HLM-NG | API

Oussama Zgheb & Tobias Zahner

13. Mai 2015

Datum	Version	Änderung	Autor
11.03.15	1.0	Erste Version	Oussama Zgheb
13.03.15	1.1	Calls für Events & Speaker	Oussama Zgheb
17.03.15	1.2	Calls für Media	Oussama Zgheb
23.03.15	2.0	Calls für Event Item & Event Room	Oussama Zgheb
24.03.15	2.1	JSON Beispiele hinzugefügt	Oussama Zgheb
25.03.15	2.2	Delete Operationen sowie Formatierung	Oussama Zgheb
27.03.15	2.3	Vote Vorgang in UML	Oussama Zgheb
30.03.15	2.4	Erneuerungen bei Push	Oussama Zgheb
30.03.15	2.5	Code Formatierung, Cals für QR Code	Oussama Zgheb
31.03.15	2.6	QR Code Überprüfung , File Upload	Oussama Zgheb
31.03.15	2.7	Null Kennzeichnungen	Oussama Zgheb
01.04.15	2.8	Unique Kennzeichnung, Voting Duration	Oussama Zgheb
09.04.15	2.9	»Newest« Calls, Spam Schutz	Oussama Zgheb
12.04.15	2.91	Lastupdatetime Call	Oussama Zgheb
14.04.15	3.0	Informationen & Konventionen gruppiert und erweitert	Oussama Zgheb
17.04.15	3.1	Social Items haben AuthorName Feld	Oussama Zgheb
20.04.15	3.2	Event Active Felder	Oussama Zgheb
27.04.15	4.0	Adm / Pub URL's (Änderungen v. Sitzung 23.04.15)	Oussama Zgheb
28.04.15	4.0.1	Fertigstellung Adm / Pub, »/sliders« und »/votes« Calls für Voting	Oussama Zgheb
06.05.15	4.1	Voting Triggers, »/sliders« Call nun auch für Pub	Oussama Zgheb
07.05.15	4.1.1	Inhaltliche Verbesserungen, X-QRCode Header Feld	Oussama Zgheb
08.05.15	4.2	User Listing für Pub entfernt, Social Status update	Oussama Zgheb
13.05.15	4.2.1	Media Thumbnail	Oussama Zgheb
14.05.15	4.3	SpeakerIDFK in Event Item nachgeführt	Oussama Zgheb

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument soll eine ausführliche Übersicht über die Rest API des Backend geben. Dabei sollen alle möglichen »Calls«, die zu erwartenden Antworten sowie Konventionen und Benutzungsinformationen aufgeführt werden. Ziel ist es den Entwickler des Mobile Apps sowie des Backends eine Hilfe bei der Erweiterung sowie Unterhalt von HLM-NG zu sein.

2 API

2.1 Informationen und Konventionen

2.1.1 Legende

- **S & Q** User Authentisierung benötigt (Siehe Kapitel »Login«)
- **[Null]** kennzeichnet Felder die »null« sein dürfen und können
- **[!]** kennzeichnet Felder die automatisch gesetzt werden. Dies bedeutet, dass bei einem POST / PUT diese nicht mitgeliefert werden müssen. Falls diese trotzdem mit gegeben werden, werden diese schlichtweg ignoriert.
- **[Unique]** kennzeichnet Felder welche Einzigartig sein müssen, falls etwas hinzugefügt / geändert wird und einen nicht einzigartigen Wert enthält wird ein Bad Request zurückgeliefert.
- **{varname}** Variablen werden in dieser Dokumentation mit geschweiften Klammern gekennzeichnet

2.1.2 Zeit und Daten

Das Format für »Date« **YYYY-MM-DD**.
Das Format für »Time« **HH:MM**
Das Format für »Date Time« **YYYY-MM-DD HH:MM**

2.1.3 Login

- Bei den als **S** gekennzeichneten Calls wird ein Basic **HTTP-Auth** erwartet[1]
  Das Passwort ist dabei gleich der DeviceID. Kann 401, 403 zurückliefern.
  - Bei erfolgreicher Authentifizierung wird die eigentliche Aktion durchgeführt
  - Bei fehlgeschlagener Authentifizierung wegen falschen Anmeldedaten wird erneut ein HTTP-Auth geschickt
  - Bei einem Fehler bei dem Anmeldedaten Parser wird ein Bad Request geschickt
- Bei den als Q gekennzeichneten Calls kann nebst der Basic HTTP-Auth ein zusätzliches Header Feld mit dem Namen »X-QRCode« mitgesendet werden. Bei erfolgreicher Auswertung des Wertes werden die Calls mit spezieller Berechtigung ausgeführt (z.B. Voting als Jury). Der Wert dieses Header ist gleich der Payload des QR Codes.

2.1.4 Diverses

- Eine fixe Reihenfolge der Properties ist bei JSON (standardmässig) nicht gewährleistet [2]
- Der Befehl »POST«
  - Liefert (falls erfolgreich) das eingefügt Element als JSON zurück
- Der Befehl »PUT »
  - Updatet lediglich - Es wird kein neues Element erstellt falls unter die definierte ID nicht existiert

[1] https://de.wikipedia.org/wiki/HTTP-Authentifizierung#Basic_Authentication
[2] 4.3.3 Object - http://goo.gl/eAjLb

  - Ignoriert eine abweichende ID

- Spam Schutz
  Bei folgenden Aktionen wird beim überschreiten einer Threshold für eine gewisse Zeit zu jeder Anfrage ein 429 zurückgeliefert (too many requests).
  Die genauen Einstellungen sind »FileSettings« zu entnehmen.

  - Bei allen Anfragen die eine Benutzerauthentisierung fordern

  - Erstellung oder Update eines Benutzerprofils

- Last Update Time
  Bei jeder Resource gibt es einen call Namens »/lastupdateimte« welcher den Zeitpunkt (in Unixzeit [3]) angibt, an welchem irgend ein Element geändert wurde. So kann der Client erfahren (durch vergleichen des letzten Zeitpunktes) ob sein Datenbestand noch aktuell ist.

- Alle Pfade werden relativ angegeben

- Thumbnails
  Bei allen »media« Links kann »_thumb« angehängt werden, um ein Thumbnail des besagten Bildes zu erhalten

- Caching
  Sämtliche Medien Ressourcen offerieren Client Caching durch ETag [4]
  Hier zu sehen ist die erste Anfrage für »shot0008.png«.
  Der Server sendet dem Client ein Etag mit welcher sich der Client speichern soll.

```
GET /hlmng/rest/pub/media/image/png/shot0008.png HTTP/1.1
Host: localhost:8443
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Cache-Control: private, no-transform, max-age=1800
Expires: Thu, 01 Jan 1970 01:00:00 CET
Etag: "1712108524"
Content-Type: image/png Transfer-Encoding: chunked
Date: Tue, 28 Apr 2015 11:23:34 GMT
```

  Hier zu sehen ist die erneute Anfrage für »shot0008.png«.
  Der Client sendet »If-None-Match« mit, anhand diesem Wert entscheidet der Server ob das Bild 304 (not modified) oder 200 (nochmals neu senden) ist.

```
GET /hlmng/rest/pub/media/image/png/shot0008.png
HTTP/1.1 Host: localhost:8443
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate
Connection: keep-alive
If-None-Match: "1712108524"
Cache-Control: max-age=0
.
HTTP/1.1 304 Not Modified
Server: Apache-Coyote/1.1
Cache-Control: private, no-transform, max-age=1800
Expires: Thu, 01 Jan 1970 01:00:00 CET
Etag: "1712108524"
Date: Tue, 28 Apr 2015 11:23:37 GMT
```

[3] https://de.wikipedia.org/wiki/Unixzeit
[4] www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.19

2.2 Event

eventID - Die eindeutige ID des Events
name - Der Name des Events
description - Ein kurzer Info Text zum Event
startDate - Datum beginn des Events
endDate - Datum letzter Tag des Events (kann gleich wie »from« Feld sein)
active - Gibt an ob der Event aktiv sein soll. Falls ein Event nicht aktiv ist, wird er im Frontend nicht angezeigt

```
{
        "eventID": 1,
        "name": "test",
        "description": "desc",
        "from": "2015-06-10",
        "to": "2015-06-11",
        "active": true
}
```

pub/event
GET - Liefert alle Events

pub/event/{id}
GET - Liefert den besagten Event

pub/event/{id}/eventrooms
GET - Liefert alle Event Rooms welche zu Event {id} gehören

pub/event/{id}/eventitems
GET - Liefert alle Event Items welche zu Event {id} gehören

adm/event
GET - Liefert alle Events
POST - Erstellt neuen Event

adm/event/{id}
GET - Liefert den besagten Event
PUT - Updatet den besagten Event
DELETE - Löscht den besagten Event

adm/event/{id}/eventrooms
GET - Liefert alle Event Rooms welche zu Event {id} gehören

adm/event/{id}/eventitems
GET - Liefert alle Event Items welche zu Event {id} gehören

2.3 EventItem

eventItemID - Die eindeutige ID des Event Item
name - Der Name des Item
description - Die Beschreibung des Item
date - Das Datum an welchem das Item stattfindet
startTime - Der Beginn des Item
endTime - Das Ende des Item
roomIDFK - Ein FK zu dem Raum in welchem das Item stattfindet
eventIDFK - Ein FK zu dem Event in welchem das Item stattfindet
speakerIDFK - Ein FK zu dem Speaker welcher das Item moderiert

```
{
        "eventItemID": 1,
        "name": "Advanced Cryptography",
        "description": "description of item..",
```

```
        "date": "2015-06-15",
        "startTime": "13:00",
        "endTime": "14:15",
        "roomIDFK": 1,
        "eventIDFK": 1,
        "speakerIDFK": 1
}
```

pub/eventitem
GET - Liefert alle Event Items

pub/eventitem/{id}
GET - Liefert besagtes Event Item

adm/eventitem
GET - Liefert alle Event Items
POST - Erstellt neues Event Item

adm/eventitem/{id}
GET - Liefert besagtes Event Item
PUT - Updatet besagtes Event Item
DELETE - Löscht besagtes Event Item

2.4 EventRoom

eventRoomID - Die eindeutige ID des Raumes
name - Die Raumbezeichnung
location - **[Null]** Eine genauere Bezeichnung zum Raum
eventIDFK - Ein FK zu dem Event in welchem der Raum verwendet wird

```
{
        "eventRoomID": 1,
        "name": "1.206",
        "location": "1",
        "eventIDFK": 1
}
```

pub/eventroom
GET - Liefert alle Event Rooms

pub/eventroom/{id}
GET - Liefert den besagten Event Room

adm/eventroom
GET - Liefert alle Event Rooms
POST - Erstellt neuen Event Room

adm/eventroom/{id}
GET - Liefert den besagten Event Room
PUT - Updatet den besagten Event Room
DELETE - Löscht den besagten Event Room

2.5 Media

mediaID - Die eindeutige ID jedes Medien Objektes
type - Der Kennzeichner des Medien Objektes. »jpeg«, »png« (TODO weitere Typen wie Video etc.)
link - Ein absoluter Link zur Ressource

```
{
        "mediaID": 1,
```

```
        "type": "jpg",
        "link": "http://localhost:8080/hlmng/rest/media/jpeg/1.jpg",
}
```

pub/media
GET - Liefert alle Medien

pub/media/{id}
GET - Liefert das besagte Medien Objekt

pub/media/{typ}/{filename}
GET - Liefert die gewünschte Medien Ressource
z.B. http://.../rest/media/image/png/image.png"

pub/media/upload
POST - **S** - Ermöglicht einen Upload von Media Dateien mittels Multipart-Data [5].
Die möglichen Return Codes sind (nebst den üblichen Authorization needed / Bad Request).

- 500 - Server konnte das File nicht speichern
- 415 - Falscher / fehlender / unbekannter Mimetype
- 422 - Datei mit gleichem Namen existiert bereits
- 413- Datei zu gross
- 200 - Hochgeladen, Objekt wird als JSON geliefert, sodass sich der Client die ID speichern kann

adm/media
GET - Liefert alle Medien

adm/media/{typ}/{filename}
GET - Liefert die gewünschte Medien Ressource
z.B. http://.../rest/media/image/png/image.png"

adm/media/upload
Gleich wie /pub/media/upload, ohne Authentisierung

adm/media/{id}
GET - Liefert das besagte Medien Objekt
DELETE - Löscht die verlinkte Medien Datei, der Eintrag in der Datenbank wird **nicht gelöscht.**
Dies da auf das Medien Objekt verlinkende Objekte weiterhin den Link benutzen können und dieser dann ein 404 zurückliefert.

2.6 News

newsID - Die eindeutige ID zu dem News Eintrag
title - Der Titel des News Eintrag
text - Der Text des News Eintrag
media - Der direkte Link zur angehängten Medien Datei
author - Der Autor (frei wählbarer String)
mediaIDFK - **[Null]** Ein FK zu der angehängten Medien Datei
eventIDFK - Ein FK zu dem Event zu welchem die News gehören

```
{
        "newsID": 1,
        "title": "Breaking news!"
        "text": "Team Switzerland takes the lead!",
        "media": "http://localhost:8080/hlmng/rest/media/jpg/1.jpg",
        "author": "Max Muster",
```

[5] https://stackoverflow.com/questions/913626

```
        "mediaIDFK": 1,
        "eventIDFK": 1
}
```

pub/news
GET - Liefert alle News Einträge

pub/news/newest
GET - Liefert nur die neusten (höchste ID zuerst) 15 (Standardwert) Einträge. Es wird empfohlen diesen Call zu werden, ausser der User wünscht explizit alle Einträge.

pub/news/{id}
GET - Liefert den besagten News Eintrag

adm/news
GET - Liefert alle News Einträge
POST - Erstellt neuen News Eintrag

adm/news/newest
GET - Liefert nur die neusten (höchste ID zuerst) 15 (Standardwert) Einträge. Es wird empfohlen diesen Call zu werden, ausser der User wünscht explizit alle Einträge.

adm/news/{id}
GET - Liefert den besagten News Eintrag
PUT - Updatet den besagten News Eintrag
DELETE - Löscht den besagten News Eintrag

2.7 Presentation

presentationID - Die eindeutige ID zu der Präsentation
name - Der durch das Team gewählte Name der Präsentation
teamName - Der Teamname
date - Das Datum an welchem die Präsentation durchgeführt wurde
duration - **[!] [Null]** Die Dauer der Präsentation

```
{
        "presentationID": 1,
        "name": "presentation 1",
        "teamName": "1336+1",
        "date": "2015-05-10",
        "duration": "00:12:35"
}
```

pub/presentation
GET - Liefert alle Präsentationen

pub/presentation/{id}
GET - Liefert die besagte Präsentation

adm/presentation
GET - Liefert alle Präsentationen
POST - Erstellt neue Präsentation

adm/presentation/{id}
GET - Liefert die besagte Präsentation
PUT - Updatet die besagte Präsentation
DELETE - Löscht die besagte Präsentation

2.8 Push

pushID - Die eindeutige ID des Push
title - Der Titel der Push Nachricht
author - Der Autor (frei wählbarer String)
date - Das Datum an dem die Nachricht gesendet wurde
time - Der Zeitpunkt an dem die Nachricht gesendet wurde
receivedCounter - Die Anzahl Mobile Apps welche die Nachricht erhalten haben
failedCounter - Die Anzahl Mobile Apps an welche die Nachricht nicht gesendet werden konnte (Details siehe Log)

```
{
        "pushID": 1,
        "title": "push title",
        "author": "mmuster",
        "date": "2015-05-20",
        "time": "14:15",
        "receivedCounter": 10,
        "failedCounter" : 2
}
```

pub/push
GET - Liefert alle Push Einträge

/push/{id}
GET - Liefert besagten Push Eintrag

adm/push
GET - Liefert alle Push Einträge
POST - Erstellt neuen Push Eintrag
Der Post sollte etwa wie folgt aussehen:

```
{
        "author": "mmuster",
        "text": "it works",
        "title": "unbelievable"
}
```

adm/push/{id}
GET - Liefert besagten Push Eintrag
PUT - Updatet besagten Push Eintrag *****
DELETE - Löscht den besagten Push Eintrag *****
***** Die Push Notifaction auf dem Mobile App kann logischerweise nicht nachträglich geändert / gelöscht werden.

2.9 QR Code

qrCodeID - Die eindeutige ID des QR Code
createdAt - **[!]** Der Erstellungszeitpunkt des QR Code
claimedAt - **[!] [Null]** Der Zeitpunkt an dem der QR Code eingelöst wurde
payload - **[!] [Unique]** Der geheime Wert eines QR Code.
role - Die Rolle welche der QR Code legitmiert [jury,author]
userIDFK - **[Null]** Ein FK zu dem User welcher den QR Code »geclaimt« hat
eventIDFK - Ein FK zu dem Event in welchem der QR Code gültig ist

```
{
        "qrCodeID": 1,
        "createdAt": "2015-06-21 22:59:59",
        "claimedAt": "2015-06-21 23:09:13",
        "role": "jury",
        "userIDFK": 1,
        "eventIDFK": 1
}
```

adm/qrcode
GET - Liefert alle QR Codes
POST - Erstellt neuen QR Code

adm/qrcode/{id}
GET - Liefert den besagten QR Code
PUT - Updatet den besagten QR Code
DELETE - Löscht den besagten QR Code

adm/qrcode/{id}/render
GET - Liefert den QR Code als Bild zurück

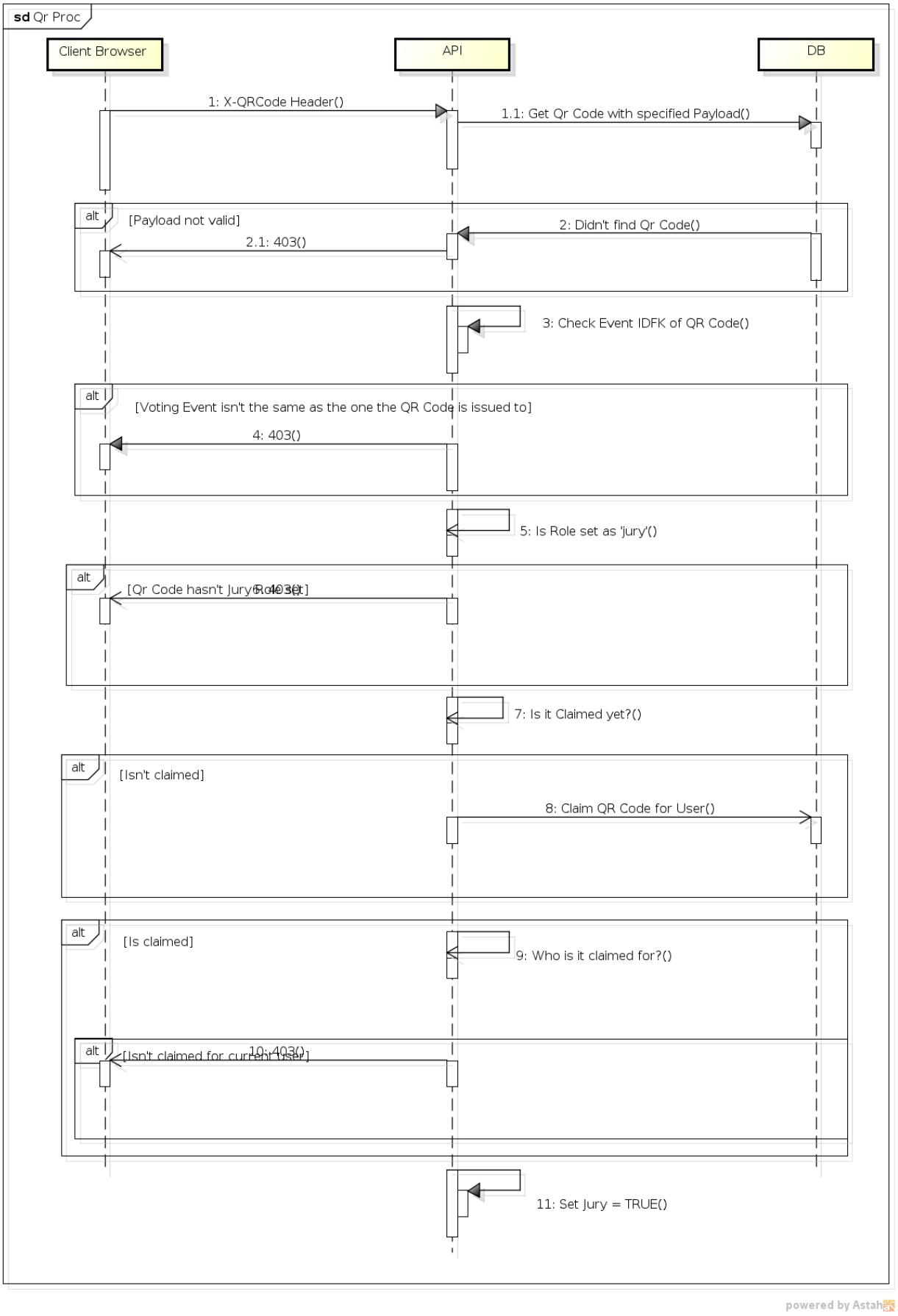
2.9.1 QR Code Payload

Der QR Code dient als geheimer Schlüssel, welche ein Mobile App User einlesen kann. Aus dem QR Code gewinnt man die Payload, welche als geheimer Schlüssel dient um beim Server gewisse Privilegien (siehe Rolle) zu erlangen. Gleichzeitig enthält die Payload Informationen für das Mobile App, welche die Rolle sowie den Gültigkeitsbereich festlegen. Dies ist so gelöst, da kein Mobile App User auf die QR Codes zugreifen darf und so auch keine Metainformationen zu dem eingescannten QR Code erhalten kann.

Payload:
[role]-[eventId]-[secret]
Role - Siehe Feld oben »role«
EventId - Die ID zu dem Event welcher der QR Code berechtigt ist
Secret - 20 Zeichen zufälliger geheimer Schlüssel

2.9.2 QrCode Überprüfung

Folgendes Diagramm soll den Ablauf und die möglichen Return Codes aufzeigen,



2.10 Slider

sliderID - Die eindeutige ID des Slider
name - Die Bezeichnung des Slider
weigth - Die Gewichtung des Sliders
votingIDFK - Ein FK zu dem dazugehörigen Voting

```
{
        "sliderID": 1,
        "name": "Language",
        "weigth": 2,
        "votingIDFK": 1
}
```

pub/slider
GET - Liefert alle Slider

pub/slider/{id}
GET - Liefert dem besagten Slider

adm/slider
GET - Liefert alle Slider
POST - Erstellt neuen Slider

adm/slider/{id}
GET - Liefert dem besagten Slider
PUT - Updatet dem besagten Slider
DELETE - Löscht dem besagten Slider

adm/slider{id}/votes
GET - Liefert alle Votes zu dem besagten Slider

2.11 Social

socialID - Die eindeutige ID des Social Eintrages
text - Der Text des Eintrages
status [!] - Der Moderationsstatus [pending,accepted,rejected]
authorName [!] - Der Username des Social Items, falls der User nicht gefunden wird ist das Feld »unknown«
media - **[Null]** Der direkte Link zu der angehängten Medien Datei (falls vorhanden)
userIDFK - Ein FK zu dem User der den Social Eintrag erstellt hat
mediaIDFK - **[Null]** Ein FK zu dem Medien Objekt welches angehängt sein kann
eventIDFK - Ein FK zu dem Event in welchem der Social Eintrag erstellt wurde

```
{
        "socialID": 1,
        "text": "Nice presentation!",
        "status": "pending",
        "authorName": "Harrison Ford",
        "media": "http://localhost:8080/hlmng/rest/media/jpg/1.jpg",
        "userIDFK": 1,
        "mediaIDFK": 1,
        "eventIDFK": 1
}
```

pub/social
GET - Liefert alle Social Einträge
POST - **S** - Erstellt neuen Social Eintrag, immer mit dem Moderationsstatus »pending«

pub/social/{id}
GET - Liefert den besagten Social Eintrag

pub/social/newest
GET - Liefert nur die neusten (höchste ID zuerst) 15 (Standardwert) Einträge. Es wird empfohlen diesen Call zu werden, ausser der User wünscht explizit alle Einträge.

adm/social
GET - Liefert alle Social Einträge
POST - Erstellt neuen Social Eintrag

adm/social/{id}
GET - Liefert den besagten Social Eintrag
PUT - Updatet den besagten Social Eintrag
DELETE - Löscht den besagten Social Eintrag

adm/social/newest
GET - Liefert nur die neusten (höchste ID zuerst) 15 (Standardwert) Einträge. Es wird empfohlen diesen Call zu werden, ausser der User wünscht explizit alle Einträge.

2.12 Speaker

speakerID - Die eindeutige ID des Speaker
name - Der Vor- und Nachname des Speaker
description - Ein kurzer Info Text zum Speaker
media - **[Null]** Der direkte Link zu der angehängten Medien Datei (falls vorhanden)
nationality - Das Kürzel zu dem Heimastland des Speaker. **Achtung:** Die Nationality wird immer in toUpperCase umgewandelt.
title - **[Null]** Der akademische Titel des Speaker, falls vorhanden
mediaIDFK - Ein FK zu dem Speaker Bild

```
{
        "speakerID": 1,
        "name": "Richard Stallman",
        "title": "PhD MIT",
        "media": "http://localhost:8080/hlmng/rest/media/jpg/1.jpg",
        "description": ".. freedom activist and computer programmer ..",
        "nationality": "USA"
        "mediaIDFK": 1
}
```

pub/speaker
GET - Liefert alle Speaker

pub/speaker/{id}
GET - Liefert den besagten Speaker

adm/speaker
GET - Liefert alle Speaker
POST - Erstellt einen neuen Speaker

adm/speaker/{id}
GET - Liefert den besagten Speaker
PUT - Updatet den besagten Speaker
DELETE - Löscht den besagten Speaker

2.13 User

userID - Die eindeutige ID welche den Benutzer identifiziert
name - Der Benutzername

deviceID - Die eindeutige ID jedes Android Gerätes
regID - Die Registrations ID bei dem Google Cloud Messaging Dienst

```
{
        "userID": 3,
        "name": "hmuster",
        "deviceID": "1234567890987654321",
        "regID": "1aa23cd45ef6789fg098hk765432ff1..."
}
```

pub/user
POST - Erstellt neuen user. Falls der Username bereits existiert wird 422 (unprocessable entity / exists) zurückgeliefert.

pub/user/{id}/changeregid
PUT - **S** - Updatet die »regID« des Users welcher sich anmeldet. Ein im JSON abweichender Username wird zurückgewiesen. Alle Variablen ausser »regID« werden ignoriert.

adm/user
GET - Liefert alle user
POST - Erstellt neuen user. Falls der Username bereits existiert wird 422 (unprocessable entity / exists) zurückgeliefert.

adm/user/{id}
GET - Liefert den besagten User
PUT - Updaten den besagten User
DELETE - Löscht den besagten

2.14 Vote

voteID - Die eindeutige ID des Vote (= eine Stimme)
score - Die abgebende Bewertung, beginnend bei 0 bis zu sliderMaxValue von Voting
isJury - Sagt aus ob die Vote von einem Jury Mitglied gemacht wurde oder nicht
sliderIDFK - Ein FK zu dem Slider über den die Bewertung abgeben wurde
userIDFK - Ein FK zu dem Benutzer welcher »gevotet« hat

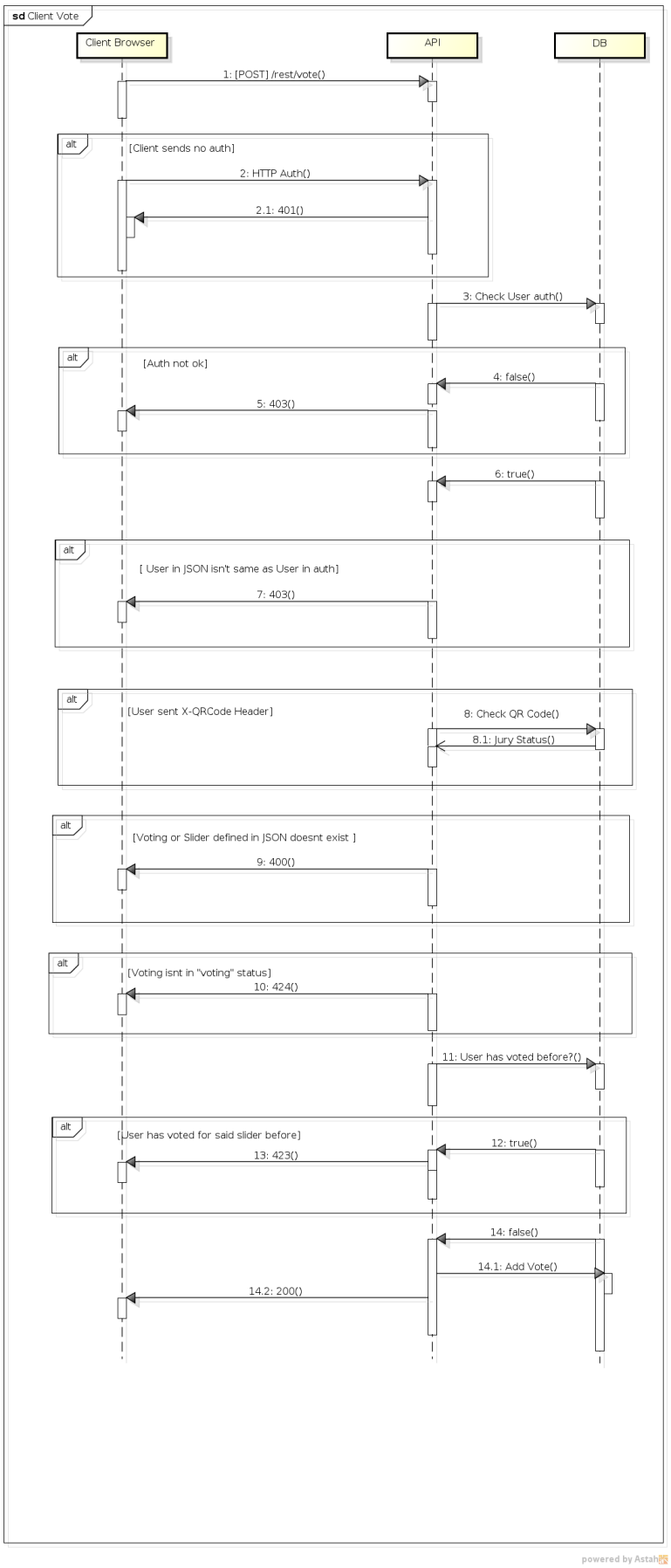
```
{
        "voteID": 1,
        "score": 7,
        "isJury": true,
        "sliderIDFK": 1,
        "userIDFK": 1
}
```

pub/vote
POST - **S** - **Q** - Erstellt neuen Vote
Es ist dabei nur ein Vote pro User für ein Slider möglich, bei erneutem Versuch wird HTTP Code 423 (locked) zurückgegeben.

adm/vote/{id}
GET - Liefert den besagten Vote
PUT - Updatet den besagten Vote

2.14.1 Vote Ablauf

Folgendes Diagramm soll den Ablauf und die möglichen Return Codes aufzeigen,



2.15 Voting

votingID - Die eindeutige ID des Votings
name - Der Name des Votings
juryCount - Die Anzahl anwesender Jury Mitglieder
status - Der Status des Voting (pre_presentation,presentation,presentation_end,voting,voting_end)
sliderMaxValue - Der maximale Wert der Slider
votingStarted - **[Null]** Der Zeitpunkt an welchem das Voting gestartet wurde (also status=voting)
votingDuration - Die Zeit in der, nach dem erhaltenen Push, abgestimmt werden darf
arethmeticMode - Die Auswahl der mathematischen Funktion zur Auswertung (median,todo)
round - Die Runde **TODO**
presentationIDFK - Ein FK zu der Präsentation über welche abgestimmt wird

```
{
        "votingID": 1,
        "name": "Voting 1",
        "juryCount": 15,
        "status": "running",
        "votingStarted": "14:04:27",
        "votingDuration": "00:00:50",
        "sliderMaxValue": 10,
        "arethmeticMode": "median",
        "round": 1,
        "presentationIDFK": 1
}
```

pub/voting
GET - Liefert alle Votings

pub/voting/{id}
GET - Liefert besagtes Votings

pub/voting/{id}/sliders
GET - Liefert alle Slider des besagten Voting

adm/voting
GET - Liefert alle Votings
POST - Erstellt neues Voting

adm/voting/{id}
GET - Liefert besagtes Voting
PUT - Updatet besagtes Voting

adm/voting/{id}/sliders
GET - Liefert alle Slider des besagten Voting

adm/voting/{id}/votes
GET - Liefert alle Votings aller Slider des Voting

2.15.1 Voting Triggers

Falls ein Voting durch einen Put Befehl in den Zustand »presentation_end« oder »voting« gesetzt wird, werden die Mobile App's per Push über diese Änderung informiert. Der Push der automatisch versendet wird, sieht wie folgt aus:

```
{
        "author": "vote_event",
        "date": "2015-05-06",
        "failedCounter": 2,
        "pushID": 9,
```

```
        "receivedCounter": 1,
        "text": "{ \"votingID\": 0 , \"name\": \"TEST\" }",
        "time": "16:24:55",
        "title": "presentation_end"
}
```

Das Feld »author« bezeichnet dabei, dass es sich nicht um einen regulären Push handelt und enthält »vote_event« als Wert. Das Feld »title« bezeichnet dabei den Zustand (also presentation_end oder voting). Das Feld Text enthält wiederum als JSON in einem String die notwendigen Metainformationen wie die ID und Namen des gestarteten Voting.

2.16 Time

/pub/time
Liefert die aktuelle Serverzeit im Format HH-mm-ss.SSS.

3 Fussnoten Index