Assignment #4: Game Client, Server, User-Service

# Goal

In this assignment, you will modify a simple Game Client to first login to your Assignment 3 user service and then call a new API you will add to your service: *connect*. This new API will provide the game client with a url to the game server and replicate a secure (sic) connection.

# Setup

First, in Windows, use a Git client to retrieve the initial Game project:

git clone https://github.com/digipen-network-classes/cs261-game

This uses the x86 (only) version of CProgramming and hasn’t been updated to the latest version or x64.

## Windows & Visual Studio: Install openssl and cpprestsdk

In the C++ REST SDK Lab, you used vcpkg to install the **cpprestsdk** library. You will also need to install **openssl**: **vcpkg install openssl** in the directory where you cloned vcpkg (during the lab). You’ll need the x86 version, because that’s what the test client supports.

Install the x86 packages:

cd C:/users/(yourname)/source/repos

./vcpkg install cpprestsdk   
./vcpkg install openssl

./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.

First,make sure your node user service from assignment 3 is running!

To create a test user (Example):

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

You should see a bunch of ‘very verbose’ output (that’s the -vv flag) which will include HTTP/1.1 200 OK and the result which will look *something like*...

{"id":"50b5d00900474eafaa7af60f", "username":"test\_user", "avatar":"test\_avatar","password":"test\_password"}

If that didn’t work, then you get to debug why user creation on your web service isn’t working! (Take a close look at the punctuation in the curl command, it’s easy to get wrong.)

## 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.

# User Service Modifications

## Create new "Connect" endpoint

Add a new API to the user service – *connect* – per the specification. You may want to review the README in the cs261-game project (https://github.com/DigiPen-Network-Classes/cs261-game/tree/main) for more information.

Note: This endpoint requires authentication.

The request body:

{

"session": "valid-session-token",

"game\_type": "some-game-type"

}

and the response is:

{

"username": "test\_user",

"avatar": "test\_avatar",

"game\_port": 4200,

"token": "aabbccddee"

}

# Game Client and Server

### Starting the Client

CS261\_Assignment4\_Client.exe user\_service\_url user\_name password

Example:

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

### Starting the Server

CS261\_Assignment4\_Server.exe game\_server\_port shared\_secret

Example:

CS261\_Assignment4\_Server.exe 4200 cs261secret

# Modifications to the Game Client

## UserLoginState.cpp

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 -- see the TODOs in UserLoginState.cpp:

1. call Login on your user-service
2. Retrieve the login response data
3. Use that login response data, plus the game type, to call *connect* on the user service
4. Extract data from the connect response and store it in the client configuration, for sending to the game server later.

### ConnectingMenuState.cpp

Once you’ve gotten the response values into the configuration, you need to modify the ConnectingMenuState.cpp in the client to send these values to the Game Server so that the Game Server can build a token and compare theirs to the one the client submits.

# Modifications to the Game Server

## HostingMenuState.cpp

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.

# User Service Data Flow Diagram

For this assignment you will need to prepare a data flow diagram demonstrating the interactions between the game client, game server, user-service, redis, and monogdb -- using the Threat Modeling Tool (introduced in CS 260): https://aka.ms/threatmodelingtool

Create the model for the interactions between Game Client, Game Server, the databases, and your user service, specifically for the login and connect requests and the client and game server interactions. You can use the generic form for each entity, no need to select a ‘redis datastore’ etc.

Next, use the tool to export a CSV of the potential threats. Include both the csv and the diagram with your submission.

# Testing

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

# 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 Moodle.

The submission should include:

* **node** directory, containing:
  + **user-service.js**, the primary JavaScript file
  + Your other JavaScript files (modules)
  + package.json
  + package-lock.json
  + *No node\_modules directory in here!*
  + *No .env file!*
* **game** directory:
  + Assets *(as provided in Initial)*
  + CProcessing *(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!*
* **Docs** directory
  + Threat Model File (Data Flow Diagram)
  + Threats CSV

Note that you should not include any built binaries (Debug or Release directories, at any level), nor any .git or .vs directories.

# 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).**

## 30 points: CS261\_Assignment4\_Client Test

Your version of CS261\_Assignment4\_Client.exe will be built and run against a test server and service. 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.).

## 30 points: CS261\_Assignment4\_Server Test

Your version of CS261\_Assignment4\_Server.exe will be built and run a test client and service. 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.).

## 30 points: Threat Model and Threats

Include your threat model exported list of threats.

## 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
* 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 Putter unit tests**. There are HTTP results 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 should be calculated in a particular way:

* Calculate the plaintext token by adding four strings together, for example:
  + **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');**

Hint: The game server will need to construct its token in the same order!