Entwurf REST-API

Inhaltsverzeichnis

[Einleitung 3](#_Toc467584636)

[Ressourcen 3](#_Toc467584637)

[Was muss der REST-Service potenziell bereitstellen können? 3](#_Toc467584638)

[Interaktion mit Ressourcen 3](#_Toc467584639)

[Entwurf der URIs 4](#_Toc467584640)

[Anforderungen: 4](#_Toc467584641)

[Fehlerbehandlung 4](#_Toc467584642)

[/benutzer 6](#_Toc467584643)

[POST 6](#_Toc467584644)

[/tag 7](#_Toc467584645)

[GET 7](#_Toc467584646)

[POST 7](#_Toc467584647)

[/tag/{id} 7](#_Toc467584648)

[DELETE 7](#_Toc467584649)

[/freundschaft 8](#_Toc467584650)

[GET 8](#_Toc467584651)

[/freundschaft/{id} 8](#_Toc467584652)

[DELETE 8](#_Toc467584653)

[/raum 9](#_Toc467584654)

[GET 9](#_Toc467584655)

[/raum/{id} 10](#_Toc467584656)

[GET 10](#_Toc467584657)

[PUT 10](#_Toc467584658)

[/veranstaltung/{id} 11](#_Toc467584659)

[GET 11](#_Toc467584660)

[PUT 11](#_Toc467584661)

[DELETE 12](#_Toc467584662)

[/sitzung 13](#_Toc467584663)

[GET 13](#_Toc467584664)

[POST 13](#_Toc467584665)

[/sitzung/{id} 14](#_Toc467584666)

[PUT 14](#_Toc467584667)

[DELETE 14](#_Toc467584668)

[Notizen für Später / Nützliches 14](#_Toc467584669)

[Offene 14](#_Toc467584670)

# Einleitung

Interaktion mit dem Webservice erfolgt nur per JSON!

Das bedeutet es müssen ACCEPT und CONTENT im HTTP-Header gesetzt sein, ansonsten werden entsprechende Fehlermeldungen erzeugt!

# Ressourcen

Der Webservice soll folgende Ressourcen repräsentieren.

Für den Bau der URI ist es wichtig, sich auf eine Bezeichnung der Ressource zu einigen (Singular / Plural). Abstimmung!

Eine einheitliche URI ist ein Maß für die Qualität des Rest-Services!

|  |  |
| --- | --- |
| Singular | Plural |
| Freundschaft | Freundschaften |
| Benutzer | Benutzer |
| Sitzung | Sitzungen |
| Veranstaltung | Veranstaltungen |
| Raum | Räume |
| Tag | Tags |
| Konfiguration | Konfigurationen |
| Karte | Karten |

Es wurde im Team abgestimmt, dass wir für die Beschreibung der Ressourcen einheitlich das Singular benutzen!

# Was muss der REST-Service potenziell bereitstellen können?

Der REST-Service muss potenziell alle Instanzen von Ressourcen und Listen dieser bereitstellen können.  
Verlinkte Objekte bzw. Aggregationen sollen als <<Entscheidung>> bereitgestellt werden.

TODO: Aggregation mit einem Verlinken darstellen, oder direkt integrieren???

# Interaktion mit Ressourcen

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP-Methode | CRUD-Operation | Beschreibung | Idem potent |
| POST | CREATE |  | (NEIN) am besten |
| GET | READ |  | JA |
| PUT | UPDATE |  | JA |
| DELETE | DELETE |  | JA |

Das Einfügen von Ressourcen soll Idem potent implementiert werden! (Insert or Update)

# Entwurf der URIs

|  |  |  |  |
| --- | --- | --- | --- |
| URI | HTTP-Methode | Ergebnis | Status |
| /benutzer | POST | Neuer Benutzer / Registrieren |  |
| /benutzer | PUT | Masterpasswort senden |  |
|  |  |  |  |
| /tag | GET | Liste der Tags |  |
| /tag | POST | Neuer Tag |  |
| /tag/{id} | DELETE | Tag löschen |  |
|  |  |  |  |
| /freundschaft | GET | Freundschaften (alle Status) |  |
| /freundschaft | POST | Neue Freundschaftsanfrage |  |
| /freundschaft/{id} | PUT | Freundschafts-Status ändern |  |
| /freundschaft/{id} | DELETE | Freundschaft löschen |  |
|  |  |  |  |
| /raum | GET | Liste der Räume |  |
| /raum/{id} | GET | Raumdetails |  |
| /raum/{id} | PUT | TAG SETZEN |  |
|  |  |  |  |
| /veranstaltung | GET | Liste der Veranstaltungen |  |
| /veranstaltung | POST | Neue Veranstaltung |  |
| /veranstaltung/{id} | GET | Veranstaltungsdetails |  |
| /veranstaltung/{id} | PUT | Veranstaltung ändern |  |
| /veranstaltung/{id} | DELETE | Veranstaltung löschen |  |
|  |  |  |  |
| /sitzung | GET | Aktive Sitzung des Fragenden |  |
| /sitzung | POST | Neue Sitzung |  |
| /sitzung/{id} | PUT | Sitzung ändern |  |
| /sitzung/{id} | DELETE | Sitzung löschen |  |
|  |  |  |  |
| TODO: |  |  |  |
| /konfiguration | GET | Anfordern der Konfiguration |  |
| /konfiguration | PUT | Ändern der Konfiguration |  |

# Anforderungen:

1. JEDE Anfrage muss die Benutzer-Identifikation und ein Token enthalten, diese Angaben werden im Body der Anfrage übergeben und serverseitig geprüft.
2. Der Client erhält den Token als Anwort auf seine Registrierung(POST: /benutzer)
3. Der Client erhält Daten nur nach seiner Benutzer-Identifikation gefiltert  
   Beispiel: Benutzer A erhält nur seine Freunde in der Freundesliste!
4. Zeitstempel und Zeit-Angeben werden nur im UNIX-Format geliefert und gesendet!

# Fehlerbehandlung

Es sollen keine eigenen Fehlercodes generiert werden, anstatt sollen die Standart http-Fehlercodes genutzt werden!

|  |  |
| --- | --- |
| Statuscode | Fehlerbeschreibung |
| 200 | OK |
| 201 | CREATED |
|  |  |
| 403 | FORBIDDEN |
| 404 | NOT FOUND |
|  |  |
| 500 | INTERNAL SERVER ERROR |
| 501 | NOT IMPLEMENTED |
|  |  |
| 900 | NO ACTIVE SESSION |
| 910 | ROOM BLOCKED |

Status 404 NOT FOUND:

1. {
2. "fehlermeldung": " Die angeforderte Ressource wurde nicht gefunden",
3. }

Status 403 FORBIDDEN:

Zum Beispiel, wenn schon ein Tag gesetzt ist!

1. {
2. "fehlermeldung": " Dazu haben sie keine Berechtigung",
3. }
4. {
5. "fehlermeldung": "Token nicht gültig",
6. }

Status 501 – NOT IMPLEMENTED

Wird bei nicht implementierten Methoden geliefert!

1. {
2. "fehlermeldung": "Bisher nicht implementiert",
3. }

## /benutzer

### POST

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "email":"thomas.gorgels@googlemail.com",
4. "name":"Gorgels",
5. "vorname":"thomas",
6. "foto":"http:://google.com/profilbild"
7. }

Antworten:

Status 201 OK - Created

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "email":"thomas.gorgels@googlemail.com",
4. "name":"Gorgels",
5. "vorname":"thomas",
6. "foto":"http:://google.com/profilbild",
7. "token":" asdjkbaws923enadnh9q3qdn398z23e",
8. "professor": "false"
9. }

Status XXX – Benutzer schon vorhanden

1. {
2. "fehlermeldung": " Fehler: Benutzer schon vorhanden!",
3. }

## PUT:

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. "masterpasswort": "MatheIstToll"
5. }

Antworten:

Status 200 OK:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "email":"thomas.gorgels@googlemail.com",
4. "name":"Gorgels",
5. "vorname":"thomas",
6. "foto":"http:://google.com/profilbild",
7. "token":" asdjkbaws923enadnh9q3qdn398z23e",
8. "professor": "true"
9. }

Status 403 FORBIDDEN

## /tag

### GET

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e"
4. }

Antworten:

Status 200 OK

1. [
2. {
3. "id": "4711",
4. "name":"Präsentation"
5. },
6. {
7. "id": "4712",
8. "name":"Ruhe"
9. }
10. }

### POST

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e ",
4. "tag":
   1. {
   2. "name": "Gruppenarbeit"
   3. }
5. }

Antworten:

Status 200 OK

1. {
2. "id": "4713",
3. "name": "Gruppenarbeit"
4. }

## /tag/{id}

### DELETE

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e "
4. "tag": {"id":"4711"}
5. }

Antwort:  
Status 200 OK

## /freundschaft

### GET

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e "
4. }

Antwort

Status 200 OK

1. {
2. "freundschaften": [
3. {
4. "Benutzer":{Benutzer-Objekt},
5. "Status":"freund" //Boolean
6. "Raum":{Raumobjekt, wenn vorhanden} //Boolean
8. },
9. {
10. "Benutzer":{Benutzer-Objekt},
11. "Status":"anfrage" //Boolean
12. "Raum":{} //Boolean
13. },
14. ]
15. }

## POST

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e ",
4. "freundschaft": {"beuntzer":{email: "thomas.gorgels@googlemail.com"}}
5. }

Antwort

Status 200 OK

* Der Server erstellt eine neue Freundschaft mit dem Status „anfrage“
* Besteht schon eine Freundschaft zwischen den Benutzern, wird trotzdem 200 OK ohne Änderungen am Datenbestand zurückgegeben
* Gibt es keinen Benutzer mit der angegebenen Email-Adresse, wird trotzdem 200 OK zurückgegeben.

## /freundschaft/{id}

### PUT

Die Freundschaftsanfrage wird bestätigt!

Client:

{id} = Benutzeridentifikation des Freundes

1. {
2. "token":" asdjkbaws923enadnh9q3qdn398z23e ",
3. "id": "32423423jok4n23oasibf23ri2b3ufb2"
4. }

### 

### DELETE

Um die Freundschaftsanfrage zu identifizieren braucht man beide Benutzer-identifikationen (kombinierter Primärschlüssel). Client sendet also seine Benutzer-Identifikation und die des zu löschenden Freundes im Link!

Client:

1. {
2. "token":" asdjkbaws923enadnh9q3qdn398z23e ",
3. "id": "32423423jok4n23oasibf23ri2b3ufb2",
4. }

Antworten:

Status 200 –OK

## /raum

### GET

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":"asdjkbaws923enadnh9q3qdn398z23e"
4. }

Antworten:

Status 200 OK

1. [
2. {
3. "id": "du2ndhn23989dhx2389ch",
4. "raumnummer": "G101",
5. "teilnehmer\_max":"32",
6. "teilnehmer\_aktuell": "15",
7. "status": "rot/gelb/grün/grau" // enum??,
8. "foto": "http://<server>/raum/{id}/foto",
9. "tag":{
   1. "id": "4711",
   2. "name":"Präsentation",
   3. }
10. },
11. {
12. "raumnummer": "G102",
13. "teilnehmer\_max":"32",
14. "teilnehmer\_aktuell": "17",
15. "status": "rot/gelb/grün",
16. "foto": "http://<server>/raum/{id}/foto",
17. "tag":{
    1. "id": "4711",
    2. "name":"Präsentation",
    3. }
18. }
19. …
20. ]

## /raum/{id}

### GET

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":"asdjkbaws923enadnh9q3qdn398z23e"
4. }

Antwort:

Status 200 OK

1. {
2. "raumnummer": "G102",
3. "teilnehmer\_max":"32",
4. "teilnehmer\_aktuell": "17",
5. "status": "rot/gelb/grün",
6. "foto": "http://<server>/raum/{id}/foto",
7. "tag":{
   1. "id": "4711",
   2. "name":"Präsentation",
   3. }
8. "benutzer":[{Benutzer-Objekt1}, {Benutzer-Objekt2}] //nur nicht anonyme
9. }

### PUT

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":"asdjkbaws923enadnh9q3qdn398z23e",
4. "raum": {"tag": "4712"},
5. }

Antwort:

Status 200 OK

1. {
2. "raumnummer": "G101",
3. "teilnehmer\_max":"32",
4. "teilnehmer\_aktuell": "17",
5. "status": "rot/gelb/grün",
6. "foto": "http://<server>/raum/{id}/foto",
7. "tag":{
   1. "id": "4712",
   2. "name":"Ruhe",
   3. }
8. }

Status 403 FORBIDDEN: siehe oben!

* Wenn ein Benutzer keine Berechtigung hat, den Tag zu setzen
* Antwort soll trotzdem Raumobjekt enthalten

## /veranstaltung

### GET

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e "
4. }

Antworten

Status 200 OK

1. {
2. "verantaltungen": [{Veranstaltungs-Objekt},{Veranstaltungs-Objekt}]
3. }

Status 403 FORBIDDEN

### POST

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e ",
4. "veranstaltung": {{
5. "datum": "11.11.2016",
6. "von": "13:00"
7. "bis": "15:00",
8. "raum": "4711"
9. }
10. }

Antwort:

Status 200 OK:

1. {
2. "id": "1188",
3. "benutzer":{}, //Professor
4. "datum": "11.11.2016",
5. "von": "13:00"
6. "bis": "15:00",
7. "raum":{} // Raum der Veranstaltung
8. }

Status 403 FORBIDDEN:

Status 910 RAUM BEREITS BELEGT

## /veranstaltung/{id}

### GET

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e "
4. }

Antwort:

Status 200 OK

1. {
2. "id": "1188",
3. "benutzer":{}, //Professor
4. "datum": "11.11.2016",
5. "von": "13:00"
6. "bis": "15:00",
7. "raum":{} // Raum der Veranstaltung
8. }

### PUT

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. "veranstaltung":{
5. "datum": "11.11.2016",
6. "von": "13:00"
7. "bis": "15:00",
8. "raum":{}
9. }
10. }

Status 200 OK:

1. {
2. "id": "1188",
3. "benutzer":{},
4. "datum": "11.11.2016",
5. "von": "13:00"
6. "bis": "15:00",
7. "raum":{}
8. }

Status 403 FORBIDDEN

Status 910 RAUM BEREITS BELEGT

### DELETE

Client

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. }

Antworten:

Status 200 OK

Status 403 FORBIDDEN

Status 404 NOT FOUND

## /sitzung

### GET

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. }

Antworten:

Status 200 OK:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "raum": {Raum-Objekt},
4. "myTag": "true/false",
5. "endzeit": "2016-11-21 13:44" //Unix timestamp?
6. }

Status 900 KEINE AKTIVE SITZUNG

1. {
2. "räume": [{Raum-Objekt1},{Raum-objekt2}],
3. "karte": ??????,
4. }

### POST

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. "sitzung":{"raum": "4555455"},
5. }

Antworten:

Status 201 CREATED

Besteht bereits eine aktive Sitzung des Benutzers, wird diese überschrieben und trotzdem 201 zurückgegeben! Hier greift also nicht „insert und update“

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "raum": {Raum-Objekt},
4. "myTag": "true/false",
5. "endzeit": "2016-11-21 13:44" //Unix timestamp?
6. }

## /sitzung/{id}

### PUT

Client:

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. "sitzung":{} // Vielleicht noch für später,
5. }

Antworten

Status 200 OK

Server setzt Endzeit auf aktuellen Zeitpunkt + SITZUNGSINTERVALL

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "raum": {Raum-Objekt},
4. "myTag": "true/false",
5. "endzeit": "2016-11-21 13:44" //Unix timestamp?
6. }

400 BAD REQUEST

### DELETE

Client

1. {
2. "id": "32423423jok4n23oasibf23ri2b3ufb2",
3. "token":" asdjkbaws923enadnh9q3qdn398z23e",
4. "sitzung":{"id" : "4711"} // Vielleicht noch für später,
5. }

Antwort

Status 200 OK

Status 400 BAD REQUEST

Status 403 FORBIDDEN

## /konfiguration

### GET

### PUT

## /karte

TODO: Konzeptionelle arbeit leisten!

# Notizen für Später / Nützliches

* Für das Sortieren der Informationen / Listen kann URI-Parameter **sort** genutzt werden, dem eine kommaseparierte Liste von Attributen der Ressource mit jeweils führendem + (aufsteigende Sortierung), oder – (absteigende Sortierung) zugewiesen wird
* Wie lösen wir das mit der Sicherheit, Jeder kann nur seine Freunde sehen etc.