Housing Posting Web Application

Group:
Kyaw Soe
Ally Quan

Overview

The Housing REST Service (HHS) provides the interface needed to interact with a site that tracks users, house listing, and images posted to those listings. The site tracks many different listings, and any registered user may make a listing, see other user's listings, filter all the user listings, etc.

General Points

The following design points apply across the document.

1. All resource URLs are prefixed by some root URL, (e.g. http://www.softwareinventions.com/CHS/)
2. All resources accept and provide only JSON body content. And per REST standards, all successful (200 code) DELETE actions return empty body.
3. Some GET operations allow get-parameters. These are listed directly after the GET word. All get-parameters are optional unless given in bold.
4. Absent documentation to the contrary, all DELETE calls, POST, and PUT calls with a non-200 HTTP response return as their body content, a list of JSON objects describing any errors that occured. Error objects are of form {tag: {errorTag}, params: {params}} where errorTag is a string tag identifying the error, and params is an array of additional values needed to fill in details about the error, or is null if no values are needed. E.g. {tag: "missingField", params: ["lastName"]}
5. Resource documentation lists possible errors only when the error is not obvious from this General Points section. Relevant errors may appear in any order in the body. Missing field errors are checked first, and no further errors are reported if missing fields are found.
6. All resource-creating POST calls return the newly created resource as a URI via the Location response header, not in the response body. The response body for such POSTs is reserved for error information, per point 4.
7. GET calls return one of the following. Response body is empty in the latter three cases. Get calls whose specified information is a list always return an array, even if it has just one or even zero elements.
    1. HTTP code OK and the specified information in the body.
    2. BAD_REQUEST and a list of error strings.
    3. UNAUTHORIZED for missing login.
    4. FORBIDDEN for insufficient authorization despite login

    5. NOT_FOUND for a URI that is not described in the REST spec if logged in, 401 if not.

8. Fields of JSON content for POST and PUT calls are assumed to be strings, booleans, ints, or doubles without further documentation where obvious by their name or intent. In non-obvious cases, the docs give the type explicitly.
9. All access requires authentication via login to establish the Authenticated User (AU); no resources are public except for Prss/POST (for initial registration), and Ssns/POST (to log in). Other resources may be restricted based on admin status of AU. The default restriction is to allow only access relevant to the AU, unless the AU is admin, in which case access to any Person's info is allowed.
10. Any database query failure constitutes a server error (status 500) with a body giving the error object returned from the query. Ideally, no request, however badly framed, should result in such an error except as described in point 11
11. The REST interface does no general checking for forbiddenField errors, unless the spec specifically indicates it will. Absent such checking, non-specified body fields in PUT/POST calls may result in database query errors and a 500 code, as may an empty body when body content is expected.
12. Required fields may not be passed as null, undefined or "". Doing so has the same outcome as if the field were entirely missing
13. All times are integer values, in mS since epoch.
14. Non JSON parseable bodies result in 500 error.

Error Codes

The possible error codes, and any parameters, are as follows.
missingField Field missing from request. Params[0] gives field name
badValue Field has bad value. Params[0] gives field name
notFound Entity not present in DB -- for cases where a Conversation, Person, etc. is not there.
badLogin Email/password combination invalid, for errors logging.
dupEmail Email duplicates an existing email
noTerms Acceptance of terms is required
noOldPwd Change of password requires an old password
oldPwdMismatch Old password that was provided is incorrect.
dupListingTitle Conversation title duplicates an existing one
forbiddenField Field in body not allowed. Params[0] gives field name.
queryFailed Query failed (server problem)

Resources for User Management, including Registration

(Admin use in purple)

Prss

Collection of all current persons or other users.

GET email={email or email prefix}

Returns list of zero or more Persons. Limits response to Persons with specified email or email prefix, if applicable. No data for other than the AU is returned in any event, unless the AU is an admin. This may result in an empty list if e.g. a non-admin asks for an email not their own. Data per person:

- *email* principal string identifier, unique across all Persons -
- *id* id of person with said email, so that URI would be Prss/{id}

POST

Adds a new Person. No AU required, as this resource/verb is used for registration, but an AU is allowed, and an admin AU gets special treatment as indicated.

- *email* unique Email for new person *firstName*
- *lastName*
- *password*
- *role* 0 for student, 1 for admin
- *termsAccepted* boolean--were site terms and conditions accepted?

Email, role and lastName required and must be nonempty. Error if email is nonunique. Error if terms were not accepted and AU is not admin. Error forbiddenRole if role is not student unless AU is admin. Nonempty password required unless AU is admin, in which case if no password is provided a blocking password of * is recorded, preventing further access to the account (once encryption is enforced).

Prss/{prsId}

GET

Returns array with one element for Person {prsId}, with fields as specified in POST for Prss, plus dates *termsAccepted* and *whenRegistered*, less *password*. (*termsAccepted* may be falsey if terms were not accepted.) The dates give time of term acceptance and registration, and will generally be equal, but are listed separately for legal reasons. AU must be person {prsId} or admin.

PUT

Update Person {prsId}, with body giving an object with zero or more of *firstName*, *lastName*, *password*, *role.* Attempt to change other fields in Person such as *termsAccepted* or *whenRegistered* results in BAD_REQUEST and forbiddenField error(s). Role changes result in BAD_REQUEST with badValue tag for nonadmins. All changes require the AU be the Person in question, or an admin. Unless AU is admin, an additional field *oldPassword* is required for changing *password,* with error oldPwdMismatch resulting if this is incorrect. Password, if supplied, must be nonempty and nonnull or badValue error results, even if AU is admin.

DELETE

Delete the Person in question, including all Cnvs and Msgs owned by Person. Requires admin AU.

Ssns

Login sessions (Ssns) establish an AU and are required for most service access. A user obtains one via POST to Ssns.

GET

Returns a list of all active sessions. Admin privileged AU required. Returns array of

- *cookie* Unique cookie value for session
- *prsId* ID of Person logged in
- *loginTime* Date and time of login

POST

A successful POST generates a browser session cookie that will permit continued access for 2 hours. Indicated Person becomes the AU. An unsuccessful POST results in a 400 with a badLogin tag and no further information.

- *email* Email of user requesting login
- *password* Password of user

Ssns/{cookie}

GET

Returns, for the indicated session, a single object with same properties as one element of the array returned from Ssns GET. AU must be admin or owner of session.

DELETE

Log out the specified Session. AU must be owner of Session or admin.

Listing

POST

Make a new listing for the house. The *price, title, description,* and *contactInfo* are required for all AU to create a listing. *contactInfo* is a string - either email or phone number. *contactInfo* email doesn't have to be the same as the owner email. Listing *title* cannot be a duplicate (Error dupTitle).

- *title*: title of the posting
- *description*: the detail description of their housing
- *price*: the rent per month for the listed housing
- *numBed*: the number of bed room
- *location*: the house location which could be area or apartment name
- *contactInfo*: could be email or phone number

GET owner={ownerId} numbed={num} price={1/-1}

Any AU is acceptable, though some login is required. Return an array of 0 or more elements in decreasing datetime order, with one element for each Listing in the system, limited to Listing with the specified owner if query param is given. Limit to the specific numBeds when numbed is provided. Sort by price ascending/descending if price is provided:1 is ascending, and -1 is descending:

- *id* Id of the Listing
- *title* Title of the Listing

- *ownerId* Listing's owner
- *price*: the rent per month for the listed housing
- *numBed* number of bedrooms in the house
- *Location* location of the house
- *postedDate* when the listing was posted

Listing/{listingId}

PUT

Update the listing for a house with the *listingId*. Users are allowed to edit any of the fields appear in the body of the listing except images. However, the title still need to be unique.

GET

Get all listings of a user with the same fields as in the post with addition of the *postedDate* and *listingId*. It will return as an array of JSON object. If the user is not admin and ownerId doesn't match with current login *prssId*, it will return an empty array.

DELETE

Delete the listing with given listingId. Return 404 with "not found" message if listing is not found. AU must be Listing owner or admin.

/Listing/:listingId/Images

POST

Upload an image for the properties listing. AU must be Listing owner or admin.

- *imageUrl* path to image (local or internet)

GET

Any AU is acceptable, though some login is required. Return all Images for the indicated Listing.

- *id* Image ID
- *imageUrl* path to the Image

/Listing/:listingId/Images/{imageId}

DELETE

Delete an image with the given id.

Special DB Resource for Testing Purposes

DB

DELETE

Clear all content from the database, reset all auto increment IDs to 1, and add back one Person, an admin named Joe Admin with email adm@11.com and password "password". Clear all current sessions. AU must be an admin.