Demo: <http://web.engr.oregonstate.edu/~jurczakn/cs496final.MP4>

API: http://cs-496-assignment-3.appspot.com

This is the API I designed in week 4 of the class.

The main page simply provides a list of resources available, and the base url for each.

Each resource has their base url. These are:

School: http://cs-496-assignment-3.appspot.com/school

Teacher: http://cs-496-assignment-3.appspot.com/teacher

Student: http://cs-496-assignment-3.appspot.com/student

Course: http://cs-496-assignment-3.appspot.com/course

Question: http://cs-496-assignment-3.appspot.com/question

From each, you can send a GET request to receive a list of all of the resources in the database.

From http://cs-496-assignment-3.appspot.com/school you can also POST to create a new school. The name must be included.

Using the id of a school, http://cs-496-assignment-3.appspot.com/school/<school\_id> you can send a GET request to receive that specific school, a PUT request to update the school, or send a DELETE request to delete the school.

Using a schools id you can access any of its resources using the following urls:

Teacher: http://cs-496-assignment-3.appspot.com/school/<school\_id>/teacher

Student: http://cs-496-assignment-3.appspot.com/school/<school\_id>/student

Course: http://cs-496-assignment-3.appspot.com/school/<school\_id>/course

From any of those urls, you can send a GET request to receive a list of all the resources that belong to that school, or send a POST request to add a resource to that school, with the following attributes.

Teacher: fname(required), lname(required), username(required), password(required)

Student: fname(required), lname(required), username(required), password(required), courses(list of keys), grade

Course: name(required)

Individual resources can also be accessed using their id through the following urls:

Teacher: http://cs-496-assignment-3.appspot.com/school/<school\_id>/teacher/<teacher\_id>

Student: http://cs-496-assignment-3.appspot.com/school/<school\_id>/student/<student\_id>

Course: http://cs-496-assignment-3.appspot.com/school/<school\_id>/course/<course\_id>

When accessing a specific resource through the school, you can send a GET request to receive the resource, a PUT request to update the resource, or a DELETE request to delete it.

The question resource can be accessed through the school and the course by using the following url:

Question: http://cs-496-assignment-3.appspot.com/school/<school\_id>/course/<course\_id>/question

From there you can send a GET request to get a list of question for the course, or send a POST request to add question. It requires a body and answer.

An individual question can be accessed using its id with the following url:

http://cs-496-assignment-3.appspot.com/school/<school\_id>/course/<course\_id>/question/<question\_id>

When accessing an individual question, you can send a GET request to receive the question, send a PUT request to update the question, or send a DELETE request to delete the question.

To associate a student with a course, you must access the individual student, and then add the course to the end of the url:

http://cs-496-assignment-3.appspot.com/school/<school\_id>/student/<student\_id>/course/<course\_id>

You must do this using a PUT request, and if the course is deleted after, the student will have the course deleted from there course list attribute, and the delete request will return the students username.

All school names, student’s usernames, and teacher’s usernames must be unique. Course names must be unique to the school they belong to, and question bodies must be unique to the course they belong to.

The API is a resource based API. It uses all four HTTP verbs, like a RESTful API. It also follows the url naming standard. It does not create urls, and it does not use POST and PUT properly. POST should be used for all request that should be repeated, and PUT for those that should not. I only used POST to create resources and PUT to update them. I did not do anything with cache time, which is also required in a RESTful API, and it only works with JSON output.

I was able to keep the schema from my API last week intact. I think that the strongly hierarchical system I devised was not the best. It became difficult to access individual resources. Especially with the questions. You must keep track of the school’s id, the course’s id, and the question’s id just to access the question, and make sure that the url is formatted properly with all the ids in the correct places. It may have been easier to just make questions part of the course, since it is storing so little information already. But with it the way it is, it may make things easier if I want to add more features to the question entity.

I was able to use more functions this time for writing responses. This will make it much easier if in the future I want to include more than just JSON. I still think there is more I could have done to modularize my code. Most of the resources work exactly the same, except taking in different variable. I think it would have been possible to create some template functions to take care of this.

Navigating the api, if feel, is actually quite difficult. As I said before, this is because it involves keeping track of multiple ids and organizing them. I read about how some api’s combine the parent and child ids for the child and then parse through them when they are used. This would have taken more work, but I feel that it would have improved immensely with the usability of the api.

# Login:

I implemented the login system myself. It is very simple. The passwords and username are stored as simple string as attributes of the users, either teachers or students. It is important to know which the user is logging in with, as the front-end app treats each differently. Each teacher must have unique username to other teachers, even of different schools. The same goes for students, but teachers and students are treated different, so a teacher may have the same username as a student. Teachers can only access entities from the same school, and the same goes for students.