Radical Restaurant Reviews REST Service

Adapted from Chat REST Service by Clint Staley

Overview

The Radical Restaurant Reviews REST Service (R^3RS) provides the interface needed to interact with a site that tracks users, restaurants, and reviews posted to those restaurants. The site tracks many different restaurant, and any user may make a post to any restaurant, see other user's reviews, 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.example.com/RRR/)
2. All resources accept and provide only JSON body content. And per REST standards, all successful (200 code) DELETE actions return an 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.
    a. HTTP code 200 **OK** and the specified information in the body.
    b. **BAD_REQUEST** and a list of error strings.
    c. **UNAUTHORIZED** for missing login.
    d. **FORBIDDEN** for insufficient authorization despite login
    e. **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. **Not all** access does require authentication via login to establish the Authenticated User (AU); all resources are **public** *except* for the list of private resources below (9a and on). Some of these non-public resources may be further restricted based on admin status of AU. The default

restriction is to allow access to all public resources to anyone, plus to allow access relevant to the AU, unless the AU is admin, in which case access to any Person's info is allowed.

a. GET Ssns [admin only]
b. POST Rsts
c. PUT Rsts/{id}
d. DELETE Rsts/{id}
e. POST Rsts/{id}/Revs
f. POST Revs/{id}
g. POST Vots/{rstId}/{revId}
h. DELETE DB [admin only]

10. Any database query failure constitutes a server error (HTTP status 500 **INTERNAL_SERVER_ERROR**) 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 an HTTP code 500 **INTERNAL_SERVER_ERROR**, 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 an HTTP code 500 **INTERNAL_SERVER_ERROR**.

Error Codes

The possible error codes, and any parameters, are as follows. An asterisk (*) before the description indicates that params[0] gives the field name for the offending value.

missingField	*	Field missing from request
badValue	*	Field has bad value
notFound		Entity not present in DB – for cases where a Restaurant, Person, etc. is not there
badLogin		Email/password combination is invalid, for error 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
dupTitle		Restaurant title duplicates an existing one
forbiddenField	*	Field in body not allowed
queryFailed		Query failed (server problem)

Resources for User Management, including Registration

Unless otherwise specified (for example, by one of the below color codes), all resources are **public** (are visible to all AUs *and* are also visible without logging in).

(Admin use in purple)

Indicates that the resource is only available to Admin AUs

(Non-public in blue)

Indicates that the resource requires an AU.

Prss

Collection of all current 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 standard user, 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 inBAD_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 Rsts, Revs, and Vots owned by Person. Requires admin AU.

Ssns

Login sessions (Ssns) establish an AU. 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.

Resources for Restaurants

The following resources allow creation, deletion, and management of Restaurants -- each a series of Reviews. Any user may GET information on any Restaurant or Review. Any AU may POST Reviews to any Restaurant and may POST an entirely new restaurant, thus becoming its owner. The owner of the Restaurant and Admin AUs may post comments on a Restaurant's reviews.

Rsts

GET owner=<ownerId>

No login required. Return an array of 0 or more elements, with one element for each Restaurant in the system, limited to Restaurants with the specified owner if query param is given:

id Id of the Restaurant
title Title of the Restaurant
ownerId Owner of the Restaurant
url URL for the Restaurant.
category Category of the Restaurant.

POST

Any AU is acceptable, though some login is required. Create a new Restaurant, owned by the current AU. Error dupTitle if title is a duplicate. Fields are:

title Title of the new Restaurant, limited to 80 chars, required
description Description of the new Restaurant, limited to 300 chars
url URL for the new Restaurant, limited to 80 chars, required
category Category of the new restaurant. Limited to one of the following:

- Bakery
- Barbeque
- Chinese
- Deli
- Fine Dining
- Ice Cream
- Seafood
- Vegetarian
- Breakfast
- Burgers
- Coffee
- Italian
- Sandwiches
- Pizza

Rsts/{rstId}

GET

No login required. Return single object having same properties as one of the array elements returned by Rsts GET, for just the indicated Rst.

PUT

Update the title of the Restaurant. Fields as for Restaurants POST, including required title. Error dupTitle if title is duplicate. AU must be Restaurant owner or admin.

DELETE

Delete the Restaurant, including all associated Reviews and Votes. AU must be Restaurant owner or admin.

Rsts/{rstId}/Revs

GET dateTime ={dateTime} num={num}

No login required. Return all Reviews for the indicated Restaurant. Limit this to at most num Reviews (if num is provided) posted on or before dateTime (if dateTime is provided). Return for each Review, in increasing datetime order, and for same datetimes, in increasing ID order:

id Review ID
whenMade when the Review was made
email Email of the poster
content Content of the Review
rating Rating value as an integer out of 5.
numUpvotes Number of upvotes. Could be negative.
ownerResponse Content of the owner's response, as follows (could be null):
whenMade When the response was made
content Content of the response

POST

Any AU is acceptable, though some login is required. Add a new Review, stamped with the current AU and date/time. Number of upvotes will be set to zero.

content Content of the Review, up to 5000 chars
rating Restaurant rating as an integer value out of 5.

Resources for Reviews

Revs/{revId}

GET

No login required. Return the following for the indicated review.

rstId the ID of the Restaurant for which the review was made
whenMade when the Review was made
email Email of the poster
content Content of the Review
rating Rating value as an integer out of 5.
numUpvotes Number of upvotes. Could be negative.
ownerResponse Content of the owner's response, as follows (could be null):
whenMade When the response was made
content Content of the response

POST

AU must be restaurant owner or an admin. Add an owner response to a review. Response is stamped with the current AU and date/time.

ownerResponse Content of the owner's response, up to 5000 chars.

Resources for Review Votes

Vots/{rstId}/{revId}

GET

Any AU is acceptable, though some login is required. Gets the user's current vote on a review. If the user has not voted on a review yet, returns error *notFound*.

voteValue Value of the user's vote. Will be either -1 or 1, indicating an upvote or downvote (respectively).

POST

Any AU is acceptable, though some login is required. Add a user vote to a review. Vote is stamped with the current AU and date/time.

voteValue Value of user's vote. Must be 1 or -1, indicating an upvote or downvote (respectively).

DELETE

Any AU is acceptable, though some login is required. Delete a user's vote on a review. If a user does not have a vote on the review yet, the DELETE call returns error *notFound*.

Special DB Resource for Testing Purposes

DB

DELETE

Clear all content from the database, reset all autoincrement 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.