ECE470:Network Client-Sever Programming

Project 1: Part 1

Aaron Jesus Valdes

February 23, 2021

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.

Specifications

- Supports smart lights
  - Can turn on and off lights
  - Can change the color and brightness of lights
  - 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
- Supports Electronic Locks
  - Can open and lock the locks
  - Can have more than 4 locks
  - Can store a 5 digit pin per lock
- Supports more devices
  - Can control the thermostat
    Turn on or off heat/cold
    Set temperature
  - Can check the security cameras
    Can store the video from a security camera
  - Can control blinds and drapes
    Opens and closes the blinds

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

Initial Design

Data Model

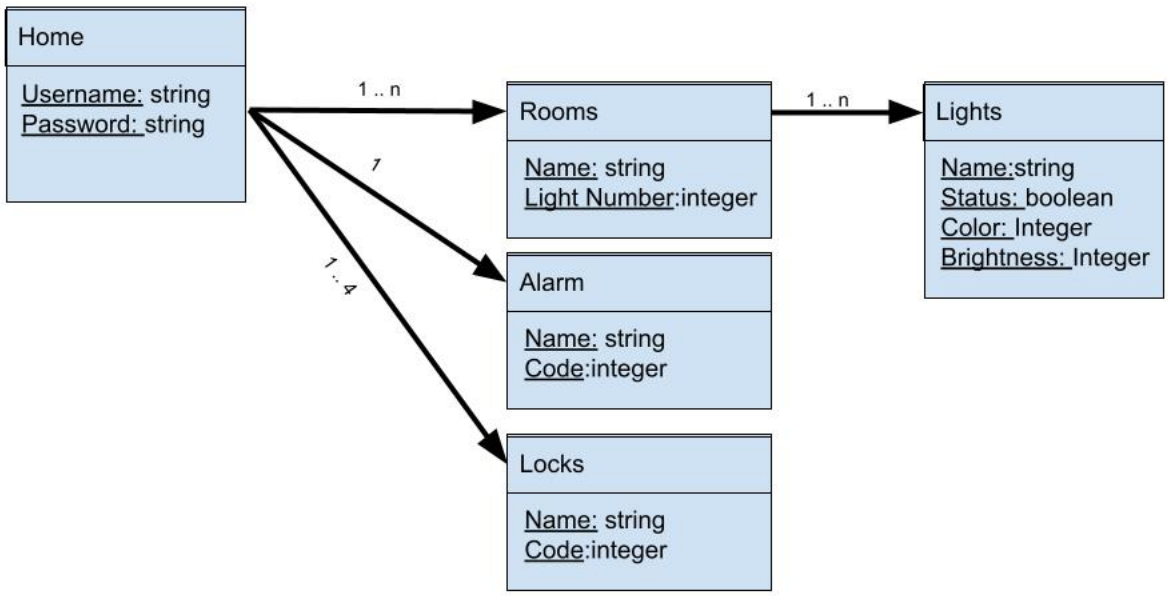


Figure 1: Data models follow the specifications

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

Business Logic

Basic Operations

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

Storyboard

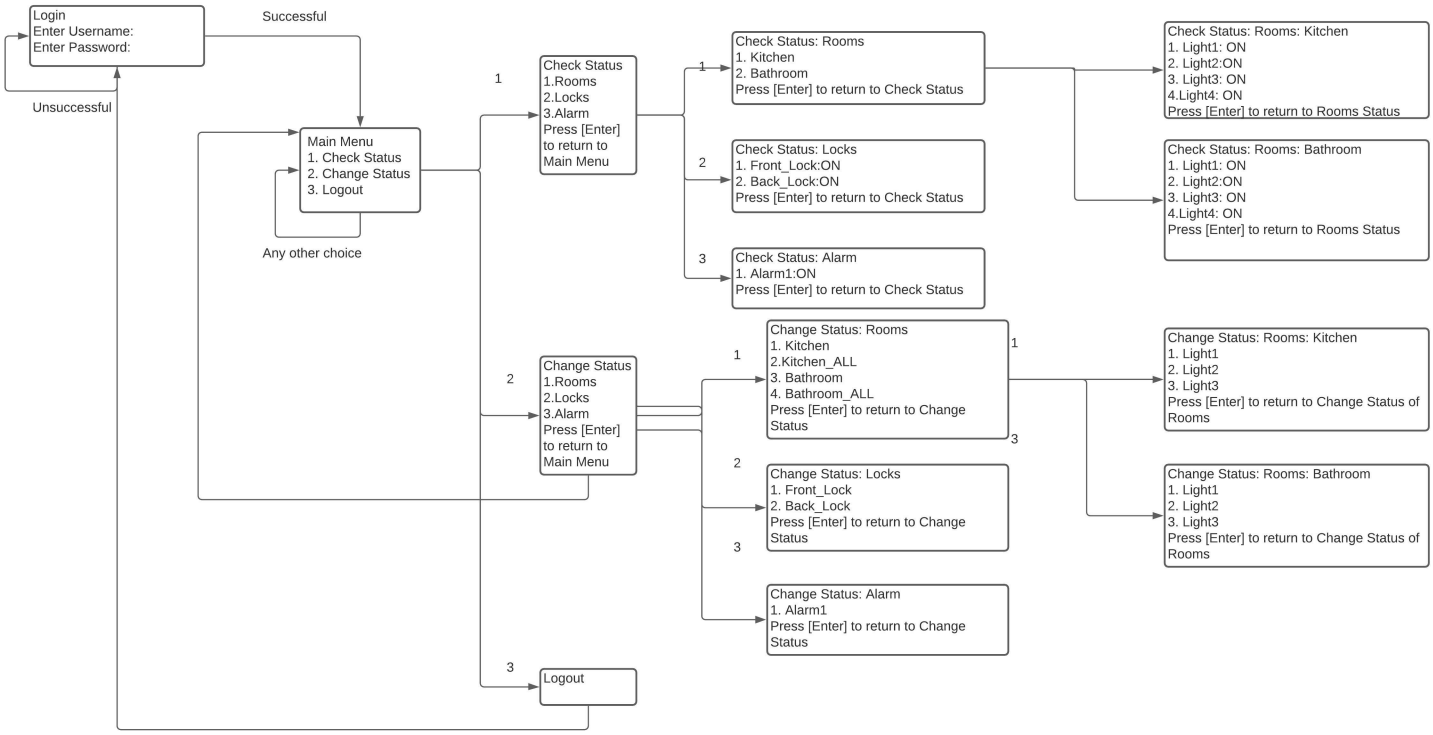


Figure 2:

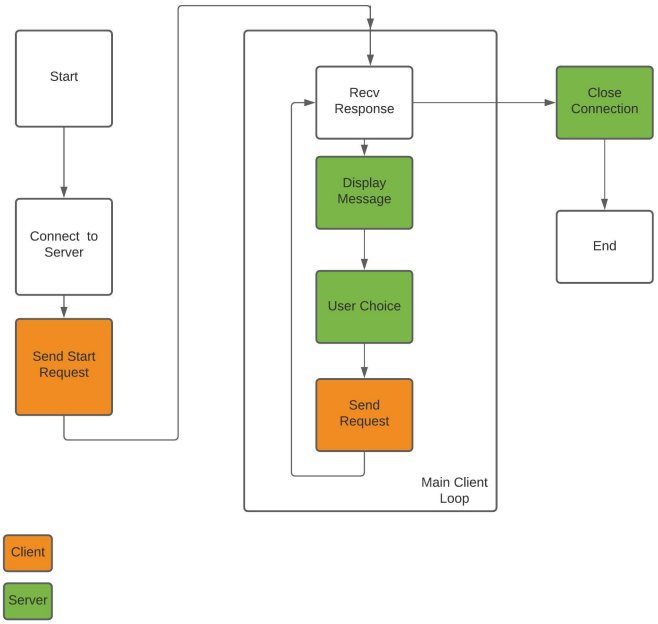
The reason I chose to divide my check status and change status based on devices is because that is the method that my google home uses show the status of my devices and change the status of my devices. In my opinion the most effective method is by simply having a single main menu with a all the options since it leads to less amount of packets being transferred back and forth between the client and the server and can therefore reduce 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

Sever state chart

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.

Client State Chart



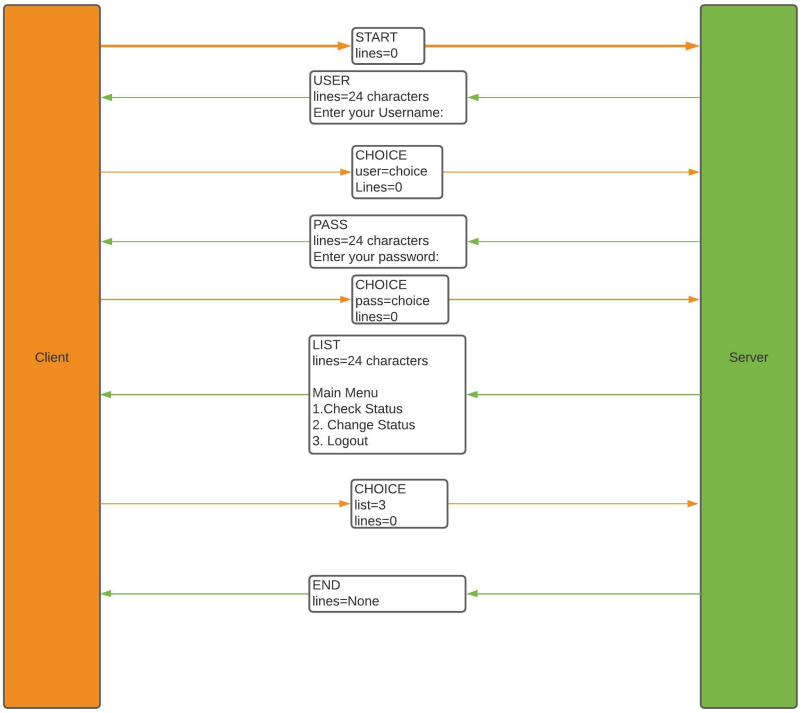
The client state diagram starts by simply establishing a connection between the server and the client. Then the server sends the client a message,which the user uses to make a request. After the request is sent to the server, the server makes a decision based on that request and executes a command. The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

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

Example



1 Code for the Data model

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

class light
{
public:
string name;
bool status;
};




class room
{
public:
vector<light> l;
};

```

```
24
25          class rooms
26          {
27          public:
28          vector<room> r;
29          string name;
30          int light_number;
31          };
32
33
34          class alarm
35          {
36          public:
37          string name;
38          int code;
39          };
40
41
42
43          class lock
44          {
45          public:
46          string name;
47          int code;
48          };
49
50
51
52          class locks
53          {
54          public:
55          vector<lock> llock;
56          };
57
58
59          class home
60          {
61          private:
62          string password;
63          string username;
64          public:
65          rooms n;
66          alarm a;
67          locks lo;
68          void setUsername(string newUsername);
69          void setPassword(string password);
70          };
71          #endif
72
```

2 Code for the Sever marshal

```
1       #ifndef _SERVER_MARSHAL_H_
2       #define _SERVER_MARSHAL_H
3       #include <iostream>
4       #include <string>
5       #include <stdio.h>
6       #include <stdlib.h>
7       #include <sstream>
8       #include <vector>
9       using namespace std;
10
11      class client_message
12      {
13      public:
14      string command;
15      int decision;
16      void printing()
17      {
18      cout<<command<<"    "<<decision<<endl;
19      }
20      };
21
22      class server_message
23      {
24      public:
25      string command;
26      string messa;
27      vector<string> lines;
28      int number;
29
30      };
31
32      client_message unmarshal(string message)
33      {
34      string command;
35      int decision;
36      client_message res;
37      istringstream ss(message);
38      ss>>command>>decision;
39      res.command=command;
40      res.decision=decision;
41      return res;
42      }
43
44      string marshal(server_message sm)
45      {
46      stringstream ss;
47      string result;
48      string c[5]={"USER","PASS","LIST","ERROR","END"};
```

```
49      if(sm.command == c[0])
50      {
51      ss<<c[0]<<" "<<sm.messa;
52      }
53      else if(sm.command == c[1])
54      {
55      ss<<c[1]<<" "<<sm.messa;
56      }
57      else if(sm.command == c[2])
58      {
59      ss<<c[2]<<" ";
60      for(unsigned int i=0;i<sm.lines.size();i++)
61      {
62      ss<<sm.lines[i]<<" ";
63      }
64      ss<<"\\\\";
65      }
66      else if(sm.command == c[3])
67      {
68      ss<<c[3]<<" "<<sm.messa;
69      }
70      else if(sm.command == c[4])
71      {
72      ss<<c[4]<<" "<<sm.messa;
73      }
74      else
75      {
76      }
77      ss>>result;
78      return result;
79      }
80      #endif
81
```

3 Code for the client marshal

```
1       #ifndef _CLIENT_MARSHAL_H_
2       #define _CLIENT_MARSHAL_H
3       #include <iostream>
4       #include <string>
5       #include <stdio.h>
6       #include <stdlib.h>
7       #include <sstream>
8       #include <vector>
9       using namespace std;
10
11      class server_message
12      {
13      public:
14      string options;
15      string message;
```

```
16      vector<string> lines;
17
18      };
19
20      class client_message
21      {
22      public:
23      string options;
24      int decision;
25      };
26
27      server_message unmarshal(string message)
28      {
29      string command;
30      server_message res;
31      istringstream ss(message);
32      ss>>command;
33      if(command=="USER")
34      {   res.options="USER";
35      ss>>res.message;
36      }
37      else if(command=="PASS")
38      {
39      res.options="PASS";
40      ss>>res.message;
41      }
42      else if(command=="LIST")
43      {
44      res.options="PASS";
45      while(ss>>res.message)
46      {
47      if(res.message=="\\\\")
48      {
49      break;
50      }
51      else
52      {
53      res.lines.push_back(res.message);
54      }
55      }
56      }
57      else if(command=="ERROR")
58      {
59      res.options="ERROR";
60      ss>>res.message;
61      }
62      else if(command=="END")
63      {
64      res.options="END";
65      ss>>res.message;
66      }
```

```
67      else
68      {
69      }
70      return res;
71      }
72
73
74      string marshal(client_message cm)
75      {
76      stringstream ss;
77      string result;
78      if(cm.options=="START")
79      {
80      ss<<cm.options;
81      }
82      else if(cm.options=="CHOICE")
83      {
84      ss<<cm.options<<" "<<cm.decision;
85      }
86      else if(cm.options=="END")
87      {
88      ss<<cm.options<<" ";
89      }
90      else if(cm.options=="ERROR")
91      {
92      ss<<cm.options;
93      }
94      else
95      {
96      }
97      ss>>result;
98      return result;
99      }
100     #endif
101
```