Code Review

Babka

1. Connection id (uintptr_t) datatype is used as the key of a map (int). This leads to implicit casting from uintptr_t to int. This implicit casting seems unreliable and can lead to casting errors in the future. It also increases complexity for the reader/future developer leaving it prone to human errors. Moreover, as Connection is a small structure, it is preferable to pass the Connection instead of the id as indicated by the prof.

2. It seems that the object User is created by the caller. The class UserManager responsible for managing the users does not have complete control over the User objects. It appears tightly coupled to the caller and the User. Moreover, the ownership of a User object is unclear as it is being created and managed in different places. An alternative will be to let the UserManager own Users and abstract the implementation details. It should expose an interface to manager users (which it seems to do). The caller should be able to communicate through the interface. This will reduce the complexity, hide implementation details and make the code less coupled.

3. Retrieving and passing all the user information in one string is bad interface (Authenticate function). Its misleading for the developers working on the code and leads to malformed inputs that can be difficult to test and debug. Moreover, using regex to parse a string containing username and password introduces accidental complexity that is unnecessary. A better approach would be to retrieve username and password separately and pass them as two different arguments to the authenticate function. This makes the code simpler, less complex and saves unnecessary parsing.

4. Authenticate function does not return any information to the user nor does it throw any exceptions. It can lead to unintentional errors. For instance, a user typing invalid password. With the current design, the only way to infer malformed input would be to scan the server console. This increases overhead for a developer and lets unauthorized users access the application. Authenticate function must either return a state or throw error when an unauthorized user tries to access the application.

5. A few stylistic concerns:
    - Authenticate function is extremely complex and can be simplified by decreasing the unnecessary ‘if’ statements and proper error handling.
    - Line 56 has a relative path to an included header file. This is bad practice and can be avoided. The relative paths should be handled in cmake files.
    - There seems inconsistency in the function naming scheme. Some function use camel case while others use title case. It is always good practice to use a standard naming convention to improve readability for a future developer.

CODE REVIEW - BABKA

PROS:

Intention and use of this class is commented on clearly, so it is well documented and easy to see what is supposed to be happening.

Good use of whitespace and style. Function names are clear.

Nothing returns true or false.

Parameters use const and are passed by value.

CONS:

There is a lot going on in the Authenticate function. I understand that any message incoming will be processed by this class first to determine authentication status, but maybe there's a better way to do this besides authenticating every message? Since there is no context on the rest of the project, I can't comment conclusively on this, but I thought I'd mention it since that's the main purpose of this class.

Also, perhaps it would be better to separate the new user behavior into a separate login function, so that Authenticate isn't responsible for more than one function.

It's not clear what conId is supposed to be from the variable name.

It could be hard to test this code as there doesn't seem to be a lot of exceptions thrown and all but one of the useful functions are void. For example, in the Logout function, it's

clear that if the user is found, then the user's setAuthenticated is set to false (which logs them out), but what happens when the user isn't found… and how is that exception handled when this function is called elsewhere.

ICR 3

1. I like the idea of having a findUser and moving returning a user from a remove into there. Returning a removed item has never intuitively made sense to me anyways.
2. I'm having a pretty hard time figuring out what the buildOutgoing function does. If I spent an extra minute or two I'm sure I'd be able to figure it out, but it isn't exactly the most intuitive function. Consider some documentation!
3. In logout, you seem to use a .find() despite already implementing a findUser within the class. Use the code you've already written! ☺
4. I really don't have a 4th point here; 50% of the code is just declarations for the implementations later in the .cpp.

Babka Code Review

Line 56 has #include "../include/UserManager.h"

- This should be #include "UserManager.h"
- CMake not being used properly. It exists so things like this do not have to happen.
- Will lead to problems if files are moved

UserManager has a Logout method

- First issue is method name starts with capital letter which is bad
- Second issue is this UserManager class has a method for log out, but not log in
- Can't see how log in is handled so probably in a different class adding dependencies
- Putting logout and login in same class will have better layer of abstraction and less dependencies

Variable names unclear

- Variables like conId and sep are hard to tell what they are
- Can't understand logic because I am not sure what variables mean
- Hard to tell what to put in arguments for some methods because name is not clear
- Examples:
    - Line 123: can't tell what "log" is. I know it is a string, but if I am using this method I will not know for sure what to put in argument
    - Line 67: conID gives me the idea that it is supposed to be an ID for a user?
- Names can be made clearer by using pronounceable words

Many methods check if input is correct first

- For methods buildOutgoing, Logout, and findUser the method checks if input is valid which is good
- However, for methods addUser and removeUser there is no checking. May have been done because is not necessary, but what would happen if you erase something from a map that doesn't exist

std::deque<networking::Message> buildOutgoing(const std::string& log);

- should this function be in the UserManager? I don't think this is specific to the users or a class holding a collection of users.

- This function could live inside your server code since Messages are something your server uses quite often and its the one that needs a deque of messages.

- in your Authentication method, you check for valid user input using regex, and a couple lines down use a

try/catch block when you do actually parse the string input

- Is the regex check actually doing its job and catch invalid inputs? If you have a try/catch block it implies

that it does not, which asks why have both. You could use one that is more readable for you and easy to understand

```
User& UserManager::findUser(const uintptr_t &conId) {
    auto user = Users.find(conId);
    if(user != Users.end()){
        return user->second;
    }
```

- this doesn't return anything if the user is not found, which can cause compile/runtime errors. You could return a nullptr and have the caller check if this method returns a nullptr

```
std::unordered_map<int, User> Users;
void UserManager::addUser(User &newUser) {
    Users.insert({newUser.getConnection().id, newUser});
}
```

- You have a map that has int as the key, but you insert the connection id as key which is a uintptr_t. Its generally not a good idea to implicitly cast the connection id into a different type when storing it in a variable, since it might cause some problems when you do check if two ids are equal

Code Review 3: Babka

1.

```
void removeUser(const uintptr_t& conId);

User& findUser(const uintptr_t& conId);

void Authenticate(const uintptr_t& conId, const std::string& userInfo);

void Logout(const uintptr_t& conId);
```

In these methods, instead of a uintptr_t you should use a wrapper class, like was discussed in class, such as the provided network::Connection class.

2.

```
std::unordered_map<int, User> Users;
```

It's smart to use an unordered_map here since the order doesn't matter. Map uses a self-balancing binary search tree in its implementation, so extra overhead is used to balance the tree and insert a new item in the right position.

3.

```
#include "../include/UserManager.h"
```

You shouldn't need to include the relative path like this if you have cmake set up properly. You should just be able to use #include "UserManager.h".

4.

```
User& UserManager::findUser(const uintptr_t &conId) {

    auto user = Users.find(conId);
    if(user != Users.end()){
        return user->second;
    }

}
```

This method does not have a return statement for if a user is not found. I suggest being explicit and return nullptr if a user is not found.

5.

```
std::string userName;
std::string pwd;
try {
    boost::char_separator<char> sep{" "};
    tokenizer<boost::char_separator<char>> tok(userInfo, sep);
    tokenizer<boost::char_separator<char>>::iterator beg = tok.begin();
    beg++;

    userName = *beg;
    beg++;

    pwd = *beg;
    //std::cout << userName << " " << pwd << "\n";
```

I think a better way to do this part would be to convert the whole login command string into a vector of strings, rather than go one by one using the iterator. It would be easier to change in the future if more parameters are added. Also it's easier to read.

The code would look like this:

```
#include <boost/algorithm/string/classification.hpp>
#include <boost/algorithm/string/split.hpp>

...
std::string username;
std::string pwd;
try {
     std::vector<std::string> params;
     boost::split(params, userInfo, boost::is_any_of(" "),
     boost::token_compress_on);
     username = params[1];
     pwd = params[2];
     …
```