ASKDj Server commands

user string: persistent third party (e.g. Google account hash)
session ID: cleared at logout (e.g. session cookie)

Data types are indicated in <>. Optional fields are enclosed in []. All undefined key-value pairs are probably ignored by the server, but don't use them!

You must store all cookies the server sends. We assume the user can handle cookies.

- create room (/room/create)
    - POST parameters:
        - "name": <string (max 512 chars)>
        - ["location": <string (max 512)>]
        - ["latitude": <float>
        - "longitude": <float>] (need both or none of lat and long to validate)
        - ["description": <string>]
        - ["password": <string (max 512)>]
    - return:
        - json: {"roomId": <string>}
        - create cookie information
        - 400 error if something is wrong
- get rooms (/room/list)
    - GET parameters:
        - ["q" (search query): <string>]
        - [(search center)
        - "latitude": <float>,
        - "longitude": <float> (all or nothing required)]
        - ["maxResults": <integer>]; ignores if not positive
    - return:
        - JSON
        - {rooms: [{
            - "roomId": <string>
            - "name": <string>
            - "size": <integer>
            - “hasPassword”: <boolean>
            - ["location": <string>]
            - ["coords": {"latitude": <float>, "longitude": <float>}]
            - ["description": <string>]
            - } (repeat for total number of rooms)
        -]}
- join room (/room/join)

    - POST parameters:
        - "roomId": <string>
        - "password": <string>
    - return:
        - blank response? or {}?
        - create cookie information
        - 401 error if room has password and wrong password given
        - 400 error if something else is wrong
- room info (/room/info)
    - GET/POST: no parameters, requires user to be in a room
    - return JSON: {
        - "roomId": <string>
        - "name": <string>
        - "size": <integer>
        - “hasPassword”: <boolean>
        - ["location": <string>]
        - ["coords": {"latitude": <float>, "longitude": <float>}]
        - ["description": <string>]}
- leave room (/room/leave)
    - **note:** if user is last DJ in the room, also deletes the room
        - deletes user songs from pool
    - POST (DELETE?) parameters:
        - "roomId": <string>
    - return:
        - blank response? or {}?
        - wipe cookie information
        - 400 error if something is wrong
- ~~DJ-remove from room? [session ID of DJ, session ID to kick, room ID]~~

- add songs to pool [/pool/add]
    - POST parameters:
      ** Dictionary of “songs” with a value of a list of dictionaries with the following..
        - "title": <string (max 1024 chars)>
        - ["artist": <string (max 1024)>]
        - [“localUrl”: <string>]
            - (a local device-dependent identifier, which will uniquely identify the song)
        - ["spotifyUrl": <string>]
        - ["youtubeUrl": <string>]
    - return:
        - empty respose
            - {“Warning”: <string>} if non-fatal error
        - cookie with roomId

        - 400 error if something is wrong
- set songs in pool [/pool/set]
    - same specs as /pool/add, but clears all songs from user first
- ~~DJ-delete songs from pool [session ID, room ID, list<songs>]~~
- get pool (/pool/list)
    - cookie required to get user and room
    - GET parameters:
        - [“q”: <string>]
            - filters to only return songs with title or artist matching
    - return:
        - json: {“songs”: list of
            - {“songKey”: <integer> (server identifier for a song)
            - “title”: <string> (human-readable song title)
            - [“artist”: <string> (if exists)]
            - “voteTotal”: <integer> (number of)
            - “userHasVote”: <boolean>}

- upvote song request (/votes/add)
    - make new song request if not exists, else request count++
    - GET/POST parameters:
        - “songKey”: <string>
    - return {}
- revoke vote request (/votes/revoke)
    - cancels the upvote request for the given son
    - GET/POST parameters:
        - “songKey”: <string>
    - return {}
- get votes list (/votes/list)
    - cookie required to get user and room
    - no parameters, all returned voteTotal values are > 0
    - return:
        - json: {“songs”: list of
            - {“songKey”: <integer> (server identifier for a song)
            - “title”: <string> (human-readable song title)
            - [“artist”: <string> (if exists)]
            - “voteTotal”: <integer> (number of)
            - “userHasVote”: <boolean>}

- DJ-move song to queue (up next) [/queue/add]
    - GET/POST parameters:
        - “songKey”: <string>

    - return: {} or error
        - **may give a 423 error if too many simultaneous requests**
- DJ-revise song queue (up next) [/queue/addmany]
    - POST parameters: (json)
        - [“doClear”: <boolean> (default false)]
            - if true, makes the current queue the given list
            - if false, appends the given list to the current queue
        - “songs”: (list of) [
            - {“songKey”: <integer>}
    - return: {} or error
        - for all songs that have more entries in the queue than before, remove all votes for that song
        - **may give a 423 error if too many simultaneous requests**
- get up next list (/queue/list)
    - cookie required, no parameters
    - return:
        - json: {“queue”: ordered list of
            - {“songKey”: <integer> (server identifier for a song)
            - “title”: <string> (human-readable song title)
            - [“artist”: <string> (if exists)]
            - [“owner”: <string>]
            - [“localUrl”: <string>]
            - [“spotifyUrl”: <string>]}
- remove next song in queue (/queue/pop)
    - cookie required, no parameters
    - return:
        - json: (one entry)
            - {“songKey”: <integer>
            - “title”: <string>
            - [“artist”: <string> (if exists)]
            - [“owner”: <string>]
            - [“localUrl”: <string>]
            - [“spotifyUrl”: <string>]}
- set now playing list (/nowplaying/set)
    - GET/POST parameters:
        - “songKey”: <string>
    - return: {} or error
- get now playing list (/nowplaying/get)
    - cookie required, no parameters
    - return:
        - json: (one entry)
            - {“songKey”: <integer>
            - “title”: <string>

- [“artist”: <string> (if exists)]
- [“owner”: <string>]
- [“localUrl”: <string>]
- [“spotifyUrl”: <string>]}

song format: polymorphic (one of below)

- device song UUID (DJ’s song)
- user session ID, device song UUID
- YouTube URL
- Spotify URL
- etc.