Aaron Schwartz-Messing
Noah Potash

We both came with our own system of URI, but when we teamed up we decided on to use Aaron's URI. Noah did all the research and syntax for the headers. The typing and editing was split between us. We both gave input on each other's work and corrected mistakes.

Distributed Systems Hw2

Headers

For both Request and Response

Content-type: application/json← **Only if there is a json payload**
Content-encoding: gzip ← **Only if there is a json payload**
Cache-control: no-cache, no-store, must-revalidate.
Date: <three_letter_day_of_week>, <day_of_month><three_letter_month> <year> <hours_military>:<minutes>:<seconds> GMT
Ex. Date: Wed, 21 Oct 2015 21:28:00 GMT

For Request Only

<HTTP request> <URL> HTTP/1.1 - GET/POST/PUT/DELETE followed by URL and HTTP version
Host: http://www.registrar.yu.edu
Accept: application/json ← **Only if function is GET**, because only GET expects response payload.
Accept-Language: en-US,en;q=0.5 ← **Only if function is GET**
Accept-Encoding: gzip ← **Only if function is GET**

For Response Only

HTTP/1.1 <return_code> <reason-phrase>
Transfer-encoding: gzip ← **Only if request function is GET**

Return-Code

- Any function request that is executed successfully by the server gets a return code of 200 (OK). If it was a PUT it gets a 201 (Created).
- Any GET request for which the URI is valid but there is no information in the database to return, no payload is returned, the return code is 204 (No Content).
- Any URI not discussed in this document (including any URI for which the ID was left out) receives a return code of 400 (Bad Request) in the response header. No changes are made and no information is sent from the server to the client.
- If the value entered for a parameter violates the constraints of that field in the database (providing a varchar for an int, too many characters in a varchar, etc.) then nothing happens and the return code is 400 (Bad Request).
- Whenever the client does not have the requisite authorization to execute whichever function that he is trying to execute, the return code is 401 (Unauthorized).
- When an ID parameter in a URI does not correspond to an object in the database, no payload is returned, and the return code is 404 (Not Found).
- Other information about return codes will be discussed per function below.

General Information

Permissions - “<ID>/URI”

Before every URI the requester has to provide his “ID”. This is unique for each person. The server should be able to check the permissions for this user based on his ID. How this ID is assigned to the users and keeping track of the permissions of each user is the server’s job.

Json Collections

Here is the format for Json collections:

```
{
        “Majors” : [
                {
                        MAJOR #1
                },
                {
                        MAJOR #2
                },
                …
                {
                        MAJOR #n
                }
```

```
        ]
    }
```

We used “Majors” here as just an example. Any time that we provide a collection of information it will use this format. The json for each piece of information will be specified below.

Functions & URIs

Majors/Courses/Classes

URIs

GET/POST/PUT/DELETE <ID>/MAJORS/<major_id>
GET/POST/PUT/DELETE <ID>/COURSES/<course_id>
GET/POST/PUT/DELETE <ID>/CLASSES/<class_id>

GET

Returns the requested information in Json format, specified below.

If the parameter is left out, the server returns a Json payload containing a collection of every Major/Course/Class respectively.

Anyone with a valid ID can request a GET.

DELETE

If the parameter is left out, all of the Majors/Courses/Classes are deleted.
Only Professors and Deans can use DELETE, POST, and PUT.

POST

Json: the client must include valid Json (shown below). Not all fields must be included.

If the Json includes fields that do not exist in the database, response code is 400 (Bad Request).

If the Json is empty, return code is 400 (Bad Request).
Only Professors and Deans can use DELETE, POST, and PUT.

PUT

Parameter: The id can either be included in the URI, or can be included in the json and left out of the URI. If it is supplied neither in the URI nor in the Json, the return code in the response header is 400 (Bad Request).

Json: the client must include valid Json (shown below).

Fields: The following fields must be included in the Json of a PUT:

- Major
- Course
    - Prerequisites
    - Title
    - Credits
    - Description
- Class
    - Semester
    - Time
    - Day
    - Location
    - Professor
    - Maximum number of students

If any of these fields are left out, nothing is created, and the return code is 403 (Forbidden).

If the Json includes fields that do not exist in the database, response code is 400 (Bad Request).

Only Professors and Deans can use DELETE, POST, and PUT.

Json

Major

{
“shortname”:VARCHAR(3) <shortname>,
“fullname”:VARCHAR(45) <fullname>,
“school_shortname”:VARCHAR(5) <school>,
“approved”: VARCHAR(1) <T(for true) or F (for false)>
}

Course

{
“Id”: INT <course-id>,
“School_shortname”: VARCHAR(5) <school>,
“description”: VARCHAR(255) <description>,
“name”: VARCHAR(45) <name>,

“Credits”: INT<number_of_credits>,
“Approved”: VARCHAR(1) <T(for true) or F (for false)>,
“Prerequisites”: INT [INT id1, INT id2… INT idn]
}

Class

{
“Id” : INT <id>
“Course_id”: INT <id>,
“Semester”: VARCHAR(1) <semester>,
“Year”: INT <year>,
“Time”: INT <time>,
“Day”: INT <day>,
“Location”: VARCHAR(45) <location>,
“Professor”: VARCHAR(45) <professor>,
“Max_students”: INT <max_students>,
“Current_number_of_students”: INT <number of students currently in the class)
}

Grades

Access Grades by Class

GET

Specific Class

All Grades

GET <ID>/GRADES/CLASSES/<class_ID>

Json response payload:
{
“student_id”: INT <id>,
“Grade”: VARCHAR(2) <grade>
}
You would get a Json collection of these.
Permission: All non-student users

Grade for Specific Student in Class

GET <ID>/GRADES/CLASSES/<class_ID>/<student ID>

Json response payload:

{

“Grade”: VARCHAR(2)<grade>

}

Permissions: Student himself and all non-student users

Mode Grade of Class

GET <ID>/GRADES/CLASSES/<class_ID>/MODE

Json response payload:

{

“Mode”: VARCHAR(2)<mode_grade>

}

Permissions: Deans and Department Chairs

Median Grade of Class

GET <ID>/GRADES/CLASSES/<class_ID>/MEDIAN

Json response payload:

{

“Median”: VARCHAR(2)<median_grade>

}

Permissions: Deans and Department Chairs

Frequency of Specific Grade in the Class

GET <ID>/GRADES/CLASSES/<class_ID>/AGGREGATE/<grade>

Json response payload:

{

“Frequency”: INT <how many of this grade>

}

Permissions: Deans and Department Chairs

Aggregate - Frequency of All grades in Class

GET <ID>/GRADES/CLASSES/<class_ID>/AGGREGATE

Json response payload:

{

“Aggregate”: [

{

“Grade”: VARCHAR(2)<grade>,

"Frequency": INT <how many of this grade>,
"Percentage": INT <% of student in question>
},
{
"Grade": VARCHAR(2)<grade>,
"Frequency": INT <how many of this grade>,
"Percentage": INT <% of student in question>
}
...etc. For every letter grade that was given in this class. Does not include grades that nobody got (i.e. the frequency will never be zero).
]
}

Permissions: Deans and Department Chairs

Statistics

GET <ID>/GRADES/STATISTICS/<first_course_ID>/<first_professor_ID>/<second_course_ID>/<second_professor_ID>

Json for this query is the same as the Json for the Aggregate query directly above.

This returns an aggregate of the grades of students taking the second course with the second professor, who had already taken the first course with the first professor.

If parameter is left out, no information payload is returned, and the return code is 400 (Bad Request).

Permissions: Deans and Department Chairs

For All Classes

The following URI are identical to the previous ones, except that they request the information for all of the classes, stored in a Json collection. Each object in the collection will have a new field "Class_offering_id": INT <id>.

GET <ID>/GRADES/CLASSES

GET <ID>/GRADES/CLASSES/MODE
GET <ID>/GRADES/CLASSES/MEDIAN
GET <ID>/GRADES/CLASSES/AGGREGATE

DELETE

DELETE <ID>/GRADES/CLASSES/<class ID>/<student ID>
If you leave out <student_ID> you delete all of this guy's grades.
If you leave out both you delete all of the grade for the whole class.

POST/PUT

POST <ID>/GRADES/CLASSES/<class_ID>/<student ID>/<grade>
The grade parameter may not be left out. Return Code 400 (Bad Request) if left out.
PUT is not a supported operation for posting grades. Return Code 400 (Bad Request).

Access Grades by Student

GET all grades for all students

GET <ID>/GRADES/STUDENTS
Json response payload:

```
{
    "Student_IDs": INT <id>,
    "Classes": [
        {
            "Class_offering_id": INT <ID>,
            "Grade": VARCHAR(2) <grade>
        },
        {
            "Class_offering_id": INT <ID>,
            "Grade": VARCHAR(2) <grade>
        }... etc.
    ]
}
```

Collection of these for all students.

GET all grades for particular student

GET <ID>/GRADES/STUDENTS/<student_id>
Json response payload:

```
{
    "Class_offering_id": INT <id>,
    "Grade": VARCHAR(2) <grade>
```

}
Collection of these for all classes.

Get grade for particular student for particular class

GET <ID>/GRADES/STUDENTS/<student_id>/<class_id>
Json response payload:
{
"Grade": VARCHAR(2) <grade>
}

Viewing Student's Courses

GET <ID>/STUDENTS/<student_id>/COURSES
Json Response payload: Same syntax as the list of courses in the Majors/Courses/Classes section. Here's a link. Contains a list of all of the student's past and current courses.

Clearing Students to Register

POST <ID>/CLEAR/<student_ID>
If the student_ID is left out, clears all students to register.

Permissions: only academic advisors may clear a student for registration.

Registering for Classes

PUT <ID>/STUDENTS/<student_id>/REGISTER/<class_offering_ID>
If any of the parameters are left out, nothing is done, return code is 400 (Bad Request).
If the student is not cleared to register, nothing is done, return code is 403 (Forbidden).
If there are not open spots in the class, nothing is done, return code is 403 (Forbidden).

Permissions: only the student can register himself for classes.