Zachary DeVita
CS 496
Assignment#3

URL: http://www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com

I have done my assignment 3 using a DynamoDB, non-relational database. My backend server side programming is done in Node.js. Both my database and api are being hosted by Amazon aws platform. For testing the api, I have chosen to use the Postman browser app. I was able to accomplish all 4 styles of queries with my api, and I will list the URL with parameters and results next.

GET requests

To show all students in the database.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/student

Request Header:

```
GET /student HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: localhost:8081
X-Amz-Date: 20161020T032312Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
Postman-Token: a2395e0d-2a88-46ef-2d45-9ec8f9248abf
```

Result:


```
{
  "Items": [
    {
      "course": {
        "L": [
          {
            "S": "ece375"
          },
          {
            "S": "cs344"
          },
          {
            "S": "cs381"
          },
          {
            "S": "cs241"
          }
        ]
      },
```

```
  "name": {
    "S": "Lashawante Ehlers"
  },
  "student_id": {
    "N": "3"
  }
 },
 {
  "course": {
    "L": [
      {
        "S": "cs321"
      },
      {
        "S": "cs344"
      },
      {
        "S": "cs375"
      },
      {
        "S": "cs496"
      }
    ]
  },
  "name": {
    "S": "Tyrone Hartzell"
  },
  "student_id": {
    "N": "2"
  }
 },
 {
  "course": {
    "L": [
      {
        "S": "cs361"
      },
      {
        "S": "cs161"
      },
      {
        "S": "ece275"
      },
      {
        "S": "cs241"
      }
    ]
  },
```

```
    "name": {
     "S": "Xavier Jackson"
    },
    "student_id": {
     "N": "4"
    }
   },
   {
    "course": {
     "L": [
      {
       "S": "cs496"
      },
      {
       "S": "cs162"
      },
      {
       "S": "cs344"
      },
      {
       "S": "cs361"
      }
     ]
    },
    "name": {
     "S": "Jamal Ingraham"
    },
    "student_id": {
     "N": "1"
    }
   }
  ],
  "Count": 4,
  "ScannedCount": 4
}
```


To show all courses in the database.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/class

Request Header:

```
GET /class HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: localhost:8081
X-Amz-Date: 20161020T032312Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-
2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date,
Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
```

Postman-Token: 84298a19-6bf3-27a9-ad4b-5fd0358a4afc

Result:


```
{
  "Items": [
    {
      "course": {
        "S": "ece275"
      },
      "instructor": {
        "S": "Graig Geib"
      }
    },
    {
      "course": {
        "S": "cs241"
      },
      "instructor": {
        "S": "Roman Owen"
      }
    },
    {
      "course": {
        "S": "cs344"
      },
      "instructor": {
        "S": "Fallon Hartzell"
      }
    },
    {
      "course": {
        "S": "cs162"
      },
      "instructor": {
        "S": "Joe Paris"
      }
    },
    {
      "instructor": {
        "S": "Billy Schramm"
      },
      "course": {
        "S": "cs375"
      }
    },
    {
      "course": {
        "S": "cs361"
```

```
    },
    "instructor": {
      "S": "Josh Bourne"
    }
   },
   {
    "course": {
      "S": "cs161"
    },
    "instructor": {
      "S": "Dodi Coreson"
    }
   },
   {
    "course": {
      "S": "cs381"
    },
    "instructor": {
      "S": "Terrell Calo"
    }
   },
   {
    "course": {
      "S": "cs321"
    },
    "instructor": {
      "S": "Raymond Kuiper"
    }
   },
   {
    "course": {
      "S": "cs496"
    },
    "instructor": {
      "S": "Dwain Chery"
    }
   }
  ],
  "Count": 10,
  "ScannedCount": 10
}
```


To show individual student using their ID number.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/student-lookup?student_id=3

Request Header:

```
GET /student-lookup?student_id=3 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
```

Host: localhost:8081
X-Amz-Date: 20161020T032312Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
Postman-Token: 37e737bf-fe38-4190-ecd2-10f4d4a06c6f

Result:


```
{
  "Item": {
    "course": {
      "L": [
        {
          "S": "ece375"
        },
        {
          "S": "cs344"
        },
        {
          "S": "cs381"
        },
        {
          "S": "cs241"
        }
      ]
    },
    "name": {
      "S": "Lashawante Ehlers"
    },
    "student_id": {
      "N": "3"
    }
  }
}
```


PUT request

Adds a new student to the database. The new student's name is Christopher Douglas and their ID number is 5.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/add-student?student_id=5&name=Christopher Douglas

Request Header:

PUT /add-student?student_id=5&name=Christopher Douglas HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
Host: localhost:8081
X-Amz-Date: 20161020T032312Z

Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
Postman-Token: bb2c27a8-2cc6-7942-e70d-666283fca328

Result: No results are returned from this action, but I will display the results of querying all students to show that Christopher Douglas is indeed in the database.


```
{
  "Items": [
    {
      "course": {
        "L": [
          {
            "S": "ece375"
          },
          {
            "S": "cs344"
          },
          {
            "S": "cs381"
          },
          {
            "S": "cs241"
          }
        ]
      },
      "name": {
        "S": "Lashawante Ehlers"
      },
      "student_id": {
        "N": "3"
      }
    },
    {
      "course": {
        "L": [
          {
            "S": "cs321"
          },
          {
            "S": "cs344"
          },
          {
            "S": "cs375"
          },
          {
```

```
        "S": "cs496"
       }
      ]
     },
     "name": {
      "S": "Tyrone Hartzell"
     },
     "student_id": {
      "N": "2"
     }
    },
    {
     "course": {
      "L": [
       {
        "S": "cs361"
       },
       {
        "S": "cs161"
       },
       {
        "S": "ece275"
       },
       {
        "S": "cs241"
       }
      ]
     },
     "name": {
      "S": "Xavier Jackson"
     },
     "student_id": {
      "N": "4"
     }
    },
    {
     "course": {
      "L": [
       {
        "S": "cs496"
       },
       {
        "S": "cs162"
       },
       {
        "S": "cs344"
       },
       {
```

```
          "S": "cs361"
        }
       ]
     },
     "name": {
      "S": "Jamal Ingraham"
     },
     "student_id": {
      "N": "1"
     }
    },
    {
     "name": {
      "S": "Christopher Douglas"
     },
     "student_id": {
      "N": "5"
     }
    }
  ],
  "Count": 5,
  "ScannedCount": 5
}
```


POST request

Modifies an existing students information. The student with ID=5 will have their name changed from Christopher Douglas to Deshawn Evans.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/update-student?student_id=5&name=Deshawn Evans

Request Header:

```
POST /update-student?student_id=5&amp;name=Deshawn Evans HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
Host: localhost:8081
X-Amz-Date: 20161020T032312Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
Postman-Token: 4651250f-8212-25da-f23d-7d930b6d81e9
```

Result: The only result returned is the change to the record with ID=5.


```
{
  "Attributes": {
    "name": "Deshawn Evans"
  }
}
```

DELETE request

Removes a student from the database. The student with ID=5 will be removed. The name of this student is Deshawn Evans.

URL: www.assignment2-1.z9cqkpjzc2.us-west-2.elasticbeanstalk.com/remove-student?student_id=5

Request Header:

DELETE /remove-student?student_id=5 HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
Host: localhost:8081
X-Amz-Date: 20161020T032312Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIAJMOOOX542KOCUTRA/20161020/us-west-2/DynamoDB/aws4_request, SignedHeaders=content-type;host;x-amz-date, Signature=7eaf09a97fe569c57431dda3502f44e8aa7c3a7a6191c8cd065e7ed33aa7b1cb
Cache-Control: no-cache
Postman-Token: c3c8176d-680b-f423-a776-193a767dfac5

Result: No results are returned from this action, but I will display the results of querying all students to show that Deshawn Evans is no longer in the database.


```
{
  "Items": [
    {
      "course": {
        "L": [
          {
            "S": "ece375"
          },
          {
            "S": "cs344"
          },
          {
            "S": "cs381"
          },
          {
            "S": "cs241"
          }
        ]
      },
      "name": {
        "S": "Lashawante Ehlers"
      },
      "student_id": {
        "N": "3"
      }
    },
    {
      "course": {
        "L": [
```

```
      {
        "S": "cs321"
      },
      {
        "S": "cs344"
      },
      {
        "S": "cs375"
      },
      {
        "S": "cs496"
      }
    ]
  },
  "name": {
    "S": "Tyrone Hartzell"
  },
  "student_id": {
    "N": "2"
  }
},
{
  "course": {
    "L": [
      {
        "S": "cs361"
      },
      {
        "S": "cs161"
      },
      {
        "S": "ece275"
      },
      {
        "S": "cs241"
      }
    ]
  },
  "name": {
    "S": "Xavier Jackson"
  },
  "student_id": {
    "N": "4"
  }
},
{
  "course": {
    "L": [
```

```
        {
          "S": "cs496"
        },
        {
          "S": "cs162"
        },
        {
          "S": "cs344"
        },
        {
          "S": "cs361"
        }
      ]
    },
    "name": {
      "S": "Jamal Ingraham"
    },
    "student_id": {
      "N": "1"
    }
   }
 ],
 "Count": 4,
 "ScannedCount": 4
}
```


Test Cases

/student
URL Path: /student
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /student?student_id=5
Expected: Fail
Result: SUCCESS
Reason: Apparently when no parameters are necessary, than the function simply ignores them

URL Path: /student?i_have_no_parameters
Expected: Fail
Result: SUCCESS
Reason: Apparently when no parameters are necessary, than the function simply ignores them

/class
URL Path: /class
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /class?course=cs321
Expected: Fail
Result: SUCCESS
Reason: Apparently when no parameters are necessary, than the function simply ignores them

URL Path: /class?i_have_no_parameters
Expected: Fail
Result: SUCCESS
Reason: Apparently when no parameters are necessary, than the function simply ignores them

/student-lookup
URL Path: /student-lookup?student_id=3
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /student-lookup?student_id=9
This update is set for a non-existent ID number
Expected: Fail
Result: SUCCESS
This doesn't cause an error but it returns nothing because the ID doesn't exist.

URL Path: /student-lookup
Expected: Fail
Result: Fail
This returns all data from all tables stored in the database when input into a browser which is incorrect, but it completely fails and returns nothing in Postman.

URL Path: /student-lookup?student_id=3&I_am_a_BS_parameter
Expected: Fail
Result: Success
Apparently when extra parameters are included, than the function simply ignores them

/update-student
URL Path: /update-student?student_id=3?student_id=5&name=Deshawn Evans
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /update-student?student_id=3
Expected: Fail
Result: Fail
Cannot GET /update-student?student_id=5

URL Path: /update-student?student_id=9?student_id=5&name=Deshawn Evans
This update is set for a non-existent ID number
Expected: Fail
Result: Fail

Cannot GET /update-student?student_id=9?student_id=5&name=Deshawn%20Evans

/add-student
URL Path: /add-student?student_id=5&name=Christopher Douglas
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /add-student?student_id=5
This adds a student without a name being specified. 'name' is not a key so an ID for a student should be added to the database with no name associated with it.
Expected: SUCCESS
Result: SUCCESS


```
{
    "student_id": {
      "N": "5"
    }
}
```


URL Path: /add-student?name=Christopher Douglas
No ID is included. ID is the key for the table.
Expected: Fail
Result: Fail
Cannot GET /add-student?name=Christopher%20Douglas

URL Path: /add-student?student_id=5&name=Christopher Douglas
A row in student already exists with this ID number. I expect the value to overwrite the existing entry.
Expected: SUCCESS
Result: Fail
Cannot GET /add-student?student_id=5&name=Christopher%20Douglas
I was incorrect in my assumption. An error is returned when there is an already existing value.

/remove-student
URL Path: /remove-student?student_id=5
Expected: SUCCESS
Result: SUCCESS. Details are shown in previous section.

URL Path: /remove-student
Expected: Fail
Result: Fail
Cannot GET /remove-student

URL Path: /remove-student?student_id=9
This is a query to remove an entry in the student database for a non-existent ID number
Expected: Fail
Result: Fail
Cannot GET /remove-student?student_id=9

Explanation of the URL structure used to access resources

I don't know what this requirement is looking for. The protocol and domain name for my URL were chosen for by Amazon AWS. I assume the domain name is combination of an Amazon-specific string concatenated with some sort of a hash value. For my paths, I just chose arbitrary strings that seemed to make sense and were related to the action which occurred when they are triggered. To show all elements of the student table I chose the path to be '/student', but I could have just as easily chosen '/all-students'. I made extensive use of the dash (-) in between words in the path. I believe this increases the readability for the non-technical, opposed to having a '%20' between words. I similarly used the path '/class' to show all elements of the class table.I would think this is not what this question is looking for since I would assume everyone did something of this sort.

Part of the path in a URL is the parameters for the query which are appended to the end of the path. To query an entire table, i.e. 'class' or 'student' table, no parameters are appended to the path. The table name is encoded, I believe, inside the request header by the server, but, since I gave the tables separate paths, there was not a need to include parameters. I should also mention that giving every request in my server a different path made it so that the order of my request functions did not matter.

To query an individual student there was a specific path '/student-lookup' which was necessary for the request to find the appropriate GET request in the server. For this query it is also necessary to include one parameter; a parameter to specify the ID number of the student who is being queried. An example might look like this:

/student-lookup?student_id=3

This specifies the specific row in the 'student' table to return. I could just as easily created a GET request using the students name as the parameter, but it is fairly common for people to have the same name and ID numbers can be controlled so that they do not repeat themselves.

To add a student to the database I created a PUT request with the path being '/add-student'. This request requires two parameter. One parameter must be the ID of the student, and the other would be the student's name. An example where there is a student added to the database who had the ID=5 might look like this.

/add-student?student_id=5&name=Christopher Douglas

This indicates the a new student is to be added to the database with an ID=5, and whose name is Christopher Douglas. This seems like one of the most fundamental actions for a program of this sort to have. Additional attributes can be added using the update function.

To update a student's information I created a POST request with the path being '/update-student'. This request requires more than one parameter. One parameter must be the ID of the student, and the other would be the information that is being changed. An example where there is a change to the student's name who had the ID=5 might look like this.

update-student?student_id=5&name=Deshawn Evans

This specifies the row to be updated and which column is being updated. The only variable which can be changed in my current configuration is the name. This might seem not very useful, but if there were more columns to change than this same code could be used there as well.

To remove a student from the database I created a DELETE request with the path being '/remove-student'. This request only provides a single parameter. Since the student ID is a unique value we can again conduct this action bases solely on the ID parameter. If there were no unique columns, than we would need to use a second parameter so we could search using multiple keys. An example looks like this:

/remove-student?student_id=5

This specifies the row to be removed. This could just as well require a parameter for the student's name, but it would be completely redundant and unnecessary.

Description of which RESTful constraints you met and which you did not (you do not need to make a RESTful app, but you do need to know why it does not meet all the constraints of a RESTful API)

REST, or Representational State Transfer is a style of web architecture and not a protocol. There are many strict regulations and requirements for an api to be "RESTful" so I will be discussing the primary requirements which we were taught in lecture.

As far as the client-server separation, since I am currently sending back data as a response to a client request via jQuery, it would seem that I am setting up a schema where the client is forced to parse the data and display it for the user. This would seem to not completely separate client-server. I don't however know of a way in which the client would not need to parse data. I would assume since the client is not requesting anything themselves from the database, that this does fit the requirement for client-server separation.

My current configuration is stateless. The client's state is not maintained, and there are no sessions. I have simply created a service for them to use, and the client must maintain their own state. Since I have not developed any client side programming, and because of the nature of my api, this seems very natural for the api.

The resources retrieved by the client are not cacheable, but they are marked non-cacheable. There is a reference to the cacheability in the request header. This is another requirement of RESTful architecture which I do seem to include.

Unless one layer counts as a layered system, than I do not have a layered system. I do not have multiple servers; there is just a single access point. My single layer is opaque though, and the client would have no way to know whether there is one or more layers. I calling this requirement met since there is not an explicit requirement to include multiple layers.

I do not allow for the client to request alternate resource identifiers at this point. They are stuck with the pseudo-JSON which is returned via DynamoDB. The messages sent to client are self-descriptive, they include cachability, as well as, a MIME type and the type of request which was made.

My api does not follow HATEOAS, or hypermedia as the engine of application state requirement. If you only enter the entry point of the program, than you will receive a get error. I do not include resources or information on how to navigate to resources when only the program entry point is used in the URL.

Because of not following the HATEOAS requirement, and by not allowing the client to request alternate resource identifiers, my api does not quite fit the requirements for a RESTful architecture. It does however follow more requirements than I had initially anticipated that it would.

Discussion of any changes you needed to make to the schema from last week.

Last week I didn't completely understand what a non-relational database was. I managed to follow most of the schema completely, but the structure of my database changed dramatically. I was a bit ambitious and included several entities in my database schema from last week, but I didn't quite understand the functionality of these databases yet. Because of this I made a complex structure of lists where a single table would have a list which led to other lists. This would be a nightmare for server side coding, which I recently found out, so I got rid of one entity and turned the other two entities into their own separate tables.

Having finished the API, what you would have done differently

Where do I start? I was super ambitious for this project and had tons of ideas for it, but it turned out to be far more difficult to learn DynamoDB than I had ever anticipated. If you don't like Amazon's horrible documentation, than good luck finding resources elsewhere. I originally planned to have my student ID's auto-increment, or at least some equivalent of that. I spent the majority of a day attempting to do this before I moved on.

I wanted to have many more options for the various types of requests, but I ran out of time to get all of them completed. I wanted a query to just return the classes a student was taking, without any other information. I wanted a query to return all students who were enrolled in one specific course. I wanted a query batch add a list of students, one to batch delete a list of students, and one to batch update just the courses a student were taking. I could go on for a while but I think the point is made.

There are many functionalities that this api could have which would improve its usability, but one week to learn a drastically new query language did not allow much time for adding functionality.

I would have also like to add some form of input validation. If the correct path and parameter names are used, but the ID, for example, doesn't exist, I would have liked there to be something other than an error returned.