ECE470:Network Client-Sever Programming

Project 1: Part 1

Aaron Jesus Valdes

March 19, 2021

Contents

1 Project Description

Designing a smart home remote access system which follows the Network/Client architecture. This system allows the user to control certain devices from their home such lights, alarm, and locks from a remote location.

2 Project Purpose

3 Project Goals

4 Business Requirement

4.1 Assumptions

- We are only working with a single house
- There are multiple rooms in the house
- Each room has multiple lights
- There is only one alarm
- There are at least 4 locks
- One user at a time can connect
- The server is in the home
- The server can be accessed both inside and outside of the house
- The user needs to check and alter status of the devices
- The method of communication is
- The server must be protected by having a login page

4.2 Constraints

- Does not allow the user to add rooms to the house
- Does not allow the user to add lights to the house
- Does not allow the user to add locks to the house
- Does not allow the user to add new devices to the house

4.3 Deliverable

- Supports smart lights
  - Can turn on and off lights
  - Can turn off all lights for a room
  - Groups lights into rooms
- Supports alarm system
  - Can arm and disarm the alarm
  - Can have more than one alarm system
  - Can store 4 digit pin per alarm
  - Ask the user for a pin to change the alarm
- Supports Electronic Locks
  - Can open and lock the locks
  - Can have more than 4 locks
  - Can store a 5 digit pin per lock
  - Ask the user for a pin to change the lock

5 Design

5.1 Data Model

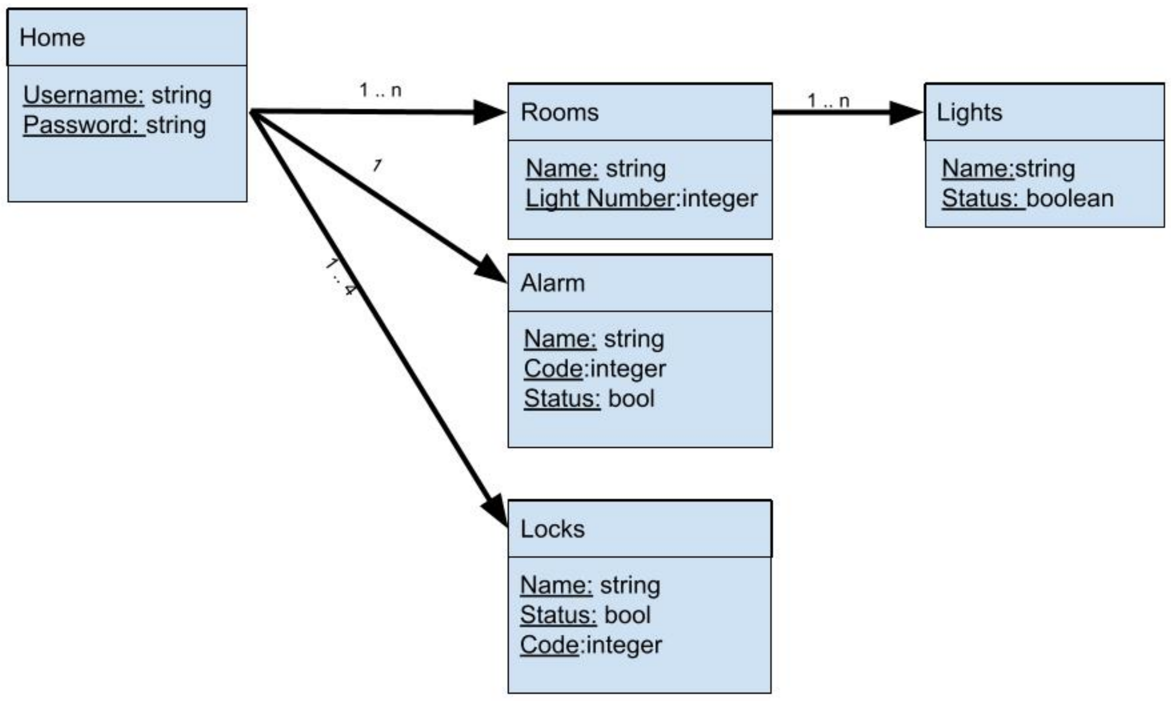


Figure 1: Data models follow the specifications

This is the simplest data model which we can use as a template given the fact that as we move one with this project we are going to be adding more features such as setting RGB colors,change lock combinations, setting the brightness, and later on be able to add more devices

5.2 Business Logic

5.2.1 Basic Operations

- Login/Logout
- Check Status of each device
- Change Status of each devices

5.2.2 Storyboard

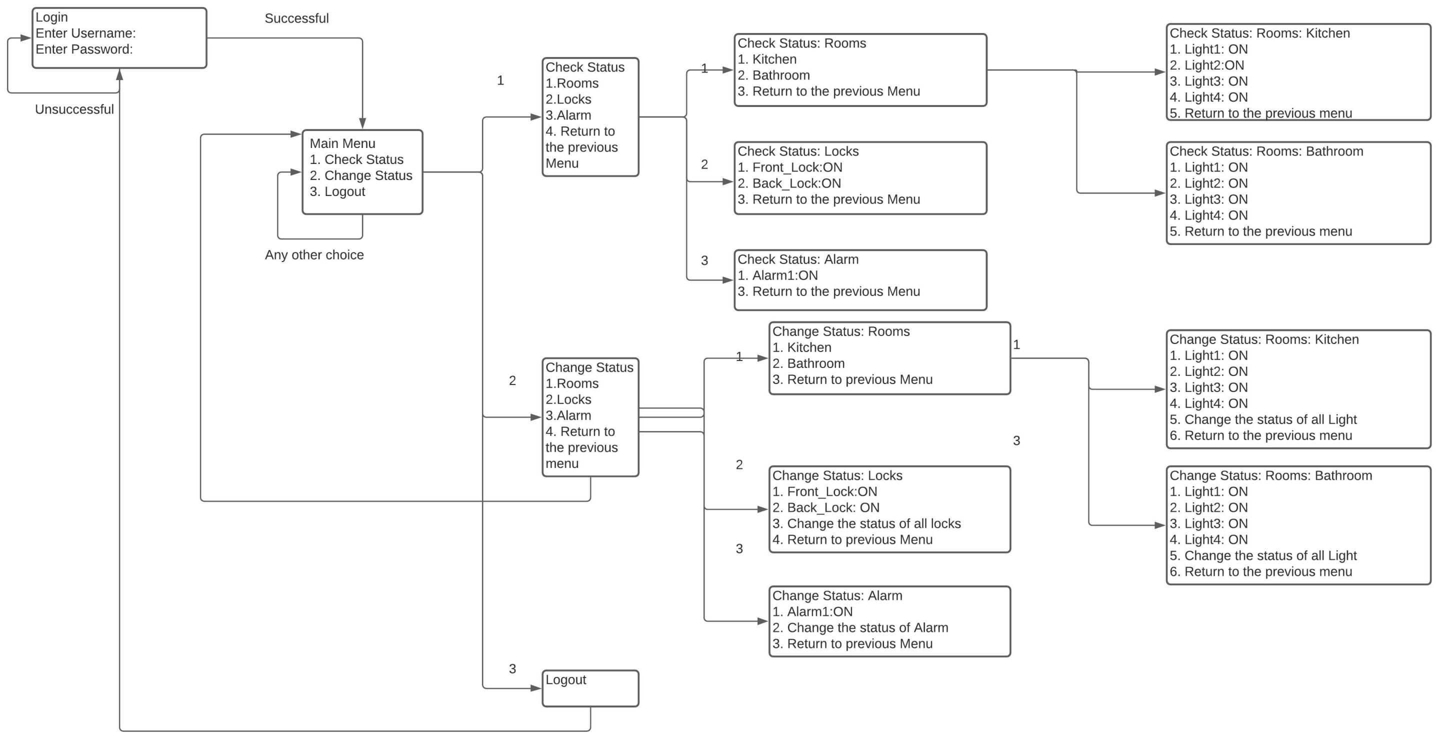
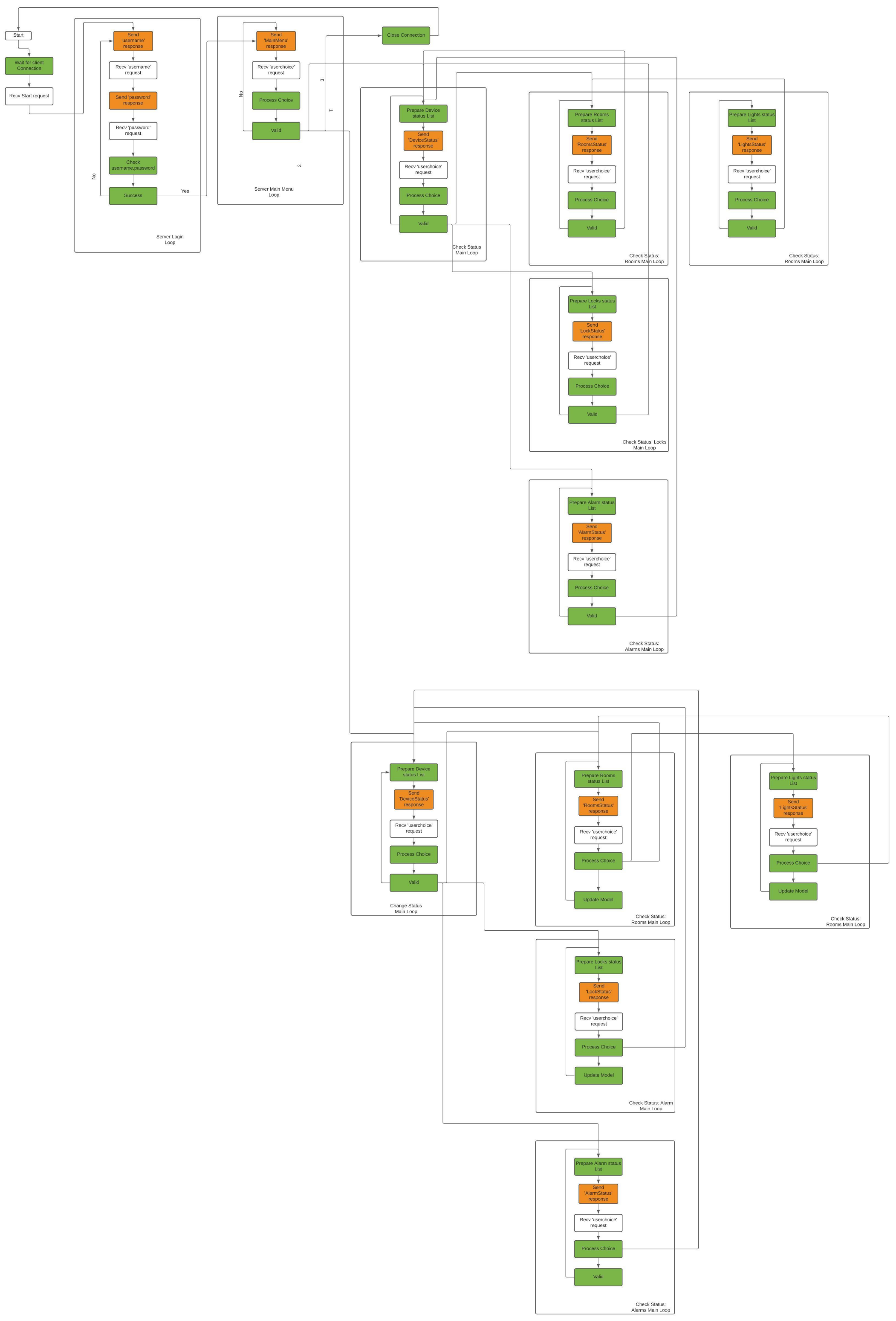


Figure 2:

The reason I chose to divide my check status and change status based on devices is because that is the method that my google home uses show the status of my devices and change the status of my devices. In my opinion the most effective method is by simply having a single main menu with a all the options since it leads to less amount of packets being transferred back and forth between the client and the server and can therefore reduces bandwidth. As we move on, I am going to implement more menus for the alarm,locks, and lights as way to provide more features for each device

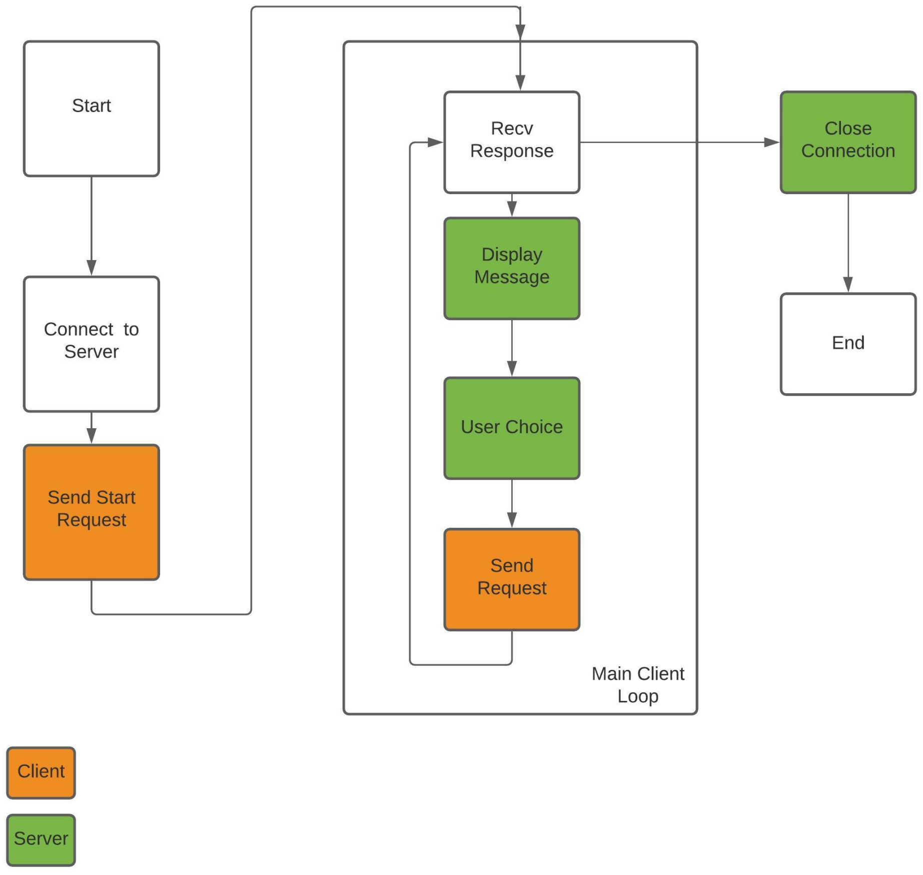
5.2.3 Sever state chart



Start
Wait for client Connection
Recv Start request
Send 'username' response
Recv 'username' request
Send 'password' response
Recv 'password' request
Check username,password
Success
Yes
No
Server Login Loop
Send 'MainMenu' response
Recv 'userchoice' request
Process Choice
Valid
No
3
1
2
Server Main Menu Loop
Close Connection
Prepare Device status List
Send 'DeviceStatus' response
Recv 'userchoice' request
Process Choice
Valid
Check Status Main Loop
Prepare Rooms status List
Send 'RoomsStatus' response
Check Status: Rooms Main Loop
Prepare Lights status List
Send 'LightsStatus' response
Check Status: Rooms Main Loop
Prepare Locks status List
Send 'LockStatus' response
Check Status: Locks Main Loop
Prepare Alarm status List
Send 'AlarmStatus' response
Check Status: Alarms Main Loop
Change Status Main Loop
Update Model
Check Status: Rooms Main Loop
Check Status: Rooms Main Loop
Check Status: Alarm Main Loop
Check Status: Alarms Main Loop

The server state diagram starts by simply waiting for the client to establish a connection. After establishing a connection, the user is going to be asked to send their username and password which is going to be check by the server based on the data model. If the username and password is correct then we move on to the main menu else we return back to the server login menu. After entering the main menu, the user needs to send the server a request for either checking status of a device, changing the status of a device, or logging out . After the user request the either check or change the status, they are going to receive the list of devices available to choose from. After requesting a device, the user is going to either the rooms, locks, or alarms available to them so they can either change or check the status. For changing the status there is an extra step in which the server updates the data model for each change and returns back to displaying the status of that device. For each menu there is always a return option which is simply by requesting to press enter.

5.2.4 Client State Chart



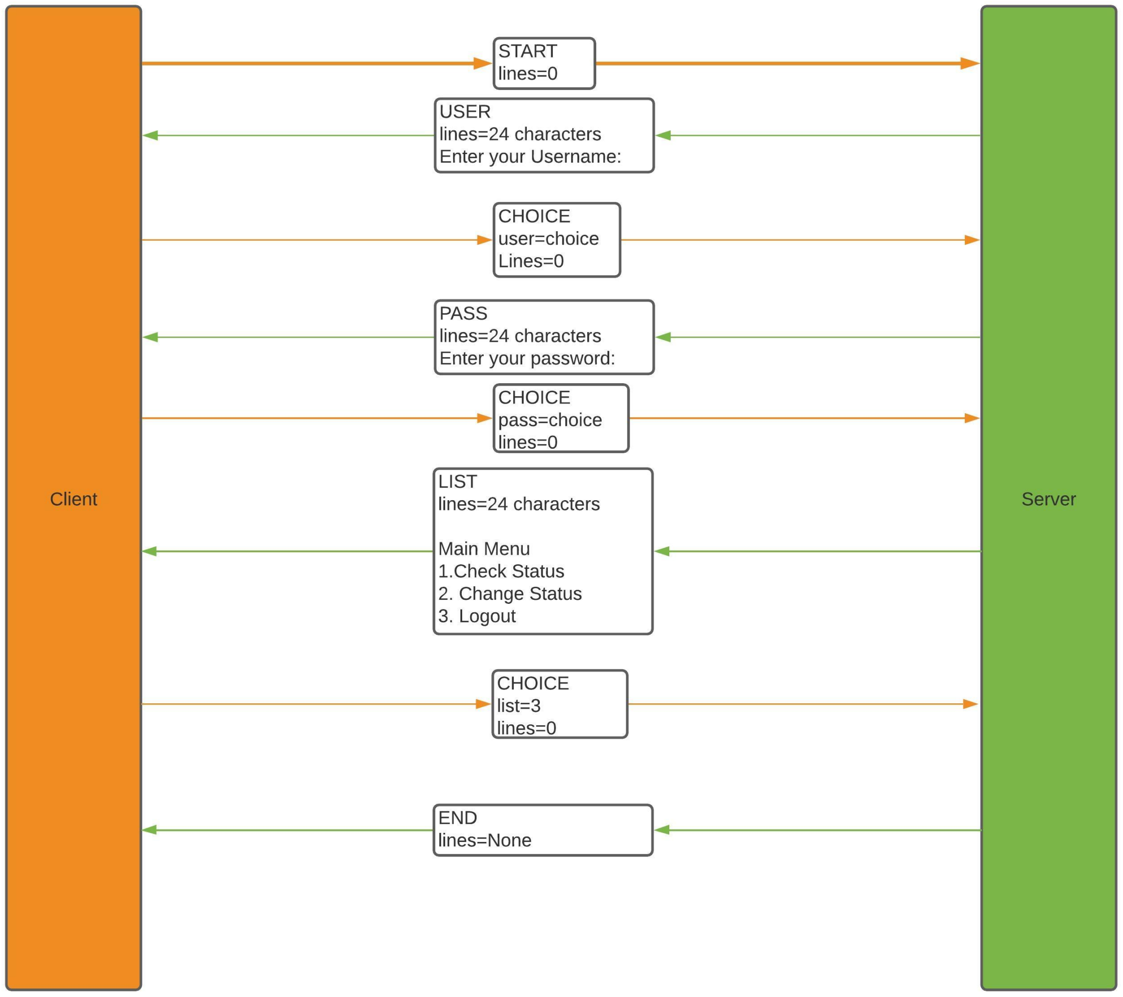
The client state diagram starts by simply establishing a connection between the server and the client. Then the server sends the client a message,which the user uses to make a request. After the request is sent to the server, the server makes a decision based on that request and executes a command. The connection between the server and the client ends when the user decides to log out.

5.2.5 Application Protocol Design

Client Application Protocols	Parameters	Body	Description
START	none	none	Establishes connection with the server
CHOICE	Lines=24 character,lines=0	None	Based on the options of the server, the user can send up to 24 character to the server
END	None	none	Ends the connection between the client and the server

Server Application Protocols	Parameters	Body	Description
USER	Lines=24 character	Line	Ask the user for the username
PASS	Line=24 character	Line	Ask the user for the password
LIST	Line=24 character	Line(s)	Gives the user options which they can chose from
ERROR	Line=24 character	Line(s)	Send the user information about an error in the system and waits for his choice
END	None	None	The server ends the connection between the client and the server

5.2.6 Example



6 Implementation

My steps for the implementation

1. Create the client marshal and test that it works

2. Create the server marshal and test that it works
3. Create the network for the client to send and receive
4. Create the network for the server to send and receive
5. Create the main for the client

   Follow the business model for the client

   Make sure that it is able to send and receive
6. Create the login menu for the server
7. Create the main menu for the server
8. Create the check status menu
9. Create the change status menu
10. Create the the an alarm menu

    With the option to create a specific menu for check and change status

    For change alarm status ask the user for a pin to be able to change the status
11. Create the rooms menu
12. Create the lights menu

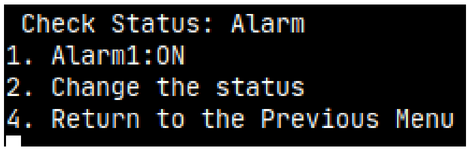
    With the option to create a specific menu for check and change status and for the individual room

    Add the option to either change all lights or a single light
13. Create the locks menu

    With the option to create a specific menu for check and change status

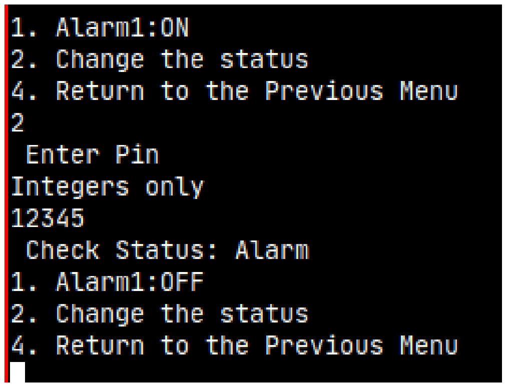
    For change lock status ask the user for a pin to be able to change the status of either all locks or individual locks

7 Testing



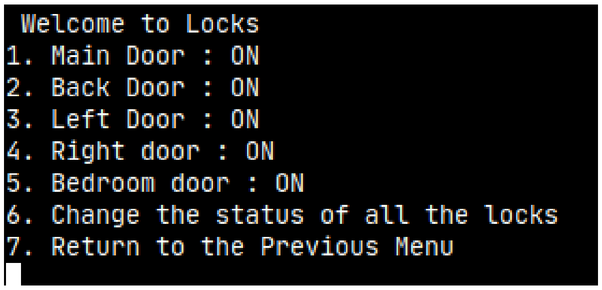
```
 Check Status: Alarm
1. Alarm1:ON
2. Change the status
4. Return to the Previous Menu
```

Figure 3:



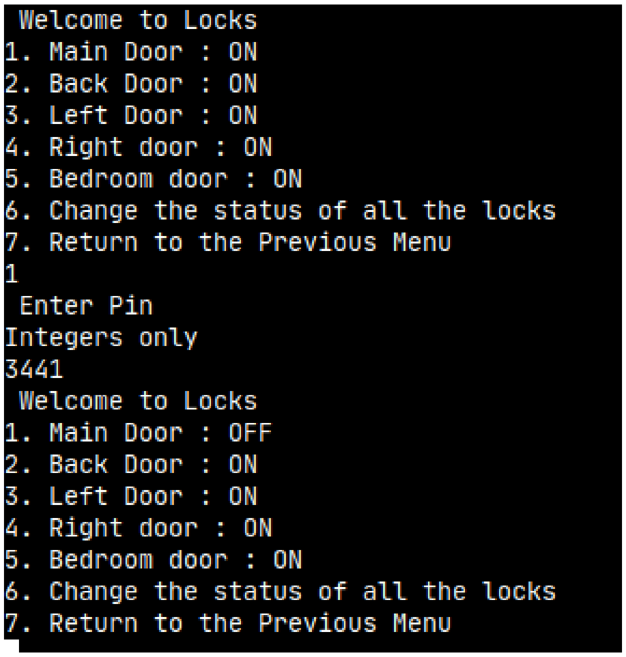
```
1. Alarm1:ON
2. Change the status
4. Return to the Previous Menu
2
 Enter Pin
Integers only
12345
 Check Status: Alarm
1. Alarm1:OFF
2. Change the status
4. Return to the Previous Menu
```

Figure 4:



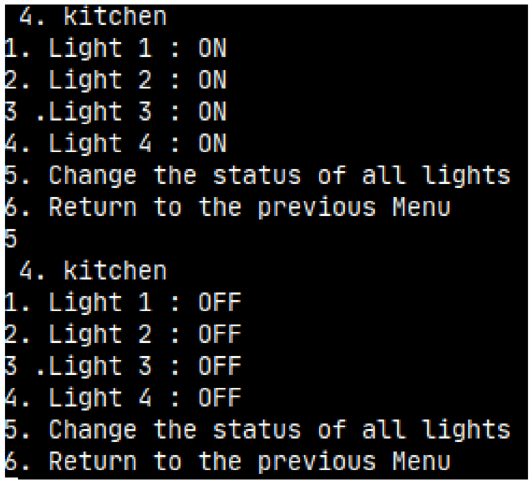
```
 Welcome to Locks
1. Main Door : ON
2. Back Door : ON
3. Left Door : ON
4. Right door : ON
5. Bedroom door : ON
6. Change the status of all the locks
7. Return to the Previous Menu
```

Figure 5:



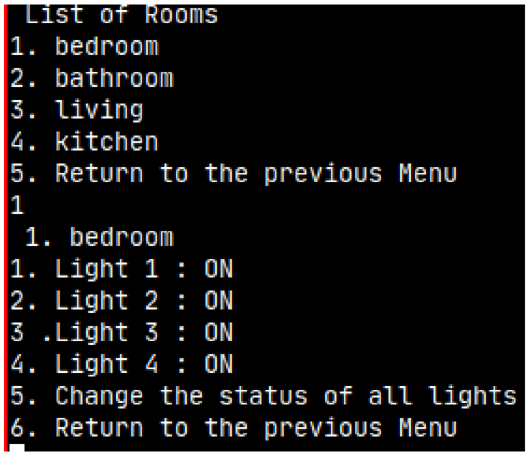
```
 Welcome to Locks
1. Main Door : ON
2. Back Door : ON
3. Left Door : ON
4. Right door : ON
5. Bedroom door : ON
6. Change the status of all the locks
7. Return to the Previous Menu
1
 Enter Pin
Integers only
3441
 Welcome to Locks
1. Main Door : OFF
2. Back Door : ON
3. Left Door : ON
4. Right door : ON
5. Bedroom door : ON
6. Change the status of all the locks
7. Return to the Previous Menu
```

Figure 6:



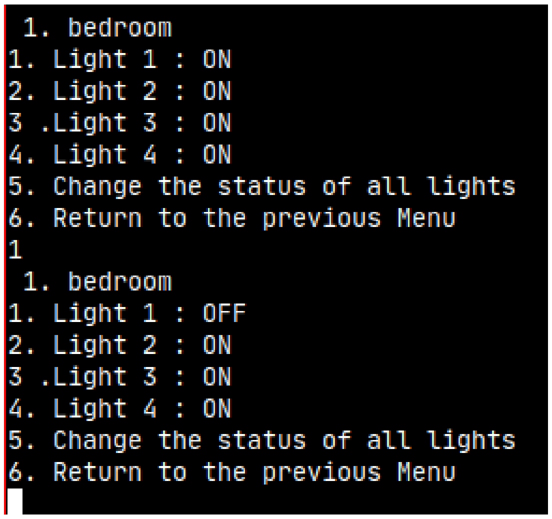
```
 4. kitchen
1. Light 1 : ON
2. Light 2 : ON
3 .Light 3 : ON
4. Light 4 : ON
5. Change the status of all lights
6. Return to the previous Menu
5
 4. kitchen
1. Light 1 : OFF
2. Light 2 : OFF
3 .Light 3 : OFF
4. Light 4 : OFF
5. Change the status of all lights
6. Return to the previous Menu
```

Figure 7:



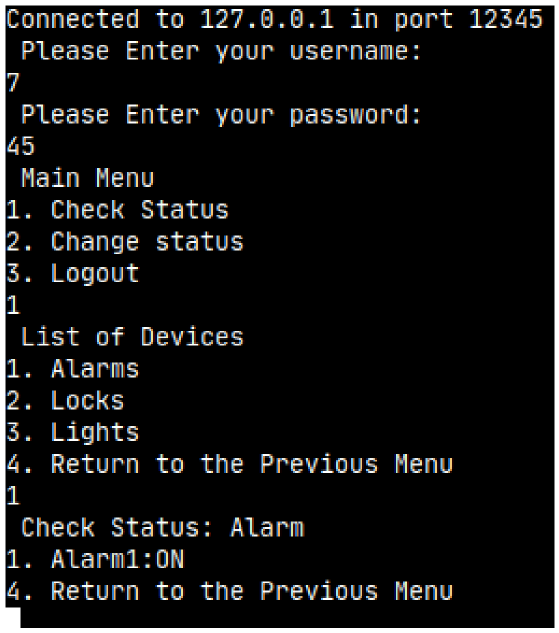
```
 List of Rooms
1. bedroom
2. bathroom
3. living
4. kitchen
5. Return to the previous Menu
1
 1. bedroom
1. Light 1 : ON
2. Light 2 : ON
3 .Light 3 : ON
4. Light 4 : ON
5. Change the status of all lights
6. Return to the previous Menu
```

Figure 8:



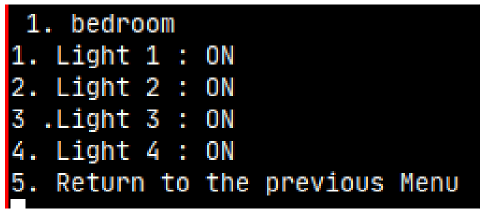
```
 1. bedroom
1. Light 1 : ON
2. Light 2 : ON
3 .Light 3 : ON
4. Light 4 : ON
5. Change the status of all lights
6. Return to the previous Menu
1
 1. bedroom
1. Light 1 : OFF
2. Light 2 : ON
3 .Light 3 : ON
4. Light 4 : ON
5. Change the status of all lights
6. Return to the previous Menu
```

Figure 9:



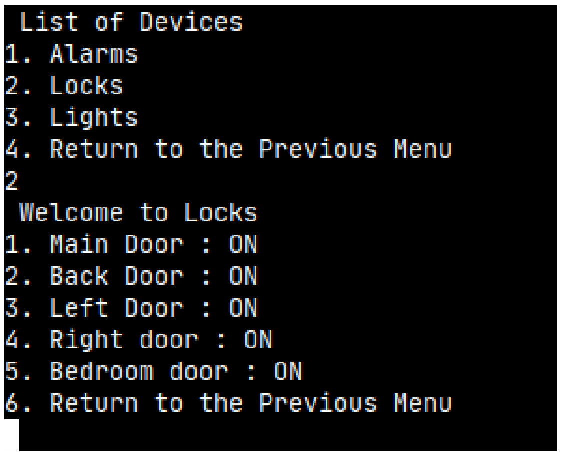
```
Connected to 127.0.0.1 in port 12345
 Please Enter your username:
7
 Please Enter your password:
45
 Main Menu
1. Check Status
2. Change status
3. Logout
1
 List of Devices
1. Alarms
2. Locks
3. Lights
4. Return to the Previous Menu
1
 Check Status: Alarm
1. Alarm1:ON
4. Return to the Previous Menu
```

Figure 10:



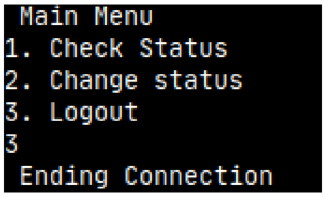
```
 1. bedroom
1. Light 1 : ON
2. Light 2 : ON
3 .Light 3 : ON
4. Light 4 : ON
5. Return to the previous Menu
```

Figure 11:



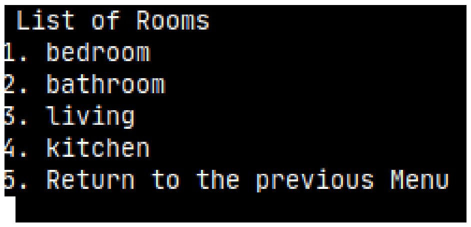
```
 List of Devices
1. Alarms
2. Locks
3. Lights
4. Return to the Previous Menu
2
 Welcome to Locks
1. Main Door : ON
2. Back Door : ON
3. Left Door : ON
4. Right door : ON
5. Bedroom door : ON
6. Return to the Previous Menu
```

Figure 12:



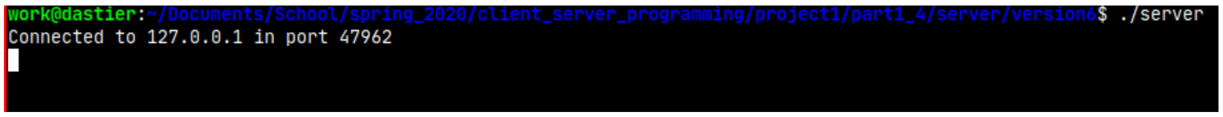
```
 Main Menu
1. Check Status
2. Change status
3. Logout
3
 Ending Connection
```

Figure 13:



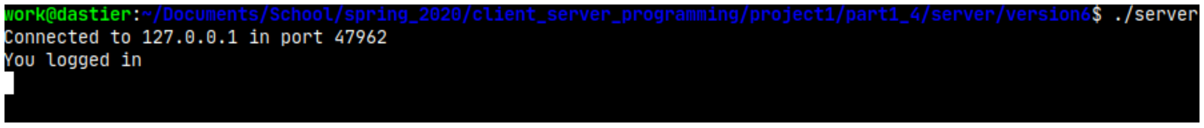
```
 List of Rooms
1. bedroom
2. bathroom
3. living
4. kitchen
5. Return to the previous Menu
```

Figure 14:



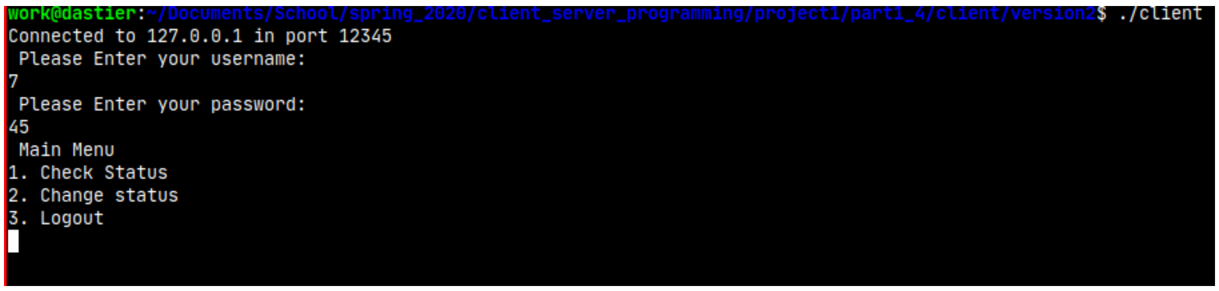
```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_4/server/version6$ ./server
Connected to 127.0.0.1 in port 47962
```

Figure 15:



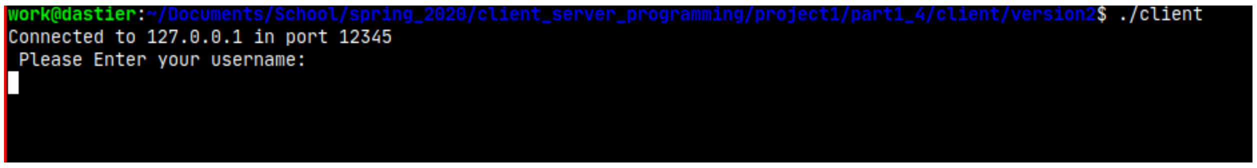
```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_4/server/version6$ ./server
Connected to 127.0.0.1 in port 47962
You logged in
```

Figure 16:



```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_4/client/version2$ ./client
Connected to 127.0.0.1 in port 12345
 Please Enter your username:
7
 Please Enter your password:
45
 Main Menu
1. Check Status
2. Change status
3. Logout
```

Figure 17:



```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_4/client/version2$ ./client
Connected to 127.0.0.1 in port 12345
 Please Enter your username:
```

Figure 18:

8 Conclusion

Overall this project took a lot of time, which I didn't expect it would take since I though the implementation was going to be easy. Either way I was able to implement a working program that can handle the simplest features and can easily expand to others such having smart lights which can be dimmed or change the color. Furthermore, this program can be further improve by having a hashing function for the password and username since that can help us with the security aspect of it. Furthermore we can add more features to make sure that the user does not put anything that is not an integer as a choice.

9 Appendix-Code

9.1 Data model

```
#ifndef _DATAMODEL_H_
#define _DATAMODEL_H_
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
using namespace std;


class light
{
public:
string name;
bool status;
light();
light(string name,bool status);
};

light::light()
{
name="Bedroom Light";
status=true;
}
light::light(string name1, bool status1)
{
name=name1;
status=status1;
}


class room
{
public:
vector<light> lights;
vector<string> lightsList;
string roomName;
int roomNumber;
int numLights;
room();
room(string roomMa ,int ro);
void setLightList();
```

```cpp
void changeAllLights();
void changeSingleLights(int option);
};

void room::changeAllLights()
{
for(unsigned int i=0;i<lights.size();i++)
{
lights[i].status=!lights[i].status;
}
setLightList();
}

void room::changeSingleLights(int option)
{
lights[option].status=!lights[option].status;
setLightList();
}

void room::setLightList()
{
vector<string> tempLight;
for(unsigned int i=0;i<lights.size();i++)
{
stringstream ss;
string lightSta;
string temp;
if(lights[i].status)
{
lightSta="ON";
}
else
{
lightSta="OFF";
}
ss<<lights[i].name<<" : "<<lightSta;
getline(ss,temp);
tempLight.push_back(temp);
}
lightsList=tempLight;
}


room::room()
{
light t1("Light 1",true);
light t2("Light 2",true);
light t3("Light 3",true);
light t4("Light 4",true);
lights.push_back(t1);
lights.push_back(t2);
lights.push_back(t3);
lights.push_back(t4);
roomName="Bedroom";
numLights=lights.size();
roomNumber=1;
setLightList();
}
```

```
room::room(string roomNa,int ro)
{
roomNumber=ro;
stringstream ss;
string temp;
ss<<roomNumber<<". "<<roomNa;
getline(ss,temp);
light t1("1. Light 1",true);
light t2("2. Light 2",true);
light t3("3 .Light 3",true);
light t4("4. Light 4",true);
lights.push_back(t1);
lights.push_back(t2);
lights.push_back(t3);
lights.push_back(t4);
roomName=temp;
numLights=lights.size();
setLightList();
}



class rooms
{
private:
public:
vector<room> house_room;
int numRoom;
rooms();
vector<string> roomList;
int obtainNumberofRooms()
{
return numRoom;
}
void setRoomList();
};

rooms::rooms()
{
room bedroom("bedroom",1);
room bathroom("bathroom",2);
room livingroom("living",3);
room kitchen("kitchen",4);
house_room.push_back(bedroom);
house_room.push_back(bathroom);
house_room.push_back(livingroom);
house_room.push_back(kitchen);
numRoom=house_room.size();
setRoomList();
}

void rooms::setRoomList()
{
for(unsigned int i=0;i<house_room.size();i++)
{
roomList.push_back(house_room[i].roomName);
}
}
```

```
class alarmSystem
{
private:
bool status;
int code;
public:
alarmSystem()
{
status=true;
code=12345;
};
string name;

void changeStatus(int opt)
{
if(opt==code)
{
status=!status;
}
}

bool getStatus()
{
return status;
};
};



class lock
{
public:
string name;
int code;
bool status;
lock();
lock(string nam,bool stat,int codd);
};

lock::lock()
{
name="Main";
code=12345;
status=true;
}

lock::lock(string nam,bool stat,int codd)
{
name=nam;
code=codd;
status=stat;
}

class locks
{
private:
int masterLock;
public:
```

```
vector<lock> llock;
vector <string> locksList;
int numLocks;
locks();
void setLocksList();
void changeAllLocks(int opt);
void changeSingleLock(int option, int opt);
};

void locks::changeAllLocks(int opt)
{
if(opt==masterLock)
{
for(unsigned int i=0;i<llock.size();i++)
{
llock[i].status=!llock[i].status;
}
setLocksList();
}
}

void locks::changeSingleLock(int option,int opt)
{
if(opt==llock[option].code)
{
llock[option].status=!llock[option].status;
setLocksList();
}
}

locks::locks()
{
lock ll1("1. Main Door",true,3441);
lock ll2("2. Back Door",true,1234);
lock ll3("3. Left Door",true,7812);
lock ll4("4. Right door",true,2232);
lock ll5("5. Bedroom door",true,2131);
llock.push_back(ll1);
llock.push_back(ll2);
llock.push_back(ll3);
llock.push_back(ll4);
llock.push_back(ll5);
numLocks=llock.size();
setLocksList();
}

void locks::setLocksList()
{
vector<string> tempLock;
for(unsigned int i=0;i<llock.size();i++)
{
stringstream ss;
string lightSta;
string temp;
if(llock[i].status)
{
lightSta="ON";
}
else
{
lightSta="OFF";
}
ss<<llock[i].name<<" : "<<lightSta;
getline(ss,temp);
tempLock.push_back(temp);
}
locksList=tempLock;
}

class home
{
private:
string password;
string username;
public:
vector<string> deviceList;
rooms ro;
alarmSystem al;
locks lo;
home();
void setDeviceList();
};


void home::setDeviceList()
{
deviceList.push_back("1. Alarms");
deviceList.push_back("2. Locks");
deviceList.push_back("3. Lights");
}

home::home()
{
setDeviceList();
}
#endif
```

9.2 Message Protocol

9.3 Client Marshal

9.3.1 Header file

```
#ifndef _CLIENT_MARSHAL_H_
#define _CLIENT_MARSHAL_H_
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <vector>
using namespace std;
class server_message
{
public:
string options;
string message;
vector<string> lines;
```

```
int num_lines;
void printingDebug()
{
cout<<options<<"    "<<num_lines<<"     "<<message<<endl;
for(unsigned int i=0;i<lines.size();i++)
{
cout<<lines[i]<<endl;
}

}
void printing()
{
cout<<message<<endl;
for(unsigned int i=0;i<lines.size();i++)
{
cout<<lines[i]<<endl;
}

}
};

class client_message
{
public:
string options;
int decision;
void printing()
{
cout<<options<<"    "<<decision;
}
};

server_message unmarshal(string message)
{
string command;
server_message res;
stringstream ss(message);
ss>>command;
if(command=="USER")
{   res.options="USER";
getline(ss,res.message);
}
else if(command=="PASS")
{
res.options="PASS";
getline(ss,res.message);
}
else if(command=="LIST")
{
res.options="LIST";
getline(ss,res.message,'\\');
ss>>res.num_lines;
vector<stringstream> tt(res.num_lines);
string temp;
int i=0;
while(ss>>temp)
{
if(temp=="\\")
{
```

```
i++;
}
else
{
tt[i]<<temp<<" ";
}
}
for(unsigned int j=0;j<tt.size();j++)
{
string temp2;
getline(tt[j],temp2);
res.lines.push_back(temp2);
}
}
else if(command=="ERROR")
{
res.options="ERROR";
getline(ss,res.message);
ss>>res.message;
}
else if(command=="END")
{
res.options="END";
getline(ss,res.message);
ss>>res.message;
}
else
{
}
return res;
}


string marshal(client_message cm)
{
stringstream ss;
string result;
if(cm.options=="START")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="CHOICE")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="END")
{
ss<<cm.options<<" "<<cm.decision;
}
else if(cm.options=="ERROR")
{
ss<<cm.options<<" "<<cm.decision;
}
else
{
}
getline(ss,result);
return result;
}
```

```
#endif
```

9.3.2 Test File

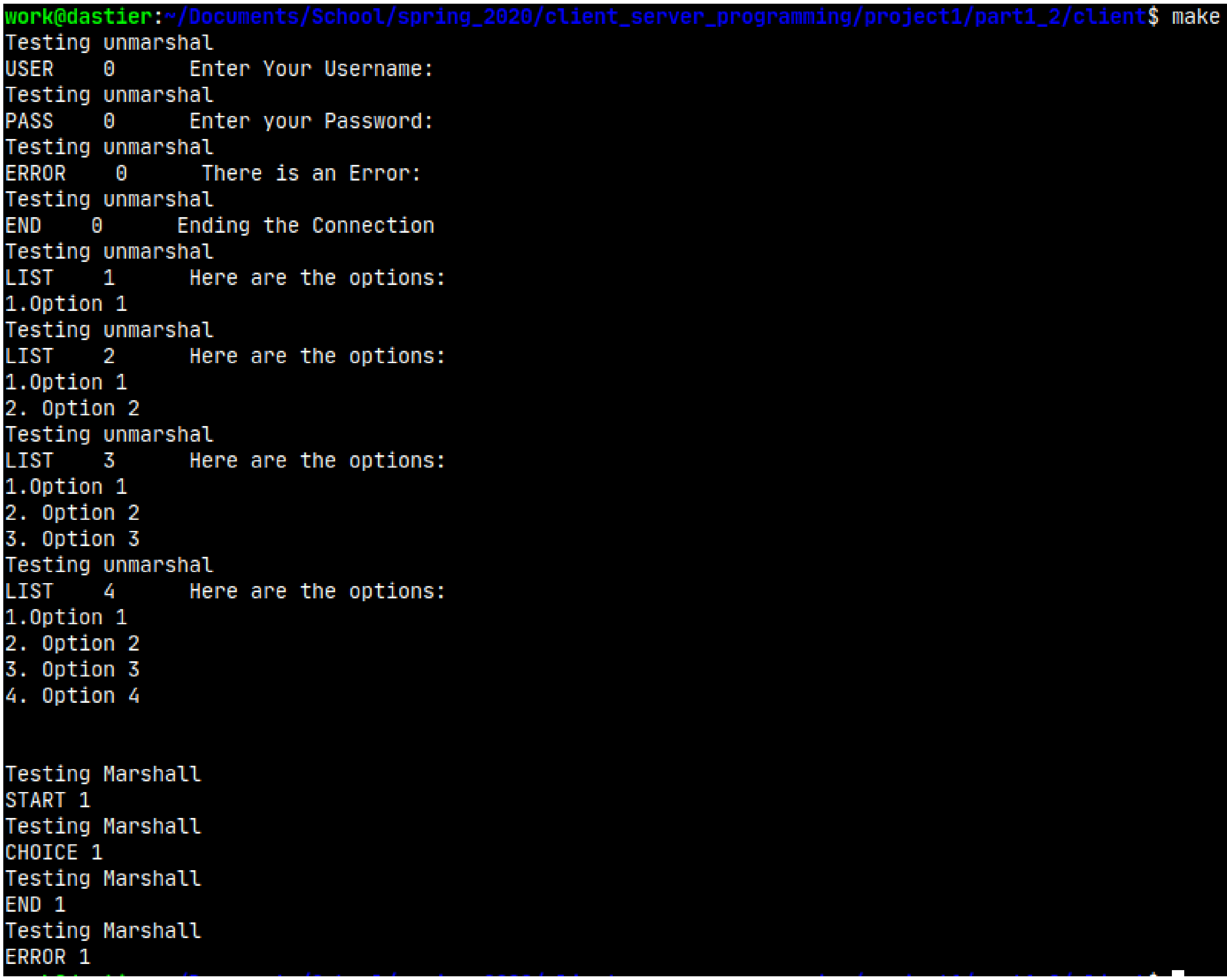
```
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include "client_marshal.h"
using namespace std;
int main(int argc,char*argv[])
{
if(argc==2)
{
cout<<"Testing unmarshal"<<endl;
server_message test1;
test1 =unmarshal(argv[1]);
test1.printing();
}

if(argc>2)
{
cout<<"Testing Marshall"<<endl;
client_message test;
test.decision=atoi(argv[2]);
test.options=argv[1];
cout<<marshal(test)<<endl;
}
return 0;
}
```

9.3.3 Makefile

```
all:client server_marshal.h client_marshal.h
echo Testing the client
client:client.cpp client_marshal.h
g++ client.cpp -o client
test_client: client client.cpp client_marshal.h
@./client "USER Enter Your Username:"
@./client "PASS Enter your Password:"
@./client "ERROR There is an Error:"
@./client "END Ending the Connection"
@./client "LIST Here are the options: \ 1 1.Option 1 \\"
@./client "LIST Here are the options: \ 2 1.Option 1 \ 2. Option 2 \\"
@./client "LIST Here are the options: \ 3 1.Option 1 \ 2. Option 2 \ 3. Option 3 \\"
@./client "LIST Here are the options: \ 4 1.Option 1 \ 2. Option 2 \ 3. Option 3 \ 4. Option 4 \\"
@ echo
@ echo
@./client "START" "1"
@./client "CHOICE" "1"
@./client "END" "1"
@./client "ERROR" "1"
clean:
rm client
```

9.4 Results



```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_2/client$ make
Testing unmarshal
USER    0       Enter Your Username:
Testing unmarshal
PASS    0       Enter your Password:
Testing unmarshal
ERROR    0       There is an Error:
Testing unmarshal
END    0       Ending the Connection
Testing unmarshal
LIST    1       Here are the options:
1.Option 1
Testing unmarshal
LIST    2       Here are the options:
1.Option 1
2. Option 2
Testing unmarshal
LIST    3       Here are the options:
1.Option 1
2. Option 2
3. Option 3
Testing unmarshal
LIST    4       Here are the options:
1.Option 1
2. Option 2
3. Option 3
4. Option 4


Testing Marshall
START 1
Testing Marshall
CHOICE 1
Testing Marshall
END 1
Testing Marshall
ERROR 1
```

Figure 19: Results for the client marshalling

9.5 Server Marshal

9.5.1 Header File

```
#ifndef _SERVER_MARSHAL_H_
#define _SERVER_MARSHAL_H_
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <vector>
using namespace std;
```

```
class client_message
{
public:
string command;
int decision;
void printing()
{
cout<<command<<"    "<<decision<<endl;
}
};

class server_message
{
public:
string command;
string messa;
vector<string> lines;
int num_lines;
};

struct samples_serverMessage
{
server_message userNameResponse;
server_message passWordResponse;
server_message create_userNameResponse()
{
userNameResponse.command="USER";
userNameResponse.messa="Please Enter your username:";
return userNameResponse;
};

server_message create_passWordResponse()
{
passWordResponse.command="PASS";
passWordResponse.messa="Please Enter your password:";
return passWordResponse;
};
};


client_message unmarshal(string message)
{
string command;
int decision;
client_message res;
istringstream ss(message);
ss>>command>>decision;
res.command=command;
res.decision=decision;
return res;
}

string marshal(server_message sm)
{
stringstream ss;
string result;
string c[5]={"USER","PASS","LIST","ERROR","END"};
if(sm.command == c[0])
{
ss<<sm.command<<" "<<sm.messa<<" ";
}
else if(sm.command == c[1])
{
ss<<sm.command<<" "<<sm.messa<<" ";
}
else if(sm.command == c[2])
{
ss<<sm.command<<" "<<sm.messa<<" "<<"\\"<<" "<<sm.num_lines;
for(int i=0;i<sm.num_lines;i++)
{
ss<<" "<<sm.lines[i]<<" "<<"\\";
}
}
else if(sm.command == c[3])
{
ss<<sm.command<<" "<<sm.messa;
}
else if(sm.command == c[4])
{
ss<<sm.command<<" "<<sm.messa;
}
else
{
}
getline(ss,result);
return result;
}
#endif
```

9.5.2 Test program

```
#include "server_marshal.h"
using namespace std;

int main(int argc,char*argv[])
{
if(argc<3)
{
cout<<"Testing unmarshal"<<endl;
client_message test=unmarshal(argv[1]);
test.printing();
}
else
{
cout<<"Testing Marshal"<<endl;
server_message test1;
test1.command=argv[1];
test1.messa=argv[2];
switch(argc)
{
case 5:
{
test1.num_lines=atoi(argv[3]);
test1.lines.push_back(argv[4]);
break;
}
case 6:
```

```
{
test1.num_lines=atoi(argv[3]);
test1.lines.push_back(argv[4]);
test1.lines.push_back(argv[5]);
break;
}
case 7:
{
test1.num_lines=atoi(argv[3]);
test1.lines.push_back(argv[4]);
test1.lines.push_back(argv[5]);
test1.lines.push_back(argv[6]);
break;
}
}
cout<<marshal(test1)<<endl<<endl;
}
return 0;
}
```

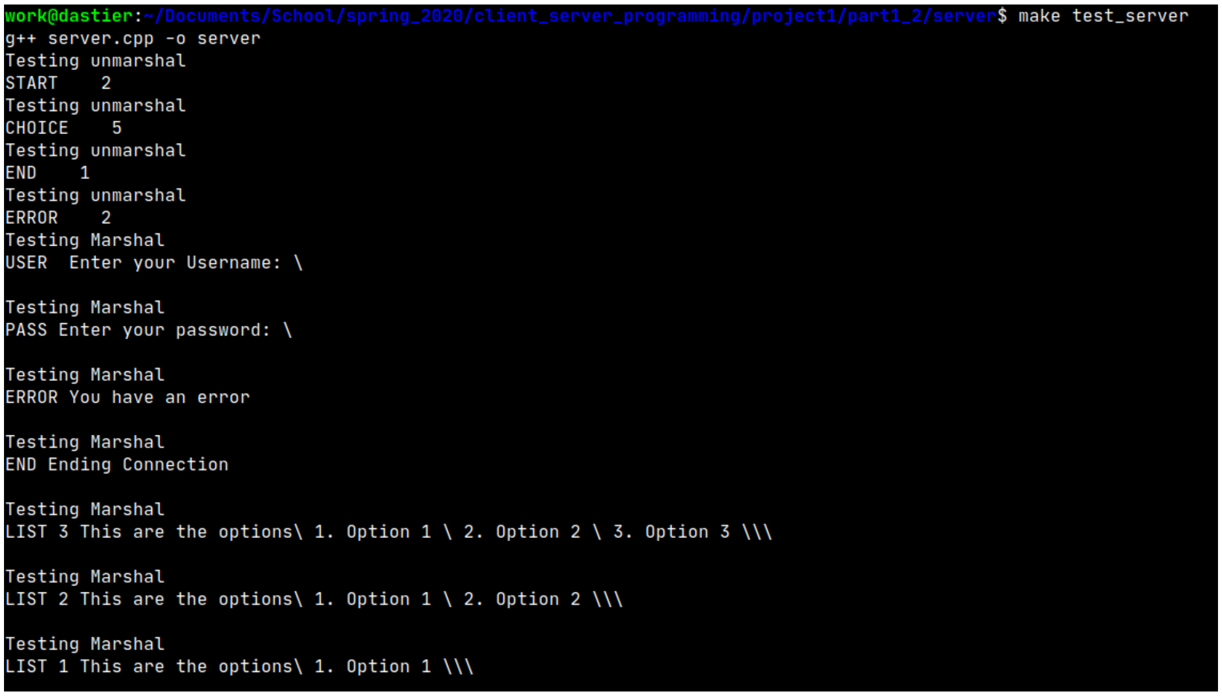
9.5.3 Makefile

```
all:server  server_marshal.h
echo Testing the client

server:server.cpp server_marshal.h
g++ server.cpp -o server

test_server: server server.cpp server_marshal.h
@./server "START 2"
@./server "CHOICE 5"
@./server "END 1"
@./server "ERROR 2"
@./server "USER" " Enter your Username: "
@./server "PASS" "Enter your password: "
@./server "ERROR" "You have an error "
@./server "END" "Ending Connection"
@./server "LIST" "This are the options" "3" " 1. Option 1 " " 2. Option 2 " " 3. Option 3 "
@./server "LIST" "This are the options" "2" " 1. Option 1 " " 2. Option 2 "
@./server "LIST" "This are the options" "1" " 1. Option 1 "
clean:
rm server
```

9.5.4 Result



```
work@dastier:~/Documents/School/spring_2020/client_server_programming/project1/part1_2/server$ make test_server
g++ server.cpp -o server
Testing unmarshal
START    2
Testing unmarshal
CHOICE    5
Testing unmarshal
END    1
Testing unmarshal
ERROR    2
Testing Marshal
USER  Enter your Username: \

Testing Marshal
PASS Enter your password: \

Testing Marshal
ERROR You have an error

Testing Marshal
END Ending Connection

Testing Marshal
LIST 3 This are the options\ 1. Option 1 \ 2. Option 2 \ 3. Option 3 \\\

Testing Marshal
LIST 2 This are the options\ 1. Option 1 \ 2. Option 2 \\\

Testing Marshal
LIST 1 This are the options\ 1. Option 1 \\\
```

Figure 20: Results for my message protocol

10 TCP Protocols

10.1 Client

10.1.1 Header File

```
#ifndef _TCP_CLIENT2_H_
#define _TCP_CLIENT2_H_
#include <iostream>
#include <string>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#define DEFAULT_PORT 12345
#define BACKLOG 10
#define MAXBUFFERLEN 4096
#define DEFAULT_IP "127.0.0.1"
using namespace std;

class client
{
private:
int sockfd;
struct sockaddr_in their_addr;
int connecting;
int sending;
int receiving;
public:
client();
client(string serverIP,int serverPort);
int initialize_client();
int send_message(string message);
string receive_message();
int closeSockets();
};

client::client()
{
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
perror("Failed to create socket");
exit(EXIT_FAILURE);
}
//Configure the server address
string serverAddress=DEFAULT_IP;
their_addr.sin_family=AF_INET;
their_addr.sin_port=htons((short)DEFAULT_PORT);
inet_pton(AF_INET,serverAddress.c_str(),&their_addr.sin_addr);
memset(&(their_addr.sin_zero),'\0',8);
}

client::client(string serverIP,int serverPort)
{
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
perror("Failed to create socket");
exit(EXIT_FAILURE);
}
//Configure the server address
string serverAddress=serverIP;
their_addr.sin_family=AF_INET;
their_addr.sin_port=htons((short)serverPort);
inet_pton(AF_INET,serverAddress.c_str(),&their_addr.sin_addr);
memset(&(their_addr.sin_zero),'\0',8);
}


int client::initialize_client()
{
connecting=connect(sockfd,(struct sockaddr *)&their_addr,sizeof(struct sockaddr));
if(connecting==-1)
{
perror("Failed to connect to server");
exit(EXIT_FAILURE);
close(sockfd);
return -1;
}
printf("Connected to %s in port %i \n",inet_ntoa(their_addr.sin_addr),ntohs(their_addr.sin_port));
return 1;
}
```

```
int client::send_message(string message)
{
char buffer[MAXBUFFERLEN];
FILE *stream;
int tn=message.size();
char temp[tn+1];
int num_char_read;
strcpy(temp,message.c_str());
stream=fmemopen(temp,strlen(temp),"r");
num_char_read=fread(buffer+1,sizeof(char),sizeof(buffer),stream);
buffer[0]=num_char_read;
int sending=send(sockfd,(const char*)buffer,strlen(buffer)+1,0);
if(sending==-1)
{
perror("Failure to send Package");
exit(EXIT_FAILURE);
close(sockfd);
close(sockfd);
return -1;
}
return 1;
}

string client::receive_message()
{
char buffer[MAXBUFFERLEN];
receiving=recv(sockfd,(char*)buffer,MAXBUFFERLEN,0);
if(receiving==-1)
{
perror("Failed to received package");
exit(EXIT_FAILURE);
close(sockfd);
close(sockfd);
}
string temp(buffer);
return temp;
memset(buffer,'\0',MAXBUFFERLEN);
}

int client::closeSockets()
{
close(sockfd);
return 1;
}
#endif
```

10.2 Server

10.2.1 Header File

```
#ifndef _TCP_CLIENT2_H_
#define _TCP_CLIENT2_H_
#include <iostream>
#include <string>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
```

```
#include<arpa/inet.h>
#include<netdb.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#define DEFAULT_PORT 12345
#define BACKLOG 10
#define MAXBUFFERLEN 2048
using namespace std;

class server
{
private:
int sockfd;
int new_fd;
struct sockaddr_in my_addr;
struct sockaddr_in their_addr;
int binding;
int listening;
int receiving;
int sending;
unsigned int sin_size;
public:
server();
server(int serverPort);
int initialize_server();
int send_message(string message);
string receive_message();
int closeSockets();
};

//Creating the default constructor
server::server()
{
//Initiate the socket
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
perror("Failed to create socket");
exit(EXIT_FAILURE);
}
//Configure my address
my_addr.sin_family=AF_INET;
my_addr.sin_port=htons((short)DEFAULT_PORT);
my_addr.sin_addr.s_addr=INADDR_ANY;
memset(&(my_addr.sin_zero),'\0',8);
//Bind the socket
binding=bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct sockaddr));
if(binding==-1)
{
perror("Failed to bind");
exit(EXIT_FAILURE);
}
}

server::server(int serverPort)
{
//Initiate the socket
sockfd=socket(AF_INET,SOCK_STREAM,0);
```

```
if(sockfd==-1)
{
perror("Failed to create socket");
exit(EXIT_FAILURE);
}
//Configure my address
my_addr.sin_family=AF_INET;
my_addr.sin_port=htons((short)serverPort);
my_addr.sin_addr.s_addr=INADDR_ANY;
memset(&(my_addr.sin_zero),'\0',8);
//Bind the socket
binding=bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct sockaddr));
if(binding==-1)
{
perror("Failed to bind");
exit(EXIT_FAILURE);
close(sockfd);
}
}

int server::initialize_server()
{
listening=listen(sockfd,BACKLOG);
sin_size=sizeof(struct sockaddr_in);
if(listening==-1)
{
perror("Failed to listen");
exit(EXIT_FAILURE);
close(sockfd);
return -1;
}
new_fd=accept(sockfd,(struct sockaddr *)&their_addr,&sin_size);
if(new_fd==-1)
{
perror("Failed to accept connection");
exit(EXIT_FAILURE);
close(sockfd);
return -1;
}
printf("Connected to %s in port %i \n",inet_ntoa(their_addr.sin_addr),ntohs(their_addr.sin_port));
return 1;
}

string server::receive_message()
{
FILE *stream;
char *bp;
size_t size;
char buffer[MAXBUFFERLEN];
stream=open_memstream(&bp,&size);
receiving=recv(new_fd,(char*)buffer,MAXBUFFERLEN,0);
if(receiving==-1)
{
perror("Failed to received package");
exit(EXIT_FAILURE);
close(sockfd);
close(new_fd);
}
fwrite(buffer+1,sizeof(char),buffer[0],stream);
```

```
fflush(stream);
string temp(bp);
return temp;
}


int server::send_message(string message)
{
char buffer[MAXBUFFERLEN];
strcpy(buffer,message.c_str());
int sending=send(new_fd,(const char*)buffer,strlen(buffer)+1,0);
if(sending==-1)
{
perror("Failure to send Package");
exit(EXIT_FAILURE);
close(sockfd);
close(new_fd);
return -1;
}
return 1;
}

int server::closeSockets()
{
close(sockfd);
close(new_fd);
return 1;
}
#endif
```

11 Menus

11.1 Client

11.1.1 Client Main

```
#include <iostream>
#include "tcp_client2.h"
#include "client_marshal.h"
using namespace std;
int main()
{
int userInput;
//Start Client
client test;
test.initialize_client();
//Creating sample client message
client_message starting;
starting.options="START";
starting.decision=1;
//Marshal the client message
string sendMessage=marshal(starting);
//Send starting message
test.send_message(sendMessage);
while(true)
{
string recMessage;
string sendingMessage;
//Receive message
recMessage=test.receive_message();
server_message servMessage;
//Unmarshall the message
servMessage=unmarshal(recMessage);
//Print the Message
servMessage.printing();
if(servMessage.options=="END")
{
break;
}
//Ask the user for input
cin>>userInput;
client_message userChoice;
userChoice.options="CHOICE";
userChoice.decision=userInput;
sendingMessage=marshal(userChoice);
test.send_message(sendingMessage);
}
test.closeSockets();
return 0;
}
```

11.2 Server

11.2.1 Server Main

```
#include <iostream>
#include <string>
#include <vector>
#include "server_marshal.h"
#include "tcp_server2.h"
#include "server_menus.h"
using namespace std;

int main()
{
string receiveStartingMessage;
//Initialize server
server test;
test.initialize_server();
//Receive
receiveStartingMessage=test.receive_message();
//Unmarshall the message from the client
client_message startingMessage;
startingMessage=unmarshal(receiveStartingMessage);
//Check if you have the starting message
if(startingMessage.command!="START")
{
test.closeSockets();
exit(EXIT_SUCCESS);
}
// Create the sample
samples_serverMessage temp;
server_message userServMess;
server_message passServMess;
userServMess=temp.create_userNameResponse();
string userMessage=marshal(userServMess);
//Password
passServMess=temp.create_passWordResponse();
string passMessage=marshal(passServMess);
string userInput;
while(true)
{
//Send username response
test.send_message(userMessage);
//Receive User Response
userInput=test.receive_message();
client_message tempUserMessage;
tempUserMessage=unmarshal(userInput);
//Send Password
test.send_message(passMessage);
userInput=test.receive_message();
client_message tempPassMessage;
tempPassMessage=unmarshal(userInput);
if(tempPassMessage.decision==45 && tempUserMessage.decision==7)
{
cout<<"You logged in"<<endl;
int mmMenu=mainMenu(test);
if(mmMenu==-1)
{
test.closeSockets();

break;
}
}
}

return 0;
}
```

11.2.2 Server Menus

```
#ifndef _SERVER_MENUS_H_
#define _SERVER_MENUS_H_
#include <iostream>
#include <string>
#include "server_marshal.h"
#include "tcp_server2.h"
#include "datamodel.h"
using namespace std;

home sample;
server_message devicesLists()
{
vector<string> temp=sample.deviceList;
temp.push_back("4. Return to the Previous Menu");
server_message deviceListMess;
deviceListMess.command="LIST";
deviceListMess.messa="List of Devices";
deviceListMess.num_lines=temp.size();
deviceListMess.lines=temp;
return deviceListMess;
}

server_message roomsList()
{
vector<string> temp=sample.ro.roomList;
```

```
int tempNum=sample.ro.obtainNumberofRooms();
stringstream ss;
string tempAA;
ss<<tempNum+1<<". "<<"Return to the previous Menu";
getline(ss,tempAA);
temp.push_back(tempAA);
server_message roomListMess;
roomListMess.command="LIST";
roomListMess.messa="List of Rooms";
roomListMess.num_lines=temp.size();
roomListMess.lines=temp;
return roomListMess;
}

server_message listsLock(int option)
{
vector<string> temp=sample.lo.locksList;
int tempNum=sample.lo.numLocks;
stringstream ss;
if(option==1)
{
stringstream ss1;
string tempAA1;
tempNum++;
ss1<<tempNum<<". "<<"Change the status of all the locks";
getline(ss1,tempAA1);
temp.push_back(tempAA1);
}
string tempAA;
ss<<tempNum+1<<". "<<"Return to the Previous Menu";
getline(ss,tempAA);
temp.push_back(tempAA);
server_message listsLockMess;
listsLockMess.command="LIST";
listsLockMess.messa="Welcome to Locks";
listsLockMess.num_lines=temp.size();
listsLockMess.lines=temp;
return listsLockMess;
}

server_message lightList(int option1,int type)
{
vector<string> temp=sample.ro.house_room[option1-1].lightsList;
int tempNum=sample.ro.house_room[option1-1].numLights;
stringstream ss;
if(type==1)
{
stringstream ss1;
string tempAA1;
tempNum++;
ss1<<tempNum<<". "<<"Change the status of all lights";
getline(ss1,tempAA1);
temp.push_back(tempAA1);
}
string tempAA;
ss<<tempNum+1<<". "<<"Return to the previous Menu";
getline(ss,tempAA);
temp.push_back(tempAA);
server_message lightListMess;
```

```
lightListMess.command="LIST";
lightListMess.messa=sample.ro.house_room[option1-1].roomName;
lightListMess.num_lines=temp.size();
lightListMess.lines=temp;
return lightListMess;
}


server_message alarmStatus(int option1)
{
vector<string> temp;
string tempAlarm;
bool alarmStatus=sample.al.getStatus();
if(alarmStatus)
{
tempAlarm="ON";
}
else
{
tempAlarm="OFF";
}
string alarmSt="Alarm";
string temp1;
stringstream ss;
int number=1;
ss<<number<<". "<<alarmSt<<number<<":"<<tempAlarm;
getline(ss,temp1);
temp.push_back(temp1);
if(option1==1)
{
temp.push_back("2. Change the status");
}
temp.push_back("4. Return to the Previous Menu");
server_message alarmStatusMessage;
alarmStatusMessage.command="LIST";
alarmStatusMessage.num_lines=temp.size();
alarmStatusMessage.messa="Check Status: Alarm";
alarmStatusMessage.lines=temp;
return alarmStatusMessage;
}
//Status
//Alarm Menu
//
//
//

int alarmMenu(server test,int option1)
{
server_message options1;
options1.command="LIST";
options1.messa="Enter Pin";
options1.num_lines=1;
options1.lines.push_back("Integers only");
string maOptions1=marshal(options1);

string userInput;
server_message alarmMenuMess;
while(true)
{
```

```
alarmMenuMess=alarmStatus(option1);
string maAlarmMenu=marshal(alarmMenuMess);
test.send_message(maAlarmMenu);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(tempMessage.decision==4)
{
break;
}
else if(tempMessage.decision==2)
{
test.send_message(maOptions1);
userInput=test.receive_message();
tempMessage=unmarshal(userInput);
sample.al.changeStatus(tempMessage.decision);
}
else
{
}
}
return -1;
}

//Light Menu
int lightMenu(server test,int option,int type)
{
string userInput;
while(true)
{
server_message lightMenuMess=lightList(option,type);
string maLightMenuMess=marshal(lightMenuMess);
int tempNum=sample.ro.house_room[option-1].numLights;
test.send_message(maLightMenuMess);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(type==0)
{
if(tempMessage.decision==(tempNum+1))
{
break;
}
}
else
{
if(tempMessage.decision==(tempNum+2))
{
break;
}
else if(tempMessage.decision==(tempNum+1))
{
sample.ro.house_room[option-1].changeAllLights();
}
else
{
for(int i=1;i<tempNum+1;i++)
{
if(tempMessage.decision==i)
{
sample.ro.house_room[option-1].changeSingleLights(i-1);
}
}
}
}

}
return -1;
}

//Room Menu
//
//
//
int roomMenu(server test,int type)
{
string userInput;
server_message roomMenuMess;
while(true)
{
roomMenuMess=roomsList();
string maRoomMenuMess=marshal(roomMenuMess);
int tempNum=sample.ro.obtainNumberofRooms();
test.send_message(maRoomMenuMess);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(tempMessage.decision==(tempNum+1))
{
break;
}
else
{
for(int i=1;i<tempNum+1;i++)
{
if(tempMessage.decision==i)
{
int llLightMenu;
llLightMenu=lightMenu(test,i,type);
}
}
}
}
return -1;
}


//Locks
//
//
//
int lockMenu(server test,int option)
{
server_message options1;
options1.command="LIST";
options1.messa="Enter Pin";
options1.num_lines=1;
options1.lines.push_back("Integers only");
```

```
string maOptions1=marshal(options1);

string userInput;
while(true)
{
server_message lockMenuMess=listsLock(option);
string maLockMenuMess=marshal(lockMenuMess);
int tempNum=sample.lo.numLocks;
test.send_message(maLockMenuMess);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(option==0)
{
if(tempMessage.decision==(tempNum+1))
{
break;
}
}
else
{
if(tempMessage.decision==(tempNum+1))
{
test.send_message(maOptions1);
userInput=test.receive_message();
tempMessage=unmarshal(userInput);
sample.lo.changeAllLocks(tempMessage.decision);
}
else if(tempMessage.decision==(tempNum+2))
{
break;
}
else
{
for(int i=1;i<tempNum+1;i++)
{
if(tempMessage.decision==i)
{
test.send_message(maOptions1);
userInput=test.receive_message();
tempMessage=unmarshal(userInput);
sample.lo.changeSingleLock(i-1,tempMessage.decision);
}

}
}
}
}
return -1;
}



//Check Status Menu
//
//
//

int checkStatusMenu(server test)
```

```
{
string userInput;
server_message deviceListMess;
deviceListMess=devicesLists();
string maDeviceListMess=marshal(deviceListMess);
while(true)
{
test.send_message(maDeviceListMess);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(tempMessage.decision==1)
{
int aaAlarmSt;
aaAlarmSt=alarmMenu(test,0);
}
else if(tempMessage.decision==2)
{
int llLockMenu;
llLockMenu=lockMenu(test,0);
}
else if(tempMessage.decision==3)
{
int rrRoomSt;
rrRoomSt=roomMenu(test,0);
}
else if(tempMessage.decision==4)
{
break;
}
else
{
continue;
}
}
return -1;

}



//Change Status Menu
//
//
//
int changeStatusMenu(server test)
{
string userInput;
server_message deviceListMess;
deviceListMess=devicesLists();
string maDeviceListMess=marshal(deviceListMess);
while(true)
{
test.send_message(maDeviceListMess);
userInput=test.receive_message();
client_message tempMessage;
tempMessage=unmarshal(userInput);
if(tempMessage.decision==1)
{
```

```
int aaAlarmSt;
aaAlarmSt=alarmMenu(test,1);

}
else if(tempMessage.decision==2)
{
int llLockMenu;
llLockMenu=lockMenu(test,1);
}
else if(tempMessage.decision==3)
{
int rrRoomSt;
rrRoomSt=roomMenu(test,1);
}
else if(tempMessage.decision==4)
{
break;
}
else
{
continue;
}
}
return -1;
}



//Main Menu
//
//
//
int mainMenu(server test)
{
string maMenu;
server_message cm;
cm.command="LIST";
cm.messa="Main Menu";
cm.num_lines=3;
cm.lines.push_back("1. Check Status");
cm.lines.push_back("2. Change status");
cm.lines.push_back("3. Logout");
maMenu=marshal(cm);
server_message ending;
ending.command="END";
ending.messa="Ending Connection";
string maEnding=marshal(ending);

server_message options1;
options1.command="LIST";
options1.messa="You entered option 1";
options1.num_lines=1;
options1.lines.push_back("Welcome to the check Status");
string maOptions1=marshal(options1);

server_message options2;
options2.command="LIST";
options2.messa="You entered option 1";
options2.num_lines=1;
```

```
options2.lines.push_back("Welcome to the check Status");
string maOptions2=marshal(options2);
while(true)
{
string userInput;
test.send_message(maMenu);
client_message tempMessage;
userInput=test.receive_message();
tempMessage=unmarshal(userInput);
if(tempMessage.decision==1)
{
int ccCheck;
ccCheck=checkStatusMenu(test);
}
else if(tempMessage.decision==2)
{
int ccChange;
ccChange=changeStatusMenu(test);
}
else if(tempMessage.decision==3)
{
test.send_message(maEnding);
break;
}
}
return -1;
}
#endif
```