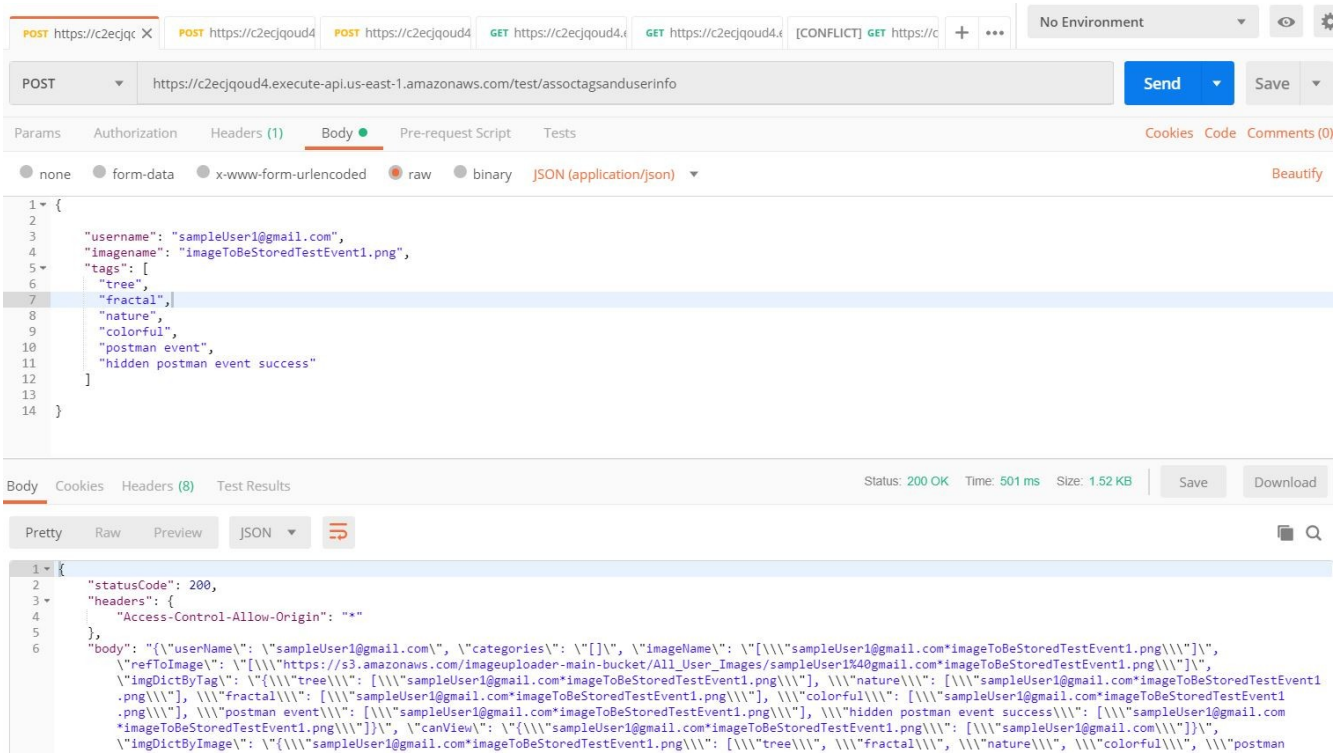
Updated 10/6/2019

Note: Screenshots do not necessarily represent consecutive order of events!

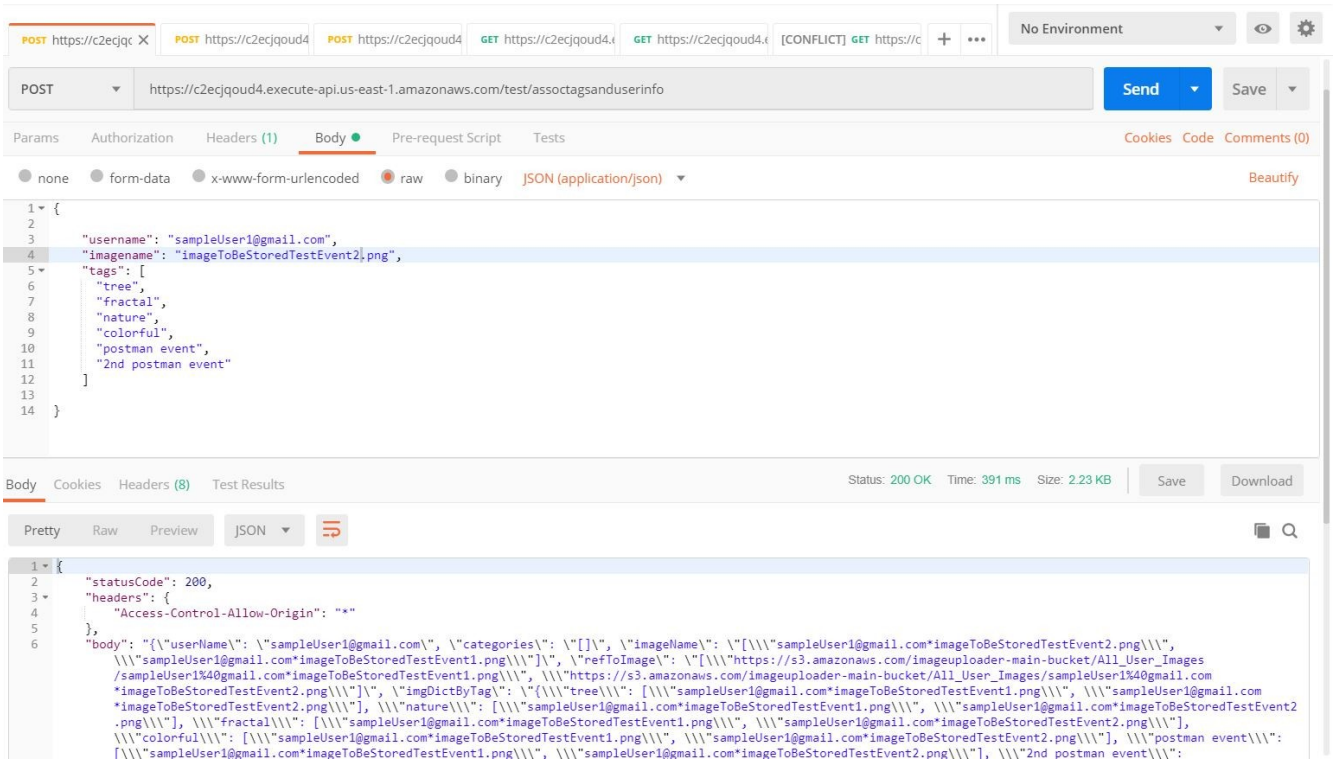
POST:
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/assoctagsanduserinfo

Description: Use this to associate a list of tags for a particular image of a user. This is basically adding an image, its tags for a user, and adds the user itself as the true owner of the image. It also updates the tags for an image, so whatever list of tags you pass for an image, all those tags will become the new tags for that image.

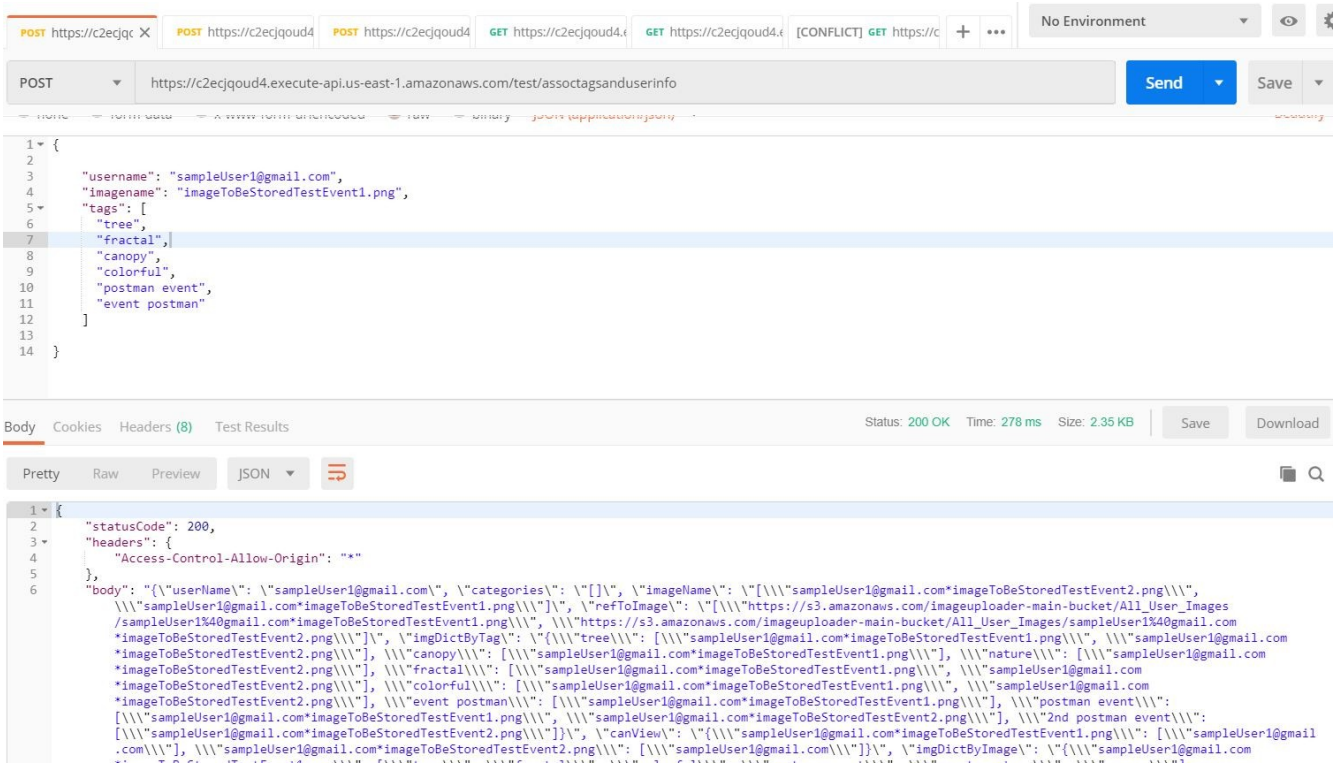
Screenshot of when user isn't yet in database (brand new user/owner):



Screenshot of when you want to add a new image and tags:



Screenshot of updating tags for a current user's image (includes adding and deleting tags):



Screenshot of Headers (for all api's so far):

Body Cookies Headers (8) Test Results

Date → Thu, 21 Mar 2019 05:39:31 GMT

Content-Type → application/json

Content-Length → 2081

Connection → keep-alive

x-amzn-RequestId → b356b928-4b9b-11e9-8c7d-fbd4794c3f4e

Access-Control-Allow-Origin → *

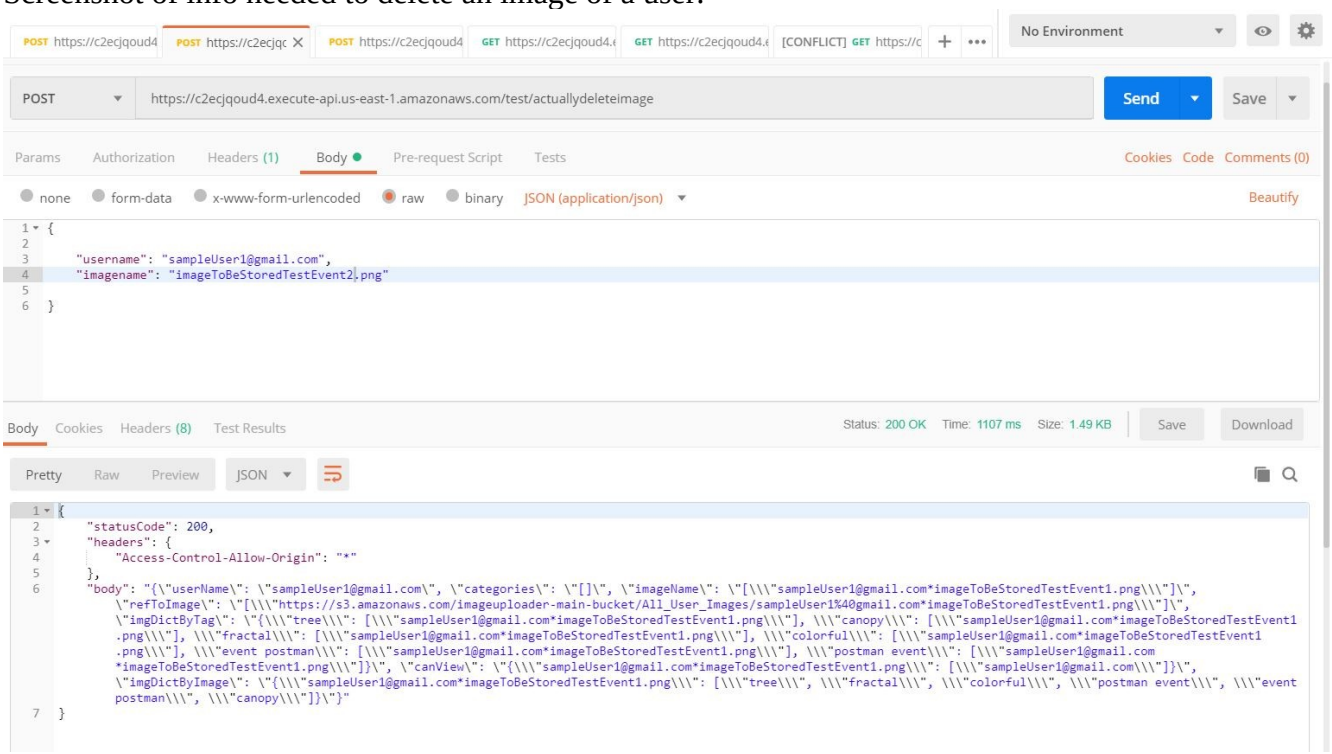
x-amz-apigw-id → W4J_AEtjIAMFxqg=

X-Amzn-Trace-Id → Root=1-5c932393-77311883bf8cf7f9e70b6f9a;Sampled=0

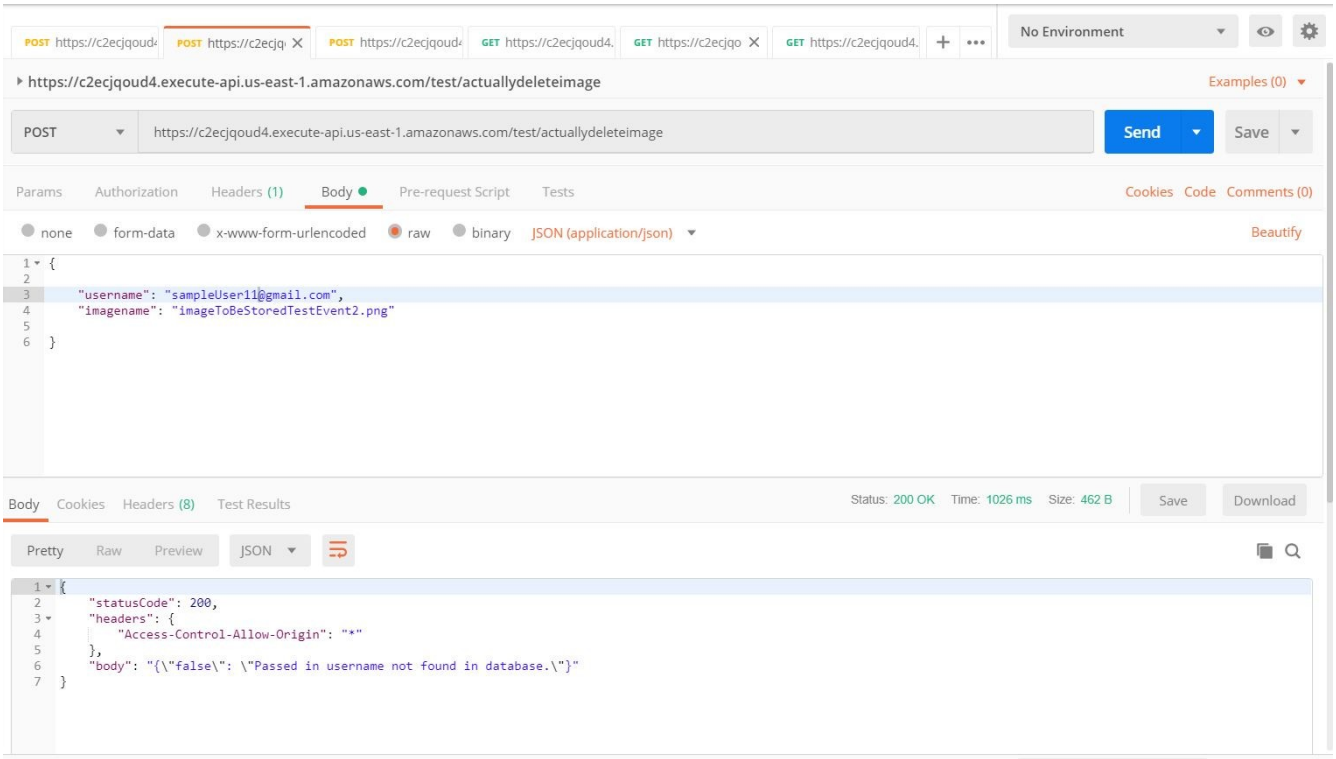
POST:
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/actuallydeleteimage

Description: Deletes an image related to target user/owner.

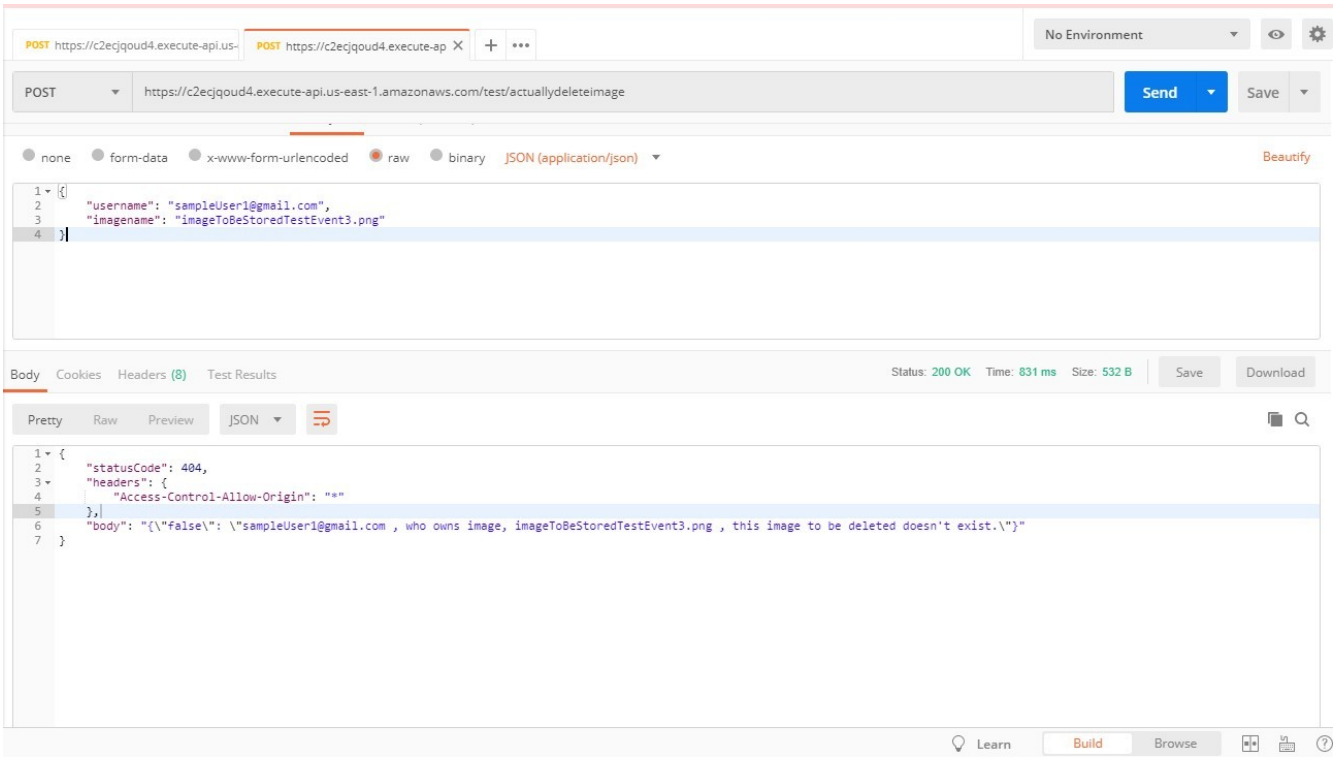
Screenshot of info needed to delete an image of a user:



Screenshot of when given username is not found:



Screenshot of when image of an existing user cannot be found:

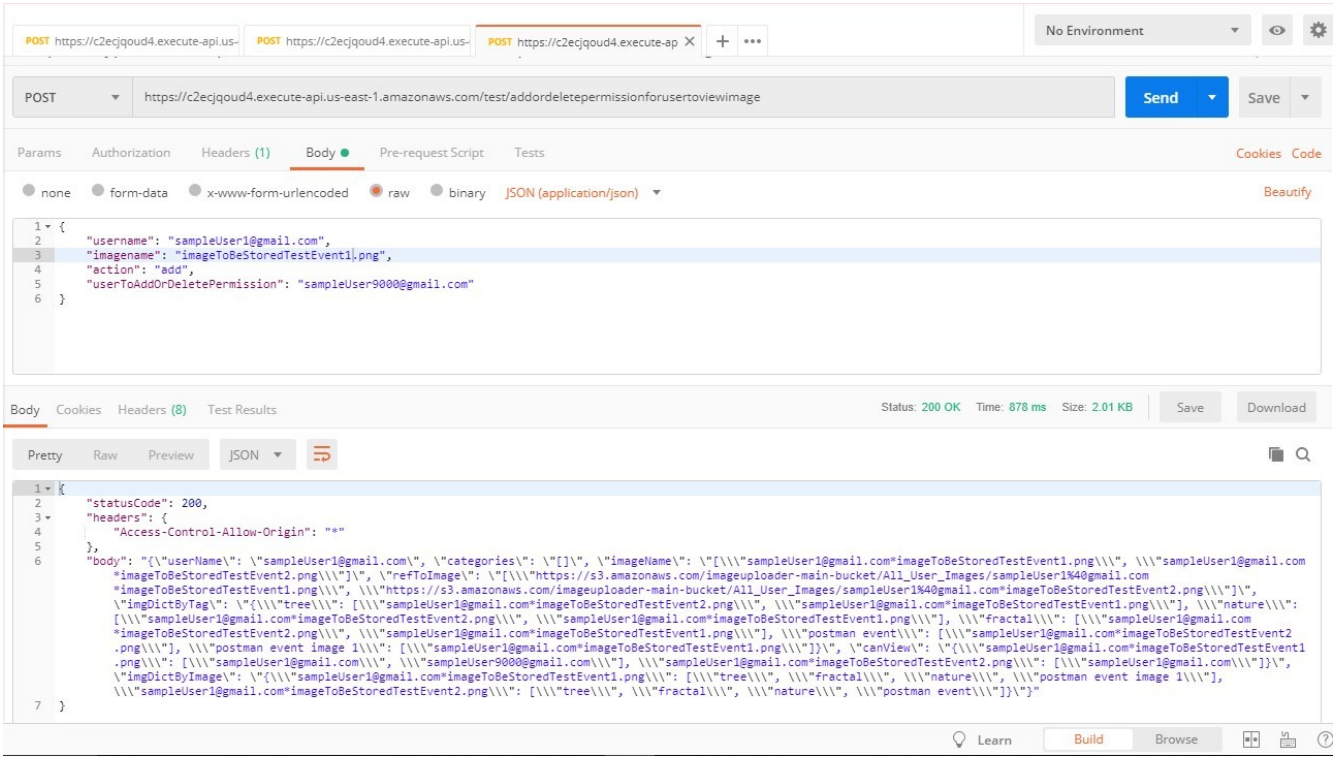


POST:
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/addordeletepermissionforusertoviewimage

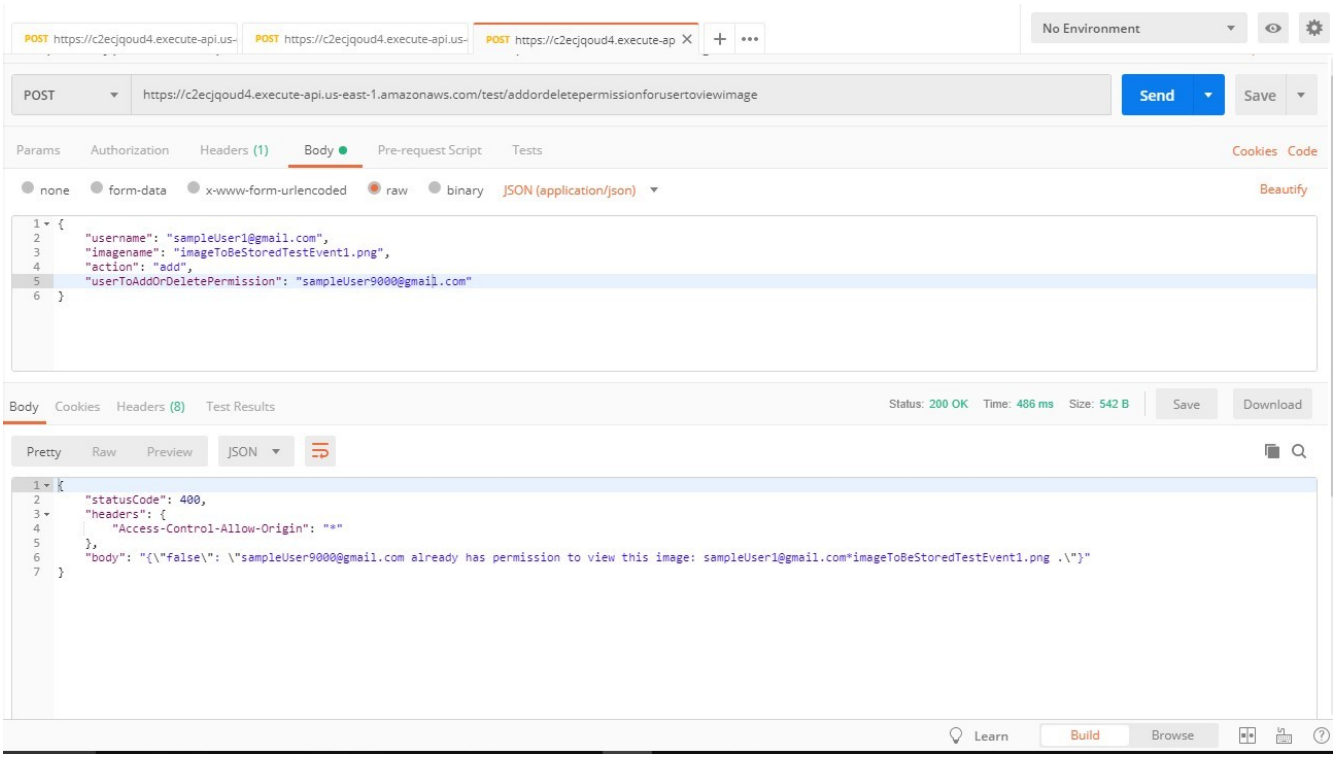
Description: Given an existing username (the owner) and image name, can either "add" or "delete" the access authorization for userToAddOrDeletePermission (the non-owner).

Note: When giving/revoking authorization to a user that is not the owner of the image, the user does not necessarily have to be an existing user.

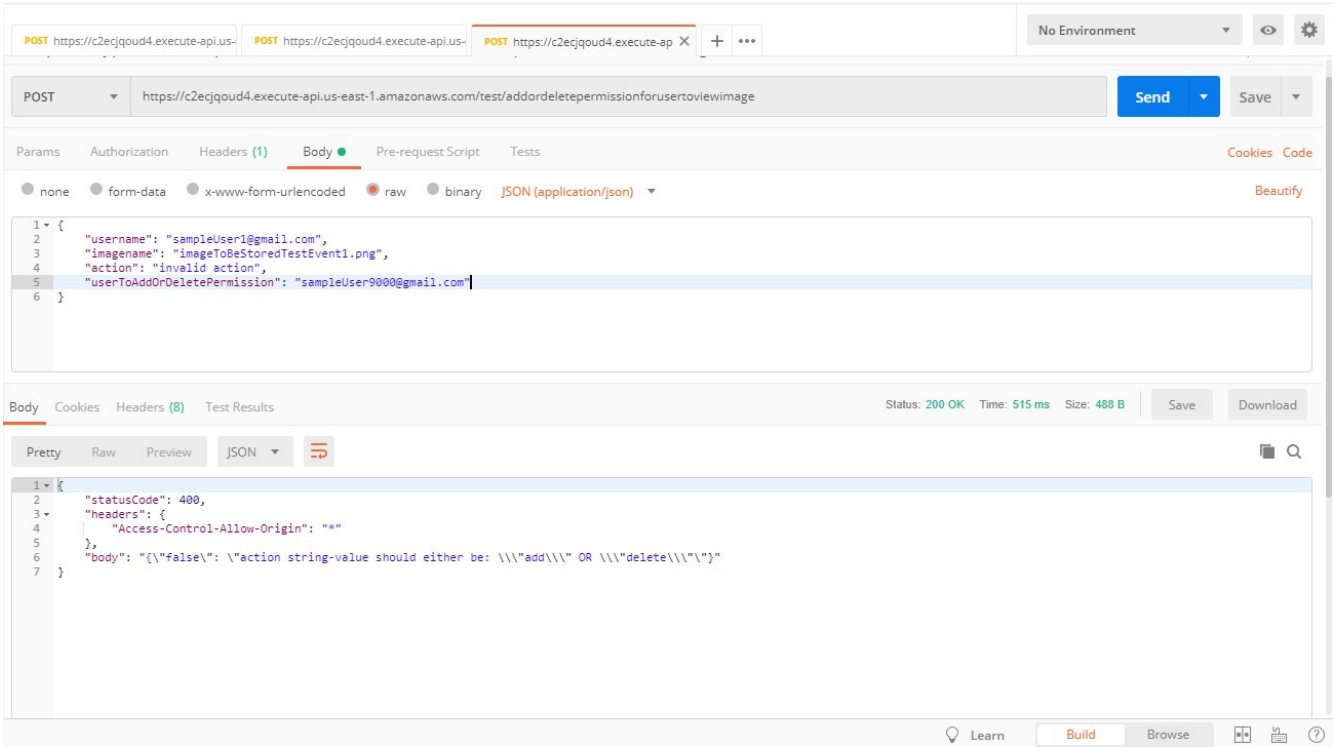
Screenshot of giving access to a user that isn't the owner ("username") of the image:



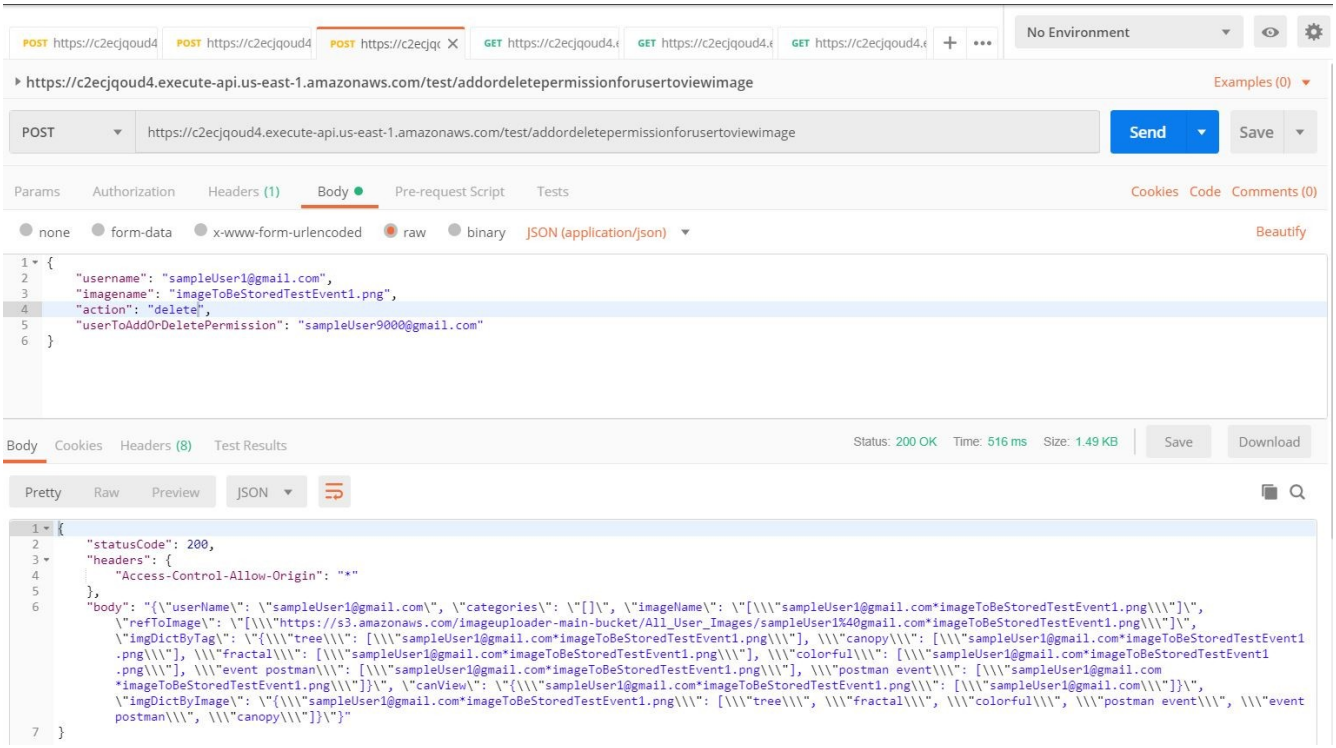
Screenshot of when attempting to give a non-owner (already has access) access to an image:



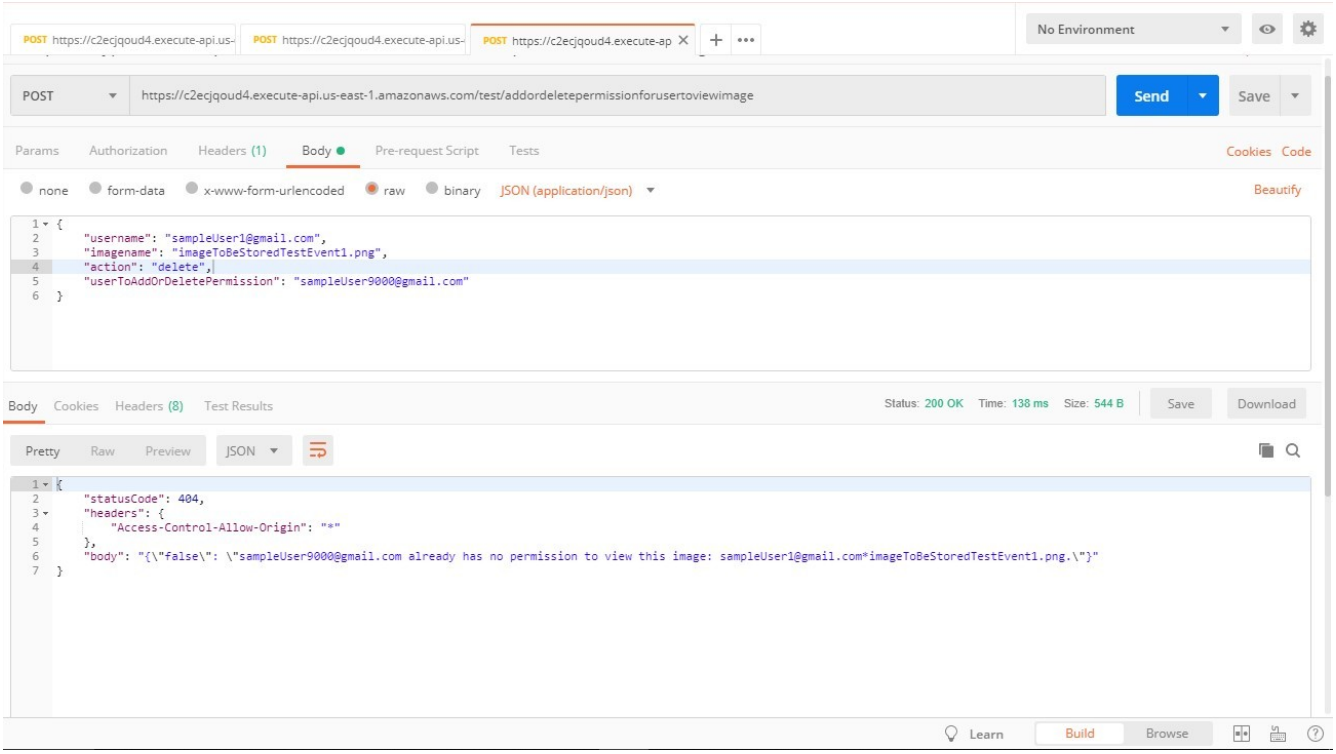
Screenshot of when attempting to pass an “action” that is neither "add" nor "delete":



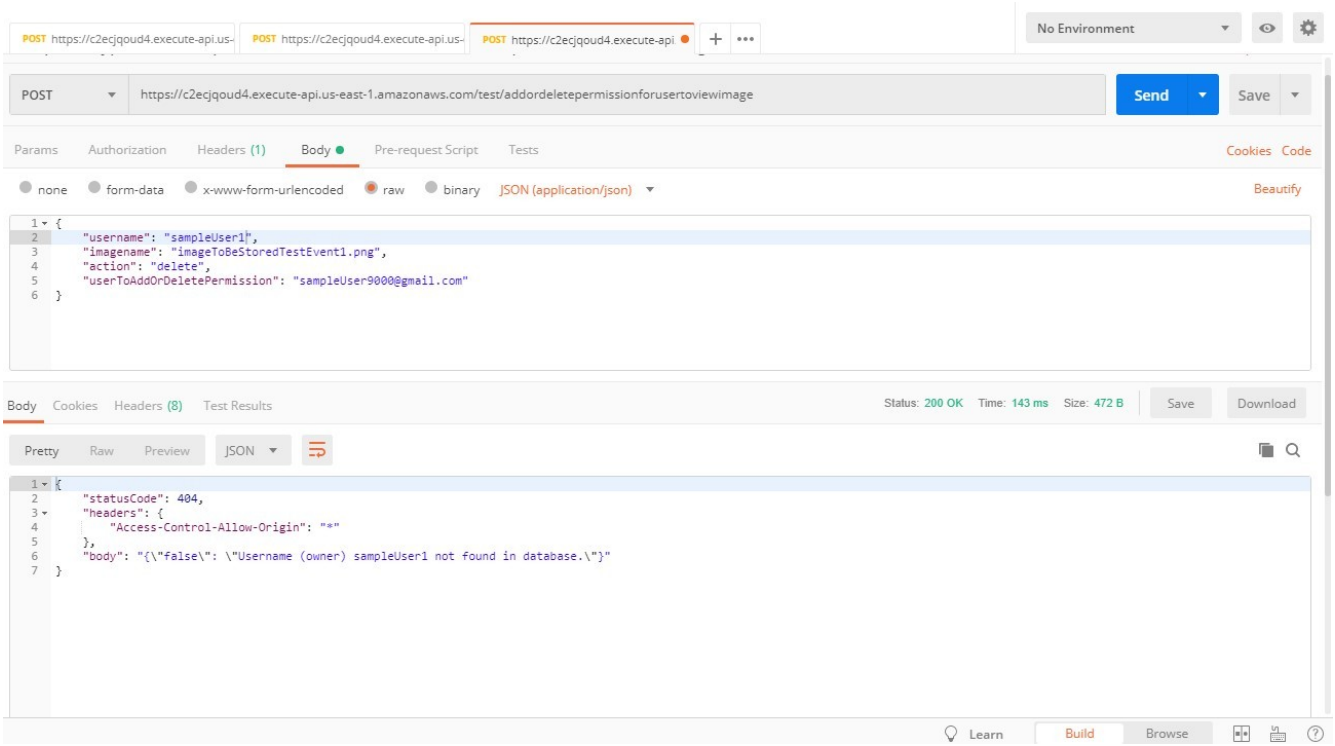
Screenshot of revoking access to a user that isn’t the owner (“username”) of the image:



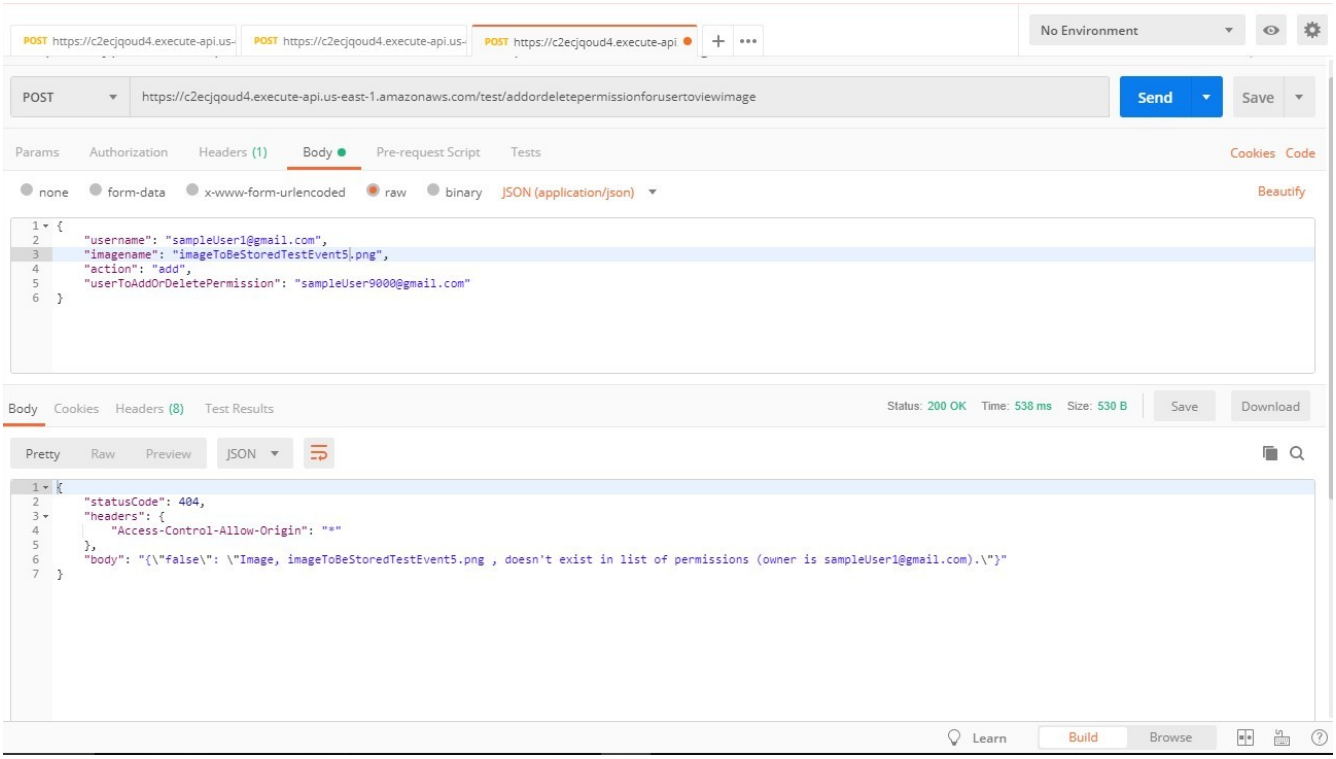
Screenshot of when attempting to revoke non-owner (already had no access) access to an image:



Screenshot of when username (owner) doesn't exist in database:



Screenshot of when given a valid owner (username), an image's viewing permissions are attempted to be modified with either "add" or "delete", but the image itself does not currently exist.



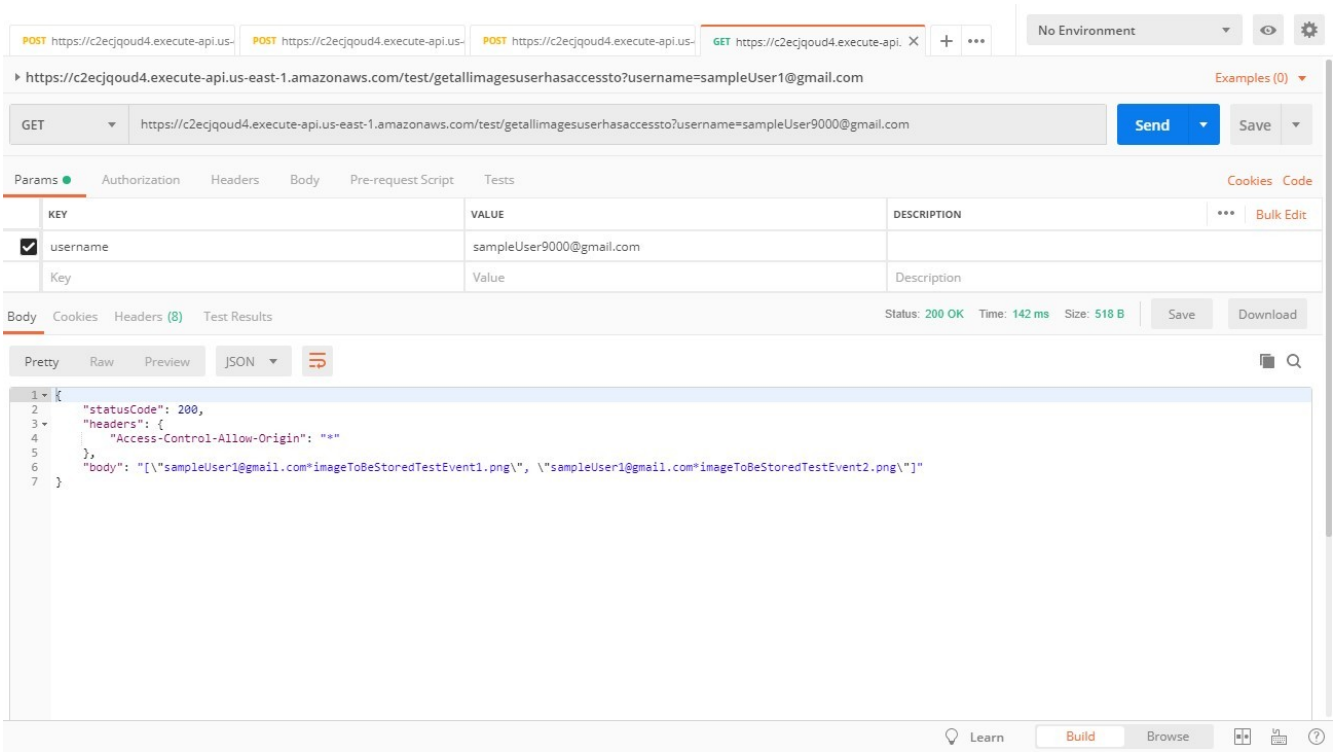
GET:

https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/getallimagesuserhasaccessto?username=sampleUser9000@gmail.com

Description: Given a username, return a list of all images shared with username. The list EXCLUDES images that the username owns. Each image's name format is "username@domain.com*sampleImageName.png".

If username cannot be found in database, an empty list is returned.

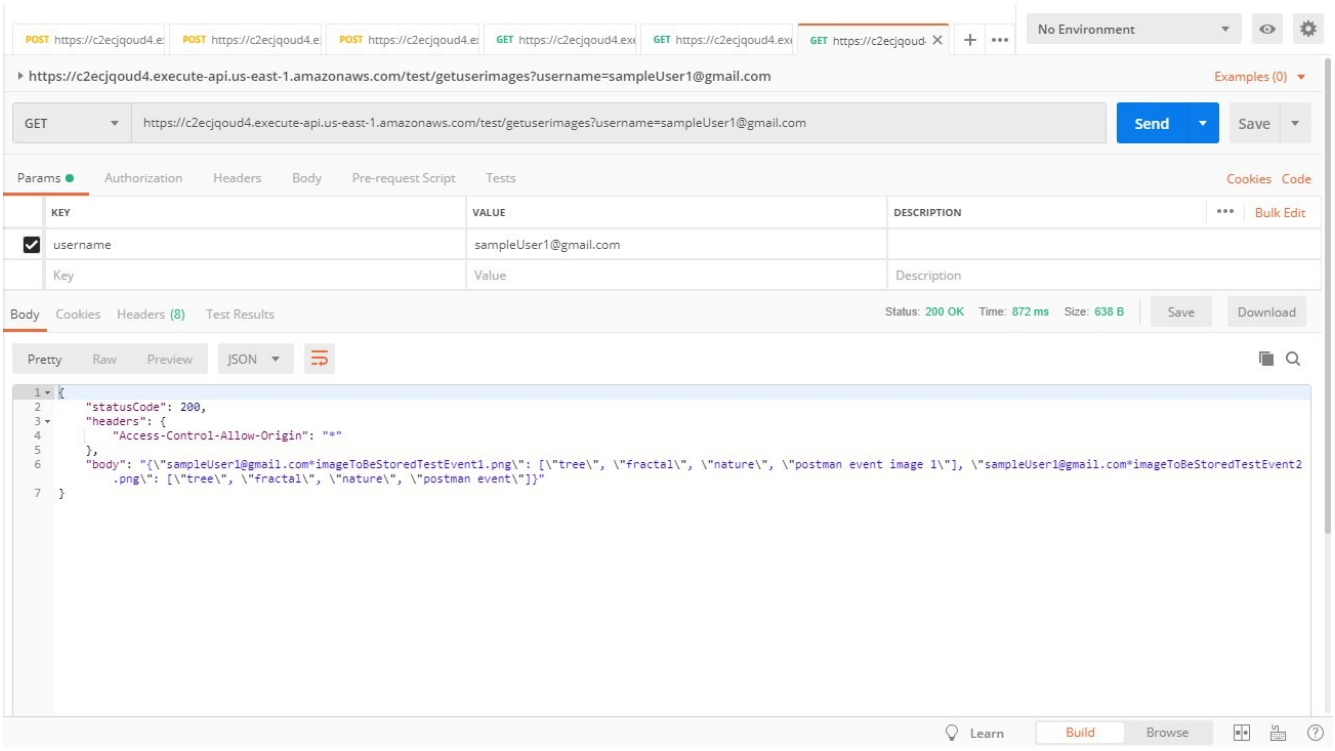
Screenshot:



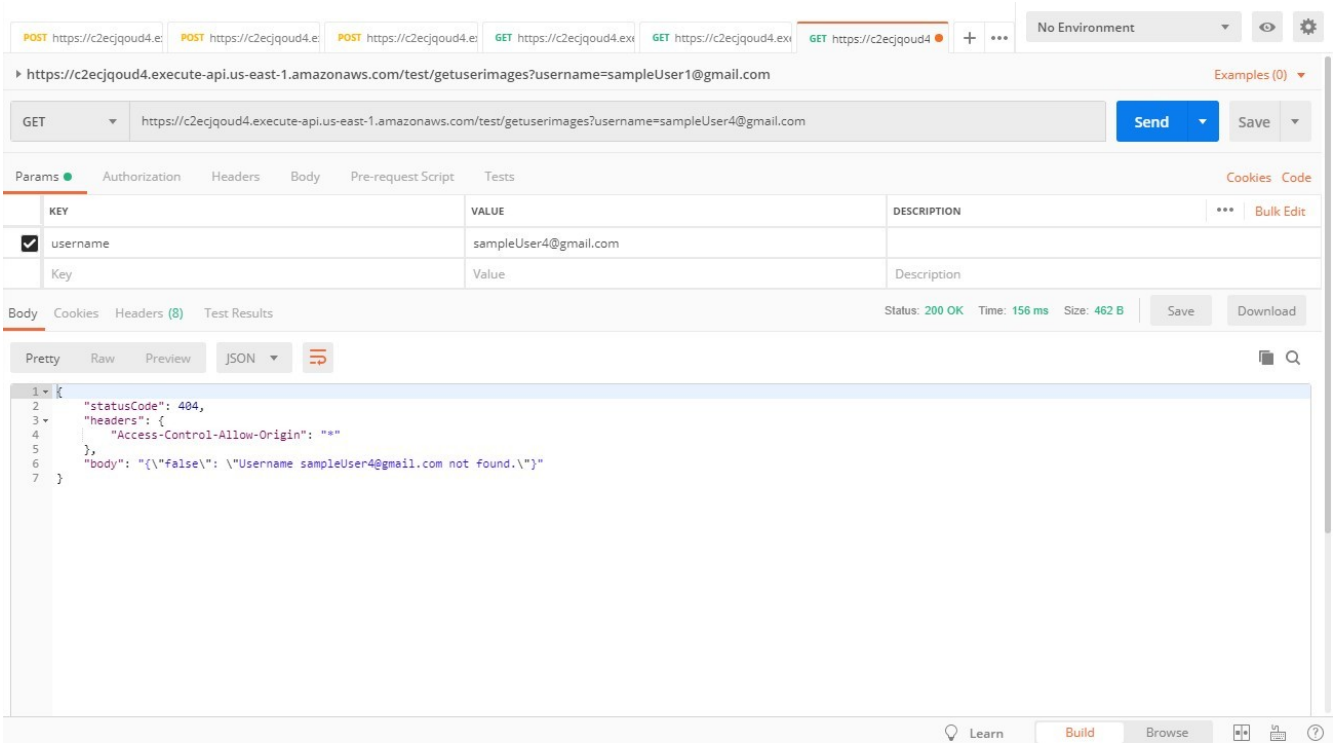
GET:

https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/getuserimages?username=sampleUser1@gmail.com

Description: Use this endpoint to get a dictionary of all images, each with a list of tags, for a particular user. Username is the owner of the images. This essentially gets all images owned by username where each image is in format of "username@domain.com*sampleImageName.png".



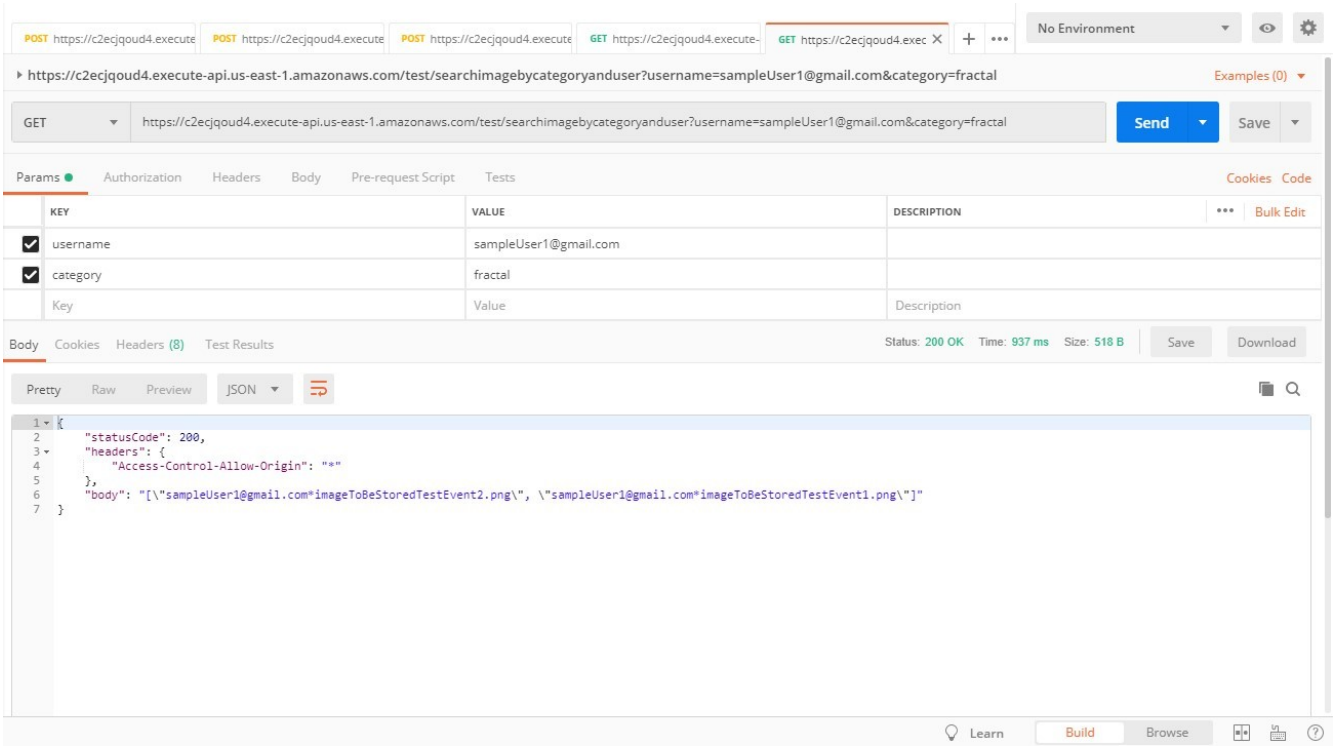
Screenshot of when user is not found in database:



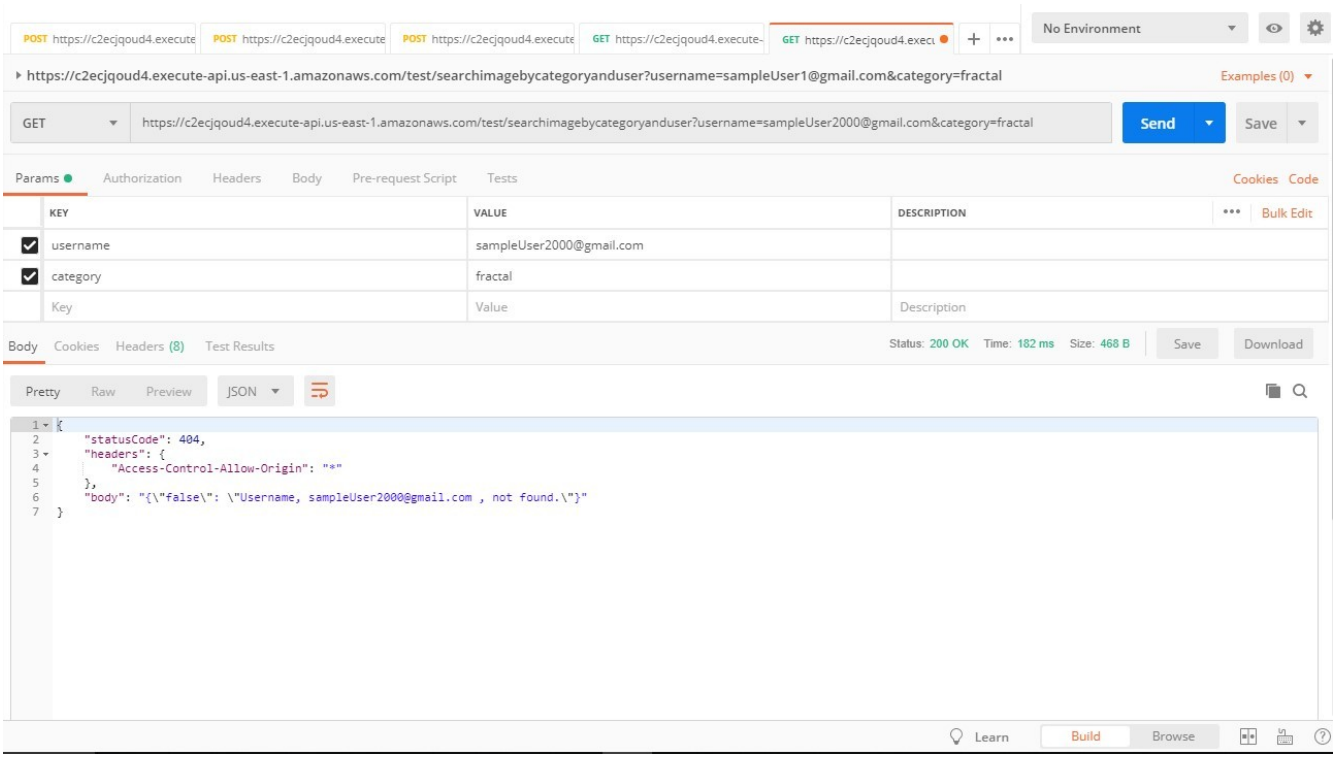
GET:
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/searchimagebycategoryanduser?username=sampleUser1@gmail.com&category=fractal

Description: Given an existing username (owner) and category (or tag), returns the list of images (must extract image name from each corresponding element in the list) associated with the passed in tag. Each image's name format is "username@domain.com*sampleImageName.png".

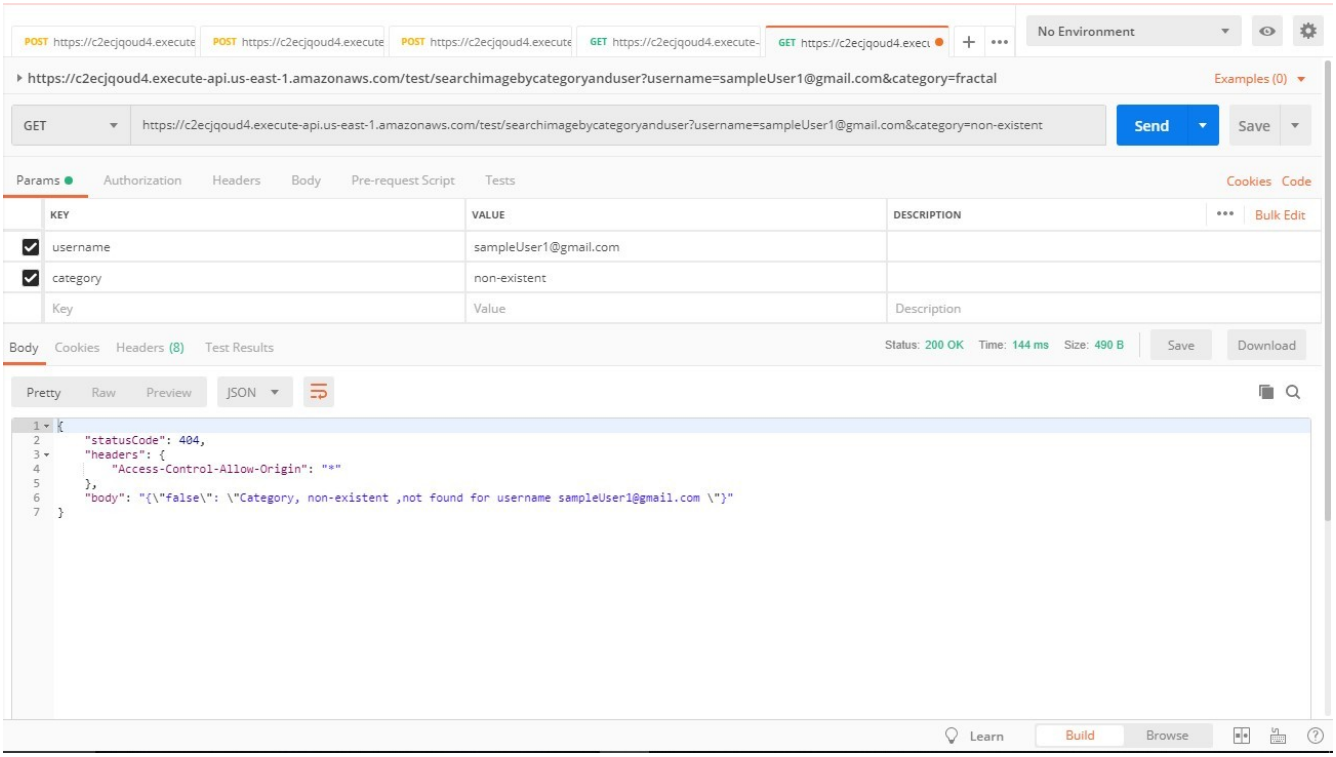
Screenshot:



Screenshot of when username specified doesn't exist in database:



Screenshot of when category not found for a user:

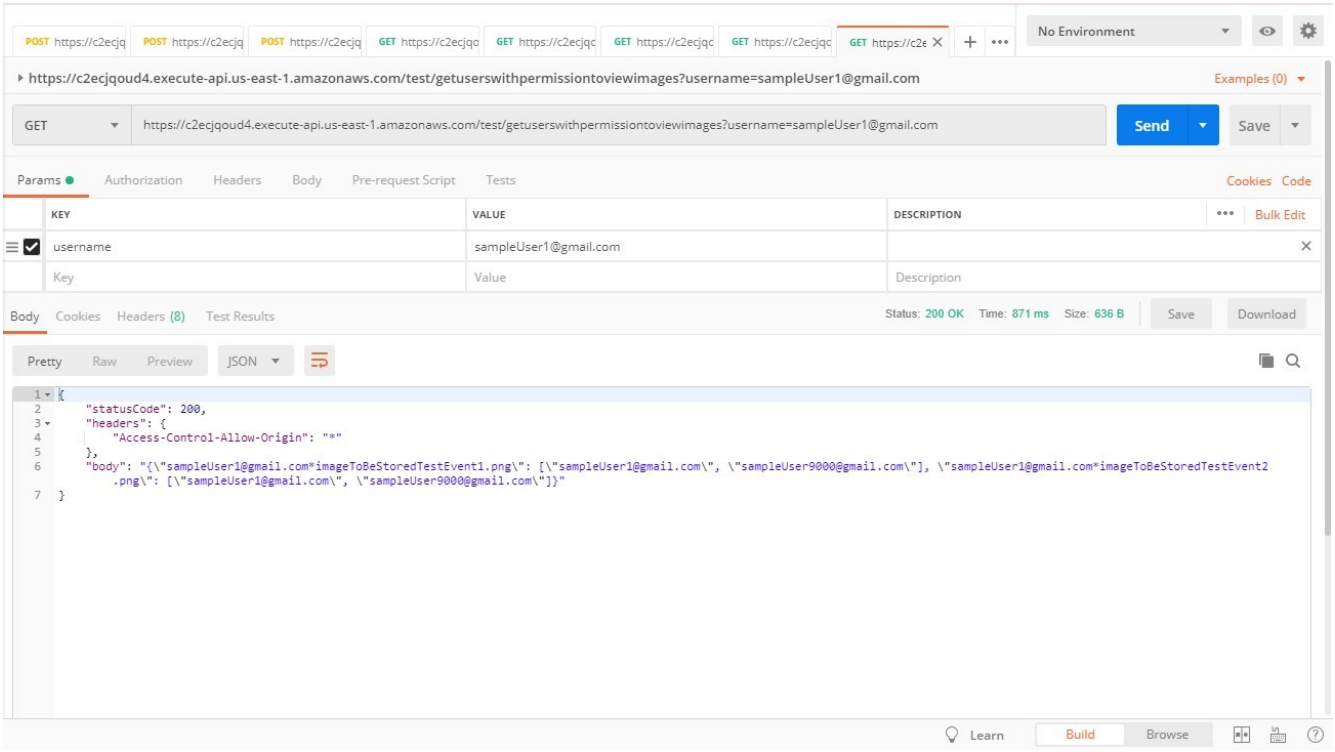


GET:

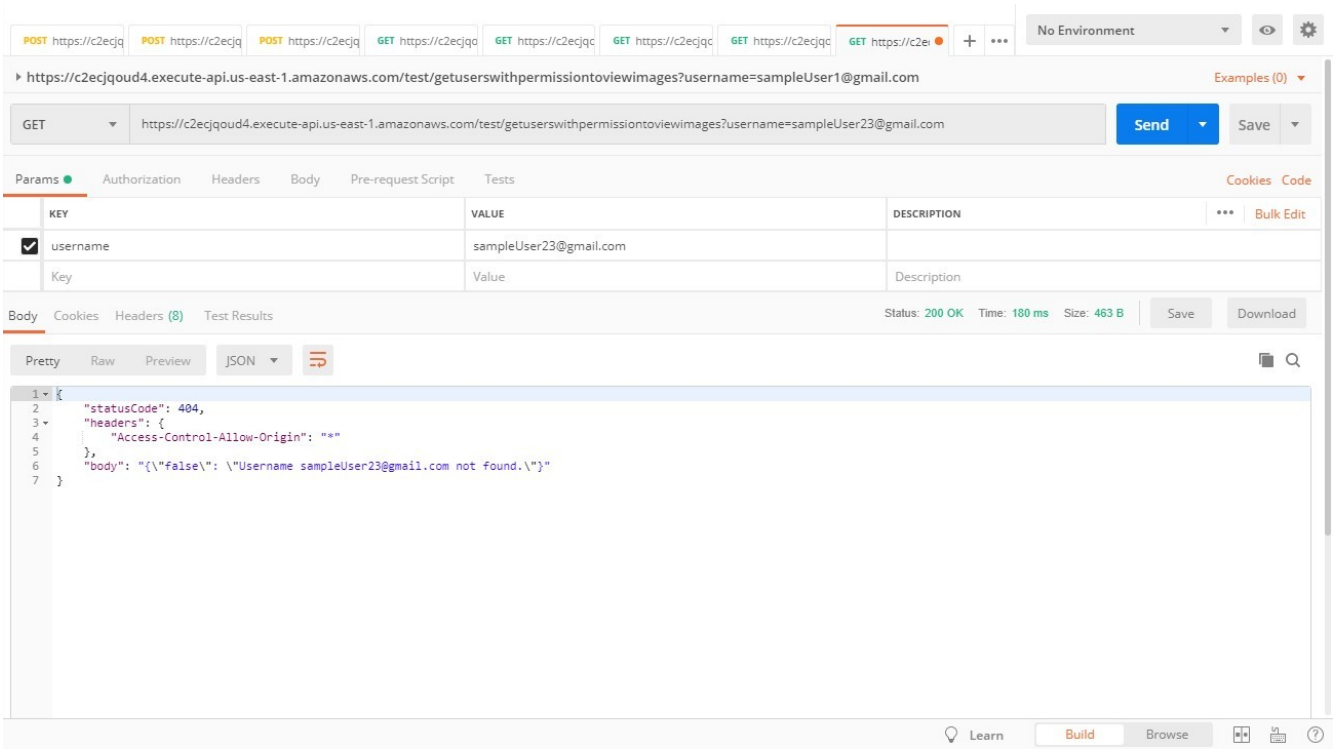
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/getuserswithpermissiontoviewimages?username=sampleUser1@gmail.com

Description: Given an existing username (owner), returns a dictionary of all images (image name must be extracted first with some sort of string split() function where the delimiter is the '*' character), each with a list of usernames that are allowed to access their respective images.

Screenshot:



Screenshot of when username (owner) not found:


POST https://c2ecjq
POST https://c2ecjq
POST https://c2ecjq
GET https://c2ecjqo
GET https://c2ecjqc
GET https://c2ecjqc
GET https://c2ecjqc
GET https://c2e
No Environment
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/getuserswithpermissiontoviewimages?username=sampleUser1@gmail.com
Examples (0)
GET
https://c2ecjqoud4.execute-api.us-east-1.amazonaws.com/test/getuserswithpermissiontoviewimages?username=sampleUser23@gmail.com
Send
Save
Params
Authorization
Headers
Body
Pre-request Script
Tests
Cookies
Code
KEY
VALUE
DESCRIPTION
Bulk Edit
username
sampleUser23@gmail.com
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 180 ms
Size: 463 B
Save
Download
Pretty
Raw
Preview
JSON
{
"statusCode": 404,
"headers": {
"Access-Control-Allow-Origin": "*"
},
"body": "{\"false\": \"Username sampleUser23@gmail.com not found.\"}"
}
Learn
Build
Browse