Details for Project Check-in #2

Submission Details:
The second project check-in will be submitted through cuLearn. The due date for the check-in will be 11:59pm on Wednesday, October 21st. You should submit a single .zip file containing all of your project's required resources and a README.txt file containing:

1. What project you are working on
2. The name of both partners, if applicable
3. Instructions that specify how to setup, run, and test your server
4. A description of the files the TA should look at to evaluate your business logic code
5. A description of anything the TA should look at to evaluate any of the additional expectations that you have addressed in the submission

If your project is making use of modules installed via NPM, you should not include your *node_modules* folder in the submission zip. Instead, provide the required NPM resources to automate the installation of required modules using the *npm install* command.

Base Expectations:
The two deliverables below are the base expectations for project check-in #2. Note that you can keep these components completely separated at this point (i.e., a server JS file and a logic JS file). Successfully completing quality implementations of both would put your grade for check-in #2 in the 7/10 or 8/10 range.

1. You have a basic Node.js server capable of serving the static resources created within project check-in #1. You have also added some links between your static pages, where appropriate.
2. You have implemented the majority of the 'business logic' for your project. In essence, this will be an offline version of the system implemented in Node.js. You should also test this functionality to ensure it is working correctly. You can consult the trivia business logic example and lecture discussion on cuLearn for an example of what this business logic should look like.

At this point, it is still acceptable to store your data in RAM when running the server. You do not need to use files or a database. You are not expected to incorporate Connect/Express in your project at this point either, though some of the basic functionality (e.g., static serving middleware) may be of use.

Additional Expectations:
Successfully completing at least one of the below may allow you to boost your grade into the 9/10 or 10/10 range for check-in #2:

1. The static pages from check-in #1 are rendered by passing an example object into a template engine, where applicable (e.g., user profile page, etc.). You do not have to support multiple resources. Passing a single example object into the template is sufficient.
2. Support a few basic AJAX interactions to modify information on the server from the client and/or dynamically update the client view. At this point, these operations may manipulate some basic example data and do not need to be connected directly to the

underlying business logic that you have developed. A basic example would involve the client sending the registration information for a new user or the login information for an existing user. Your server can verify the credentials and send an appropriate response to the client, and the client can notify the user of the success/failure.

3. Outline some details regarding your planned REST API. The project specification includes a minimum API that you must support, but you will likely want to add additional routes/parameters/etc. to support the functionality of your application.

Hints and Tips:

1. Create some basic data for your server and store it in a file so it can be used for server initialization. You can directly load JSON from a file into an object in Node.js using the *require* directive. It may be worth creating several sets of base data to facilitate testing different scenarios.
2. Ensure every object in your system is given a unique ID. You can use simple integers or look into using the *uuid* module.
3. Where possible, use these unique IDs to reference these objects within your data and functions. For example, if storing an array of friends for a user, store the IDs of those friends instead of copies of the object. If you are writing a function to retrieve a specific user, have it accept a given user ID as an input argument. This will facilitate connecting your backend logic to the web for project check-in #3.
4. It is acceptable to completely disregard the concepts of logging in/out at this point. You will incorporate sessions and logging in/out for check-in #3 once you have connected your web interface with the backend logic you are creating for this check-in. You should still implement features such as private/public and limiting what a user can access (e.g., only seeing public games, only seeing your own stock watchlists, etc.), as these will ultimately be based on values stored on the server.
5. Once we introduce sessions into the server (check-in #3), your server will have access to the user that is making the current request, assuming that user is logged into your system. It is advisable to include a *requestingUser* argument in any function where you may need to access information about the user who is trying to do something. This is useful for functionality where the returned result depends on who has made the request (e.g., you have to be a friend of a player to see a non-public game, you have to be the owner of a stock portfolio to access that portfolio, you have to be a contributing user to create a new movie, etc.).
6. Consult the trivia business logic example and associated lecture content posted on cuLearn for an example of what your business logic may look like.