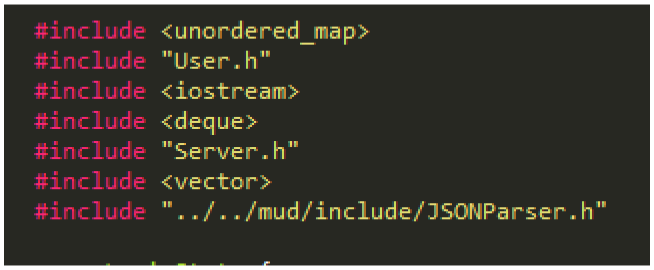
Code Review

1:

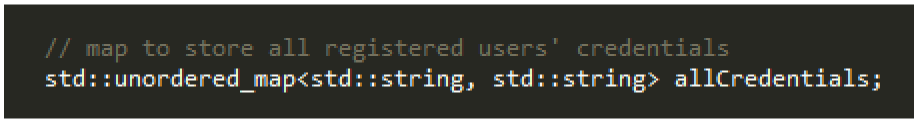


```
#include <unordered_map>
#include "User.h"
#include <iostream>
#include <deque>
#include "Server.h"
#include <vector>
#include "../../mud/include/JSONParser.h"
```

The last **#include** has a relative path instead of just the header file's name. This signifies that the **CMakeLists** were not setup properly and should be fixed instead of working around the problem.

It seems like the **JSONParser.h** exists in a different directory, so the correct solution would be to link the libraries together via another **CMakeLists** file.

2:

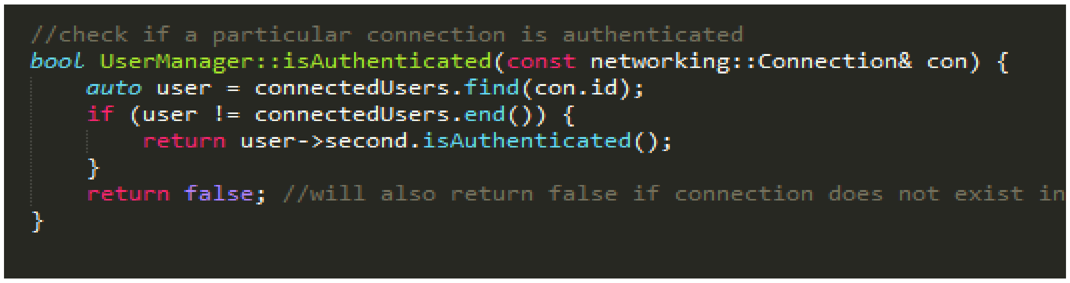


```
// map to store all registered users' credentials
std::unordered_map<std::string, std::string> allCredentials;
```

This **allCredentails** is a bit unclear as to what is supposed to be passed in the first and second arguments in terms of order. Having two same argument types for a map should be avoided.

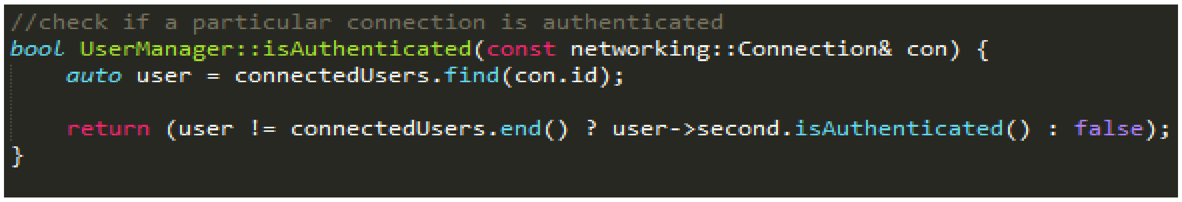
Possible solution to this is to create a typedef for whatever those two values should be so that a clearer distinction is made.

3.



```
//check if a particular connection is authenticated
bool UserManager::isAuthenticated(const networking::Connection& con) {
    auto user = connectedUsers.find(con.id);
    if (user != connectedUsers.end()) {
        return user->second.isAuthenticated();
    }
    return false; //will also return false if connection does not exist in
}
```

A routine should aspire to only have one exit point. This function contains two. One in the if-statement, and one at the end (which is hardcoded to return false). A solution to this would look something like:

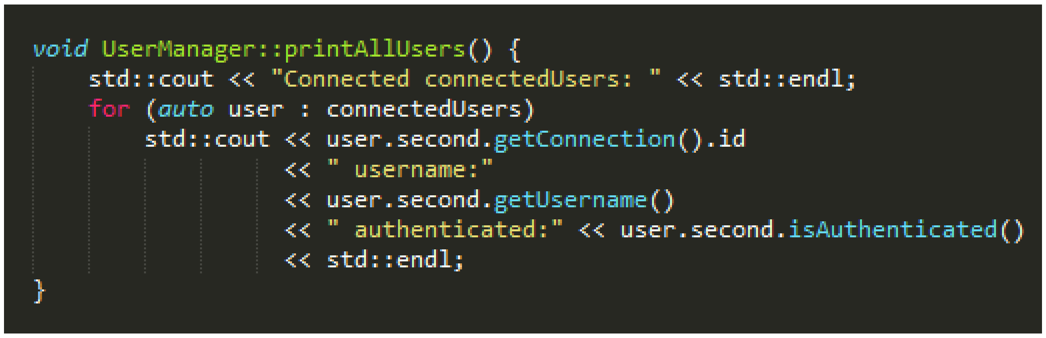


```
//check if a particular connection is authenticated
bool UserManager::isAuthenticated(const networking::Connection& con) {
    auto user = connectedUsers.find(con.id);

    return (user != connectedUsers.end() ? user->second.isAuthenticated() : false);
}
```

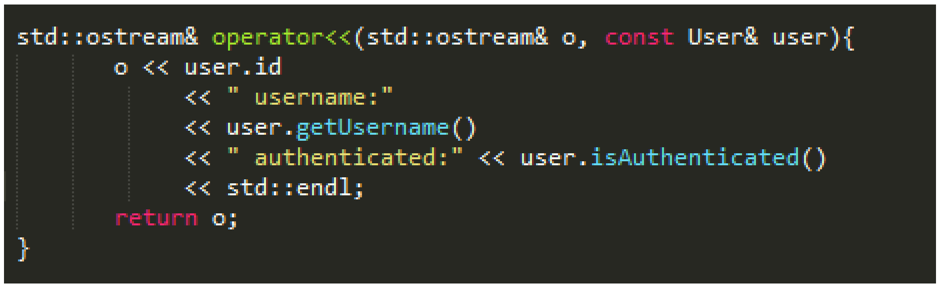
This looks a lot cleaner and doesn't require having an if-statement with two separate end-points.

4.



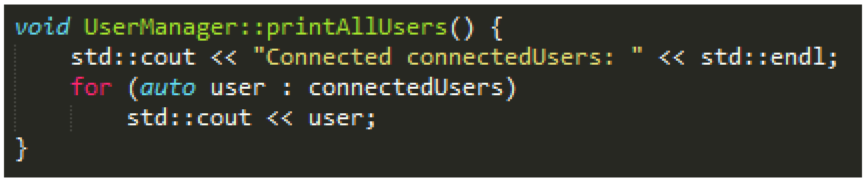
```cpp
void UserManager::printAllUsers() {
    std::cout << "Connected connectedUsers: " << std::endl;
    for (auto user : connectedUsers)
        std::cout << user.second.getConnection().id
                  << " username:"
                  << user.second.getUsername()
                  << " authenticated:" << user.second.isAuthenticated()
                  << std::endl;
}
```

This could probably be done in the **User** class by overriding the **<< ostream** operator.



```cpp
std::ostream& operator<<(std::ostream& o, const User& user){
        o << user.id
          << " username:"
          << user.getUsername()
          << " authenticated:" << user.isAuthenticated()
          << std::endl;
        return o;
}
```

Your new **printAllUsers()** method would look something like this now:



```cpp
void UserManager::printAllUsers() {
    std::cout << "Connected connectedUsers: " << std::endl;
    for (auto user : connectedUsers)
        std::cout << user;
}
```

Syntactically, something may be off here so it's best to do your own research overriding the **<<ostream** operator, but this is the general idea of what you should be striving for.

1.

These are a few small things I think need revising. In UserManager.h, the professor has told my group that includes with a direct path like "../../mud/include/JSONParser.h" could lead to unintended problems with cmake. Wrapping username/password in something like a struct would probably be better design although it might be overkill. In validateUser, you should probably fill in the default case with something, so it becomes easily detectable when a default case happens.

2.

As mentioned in class, it is better design to use the json library as it is intended. You should probably use the to_json / from_json functions from the library. This also leads to a probably where it seems like your data classes that you use in the game are coupled with the json. Adding in some middle layer would help.

3.

We discussed in class that studies have shown that a flatter structure with early exits is better than a deep structure. In validateCredentials, you could probably invert the first condition, have it return LoginState::WRONG_LOGIN, and then bring the second if condition out and have it return the other enum states.

4.

There is a TODO to hash and store password and update the user credentials file which could lead to some performance problems later. It would make sense to store the changes and to update the file periodically rather that constantly since it takes a lot of time to write to files. The Enum used is also oddly named. There is a conflict whether this is for login or for register. For example, if the status is a wrong login, it is set to display a successful message which doesn’t really makes sense.

- For LoginState

    - there are some instances where you are using if(enum == LoginState::...) and some cases where you are using a switch statement. Usage should be consistent, expecially since both cases where the enums were used one way could work as well as the other

- Nice use of the json/nlohmann library in JSONParser. Although if you are passing the json by reference you might want it to be const since you're only accessing those json not modifiying them

```
bool UserManager::isAuthenticated(const networking::Connection& con) { ….

   return false; //will also return false if connection does not exist in UserManager

}
```

- instead of returning a boolean literal at the end you can do:

```
        bool isAuthenticated(..){

        auto user = connectedUsers.find(con.id);

                return user != connectedUsers.end() && user->second.isAuthenticated();

        }
```

    - you won't get runtime errors or segfault because this boolean expression short circuits on "user != connectedUsers.end()" if false and it won't evaluate the second expression

```
case (LoginState::CORRECT_PASSWORD):

      os << "No such user found. Please, register first!" << std::endl;

      ifValidated = true;
```

- Maybe a simple mistake in the output but, if the user is found(otherwise the loginstate would be WRONG_LOGIN) and has the correct password then this should not output this message

```
if(!user.isMessageEmpty()){

      outgoing.push_back({user.getConnection(), user.getUserMessagesConcatenated()});

      user.clearMessages();
```

- Instead of using a getConnection() and getUserMessagesConcatenated(), maybe have a getMessage() in User class model to generate the Message for you and calls the clearMessages()

- This way you're caller won't have to call clearMessages() each time getUserMessagesConcatenated() is called, removes some of the responsibility of maintaining integrity of model from the caller.

ICR 4 – CMPT 373 – Rishabh Bhargava – 301243452

line 101: Send message and build outgoing should be in a different class, that handles communication between client and server.

Line 244: This function should not exist for privacy reasons as it exposes passwords of users

line 207 and 228: These functions are too similar to each other, they should be combined into one.

Line 281: An npc already has a short description, and a long description, It should not have the “description” variable. Maybe you meant extended description

Line 346: Storing id as a struct is a better idea.

Ling 408: Initializing a user with only a username, the password is not stored.