Chris Shelton

11/26/17

CS 496

Final Project Cloud Only Report

Intro:

So I've created a RESTful API for my cloud only final project that has 2 entities with a relationship between them. The first entity is a user entity that holds 5 attributes including an ID and a "user ID" acquired from the user's Google Plus account. The second is a book entity that holds 6 attributes including an ID and has a relationship to the user account through the "user ID" we got earlier. This way, any one book can only belong to one user in particular. Not only that, it's near impossible for one user to access another user's books through this API as that would require knowledge of the "Token" each user has to authorize any changes or GETs to the API. Speaking of this "Token", each user will be provided a "Token" to use as authorization to use this API through Google. Using Oauth 2.0 to get to Google Plus, we can acquire a "Token" we can then use that's unique to each user that allows them to make edits to their profile and only their profile. The only hitch to this is that it requires the user have a valid Google Plus account or else the program won't work. We will be reusing some aspects of our Oauth 2.0 code for this project as well. Hopefully that's not a concern.

Working Project:

https://assignment3-496.appspot.com/

Entities:

USER:

- ID (auto generated)
- user ID (acquired from Google Plus account)
- fname (must be entered in a post as a string as JSON data called "first")
- lname (must be entered in a post as a string as JSON data called "last")
- email (must be entered in a post as a string as JSON data called "email")

USER:

- ID (auto generated)
- user ID (acquired from Google Plus, relates to the user who "owns" this book)
- title (must be entered in a post as a string as JSON data called "title")
- author (must be entered in a post as a string as JSON data called "author")
- review (must be entered in a post as a string as JSON data called "review")
- yearPub (must be entered in a post as an interger as JSON data called "published")

Verbs:

To use any of these, the user will need to enter their “Token” as an “Authorization” in a header. This verifies the user for each of these and accesses their information accordingly.

{{bookID}} refers to the ID associated with a particular book. This is necessary for edits and deletes to a book and for a GET for a single, particular book.

POST /user
Using the authorization token we talked about earlier, this allows the user to POST a user entity to the API. Users are restricted to 1 user entity per user. This will throw an error if the user already exists or if the authorization is incorrect. Users will need to enter a “first”, “last”, and “email” category as JSON data for this to work.

GET /user
This will get the user entity data from the API and display it to the user whose authorization is used. It will display all attributes of a user entity as JSON data. This will throw an error if the authorization is incorrect or if the user doesn’t exist in the API yet.

PATCH /user
This will edit 1 attribute of the user entity that makes the request. Only “fname”, “lname”, or “email” can be patched by entering “first”, “last”, or “email” respectively. Any attempt to edit more than 1 will throw an error. An error will arise if the user doesn’t exist too.

PUT /user
This will edit all 3 editable attributes of a user entity by entering “first”, “last”, and “email” as JSON data. Attempting to edit any other number of attributes will throw an error. Errors will also arise if the user doesn’t exist.

DELETE /user
This will delete the user entity for the user whose authorization is entered. This will also delete all associated books with that user (if any). No JSON data is needed. This will have an error if the user doesn’t exist.

POST /bookadd
This will add a book to the list of books a user has based on whose authorization is used. The user will need to enter “author”, “title”, “review”, and “published” information as JSON data. The “published” data needs to be an integer, the rest need to be strings. This will throw an error if the user doesn’t exist or if they input the wrong information

GET /bookadd
This will get a list of JSON objects of every book the requesting user has in this API. This will throw an error if they have no books or if they don’t have a user entity.

GET /bookedit/{{bookID}}
This will get the information of a particular book for the user based on the {{bookID}}. It will return the information as JSON data. This will error if you enter the wrong {{bookID}} or if the user doesn’t exist.

PATCH /bookedit/{{bookID}}
This will allow the owner of a book to edit one particular attribute of a particular book. Information must be sent as “title”, “author”, “published”, or “review” depending on what attribute they wish to edit. This will error if they attempt to edit more than 1 attribute, if the book doesn’t exist, or if the user doesn’t exist.

PUT /bookedit/{{bookID}}
This will allow the owner of a book to edit all 4 editable attributes of a particular book. Information must be sent as “title”, “author”, “published”, and “review” depending on what attribute they wish to edit. This will error if they attempt to edit anything other than all 4 attributes, if the book doesn’t exist, or if the user doesn’t exist.

DELETE /bookedit/{{bookID}}
This will allow a user to delete a particular book from the API. This will error if the user or the book doesn’t exist.