Details for Project Check-in #3

Submission Details:
The third project check-in will involve submitting your code through cuLearn and also deploying your server to an OpenStack instance. The due date for the check-in will be 11:59pm on Friday, November 13th. Once this deadline has passed, you will temporarily lose access to your OpenStack instance until the grading has been completed. You will then gain access to the instance again so you can prepare for the final submission.

Through cuLearn, you should submit a single .zip file containing all of your application's code. You only need to include the code. You should not include additional data, the node_modules folder, etc. You must also include a README.txt file containing:

1. What project you are working on
2. The name of both partners, if applicable
3. A summary of any extensions you have added to your system.
4. A summary of any design decisions you feel have increased the quality of your system
5. Your OpenStack instance information, including its public IP address (134.117.x.y) and username/password.
6. Instructions to the TA specifying how to initialize and run your server on the OpenStack instance.

Within your OpenStack instance, you should have everything your server requires already installed (e.g., dependencies, etc.). You do not need to have your server running; the TA will follow your instructions to start the server. You should include in your instance a script that will reset your server to its default starting state in case any errors occur during grading that require resetting everything. To evaluate your server's functionality, the TA will connect via SSH to your server and follow your given steps to initialize and run the server. The TA will then connect to your OpenStack instance using the SSH tunnel technique described in the course's OpenStack guide to test your server.

Base Expectations:
The main goals for this project check-in include:

1. Incorporating Express into your server
2. Adding support for user sessions to your server
3. Finalizing the design of your RESTful API
4. Using Express' functionality to connect incoming requests to the business logic you implemented for check-in #2 and to send appropriate responses
5. Supporting almost all of the functionality of the end application

At this point, you should be able to implement all required functionality for each project. You are still not expected to have incorporated a database into the project. It is expected that your server will store all of its data during operation in RAM. Adding a database and any final extensions or improvements to your project will be the focus of the final submission.

Unlike past check-ins, there will not be a defined list of additional expectations that will distinguish good projects (7/10 or 8/10) from great projects (9/10 or 10/10). Instead, the overall quality of your system's design will be the main determining factor. For example, your REST API should adhere closely to the principles outlined in the REST lecture (e.g., a logical naming scheme, use of proper HTTP methods, proper response codes, etc.). You should make an effort to ensure any processing is efficiently implemented and that you make use of asynchronous operations where possible. If you have already completed implementations of any extensions, this may also positively impact your grade for this check-in.

Hints and Tips:

1. As mentioned in the check-in #2 document, the required API in the specification is minimal. You will likely want to add significantly more to support all your operations.
2. If you have not started to design your REST API yet, think about the functionality your backend supports from check-in #2. Identify the types of resources and the types of operations that can be carried out on those resources (i.e., create, read, update, delete). Outline the required data for these operations and the expected response. Use this information to guide your definition of the routes/operations that the API will support.
3. It may be easiest to add basic session support into your system first (i.e., be able to log in/out and identify the user making a request). Once you have this working correctly, you can start to connect the rest of your functionality. This will allow you to work out the session problem in a less complex system, and also allow you to add in authorization of actions as you connect the rest of the system's functionality.
4. Use one regular browser window and one incognito window to support two sessions on the same computer at once (so you can test users interacting).
5. Continue to store your data in RAM and use your base data from check-in #2 to initialize your server state. If you haven't created any base data yet, it is advisable to do so.
6. Use the middleware chaining offered by Express to organize and synchronize your various business logic functions (e.g., for GET /someResource, you need to do func1(), func3(), then func6). Using *next()* to proceed to the following handler when your current step has finished is useful for managing asynchronous code. Your project may be lacking asynchronous server operations at this point but dividing your work this way will make it easier when you add your database, which will introduce a lot of asynchronous operations.
7. It is suggested that you develop and test your application locally (i.e., on your own computer) until you are satisfied it is working correctly. You can then deploy your application to OpenStack and give it a final test to ensure everything was set up correctly before the deadline.