Design Documentation-Devices

HomeDork - Interactive Smart House

Project Members

Reference	Name	Email
A	Samuel Mcmurray	Samuel_joseph.mcmurray0004@stud.hkr.se
B	Mustafa Ismail	mustafa.ismail0007@stud.hkr.se
C	Ibrahim Ahmed Ali	ibrahim.ahmed_ali0003@stud.hkr.se
D	Osayomore Edugie	Osayomore.edugie0004@stud.hkr.se

Revision History

Date	Version	Description	Author
04/10/2021	1.0	Initial Design	A, B, C, D
24/10/2021	1.1	Updated – All Classes Added Classes – Radiator, Twilight Automatic System, Window, Timer, Stove, Electric Consumption, Power Cut Off, Water Leakage, Alarm Controller, Sensor Controller and Device Controller	A, B, C, D
14/11/2021	1.2	Updated – All class figures, some descriptions have been updated as well, Added – Sequence diagram device controller, State diagram device controller handling output devices	A, B, C, D
4/12/2021	1.3	Added – Sequence diagram sensor controller Updated – All Class diagram figures Use Case Diagram	A, B, C, D
05/01/2022	1.4	Added – Sequence diagram alarm controller Updated – All class diagrams, text, and the useCase Diagram	A, B, C, D

Design item List

Requirement Name	Priority
D1. The Overall Design	Essential
D2. Use Case Diagram	Essential
D3. Class Diagram – Overall	Essential
D4. Class Diagram – Abstract Devices and the classes that inherit.	Essential
D5. Class Diagram – The Alarm class and the devices that make up its composition	Essential
D6. Class Diagram – The Twilight Automatic System Class	Essential
D7. Class Diagram – The Temperature Controller	Essential
D8. Class Diagram – The Device Controller	Essential
D9. Class Diagram – The Sensor Controller	Essential
D10. Class Diagram – The Alarm Controller	Essential
D11. Class Diagram – Response Class/ Request Class	Essential
D12. Class Diagram – The Main Class	Essential
D13. Class Diagram – The HUB	
D14. Sequence Diagram – Read Temperature	Essential
D15. Sequence Diagram – Handle Command for Output Device	Essential
D16. Sequence Diagram – Device Controller	Essential
D17. Sequence Diagram – Temperature Controller	Essential
D18. Sequence Diagram – Sensor Controller	Essential
D19. Sequence Diagram – Alarm Controller	Essential
D20. State Diagram – Temperature Control	Essential

D21. State Diagram – Device Controller/ Output Device	Essential
	Essential/Desirable/Optional

Design Item Descriptions

D1

Overall design (Figure 1) black box here we have an example of turning on a light the user interacts with a UI from one of their devices. The request is sent to the API server, the server then makes a request to the Arduino. The Arduino handles the request in this case turn on a light and sends a response back to the API server as to the success or failure of the request. The API updates the database with the current state of the device receives a response and sends a response back to the user.

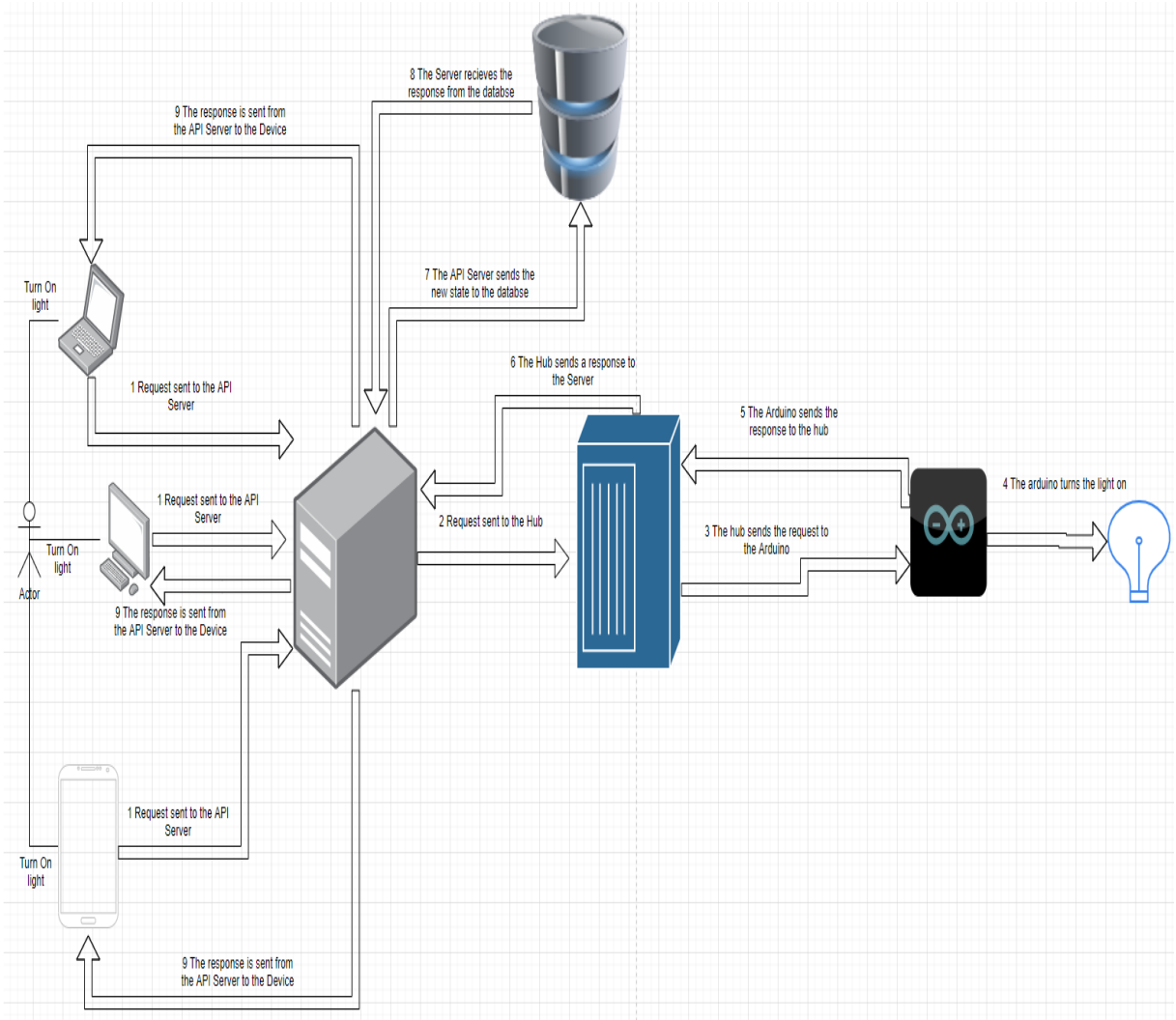

Figure 1 - The HomeDork Design

D2

The use case diagram (Figure 2) depicts a user who access the UI application, which is connected to the server, the server is then connected to the Arduino. The user can communicate with several devices through the server, the server will store sensor data and the states of the devices in a database. The Arduino hub will communicate with the server receiving commands and sending sensory information for the data to be stored. These devices range in functionality such as a security alarm which can be armed and disarmed when the sensor is activated the alarm will sound, and something as simple as light switch.

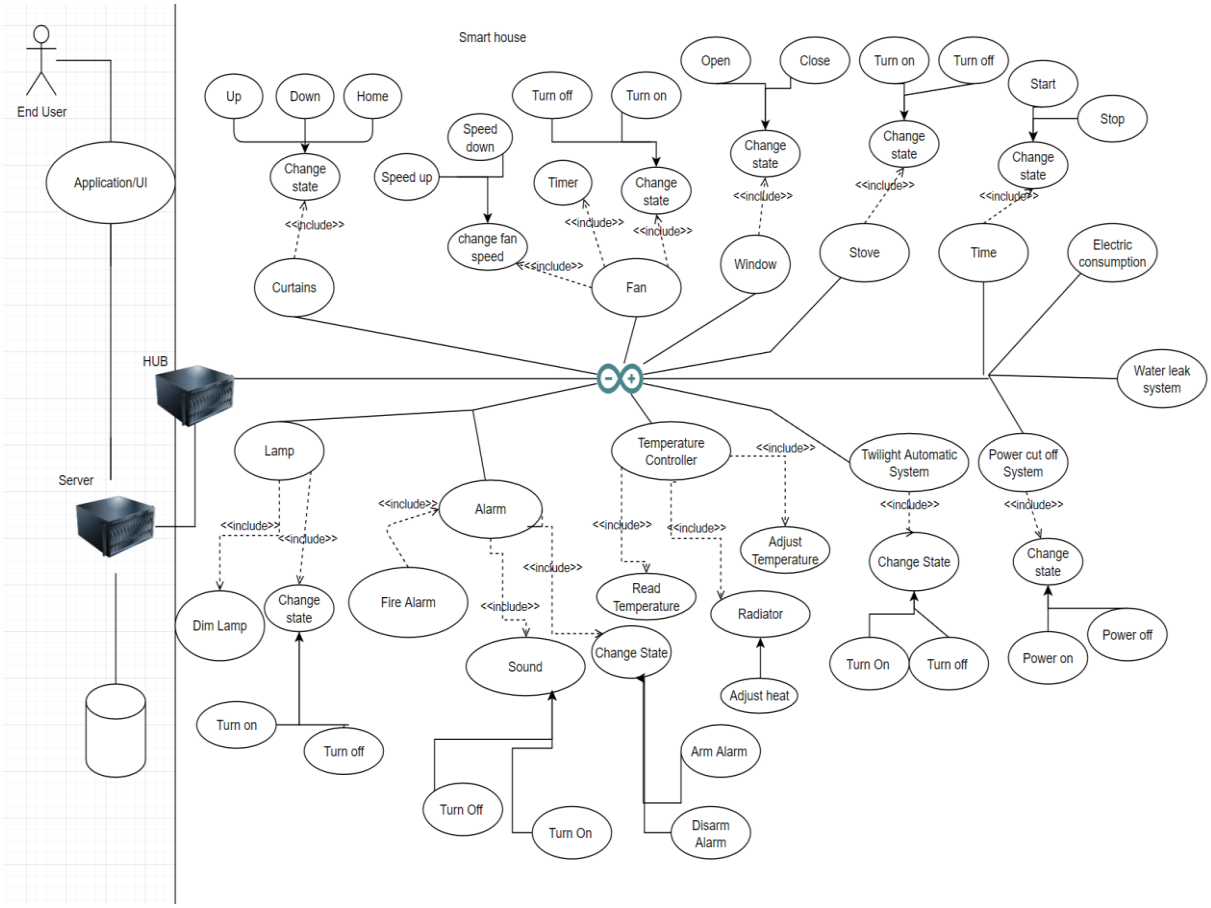

Figure 2 - Use Case Diagram

D3

This is the class diagram for the devices we have the overall structure here (Figure 3). There are 3 packages Models (Figure 5), Utils, and Controllers (Figure 4), there is also a Main that is the ino filetype, these packages are the ones located on the right-hand side in (Figure 3) is the HUB which is the communications package more information on this package can be found at D13 where (Figure 18) has a closer look.

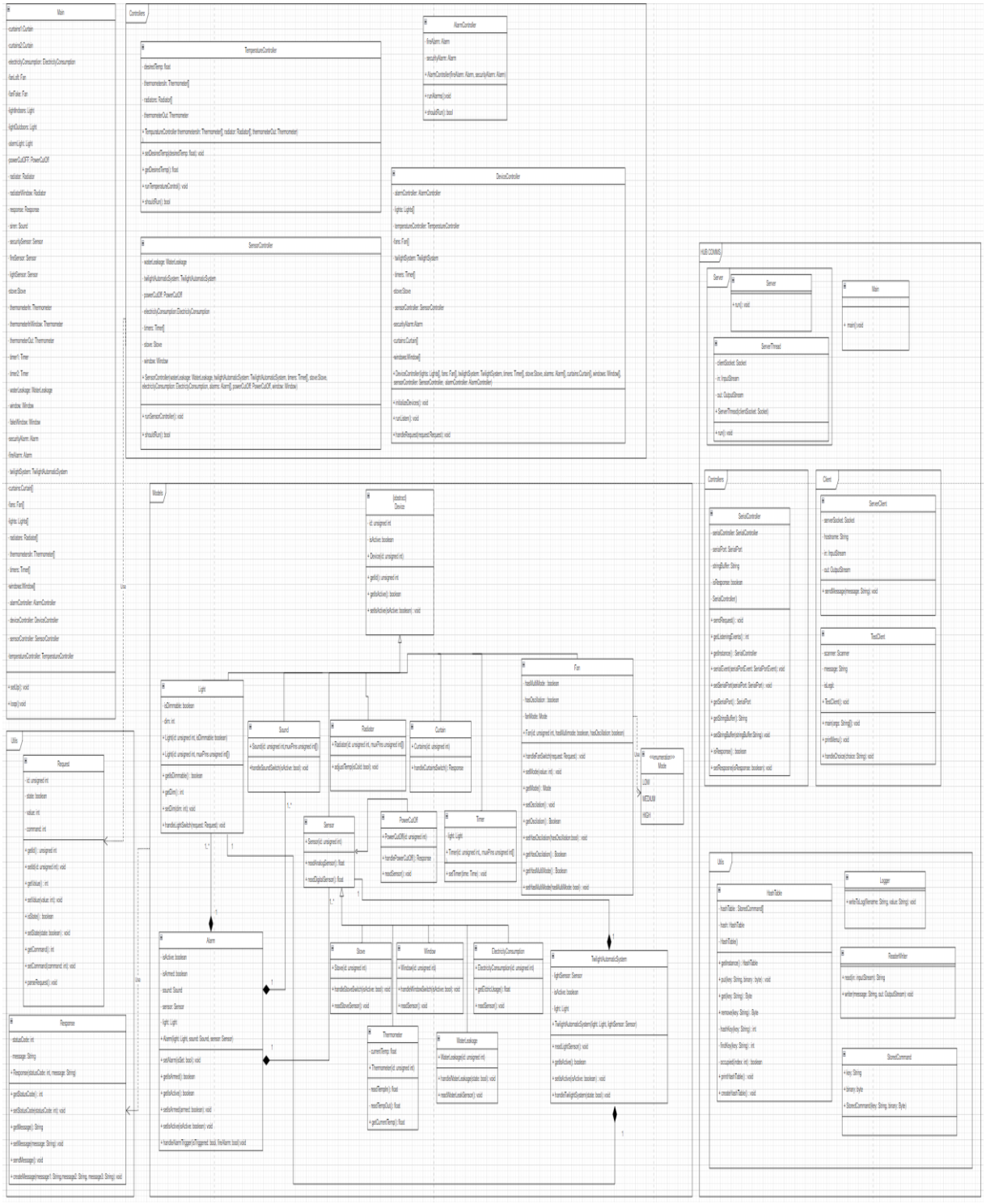

Figure 3 - Class Diagram

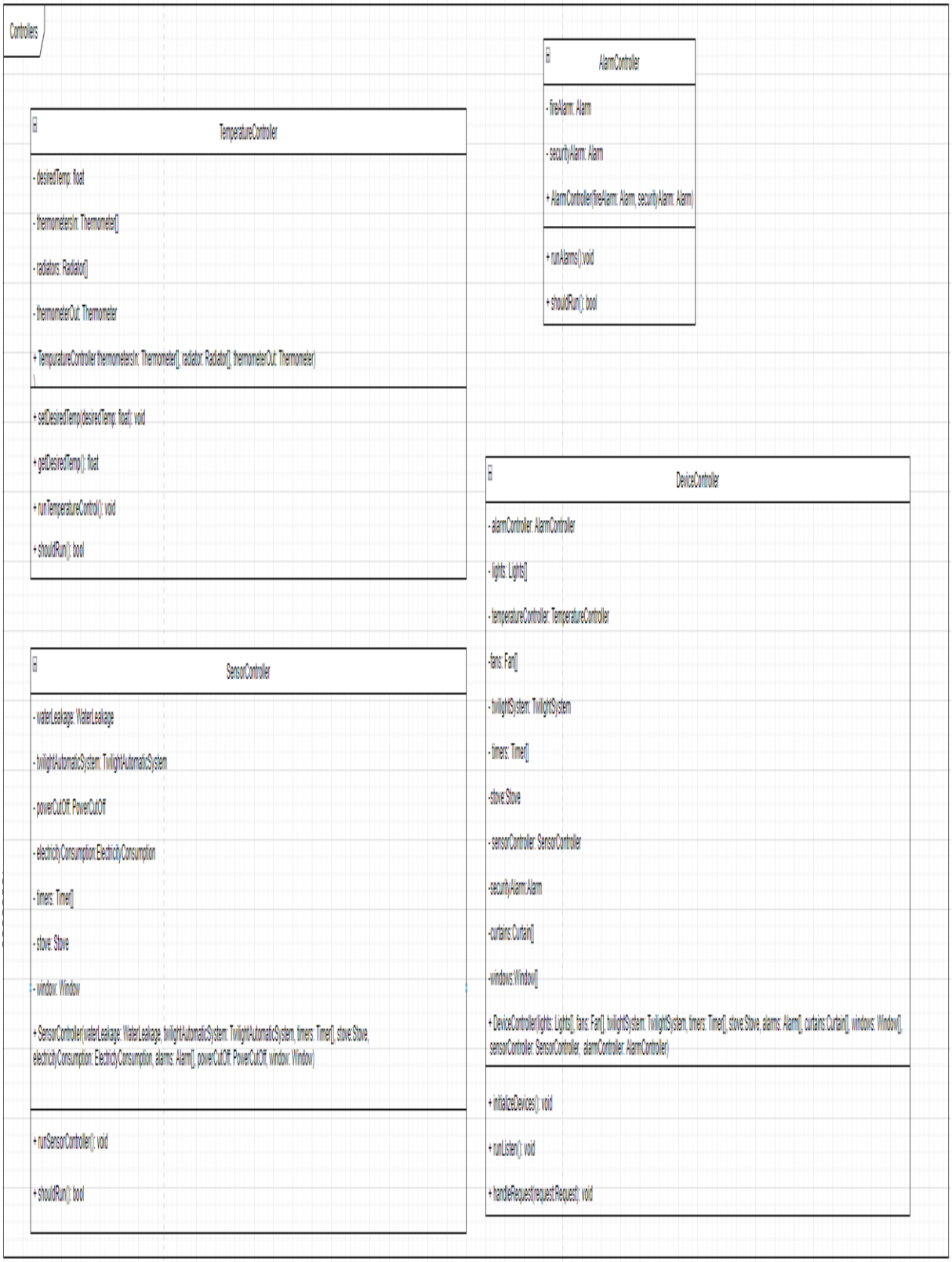

Figure 4 - Controllers

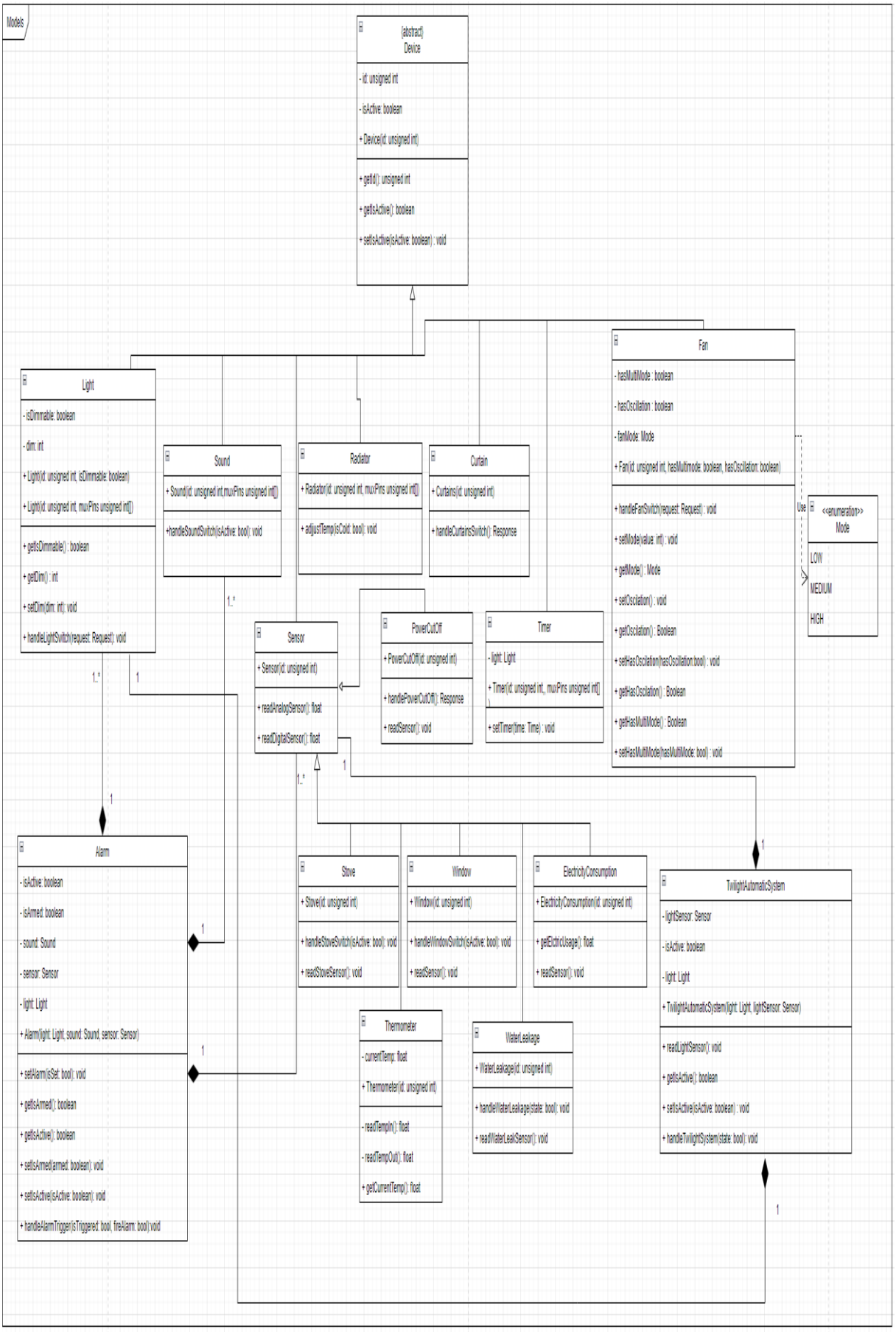

Figure 5 - Models

D4

The first class that was created was an abstract device class (Figure 6) which has a variable for the id of the device as an unsigned int we will be using this as the pin in some cases when just a single pin is necessary, and a Boolean that is used to determine the state of the device. The functions used in this class are getId which returns an unsigned int, getIsActive which returns a Boolean, and setIsActive which takes a Boolean and returns a Boolean.

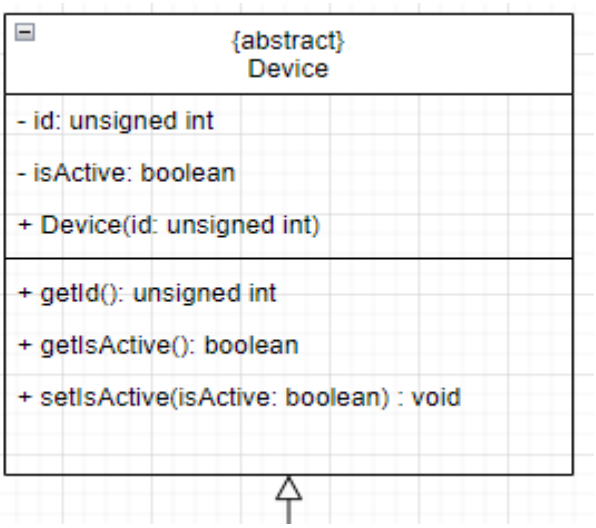

Figure 6 - Abstract Device Class

The classes that inherit from the device (Figure 5, Figure 7, Figure 8, Figure 9) class are light which is for the lamps, the sound class which emits a tone, the sensor class which also has subclasses that makes use of sensors for their primary function, the radiator class which will be used as a heating element, the fan class which will control the fans on the system, the curtain class which is used to represent curtains in the smart home, the timer class which will handle automatic timers on some devices, and the power cut off class which will be able to detect when the power is shut off within the system.

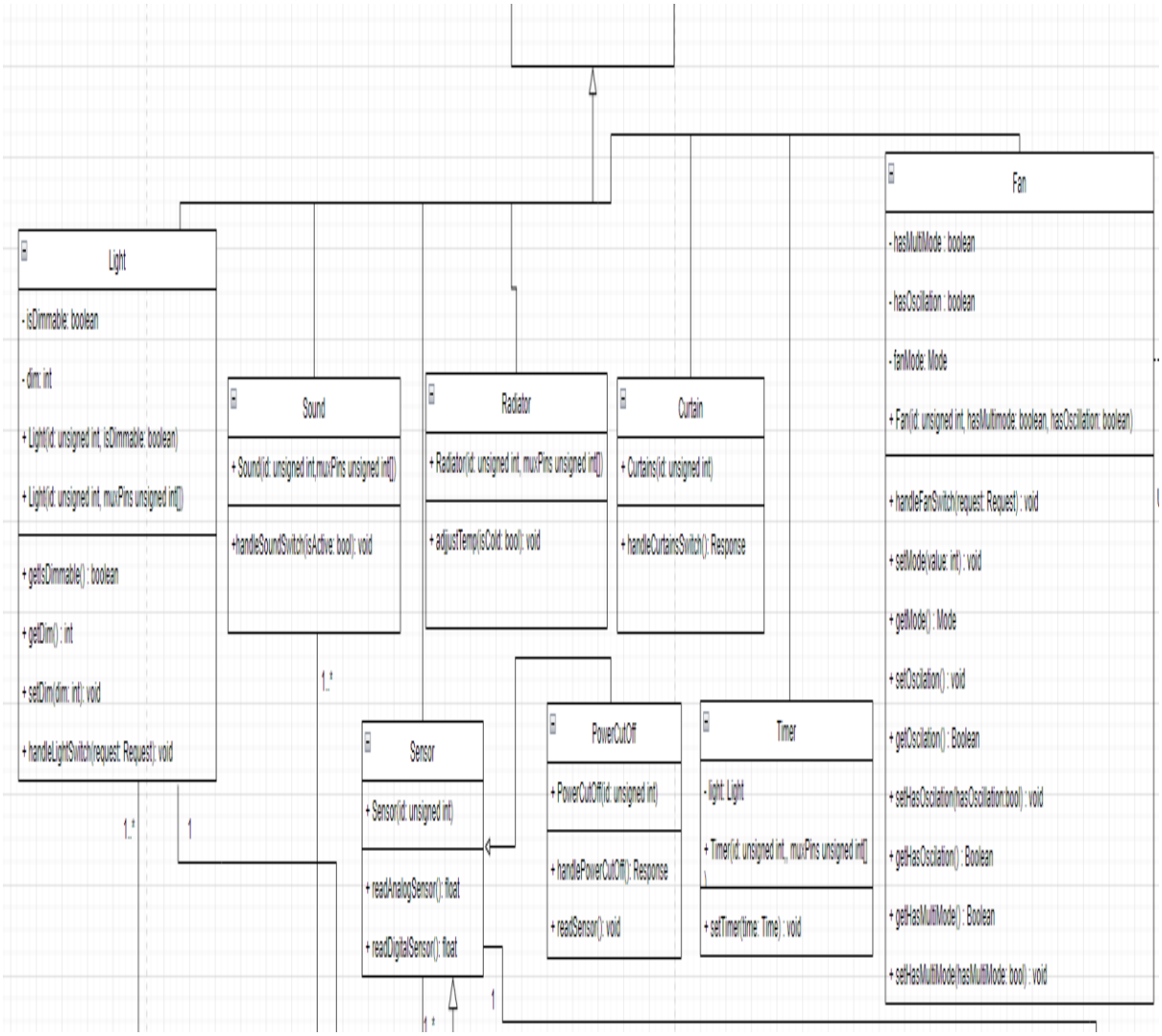

Figure 7 - Classes That Inherit From Device directly are Light, Sound, Radiator, Sensor, Fan, and Curtain.

The light class (Figure 7) has a isDimmable boolean to determine the type of the light it is, dim which will control the voltage, the constructor which takes an id and isDimmable. The functions for the light class are getIsDimmable which returns a boolean, getDim which returns an int, and setDim which takes an int. The sound class has a handleSoundOn and a handleSoundOff functions both of which return a response the constructor takes only the id.

The subclasses of the Sensor class(Figure 8) the window class has a handleWindowSwitch function the constructor contains only the id. The electricity consumption class monitors electric consumption, has a readElectricUsage function the constructor contains only the id. The water leakage class monitors water leakage, the constructor only contains an id, the function handles the read from the device to determine if a leak is present. The stove class takes only the id in the constructor and handles the switch if the stove state changes. The thermometer class handles the readings of the indoor and outdoor thermometers. The sensor classes all contain a function to read the sensor which will automatically handle any functions or messages needed to update the user.

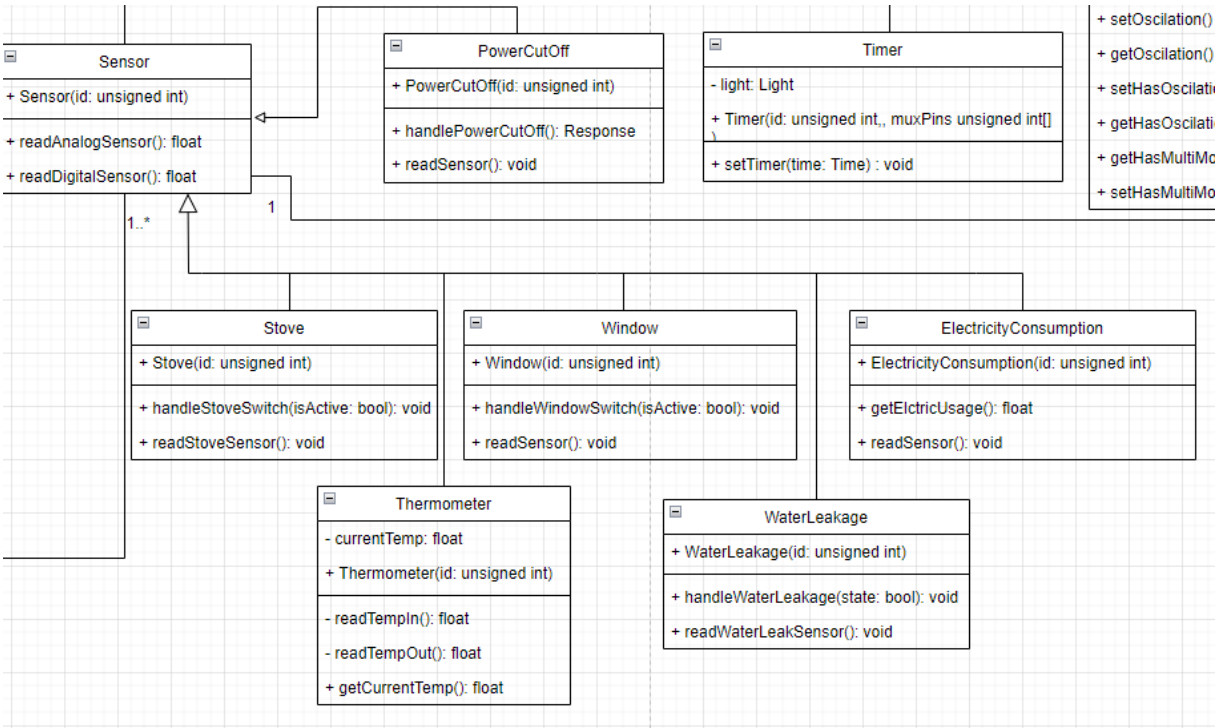

Figure 8 - Classes That Inherit From Sensor class Window, Electricity Consumption, PowerCutOFF, Stove, Thermometer and Water Leakage

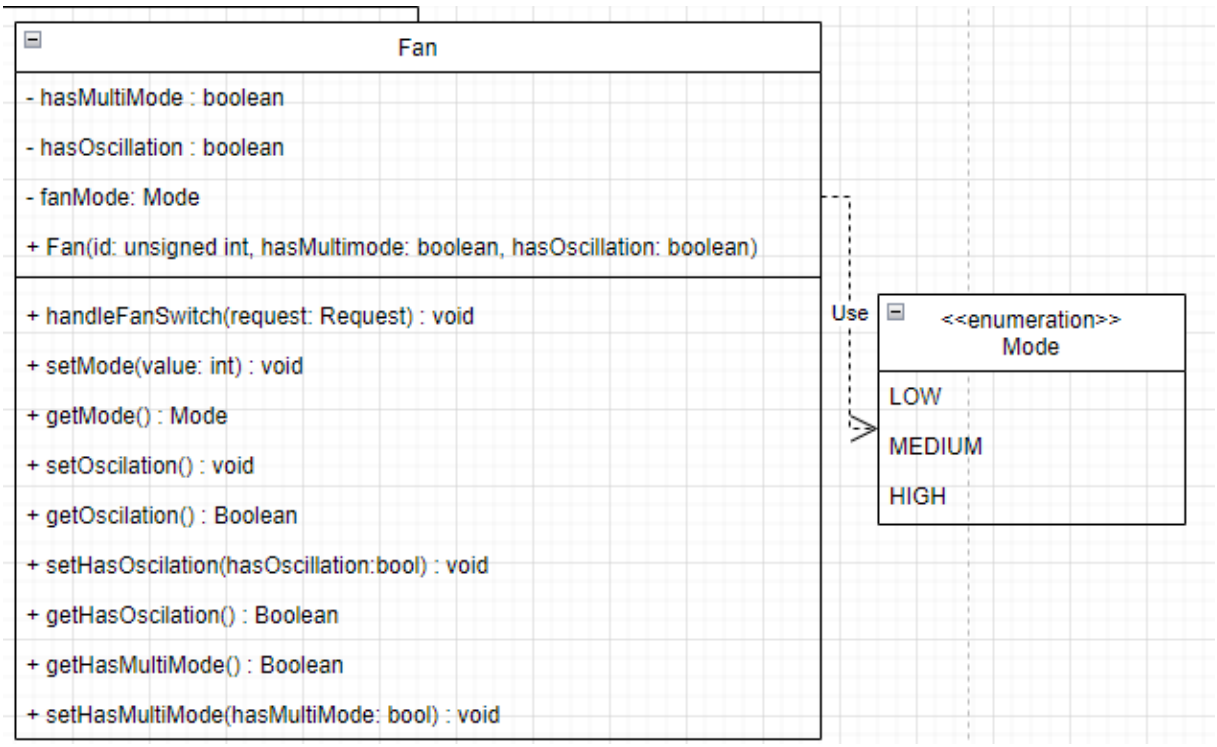

Figure 9 – Classes That Inherit From Device Fan, and Stove

The fan class (Figure 9) has 3 variables hasMultiMode a Boolean, hasOscillation a Boolean, mode which is an enum of LOW, MEDIUM and HIGH. The constructor takes id, hasMultimode and hasOscillation. The functions associated with the fan class are setMode which takes integer as a parameter, getMode which returns mode, setOscillation getOscillation which returns a Boolean, getHasOscillation which returns a Boolean, and

getHasMultimode which also returns a Boolean. The stove class has a variable sensor which is passed through the constructor and id, the function is like other sensors has a read sensor function that handles the state change.

D5

The alarm class is (Figure 10) a composition of the light device, the sound device, and the sensor device so the instances of objects are created and used by this class. The alarm class also has isActive Boolean and takes the light, sound and sensor as parameters in the constructor. The functions associated with the class are setAlarm which takes a boolean, a getIsActive which returns a Boolean, and setIsActive with a isActive Boolean as a parameter and returns a Boolean the handleLightSwitch is a function that takes a request class object and handles the request.

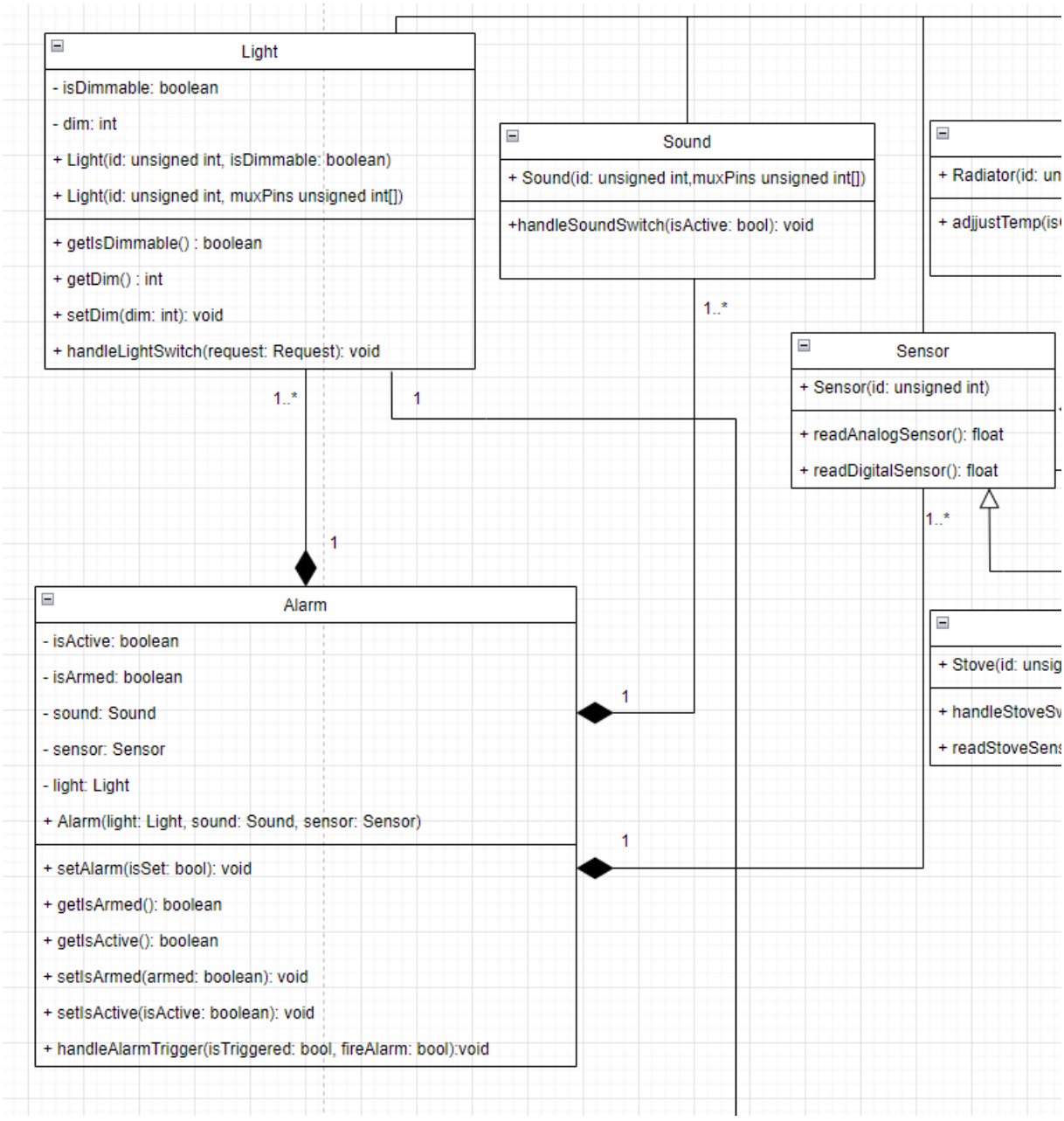

Figure 10 - Alarm Class Composition - Light, Sound, and Sensor

D6

The twilight automatic system class (Figure 11) has a sensor which will be taken as in the constructor and has a readSensor and handleTwilightSwitch functions.

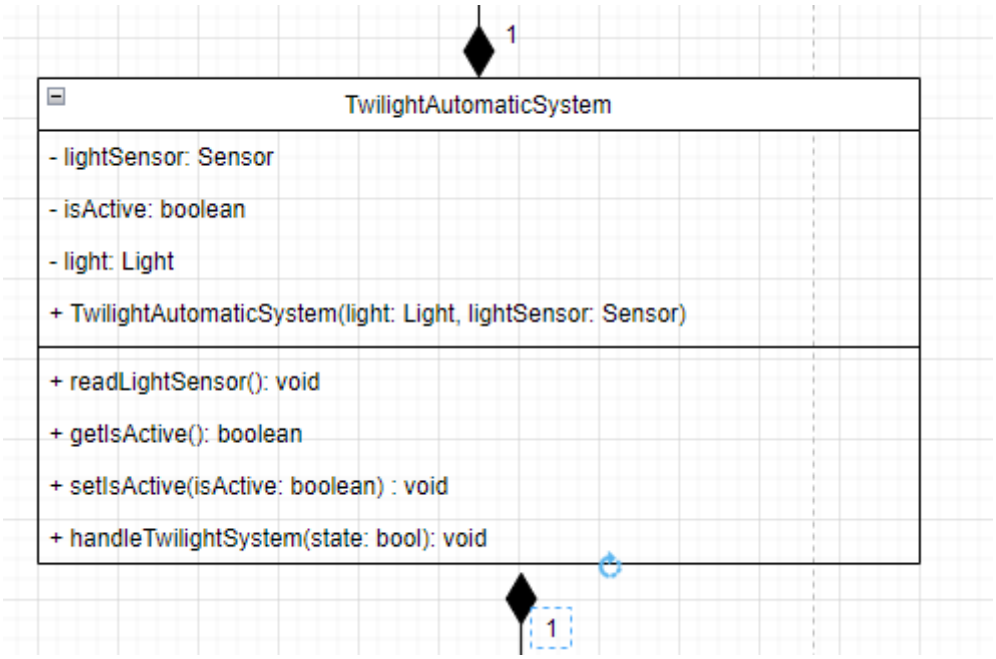

Figure 11 - Twilight Automatic System

D7

Temperature controller class (Figure 12) at this moment is a device which as of right now we are using as a place holder for whatever thermostat that would be used to adjust the temperature. The class has a desiredTemp as a variable which is of type float and the thermometerIn of the thermometer device. The functions used in the temp. controller class is the setDesiredTemp which takes desiredTemp float as a parameter, the getDesiredTemp function which returns a float and runTemperatureControl.

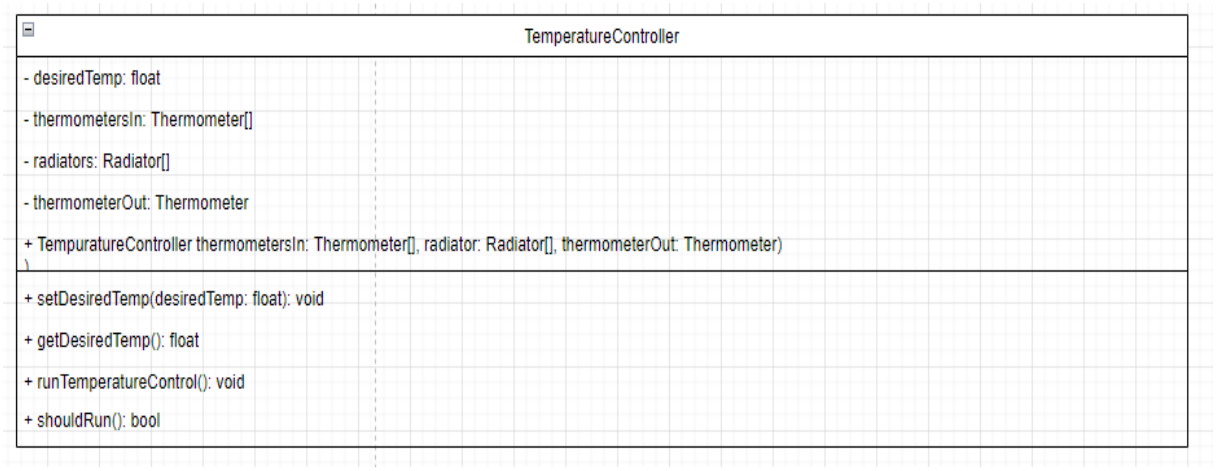

Figure 12 - Temperature Controlle

D8

The device controller class (Figure 13) will be where the command from the user comes in as such it will take all the devices to handle any specific request which will be parameters in the constructor, The functions in this class are the runListen, initializeDevices, and handle request. The Device controller class handles all other controllers as well as checks for and runs methods based on the commands received or the local programs running.

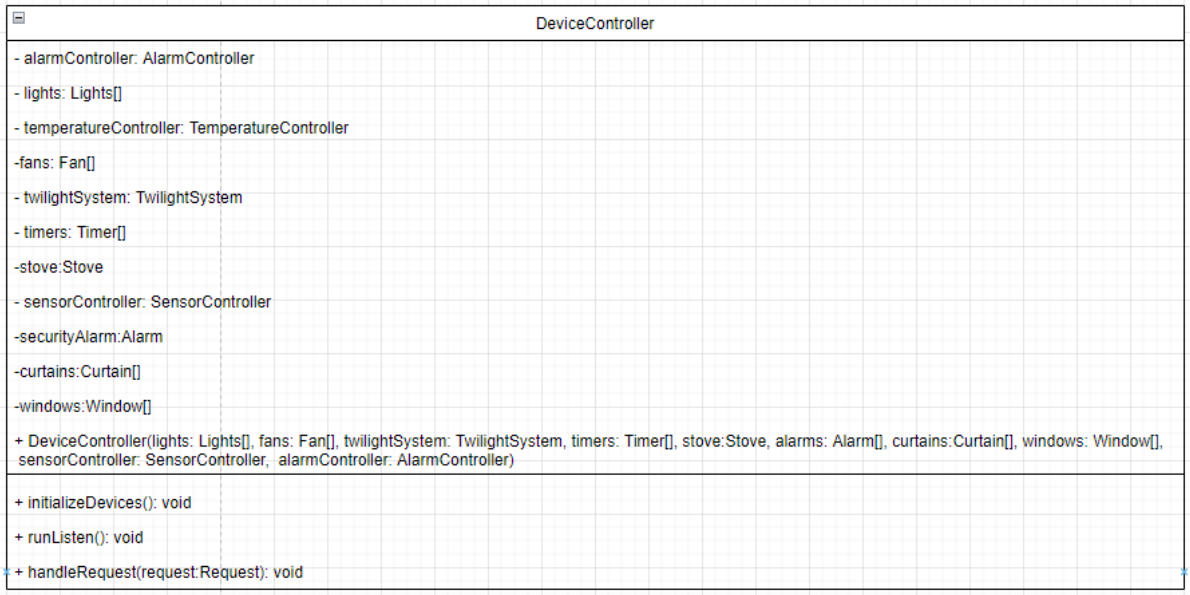

Figure 13 - Device Controller

D9

The sensor controller class (Figure 14) will handle all the classes that have to deal with input they will be passed through the constructor the only function in the class is runSensorController and the shouldRun method which is a method that all controllers contain in order to check the serial port for data coming in and exit the current operations and return to the device controller.

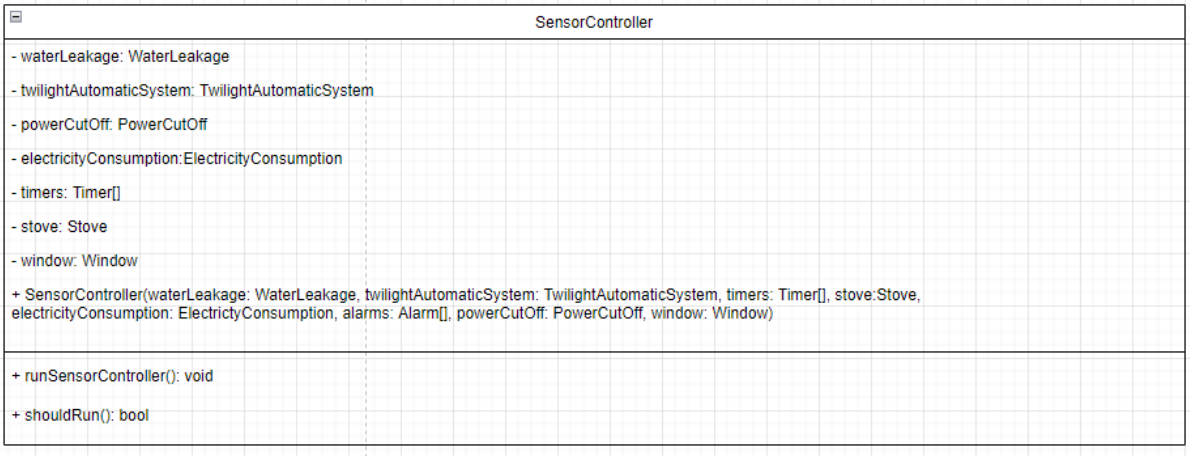

Figure 14 - Sensor Controller

D10

Alarm Controller will be used to control the alarms and monitor the sensors (Figure 15). The security alarm and fire alarm will be passed as parameters to the constructor.

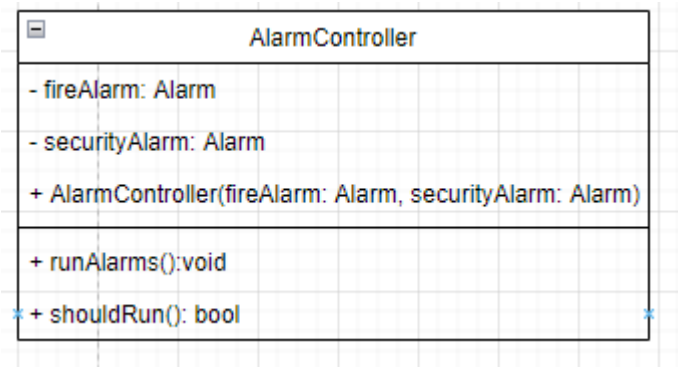

Figure 15 - Alarm Controller

D11

The response class (Figure 16) is used to relay a code and a message to the server to confirm the changes being made on the Arduino side of things or if a failure occurs it can be relayed and give a possible reason for it. The response class takes statusCode of int type as a variable and a message of type String. The constructor takes both variables and has getters and setters for both. The request class handles the command from the server and converts the message to a request object.

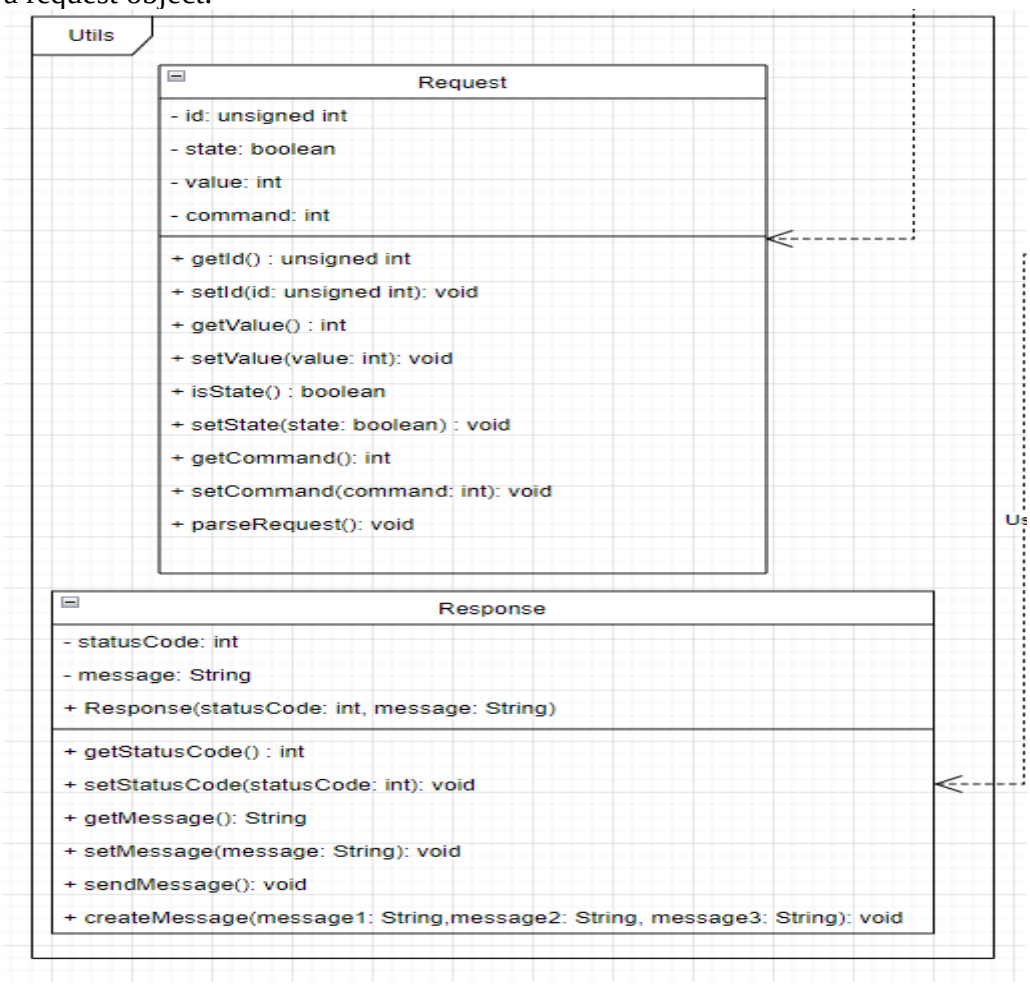

Figure 16 - Response & Request Classes

D12

The main class (Figure 17) sets up the global variables and runs the deviceController classes run method within the loop method.

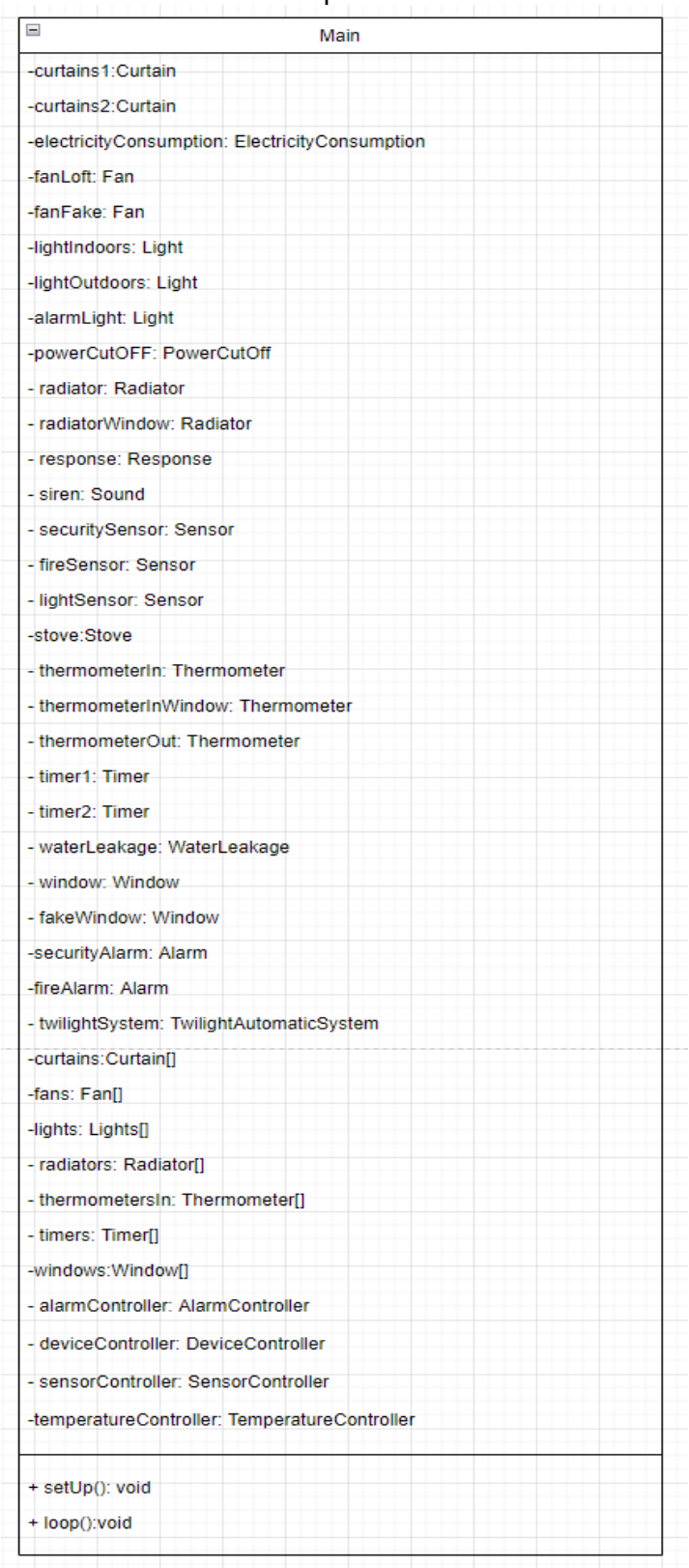

Figure 17 – Main Class

D13

The HUB found in (Figure 18) is composed of 4 packages. The first located in the top right is the server and the server thread which handles the server running on the machine connected via serial port to the Arduino device. The controller package contains one class a Serial Controller which is a singleton holds the serial port connection with the Arduino through the use of the package JComms, this class handles the reading and writing to the Arduino. The client class contains a server client which is the client to the main server and the Test client used for testing server communications with a client sending commands. The Util class contains a HashTable a singleton class that holds the key value pair of the command and the byte associated with that command on the Arduino device. The store command is a class of the key value pair in the hash table the other 2 are logging and I/O operations.

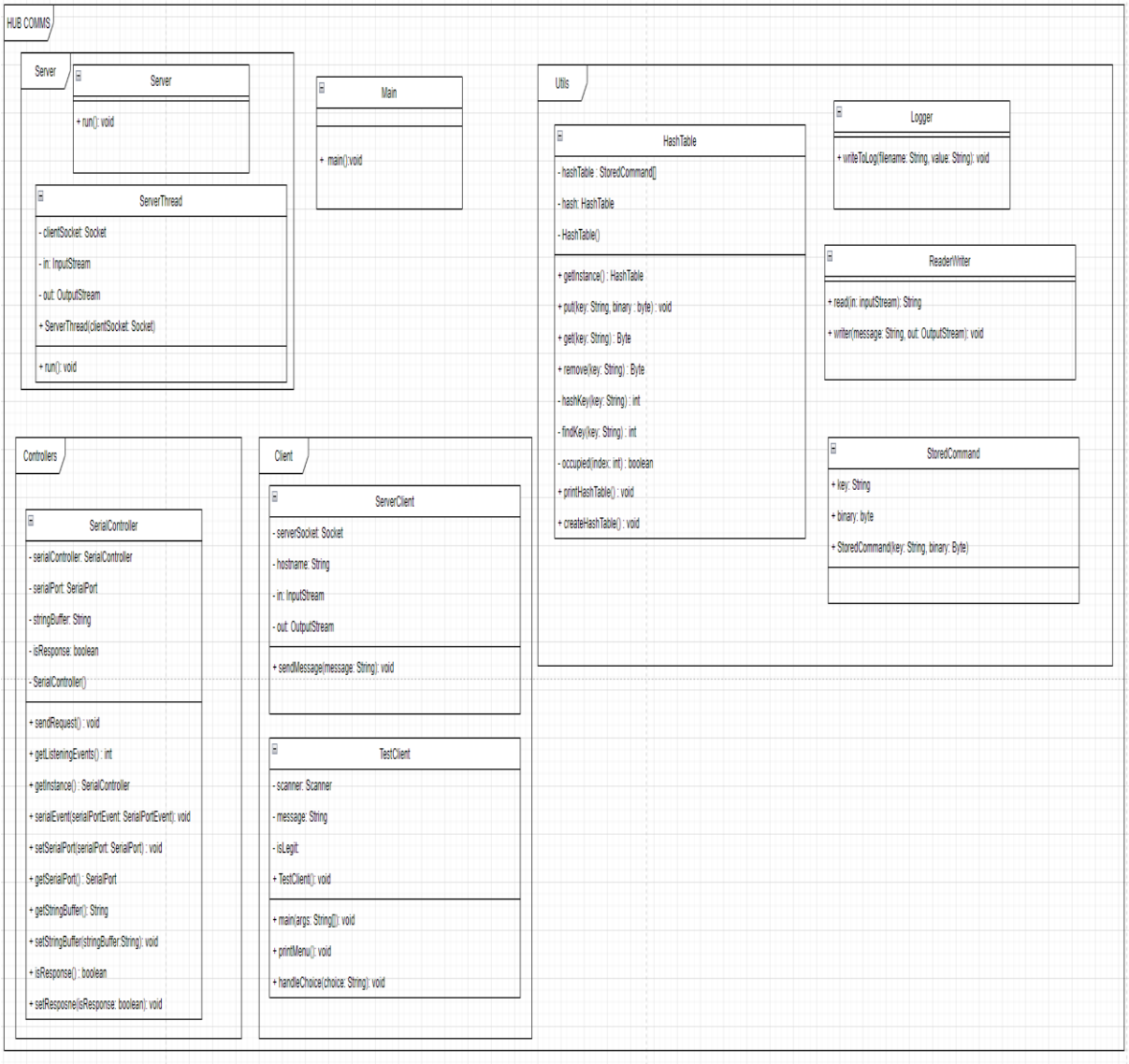

Figure 18 - HUB for Communication with Server and Arduino

D14

The sequence diagram (Figure 19) depicts the reading of the temperature inside the home, the room temperature, a request is sent to the server, the server call the Arduino (Hub), the hub dispatches a request to the temperature controller the, the controller dispatches a request to the inside thermometer and returns a message to the temperature controller. The temperature controller creates and sends a response to the hub in turn sends a response to the server, the server then handles the response.

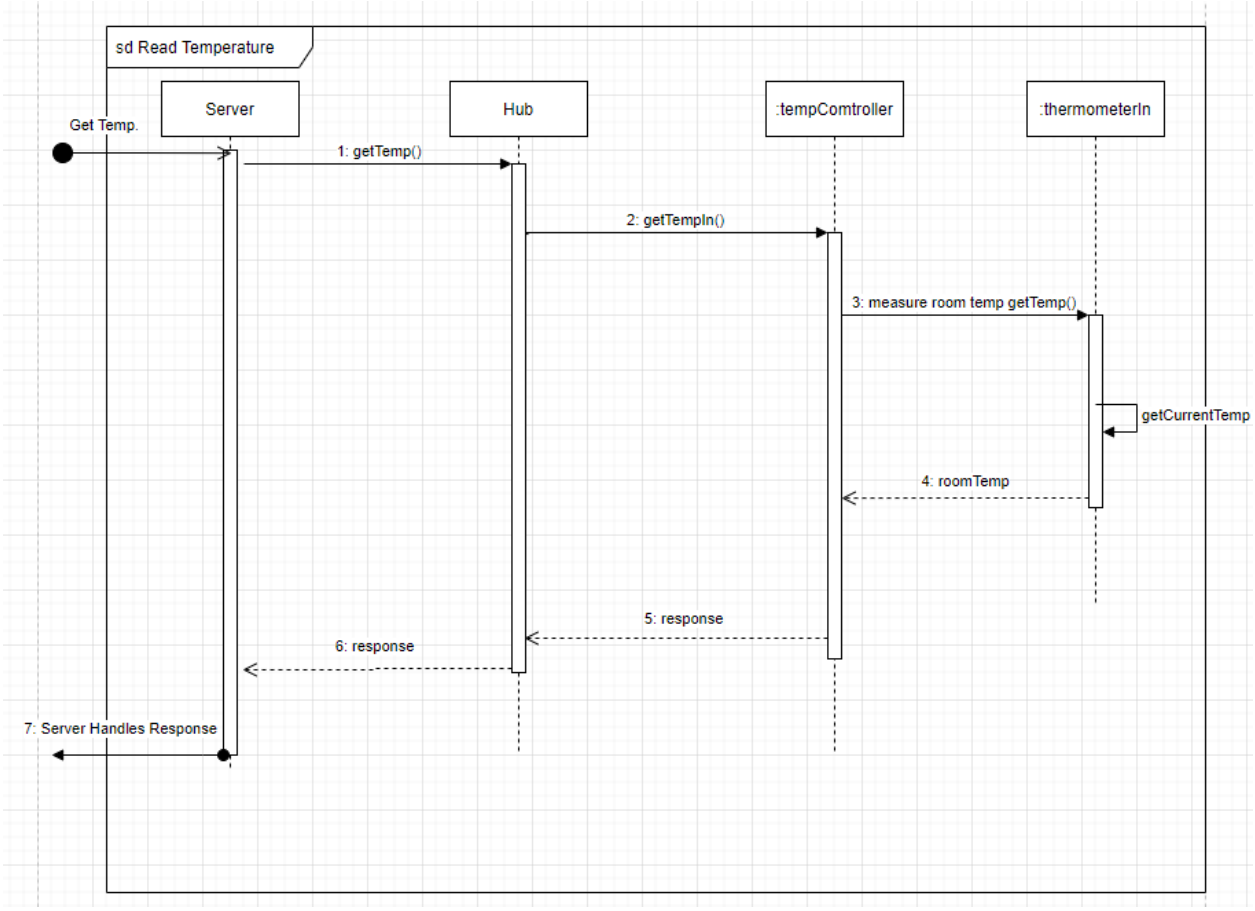

Figure 19- Sequence Diagram Read Temperature

D15

The sequence diagram (Figure 20) depicts the handling a command from the server to the Arduino, the hub is a server that handles communication between the server and Arduino, the server dispatches a request with a message to the hub to change the state of the device. The command is received in the device controller creates a request object out of the message and handles the request to the device the device returns a response that is carried back to the server.

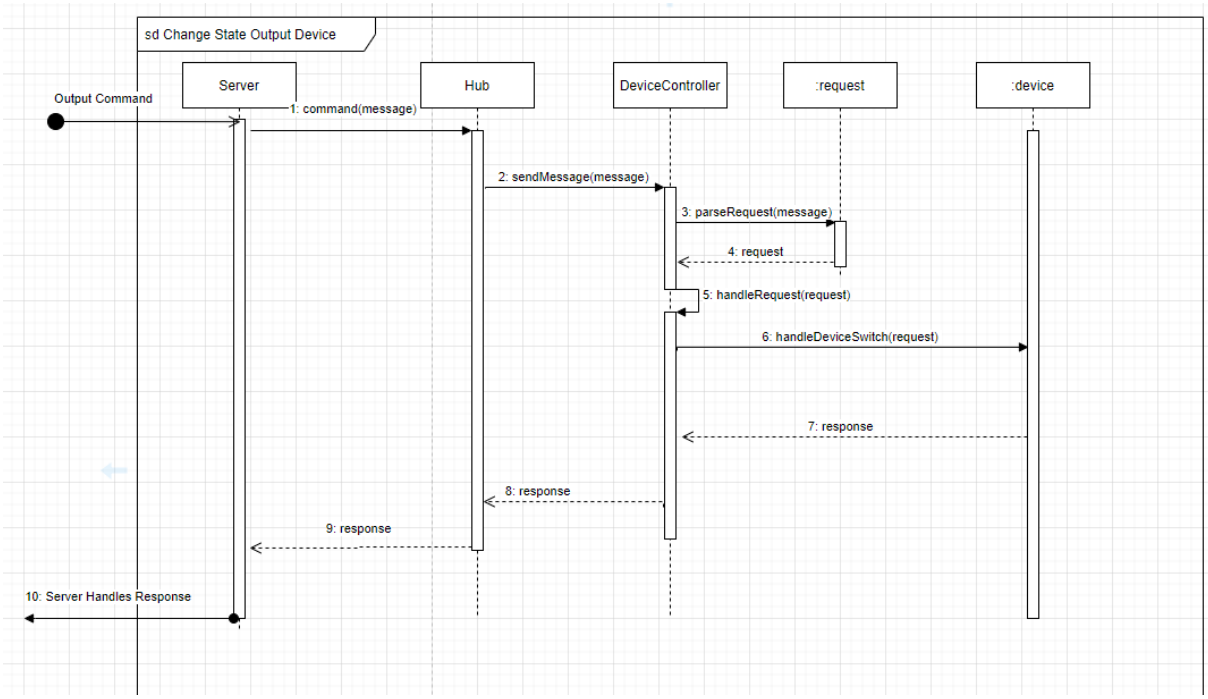

Figure 20 - Sequence Diagram Output Device (example Light)

D16

The sequence diagram (Figure 21) how the device controller operates, the device controller is listening for a message from the hub once received it will construct a request from the message then it will handle the request checking to see if both the device name exists and the id associated with the command exists under that device type.

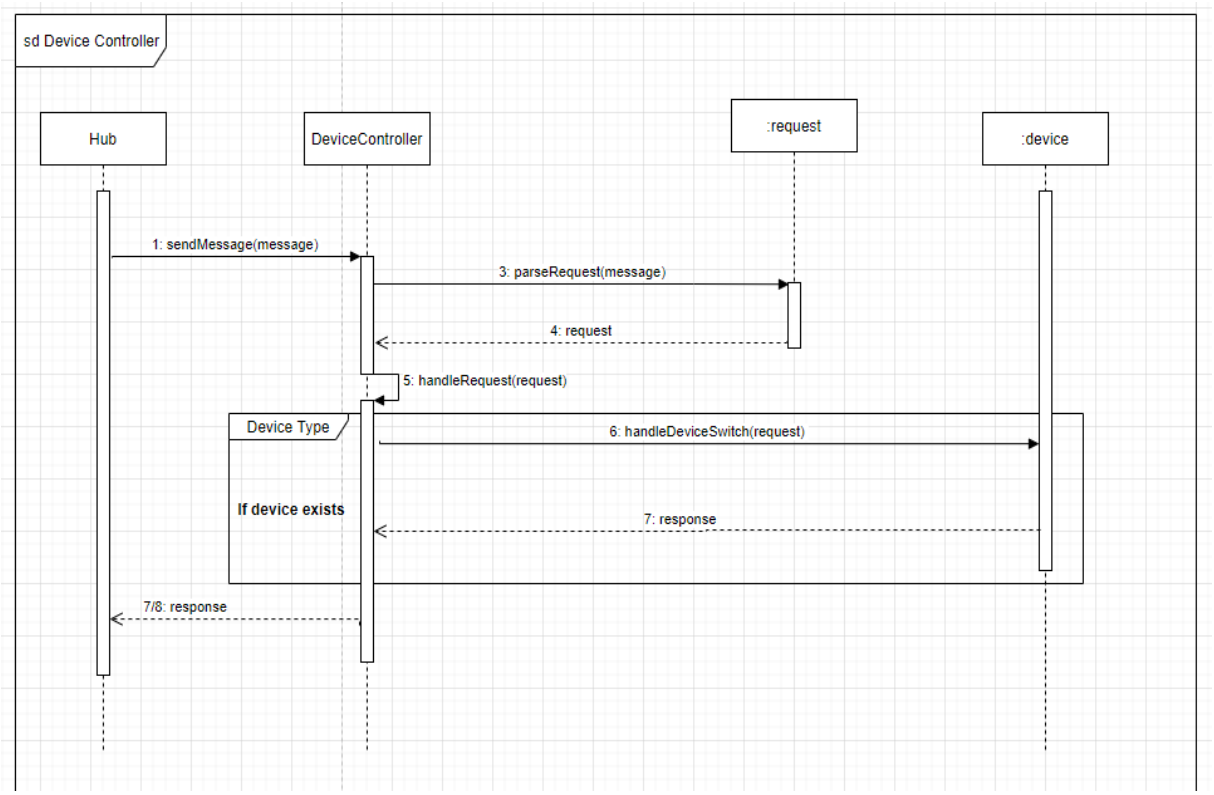

Figure 21 - Sequence Diagram Open Curtain

D17

The sequence diagram (Figure 22, Figure 23) depicts temperature control on the Arduino (Hub), as of right now the hub will initiate a check on the temperature through a dispatch on a timer for every hour or thirty minutes (theoretical) the tempController will dispatch a request for the current temperature to the inside thermometer which will return a message of the room temp. The temperature controller will compare the room temp to the desired room temp which will return the difference if the difference is greater than 0 the temperature is to high. The temperature controller will dispatch a request to the ac to lower the output. The ac will send a response back to the temperature controller. If the difference is less than 0 the temperature is to low. The temperature controller will dispatch a request to the ac to raise the output. The ac will send a response back to the temperature controller. If there is no difference the temperature controller will dispatch a response to the hub.

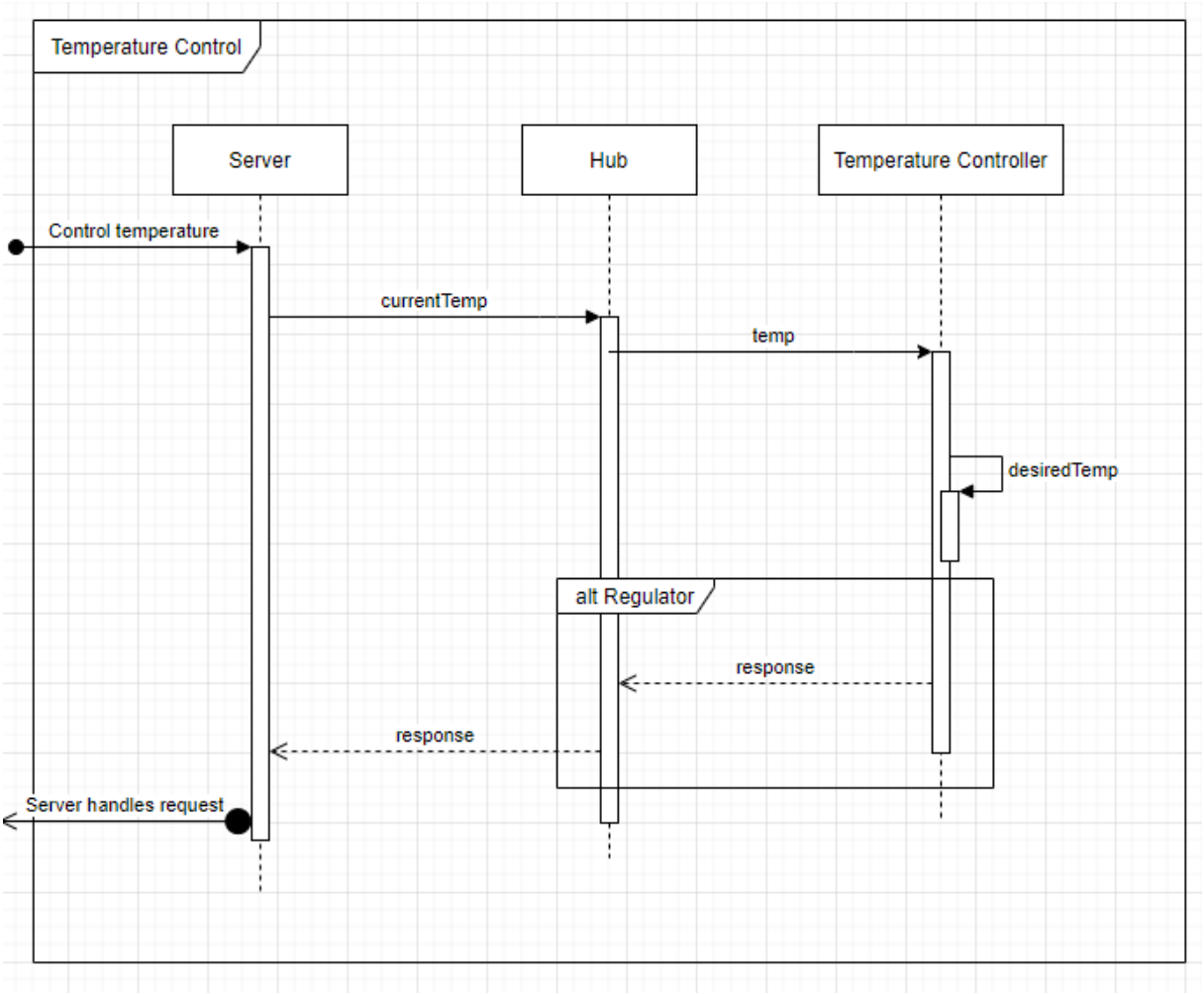

Figure 22 - Sequence Diagram Temperature Control

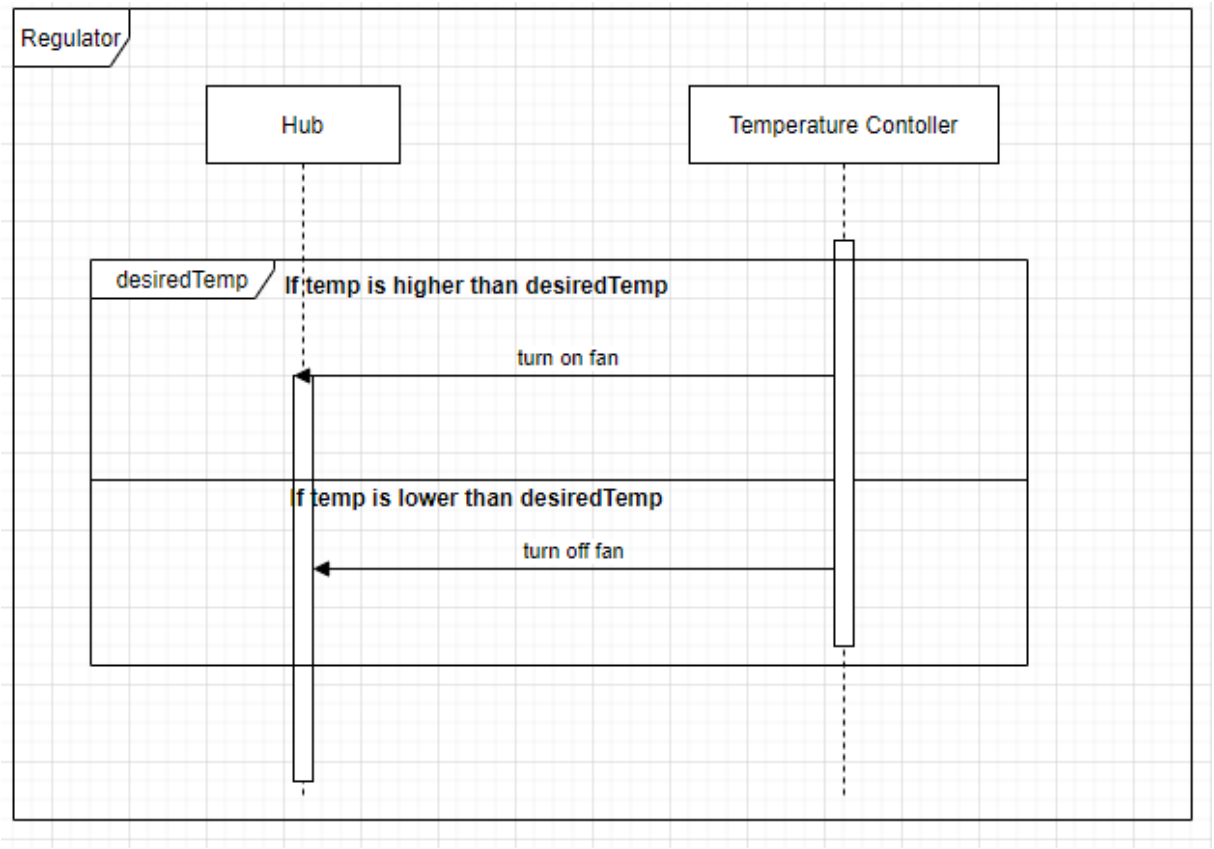

Figure 23 - Sequence Diagram Regulator of Temperature Controller

D18

The sequence diagram (Figure 24) depicts sensor control on the Arduino (Hub), the hub will initiate a check by using the classes read sensor class. If the sensor reads that a trigger has occurred it will call a method to handle the event being triggered.

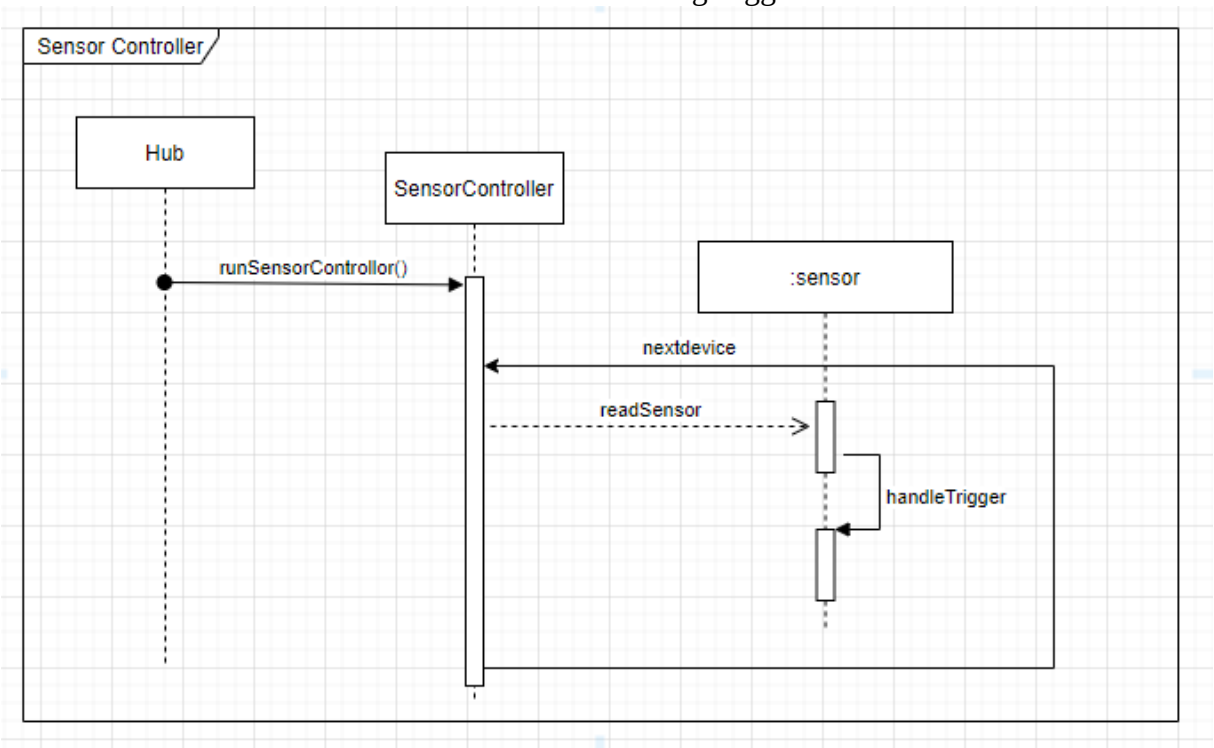

Figure 24 - Sequence Diagram Sensor Controller

D19

The sequence diagram (Figure 25) depicts the alarm controller running locally on the Arduino handling the events with the alarm triggers.

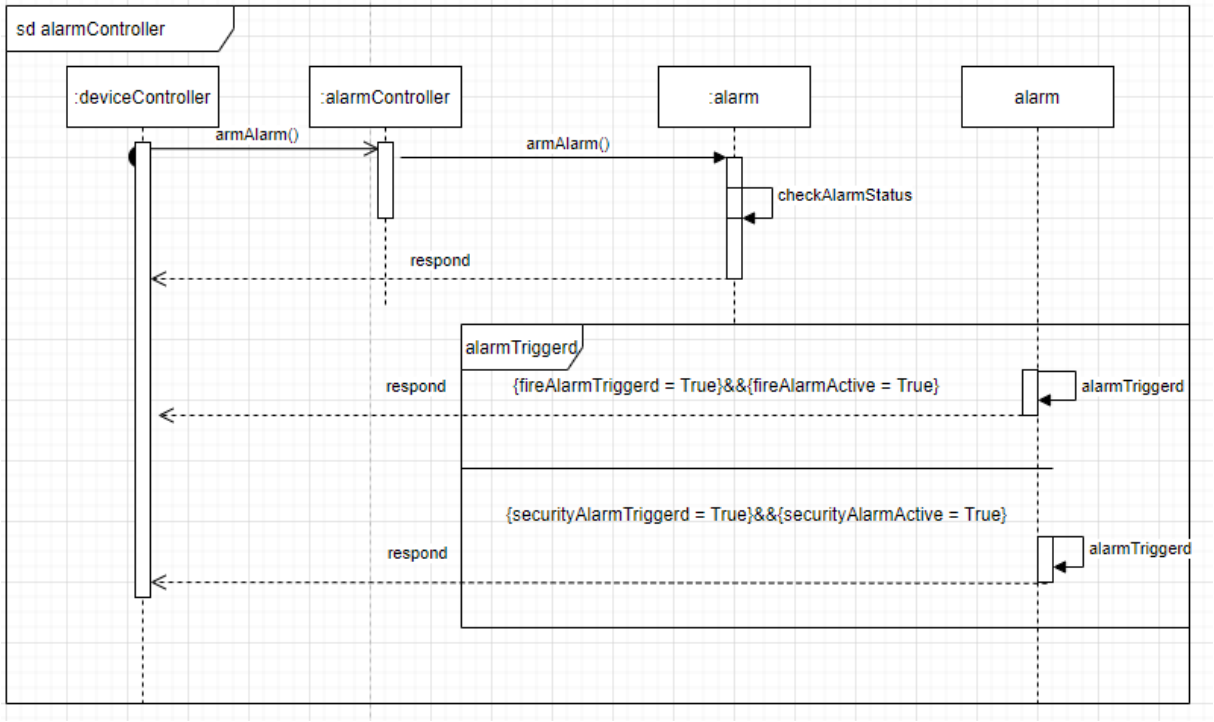

Figure 25 - Sequence Diagram Alarm Controller

D20

The State diagram for the temperature controls system (Figure 26).

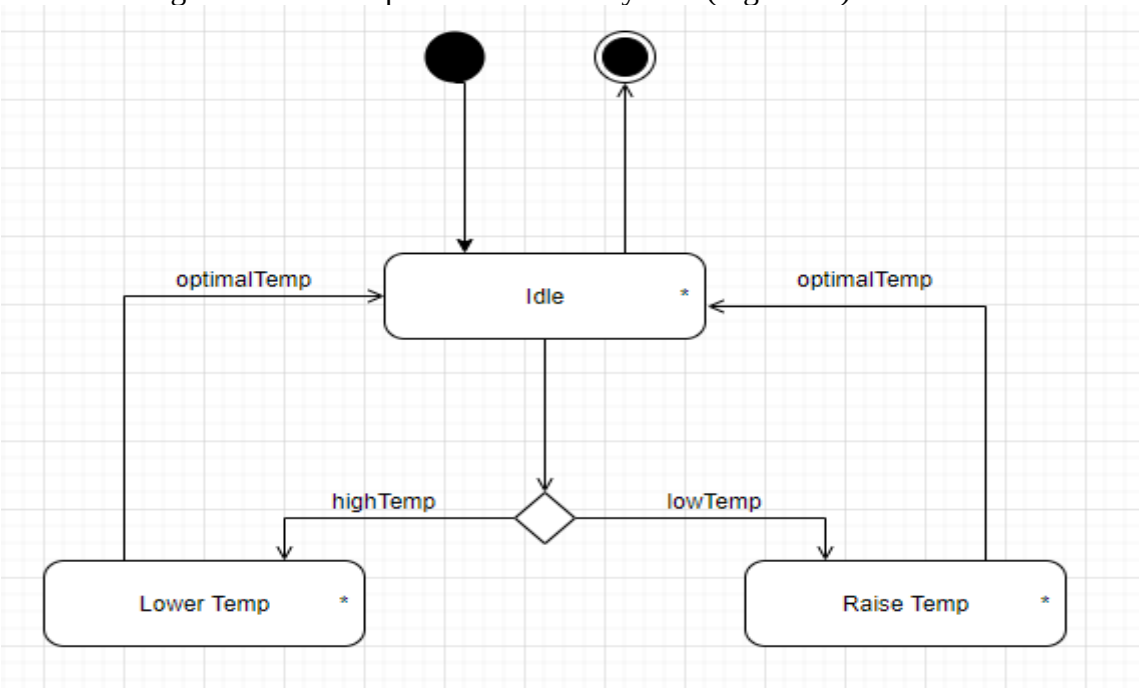

Figure 26 - State Diagram Temperatur Control

D21

State diagram represented for the device controller (Figure 21, Figure 27). The controller listens for a command from the server once that is received the message is parsed to a request and then the request is then handled the initial state then handles the command and then the output device is rendered on or off in some cases set to a specific level.

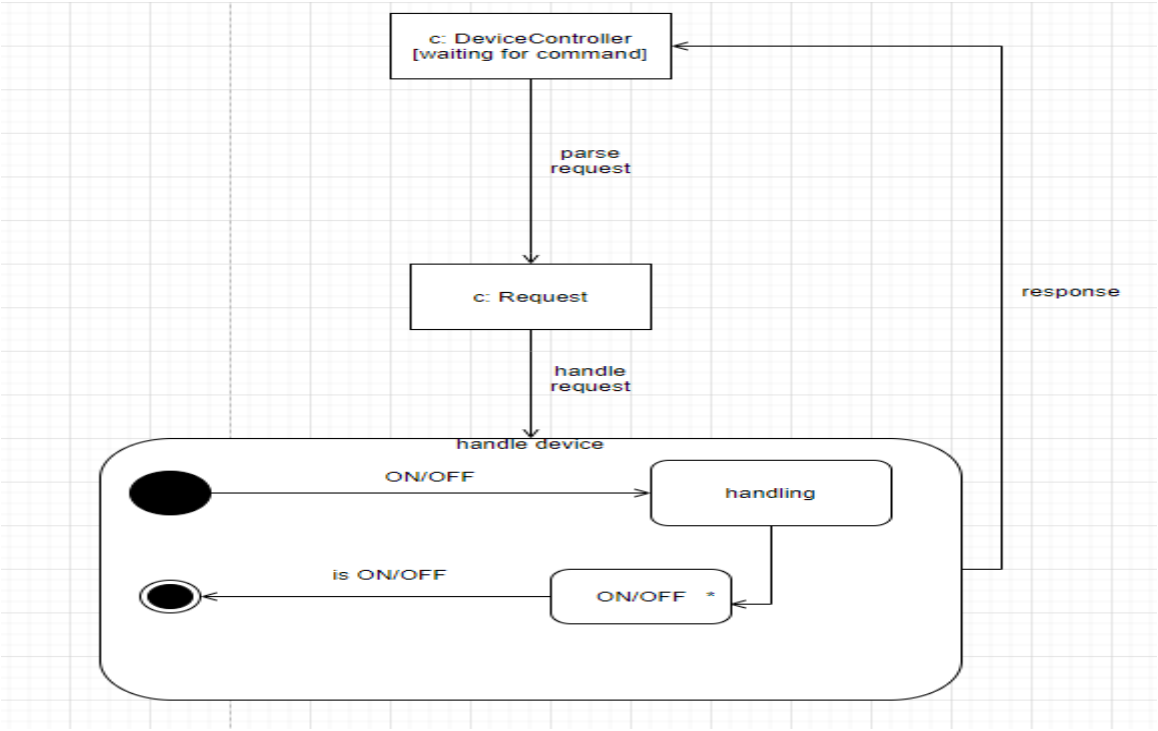

Figure 27 - State Diagram Device Controller and An Output Device