CS496 Assignment 3 part 2  
Eric Rouse

## URL Structure:

The url structure of this site is simple and is based on <http://rothbowl.herokuapp.com/api>. A call to this base folder is interpreted as a GET (by browser default) and lists all elements (and their properties) in all databases in order. Further refinement is possible, by appending “/scan”, “/player”, “/bowl”, “/count” or “/path” to this base URL. Those will:

* Scan – list all the bowl objects in the bowl\_list set
  + Ex: <http://rothbowl.herokuapp.com/api/scan>
* Player – list all the players in the player hash
  + Appending a player id lists only that players attributes
  + Ex: <http://rothbowl.herokuapp.com/api/player>
  + Ex: <http://rothbowl.herokuapp.com/api/player/0>
* Bowl – list all the bowls in the bowl has
  + Appending a bowl id lists only that bowls attributes
  + Ex: <http://rothbowl.herokuapp.com/api/bowl>
  + Ex: <http://rothbowl.herokuapp.com/api/bowl/0>
* Count – shows the current count of bowls and players
  + Ex: <http://rothbowl.herokuapp.com/api/count>
* Path –shows the current path, this was used for debug
  + Appending any value will include these values in the returned path
  + Ex: <http://rothbowl.herokuapp.com/api/path>
  + Ex: <http://rothbowl.herokuapp.com/api/path/count/ex/12345>

That concludes the GET commands, the rest of the commands are based on POST; they are post, put, patch and delete. To use these one must submit a POST request to the base url of <http://rothbowl.herokuapp.com/api/> with an identification in the POST data that specifies the type of POST (Ex: put=True, patch=True, or delete=True, any other type will be interpreted as post). For example: curl --data "put=True&player-put=True&put-id=12&fname=put&lname=test" rothbowl.herokuapp.com/api will put a new player at id 12 with the name “put test”. This player would then be located at <http://rothbowl.herokuapp.com/api/player/12>.

## RESTful:

My app is a little restful in that it applies HATEOAS, as interaction occurs through hypertext provided dynamically by the application server. The verbs delete, put and patch are applied to resources as instructed by the server. The browser doesn’t know in advance what to do with the information, the server decides based on the verbs.

Also, it observes the client-server model and it is stateless, no client context is stored on the server between requests.

But, in most other ways my app is not RESTful at all. I’m not observing the cacheable requirement at all, and I’m not applying layering either. The api only works directly, no intermediary is implemented.

## Schema Changes:

I added a list of all bowls so the created and deleted bowls could be associated with something.

## What would I do differently:

So many things. I started this knowing nothing about building web APIs and knowing even less about Redis. One of my first items of business was getting familiar with the Redis calls and data types and then getting familiar with calling those from python. Not to mention getting the database set up in the first place.

Deploying to heroku and testing takes time, if I were to do it again I’d make a carte blanche redis test resource so I didn’t have to keep making minor code changes, uploading them and testing them only to find I’d made a minor mistake.

I’d also build my GETs first so I could see the data as it is generated more easily (I figured this out about 1/3 through).

I might build a POST form rather than use curl, although curl is easier to automate.

## Tests and Results:

The tests are contained in the test\_set.sh file, the results are stored in test\_results.txt. They are a series of curl calls that

1. Tests the GET “all” feature
2. Test the GET “all players” and “single player” feature
3. Test the GET “all bowls” and “single bowl” feature
4. Test the POST with no data (no response expected)
5. Test the POST method of adding a player
6. Test the POST method of adding a bowl
   1. Added bowl is automatically associated with the bowl list
7. Test the POST method of putting a player at a specified resource location
8. Test the POST method of putting a bowl at a specified resource location
9. Test the POST method of patching a player
10. Test the POST method of patching a bowl
11. Test the POST method of deleting a player
12. Test the GET method of showing the bowl list
13. Test the POST method of deleting a bowl
14. Test the GET method of showing the bowl list, less the deleted bowl