

# **REST Specfications**

## **Projekt BierIdee**

Danilo Bargaen, Christian Fässler, Jonas Furrer

14. Mai 2012

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
<b>2</b>	<b>Repräsentations-Designprinzipien</b>	<b>4</b>
<b>3</b>	<b>REST Ressourcen</b>	<b>6</b>
3.1	Beer . . . . .	6
3.2	Beers . . . . .	6
3.3	Beertype . . . . .	7
3.4	Beertypes . . . . .	7
3.5	User . . . . .	7
3.6	Users . . . . .	8
3.7	Usercredentials . . . . .	8
3.8	Recommendations . . . . .	8
3.9	Rating . . . . .	9
3.10	Consumption . . . . .	9
3.11	Consumptions . . . . .	10
3.12	Brewery . . . . .	10
3.13	Breweries . . . . .	10
3.14	Timeline . . . . .	11
3.15	Tag . . . . .	11
3.16	Tags . . . . .	12

## Änderungshistorie

Version	Datum	Änderung	Person
v1.0	03.04.2012	Dokument erstellt	dbargen
v1.1	05.04.2012	Tags und JSON Formate hinzugefügt	jfurrer
v1.2	06.04.2012	Überarbeitet, BeerType hinzugefügt	dbargen
v1.3	24.04.2012	Tagresource Beers entfernt, Paging bei Beer-List hinzugefügt	jfurrer
v1.4	03.05.2012	Definitionen von Consumptions und Ratings Ressourcen angepasst	cfaessle
v1.5	12.05.2012	Definitionen von Tag und Timeline Ressourcen angepasst	jfurrer

## 1 Einleitung

Die Definition der Ressourcen orientiert sich an den Regeln des Buches *REST API Design Rulebook* [Mas11] aus dem O'Reilly Verlag.

### URI Definition

Bei der Bezeichnung der URIs<sup>1</sup> wurde folgende Terminologie gemäss RFC 3986 verwendet:

URI = scheme "://" authority "/" path [ "?" query ] [ "#" fragment ]

### Ressource-Archetypen

Nachfolgend die Ressource-Archetypen gemäss [Mas11]. Die Erklärungstexte wurden direkt dem besagten Buch entnommen.

**Document** A document resource is a singular concept that is akin to an object instance or database record. A document's state representation typically includes both fields with values and links to other related resources.

**Collection** A collection resource is a server-managed directory of resources. Clients may propose new resources to be added to a collection. However, it is up to the collection to choose to create a new resource, or not.

**Store** A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them. On their own, stores do not create new resources; therefore a store never generates new URIs. Instead, each stored resource has a URI that was chosen by a client when it was initially put into the store.

**Controller** A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs. Like a traditional web application's use of HTML forms, a REST API relies on controller resources to perform application-specific actions that cannot be logically mapped to one of the standard methods (create, retrieve, update, and delete, also known as CRUD).

## 2 Repräsentations-Designprinzipien

Die JSON-Repräsentation ist abhängig vom Ressource-Archetypen.

---

<sup>1</sup>Uniform Resource Identifier

**Document** Ein Document gibt ein JSON Objekt zurück, welches alle relevanten Felder enthält. Informationen, welche durch die Ressource-URI bereits gegeben sind (zB type), müssen nicht erneut in der Liste erscheinen.

Falls im Document Unterobjekte auftauchen (zB das Bier einer Bewertung), wird für das Feld ein JSON-Objekt erstellt, welches die Ressource-URI und falls sinnvoll ein oder zwei relevante Felder enthält.

Beispiel:

```
{
  feld1: "{wert1}",
  feld2: "{wert2}",
  unterobjekt: {
    feld1: "{wert1}",
    uri: "{ressource-uri}"
  },
  untercollection: [
    {
      feld1: "{wert1}",
      uri: "{ressource-uri}"
    },
    {
      feld1: "{wert1}",
      uri: "{ressource-uri}"
    }
  ]
}
```

**Collection** Eine Collection gibt eine Liste mit allen enthaltenen (ggf. gefilterten) JSON-Objekten zurück. Die Objekte werden zusätzlich jeweils um ein Feld `uri` ergänzt, welches die Ressource-URI des jeweiligen Objektes enthält.

Beispiel:

```
{
  {
    feld1: "{wert1}",
    uri: "{ressource-uri}"
  },
  {
    feld1: "{wert1}",
    uri: "{ressource-uri}"
  },
}
```

**Store** Ein Store verhält sich wie eine Collection, wenn sie direkt angesprochen wird (`/store`) und wie ein Document, wenn ein spezifisches Element des Store angesprochen wird (`/store/{element-id}`).

**Controller** Der Output des Controllers ist abhängig vom Verwendungszweck.

## 3 REST Ressourcen

Nachfolgend sind die verfügbaren REST Ressourcen definiert. Alle Ressourcen sind unter der URI Authority `http://brauhaus.nusszipfel.com/` erreichbar.

### 3.1 Beer

Ein spezifisches Bier, identifiziert durch die ID.

**URI Path** `/beers/{beer-id}`

**Archetype** Document

**Methods** GET, PUT, DELETE

**JSON Format**

```
{
  id: "{beer-id}",
  name: "{beer-name}",
  image: "{image-path}",
  brand: "{brand-name}",
  beertype: "{resource-URI}",
  tags: [{
    name: "{tag-name}",
    uri: "{resource-URI}"
  }]
}
```

### 3.2 Beers

Der Bestand aller Biere.

**URI Path** `/beers`

**Query Parameters** `tag={tag-name}`, `pageSize={size}`, `pageStartIndex={index}`,  
`user={username}`

**Archetype** Collection

**Methods** GET, POST

**JSON Format**

```
[<beer document + uri>, ...]
```

### 3.3 Beertype

Ein Biertyp, identifiziert durch die ID.

**URI Path** /beertypes/{beertype-id}

**Archetype** Document

**Methods** GET, PUT, DELETE

**JSON Format**

```
{
    name: "{beertype-name}",
    description: "{description}",
}
```

### 3.4 Beertypes

Der Bestand aller Biertypen.

**URI Path** /beertypes

**Archetype** Collection

**Methods** GET, POST

**JSON Format**

```
[<beertype document + uri>, ...]
```

### 3.5 User

Ein Benutzer, identifiziert durch den Benutzernamen.

**URI Path** /users/{username}

**Archetype** Document

**Methods** GET, PUT, DELETE

**JSON Format**

```
{
    username: "{username}",
    firstName: "{first-name}",
    lastName: "{last-name}",
    email: "{email}"
}
```

### 3.6 Users

Der Bestand aller User.

**URI Path** /users

**Archetype** Collection

**Methods** GET, POST

**JSON Format**

```
[<user document + uri>, ...]
```

### 3.7 Usercredentials

Eine Controller-Ressource, welche einen signierten POST-Request entgegennimmt und die HMAC-Authentifizierungsdetails überprüft.

Als Response-Statuscode wird 204 (No Content) oder 401 (Unauthorized) zurückgegeben.

**URI Path** /usercredentials

**Archetype** Controller

**Methods** POST

### 3.8 Recommendations

Bier-Empfehlungen für einen bestimmten Benutzer.

**URI Path** /users/{username}/recommendations

**Archetype** Controller

**Methods** GET

**JSON Format**

```
[{
    "name": "{beer-name}",
```



```
        "uri": "{resource-URI}"
    }]
```

### 3.9 Rating

Die letzte Bier-Bewertung durch einen bestimmten Benutzer. Eine neue Bewertung kann mittels POST hinzugefügt werden.

**URI Path** /beers/{beer-id}/ratings/{username}

**Archetype** Controller

**Methods** GET, POST

**JSON Format**

```
{
    beer: {
        name: "{beer-name}",
        uri: "{resource-URI}"
    },
    user: {
        username: "{username}",
        uri: "{resource-URI}"
    },
    value: "{value}"
}
```

### 3.10 Consumption

Ein Bierkonsum, identifiziert durch die ID.

**URI Path** /beers/{beer-id}/consumptions/{consumption-id}

**Archetype** Readonly Document

**Methods** GET

**JSON Format**

```
{
    beer: {
        name: "{beer-name}",
        uri: "{resource-URI}"
    },
    user: {
```

```
        username: "{username}",
        uri: "{ressource-URI}"
    },
    timestamp: "{value}"
}
```

### 3.11 Consumptions

Der Bestand aller Bierkonsume für ein bestimmtes Bier.

**URI Path** /beers/{beer-id}/consumptions

**Query Parameters** user={username}

**Archetype** Collection

**Methods** GET, POST

**JSON Format**

```
[<consumption document + uri>, ...]
```

### 3.12 Brewery

Eine Brauerei, identifiziert durch die ID.

**URI Path** /breweries/{brewery-id}

**Archetype** Document

**Methods** GET, PUT, DELETE

**JSON Format**

```
{
    name: "{brewery-name}"
    size: "{value}",
    profile: "{ressource-URI}"
}
```

### 3.13 Breweries

Der Bestand aller Brauereien.

**URI Path** /breweries

**Query Parameters** brewerySize={size}

**Archetype** Collection

**Methods** GET, POST

**JSON Format**

```
[<brewery document + uri>, ...]
```

### 3.14 Timeline

Die Aktivitäts-Timeline.

**URI Path** /timeline

**Query Parameters** pageSize={size}, pageStartIndex={index}, user={username}

**Archetype** Readonly Collection

**Methods** GET

**JSON Format**

```
[{
    type: "consumpton | rating",
    date: "{date}",
    timestamp: {timestamp},
    beer: {
        name: "{beer-name}",
        uri: "{beer-uri}"
    },
    user: {
        user: "{user-name}",
        uri: "{user-uri}"
    },
    rating: {rating}
}]
```

### 3.15 Tag

Ein Tag identifiziert durch die ID.

**URI Path** /tags/{tag-id}

**Archetype** Document

**Methods** GET, PUT, DELETE

**JSON Format**

```
[{
    id: "{tag-id}"
    name: "{tag-name}",
    uri: "{ressource-URI}"
}]
```

### 3.16 Tags

Liste aller Tags.

**URI Path** /tags

**Archetype** Store

**Methods** GET, POST

**JSON Format**

```
[<tag document>, ...]
```

## Literatur

[Mas11] M. Masse. *REST API Design Rulebook*. O'Reilly Media, 2011.