http://cs496-hw03-api.appspot.com/

Assignment 3: Part 2

Sam Quinn
CS496
10/23/2015

URL Structure:

For my API I need to allow people to create, modify, and delete two different datastore objects People and Public keys. I wanted to take advantage of the different types of calls and allow my API interact with minimal url changes. My API as is will if not presented with anything return every person or public key object. If you append a key to the URL it will show you only that object. If there is data in the form of a post the API will automatically assume that you are creating a new object. If there is a ‘id’ post object then the API will attempt to modify the existing object. Appending a fullname will act as a search function for the database. The only function that needs a different URL is to delete an object, which /del/ is appended before the key.

/People		Returns every “people” object stored
	/ [id]	Returns the object with id
	/ [full-name]	Returns the objects with full-name
	/ [data]	Adds new person object
	/del/ [id]	Deletes the person object with id
/Public		Returns every public_key object
	/ [id]	Returns the public_key with the id
	/ [full-name]	Returns the public_key with the full-name
	/ [data]	Adds a new public_key
	/del/ [id]	Deletes the public_key with id

Restful:

- **Client-server**
  My API by design takes advantage of the client server interaction model where the information is requested by the client and delivered by the server. This is one of the most fundamental constraints within a RESTful API design. The Client is never aware of how the server is working the server functions independently.
- **Stateless**
  The server never needs to keep track of the client state. The state of my API interactions do not require a state to be kept so I fail this constraint. When I add user authentication to allow users to login I will need to implement some sort of stateless state keeping. Stateless states are transferred between server and client with the data returned. If a user has been authenticated it will receive a token that the API will look for in the future interactions.
- **Cached**
  My API itself does not take advantage of caches, however, the datastore that I have chosen GAE NDB caches the data. Cached data allows the server to respond quicker to requests that have a tendency to be requested a lot. As of now the data returned is non-cacheable since things can change.
- **Layered**
  Google's App Engine does take advantage of the concept of a layered system. When data is requested in a high volume the main datastore can on the fly load balance the request to other databases to ensure that the request is handled in a timely manner.
- **Uniform Interface**
  My interface has an URI associated with each python class, People and Public Keys. Each URI has a uniform design as listed in the table above. Currently my API only returns JSON.

Changes made form schema:

Because I had planned my database structure out so thoroughly last week there were no changes made to my schema. Everything transitioned into an API quite nicely.