Assignment #2: RoboCat Login/Connect

Goal

In this assignment, we will start making modifications to the RoboCatAction game provided in our textbook, ***Multiplayer Game Programming.***

Currently, the game client takes a command-line argument with the game server address, and a player name. In this assignment, we have three goals:

- Add a new API to your Node application, **connect,** to return information about the game server.
- Add code to the game client to login to your user service, and then call the new **connect** API to get the information necessary to connect to the game server.
- Adapt the game server to run on Linux.

Technology

We will be starting with essentially the RoboCatAction source code from chapter 8. This is available on GitHub, but I have provided a starting point to use instead, that contains:

- A NuGet reference to the C++ Rest SDK, which we will use to connect to your service from the game client.
- A makefile to build the game server on Linux.
- A few system-header references added to some code files.

You will note that the RoboCatAction server will not build on Linux as provided. You will need to make a few fixes to the source code to allow it to build on Linux as well as Windows, using **#ifdef WIN32**.

In the client, you will find code that reads the command-line arguments and uses them to connect to the game server. You will replace that code with code that uses the C++ Rest SDK to call your server to:

- Call the login API you already wrote, with a username and password provided on the command line to the client.
- Call the connect API you will add in this assignment, passing the session and token provided by the login API.
- Use the response of the connect API to connect to the game server.

The C++ Rest SDK is a mature product, and there is extensive documentation online, as well as a tutorial and several very useful samples. In the provided starting point, there is already a NuGet reference to a usable version of the library. Once you load the project for the first time, you should **right-click on the Visual Studio solution and select "Restore NuGet packages".**

Your approach with the SDK integration should be:

1) Identify the code that is reading the command-line arguments in the client app, and replace them with the output of your login/connect functionality (see remaining steps).
2) Retrieve the new set of **three** command-line arguments (see below), using the same functions the client already provides.

3) Build a HTTP client and URL objects to login to your server, with the appropriate JSON request body.
4) Login to the server and retrieve the response JSON body.
5) Create a new URI and use the previous response body as the request body for the connect API.
6) Retrieve the data from the response body, and use it where the client originally used the command line arguments to connect to the game server.

For your user service, you will start with your existing Node project, which should operate and listen in the same way as Assignment 1, with the following changes:

- **assignment1.js** should be renamed to **assignment2.js**
  - **package.json** should be updated to reflect this change
- Create a test user automatically on startup, with a username **usr** and a password **pwd**, and an avatar **avatar1.png**
- Add a new route for POSTing to /api/v1/connect, per the documentation below.
  - Note that this API should always return a server value of 127.0.0.1:3300, which means that the game server should (and will) always be tested bound to port 3300.

Note that the API "version" in the URL is still V1. Maintaining an API version in the URL is important to guard against major breaking changes, and we have not introduced any breaking changes (just a new API). API version is *not* the same as assignment number!

Given these changes, the RoboCatAction exes will be called in this way during testing:

- **RoboCatServer.exe 3300**
- **RoboCatClient.exe http://127.0.0.1:3100 usr pwd**

Alongside this document, you will find:

- The initial RoboCat code you should start with for this assignment.
- An updated set of Postman tests.
  - I recommend deleting the existing collection from Postman, and then importing the new file.

Deployment

For this assignment, your server will continue to run within your Linux VM. We will not add any other infrastructure to your service in this assignment.

We will run Node simply, by directly calling **node** with the primary JavaScript file, which must be called **assignment2.js**. You can and should use other JavaScript files to isolate elements of the implementation.

Testing

We will test your RoboCat project in combination with my exemplar implementations, per the grading rubric below.

Your user service will also be tested with Postman again. The test suite is exactly the same as the previous assignment, adding new tests for the new APIs, for a total of 82 tests.

Your server and user service will be graded in the same VM environment provided for class. You could develop and test in any environment that supports Node, but I suggest you test in the class VM appliance provided in Euclid.

Your RoboCatAction client program should be tested on DigiPen lab computers in their current configuration.

Submission

For this assignment, you will include your updated RoboCatAction project and your Node project, in a submission zip named **YourAlias-CS261-2.zip**, where YourAlias is the username you use to log into your email and lab PC.

Within the .zip, you should include this structure

- **RoboCatAction** directory, using the same structure as the provided starting point, specifically:
  - No Bin or Debug directories
  - No .git, .svn, or .vs hidden directories.
  - No .o files, executables, or any other binary output (from Linux).
- **Node** directory, containing your user service with the same submission requirements as before:
  - assignment2.js, the primary JavaScript file
  - Your other JavaScript files (modules)
  - package.json
    - Remember to update this for the rename to assignment2.js!
  - package-lock.json
  - No other files (node modules directory, etc.)

Grading

40 points: RoboCatAction Client Test

Your RoboCatAction client will be tested with the command referenced above, against my exemplar game server and your user service. If the client connects to my game server via the user service and starts a game, then you will pass the test and receive 40 points (otherwise 0 points).

30 points: RoboCatAction Server Test

Your RoboCatAction server will be tested in the class VM with the command referenced above, against my exemplar client and your user service. If my client connects to your game server via the user service and starts a game, then you will pass the test and receive 30 points (otherwise 0 points).

30 points: User Service Unit Tests

The Assignment 2 set of unit tests will be run against your server in the class VM. If **all 82 tests** pass, then you will receive 20 points (otherwise 0 points).

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.
- -5 if your submission includes additional files.
- -10 if your server operates on an IP and port other than **localhost:3100**.
- -10 if your server's connect API returns another server location than **127.0.0.1:3300**

API Reference

As before, this API reference describes the general API, but **the true reference is the Postman unit tests**. There are HTTP errors expected in various scenarios that are not described here.

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

All unreferenced APIs should remain, implemented as they were in assignment 1.

Connect

Path: /api/v1/connect

Verb: POST

Authenticated: Yes

Request Body: None (other than the authentication)

Response Body:

- id (the user ID)
- username
- avatar
- server

In this assignment, this API returns the user information listed for the user associated with the authentication session. In addition, it will include a **server** field that is always **127.0.0.1:3300**

This may seem like an odd operation for POST, but in future assignments, this API will have additional responsibilities. Note that the password is not returned for any user.