: “../assets/img/buch.png”,
  “description”: “Bestes Buch über Betriebssysteme auf dem Markt“,
  “advertisementId”: “215”
}
```

### 2.10.1 POST-Methode

| Attribut | Beschreibung | |
|---|---|---|
| **Route** | /media/upload | |
| **Methode** | POST | |
| **Parameter** | Multipart File | |
| **Bezeichnung** | Lädt eine Datei hoch | |
| **HTTP-Response** | 200 OK | - Medium existiert |
| | 404 Not found | - Medium existiert nicht |