Riley Kraft
cs493-400
12/03/2018

Final Project

DESCRIPTION

The premise for this project is a new zoo, sponsored by OSU, which manages enclosures and animals. Owners, also know as users, are responsible for managing the enclosures. As new animals are brought to the zoo, owners can select animals to move into their respective enclosures.

Rules of the zoo API:
1) Every enclosure must have an owner. Owners establish their own enclosures.
2) An owner may own zero or more enclosures.
3) An enclosure may not be owned by more than one owner.
4) An enclosure may house more than one animal.
5) An animal may not be housed in more than one enclosure at a time.
6) Animals do not have to be housed in an enclosure. They can live in general housing (null enclosure) until an owner moves them into an enclosure.
7) An owner/user must be logged in to access any and all zoo data.
8) Any owner can add a new animal to general housing.
9) Any owner can remove an animal from the zoo, so long as that animal is in general housing or in one of their owned enclosures.
10) An owner may not remove an animal from the zoo which is housed in another owner's enclosure.
11) All animals start out in general housing.
12) All owners can view data on all enclosures and all animals.
13) Owners may only move animals from general housing into one of their enclosures.
14) An owner cannot move an animal into or out of another owner's enclosure.
15) An owner can only make changes to their own enclosure(s).
16) An owner can only make changes to an animal in general housing or in one of their owned enclosures.
17) If an owner shuts down one of their enclosures, the housed animals will return to general housing.
18) To remove an animal from an enclosure, the owner must first send the animal back to general housing, then move it into another enclosure.
19) An owner can view only their own user profile.
20) The API manager is responsible for creating, viewing, editing, and deleting all user profiles.

To demonstrate the rules and functionalities of the zoo API, I have included a Postman test suite. The test suite features all of the above rules by authenticating client and user credentials, creating a new user, authorizing that user to create, view, edit and delete an animal in an enclosure, and then delete that new user. In addition, authorization restrictions are demonstrated with a second logged in user attempting to access edit and delete functionalities of the new user's profile, enclosure and housed animal. Finally, the second user is used to demonstrate adding and removing relationships between an animal and an enclosure. Note: there are over 100 requests resulting in over 200 tests to demonstrate the functionality of the zoo API. Please take the time to read the README file in order to setup and run the tests as intended, particularly to avoid errors and/or crashes.

On another note, the Postman tests handle pagination links, but they do not handle self-links. If you manually test the pagination and/or self links, 1) you MUST setup the requests' Authorization type to "Bear Token" and token to "{{auth0_token}}", and 2) set the "Accept" header to "application/json", before hitting Send.

Lastly, I've included some CRUD operations for my Auth0 user management, in addition to Create. However, in accordance with this Piazza post https://piazza.com/class/jm704r7q4ot3s1?cid=246, I have not included features for user objects like pagination or an Update method.

ENTITY JSON SAMPLES

****** *<= Not kept in datastore, dynamically generated and returned by the API*

Enclosure

```
{
   "id": "1234123412341234",
   "number": "003"
   "type": "savannah",
   "size": 500,
   "owner": "abcd-efg-hijk-1234@osu-zoo.com",
**"animals": [
      {
         "id": "4321432143214321",
         "name": "Simba",
         "self": "https://kraftme-223903.appspot.com/animals/4321432143214321"
      },
      {
         "id": 5678567856785678,
         "name": "Nala",
         "self":"https://kraftme-223903.appspot.com/animals/5678567856785678"
      }
   ]
**"self": "https://kraftme-223903.appspot.com/enclosures/1234123412341234"
}
```

Animal

```
{
   "id": "4321432143214321",
   "name": "Simba"
   "species": "lion",
   "age": 7,
   "enclosure": "1234123412341234",
**"enclosure": {                      <= NOTE: replaces datastore "enclosure" property value in responses
         "id": "1234123412341234",
         "number": "003",
         "self": "https://kraftme-223903.appspot.com/enclosures/1234123412341234"
   },
**"self": "https://kraftme-223903.appspot.com/animals/4321432143214321"
}
```

User (from Auth0)

```
{
        "email": "32ae107a-6717-43e1-85b4-51aae7a87c81@osu-zoo.com",
        "email_verified": true,
        "updated_at": "2018-12-04T04:45:56.119Z",
        "picture": "https://s.gravatar.com/avatar/67cac680fbb84bc4a742c43b9740dab9?
                        s=480&r=pg&d=https%3A%2F%2Fcdn.auth0.com%2Favatars%2F32.png",
        "user_id": "auth0|5c0307e4649bd32de3bdb6a0",
        "nickname": "32ae107a-6717-43e1-85b4-51aae7a87c81",
        "identities": [
            {
                "user_id": "5c0307e4649bd32de3bdb6a0",
                "provider": "auth0",
                "connection": "Username-Password-Authentication",
                "isSocial": false
            }
        ],
        "created_at": "2018-12-01T22:15:00.635Z",
        "name": "32ae107a-6717-43e1-85b4-51aae7a87c81@osu-zoo.com",
        "last_login": "2018-12-04T04:45:56.119Z",
        "last_ip": "107.178.200.172",
        "logins_count": 52
}
```

API

{{app_base_url}}: https://kraftme-223903.appspot.com

The base url for my app requests is saved in my Postman environment as the global variable app_base_url. The client authentication for creating users is saved as the global variable {{client_token}}, and the user authentication for all other requests is {{auth0_token}}. Some ERROR demonstration requests are hard coded with invalid tokens.

**JWTs are used for authorization*

ALL requests require a type of authorization! Read below for authorization requirements.

**All entities have a base URL that returns the collection of that entity*
**All root entity collections implement paging of 5 items at a time*
**All entities have a self link that points to the most canonical representation of that entity instance*
**All entities support at least 3 properties in addition to any relations*
**Every entity supports read operations*

GET   /enclosures

This request does not require body data, however it does require a user id_token and the Accept header set to application/json. The authorizer must be any logged in user, the client cannot access this data. This request will return an “items” list of up to 5 enclosure objects from the datastore, a “next” link if applicable, and a “collectionSize” count. Each enclosure object features its id, number, type, size, owner, self-link, and array of animals. Each animal object in the animals array features its id, name, and self-link. A “next” link will direct the user to the next page of enclosure items. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. The same goes for using an enclosure or animal self link. Lastly, the “collectionSize” is just a count of the total number of enclosure items returned by the request. You must us the “next” links to view all of them.

Postman Tests: Get All Enclosures 1 (prerequisite request) → Get All Enclosures 2, Confirm Enclosure Count Increment Global, Confirm Deleted Enclosure Global

> *ERRORS*
> This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not for an existing user). Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.
>
> Postman Tests: Get All Enclosures ERROR 1 - 3

GET /animals
This request does not require body data, however it does require a user id_token and the Accept header set to application/json. The authorizer must be any logged in user, the client cannot access this data. This request will return an “items” list of up to 5 animal objects from the datastore, a “next” link if applicable, and a “collectionSize” count. Each animal object features its id, name, species, age, enclosure, and self-link. Each animal’s enclosure object, if not null, features its id, number, and self-link. A “next” link will direct the user to the next page of animal items. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. The same goes for using an animal or enclosure self link. Lastly, the “collectionSize” is just a count of the total number of animal items returned by the request. You must us the “next” links to view all of them.

Postman Tests: Get All Animals 1 (prerequisite request) → Get All Animals 2, Confirm Animal Count Increment, Confirm Deleted Animal

> *ERRORS*
> This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not for an existing user). Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.
>
> Postman Tests: Get All Animals ERROR 1 - 3

GET /users
This request does not require body data, however it does require a client access_token and the Accept header set to application/json. The authorizer must be a client, users cannot access this data. This request will return all user objects from Auth0. Each user object features its email, email_verified, update_at, picture, user_id, nickname, identities array, created_at, name, last_login, last_ip, and logins_count. Each user object’s identities array feautres its user_id, provider, connection, and isSocial.

Postman Tests: Get Users, Confirm User Count Increment, Confirm Deleted User

ERRORS
This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is not a client access token. This request can also result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman Tests: Get Users ERROR 1 - 3

GET /users/:userID/enclosures
This request does not require body data, however it does require a user id_token and the Accept header set to application/json. The authorizer must be a currently logged in user whose email is set as the :userID. The client and any user whose email is not the :userID cannot access this data. This request will return an “items” list of up to 5 enclosure objects from the datastore whose owner is the :userID, a “next” link if applicable, and a “collectionSize” count. Each enclosure object features its id, number, type, size, owner, self-link, and array of animals. Each animal object in the animals array features its id, name, and self-link. A “next” link will direct the user to the next page of enclosure items. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. The same goes for using an enclosure or animal self link. Lastly, the “collectionSize” is just a count of the total number of enclosure items whose owner is the :userID returned by the request. You must us the “next” links to view all of them.

Postman Tests: Get User’s Enclosures 1 (prerequisite request) → Get User’s Enclosures 2, Confirm Enclosure Count Increment User, Confirm Deleted Enclosure User, Get User’s Enclosures 3

ERRORS
This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is for the client, or if the Authorization code belongs to another user whose email is not the :userID. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman Tests: Get User’s Enclosures ERROR 1 - 4

GET /enclosures/:enclosureID/animals
This request does not require body data, however it does require a user id_token and the Accept header set to “application/json”. The authorizer must be a currently logged in user whose email matches the :enclosureID’s owner property value. The client and any user whose email is not the :enclosureID’s owner cannot successfully make this request. This request will return an “items” list of up to 5 animal objects from the datastore whose enclosure property value is the :enclosureID, a “next” link if applicable, and a “collectionSize” count. Each animal object features its id, name, species, age, enclosure object, and self-link. Each animal’s enclosure object features its id, number, and self-link. Each of the animal’s enclosure objects should be the same since all the animals should bei nthe same enclosure. A “next” link will direct the user to the next page of animal items, if applicable. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. The same goes for using an animal or enclosure self link.

Lastly, the “collectionSize” is just a count of the total number of animal items whose enclosure is the :enclosureID returned by the request. You must us the “next” links to view all of them.

Postman Tests: Get User’s Enclosures 1 (prerequisite request) → Get User’s Enclosures 2, Confirm Enclosure Count Increment User, Confirm Deleted Enclosure User

> *ERRORS*
> This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is for the client, or if the Authorization code belongs to another user whose email is not the :userID. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.
>
> Postman Tests: Get User’s Enclosures ERROR 1 - 4

GET /enclosures/:enclosureID
This request does not require body data, however it does require a user id_token and the Accept header set to application/json. The authorizer must be any logged in user, the client cannot access this data. This request will return a single enclosure object whose ID matches the :enclosureID in the URL. The enclosure object features its id, number, type, size, owner, self-link, and animals array. The enclosure’s animal array, if not empty, features a list of animal objects. Each animal object features its id, name, and self-link. Selecting any self-link will direct the user to the canonical representation of that object. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. Selecting an enclosure object’s self-link from any other request will direct you to this request.

Postman Tests: Get an Enclosure, Confirm Updated Enclosure Contents w/ Location

> *ERRORS*
> This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not from an existing user). Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.
>
> Postman Tests: Get an Enclosure ERROR 1 - 3

GET /animals/:animalID
This request does not require body data, however it does require a user id_token and the Accept header set to application/json. The authorizer must be any logged in user, the client cannot access this data. This request will return a single animal object whose ID matches the :animalID in the URL. The animal object features its id, name, species, age, enclosure, and self-link. The animal’s enclosure object, if not null, features its id, number, and self-link. Either self-link will direct the user to the canonical representation of that object. You must provide the user’s id_token again as authorization, as well as set the Accept header to “application/json” again before hitting Send. Selecting an animal object’s self-link from any other request will direct you to this request.

Postman Tests: Get an Animal, Confirm Updated Animal Contents w/ Location, Confirm Assigned Animal 1, Confirm Animal is No Longer Assigned to Enclosure, Confirm Assigned Animal 2, Confirm Removed Animal

ERRORS
This request can result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not from an existing user). Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman Tests: Get an Animal ERROR 1 - 3

GET /users/:userID
This request does not require body data, however it does require a client or user access_token and the Accept header set to application/json. The authorizer must be the client or logged in user whose user_id matches the :userID, any other user cannot access this data. This request will return a single user object from Auth0 whose user_id matches the :userID in the URL. The user object features its email, email_verified, update_at, picture, user_id, nickname, identities array, created_at, name, last_login, last_ip, and logins_count. “owner” properties do not feature self-links as accessing a single user’s profile is restricted to the client access_token and the current logged in user’s id_token. No other user can access the :userID’s profile.

Postman Tests: Get User 1, Get User 2

ERRORS
This request can result in a 401 error if the request does not include an Authorization code, or if the user access_token is not for the user whose user_id matches :userID. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman Tests: Get User ERROR 1 - 3

**Every entity supports edit operations*
**Access to items in database are appropriately limited by owner*

PUT /enclosures/:enclosureID
This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“number”: “[string]”, “type”: “[string]”, “size”: [integer]}. Submitting this request will update the data for the ENCL object in the datastore whose id matches the :enclosureID provided in the url. This request only returns a 303 response with a Location to the updated object in the event of a successful PUT. The Location is the GET request to confirm their updated data for :enclosureID. The user cannot change the enclosure to animal relationship in this request. Also, the enclosure’s owner property cannot be updated as an enclosure and a user have a permanent relationship.

Postman tests: Update User’s Enclosure, Confirm Updated User Enclosure Contents w/ Location

ERRORS
This request can result in 403 errors 1) if the :enclosureID is not a valid format, and 2) if the

user does not provide all the required data in the request body. It will result in a 409 error if the updated number provided for :enclosureID already belongs to another enclosure. As well, this request can result in a 404 error if the :enclosureID is a valid format but is not found in the datastore. Or a 401 error if an invalid or no Authorization is provided, or if the enclosure belongs to another user not identified by the Authorization id_token. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman tests: Update User’s Enclosure ERROR 1 - 8

PUT /animals/:animalID

This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“name”: “[string]”, “species”: “[string]”, “age”: [integer]}. Submitting this request will update the data for the ANIM object in the GAE datastore whose id matches the :animalID provided in the url. This request only returns a 303 response with Location in the event of a successful PUT. The Location is the GET /animals/:animalID request to confirm the updated data for :animalID. Note: the user cannot update an animal’s enclosure with this request. The user must utilize the appropriate PUT and DELETE requests to add and remove animals from their enclosures.

Postman tests: Update Animal, Confirm Update Animal Contents w/ Location

ERRORS

This request can result in 403 errors 1) if the :animalID is not a valid format, and 2) if the user does not provide all the required data in the request body. It will results in a 409 error if the updated name provided for :animalID already belongs to another   animal. As well, this request can result in a 404 error if the :animalID is a valid format but is not found in the datastore. It can result in a 401 error if no Authorization is provided, or if the animal is being housed in an enclosure that belongs to another user not identified by the Authorization id_token.. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.

Postman Updates: Update Animal ERROR 1 - 8

**Every entity supports create operations*

POST /enclosures

This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“number”: “[string]”, “type”: “[string]”, “size”: [integer]}. This request does not require the “owner” attribute as this is automatically filled in based on the user email found in the provided id_token. The POST requires the id_token for the current user as its Authorization, as well as the Accept header set to “application/json”. Submitting this request will create a new ENCL datastore object, assign it a unique id from the key, store the data provided by the user, add the user’s email from their id_token to the “owner” field, and return the new object’s key. NOTE: enclosures can not have the same name. The user must send another GET /enclosures request to confirm the creation of the new enclosure.

Postman tests: Create User Enclosure, Confirm Enclosure Count Increment Global, Confirm Enclosure Count Increment User

> *ERRORS*
> This request can result in 403 errors if the user does not provide all the required data in the request body, or a 409 Conflict error if the user attempts to give the new enclosure the same number as an existing enclosure. This request can also result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not for a user). Lastly, this request can result in a 406 Not Accepted if the body data provided is not application/json.
>
> Postman tests: Create User Enclosure ERROR 1 - 5

POST /animals
This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“name”: “[string]”, “species”: “[string]”, “age”: [integer]}. This request does not require the “enclosure” attribute as this is automatically filled in with the enclosure ID when using the appropriate PUT request to add an animal to and enclosure. The POST requires the id_token for the current user as its Authorization, as well as the Accept header set to “application/json”. Submitting this request will create a new ANIM datastore object, assign it a unique id from the key, store the data provided by the user, and return the new object’s key. NOTE: animal can not have the same name. The user must send another GET /animals request to confirm the creation of the new enclosure.

Postman tests: Create Animal, Confirm Animal Count Increment

> *ERRORS*
> This request can result in 403 errors if the user does not provide all the required data in the request body, or a 409 Conflict error if the user attempts to give the new animal the same name as an existing animal. This request can also result in 401 errors if the request does not include an Authorization code, or if the Authorization code is invalid (not for a user). Lastly, this request can result in a 406 Not Accepted if the body data provided is not application/json.
>
> Postman tests: Create Animal ERROR 1 - 5

NOTE: See further below for method to create a user account

**Every entity supports delete operations*
**Access to items in database are appropriately limited by owner*

DELETE /enclosures/:enclosureID
This request does not require any body data, however it does require a user id_token Authorization, as well as having the Accept header set to “application/json”. The requester must provide a valid :enclosureID and user id_token, in which the enclosure’s owner property value matches the user email derived from the id_token, in order to delete that enclosure’s object from the datastore. This request only returns a 204 No Content response in the event the DELETE was successful. The user must send the GET /enclosures and/or GET /users/:userID/enclosures requests again in order to confirm the enclosure object they deleted no longer exists. Note: if any animals are living in this enclosure when it is deleted, the animals will return to general housing by having their enclosure property value set to

null. The user must send GET /animals/:animalID request(s) to confirm the animal(s) enclosure properies are set to null.

Postman tests: Delete User Enclosure, Confirm Delete Enclosure Global, Confirm Deleted Enclosure User, Confirm Animal is No Long Assigned to Enclosure

> *ERRORS*
> This request can result in a 403 error if the :enclosureID is not a valid format, a 404 error if the :enclosureID in not found in the GCP datastore, or a 401 error if the :enclosureID's owner does not match the user email found in the id_token. The request can also result in 401 errors if an invalid or no user id_token is provided, or a 406 error if the Accept header is set to anything other that "application/json".
>
> Postman tests: Delete User Enclosure ERROR 1 - 3

DELETE   /animals/:animalID

This request does not require any body data, however it does require a user id_token Authorization, as well as having the Accept header set to "application/json". The requester must provide a valid :animalID and user id_token in order to delete that animal's object from the datastore. If the animal is currently housed in an enclosure, the enclosure's owner property value must match the user email derived from the Authorization id_token. This request only returns a 204 No Content response in the event the DELETE was successful. The user must send the GET /animals request again in order to confirm the animal object they deleted no longer exists.

Postman tests: Delete Animal, Confirm Deleted Animal

> *ERRORS*
> This request can result in a 403 error if the :animalID is not a valid format, a 404 error if the :animalID in not found in the GCP datastore or if the :animalID's enclosure property value is not found as a valid enclosureID. It can also results in a 401 error if the :animalID's enclosure's owner does not match the user email found in the id_token. It can also result in 401 errors if an invalid or no user id_token is provided, or a 406 error if the Accept header is set to anything other that "application/json".
> Postman tests: Delete Animal ERROR 1 - 3

DELETE   /users/:userID

This request does not require any body data, however it does require a client access_token for Autorization and the Accept header to be set to "application/json". The requester must provide a valid :userID and client access_token in order to delete that users's object from Auth0. This request is restricted to client access by the scope terms of Auth0. This request only returns a 204 No Content response in the event the DELETE was successful. The user must send the GET /users request again in order to confirm the user object they deleted no longer exists.

Postman tests: Delete User 1, Delete User 2, Confirm Deleted Users

> *ERRORS*
> This request can result in a 403 error if the :userID is not a valid format, a 404 error if the :userID in not found in Auth0. The request can also result in 401 errors if an invalid or no client

access_token is provided, or a 406 error if the Accept header is set to anything other that "application/json".

Postman tests: Delete User ERROR 1 - 3

**Relationships are correctly implemented with PUTs and DELETEs*
**Access to items in database are appropriately limited by owner*

PUT   /enclosures/:enclosureID/animals/:animalID

The objective of this request is to the placing an animal, identified by :animalID, in an enclosure, identified by :enclosureID. The request does not require body data, however it does require a user id_token and the Accept header set to "application/json". The authorizer must be a logged in user whose email matches the owner property value of the :enclosureID, any other user cannot access this data. The enclosure property value of :animalID must be null. The user can run GET /animals and GET /users/:userID/enclosures to find compatible ANIM and ENCL objects for an assignment. This request only returns a 303 response with a Location to the GET /animals/:animalID in the event the PUT was successful. The user must send the GET /enclosures/:enclsureID or GET /animals/:enclosureID request again in order to confirm the cargo assignment.

Postman tests: Put Animal in Enclosure 1, Confirm Assigned Animal 1, Put Animal in Enclosure 2, Confirm Assigned Animal 2

ERRORS
This request can result in 403 errors if 1) the :enclosureID or :animalID are not valid formats, and 2) if the :animalID is currently assigned to another enclosure. As well, this request can result in a 404 error if either the :enclosureID or :animalID are not found in the datastore. It can result in a 401 error if the request does not include a user id_token Authorization code, or if the user email from the id_token does not match the owner property value of :enclosureID. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to "application/json".

Postman tests: Put Animal in Enclosure ERROR 1 - 9

DELETE   /enclosures/:enclosureID/animals/:animalID

The objective of this request is to simulate the removal of an animal, identified by :animalID, from its enclosure, identified by :enclosureID. The request does not require body data, however it does require a user id_token and the Accept header set to "application/json". The authorizer must be a logged in user whose email matches the owner property value of the :enclosureID, any other user cannot access this data. The enclosure property value of :animalID must match the :enclosureID. The user can run GET /enclosures/:enclosureID to find an eligible enclosure from which to remove an animal. The request will merely delete the :enclosureID from the :animalID object's enclosure property. This request only returns a 204 response in the event the DELETE was successful. The user must send the GET /animals/:animalID or GET /enclosures::enclosureID requests again in order to confirm the animal removal. Neither the animal nor the enclosure objects will be deleted from or otherwise changed within the datastore. The animal object enclosure property should now be set to null. The enclosure object response will no longer feature an animal object id and self link for the :animalID.

Postman tests: Remove Animal from Enclosure, Confirm Removed Animal

> *ERRORS*
> This request can result in 403 errors if 1) the :enclosureID or :animalID are not valid formats, and 2) if the :animalID enclosure property value does not match :enclosureID. As well, this request can result in a 404 error if either the :enclosureID or :animalID are not found in the datastore. It can result in a 401 error if the request does not include a user id_token Authorization code, or if the user email from the id_token is does not match the owner property value of :enclosureID. Lastly, this request can result in a 406 Not Accepted if the Accept header is not set to “application/json”.
>
> Postman tests: Remove Animal from Enclosure ERROR 1 - 7

**405 Status Code is Supported*

PUT /enclosures

ERRORS

This request results in a 405 error because the api doesn’t allow all ENCL objects to be updated at one time.

Postman test: Update All Enclosures ERROR

PUT /animals

ERRORS

This request results in a 405 error because the api doesn’t allow all ANIM objects to be updated at one time.

Postman test: Update All Animals ERROR

PUT /users/:userID/enclosures

ERRORS

This request results in a 405 error because the api doesn’t allow all ENCL objects whose owner matches :userID to be updated at one time.

Postman test: Update All User’s Enclosures ERROR

DELETE /enclosures

ERRORS

This request results in a 405 error because the api doesn’t allow all ENCL objects to be deleted at one time.

Postman test: Delete All Enclosures ERROR

DELETE /animals

ERRORS

This request results in a 405 error because the api doesn’t allow all ANIM objects to be deleted at one time.

Postman test: Delete All Animals ERROR

DELETE /users/:userID/enclosures
ERRORS
This request results in a 405 error because the api doesn’t allow all ENCL objects whose owner matches :userID to be deleted at one time.

Postman test: Delete All User’s Enclosures ERROR

**Users can create new accounts*
**A method to login is supported*
POST /users
This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“username”: “[string]”, “password”: “[string]”}. As well, the POST requires the access_token for the client as its Authorization. The client access_token is not the same as the user access_token! You must obtain a client access_token in order to create users through the Auth0 Management API. Submitting this request will create a new user for the Auth0 Api and return the new user’s information as a JSON object.

Postman test: Create User 1, Create User Other, Confirm User Count Increment

> *ERRORS*
> This request can result in a 400 error if the user does not provide all the required data in the request body, or a 409 Conflict error if the user attempts to give the new user the same email as an existing user. Lastly, this request can result in a 406 Not Accepted if the body data provided is not application/json.
>
> Postman tests: Create User ERROR 1 - 3

POST /oauth
This request does not require any additional information within Postman as all the required information is hard coded in the server file. The server.js file defines the POST request to the Auth0 API with the client id and secret, along with the audience url and the grant_type ‘client credentials’. This grant_type is required for Auth0 in order to have access to the create:users scope. Submitting this request will return the credentials for the client to send requests, like creating and deleting users, to the Auth0 Management API.

POST /login
This request requires a body of data from the user, formated as JSON(application/json). The user must provide the following format of data in the body of the request: {“username”: “[string]”, “password”: “[string]”}. The POST does not require any addition information. Submitting this request will login the

credentials provided in the Auth0 API, and will return the OpenID credentials for the user if their username and login are valid.

Postman tests: User Authorization 1, User Authorization Other, User Authorization 2

ERRORS
This request can result in 500 errors if the username does not exist in the API, and/or if the password is incorrect for the username. This request can also result in a 406 Not Accepted if the body data provided is not application/json.

Sources:
Kraft, Riley (10/21/2018). *Assignment-3*. Oregon State University, CS-493-400 Fall 2018.

Kraft, Riley (10/28/2018). *Assignment-4*. Oregon State University, CS-493-400 Fall 2018.

Kraft, Riley (11/04/2018). *Assignment-5*. Oregon State University, CS-493-400 Fall 2018.

Kraft, Riley (11/18/2018). *Assignment-7*. Oregon State University, CS-493-400 Fall 2018.