Dennis Tat
CS496 Assignment 3 Part 2

Project: http://cs496-1373.appspot.com/
Explore API: https://cs496-1373.appspot.com/_ah/api/explorer
Click **Services**. Select **mylibrary API v1 CS496**. Select desired function. Input appropriate parameters. Click **Execute without OAuth**.
Postman: Sample calls also made with Postman Chrome app.
Most of the code pertaining to the API is located in the **models.py** file.

Tests:

	Test Case	Expected Result	Result
1	Add new location (POST)	New location added	pass
2	Edit location: change phone number (PUT)	Only phone number changed	pass
3	Get location (GET)	Returns information on location	pass
4	Delete location (DELETE)	Location is deleted	pass
5	Add new item. Associate item with location (POST)	New item added	pass
6	Edit item: change availability, release date (PUT)	Correct properties changed	pass
7	Get item (GET)	Returns information on item	pass
8	Delete item (DELETE)	Item is deleted	pass
9	Delete location associated w/ item	Location deleted. Item references deleted key	pass
10	Add new location with no parameters	Error: location not added	pass
11	Add new item with no parameters	Error: item not added	pass
12	Edit location with invalid ID	Error: location not found	pass
13	Edit item with invalid ID	Error: item not found	pass
14	Get location with invalid ID	Error: location not found	pass
15	Get item with invalid ID	Error: item not found	pass
16	Edit location with no parameters (PUT)	No changes made	pass
17	Edit item with no parameters (PUT)	No changes made	pass

A set of example calls I made and their responses are available to view in the file **tests.txt**.

All my tests passed with the expected results. There is a special case above for my API: deleting a location. I normally would not make this call available to the user because items reference locations. By deleting a location, the item would reference a key that doesn't exist. This is an example of how a non-relational database can be flexible, but also contains inconsistent data. To be fully consistent, I would create a function that checks through all the items and deletes the location key for any item that references that location, and then finally delete the location. Doing this for every location deletion would cost considerable system resources, especially if the database was very large.

A better, more economical method would for me to create a function that runs periodically, like once a day/week/month, depending on the need. This function would go through every item and delete the location reference if it is referencing a deleted key. While this operation could still be expensive, it would only run periodically instead of with every delete call. In the time between when this function would run, it is possible for there to be inconsistent data in the database. Deleting an item is no problem because I currently do not have the locations reference the items they contain.

URL structure:

The URL structure used to access resources is (see tests.txt to see examples with parameters):
[HTTP verb] https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/[path]
POST https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/newlocation{message body}
GET https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/locationmodel/[id]
PUT https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/locationmodel/[id]{message body}
DELETE https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/locationmodel/[id]
POST https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/newitem{message body}
PUT https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/itemmodel/[id]{message body}
GET https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/itemmodel/[id]
DELETE https://cs496-1373.appspot.com/_ah/api/mylibrary/v1/itemmodel/[id]

RESTful constraints:

The RESTful constraints are: client-server, stateless, cacheable, layered system, uniform interface, and code on demand. My API meets the constraints: client-server, stateless, and uniform interface.

When using my API, the browser is the client requesting the information, and the server responds to the client with the data. There is a separation of client and server, so this satisfies the client-server constraint. The server does not remember the client's activities and interactions. With each request, the server only responds with the most up-to-date information. This satisfies the stateless constraint. When the client is attempting to interact with the API via HTTP requests, the server does not have different requirements depending on what the client's system is. This satisfies the uniform interface constraint.

My API does not meet the constraints: cacheable, layered system, and code on demand. My API does explicitly state if data is cacheable or not. Each call makes a request to the server to retrieve the information. This does not satisfy the cacheable constraint. When using the API to GET information about an item, the response contains the key of the location it is associated with. However, the user may not know how to use this key to get information on this location. This does not meet the layered system constraint. To meet this constraint, the API could return a link for another GET request to the referenced location key. My API does not execute code as a response to a request, so it also does not meet the optional constraint of code on demand.

Changes to schema:

I made some small changes to the schema in the interest of development time. In my original schema, the item model had 6 properties – one of them being an image. I decided to not include the image property so I could focus on the other parts of the assignment and still meet the requirements.

Changes I would make:

If I had more time, I would like to make my API more robust. I would include additional properties to make the API more useful, such as image uploads or GPS coordinates for a location. I would also include more API calls, such as GET requests that would return lists of all items or locations, or GET requests that apply filters to items. Lastly, I would also include a function (not an API call) that would periodically run to clean up the referenced deleted keys.