Secure Software Development (2020/21) CS2S562_2020_v1

Here is some information about my project:

I have used the first Secure logger pattern and also there are three type to secure logger which I have used too:

```
enum LoggerType { LggerTextFile = 1, LoggerBinFile, LoggerConsole };
enum FormateType { NoFORMAT = 1, HTMLFormate, ENCFormate };
class Logger
{
    unsigned formatType;
    string encryptDecrypt(string toEncrypt) {}
public:
    Logger(unsigned aformatType = NoFORMAT) :formatType(aformatType) {};
    virtual void log(string message) = 0;
    virtual ~Logger() {};
    virtual string formatMessage(string message) {};
};

class LoggerDecorator :public Logger
{
    unsigned formatType;
    unique_ptr<Logger>contents;
    void log(string message)
    {
        message = formatMessage(message);
        contents->log(message);
    }
```

```
class TxtFileLogger : public Logger
{
public:
    void log(string message)
    {
        cout << " Writing " << message << " to a text file\n";
    }
};
class ConsoleLogger : public Logger
{
public:
    void log(string message)
    {
        cout << " Writing " << message << "to console\n";
    }
};
class BinFileLogger : public Logger
{
public:
    void log(string message)
    {
        cout << " Writing " << message << " to a BIN file\n";
```

The advantage of using the structure is that allows to add new functionality to existing object without altering its structure as well as wraps the original class and provides additional functionality keeping class methods signature intact.

I have added a separate class called SimpleFileStearm to save output to a file:

```
};
class SimpleFileLogger
{
    strin
    ofstream outputStream;
public:
    SimpleFileLogger(const string& filePath) : filePath (filePath)
    {
        outputStream.open(filePath, std::ios::out);
    }
    ~SimpleFileLogger()
    {
        outputStream.close();
    }
    void log(const string& message)
    {
        outputStream << message << endl << flush;
    }
};
```

class SimpleFileLogger
Search Online

Here is my second pattern which is Authenticator & Authorization combined with proof of ID pattern and there are three different type of users:

```
class AuthenticationInfo
{
    map<string, string >Users;
public:
    AuthenticationInfo(void)
    {
        Users.insert(pair<string , string >("1234" , "1111"));
        Users.insert(pair<string, string >("2345", "2111"));
        Users.insert(pair<string, string >("3456", "3111"));
    }
    bool validateUser(string Id, string password)
    {
        bool validUser = false;
        map<string, string >::iterator it;
        it = Users.find(Id);
        if (it !=Users.end())
        {
            if (!(it->second.compare(password)))
            {
                validUser = true;
            }
        }
```

(const char [5])"3111"
Search Online

008722 Controller

```
bool User(string Id, string PassWord)
{
    bool validUser = false;
    if (!(Id.compare(userID)))
    {
        if (!(password.compare(PassWord)))
        {
            validUser = true;
        }
    }
    return validUser;
}

bool Login()
{
    AuthenticationInfo ainfo;
    cout << "*******User Id and Password ******* " << endl;
    cout<<"1234  "<<  "1111 "<<endl;
    cout << "2345  " << "2111 " << endl;
    cout << "3456  " << "3111 " << endl;
    cout << "********************************* " << endl;
```

2

```
uint16_t again = 1;

string Id = subject.getId();
string password;
cout << "\nEnter password :";
cin >> password;
while (again)
{
    subject.enterFirstName();
    subject.enterLastName();
    cout << "\nTry again? Yes = 1 , No  = 0 :  " << endl;
    cin >> again;
}

if (ainfo.validateUser(Id,password))
{
    cout << "\n Welcome!   " << subject.getFirstName() << "  " << subject.getLastName() << endl;
    cout << "\n Authenticated ! Logged in successfully  \n";
    authenticated = true;
}
else
```

2 Ln: 511 Ch: 22 Col: 34 MIXED CR

Here is the machine state function but I have decided to put the state of the machine is always on:

```
enum class MachineState { Off, On };

class ReportTemperature
{
public:
    MachineState currentState = MachineState::On;
    void On()
    {
        currentState = MachineState::On;
        cout << " the machine is on \n";
    }
    void Off()
    {
        currentState = MachineState::Off;
        cout << " the machine is off \n";
    }

    MachineState getState() const
    {
        return currentState;
```

0 2

I have created three sensors and they are Report the temperature, humidity and windspeed respectively:

```
class Temperature
{
    int temp{ 25 };
    int maxTemp{ 100 };
public:
    int read() const
    {
        return temp;
    }
    void write(int t)
    {
        temp = t;
    }
    void configure()
    {
        maxTemp = 100;
    }

}
```

```
class Humidity
{
    int hum{ 80 };
    int maxTemp{ 100 };
public:
    int read() const
    {
        return hum;
    }
    void write(int h)
    {
        hum = h;
    }
    void configure()
    {
        maxTemp = 100;
    }

};
```

```
class WindSpeed
{
    int speed{ 20 };
    int maxSpeed{ 100 };
public:
    int read() const
    {
        return speed;
    }
    void write(int S)
    {
        speed = S;
    }
    void configure()
    {
        maxSpeed = 100;
    }

};
```

Their functionality:

```
96      10      81      18
 the machine state  is always   ON

97      12      87      24
 the machine state  is always   ON

98      11      79      32
 the machine state  is always   ON

99       8      71      28
 the machine state  is always   ON

10       9      79      38
 the machine state  is always   ON


Introduction to the Air temperatur and Humidity and wind speed reporter device

Samrt device simulation
Select from the following options

1: Information about the device
2: Loging in
3: Read data
4: Configure
5: Reading data from a file
0: Quit
Your choice :
```

I have added a class to check the type of permission that allows the user to modify sensors:

```
ostream& operator<<(ostream& os, AccessType c)
{
    switch (c)
    {
    case NA: os << "No access";            break;
    case W:  os << "Write access";         break;
    case R:  os << "Read access";          break;
    case FA: os << "Full Access";          break;
    default: os.setstate(ios_base::failbit);
    }
    return os;
}
class Permission
{
    typedef vector<uint16_t> resources;
    map< string, resources> userResourceAccessMap;
public:
    Permission()
    {                                       //    0    1     2
        userResourceAccessMap["1234"] = { FA , FA,   FA,   FA,   FA,   FA };
        userResourceAccessMap["2345"] = { W ,   W,    W,    W,   W,   W };
        userResourceAccessMap["3456"] = { R,    R,    R,    R,    R,    R };
```

And here is a function to read data form a file:

```
{
    char buffer[20];
    try
    {
        ifstream myfile("DeviceInfo.log"); // open a text file for reading
        // ifstream does not have exceptions by default. Let's register 3 of them to the stream:
        myfile.exceptions(ifstream::eofbit | ifstream::failbit | ifstream::badbit);
        while (myfile)
        {
            myfile.getline(buffer, 20);
            cout << buffer << endl;
        }
        myfile.close();
    }
    catch (exception e)
    {
        // strstr: finds out if 2nd string is contained within 1st string.
        // Returns a pointer to occurrence of 2nd string in 1st string
        if (strstr(e.what(), "eofbit") != NULL)
        {
            cout << buffer << endl;
            cout << "END OF FILE REACHED" << endl;
```

Here is the main menu:

```
008722                    Controller                    run(void)
        choice = view.mainMenu();
        switch (choice)
        {
        case Choice::Information:
            view.ModelDetails(model);
            break;
        case Choice::login:
            if (logged) view.message("You are already logged in ! ");
            else logged = Login();
            break;
        case Choice::Read:
            readData();
            break;
        case Choice::Configure:
            // check the permission for the sensors;
            if (permission.checkPermission(subject.getId(), 0) == FA)configureSensor();
            //if (logged) configureSensor();
            else view.message("You do not have sufficient privileges to modify sensors");
            break;
        case Choice::ReadDataFromFile:
            cout << "=====================" << endl;
            cout << "readind data from a file " << endl;
```

```
C:\Users\hossa\OneDrive\Desktop\secure software\FirstCoursework(30008722)\firstCousewok_30008722\x64\Debug\firstCousewok_30008722

Introduction to the Air temperatur and Humidity and wind speed reporter device

Samrt device simulation
Select from the following options

1: Information about the device
2: Loging in
3: Read data
4: Configure
5: Reading data from a file
0: Quit
Your choice :
```