CMSC B353 Software Engineering - Spring 2021
Programming Assignment #4: RESTful API Development
Due Friday April 9, 5pm EDT

Introduction

Previously we have looked at the design of single pieces of software, in which all components are working together in the same process, but nowadays many software systems are designed to use components that are distributed across a network, sometimes using different technologies or programming languages.

In this assignment, you will implement a distributed system using Java and JavaScript components, similar to what you will later do in your group project. In completing this assignment, you will learn how to:

- Develop a RESTful API using Node Express
- Implement a client-server system using Java web programming
- Develop robust code that tolerates faults in a distributed system

As before, your Java code should follow the code formatting standards and guidelines used in CMSC B206. Although strict adherence will not be checked per se, you may lose points or be asked to resubmit if your code exhibits consistent, egregious violations of these guidelines.

Although this assignment doesn't ask you to write a ton of JavaScript, you will need to use features such as functions, arrays, and objects. **If you have not programmed in JavaScript before, be sure to get an early start on Part 1**, and ask for help if you need it!

Background

After a natural disaster or humanitarian crisis, rescue and relief organizations can use software packages such as Sahana Eden (https://sahanafoundation.org/eden/) to coordinate aid efforts and provide services to those who are affected.

A common feature in these sorts of systems is the ability for friends and family of an affected person to know that person's status, e.g. if they have reported themselves as safe, are known to be hospitalized or deceased, or if someone else has reported them to be missing.

Such systems are often designed in a distributed manner so that information about a person's status can be stored in one piece of software but then read and updated by another. Additionally, data is often stored in multiple places for purposes of redundancy.

In this assignment, you will implement a simple status reporting application as part of a distributed, client-server system using JavaScript and Java web programming.

CMSC B353 Software Engineering - Spring 2021
Programming Assignment #4: RESTful API Development
Due Friday April 9, 5pm EDT

Part 1. Node Express Server

In this part of the assignment, you will use Node Express to implement a RESTful web API that other applications can use to get and set the status of an affected person.

To get started, follow these steps:

1. Download and install Node.js on your computer.
2. Install and create a Node Express app and be sure to accept the default entry point as "index.js".
3. Download the Node Express app starter code and put it in the directory where you installed your Node Express app.
4. Start the Node Express app from your Terminal or Command Prompt by navigating to the directory where you installed it and running the command "`node index.js`" (without the quotes); you should see "Listening on port 3000" printed to the console.
5. Using a web browser such as Google Chrome on the same computer where the Node Express app is running, open the URL http://localhost:3000/test and you should see a JSON object that reads { "message": "It works!" } appear in the browser.

If you run into any problems getting started, please post a **public** note in Piazza so that the instructor and your classmates can try to help you. If you don't get this working, you won't be able to do this assignment!

Once you have the basic app working, take a look at the starter code we provided and note that there is a global variable called "people" which is a Map of IDs to Person objects, where Person is a class defined in this file. A few Person objects have initially been added to the Map. Make sure you understand this code but **do not change it**! You do not need to use a backend database such as MongoDB for this assignment; you can store all Person objects in the "people" Map.

For this part of the assignment, you need to implement a single Node Express endpoint as follows:

Endpoint name: *get*
Query parameter: The only parameter used by this endpoint is the *id* parameter, but the caller may specify more than one ID, in which case the query parameter would be represented as an array in Node Express. For instance, if the caller used the URL http://localhost:3000/get?id=1234 then `req.query.id` would hold the value '1234' in Node Express. However, if the caller used the URL http://localhost:3000/get?id=1234&id=5678 then `req.query.id` would hold the array ['1234' , '5678'].

Hint! You can use the Array.isArray JavaScript function to determine whether a variable is an array or a single value.

Behavior: Return a JSON array containing the objects for each Person in the Map with an ID equal to one of the *id* parameter values.

Example #1 (assuming default Person objects in Map):
http://localhost:3000/get?id=1234 should return
`[{"id":"1234","status":"safe"}]`

Example #2 (assuming default Person objects in Map):
http://localhost:3000/get?id=1234&id=5678 should return
```
[{"id":"1234","status":"safe"},{"id":"5678","status":"missing"}
]
```

Error Handling:
Return an empty JSON array if the *id* parameter is not provided or has no value

Example #3
http://localhost:3000/get?id= should return an empty JSON array, i.e. `[]`

Example #4
http://localhost:3000/get should return an empty JSON array, i.e. `[]`

Also, if there is no Person in the Map with a given *id,* the array should include a JSON object for that *id* with the “id” field set to the specified *id*, and the “status” field set to “unknown” (*hint: look at the starter code we provided to see how to do this!*)

Example #5 (assuming default Person objects in Map):
http://localhost:3000/get?id=0&id=1111 should return
`[{"id":"0","status":"unknown"},{"id":"1111","status":"safe"}]`

You do not need to implement any other endpoints in this Node Express app or worry about any other URL paths besides “/get”.

Advice

We highly recommend using the `console.log(message)` command for debugging: this will write the message to the Terminal or Command Prompt window where your Node Express app is running.

Note that **all JavaScript code must be in index.js** for this assignment; please notify the instructor if you feel that it is necessary to create additional files.

Although you don't *have* to get this part completed before moving on to Part 2, it is highly recommended that you do, so that you can test your Part 2 implementation.

Part 2. Java Client

In this part of the assignment, you will implement a Java client that uses web programming to communicate over the Internet with the server from Part 1 to get the status of one or more people.

Before You Begin

First, download the starter code that you will need for this part of the assignment.

The most challenging part of this assignment is getting your Java program to communicate over the web with the Node Express server. To help you make sure that you have the basic connection working, we have provided a test program called RemoteClientTest as part of the starter code linked above. Note that you will need the JSON.simple library in order to use this code.

To test that you're able to get your Java program to communicate with the Node Express server, start the server using the index.js starter code that we provided for Part 1, or your Part 1 solution, as long as it has a "/test" endpoint defined as follows (this is what was provided in the starter code):

```
app.use('/test', (req, res) => {
        // create a JSON object
        var data = { 'message' : 'It works!' };
        // send it back
        res.json(data);
    });
```

Then, after starting your Node Express server, run the Java RemoteClientTest program that we provided on the same computer where Node Express is running. As you can see in the Java source code, it makes an HTTP Request for http://localhost:3000/test, reads the JSON that is sent back in the Response, reads that object's "message" field, and prints it. If all goes well, you should see "Message: It works!" printed to the console where your Java program is running.

If you do *not* see this, make sure that your Node Express server is running on the same machine as your Java program and is using port 3000, and that you did not change the code for the "/test" endpoint provided in the starter code.

If you still cannot figure out what's happening, please post a **public** note in Piazza so that the instructor and your classmates can try to help you out.

Specification

Once you have the basic Java-to-Express communication working, your next step in this assignment is to implement Java code that can use your RESTful API from Part 1 by passing Person IDs to it in a URL and getting back JSON Person objects, which can then be converted to Java Person objects.

The starter code that we have provided contains a class called **Person,** which represents a person, using the same fields that the Node Express server uses. It also contains a class called **RemoteDataSource** that is meant to retrieve information about each person from a remote server on the web.

This part of the assignment asks you to implement the **RemoteDataSource.get** method; you do not need to implement or modify any other Java code.

Given an array of IDs, the RemoteDataSource.get method should return a Set of Person objects with the corresponding IDs from the Node Express server that you implemented in Part 1, including those for which the status is "unknown".

That is, this method should
1. generate a URL that includes query parameters with the IDs passed to it in the array argument
2. use that URL to connect to the Node Express server
3. read the array of JSON objects that the server returns in the HTTP Response
4. create (Java) Person objects for each of the JSON objects
5. return a Set of those Person objects; you may choose to return any Set implementation but should not change the signature of the method

Use the RemoteClientTest code from the "Before You Begin" section as the starting point of your implementation of this method, keeping in mind that you will need to dynamically build the URL and the fact that your server from Part 1 returns an *array* of JSON objects, and not a single object. However, you have seen how to handle arrays of objects in a previous assignment, so go back and look at your solution if you do not recall how to address this.

This method should make a new HTTP connection to the server each time it is invoked, and should close it before returning, i.e. it should not keep the connection open.

As in previous assignments, you should use the JSONSimple library for parsing the JSON objects that are returned by the server; do not attempt to parse these yourself!

Errors should be handled as follows:

- this method should throw an IllegalArgumentException and not attempt to send any data to the server if the String array input argument:
    - is null,
    - contains a null String, or
    - contains an empty String
- the method should throw an IllegalStateException if:
    - it cannot connect to the server, or
    - an exception occurs while processing the data read from the server, e.g. if a method from the JSON.simple library throws an exception
- this method should return an empty Set if:
    - the input String array is empty, or
    - the server sends back an empty JSON array

Your method does not have to specifically handle the case in which the JSON array is missing objects with IDs that were requested (unless it is entirely empty), or in which it includes objects with IDs that were *not* requested; it is sufficient that this method simply transform the array of JSON objects to a Set of (Java) Person objects, except for the error cases listed above.

Policies

This assignment is subject to the course policy on Academic Honesty as described in the Syllabus. It is expected that the code you submit is your own work, and that you have not discussed or shared code with anyone else. Please speak with the instructor if you have questions about the policy.

You may look online for help with Node Express, JavaScript syntax, the Java API and things like that, and you may use Piazza for asking about the specification or the intent of the assignment, or about setting up your Node Express installation, but please do not publicly post code or specific questions about the Java or JavaScript implementations.

Also, you may **not** use third-party libraries other than the JSON library we provided and the standard Java API for this assignment. That is, although you are of course free to use classes in the java.lang, java.util, java.net, etc. packages, you may not use libraries from other packages, e.g. org.apache.commons, without permission from the instructor.

Likewise, you may not use third-party JavaScript libraries aside from what is provided by Node Express for your server, though you should not need to for this assignment.

CMSC B353 Software Engineering - Spring 2021
Programming Assignment #4: RESTful API Development
Due Friday April 9, 5pm EDT

Grading

This assignment is worth a total of 50 points.

Part 1 is worth 20 points, and will be graded based on the extent to which your server returns the correct JSON object(s) for different queries.

Part 2 is worth 30 points total, based on the extent to which the RemoteDataSource.get method returns correct results and exhibits correct behavior for different inputs and for different states of the web server.

To avoid double jeopardy, Part 2 will be graded using a correct implementation of the Node Express server, i.e. you can still get 100% on this part even if your server from Part 1 does not work correctly.

Note, however, that your grade is based on the extent to which your implementation adheres to the specification described here. Just because your client and server communicate, doesn't mean you'll get 100% on this! Be sure your code is exhibiting the behavior described in this document.

Submission

This assignment is due at the date and time indicated above. Although the deadline is "technically" 5pm EDT, submissions received on the due date before 11:59pm will still be considered on time.

Beyond that, short extensions will likely be granted if requested *in advance* of the 5pm deadline. If you need more time to complete this assignment, please email the Instructor *before* it is due!

Then submit two files to the "Programming Assignment #4" assignment in Moodle: your index.js file for Part 1, and the RemoteDataSource.java code for Part 2. Be sure to submit your .java file, not your .class file; you do not need to submit Person.java from Part 2.