**Doku der Express API für das Projekt IoT Smartcam**

Der **User** ist eine berechtigte Person, die das Haus betreten darf. In der Weboberfläche kann sich lediglich der **Admin** einloggen. Sein Passwort kann bei Bedarf geändert werden. Zusätzliche „Admins“ können nicht angelegt werden.

Das Format aller Anfragen und Antworten ist JSON.

1. **Login des Admins**
   1. Kurzbeschreibung: Versucht der Admin sich einzuloggen, wird der Name des Admins und das gehashte Passwort an den Server gesendet. Dieser prüft den Namen und das Passwort des Admins. War der Login erfolgreich, wird im Backend eine neue Session für den Admin erstellt. Der Client erhält dann eine Antwort, mit dem Namen des eingeloggten Admins und dem loginStatus true.
   2. HTTP-Verb: POST
   3. Route: /v1/login
   4. Request: admin : (String), password : (gehashter String)
   5. Response: admin : (String), loginStatus : (boolean)
2. **Logout des Admins**
   1. Kurzbeschreibung: Erfolgt der Logout des Admins, wird die Session zerstört und der Admin wird ausgeloggt. Bei erfolgreichem Logout bekommt der Client eine Flag mit true zugesendet. Schlägt der Logout fehl, wird die Logout Flag auf true gesetzt.
   2. HTTP-Verb: POST
   3. Route: /v1/logout
   4. Request: admin : (String), loginStatus : false
   5. Response: admin : (String), logout : (boolean)
3. **Ändere das Passwort des Admins**
   1. Kurzbeschreibung: Hierüber kann das aktuelle Passwort des Admins geändert werden. Wurde das Passwort erfolgreich geändert, wird die Flag passwordChanged auf true gesetzt. Schlägt der versuch fehl, wird die Flag auf false gesetzt.
   2. HTTP-Verb: POST
   3. Route: /v1/changePassword
   4. Request: admin : (String), oldPassword : (gehashter String), newPassword : (gehashter String)
   5. Response: admin : (String), passwordChanged : (boolean)
4. **User Liste anfordern**
   1. Kurzbeschreibung: Liefert die aktuelle Userliste zurück.
   2. HTTP-Verb: GET
   3. Route: /v1/userList
   4. Request: -
   5. Response: [ { DATA USER1 }, { DATA USERn } ]
5. **User anhand einer Id löschen**
   1. Kurzbeschreibung: Liest die UserId aus der Route aus und löscht den User aus der MYSQL-Datenbank. Bei Erfolg wird der gelöschte User zurückgegeben. Ansonsten wird ein leeres Objekt zurückgegeben.

**Wichtig:** Sollen sowohl der User in der mySQL Datenbank, als auch die Bilder in der KAIROS Galerie gelöscht werden, müssen zuerst alle Bilder in KAIROS gelöscht werden. Ansonsten können die Bilder in KAIROS keinem User zugeordnet werden.

* 1. HTTP-Verb: DELETE
  2. Route: /v1/user/userId
  3. Request: -
  4. Response: userId : (Number), firstName : (String), lastName : (String), image: (base 64 encode), accessFrom : (Date), accessTo : (Date), qrCode : (blob)

1. **Lösche die Bilder eines Users in der KAIROS Galerie**
   1. Kurzbeschreibung: Löscht alle Bilder eines Users aus der KAIROS Galerie. Bei Erfolg wird die Flag auf true gesetzt. Ansonsten auf false.
   2. HTTP-Verb: DELETE
   3. Route: /v1/image/userId
   4. Request: -
   5. Response: imageDeleted : (boolean)
2. **Füge einen neuen User hinzu und lade ein Bild des Users (optional) in der KAIROS Bibliothek hoch**
   1. Kurzbeschreibung: Es werden verschiedene Eigenschaften eines Users hochgeladen. Im Backend werden diese Daten mit einer UserId und einem qrCode angereichert. Bei Erfolg wird der angelegte User zurückgegeben. Ansonsten wird ein leeres Objekt zurückgegeben.
   2. HTTP-Verb: POST
   3. Route: /v1/addUser
   4. Request: firstName : (String), lastName : (String), image: (base 64 encode -> falls kein Bild hochgeladen wird auf null setzten), accessFrom : (Date), accessTo : (Date)
   5. Response: userId : (Number), firstName : (String), lastName : (String), image: (base 64 encode), accessFrom : (Date), accessTo : (Date), qrCode : (blob)
3. **Lade ein zusätzliches Bild zu einem User hoch**
   1. Kurzbeschreibung: Ein 64 base codiertes Bild und die UserId wird hochgeladen. Bei Erfolg, wird die flag auf true gesetzt. Ansonsten wird false zurückgegeben.
   2. HTTP-Verb: POST
   3. Route: /v1/addImage
   4. Request: userId: (Number), image: (base 64 encode)
   5. Response: imageCreated : (boolean)
4. **Erhalte Daten zu einem bestimmten User**
   1. Kurzbeschreibung: Die UserId wird über die als URL-Parameter übergeben. Wird der User gefunden, wird dieser zurückgeliefert. Ansonsten erhält der Client ein leeres Objekt.
   2. HTTP-Verb: GET
   3. Route: /v1/user/userId
   4. Request: -
   5. Response: userId : (Number), firstName : (String), lastName : (String), image: (base 64 encode), accessFrom : (Date), accessTo : (Date), qrCode : (blob)
5. **Bearbeite die Daten eines bestimmten Users**
   1. Kurzbeschreibung: Die UserId wird als URL-Parameter übergeben. Außerdem wird ein JSON Objekt mit zu überschreibenden Daten an den Server übermittelt. Schlussendlich schickt der Server den überarbeiteten User an den Client.
   2. HTTP-Verb: PUT
   3. Route: /v1/editUser/userId
   4. Request: Zu ändernde User Daten in Form eines JSON Objekts
   5. Response: userId : (Number), firstName : (String), lastName : (String), image: (base 64 encode), accessFrom : (Date), accessTo : (Date), qrCode : (blob)
6. **Starte den PI neu**
   1. Kurzbeschreibung: Wird die restartPi flag mit true an den Server versendet, wird der PI neu gestartet. War der Neustart erfolgreich, wird eine Flag mit true zurückgeliefert.
   2. HTTP-Verb: POST
   3. Route: /v1/restartPI
   4. Request: restartPi : true
   5. Response: piRestartSuccessful : (boolean)
7. **Erhalte eine Liste mit allen Usern, die das Haus in einem definierten Zeitraum betreten haben** 
   1. Kurzbeschreibung: Der Client übermittelt in der URL einen Zeitraum (from, to). Der Server liefert als Antwort eine Liste an Usern, die in dieser Zeitspanne das Haus betreten haben.
   2. HTTP-Verb: GET
   3. Route: /v1/getAccessList/from/to
   4. Request: -
   5. Response: [ { DATA USER1 }, { DATA USERn } ]