Systemdesign und technische Dokumentation Personalisierte Stationen

Jens Gruschel, contexagon i.A. Fasnachtsmuseum Schloss Langenstein
(jens.gruschel@contexagon.com)

Dieses Dokument ist entstanden im Verbundprojekt museum4punkt0 – Digitale Strategien für das Museum der Zukunft, Teilprojekt M4 Kulturgut Fastnacht digital, Modul 4. Weitere Informationen: www.museum4punkt0.de

Fasnachtsmuseum Schloss Langenstein

März 2020

Gefördert durch:

Die Beauftragte der Bundesregierung
für Kultur und Medien

aufgrund eines Beschlusses
des Deutschen Bundestages

Lizenz und Hinweis zur Nachnutzung

Change Log

Date	Version	Author
13.01.2020	0.1 Initiale Version	JG
25.03.2020	0.2 Objektbeschreibung, Bibliotheken (mit Lizenzen)	JG
30.03.2020	0.3 Schnittstellenbeschreibung	JG
31.03.2020	1.0 Finalisierung	JG

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument beschreibt das Ssystemdesign und dokumentiert die technische Umsetzung des im Rahmen des Teilprojekts M4 museum4punkt0 für das Fasnachtsmuseum Schloss Langenstein implementierte Modul 4 «Personalisierte, interaktive und individualisierte Museumsfuhrungen in sensorischen Räumen».

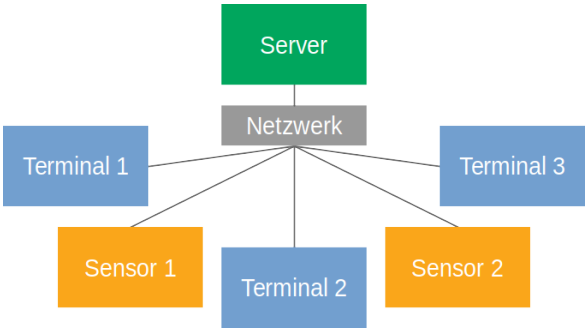
Die Besucher*innen sollen abhängig von ihrem Alter, ihrem Kenntnisstand, ihren persönlichen Interessen usw. individuell angepasste Inhalte präsentiert bekommen. Auch die Art und Weise des Besuchs selbst (z.B. ob ausgesuchte Bereiche betreten wurden oder gestellte Quizfragen beantwortet wurden) kann so Einfluss auf die präsentierten Inhalte haben.

Dazu erhalten die Besucher*innen auf Bluetooth basierende Badges zum Umhängen, die von Sensoren an den einzelnen Stationen erkannt werden können. Diese Sensoren sind genau so wie die einzelnen Stationen untereinander vernetzt, so dass jeweils passende Inhalte abgerufen werden können, nachdem die jeweiligen Besucher*innen über eine eindeutige Kennung, die jedes Bluetooth-Badge sendet, erkannt worden sind.

3D-Druck einer Maske als Badge mit integriertem Bluetooth-Beacon vor einem vernetzten Touch-Table, der auf Annäherung der Besucherin oder des Besuchers reagiert (und in diesem Fall die gewählte Maske automatisch in die Auswahl auf der rechten Seite übernommen hat)

2. Systemdesign

2.1. Systemaufbau



Sämtliche Clients (Terminals) und Sensoren sind via Netzwerk mit einem zentralen Server verbunden. Die einzelnen Komponenten kommunizieren dabei über HTTP und Websockets.

2.2. Server

Der Server stellt seine Funktionen (insbesondere die Storytelling-Engine) über eine REST-Schnittstelle (HTTP) zur Verfügung und stellt auf diese Weise auch Inhalte zur Verfügung, die von den Clients abgerufen werden können. Parallel dazu können sich Clients über Websockets verbinden, über die Informationen an die Clients gesendet werden, insbesondere um Status-Aktualisierungen zu signalisieren.

2.3. Clients (Terminals)

Clients sind z.B. handelsübliche PCs mit Linux oder Windows, Tablets (iPad oder Android) oder Einplatinencomputer (z.B. Raspberry Pi), die jeweils einen Bildschirm ansteuern, im Prinzip aber jegliches Gerät, auf dem ein moderner Webbrowser lauffähig ist. Einzige benötigte Software auf diesen Geräten ist dementsprechend ein Webbrowser (Firefox oder Chrome).

Die Client-Software selbst ist eine Web-Applikation, die vom Server bereitgestellt wird, und auf dem Client im Vollbild-Modus (Kiosk-Modus des Browsers) läuft. Diese Web-Applikation übernimmt wiederum die weitere Kommunikation mit dem Server (über REST-Schnittstelle bzw. Websockets), insbesondere um Inhalte (Texte, Bilder, Videos usw.) darzustellen und Benutzer*innen-Eingaben an den Server weiterzugeben. Aufgerufen wird der Client im Webbrowser über folgende URL:

`http://SERVERADRESSE:PORT?cbox=CLIENT-NAME`

Der Client-Name muss dabei im System hinterlegt sein (als „CBox“).

2.4. Sensoren

Auch die Sensoren kommunizieren mit dem Server über eine REST-Schnittstelle und gegebenenfalls über Websockets, insbesondere für Benachrichtigungen an den Server, wenn der Sensor ausgelöst wird (z.B. wenn jemand den überwachten Bereich betritt). Obwohl prinzipiell beliebige Geräte denkbar sind, die sich vernetzen lassen, kommen in der Praxis bisher Raspberry Pis zum Einsatz, die gemessene Signale an den Server weiterleiten.

3. Funktionalität

3.1. Datenbank

Im Backend kommt eine MongoDB-Objektdatenbank zum Einsatz. In sehr flexibel anpassbaren Datenstrukturen werden hier nicht nur Exponate und Mediendateien[1] (Bilder, Videos etc.) beschrieben, aber auch Objekte zur Steuerung der Stationen inklusive Objekte, die das Storytelling beschreiben (insbesondere „Stories“ und „Szenen“), sowie Objekte zur Verwaltung des Systems (z.B. User).

3.2. Zugriffsrechte

3.2.1. Nutzer und Rollen

Nutzer*innen des Systems sind einer oder mehreren Rollen zugeordnet (z.B. Administrator, Kurator etc.). Zugriffsrechte lassen sich global per Rolle definieren, sowie hierarchisch per Objekttyp oder konkretem Objekt (und in Zukunft denkbar auch für einzelne Felder).

Derjenige Nutzer, der ein Objekt erstellt hat (*creator*), hat prinzipiell Vollzugriff auf das Objekt, d.h. sämtliche Einschränkungen, die im folgenden beschrieben werden, sind in diesem Fall hinfällig.

3.2.2. Zugriffslevel

Prinzipiell gibt es drei verschiedene Zugriffslevel (*accessibility*): *hidden*, *readonly* und *writable* (in dieser Reihenfolge).

Jedem Objekttyp und ausgewählten konkreten Objeken kann eine allgemeine *accessibility* zugeordnet werden, wobei die Angabe optional ist und defaultmäßig das höchste Level *writable* angenommen wird (d.h. dass man explizit Einschränkungen machen muss, um den Zugriff zu beschränken).

Jeder Rolle ist zudem ein Standard-Zugriffslevel zugeordnet, so dass man bestimmten Nutzern den Zugriff auf alle Objekte grundsätzlich einschränken kann, es sei denn es existiert eine explizite Freigabe für eine Rolle, die dem jeweiligen Nutzer zugeordnet ist.

Prinzipiell haben die so allgemein hinterlegten Zugriffslevel also *einschränkenden* Charakter.

3.2.3. Explizite Freigaben

Darüber hinaus kann man einzelnen Rollen pro Objekttyp und konkretem Objekt ausnahmsweise höhere Zugriffslevel einräumen, indem man die Rollen hinterlegt als readableFor oder writeableFor (letzteres impliziert immer auch Lesbarkeit).

So lassen sich bestimmte Objekte oder Felder solchen Nutzern zugänglich machen, die eigentlich nur über beschränkte Zugriffsrechte verfügen.

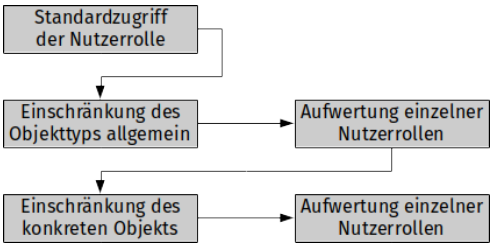
Prinzipiell haben die so hinterlegten expliziten Freigaben also *aufwertenden* Charakter.

---

[1] nur die Beschreibungen der Dateien sind Objekte in der Datenbank, die Dateien selbst werden dagegen im Dateisystem des Servers abgelegt (oder es wird via URL auf Dateien auf anderen Servern oder im Internet verwiesen)

3.2.4. Objektzugriff

Konkret ist der Zugriff auf ein Objekt folgendermaßen geregelt:



- Wenn ein neues Objekt angelegt wird, spielen nur die Berechtigungen eine Rolle, die für den jeweiligen Objekttyp hinterlegt sind.
- Beim Zugriff auf bereits existierende Objekte werden zunächst die Berechtigungen berücksichtigt, die für die Rolle des Benutzers und den jeweiligen Objekttyp hinterlegt sind, dann die für das konkrete Objekt (falls vorhanden).
- Dabei wird wird das Zugriffslevel von der Rolle über den Objekttyp bis evtl. zum konkreten Objekt immer weiter eingeschränkt.
- Erweitern lässt sich der Zugriff allerdings jeweils über explizite Freigaben für ausgewählte Rollen (auch hier werden zunächst die Freigaben berücksichtigt, die über den Objekttyp geregelt sind, dann die des Objekts).
- Mit diesem Prinzip erfolgen *Einschränkungen* und *Aufwertungen* des Zugriffslevels also abwechselnd (generelle Einschränkung für den Objekttyp, explizite Aufwertungen für den Objekttyp, generelle Einschränkung für das konkrete Objekt, explizite Aufwertungen für das konkrete Objekt, dasselbe evtl. für einzelne Fieldsets und Felder).
- Ausnahme: Hat der jeweilige Nutzer das Objekt selbst erstellt, hat er sowieso Vollzugriff

3.3. Wiedergabe von Inhalten

Inhalte werden prinzipiell in der Datenbank als solche hinerlegt und können unterschiedlichen Typs sein. Momentan implementiert sind „mediaitem“ (Bilder, Videos etc.), „textslide“ (Textfolien), „objectinfopanel“ (Anzeige von ausgewählten Feldern von Objekten) sowie „htmlcontent“ (externe Inhalte).

Clients (Terminals) können diese Inhalte wiedergeben, dargestellt über einen Webbrowser. Für jeden Client ist dazu ein ausgewählter Inhalt hinterlegt, der anfänglich dargestellt wird (siehe Kapitel über CBoxes). Durch die Storytelling-Engine können die Inhalte variabel ausgetauscht werden, etwa durch Interaktion mit den Besucher*innen (via Menüführung oder Chat) oder ausgelöst über Sensoren (Bewegungsmelder etc.). Definiert werden die Abläufe als Objekte in der Datenbank (insbesondere als „Stories“ und „Szenen“). Gesteuert werden die Terminals über den zentralen Server (nachdem dieser z.B. entsprechende Signale von Sensoren erhalten hat).

3.4. Menüführung

Auf jedem Client (Terminal) lassen sich parallel zu den Inhalten bis zu zwei Menüs darstellen (in zwei Slots „A“ und „B“). Diese werden sehr flexibel aus Einzelkomponenten wie „Labels“, „Buttons“, Abständen usw. aufgebaut sowie verschachtelbaren Untermenüs. So lässt sich ein Navigationsmenü für Inhalte genau so umsetzen wie einblendbare Steuerelemente etwa zum Neustarten eines Films. Individuell gestaltet werden können solche Menüs via CSS, hinterlegt über ein „Stylesheet-Objekt“ in der Datenbank.

ausgeklapptes Menü (Text und Design des Fasnachtsmuseums Schloss Langenstein)

3.5. Chat

Parallel zu anderen Inhalten (als „Overlay“) oder exklusiv lässt sich auch ein Dialog mit einem Chatbot auf den Clients einblenden. Textnachrichten verschickt dabei der Server in Reaktion auf (vordefinierte) User-Eingaben oder Sensorsignale. Definiert werden die Abläufe mit „Stories“ und „Szenen“ (siehe Storytelling). Mittels Stylessheet (CSS) kann die Optik frei gestaltet werden.

Chat mit User-Interaktion (Text und Design des Fasnachtsmuseums Schloss Langenstein)

Chat als Overlay vor einer Textfolie (Text und Design des Fasnachtsmuseums Schloss Langenstein)

3.6. Sensorik

Unterschiedliche Sensoren können an das System angeschlossen werden. Dies können Raspberry Pis sein, die Bluetooth-Beacons (von Besucher*innen umhängbare Badges) orten, oder an die z.B. Bewegungsmelder oder einfache Schalter angeschlossen sind.

Jeder Sensor ist in der Datenbank als Objekt mit eindeutigem Namen hinterlegt, der bei der Kommunikation mit dem Server verwendet werden muss.

Die Sensoren kommunizieren dabei mit dem Server über eine REST-Schnittstelle (siehe Schnittstellenbeschreibung). Der Server löst daraufhin „Stories“ aus (siehe Storytelling), über die einzelne Kommandos an die Clients (Anzeige von Inhalten, Chatnachrichten etc.) verschickt werden.

Manche Sensoren, etwa einfache Schalter, lösen nur bei Betätigung Aktionen aus. Andere Sensoren, etwa Bluetooth-Sensoren, können bei Annäherung andere Aktionen auslösen als bei Entfernung und zudem mehrere unterschiedliche Distanzbereiche abdecken.

Bei jeder „Story“, die für den Sensor hinterlegt ist, können dementsprechend separat Effekte für den Einstieg und Effekte für den Ausstieg der „Story“ definiert werden.

3.7. Storytelling

Zentraler Einstiegspunkt allen Geschehens sind die „Stories“. In den Story-Objekten sind ein oder mehrere Einstiegspunkte hinterlegt, die z.B. relevant werden, wenn Besucher*innen sich einer Station nähern, sowie unter Umständen Ausstiegspunkte für das Verlassen einer Station.

Einer Story ist dabei einer von vier möglichen Modi zugeordnet:

- Individuell: Effekte werden für alle Besucher*innen parallel ausgeführt, d.h. fünf gleichzeitige Beucher*innen erzeugen fünf Effekte.
- Exklusiv: Effekte werden nur für den ersten Besucher oder die erste Besucherin ausgeführt, danach ist die Station (bzw. die Story) bis zum Ausstieg dieses Besuchers oder dieser Besucherin für andere gesperrt, die leer ausgehen.
- Nacheinander: Wie beim exklusiven Modus erfolgt eine Sperrung der Story für andere. Es gibt jedoch eine virtuelle Warteschlange. Sobald die Story freigegeben wird, wird die Story sofort für den nächsten Besucher oder die nächste Besucherin aktiv, der oder die noch wartet. Dies ist z.B. sinnvoll für Info-Terminals, die von nur jeweils einer Person gleichzeitig verwendet werden können.
- Geteilt: Effekte werden nur für den ersten Besucher oder die erste Besucherin ausgeführt. Freigegeben wird die Story aber erst, wenn der letzte Besucher oder die letzte Besucherin den Bereich verlassen hat, und erst dann treten etwaige Effekte auf. Dies ist z.B. sinnvoll für Film-Präsentationen auf einer großen Leinwand, wenn der Film erst dann gestoppt werden soll, wenn der oder die letzte gegangen ist.

Sowohl für Einstieg als auch für Ausstieg lassen sich mehrere mögliche Varianten hinterlegen, von denen eine vom System ausgewählt wird. Diese Auswahl kann abhängig vom jeweiligen Besuchertyp erfolgen (so dass z.B. Kinder andere Inhalte präsentiert bekommen als Erwachsene) und / oder zufällig mit einstellbaren Wahrscheinlichkeiten.

Jede Variante mündet in den Aufruf einer oder mehrerer „Szenen“, die Bausteine für das Storytelling. Jede Szene kann eine Abfolge von Aktionen definieren. Auch bedingte Aktionen sind möglich („nur wenn der / die Besucher*in bereits hier oder dort war“). Dazu lassen sich Statusvariablen nutzen, die global oder pro Besucher existieren können. Die wichtigsten Aktionstypen sind folgende:

Aktionstyp	**Beschreibung**
Szenenwechsel	Sprung zu einer anderen Szene (sozusagen als Funktionsaufruf)
Content-Anzeige	Darstellung von Inhalten (z.B. Textfolien)
Medienwiedergabe	Wiedergabe von Bildern, Videos oder Audiodateien
Chat-Nachricht	Anzeige eines Texts und / oder Medien im Chat
Chat-Interaktion	Interaktion (Frage / Antworten) im Chat mit Verzweigung abhängig von der Auswahl der Besucher*innen
Menüanzeige	Anzeige eines neuen Menüs auf einem Terminal
Steuerbefehl	diverse Befehle z.B. um den Chat zu löschen, die Medienwiedergabe zu pausieren oder fortzusetzen, zur Steuerung des Bildschirmschoners etc.
Statusänderung	Zuweisung oder Addition / Subtraktion von Werten, die das System global oder pro Besucher*in speichert
bedingte Aktion	Verzweigung unter bestimmten Bedingungen (abhängig vom Status, von der aktuellen Uhrzeit oder vom Zufall)
Fallunterscheidung	Verzweigung ähnlich der bedingten Aktion, aber mit beliebig vielen Fällen (anstatt nur entweder / oder)

4. Implementierung

Der Software zu Grunde liegt ein System der contexagon GmbH, das im Rahmen des Verbundprojekts museum4punkt0 mit den notwendigen Erweiterungen versehen wurde, und dessen zum Betrieb der Stationen notwendigen Teile veröffentlicht wurden und im folgenden beschrieben werden.

4.1. Server

Der Server ist in *Python* implementiert und verwendet das Webframework *Flask* zur Bereitstellung seiner Schnittstelle, agiert also (auch) als Webserver. Als Datenbank kommt *MongoDB* zum Einsatz. Bereitgestellt wird die Server-Software alias *MusOS* über *Docker*.

Lauffähig ist die Server-Software unter *Linux* und *MacOS* (der Einsatz von *Windows* scheitert insbesondere am Dateisystem NTFS, das *MongoDB* unter Einsatz von *Docker* nicht unterstützt).

4.2. Clients (Terminals)

Die Client-Software läuft im Webbrowser (z.B. Firefox oder Chrome) und setzt auf dem JavaScript-Framework *React* auf. Für das State-Management wurde auf *Redux* gesetzt.

4.3. Sensoren

Die Software für die Sensoren wurde mit Python entwickelt und wird in einem separaten Dokument näher beschrieben.

4.4. Netzwerk

Obwohl ausschließlich Web-Technologien zum Einsatz kommen, wird kein Internet-Zugriff für den Betrieb benötigt, sondern die Kommunikation kann ausschließlich auf das lokale Netzwerk beschränkt sein (und genau so wurde es für das Fasnachtsmuseum umgesetzt).

Dennoch ist denkbar, den Server ohne Änderungen auch im Internet zu betreiben, etwa um von mehreren Standorten auf eine zentrale Datenbank zugreifen zu können. Alternativ kann der Server auch im lokalen Netzwerk betrieben werden, aber zusätzlich über den lokalen Internetanschluss (z.B. DSL-Router) nach außen geöffnet werden.

Auch zur Fernwartung hat sich dieses Konzept bewährt, denn es lassen sich einfach auf jedem PC im Webbrowser sämtliche Clients simulieren, einfach man dieselbe URL verwendet wie auf der tatsächlichen Station.

4.5. Bibliotheken

Folgende Bibliotheken von Dritten wurden für die Implementierung verwendet:

Bibliothek	Lizenz
Server	
Docker	Apache 2.0
MongoDB	Server Side Public License
Python 3	Python Software Foundation License
Requests	Apache 2.0
Flask	BSD 3-Clause
PyMongo	Apache 2.0
Flask-PyMongo	BSD 2-Clause
Flask-Cors	MIT
Flask-JWT-Extended	MIT
Flask-Bcrypt	BSD-artig
Flask-SocketIO	MIT
eventlet	MIT
pyinfo	MIT
Pillow	PIL Software License (MIT-artig)
opencv-python	MIT
mutagen	GPLv2+
cachelib	BSD 3-Clause
Client	
npm	Artistic License (BSD-artig)
react	MIT
react-dom	MIT
react-scripts	MIT
Redux	MIT
Redux Thunk	MIT
Material-UI (core/icons)	MIT
core-js	MIT
lodash	MIT
socket.io-client	MIT
React Redux	MIT
React DnD	MIT
react-markdown	MIT
react-pdf	MIT

5. Objektbeschreibung

5.1. Grundsätzlicher Aufbau

Alle Objekte, die in der Objektdatenbank gespeichert sind, liegen im JSON-Format vor (intern verwendet die Datenbank BSON, das minimale Unterschiede zu JSON aufweist). Grundsätzlich haben sämtliche Objekte folgenden Aufbau:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": OBJEKTTYP,
  "title": TITEL DES OBJEKTS,
  "fields": {
    FELD A: ERSTER WERT,
    FELD B: ZWEITER WERT,
    …
  },
  "tags": [ERSTES TAG, ZWEITES TAG, …]
}
```

Die eindeutige Objekt-ID wird vom Server automatisch vergeben (wenn nicht bereits vorhanden) und weist typischerweise folgenden Aufbau auf: Objekttyp gefolgt von einem Punkt und einer GUID oder einer anderen eindeutigen manchmal sprechenderen ID (z.B. `user.E972B60C-F8A9-4FFE-8736-3AF917EBFCCB` oder `user.admin`).

Intern weist die Datenbank jedem Objekt außerdem eine weitere ID (`_id`) zu, die aber nach außen normalerweise nicht sichtbar ist.

Außerdem reichert der Server die Objekte um eine Reihe von Zusatzinformationen an wie dem Zeitpunkt der Erzeugung und letzten Änderung, die URL eines Bildes für das Objekt usw., die hier der Übersichtlichkeit halber weggelassen werden (und je nach Art der Schnittstelle auch nicht unbedingt nach außen gereicht werden).

Die eigentliche Beschreibung des jeweiligen Objekts erfolgt innerhalb des Eintrags „fields“ und ist abhängig vom jeweiligen Objekttyp.

Es können außerdem beliebige Tags vergeben werden, mit Hilfe derer sich Objekte leichter auffinden oder (evtl. zusammengehörende) Objekte markieren lassen.

Optional können Objekte Zugriffsberechtigungen enthalten, die folgenden Aufbau haben:

```
{
  "objectaccess": {
    "accessibility": "readonly",
    "writeableFor": [ROLLEN-ID, …],
    "readableFor": [ROLLEN-ID, …]
  },
  …
}
```

5.2. Objekttypen

Es gibt eine Reihe vordefinierter Objekttypen und es lassen sich jederzeit neue, eigene Typen ergänzen. Die Beschreibung der Objekttypen ist ebenfalls in Form von Objekten in der Datenbank abgelegt mit folgenden Aufbau:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "type",
  "title": NAME DES OBJEKTTYPS,
  "fields": {
    "id": OBJEKTTYP-ID,
    "name": NAME DES OBJEKTTYPS,
    "parent": ELTERN-ID (OPTIONAL),
    "typeaccess": [
      {
        type: "access",
        "fields": {
          "accessibility": "readonly",
          "writeableFor": [ROLLEN-ID, …],
        }
      }
    ],
    "definition": {
      "fieldsets": [
        {
          "id": EINDEUTIGE SET-ID,
          "name": SET-NAME,
          "fields": [
            {
              "id": EINDEUTIGE FELD-ID,
              "name": FELDNAME,
              "type": FELDTYP,
              …
            },
            …
          ]
        }
      ],
      …
    }
  }
}
```

Hervorzuheben ist hier insbesondere der Eintrag „definition“, der die Definition der einzelnen Felder enthält, die in mehreren „Sets“ zusammengefasst werden können.

Abhängig vom jeweiligen Feldtyp („text“, „number“, „date“ usw.) sind unterschiedliche Attribute präsent, die im Folgenden beschrieben werden (und ausgelassen werden können, wenn der Standardwert gilt):

Attribut	Beschreibung
id	Feld-ID, muss zwingend angegeben werden und im Objekttyp eindeutig sein
name	Feldname, wird zu Anzeigezwecken verwendet
type	Datentyp des Feldes, muss zwingend angegeben werden, mögliche Werte: • text (Text) • number (Zahl) • int (ganze Zahl) • float (Gleitkommazahl) • date (Datum im ISO-8601-Format) • time (Uhrzeit im ISO-8601-Format) • datetime (Zeitstempel mit Datum und Uhrzeit im ISO-8601-Format) • bool (true oder false) • checkbox (entspricht bool) • select (Auswahl vorgegebener Werte) • element (eingebettete Objekte) • reference (Verknüpfung von anderen Objekten) • password (Kennwort) • url (URL z.B. einer Webseite) • mediaurl (URL eines Bildes, Videos etc.) • medialist (Verknüpfung von Medien-Objekten) • markdown (Text im Markdown-Format)
mandatory	Ist das Feld zwingend auszufüllen? Werte: true oder false (optional, Standard false)
default	Standardwert
maxlength	nur bei Textfeldern, maximale Länge des Texts
localizable	Soll das Feld verschiedene Werte für die definierten Sprachen halten können? Werte: true oder false (optional, Standard false)[2]
options	nur bei Feldern vom Typ „bool“ oder „checkbox“, Liste erlaubter Werte, z.B. ["left", "right"]
translations	nur bei Feldern vom Typ „bool“ oder „checkbox“, Übersetzung erlaubter Werte in verschiedene Sprachen, z.B. {"left": {"de": "links"}, "right": {"de": "rechts"}}
reftypes	nur bei Feldern vom Typ „element“ oder „reference“, Liste erlaubter Objekttypen
quantity	nur bei Feldern vom Typ „element“ oder „reference“, Wert „single“, wenn maximal ein Objekt hinterlegt werden darf
suffixes	Liste möglicher Suffixe, wird primär für Einheiten bei numerischen Feldern benötigt, z.B. `["mm", "cm", "m"]`
defaultSuffix	das vorgewählte Suffix aus der Liste „suffixes“

[2] nicht lokalisierbare Textfelder werden als normale Strings in der Datenbank gespeichert, z.B. `"Haus"`, lokalisierbare Textfelder werden als Objekte gespeichert, z.B. `{"de": "Haus", "en": "house"}`

5.3. User

Benutzer*innen sind ebenfalls als Objekte in der Datenbank hinterlegt:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "user",
  "title": USER-NAME,
  "fields": {
    "email": USER-NAME,
    "target": ZIEL-IDS ALS LISTE (OPTIONAL),
    "roles": [ERSTE ROLLEN-ID, ZWEITE ROLLEN-ID, …]
  }
}
```

Als Konvention entsprechen User-Namen E-Mail-Adressen. Zu einem späteren Zeitpunkt könnte das System auf diese Weise E-Mails an die einzelnen Benutzer*innen schicken, etwa wenn das eigene Kennwort vergessen wurde.

Die Kennwörter der Benutzer*innen sind separat als Hash-Werte in der Datenbank abgelegt und nicht im User-Objekt selbst. Als Besonderheit kann beim Erzeugen der User-Objekte jedoch ein virtuelles Feld „password“ angegeben werden, dass das Standard-Kennwort des Users enthält. Auf diese Weise lassen sich User einfacher automatisch anlegen. Das System speichert den entsprechenden Hash-Wert jedoch trotzdem separat und nicht im User-Objekt.

Jedem User können eine oder mehrere Rollen zugewiesen werden. Standarmäßig existieren Rollen mit folgenden IDs:

Rollen-ID	Beschreibung
role.standard	Standardnutzer mit Schreibzugriff auf alle nicht-administrativen Objekte
role.supervisor	Standardnutzer mit zusätzlichem Zugriff auf Überwachungsfunktionen
role.admin	Administrator mit Schreibzugriff auch auf administrative Objekte (darf z.B. Passwörter zurücksetzen)
role.guest	Standardnutzer ohne Schreibberechtigung (nur lesender Zugriff)
role.cbox	wird von den Clients (Terminals) verwendet, um Inhalte anzeigen zu können, nur Lesezugriff
role.curator	nur Zugriff auf Medien und Exponat-Datenbank
role.designer	nur Zugriff auf Gestaltungsobjekte (Medien, Stylesheets etc.)
role.scenography	nur Zugriff auf Medien, Sensorik und Storytelling
role.office	nur Zugriff auf Kontaktdatenbank und Terminals

5.4. CBoxes

Für jeden Client (also jedes Terminal) muss ein CBox-Objekt mit eindeutigem Namen angelegt werden, das folgendermaßen aussieht:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "cbox",
  "title": CLIENT-NAME,
  "fields": {
    "name": CLIENT-NAME,
    "description": BESCHREIBUNG,
    "channels": [ERSTE KANAL-ID, ZWEITE KANAL-ID, …],
    "backgroundcolor": HINTERGRUNDFARBE,
    "content": [OBJEKT-ID],
    "screensavertimeout": ZEIT IN MINUTEN,
    "screensavercontent": [OBJEKT-ID],
    "menustyle": [STYLESHEET-ID],
    "menuA": [ERSTE MENÜ-ID],
    "menuB": [ZWEITE MENÜ-ID],
    …
  }
}
```

Die meisten Felder sind optional. Zwingend anzugeben ist aber der eindeutige Name. Dieser muss identisch sein mit dem Namen, den man auf dem Endgerät im Webbrowser in der URL hinterlegt:

```
http://SERVERADRESSE:PORT?cbox=CLIENT-NAME
```

Als „content“ wird die ID des Objekts hinterlegt, die anfangs als Inhalt auf dem jeweiligen Client dargestellt werden soll. Mögliche Inhalte sind z.B. Objekte vom Typ „mediaitem“ (Bilder, Videos etc.), „textslide“ (Textfolien), „objectinfopanel“ (Anzeige von ausgewählten Feldern von Objekten) oder „htmlcontent“ (externe Inhalte).

Bis zu zwei Menüs lassen sich hinterlegen, mit denen sich eine interaktive Navigation realisieren lässt bzw. mit denen sich Buttons einblenden lassen (z.B. zur Sprachauswahl). Die Menüs sind ebenfalls als Objekte in der Datenbank hinterlegt, ebenso Stylesheets (wo sich CSS zur individuellen Gestaltung hinterlegen lässt).

5.5. Medien

Medien-Dateien (Bilder, Videos etc.) sind normalerweise auf dem Server abgelegt (Upload mittels API `/upload/media`, siehe dort) und werden dementsprechend vom Server über eine URL bereitgestellt (normalerweise in der Form `/uploads/xy.z`). Beschrieben werden die Dateien aber durch Objekte vom Typ „mediaitem“, die folgendermaßen aufgebaut sind:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "mediaitem",
  "title": TITEL,
  "fields": {
    "title": TITEL,
    "description": BESCHREIBUNG,
    "url": URL DER DATEI,
  },
  "imageurl": URL DES BILDES,
  "thumbnailurl": URL EINES THUMBNAILS,
  "mediainfo": {
    "type": MEDIENTYP,
    "width": BREITE DES BILDES ODER VIDEOS,
    "height": HÖHE DES BILDES ODER VIDEOS,
    "filesize": DATEIGRÖßE IN BYTES,
    "thumbnail": URL EINES THUMBNAILS,
    "fullhd": URL EINER VARIANTE MIT MAXIMAL FULL-HD-AUFLÖSUNG,
    "duration": DAUER DES VIDES IN SEKUNDEN,
    "suggestedFrame": URL EINES VIDEO-FRAMES,
    "frames": [URL EINES VIDEO-FRAMES, URL EINES WEITEREN FRAMES, …]
  },
}
```

Von den Benutzer*innen werden lediglich die Felder „title“, „description“, „url“ usw. ausgefüllt. Die Medien-Info sowie „imageurl“ und „thumbnailurl“ werden vom Server automatisch vergeben (sofern möglich[3]), nachdem die Medien-Datei analysiert wurde und gegebenenfalls Thumbnails, ausgesuchte Frames eines Videos und dergleichen automatisch erzeugt wurden.

Es existieren eine Reihe weiterer optionaler Felder für Quellenangaben, Aufnahmedatum usw.

[3] ist nur möglich für Dateien, die auf dem Server selbst abgelegt wurden, nicht für extern verlinkte Medien

5.6. Stories

Stories sind die Bindeglieder zwischen Auslösern wie Sensoren und den in den Szenen ausgeführten Aktionen und damit die Einstiegspunkte beim Storytelling. Aufgebaut sind Story-Objekte folgendermaßen:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "story",
  "title": NAME DER STORY,
  "fields": {
    "name": NAME DER STORY,
    "mode": MODUS,
    "target": ZIEL-IDS ALS LISTE (OPTIONAL),
    "entry": [
      {
        type: "sceneselection",
        "entry": {
          "scene": [SZENEN-ID],
          "probability": "50",
          "visitortype": [ERSTE ID, ZWEITE ID, …]
        }
      },
      …
    ],
    "exit": [
      {
        type: "sceneselection",
        "entry": {
          "scene": [SZENEN-ID],
          "probability": "100",
          …
        }
      },
      …
    ]
  }
}
```

Der Modus ist entweder `"individual"`, `"exclusive"`, `"queueing"` oder `"shared"` (siehe Storytelling).

Der Visitor-Typ bleibt leer, wenn der jeweilige Einstieg für sämtliche Besucher*innen gleichermaßen gelten soll. Das Feld „probability“ gibt bei mehreren Auswahlmöglichkeiten das Verhältnis der Wahrscheinlichkeiten an und kann sich der Intuition halber zu 100 (Prozent) aufsummieren (z.B. 50% zu 25% zu 25%), kann aber beliebige Werte annehmen (z.B. 2 zu 3).

Essentiell ist der Verweis auf eine oder mehrere Szenen, wo die eigentliche Aktion stattfindet.

5.7. Szenen

Beim Storytelling kommt Szenen eine besondere Bedeutung zu. Diese Objekte sind praktisch die Bausteine der Stories, in denen sämtliche Aktionen definiert werden, die unter bestimmten Bedingungen ausgeführt werden. Szenenobjekte haben folgenden Aufbau:

```
{
  "id": EINDEUTIGE OBJEKT-ID,
  "type": "scene",
  "title": NAME DER SZENE,
  "fields": {
    "name": NAME DER SZENE,
    "target": ZIEL-IDS ALS LISTE (OPTIONAL),
    "actions": [
      {
        type: "mediaaction",
        "fields": {
          "medialist": [MEDIEN-ID, …],
          "start": "afterprevious",
          "delay": 0,
          "duration": 30.0
        }
      },
      …
    ]
  }
}
```

Es gibt eine Reihe unterschiedlichster Aktionen, etwa zum Abspielen von Inhalten, zum Anzeigen von Chat-Nachrichten oder zum Aufruf weiterer Szenen (siehe Storytelling). Je nach Aktionstyp werden weitere Felder angegeben (wie die Medien-IDs im obigen Beispiel). Einige komplexere Aktionstypen steuern anhand von Bedingungen, ob weitere Aktionen ausgeführt werden oder nicht.

Allen Aktionen ist gemein, dass (optional) ein Timing hinterlegt werden kann. So können mehrere Aktionen in der Liste zeitlich nacheinander ausgeführt werden.

5.8. weitere Objekttypen

Sämtliche vordefinierte Objekttypen sind in JSON-Dateien hinterlegt und werden einmalig mittels eines Scripts (`init_db.sh`) in die Datenbank übertragen. Mittels dieser Dateien lässt sich der konkrete Aufbau der einzelnen Objekttypen nachvollziehen.

6. Schnittstellenbeschreibung

6.1. REST-API serverseitig

Der Server antwortet im Erfolgsfall mit dem HTTP-Statuscode 200 (oftmals in Verbindung mit einer Antwort im JSON-Format). Im Fehlerfall werden die üblichen HTTP-Statuscodes wie 400, 401, 404, 500 usw. zurückgemeldet zusammen mit einer Fehlerbeschreibung im JSON-Format in der Form:

```
{
  "error_id": FEHLER-ID ALS STRING,
  "message": FEHLERMELDUNG
}
```

Für bestimmte Funktionen sind gewisse Berechtigungen notwendig. Dazu muss der User sich zunächst anmelden (siehe unten bei `/login`) und erhält ein Zugriffs-Token (JSON Web Token, JWT), das beim Aufruf solcher Funktionen zusätzlich im HTTP-Header als „Bearer-Token“ übertragen werden muss:

```
Authorization: Bearer TOKEN
```

Bei fehlender Authorisierung oder ungültigem oder abgelaufenem Token antwortet der Server mit dem HTTP-Statuscode 401 (UNAUTHORIZED).

Folgende REST-Schnittstellen werden vom Server bereitgestellt:

- `POST /dbadmin/initdb`
  Datenbank initialisieren (muss einmalig aufgerufen werden)

- `POST /login`
  Benutzer anmelden
  als JSON übergebene Daten:
  ```
  {
    "user": USER-ID,
    "password": KENNWORT
  }
  ```
  als JSON zu erwartende Antwort (wird für andere Aufrufe benötigt):
  ```
  {
    "accessToken": TOKEN
  }
  ```

- `POST /resetuserpassword`
  User-Kennwort zurücksetzen
  als JSON übergebene Daten:
  ```
  {
    "userid": USER-ID,
    "password": KENNWORT
  }
  ```

- `POST /changeownpassword`
  User-Kennwort zurücksetzen
  als JSON übergebene Daten:
  ```
  {
    "oldpassword": ALTES KENNWORT,
    "newpassword": NEUES KENNWORT
  }
  ```

- `GET /typelist`
  alle Objekttypen auflisten (ohne Zugriffsbeschränkung)

- `GET /config`
  aktuelle globale Konfiguration abrufen

- `PUT /init/object`
  Objekt initialisieren (wird bei der Einrichtung der Datenbank verwendet, um Standardobjekte automatisiert zu erzeugen)
  als JSON übergebene Daten: das zu speichernde Objekt

- `PUT /object`
  neues Objekt hinzufügen oder vorhandenes Objekt ändern
  als JSON übergebene Daten: das zu speichernde Objekt

- `GET /object/`OBJEKT-ID
  Objekt mit der besagten ID abrufen
  als JSON zu erwartende Antwort:
  ```
  {
    "data": ANGEFRAGTES OBJEKT,
    "accesslevel": ZUGRIFFSLEVEL FÜR DEN ANGEMELDETEN USER,
    "fieldsetaccesslevels": ZUGRIFFSLEVEL DER EINZELNEN FIELD-SETS (EXPERIMENTELL)
  }
  ```

- `DELETE /object/`OBJEKT-ID
  Objekt mit der besagten ID unwiderruflich löschen

- `PUT /recycle/object/`OBJEKT-ID
  Objekt mit der besagten ID als gelöscht kennzeichen („in den Papierkorb werfen“)

- `GET /recycle/object/`OBJEKT-ID
  Objekt mit der besagten ID als nicht gelöscht kennzeichen („aus dem Papierkorb zurückholen“)

- `POST /objectquery`
  alle Objekte abrufen, die der übergebenen Anfrage entsprechen und für die der angemeldete User Zugriffsrechte besitzt
  als JSON übergebene Daten: Datenbankabfrage (siehe Dokumentation *MongoDB*[4]), z.B. für die Abfrage sämtlicher Objekte vom Typ „channel“:
  ```
  {
    "type": "channel"
  }
  ```

[4] https://docs.mongodb.com/manual/tutorial/query-documents/

- `POST /upload/media`
  Mediendatei (Bild, Video etc.) hochladen
  als JSON zu erwartende Antwort:
  ```
  {
    "url": DER DATEI ZUGEWIESENE URL,
    "info": META-INFORMATIONEN
  }
  ```
  Die zugewiesene URL hat typischerweise die Form `/uploads/xy.z`, außerdem werden Dateien erzeugt, die unter `/thumbnails/xy.z` oder `/fullhd/xy.z` oder `/videoframes/xy.z` erreichbar sind.

- `GET /logging/level/`LEVEL
  Logging-Level ändern entweder auf „debug“ oder auf „info“

- `GET /logging/get/`BYTES
  (ungefähr) die angegebene Datenmenge aus dem Log-File abrufen

- `GET /logging/server.log`
  das vollständige Log-File abrufen

- `GET /admin/mediainfodatabase/rebuild`
  die Medien-Info-Datenbank neu erzeugen (nur nötig nach manuellen Eingriffen)

- `GET /box/halt/`SESSION-ID
  besagten Client anhalten (dieser wird bis zum Neustart nicht mehr aktualisiert)

- `GET /alive/`SESSION-ID
  „Alive“-Signal für besagten Client an Server senden (erfolgt normalerweise über Websockets, kann aber auch via REST-Schnittstelle erfolgen)

- `GET /configuration/badgesensor/`SENSORNAME
  Informationen über besagten Bluetooth-Sensor abrufen
  als JSON zu erwartende Antwort:
  ```
  {
    "ranges": [
      {
       "nearthreshold": TRIGGER-ABSTAND,
       "farthreshold": UNTRIGGER-ABSTAND (HYSTERESE)
      }, …
    ],
    "signature": SIGNATUR
  }
  ```
  Diese API wird von den Sensoren aufgerufen. Die Signatur wird verwendet, um Änderungen an den Bereichen erkennen zu können.

- `GET /state/connections`
  alle aktuellen Socket-Verbindungen zum Server auflisten
  als JSON zu erwartende Antwort:

```
{
  "connections": [
    {
     "sid": SESSION-ID,
     "type": CLIENT-TYP,
     "name": CLIENT-NAME,
     "connected": VERBUNDEN JA/NEIN,
     "alive_time": ZEITPUNKT DER LETZTEN RÜCKMELDUNG,
     …
    }, …
  ],
  "time": AKTUELLE SERVER-ZEIT
}
```

- `GET /state/sensors`
  Status sämtlicher kürzlich verwendeter Sensoren abfragen
  als JSON zu erwartende Antwort:

```
{
  "badgesensorstatus": STATUS DER BLUETOOTH-SENSOREN (LISTE)
}
```

- `GET /state/stories`
  Status sämtlicher kürzlich aktivierter Stories abfragen
  als JSON zu erwartende Antwort:

```
{
  "storystatus": STATUS DER STORIES (LISTE)
}
```

- `GET /trigger/badgesensor/`SENSORNAME`/`BEREICH`?method=`METHODE`&signature=`SIGNATUR
  besagten Bluetooth-Sensor auslösen
  Diese API wird von den Sensoren aufgerufen. Der Bereich wird als Index übergeben, 0 für den ersten definierten Bereich, 1 für den zweiten Bereich usw. Die Methode ist entweder „enter“ (Bereich wurde betreten) oder „leave“ (Bereich wurde verlassen). Die Signatur muss zuvor via `/configuration/badgesensor/` ermittelt werden (der Aufruf schlägt fehl, wenn die Signatur nicht mehr gültig ist, d.h. wenn die Bereiche geändert wurden).

- `GET /trigger/sensor/`SENSORNAME
  einfachen Sensor (Schalter, Bewegungsmelder) auslösen
  Diese API wird von den Sensoren aufgerufen.

- `GET /box/`BOXNAME`/refresh`
  besagten Client initialisieren
  als JSON zu erwartende Antwort:

```
{
  "content": ANFÄNGLICHER INHALT,
  "referenced": LISTE WEITER ZUR DARSTELLUNG BENÖTIGTER OBJEKTE,
  "screensaver": {
    "timeout": WARTEZEIT IN SEKUNDEN,
    "content": SCREENSAVER-INHALT,
    "referenced": LISTE WEITER ZUR DARSTELLUNG BENÖTIGTER OBJEKTE
  },
  "properties": {
     "backgroundColor": HINTERGRUNDFARBE,
     "chat": CHAT-EIGENSCHAFTEN,
     "menu": MENÜ-EIGENSCHAFTEN,
     …
  },
  "configuration": {
     "logReduxActions": ACTION-LOGGING AN/AUS,
     "logReduxState": STATE-LOGGING AN/AUS,
     …
  }
}
```

- `GET /box/ping`
  Ping-Kommando auslösen (Ziel wird als Parameter übergeben, siehe unten)
- `GET /box/reload`
  Client neu laden (Ziel wird als Parameter übergeben, siehe unten)
- `GET /box/reset`
  Client ohne neu zu laden zurücksetzen (Ziel wird als Parameter übergeben, siehe unten)
- `GET /chat/message?text=`TEXT
  Chat-Nachricht ausgeben (Ziel wird als weiterer Parameter übergeben, siehe unten)
- `GET /notification/show?text=`TEXT
  Benachrichtigung anzeigen (Ziel wird als weiterer Parameter übergeben, siehe unten)
- `GET /play/mediaurl?url=`MEDIEN-URL
  Mediendatei (Bild, Video etc.) darstellen (Ziel wird als weiterer Parameter übergeben, siehe unten)
- `GET /play/object/`OBJEKT-ID
  im System hinterlegten Inhalt (Bild, Video, Textfolie etc.) darstellen (Ziel wird als Parameter übergeben, siehe unten)
- `POST /play/story?method=`METHODE`&visitortype=`BESUCHERTYP
  als JSON übergebene Story auslösen (Methode ist entweder „enter“ oder „leave“, Besuchertyp ist optional, Ziel wird als weiterer Parameter übergeben, siehe unten)

- `GET /play/scene/`SZENEN-ID`?visitortype=`BESUCHERTYP
  besagte Szene abspielen (Besuchertyp ist optional, Ziel wird als weiterer Parameter übergeben, siehe unten)

- `POST /play/scene?visitortype=`BESUCHERTYP
  als JSON übergebene Szene abspielen (Besuchertyp ist optional, Ziel wird als weiterer Parameter übergeben, siehe unten)

- `GET /play/pause`
  aktuelle Wiedergabe pausieren (nur bei Videos oder Audiodateien relevant)

- `GET /play/resume`
  aktuelle Wiedergabe fortsetzen (nur bei Videos oder Audiodateien relevant)

- `GET /play/stop`
  aktuelle Wiedergabe beenden (nur bei Videos oder Audiodateien relevant)

- `GET /play/volume?percent=`LAUTSTÄRKE IN PROZENT
  Wiedergabelautstärke anpassen (nur bei Videos oder Audiodateien relevant)

- `POST /interaction/`SZENEN-ID`/`AKTIONSINDEX`/`ANTWORTINDEX`?visitortype=`BESUCHERTYP
  Chat-Interaktion an Server übermitteln (Besuchertyp ist optional, Ziel wird als weiterer Parameter übergeben, siehe unten)
  als JSON übergebene Daten:
  ```
  {
    "context": KONTEXTOBJEKT
  }
  ```
  Das Kontextobjekt, die Szenen-ID und der Aktionsindex wurden zuvor (bei der Übermittlung der Chatnachricht) an den Client übergeben, und werden hier 1:1 zurückgesendet. Der Antwortindex entspricht der User-Auswahl (0 für die erste Option, 1 für die zweite usw.)

- `POST /menuaction/`MENÜ-ID`/`INDEX`?visitortype=`BESUCHERTYP
  Menü-Interaktion an den Server übermitteln (Besuchertyp ist optional, Ziel wird als weiterer Parameter übergeben, siehe unten)
  als JSON übergebene Daten:
  ```
  {
    "context": KONTEXTOBJEKT
  }
  ```
  Das Kontextobjekt und die Menü-ID wurden zuvor (bei der Übermittlung der Menüs) an den Client übergeben, und werden hier 1:1 zurückgesendet. Der Index entspricht der User-Auswahl (0 für das erste Menüelement, 1 für das zweite usw., bei Untermenüs werden die einzelnen Indizes durch Semikolon getrennt, z.B. `1;0;3`)

- `POST /changemenu/`SLOT
  neues per JSON übergebenes Menü im gewählten Slot anzeigen (Ziel wird als Parameter übergeben, siehe unten)

- `GET /showmenu/`SLOT`?visible=`SICHTBARKEIT AN/AUS
  aktuelles Menü im gewählten Slot anzeigen oder verbergen (Ziel wird als weiterer Parameter übergeben, siehe unten)

- `POST /shortcutaction/`BOXNAME`/`INDEX
  für den ausgewählten Client hinterlegtes Tastenkürzel auslösen
  als JSON übergebene Daten:
  ```
  {
    "context": KONTEXTOBJEKT
  }
  ```

Das gewünschte Ziel wird bei etlichen Kommandos als zusätzlicher Parameter in der URL übergeben. Dabei sind unterschiedliche Varianten möglich (es darf nur einer der folgenden Parameter angegeben werden):

1. via Session-ID der Socket-Verbindung: `sid=`SESSION-ID
2. via Kanal-ID: `channelid=`CHANNEL-ID
3. via Name des Clients: `boxname=`NAME
4. via Name des Kanals: `channelname=`NAME
5. via Name des Clients oder des Kanals: `boxorchannel=`NAME

6.2. Websocket-API serverseitig

Die Websocket-API basiert auf socket.io, folgende Anfragen nimmt der Server (vom Client) entgegen:

- `pingserver`
  Server pingen, Server reagiert mit `pongserver` (siehe Client-API) und reicht die erhaltenen Daten (insbesondere Zeitstempel) zurück

- `pongbox`
  Antwort von `pingbox` (siehe Client-API) entgegennehmen und Roundtrip-Zeit im Log-File festhalten

- `alive`
  Alive-Signal von Client entgegennehmen (dient zur Erkennung verlorener Verbindungen)
  als JSON übergebene Daten:
  ```
  {
    "active": AKTIV JA/NEIN,
  }
  ```

- `joinbox`
  Client in die Liste bestehender Verbindungen aufnehmen und definierte Kanäle zuweisen (die als „Rooms“ mit socket.io implementiert sind)
  als JSON übergebene Daten:
  ```
  {
    "name": CLIENT-NAME
  }
  ```

6.3. Websocket-API clientseitig

Die Websocket-API basiert auf socket.io, folgende Anfragen nimmt der Client (vom Server) entgegen:

- `pingbox`
  Client pingen, Client reagiert mit `pongbox` (siehe Server-API) und reicht die erhaltenen Daten (insbesondere Zeitstempel) sowie den eigenen Namen zurück

- `reload`
  Applikation neu laden

- `reset`
  Socket-Verbindung neu aufbauen und Inhalte neu laden

- `halt`
  Applikation einfrieren (schwarzer Bildschirm wird angezeigt)

- `playmedia`
  Medien-Datei (Bild, Video etc.) darstellen
  als JSON übergebene Daten:
  ```
  {
    "url": MEDIEN-URL
  }
  ```

- `playobject`
  übergebenes Objekt (Medien-Objekt oder anderer Inhalt) darstellen
  als JSON übergebene Daten:
  ```
  {
    "object": OBJEKT,
    "loop": IN SCHLEIFE ABSPIELEN JA/NEIN,
    "backdrop": AUDIO IM HINTERGRUND ABSPIELEN JA/NEIN,
    "referenced": LISTE WEITERER BENÖTIGTER OBJEKTE,
    "subjects": LISTE VON BEZUGSOBJEKTEN
  }
  ```

- `setcontext`
  Kontext ändern / hinzufügen
  als JSON übergebene Daten:
  ```
  {
    "subjects": LISTE NEUER BEZUGSOBJEKTE,
    "referenced": LISTE WEITERER BENÖTIGTER OBJEKTE,
    "operation": OPERATION (APPEND/INSERT/PUT)
  }
  ```

- `chatmessage`
  Chat-Nachricht anzeigen (Message-Objekt als JSON übergeben)

- `chatinteraction`
  Chat-Interaktion (Frage und Antwortmöglichkeiten) anzeigen (Interaction-Objekt als JSON übergeben)

- `showchat`
  Chatansicht anzeigen oder verbergen
  als JSON übergebene Daten:
  ```
  {
    "visible": SICHTBAR JA/NEIN
  }
  ```

- `shownotification`
  Benachrichtigung anzeigen
  als JSON übergebene Daten:
  ```
  {
    "text": TEXT
  }
  ```

- `pause`
  Video/Audio-Wiedergabe unterbrechen

- `resume`
  Video/Audio-Wiedergabe fortsetzen

- `stop`
  Video/Audio-Wiedergabe anhalten

- `setvolume`
  Wiedergabe-Lautstärke ändern
  ```
  {
    "volume": LAUTSTÄRKE IN PROZENT
  }
  ```

- `changemenu`
  Menü austauschen
  ```
  {
    "slot": MENÜ-SLOT (A/B),
    "menu": MENÜ-OBJEKT,
    "visible": SICHTBARKEIT AN/AUS,
    "context": KONTEXT-OBJEKT,
    "referencedObjects": LISTE WEITERER BENÖTIGTER OBJEKTE
  }
  ```

- `showmenu`
  aktuelles Menü anzeigen oder verbergen
  ```
  {
    "slot": MENÜ-SLOT (A/B),
    "visible": SICHTBARKEIT AN/AUS
  }
  ```

- `showscreensaver`
  Bildschirmschoner starten

- `resetidletimer`
  Idle-Timer zurücksetzen (und damit automatischen Start des Bildschirmschoners verzögern)