Design Documentation-Devices

HomeDork - Interactive 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
04/24/2021	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

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	Essential
D12. Class Diagram – The Main Class	Essential
D13. Sequence Diagram – Read Temperature	Essential
D14. Sequence Diagram – Turn Lamp On	Essential
D15. Sequence Diagram – Open Curtain	Essential
D16. Sequence Diagram – Temperature Control	Essential
D17. State Diagram – Temperature Control	Essential
D18. State Diagram – Curtain	Essential
D19. State Diagram – Light	Essential
D20. State Diagram – Window	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.

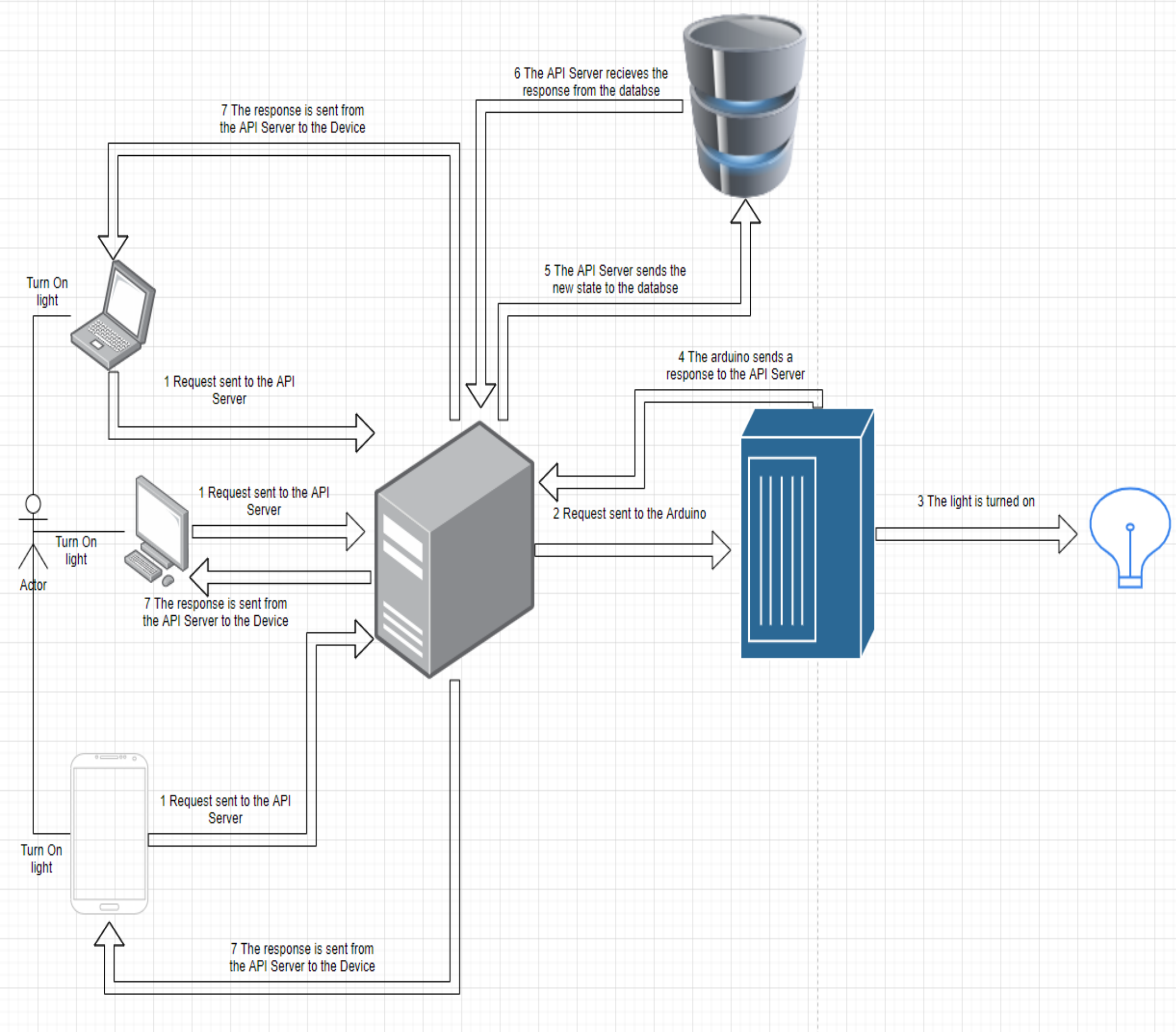

6 The API Server recieves the response from the databse
7 The response is sent from the API Server to the Device
5 The API Server sends the new state to the databse
Turn On light
4 The arduino sends a response to the API Server
1 Request sent to the API Server
1 Request sent to the API Server
3 The light is turned on
2 Request sent to the Arduino
Turn On light
Actor
7 The response is sent from the API Server to the Device
1 Request sent to the API Server
Turn On light
7 The response is sent from the API Server to the Device

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 a number of devices such as an security alarm which can be set and when the sensor is activated the alarm will sound.

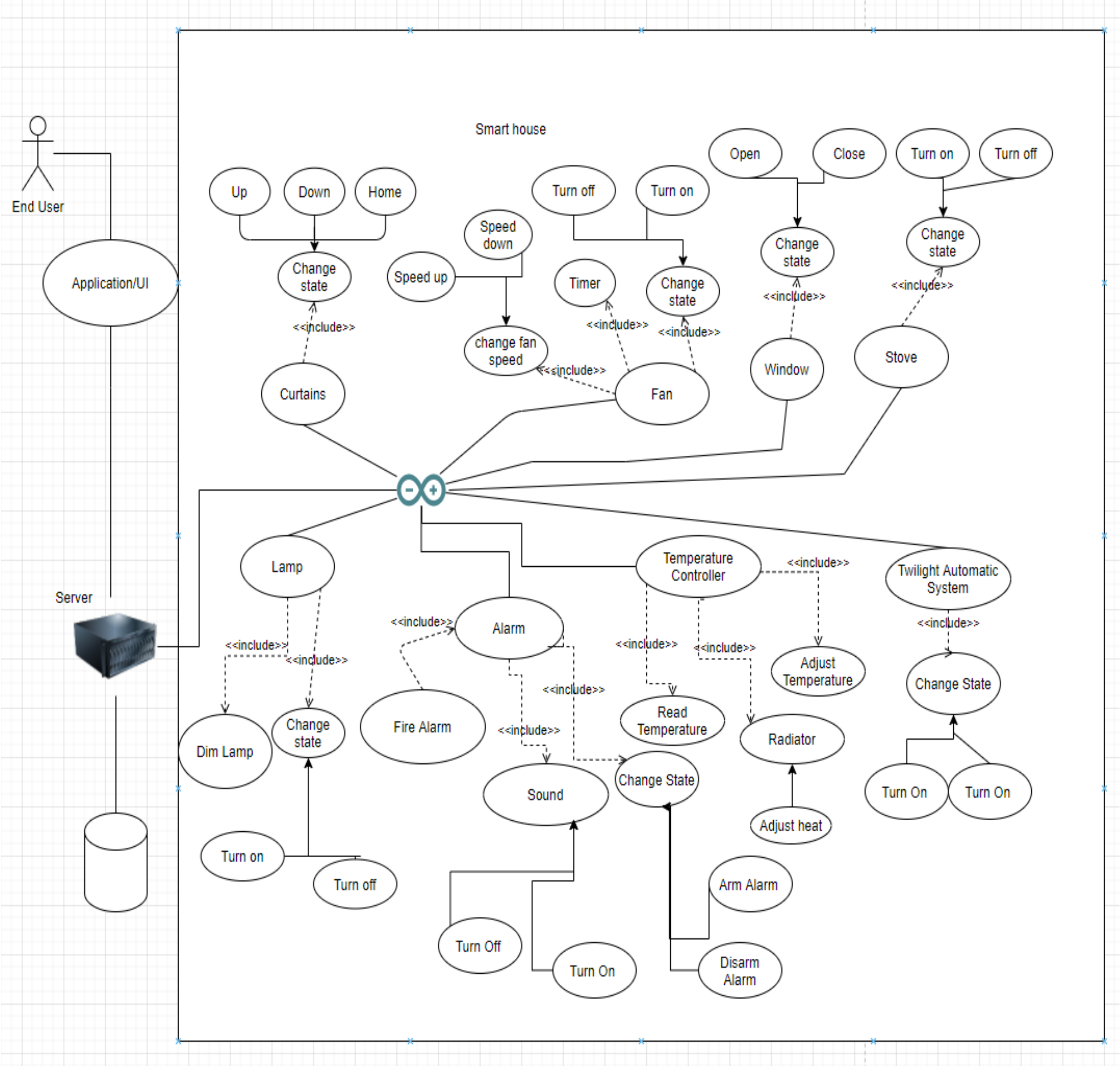


Figure 2 - Use Case Diagram

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.

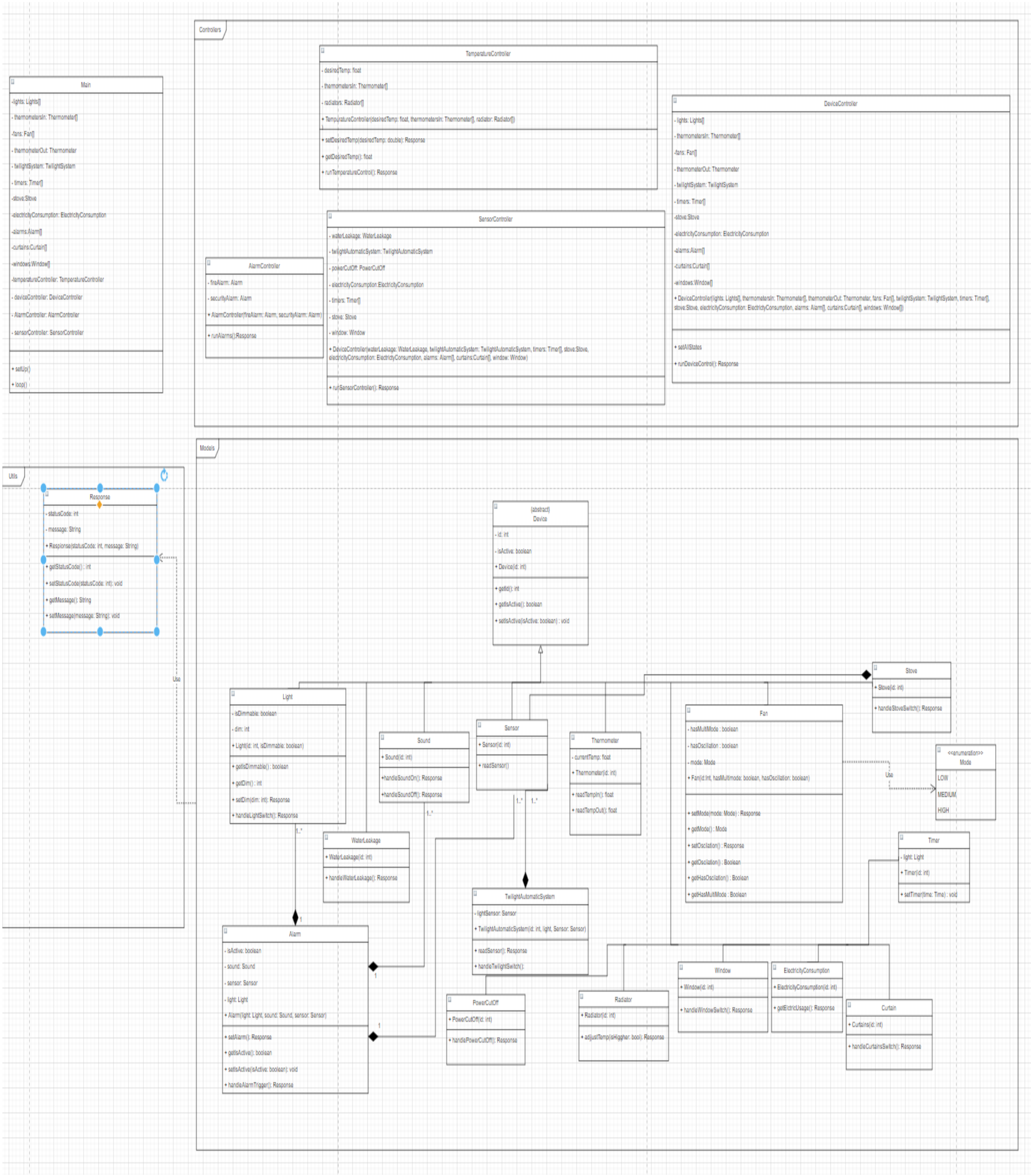

Controllers
Main
-lights: Lights[]
- thermometersIn: Thermometer[]
-fans: Fan[]
- thermometerOut: Thermometer
- twilightSystem: TwilightSystem
- timers: Timer[]
-stove:Stove
-electricityConsumption: ElectricityConsumption
-alarms:Alarm[]
-curtains:Curtain[]
-windows:Window[]
-temperatureController: TemperatureController
- deviceController: DeviceController
- AlarmController: AlarmController
- sensorController: SensorController
+ setUp()
+ loop()
TemperatureController
- desiredTemp: float
- thermometersIn: Thermometer[]
- radiators: Radiator[]
+ TempuratureController(desiredTemp: float, thermometersIn: Thermometer[], radiator: Radiator[])
+ setDesiredTemp(desiredTemp: double): Response
+ getDesiredTemp(): float
+ runTemperatureControl(): Response
SensorController
- waterLeakage: WaterLeakage
- twilightAutomaticSystem: TwilightAutomaticSystem
- powerCutOff: PowerCutOff
- electricityConsumption:ElectricityConsumption
- timers: Timer[]
- stove: Stove
- window: Window
+ DeviceController(waterLeakage: WaterLeakage, twilightAutomaticSystem: TwilightAutomaticSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectrictyConsumption, alarms: Alarm[], curtains:Curtain[], window: Window)
+ runSensorController(): Response
AlarmController
- fireAlarm: Alarm
- securityAlarm: Alarm
+ AlarmController(fireAlarm: Alarm, securityAlarm: Alarm)
+ runAlarms():Response
DeviceController
- lights: Lights[]
- thermometersIn: Thermometer[]
-fans: Fan[]
- thermometerOut: Thermometer
- twilightSystem: TwilightSystem
- timers: Timer[]
-stove:Stove
-electricityConsumption: ElectricityConsumption
-alarms:Alarm[]
-curtains:Curtain[]
-windows:Window[]
+ DeviceController(lights: Lights[], thermometersIn: Thermometer[], thermometerOut: Thermometer, fans: Fan[], twilightSystem: TwilightSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectrictyConsumption, alarms: Alarm[], curtains:Curtain[], windows: Window[])
+ setAllStates
+ runDeviceControl(): Response
Utils
Response
- statusCode: int
- message: String
+ Response(statusCode: int, message: String)
+ getStatusCode() : int
+ setStatusCode(statusCode: int): void
+ getMessage(): String
+ setMessage(message: String): void
Use
Models
{abstract}
Device
- id: int
- isActive: boolean
+ Device(id: int)
+ getId(): int
+ getIsActive(): boolean
+ setIsActive(isActive: boolean) : void
Light
- isDimmable: boolean
- dim: int
+ Light(id: int, isDimmable: boolean)
+ getIsDimmable() : boolean
+ getDim() : int
+ setDim(dim: int): Response
+ handleLightSwitch(): Response
Sound
+ Sound(id: int)
+handleSoundOn(): Response
+handleSoundOff(): Response
Sensor
+ Sensor(id: int)
+ readSensor()
Thermometer
- currentTemp: float
+ Thermometer(id: int)
+ readTempIn(): float
+ readTempOut(): float
Stove
+ Stove(id: int)
+ handleStoveSwitch(): Response
Fan
- hasMultiMode : boolean
- hasOscilation : boolean
- mode: Mode
+ Fan(id:Int, hasMultimode: boolean, hasOscilation: boolean)
+ setMode(mode: Mode) : Response
+ getMode() : Mode
+ setOsclation() : Response
+ getOsclation() : Boolean
+ getHasOsclation() : Boolean
+ getHasMultiMode : Boolean
<<enumeration>>
Mode
LOW
MEDIUM
HIGH
Use
Timer
- light: Light
+ Timer(id: int)
+ setTimer(time: Time) : void
WaterLeakage
+ WaterLeakage(id: int)
+ handleWaterLeakage(): Response
TwilightAutomaticSystem
- lightSensor: Sensor
+ TwilightAutomaticSystem(id: int, light, Sensor: Sensor)
+ readSensor(): Response
+ handleTwilightSwitch():
Alarm
- isActive: boolean
- sound: Sound
- sensor: Sensor
- light: Light
+ Alarm(light: Light, sound: Sound, sensor: Sensor)
+ setAlarm(): Response
+ getIsActive(): boolean
+ setIsActive(isActive: boolean): void
+ handleAlarmTrigger(): Response
PowerCutOff
+ PowerCutOff(id: int)
+ handlePowerCutOff(): Response
Radiator
+ Radiator(id: int)
+ adjustTemp(isHiggher: bool): Response
Window
+ Window(id: int)
+ handleWindowSwitch(): Response
ElectricityConsumption
+ ElectricityConsumption(id: int)
+ getElctricUsage(): Response
Curtain
+ Curtains(id: int)
+ handleCurtainsSwitch(): Response
1..*
1..*
1..*
1..*
1
1
1

Figure 3 - Class Diagram

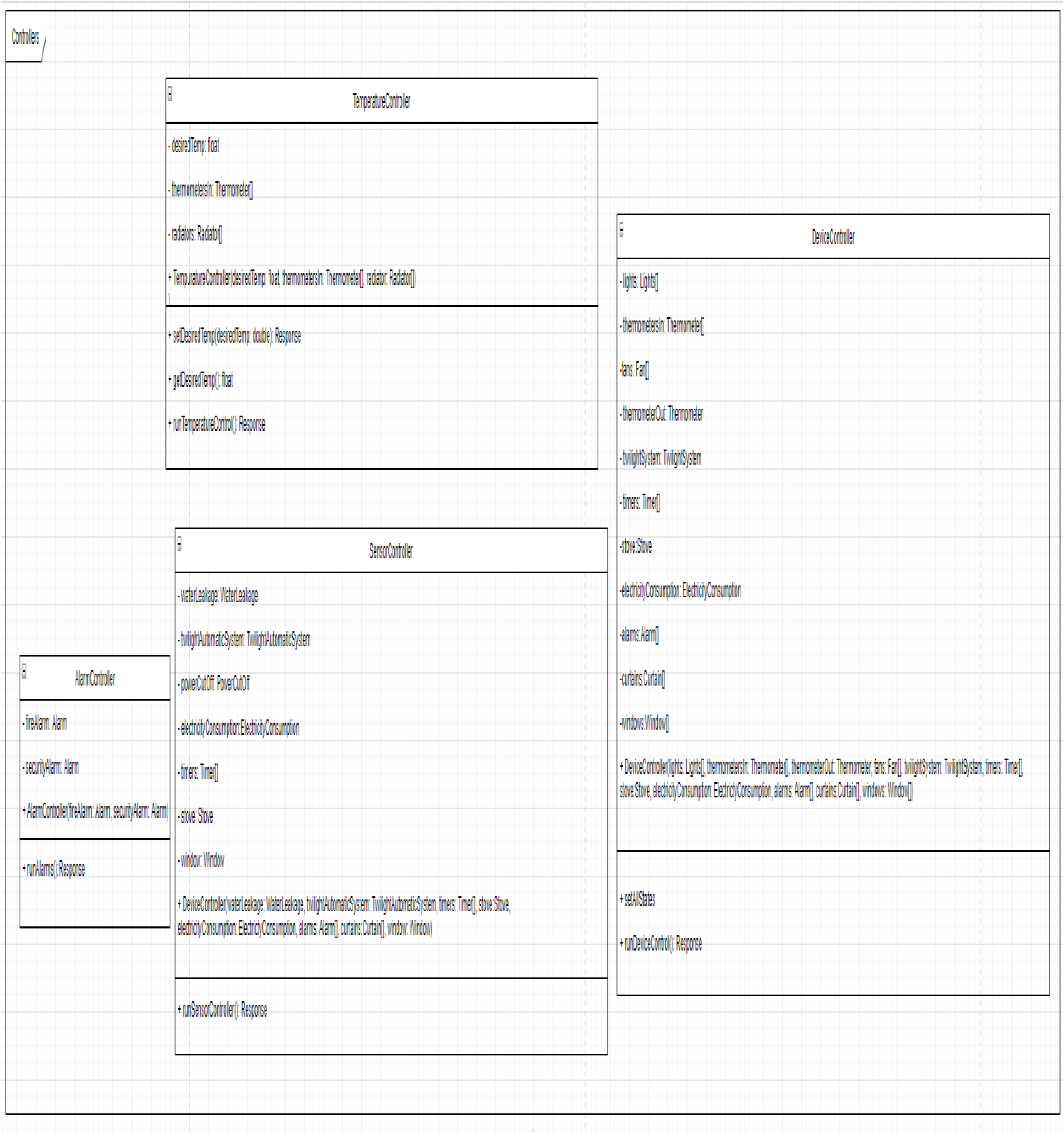

Controllers
TemperatureController
- desiredTemp: float
- thermometersIn: Thermometer[]
- radiators: Radiator[]
+ TempuratureController(desiredTemp: float, thermometersIn: Thermometer[], radiator: Radiator[])
\
+ setDesiredTemp(desiredTemp: double): Response
+ getDesiredTemp(): float
+ runTemperatureControl(): Response
DeviceController
- lights: Lights[]
- thermometersIn: Thermometer[]
-fans: Fan[]
- thermometerOut: Thermometer
- twilightSystem: TwilightSystem
- timers: Timer[]
-stove:Stove
-electricityConsumption: ElectricityConsumption
-alarms:Alarm[]
-curtains:Curtain[]
-windows:Window[]
+ DeviceController(lights: Lights[], thermometersIn: Thermometer[], thermometerOut: Thermometer, fans: Fan[], twilightSystem: TwilightSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectricityConsumption, alarms: Alarm[], curtains:Curtain[], windows: Window[])
+ setAllStates
+ runDeviceControl(): Response
SensorController
- waterLeakage: WaterLeakage
- twilightAutomaticSystem: TwilightAutomaticSystem
- powerCutOff: PowerCutOff
- electricityConsumption:ElectricityConsumption
- timers: Timer[]
- stove: Stove
- window: Window
+ DeviceController(waterLeakage: WaterLeakage, twilightAutomaticSystem: TwilightAutomaticSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectricityConsumption, alarms: Alarm[], curtains:Curtain[], window: Window)
+ runSensorController(): Response
AlarmController
- fireAlarm: Alarm
- securityAlarm: Alarm
+ AlarmController(fireAlarm: Alarm, securityAlarm: Alarm)
+ runAlarms():Response

Figure 4 - Controllers

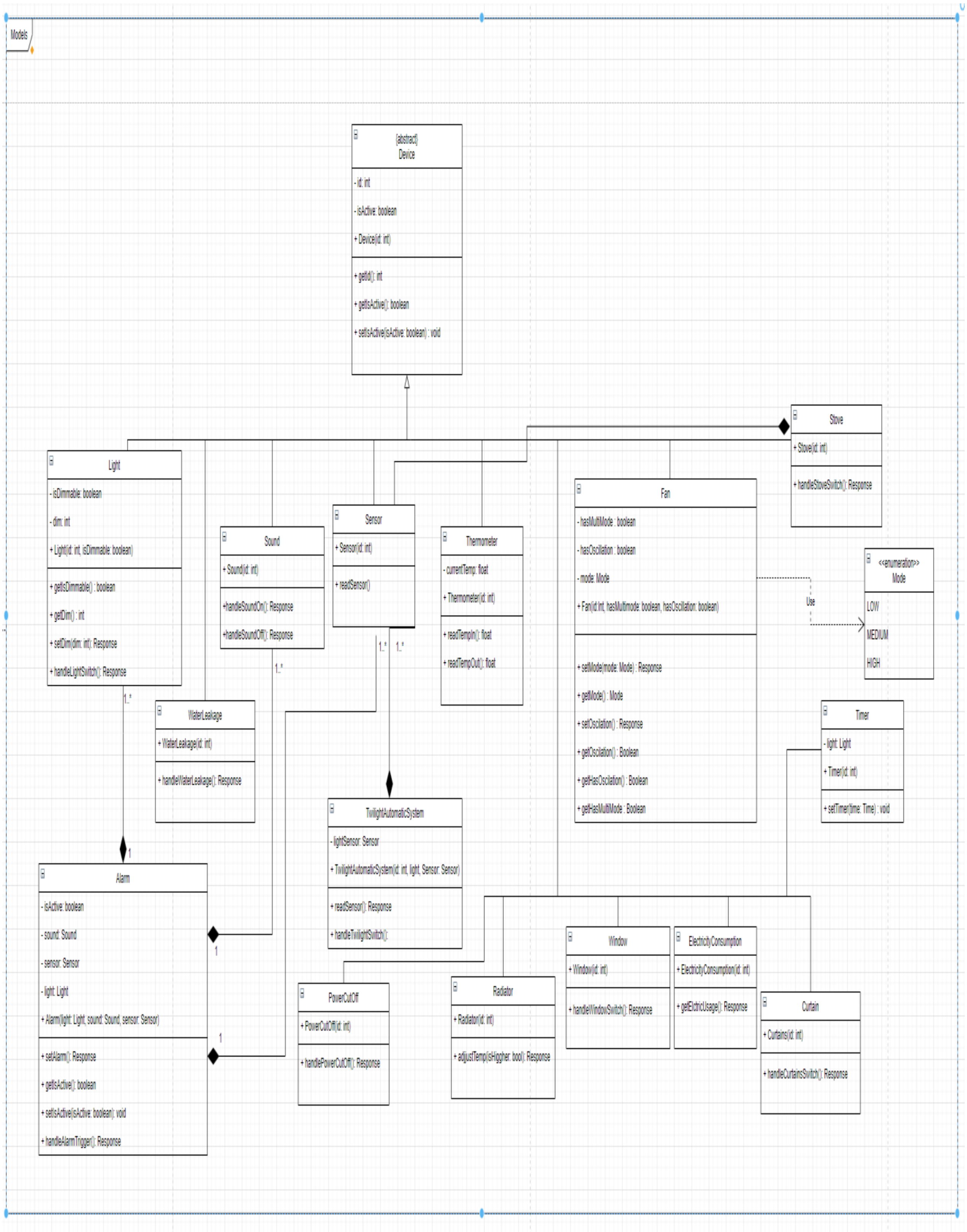

Models
[abstract]
Device
- id: int
- isActive: boolean
+ Device(id: int)
+ getId(): int
+ getIsActive(): boolean
+ setIsActive(isActive: boolean) : void
Stove
+ Stove(id: int)
+ handleStoveSwitch(): Response
Light
- isDimmable: boolean
- dim: int
+ Light(id: int, isDimmable: boolean)
+ getIsDimmable() : boolean
+ getDim() : int
+ setDim(dim: int): Response
+ handleLightSwitch(): Response
Sound
+ Sound(id: int)
+handleSoundOn(): Response
+handleSoundOff(): Response
Sensor
+ Sensor(id: int)
+ readSensor()
Thermometer
- currentTemp: float
+ Thermometer(id: int)
+ readTempIn(): float
+ readTempOut(): float
Fan
- hasMultiMode : boolean
- hasOscillation : boolean
- mode: Mode
+ Fan(id:Int, hasMultimode: boolean, hasOscillation: boolean)
+ setMode(mode: Mode) : Response
+ getMode() : Mode
+ setOscilation() : Response
+ getOscilation() : Boolean
+ getHasOscilation() : Boolean
+ getHasMultiMode : Boolean
Use
<<enumeration>>
Mode
LOW
MEDIUM
HIGH
1..*
1..*
1..*
1..*
WaterLeakage
+ WaterLeakage(id: int)
+ handleWaterLeakage(): Response
Timer
- light: Light
+ Timer(id: int)
+ setTimer(time: Time) : void
TwilightAutomaticSystem
- lightSensor: Sensor
+ TwilightAutomaticSystem(id: int, light, Sensor: Sensor)
+ readSensor(): Response
+ handleTwilightSwitch():
1
Alarm
- isActive: boolean
- sound: Sound
- sensor: Sensor
- light: Light
+ Alarm(light: Light, sound: Sound, sensor: Sensor)
+ setAlarm(): Response
+ getIsActive(): boolean
+ setIsActive(isActive: boolean): void
+ handleAlarmTrigger(): Response
1
1
Window
+ Window(id: int)
+ handleWindowSwitch(): Response
ElectricityConsumption
+ ElectricityConsumption(id: int)
+ getElctricUsage(): Response
PowerCutOff
+ PowerCutOff(id: int)
+ handlePowerCutOff(): Response
Radiator
+ Radiator(id: int)
+ adjustTemp(isHiggher: bool): Response
Curtain
+ Curtains(id: int)
+ handleCurtainsSwitch(): Response

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 cases it has just a single pin, 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 responsefrom our Response class.

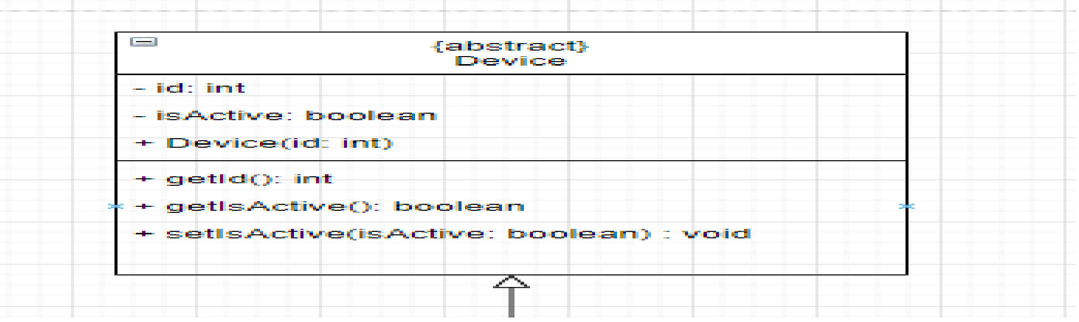


Figure 6 - Abstract Device Class

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 will act as the siren in the composition in the alarm class. The sensor class which also is apart of the composition in the alarm class aswell as twilight automatic system class, window class, water leakage class and stove class. The other classes remaining are power cut off class, timer class, thermometer class, curtains class, electricity consumption class, radiator class and fan class.

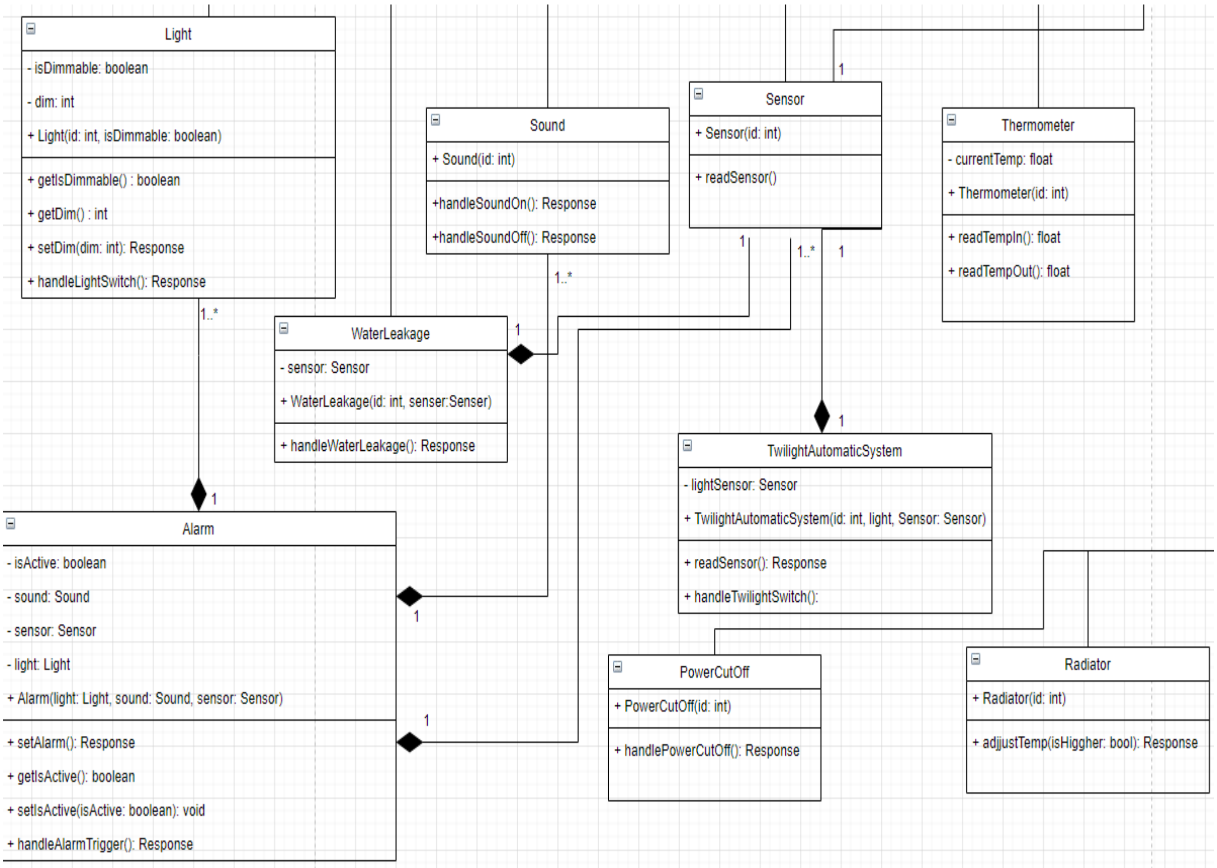

Light
- isDimmable: boolean
- dim: int
+ Light(id: int, isDimmable: boolean)
+ getIsDimmable() : boolean
+ getDim() : int
+ setDim(dim: int): Response
+ handleLightSwitch(): Response
Sound
+ Sound(id: int)
+handleSoundOn(): Response
+handleSoundOff(): Response
Sensor
+ Sensor(id: int)
+ readSensor()
Thermometer
- currentTemp: float
+ Thermometer(id: int)
+ readTempIn(): float
+ readTempOut(): float
WaterLeakage
- sensor: Sensor
+ WaterLeakage(id: int, senser:Senser)
+ handleWaterLeakage(): Response
TwilightAutomaticSystem
- lightSensor: Sensor
+ TwilightAutomaticSystem(id: int, light, Sensor: Sensor)
+ readSensor(): Response
+ handleTwilightSwitch():
Alarm
- isActive: boolean
- sound: Sound
- sensor: Sensor
- light: Light
+ Alarm(light: Light, sound: Sound, sensor: Sensor)
+ setAlarm(): Response
+ getIsActive(): boolean
+ setIsActive(isActive: boolean): void
+ handleAlarmTrigger(): Response
PowerCutOff
+ PowerCutOff(id: int)
+ handlePowerCutOff(): Response
Radiator
+ Radiator(id: int)
+ adjjustTemp(isHiggher: bool): Response
1
1..*
1
1..*
1
1
1..*
1
1
1
1
1

Figure 7 - Classes That Inherit From Device Light, Water Leakage, Sound, PowerCutOff, Radiator, Sensor, and Thermometer

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 and returns a response. The sound class has a handleSoundOn and a handleSoundOff functions both of which return a response the constructor takes only the id. The sensor class has a readSensor function the constructor contains only the id. Thermometer class has a variable currentTemp a constructor which takes id, the function it uses is the readTempIn and the readTempOut both return a response. The water leakage class has a variable sensor which is passed through the constructor and id, the function is handleWaterLeakage which returns a response. The power cut off class has a constructor and takes an id, the function is handlePowerCutOff which returns a response. The radiator class has a constructor and takes an id, the function is adjustTemp which takes a Boolean for a parameter which returns a response.

The window class (Figure 8) has a handleWindowSwitch function the constructor contains only the id. The electricity consumption class has a readElectricUsage function the constructor contains only the id. The timer class has a setTimer which takes some form of measurement of time of which is not seleceted yet for the function the constructor contains only the id. The other device class is the curtains class which only contains a constructor which takes an id and the handle.

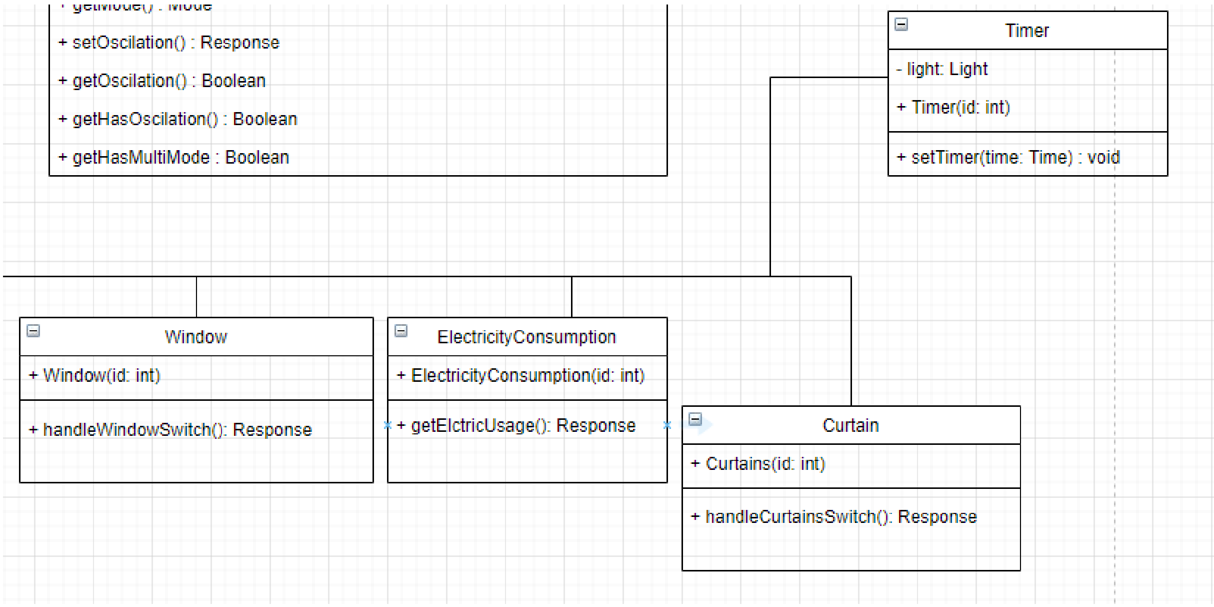

+ getMode() : Mode
+ setOscilation() : Response
+ getOscilation() : Boolean
+ getHasOscilation() : Boolean
+ getHasMultiMode : Boolean
Timer
- light: Light
+ Timer(id: int)
+ setTimer(time: Time) : void
Window
+ Window(id: int)
+ handleWindowSwitch(): Response
ElectricityConsumption
+ ElectricityConsumption(id: int)
+ getElctricUsage(): Response
Curtain
+ Curtains(id: int)
+ handleCurtainsSwitch(): Response

Figure 8 - Classes That Inherit From Device Window, Electricity Consumption, Timer, and Curtain

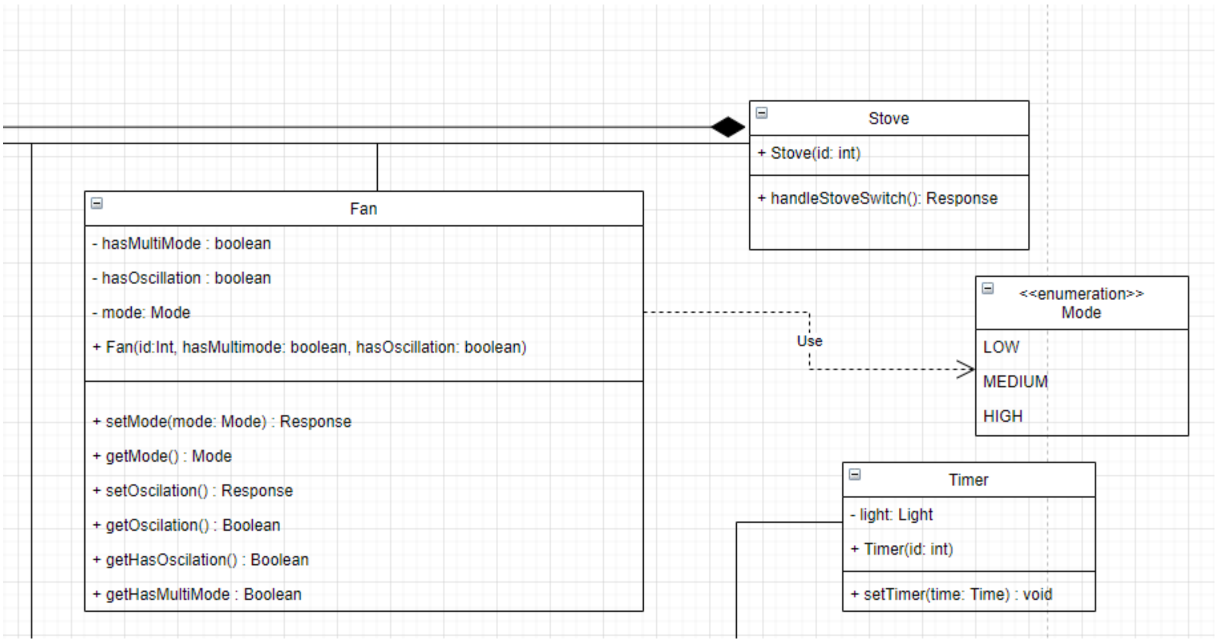

Stove
+ Stove(id: int)
+ handleStoveSwitch(): Response
Fan
- hasMultiMode : boolean
- hasOscillation : boolean
- mode: Mode
+ Fan(id:Int, hasMultimode: boolean, hasOscillation: boolean)
+ setMode(mode: Mode) : Response
+ getMode() : Mode
+ setOscilation() : Response
+ getOscilation() : Boolean
+ getHasOscilation() : Boolean
+ getHasMultiMode : Boolean
Use
<<enumeration>>
Mode
LOW
MEDIUM
HIGH
Timer
- light: Light
+ Timer(id: int)
+ setTimer(time: Time) : void

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 Mode as a parameter and gets a response as the return type, getMode which returns mode, setOscillation which returns a response 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 handleStoveSwitch which returns a response.

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 returns a response, 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 returns a response.

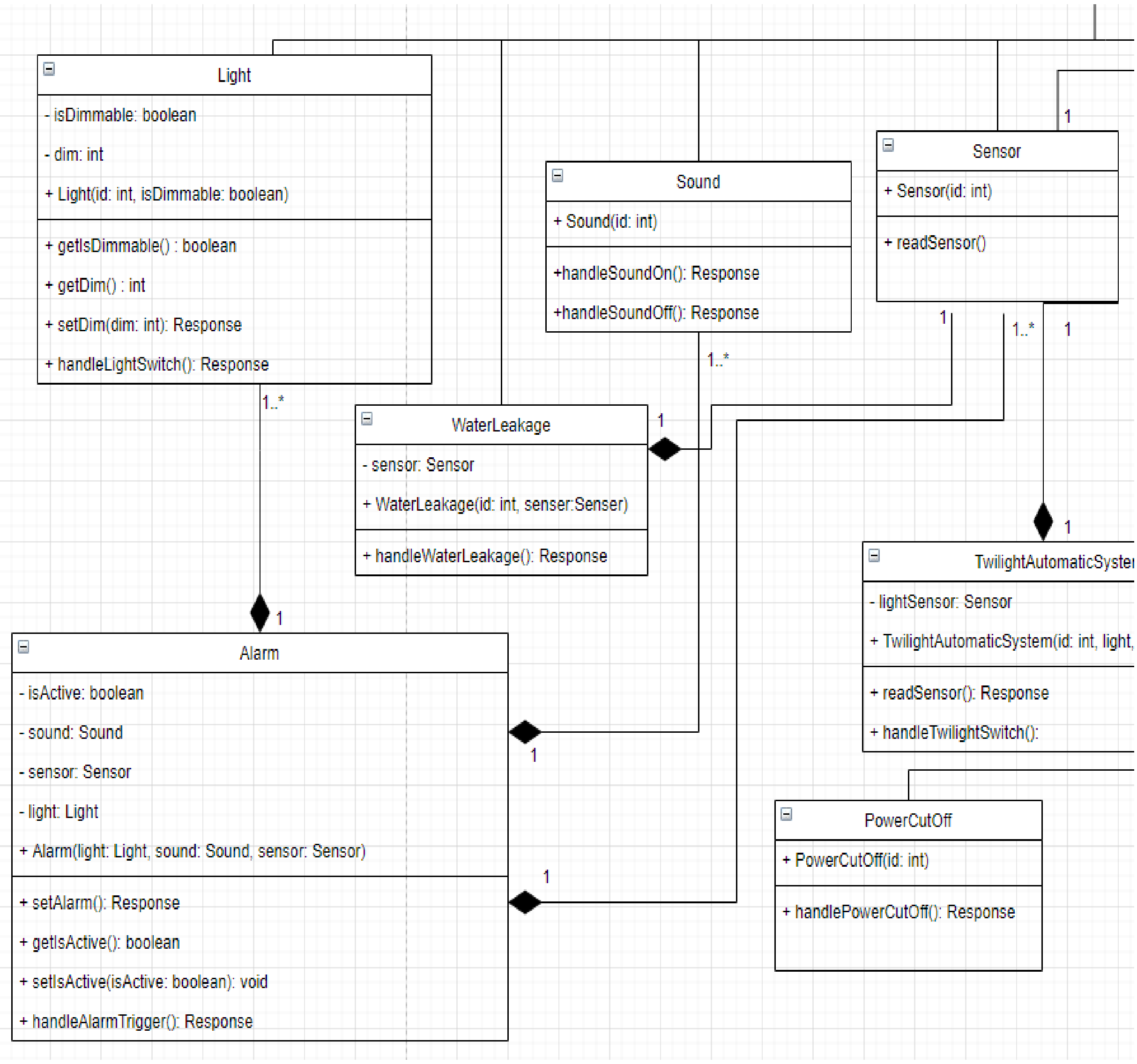

Light
- isDimmable: boolean
- dim: int
+ Light(id: int, isDimmable: boolean)
+ getIsDimmable() : boolean
+ getDim() : int
+ setDim(dim: int): Response
+ handleLightSwitch(): Response
Sound
+ Sound(id: int)
+handleSoundOn(): Response
+handleSoundOff(): Response
Sensor
+ Sensor(id: int)
+ readSensor()
WaterLeakage
- sensor: Sensor
+ WaterLeakage(id: int, senser:Senser)
+ handleWaterLeakage(): Response
TwilightAutomaticSyster
- lightSensor: Sensor
+ TwilightAutomaticSystem(id: int, light,
+ readSensor(): Response
+ handleTwilightSwitch():
Alarm
- isActive: boolean
- sound: Sound
- sensor: Sensor
- light: Light
+ Alarm(light: Light, sound: Sound, sensor: Sensor)
+ setAlarm(): Response
+ getIsActive(): boolean
+ setIsActive(isActive: boolean): void
+ handleAlarmTrigger(): Response
PowerCutOff
+ PowerCutOff(id: int)
+ handlePowerCutOff(): Response
1
1
1..*
1
1..*
1
1..*
1
1
1
1
1

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 hnadleTwilightSwitch functions.

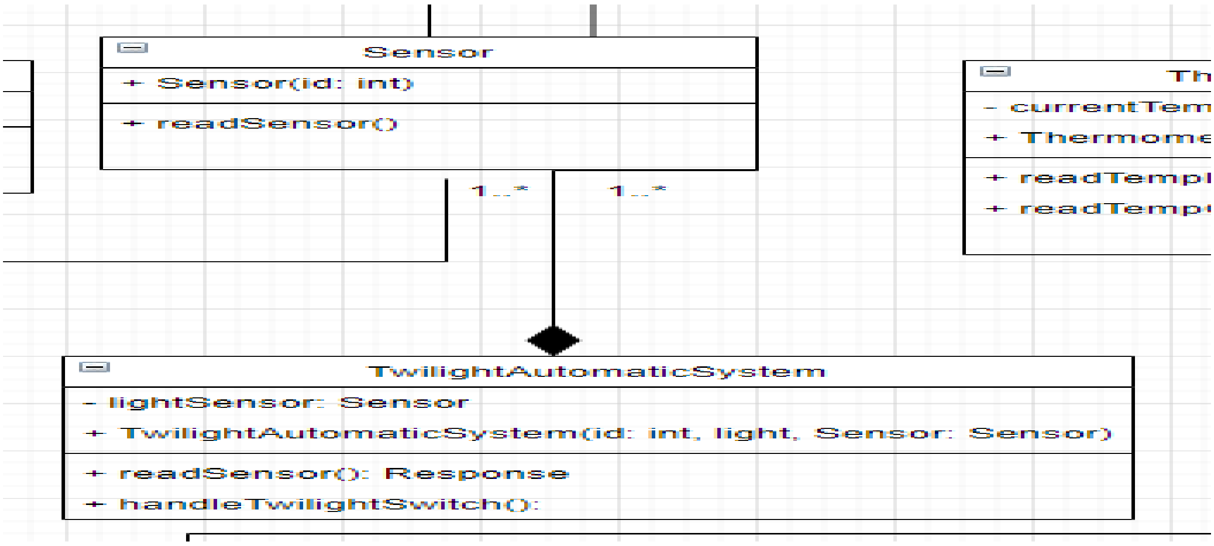


Figure 11 - Twilight Automatic System

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 double and the thermometerIn of the thermometer device. The functions used in the temp. controller class is the setDesiredTemp which takes desiredTemp double as a parameter and returns a response the getDesiredTemp function which returns a double and runTemperatureControl which returns a response.

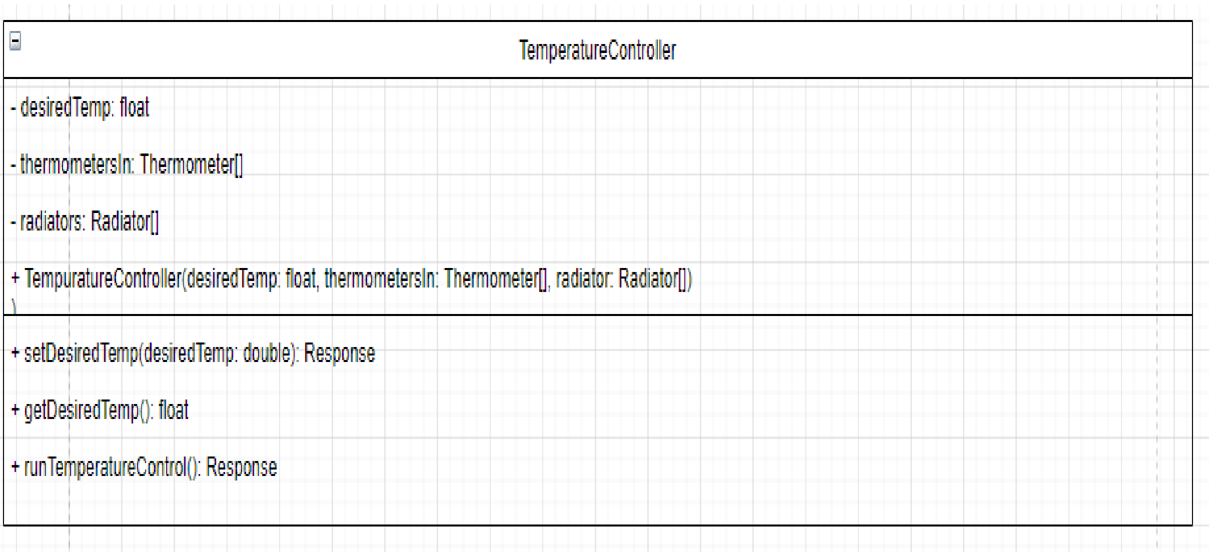
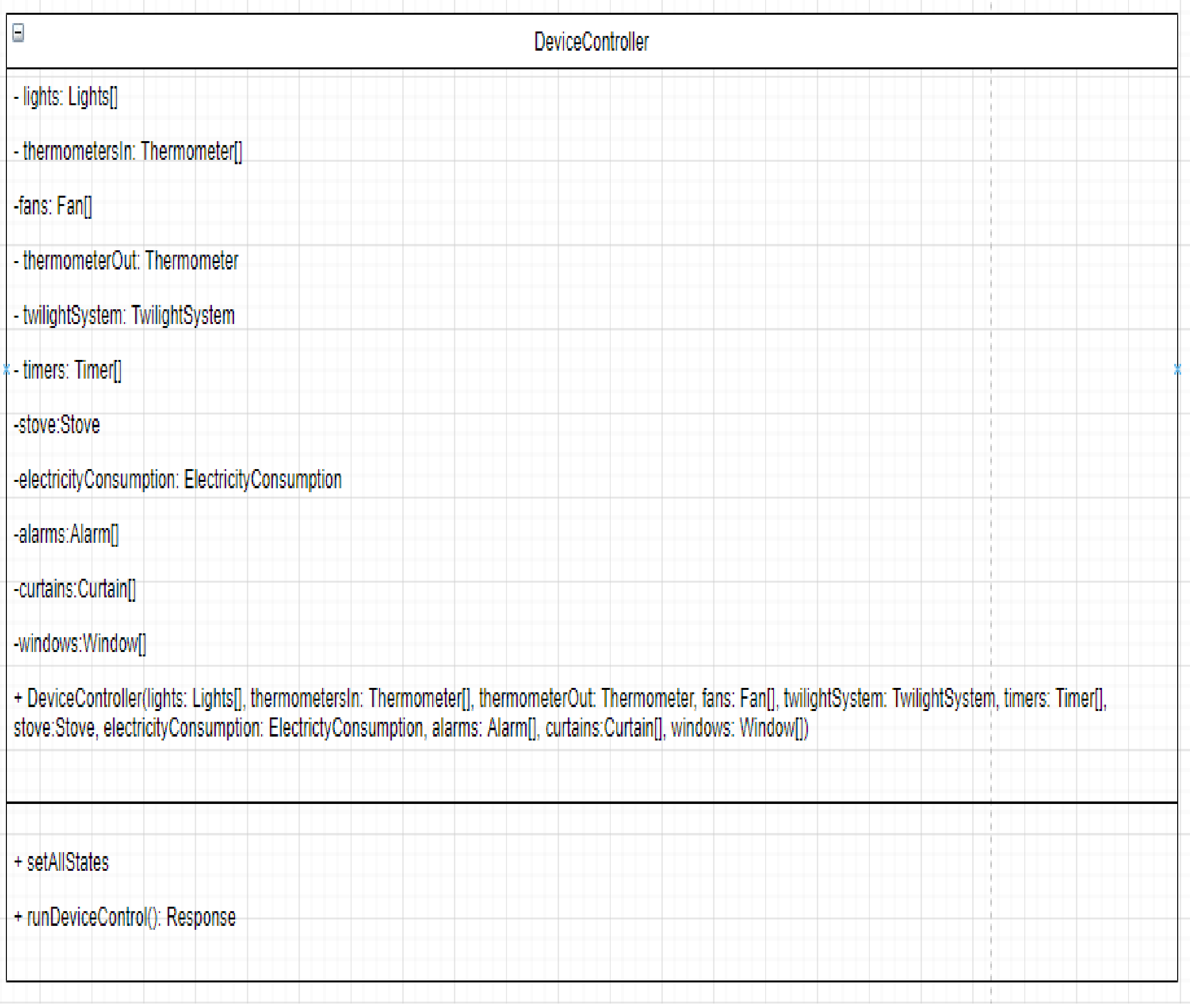

TemperatureController
- desiredTemp: float
- thermometersIn: Thermometer[]
- radiators: Radiator[]
+ TempuratureController(desiredTemp: float, thermometersIn: Thermometer[], radiator: Radiator[])
)
+ setDesiredTemp(desiredTemp: double): Response
+ getDesiredTemp(): float
+ runTemperatureControl(): Response

Figure 12 - Temperature Controller

D8

The device controller class (Figure 13) will be where the commands 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 runDeviceControl and setAllStates functions, they will handle the user requests and set the initial states of the devices when it boots.



DeviceController

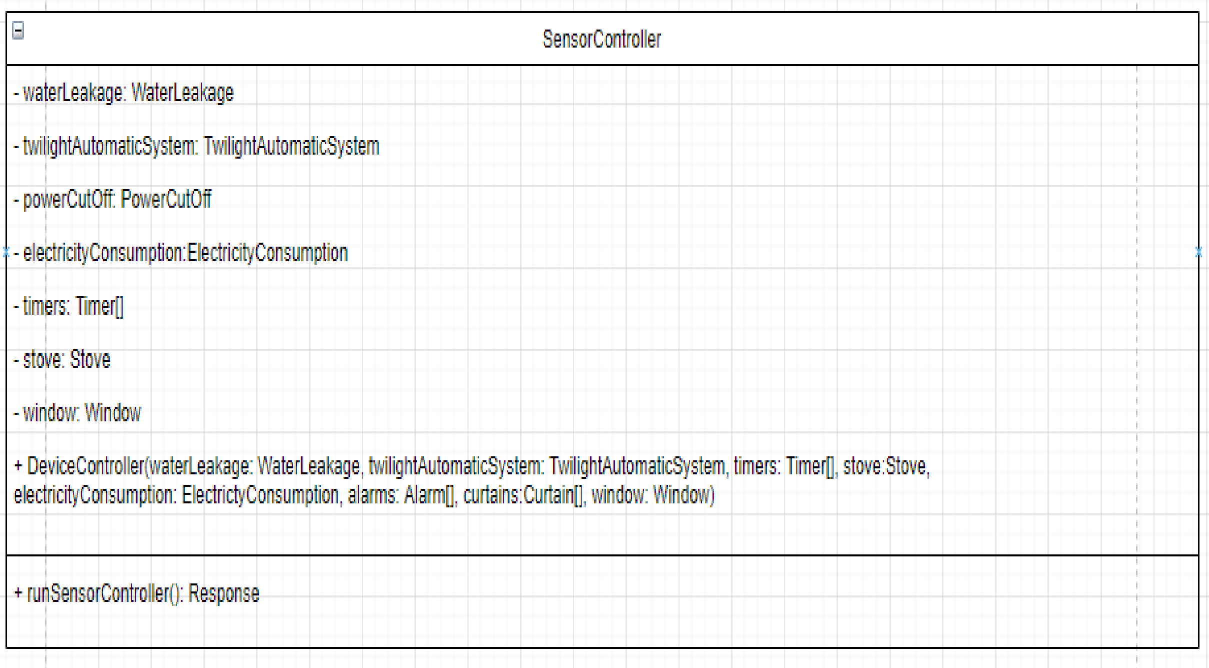
- lights: Lights[]
- thermometersIn: Thermometer[]
-fans: Fan[]
- thermometerOut: Thermometer
- twilightSystem: TwilightSystem
- timers: Timer[]
-stove:Stove
-electricityConsumption: ElectricityConsumption
-alarms:Alarm[]
-curtains:Curtain[]
-windows:Window[]
+ DeviceController(lights: Lights[], thermometersIn: Thermometer[], thermometerOut: Thermometer, fans: Fan[], twilightSystem: TwilightSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectrictyConsumption, alarms: Alarm[], curtains:Curtain[], windows: Window[])

+ setAllStates
+ runDeviceControl(): Response

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 which returns a response.



SensorController

- waterLeakage: WaterLeakage
- twilightAutomaticSystem: TwilightAutomaticSystem
- powerCutOff: PowerCutOff
- electricityConsumption:ElectricityConsumption
- timers: Timer[]
- stove: Stove
- window: Window

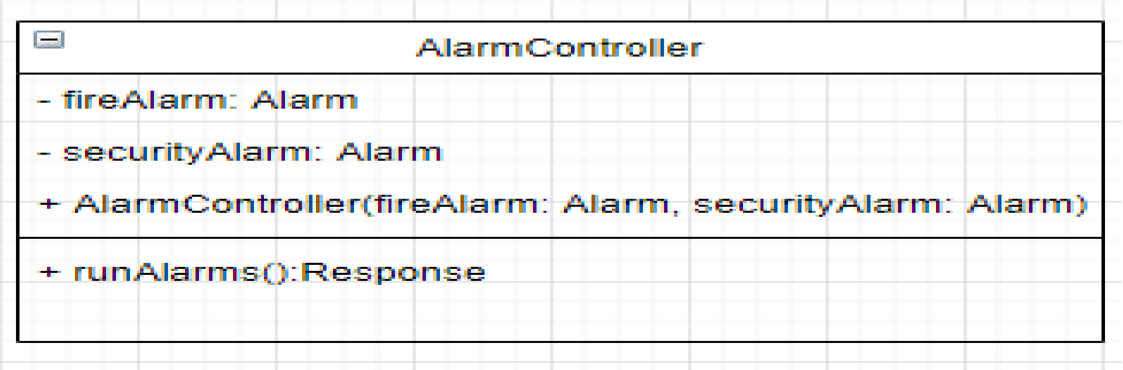
+ DeviceController(waterLeakage: WaterLeakage, twilightAutomaticSystem: TwilightAutomaticSystem, timers: Timer[], stove:Stove, electricityConsumption: ElectrictyConsumption, alarms: Alarm[], curtains:Curtain[], window: Window)

+ runSensorController(): Response

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.



AlarmController

- fireAlarm: Alarm
- securityAlarm: Alarm

+ AlarmController(fireAlarm: Alarm, securityAlarm: Alarm)

+ runAlarms():Response

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.

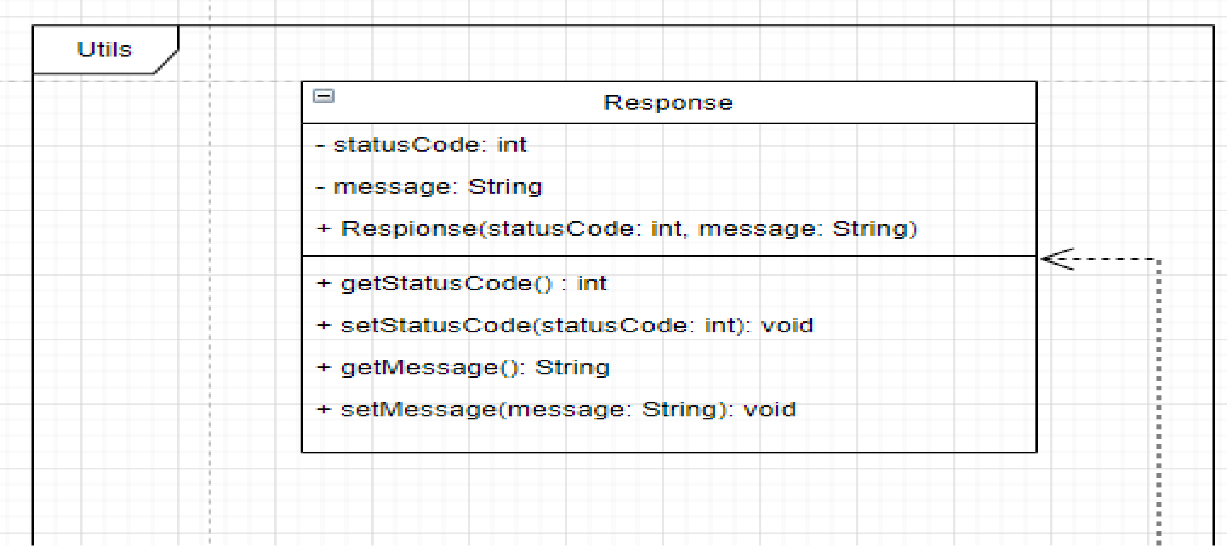
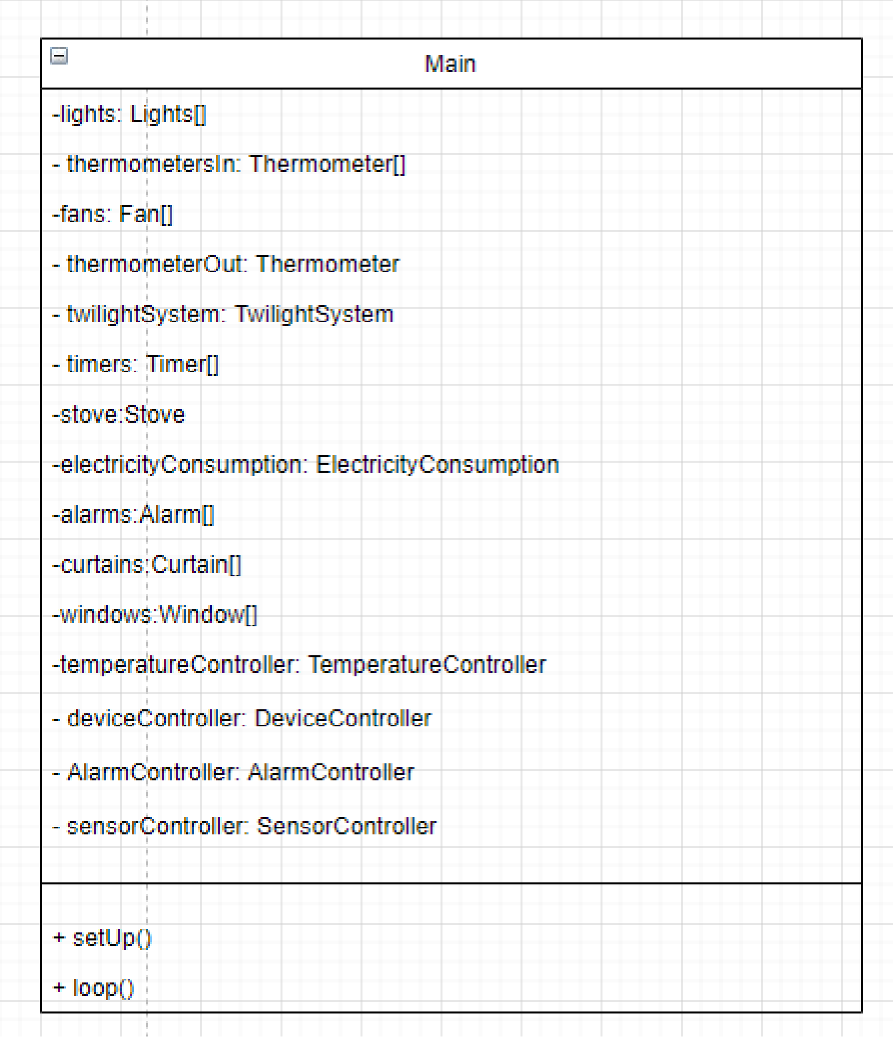

Utils
Response
- statusCode: int
- message: String
+ Respionse(statusCode: int, message: String)
+ getStatusCode() : int
+ setStatusCode(statusCode: int): void
+ getMessage(): String
+ setMessage(message: String): void

Figure 16 - Response Class

D12

The main class (Figure 17) will handle the setting up of all classes.



Main

-lights: Lights[]

- thermometersIn: Thermometer[]

-fans: Fan[]

- thermometerOut: Thermometer

- twilightSystem: TwilightSystem

- timers: Timer[]

-stove:Stove

-electricityConsumption: ElectricityConsumption

-alarms:Alarm[]

-curtains:Curtain[]

-windows:Window[]

-temperatureController: TemperatureController

- deviceController: DeviceController

- AlarmController: AlarmController

- sensorController: SensorController

+ setUp()

+ loop()

Figure 17 – Main Class

D13

The sequence diagram (Figure 18) 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.

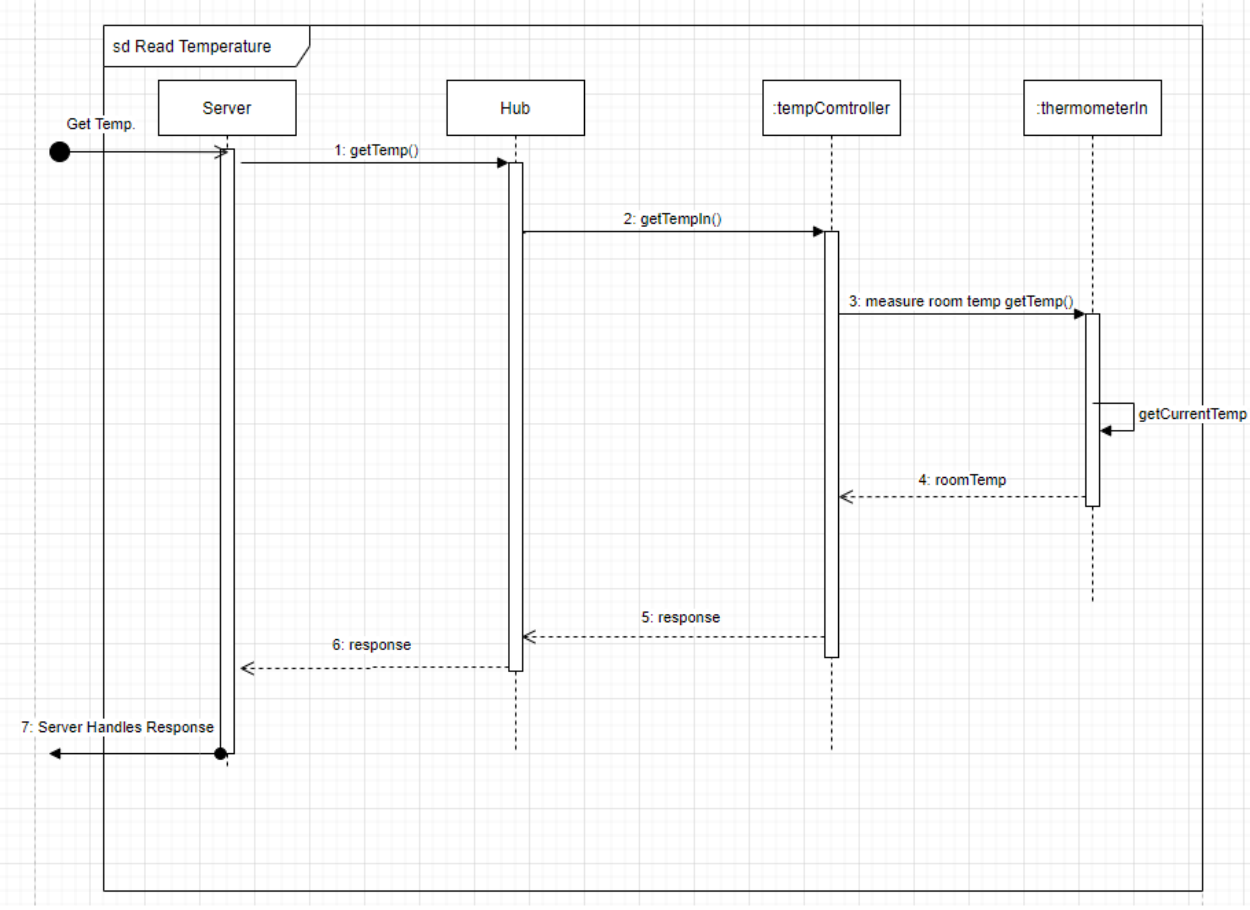


Figure 18- Sequence Diagram Read Temperature

D14

The sequence diagram (Figure 19) depicts the turning of a light on the Arduino (Hub), the server dispatches a request to the hub to turn the light on based off the id the hub then dispatches the request to the light. The light returns a message with a response and the hub in turn relays that response to the server, the server then handles the response.

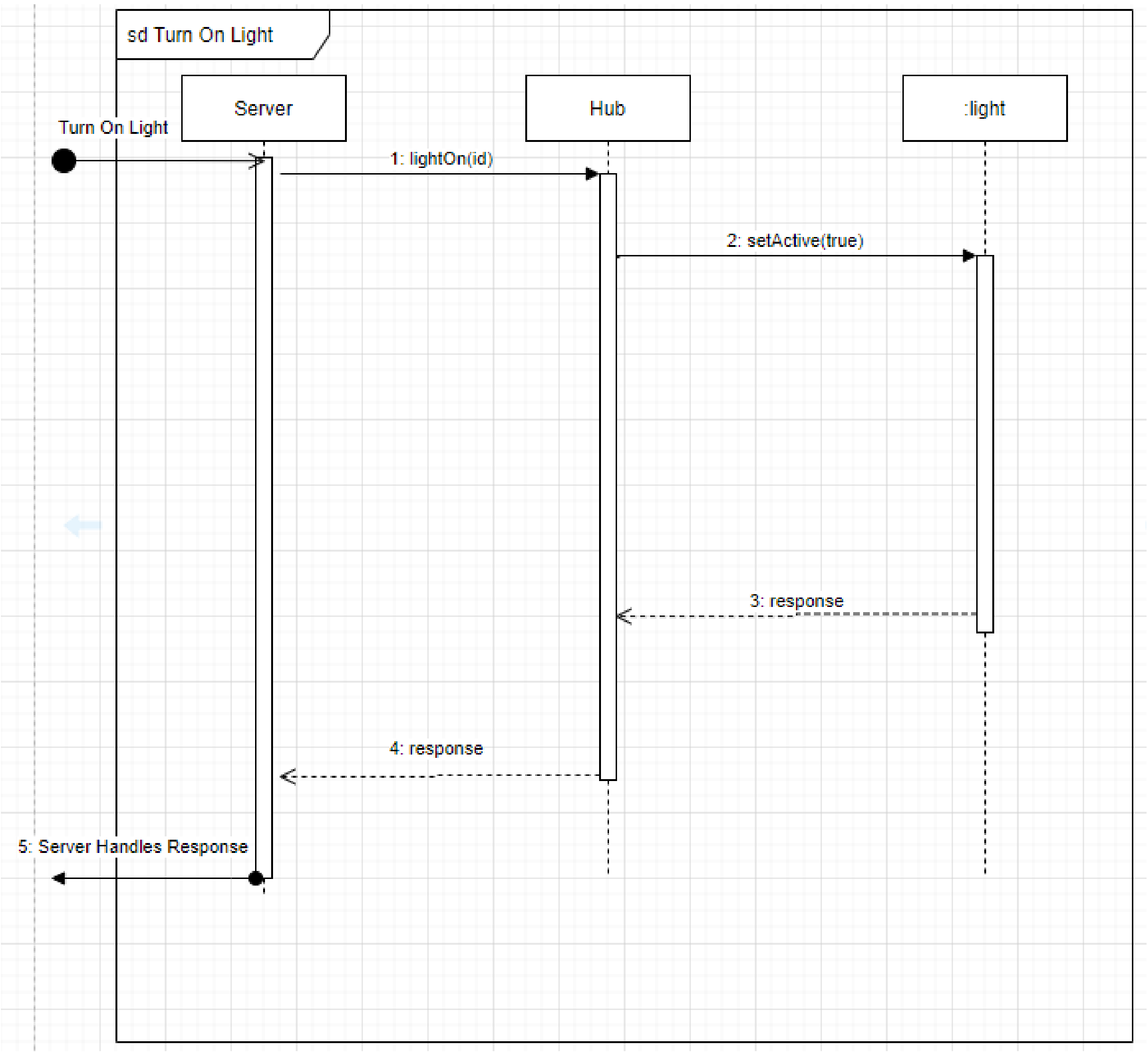


Figure 19 - Sequence Diagram Turn On Light

D15

The sequence diagram (Figure 20) depicts opening of a curtain on the Arduino (Hub), the server dispatches a request to the hub to open the curtain based off the id the hub then dispatches the request to the curtain. The curtain returns a message with a response and the hub in turn relays that response to the server, the server then handles the response.

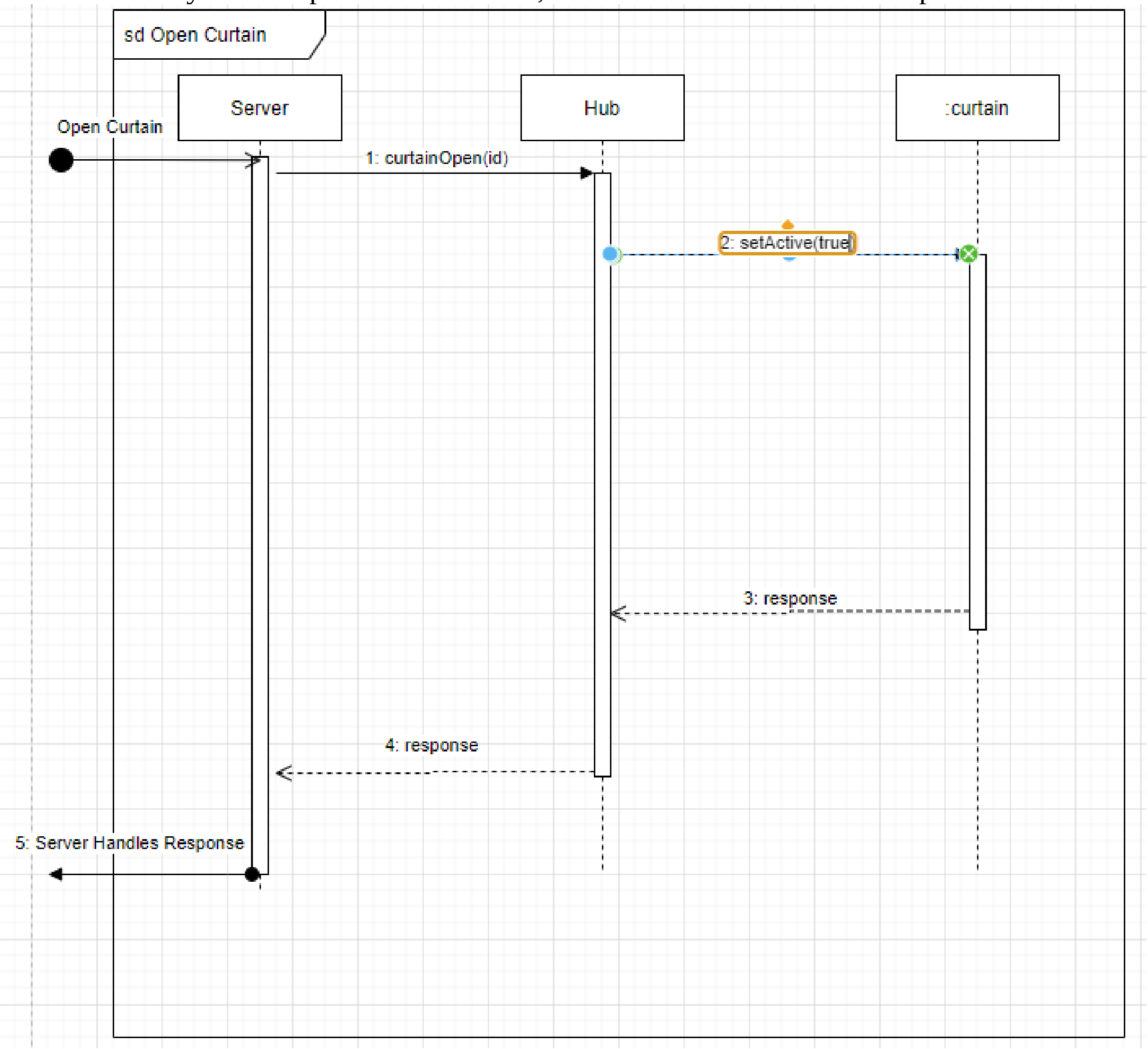


Figure 20 - Sequence Diagram Open Curtain

D16

The sequence diagram (Figure 21, Figure 22) 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.

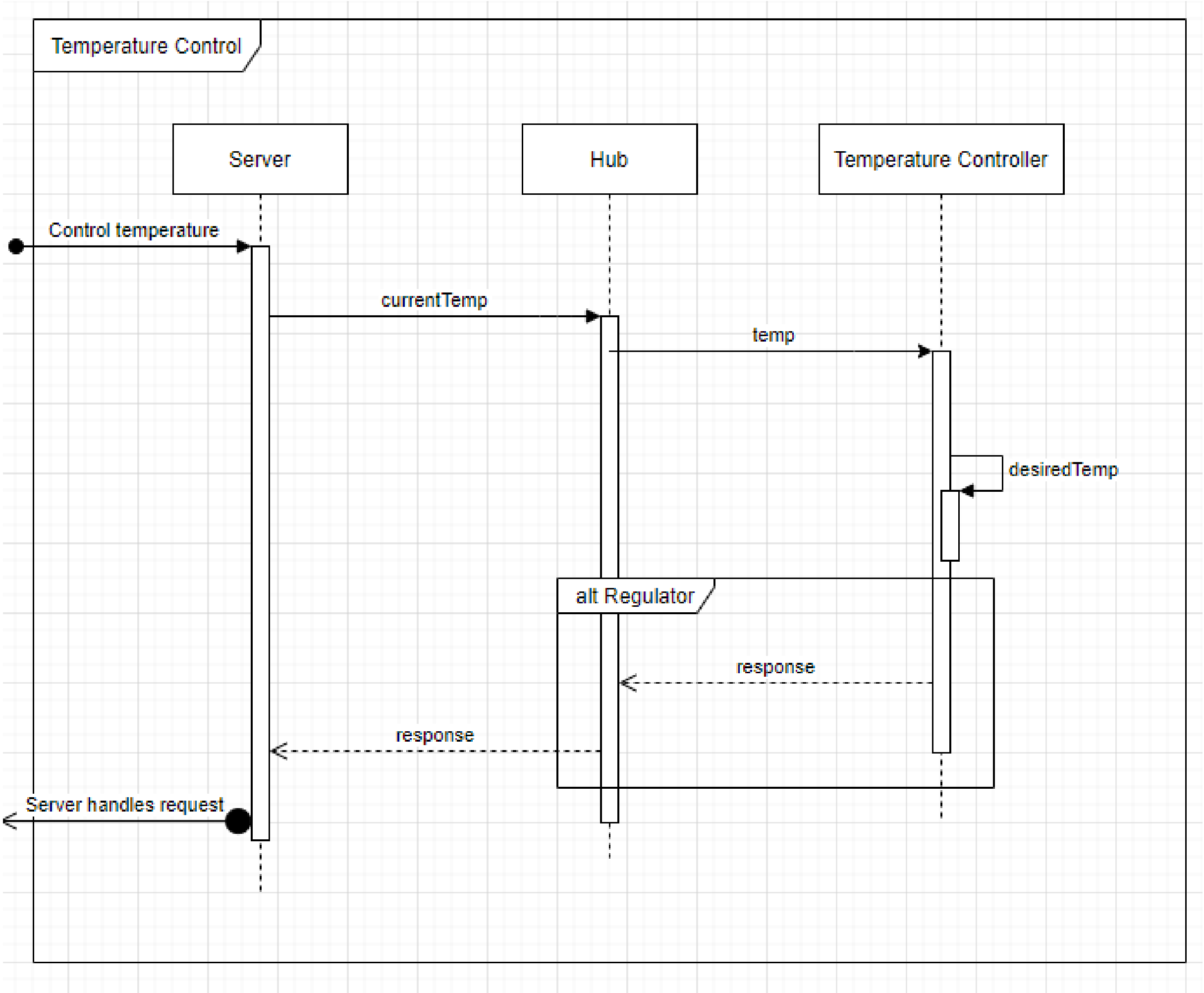


Figure 21 - Sequence Diagram Temperature Control

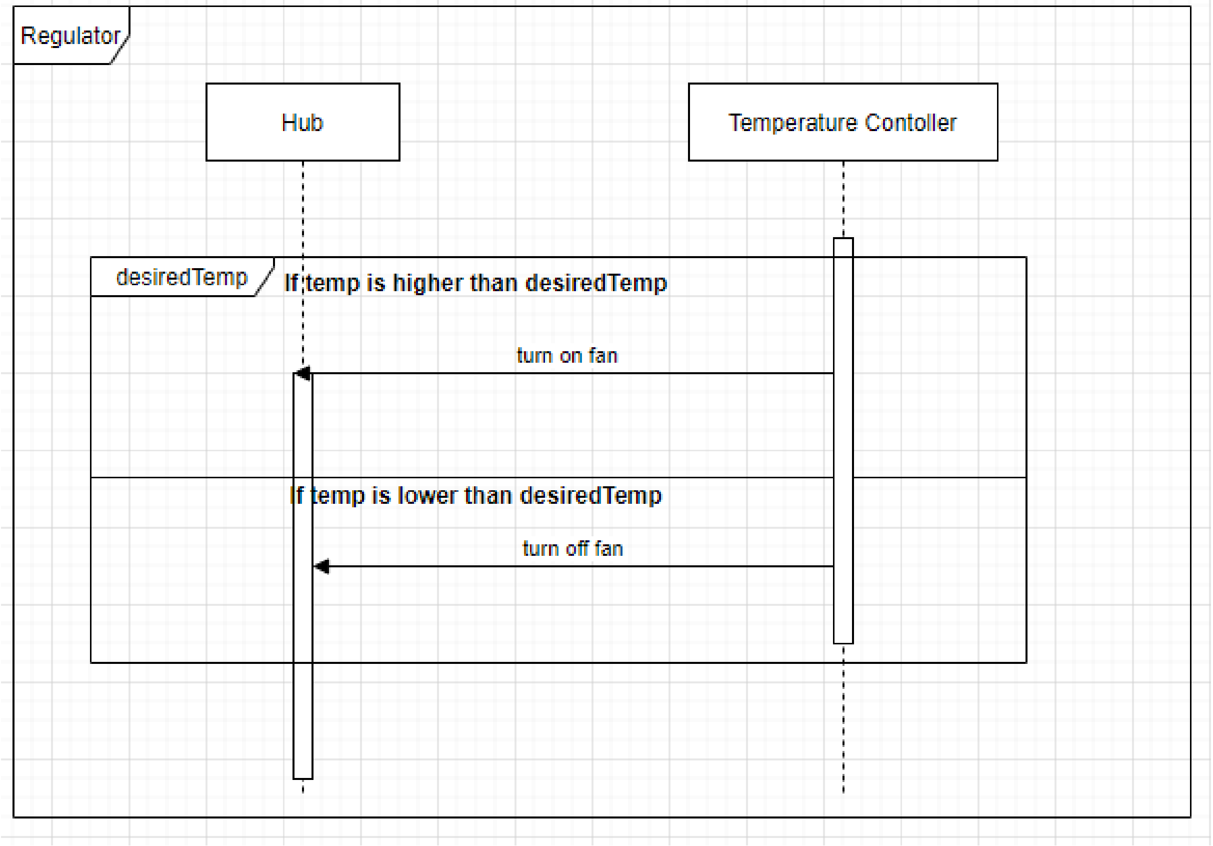

Regulator
Hub
Temperature Contoller
desiredTemp
If temp is higher than desiredTemp
turn on fan
If temp is lower than desiredTemp
turn off fan

Figure 22 - Sequence Diagram Regulator of Temperature Controller

D17

State diagram for the temperature controls system (Figure 23).

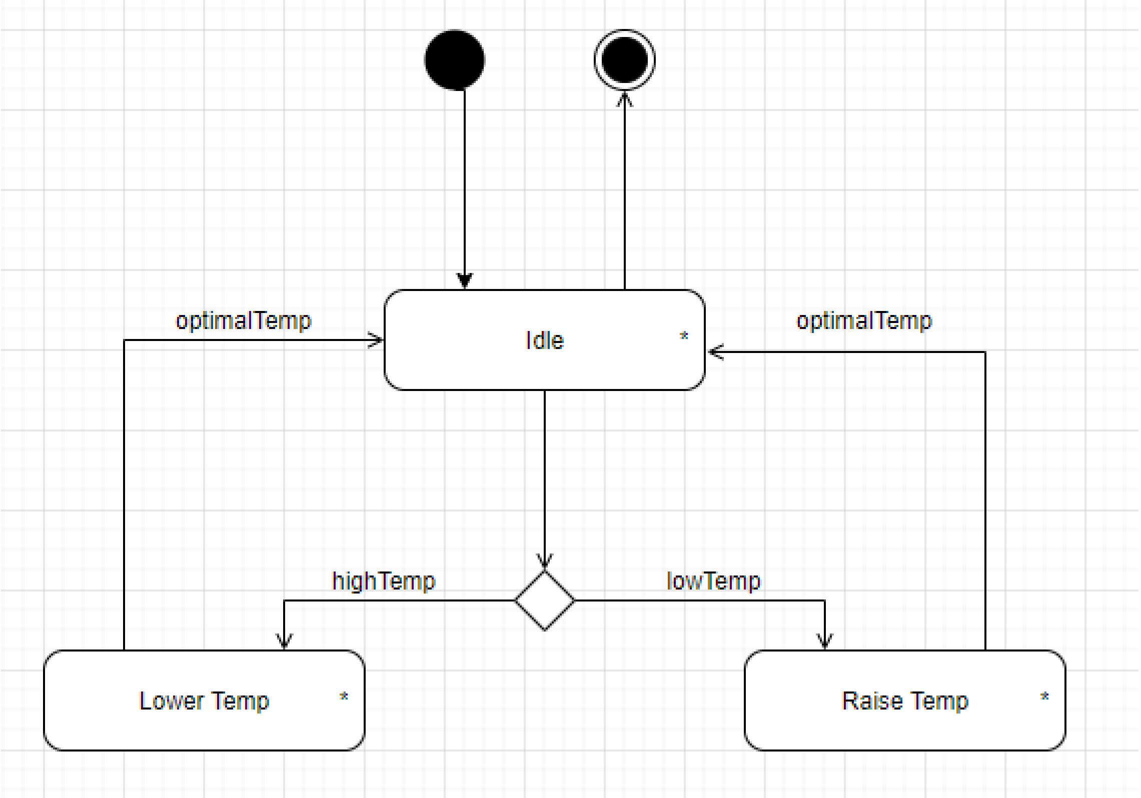


Figure 23 - State Diagram Temperatur Control

D18

State diagram for the curtains control (Figure 24).

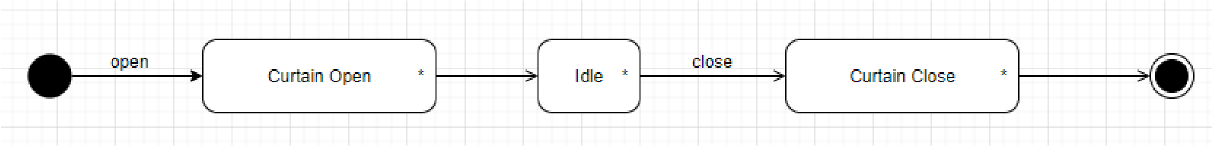


Figure 24 - State Diagram Curtains

D19

State diagram for the light control (Figure 25).

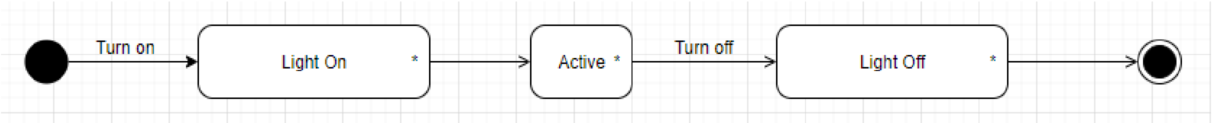


Figure 25 - State Diagram Lights

D20.

State diagram for the window control (Figure 26).

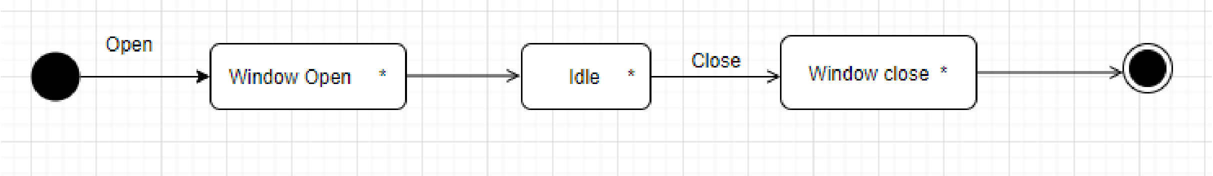


Figure 26 - State Diagram Window