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 user service will provide information to locate the

Tasks

You will start with the Node.js user service you created for Assignment 3, and the version of the replication test-bed provided with the assignment (CS261_Assignment4_Initial.zip).

Before you begin, there are two important steps of setup:

1. In the C++ REST SDK Lab, you used vcpkg to install the **cpprestsdk** library. You will also need to install **openssl**, by running **vcpkg install openssl** in the directory where you cloned vcpkg (during the lab).
    a. Note: do not install other openssl packages you might read about online, as they have been deprecated.
2. You will need to add a specific test user to your MongoDB database's user collection. You should use the MongoDB command-line interface to do this. Once you login with the command you used during the MongoDB lab, you can insert the user with this command:
    a. **db.users.insertOne({“username”:”test_user”, “password”:”test_password”, “avatar” : “test_avatar”})**
    b. If you use your own UUID-based IDs, you can include that field as well in the object. You can generate UUIDs online here.

Note that the client project won't build without **cpprestsdk**, and the server project won't install with **openssl**.

In the user service, you will:

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

    - Note that this API is authenticated, so it requires a session passed in the body.
    - Note the specification of how *token* is generated. *You will need that for the client code*.
    - The game port and secret values will be constants in your code – 4200 and CS261S21, respectively. We will formalize our approach to Node.js configuration in a later assignment.
    - The token you provide will be encoded. The instructions for encoding are provided at the end of this document.
- 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:
    1. **CS261_Assignment4_Client.exe *user_service_url user_name password***
        - **Example:** CS261_Assignment4_Client.exe http://localhost:3100 test_user test_password
    2. **CS261_Assignment4_Server.exe *port secret***
        - **Example:** CS261_Assignment4_Server.exe 4200 CS261S21

Most importantly, there are TODOs in two files you will need to implement:

- 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, available at http://54.151.57.127:3100, by providing it as an alternative first argument to CS261_Assignment4_Client.exe.

When developing in Visual Studio:

- 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 if 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 CS261S21.

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.