Assignment #4: Replication Testbed with Login and Connect

# Goal

In this assignment, you will modify the replication testbed to login to your user service, and then it will call a new API you will add to your service: *connect*. This new API will provide the replication client with information to locate the replication server and facilitate a “secure” connection.

You will start with the Node.js user service you created for Assignment 3, and the version of the replication testbed provided with the assignment (CS261\_Assignment4\_Initial.zip).

# Initial Tasks… but first…

A very important change happened to vcpkg between last year and this. Originally, if you did not specify which architecture you wanted, it would default to x86. However, the new version defaults to "whatever version architecture you're running", which for almost all of us, is likely "x64."

Unfortunately, the CProcessing (GAM 100) library that I have is x86. While I look for the source & the possibility of a 64-bit version, a quick workaround is to use the 32 bit versions, instead.

## Windows & Visual Studio: Install openssl and cpprestsdk

In the C++ REST SDK Lab, you used vcpkg to install the **cpprestsdk** library (probably the 64 bit version). You will also need to install **openssl**: **vcpkg install openssl** in the directory where you cloned vcpkg (during the lab).

Install the x86 packages:

./vcpkg install cpprestsdk:x86-windows  
./vcpkg install openssl:x86-windows

## Create a Username and Password for Your Test User

You will need to add a specific test user to your MongoDB database’s user collection. You should use your User Service to create this! This will make sure that IDs are in the correct format and that everything will be as your app expects.

An easy way to do this is with the command line tool, “curl”, which is an awesome tool for creating HTTP requests of all shapes and sizes. You should start to get familiar with this thing, it’s very helpful.

Curl should already be installed in your WSL Ubuntu environment, or if you use Cygwin, it should be there too. Otherwise, you’ll need to install it.

**Don't forget!** Make sure your node user service is running!

To create the user, from your WSL shell:

$ curl http://localhost:3100/api/v1/users \  
-d '{"username": "test\_user", "password": "test\_password", "avatar": "test\_avatar"}'\  
 -vv -H 'Content-Type: application/json'

(the “\” tells your shell to continue typing on the next line, useful for long command lines like this.)

You should see a bunch of ‘verbose’ output, including HTTP/1.1 200 OK and the result of json which will look *something like*:

{"id":"50b5d009-0047-4eaf-aa7a-f60f4ec4f61c", "username":"test\_user", "avatar":"test\_avatar","password":"test\_password"}

If it didn’t work, then you get to debug why user creation on your web service isn’t working! Good luck (and ask for help if you need it)!

## Three Things to check if your environment is Broken:

The client project won’t build without **cpprestsdk**,

and the server project won’t install with **openssl**,

and your project won’t login correctly if test\_user doesn’t exist in the right form!

# User Service Modifications

## Create assignment4 Project

Update the service to be named assignment4 (assignment4.js, etc.). Note that the API paths should not change – they’re still **v1**! The suggested method is:

1. Copy your assignment 3 service to a new directory. You could copy in Linux or via Windows Explorer, or simply unzip your submission. You should *not* copy the node\_modules directory - or if you do, you should delete it.
2. **del package\*.json** in the new directory
3. Run **mv assignment3.js assignment4.js** to rename the file.
4. Run npm init again, ensuring that you enter **assignment4.js** as the entry point.
5. Run **npm install packageName** for each package you’re using – likely express, uuid4, redis, and either mongodb or mongoose.
6. Update the MongoDB database name to **assignment4** in your code. Remember, MongoDB will automatically create databases and collections when needed.

## Create new "Connect" endpoint

Add a new API to the user service – *connect* – per the specification at the end of this document.

Notes:

* this API need authentication to be used correctly, so it requires a session passed in the body.
* Carefully read the specification of how *token* is generated. *You will need that for the client code*!

## Constants and Secrets

The game port and secret values will be constants in your javascript code – **4200** and **cs261secret**, respectively.

The token you provide will be encoded. The instructions for encoding are provided at the end of this document. Carefully read the specification to see how the token must be generated. You will need that for the javascript code!

# Replication Testbed Changes

The replication testbed has been updated in several ways:

* The solution has a separate client and server build, with different menu flows.
* The port-entry menus have been removed in favor of command-line/user-service configuration.
* Important configuration is passed in on the command line, as follows:

C:> CS261\_Assignment4\_Client.exe *user\_service\_url* *user\_name* *password*

**Example:**

CS261\_Assignment4\_Client.exe <http://localhost:3100> test\_user test\_password

**CS261\_Assignment4\_Server.exe *port secret***

* + - **Example:** CS261\_Assignment4\_Server.exe 4200 cs261secret

# TODOs in C++ Code

In **UserLoginState.cpp**, in the client project, you will need to use the C++ REST SDK (already referenced in framework.h) to perform a series of actions:

1. Login
2. Retrieve the login response data,
3. Use the login response data, plus the game type, to call *connect* on the user service
4. Finally, you will extract data from the connect response and store it in the client configuration.

In **HostingMenuState.cpp**, in the server project, you will calculate the token from the values sent by the client, plus the game type and secret, using the same pattern as the *connect* API.

You can easily find the TODOs using the Task List window in Visual Studio. Note that ***you should only worry about the TODOs in these two files*.** Depending on your filters, the Task View will likely list other TODOs in files that are part of the SDKs, which you *should* *not* try to address.

You may be tempted to make other changes to the C++ codebase – improvements or other refactoring. **You must not do so – you may only implement the TODOs, where they are located.**

# Testing

There is a new set of Postman tests for this assignment, posted alongside this document. The new tests include several cases for the new *connect* API. The new total is **93 unit tests.**

**When your project is graded, your client will be tested against the exemplar server, and your server will be tested against the exemplar client, both using *your* user service.** The exemplars are provided alongside this document (CS261\_Assignment4\_Exemplar.zip). There are batch files that run the client and server with the expected command-line arguments.

**When your project is tested for grading, it will be run with these command-line arguments:**

CS261\_Assignment4\_Client.exe <http://localhost:3100> test\_user test\_password

CS261\_Assignment4\_Server.exe 4200 CS261S21

Note that these are the same values used in the exemplar batch files.

You may test against my exemplar user service, ***URL TO BE GIVEN OUT LATER via TEAMS***, by providing it as an alternative first argument to CS261\_Assignment4\_Client.exe.

# Visual Studio Tips

Remember that you can set the command line arguments in the project properties, in the Debugging panel.

In the Solution properties, in the Startup Projects panel, you can use the “Multiple projects” setting to run both the client and server projects automatically, both under the debugger, each time you start debugging.

# Submission

Your submission should be named **yourAlias-CS261-4.zip**, where yourAlias is the username you use to log into your email and lab PC.

The submission should include:

* **node** directory, containing:
  + **assignment4.js**, the primary Javascript file (note the rename!)
  + Your other Javascript files (modules)
  + package.json
  + package-lock.json
  + *No node\_modules directory in here!*
* **game** directory, including replication testbed (the content of the Initial.zip, with TODOs in HostingMenuState.cpp and UserLoginState.cpp addressed). This would include:
  + Assets *(exactly as provided in Initial)*
  + CProcessing *(exactly as provided in Initial)*
  + CS261\_Assignment4
    - *No Debug or Release directories in here!*
  + CS261\_Assignment4\_Client
    - *No Debug or Release directories in here!*
  + CS261\_Assignment4\_Server
    - *No Debug or Release directories in here!*
  + CS261\_Assignment4.sln
  + *No Debug or Release directories out here, either!*

Note that you should not include any built binaries (Debug or Release directories, at any level), nor any .git or .vs directories. **The submission filesize should be around 1.9 MB.** Also note that CS261\_Assignment4\_Initial.zip models the structure/content of the game portion of your submission.

# Rubric

## 30 Points: Unit tests *all* pass (**93 total tests**)

The Assignment 4 set of unit tests will be run against your server in a configuration that matches what you set up in Assignment 3. If **all** **93 tests** pass, then you will receive 30 points **(otherwise 0 points).**

## 35 points: CS261\_Assignment4\_Client Test

Your version of CS261\_Assignment4\_Client.exe will be built and run against the exemplar server and your service, using the command-line argument specified above. If lockstep and dumb client scenarios can be successfully started, you will receive 35 points. Otherwise, you will lose points depending on the nature of the problem (0/35 for not compiling, etc.).

## 35 points: CS261\_Assignment4\_Server Test

Your version of CS261\_Assignment4\_Server.exe will be built and run against the exemplar client and your service, using the command-line argument specified above. If lockstep and dumb client scenarios can be successfully started, you will receive 35 points. Otherwise, you will lose points depending on the nature of the problem (0/35 for not compiling, etc.).

## Submission Penalties

In addition to the grading rubric above, you can receive these penalties if your submission is not correct:

* -5 if your submission files are not named as above.
* Up to -10 if your submission includes additional files, depending on the severity
  + The penalty has gone up… be careful!
* Up to -20 if other changes are made to the C++ codebase and projects beyond the specified TODOs, depending on the severity.

Other submission penalties may apply as needed.

# New API Reference

As before, this API reference describes the general API, but **the** **true reference is the Postman unit tests** and the **exemplar client and servers (which should work).** There are HTTP errors expected in various scenarios that are not described here.

Note that when authentication is required, that means that the **session** must also be in the client request. If it is missing or invalid, the operation should fail.

## Connect

**Path**: /api/v1/connect

**Verb:** POST

**Authenticated**: Yes

**Request Body:**

* game\_type

**Response Body:**

* username
* avatar
* game\_port
* token

The username and avatar should be for the user associated with the session.

The game\_port should always be **4200**.

The token must be calculated in a particular way:

* Calculate the plaintext token by adding four strings together, in this order:
  + **let plaintextToken = username + avatar + game\_type + secret**
* Calculate the encoded token:
  + At the top of the code file, add:
    - **const crypto = require('crypto');**
      * Note: ‘crypto’ is part of Node.js. You do not need to install any npm packages for this to work!
  + After calculating plaintextToken (with your own object name, as appropriate)
    - **output.token = crypto.createHash('sha256').update(token).digest('base64');**

**Note that the secret should be cs261secret.**

# Technical Notes

The “token” pattern is what we studied in CS 260: by demonstrating the ability to calculate identical tokens, using a shared secret, the server is able to validate that the client really did log in to the user service.