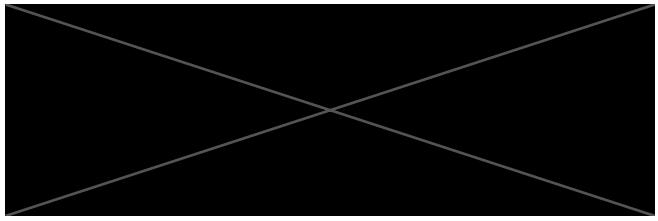

Project Name

MyBMS

Real-Time Building Management System

Author

[redacted]

Date

20/04/2022

BSc Computin Pro ect Report. [redacted]

Table of Contents

Abstract

This project report addresses how overtime control networks in building management systems have evolved and what the future holds for this kind of control network as the Internet of Things devices and low-cost data transfer using Internet protocols and super scalable and fast Cloud technologies that create new possibilities and methods for the development of these traditionally local-area network systems.

The project proposes an unusual approach to control networks for Internet of Things (IoT) devices in Building Management Systems using a real-time database as the main means of communication between a frontend web application and IoT devices. We evaluate implementations of building management systems and control networks from the past, recognize their flaws and suggest a different way of creating embedded control systems with cloud technologies.

An experimental system prototype has been developed and tested that integrates intelligent devices with a real-time database and a web application integrated with the same real-time database where the database stores information about devices' states that can be altered with the web application's interface that then impacts the internal state of prototyped intelligent devices.

The results of the prototyped system in terms of performance, security and stability were satisfying and the system accurately simulates a real-world environment of an air handling plant in a hypothetical building.

Chapter 1. Introduction

Building Management Systems (BMS) are installed in almost all large or small offices buildings, hospitals, airports, datacentres, and other commercial buildings. They help on-site engineers make them safe, comfortable, efficient, and functional by allowing authorised users to control, monitor and automate devices that are part of the building's infrastructure. Building control systems are becoming more and more complex and can include heating, ventilation and air conditioning controls, fire safety, meter readings, building access control, closed-circuit television (CCTV) are somewhat trivial examples, but with the ever-expanding Internet access, low-cost data transfer and Internet of Things (IoT) technologies we're able to connect any piece of infrastructure in a building that can somehow be measured or controlled to the Internet and interact with it using a web browser. An interesting example of a futuristic use case is automatic blinds that use on-roof sensors which track sunlight's position and adjust the angle of the blinds according to that position around a skyscraper size building. We can connect devices like this to the Internet and remotely control them using Internet protocols and a range of software development techniques.

This project's aim was to design and develop a prototype of a Cloud-based Building Management System (BMS) using modern technologies in the field of Cloud computing like serverless backend and Internet of Things (IoT) devices but also frontend frameworks that utilise component rendering and asynchronous HyperText Transfer Protocol (HTTP) requests on popular web browser engines to provide real-time view and control to authenticated users. To create this system prototype it was necessary to pick one of the realistic and potential parts of the infrastructure that can highlight cloud technology in control networks. For this purpose, an *air handling plant* and a scenario of a hypothetical building was chosen to simulate real-world application of an embedded system.

Prototype Scenario – Air Handling Plant in an Imaginary Building
Air Handling Plant consists of three main parts – *Air Handling Units (AHU)*, *Chillers* and *Pumps*. Each part can have as many units as required and their types can be *Primary* and *Secondary.* An air handling unit is responsible for circulating the air around the building by extracting air and resupplying fresh air reducing CO2 levels inside the building. It is also responsible for cooling the air, humidifying if the air is too dry, and dehumidifying if there's too much moisture in the air. A Chiller is responsible for cooling water in a closed system that allows an AHU to drop the air temperature that it is supplying using the water as a coolant. Pumps are very simple electric motors that circulate coolant around a closed circuit. The primary unit also called the *leading* unit is the main worker meaning if a building has two Air Handling Units, one being a primary and the other being secondary it means that the primary unit in automatic mode will work harder when the building's occupancy is low, and conditions are fair making the secondary unit work slower or be in a standby mode altogether.

The aims and objectives of the project:

- Analyse system requirements of a Building Management System (BMS).

- Deliver a working prototype of cloud-based BMS with scalability in mind making the system adaptable and expandable to different parts of building controls beyond this project.

- Showcase and gain experience working with the latest technologies in cloud computing, real-time database management systems, & asynchronous front-end frameworks.

- Create digital prototypes of physical hardware that could be installed in a building to simulate sensor data and actuator state in a form of digital signals.
  For that, the system focuses on an air handling plant with two air handling units, two chillers, and two pumps There's a primary and secondary device in each type of device.

Chapter 2. Literature Review and Context

For over 40 years, control networks have been playing a crucial part in Building Management Systems (BMS). The greatest advantage of integrating building controls using local, regional, or worldwide networks is that they allow for interoperability between devices associated with an “intelligent building”, such as sensors, actuators, and different controllers [1]. Early versions of BMS used proprietary networks and protocols [2] which posed many problems because devices from different manufacturers had differences in their protocols which required developers to build custom gateways to integrate these devices. Fig 1. Shows an example of proprietary networks communicating with each other via gateways. Interoperability, finding a way to link different control systems developed by different manufacturers have been among the biggest challenges when designing BMS communications as these types of systems and Internet technologies gained maturity and further improvement of these systems has led to significant changes in how BMS control networks are designed.

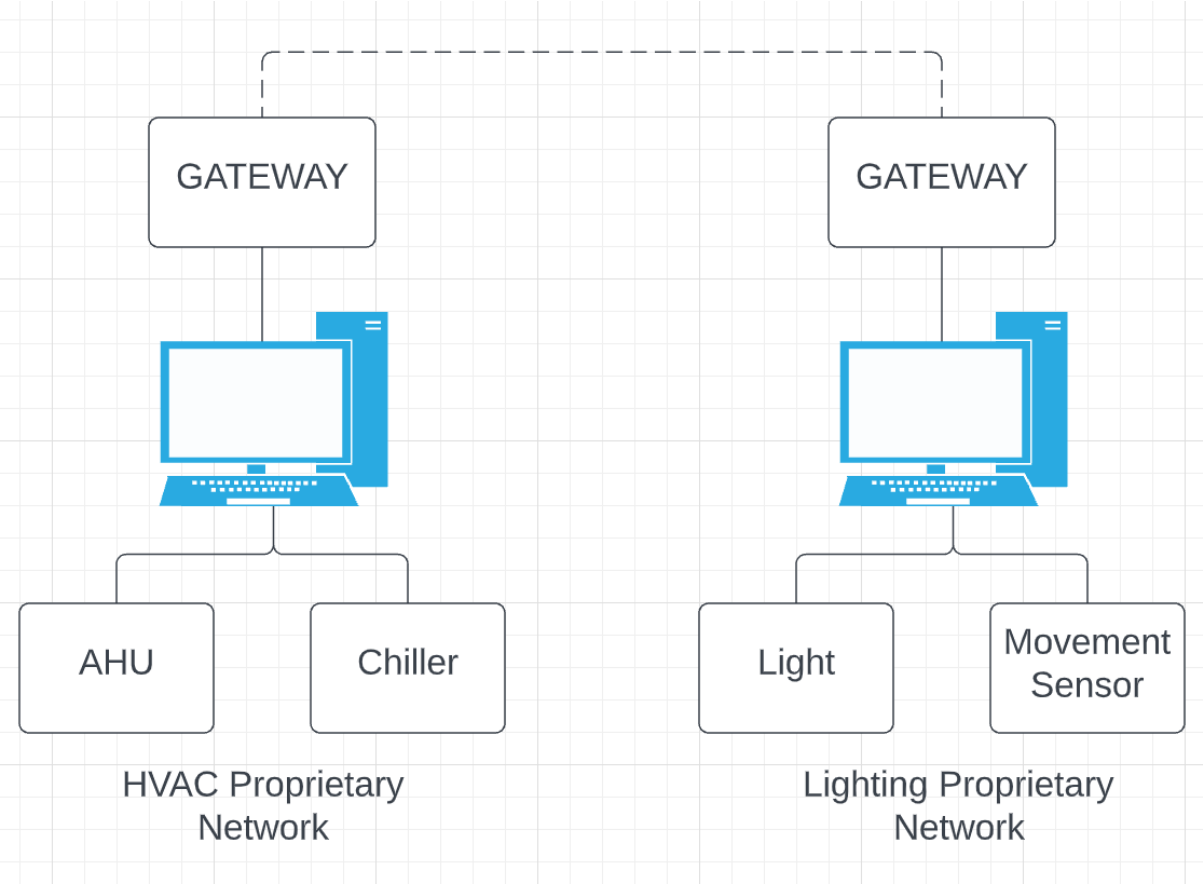

Fig1. Gateway is used to integrate proprietary networks.

To improve interoperability between control networks, manufacturers must use standard protocols which made integration of these networks easier. The first standard protocols for standardizing building-automation industry were called BACnet [3] and LonWorks [4].

Networks based on these protocols, so called “open networks” emerged and their simplified architecture is illustrated in Fig 2. Users can access the network with a local human machine interface (HMI) or remotely with a telephone connection.

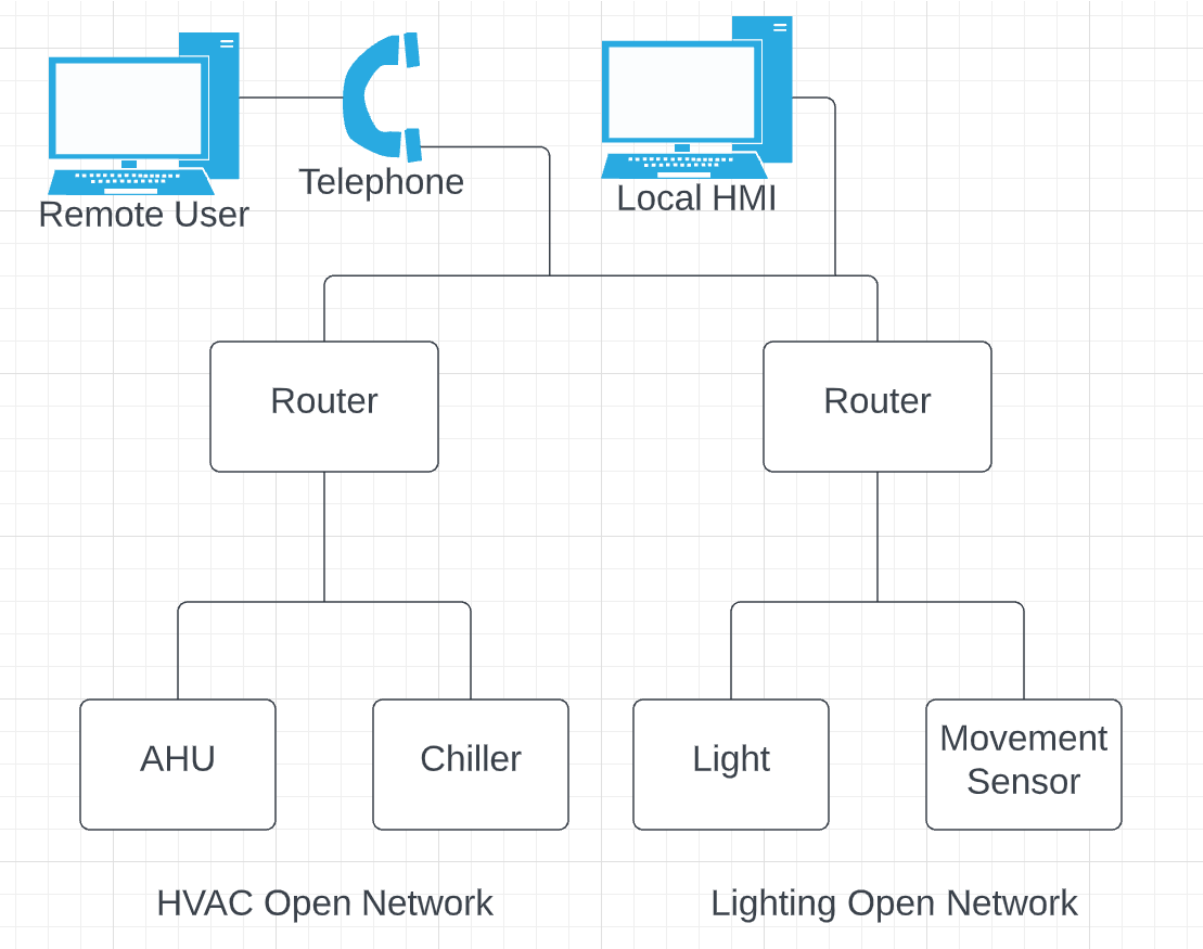

Fig 2. Open Networks Architecture

The next step that was taken in improving open networks architecture was to connect it to the Internet. There is no difference between Intranet and Internet networks as technology is concerned therefore a control network can be integrated with other computer networks using IP routing as demonstrated in Fig 3.

Due to Internet being a low cost means of data transfer between any internet connected locations around the world, it gives organisations an opportunity to improve management efficiency. Managers (privileged users) can give access to different levels of controls for other system users in the organisation also large amounts of data can be accessed from anywhere in the world with minimal delay by simultaneous users, allowing for remote maintenance and control of building systems.

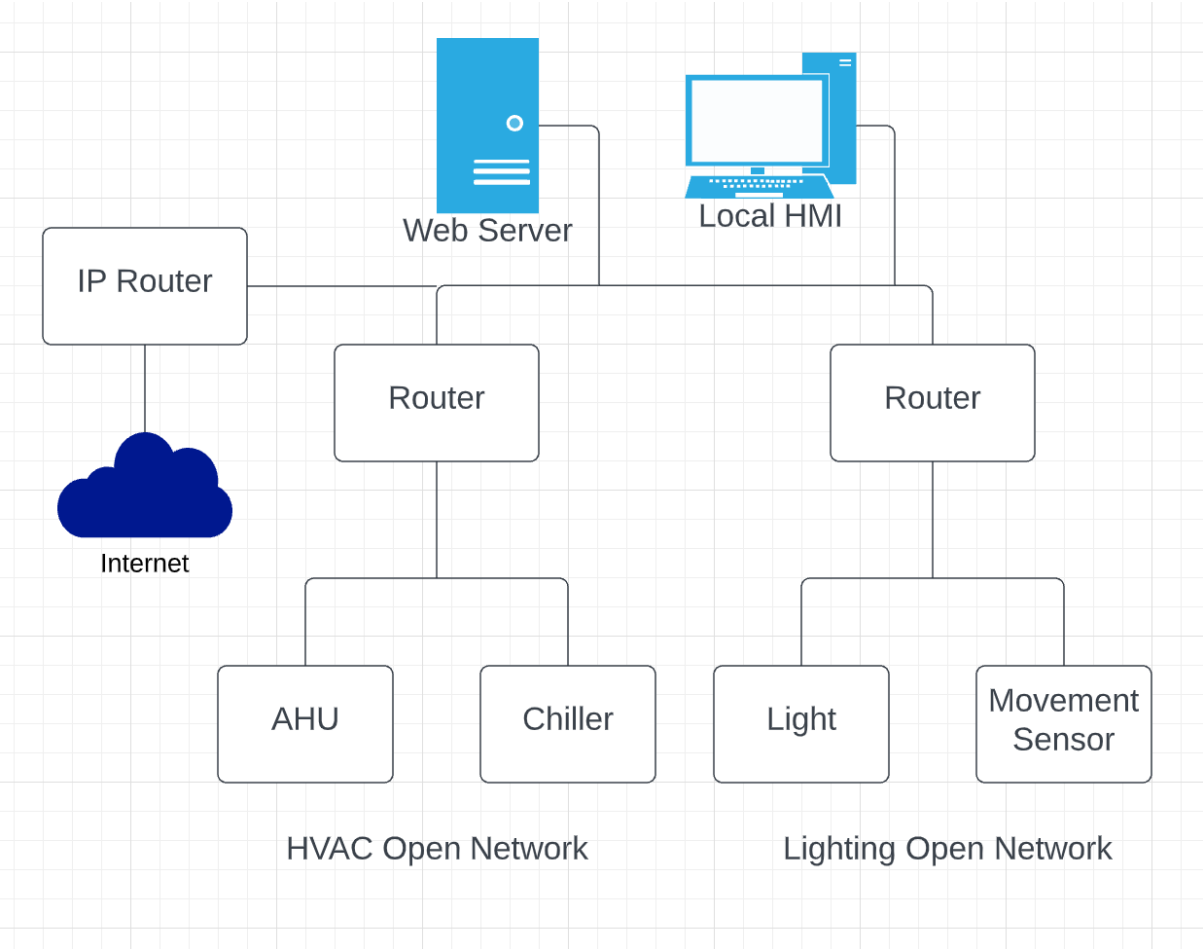

Fig 3. Internet Connected Open Networks

Previously mentioned industry standard protocols like BACnet have adapted to the internet connected open networks with the release of BACnet/IP which transmits packets over the Internet and LonWorks/IP provides similar integration possibilities with a three-layer routing architecture [1]. BACnet/IP defines a collection of 23 standard objects that manufacturers of physical devices must adhere to in avoidance of mismatched data types and data structures, and this allows BMS developers to create and link virtual infrastructures of physical components. In this prototype I have created my own, experimental infrastructure that employs a similar approach and defines datatypes similar to BACnet objects.

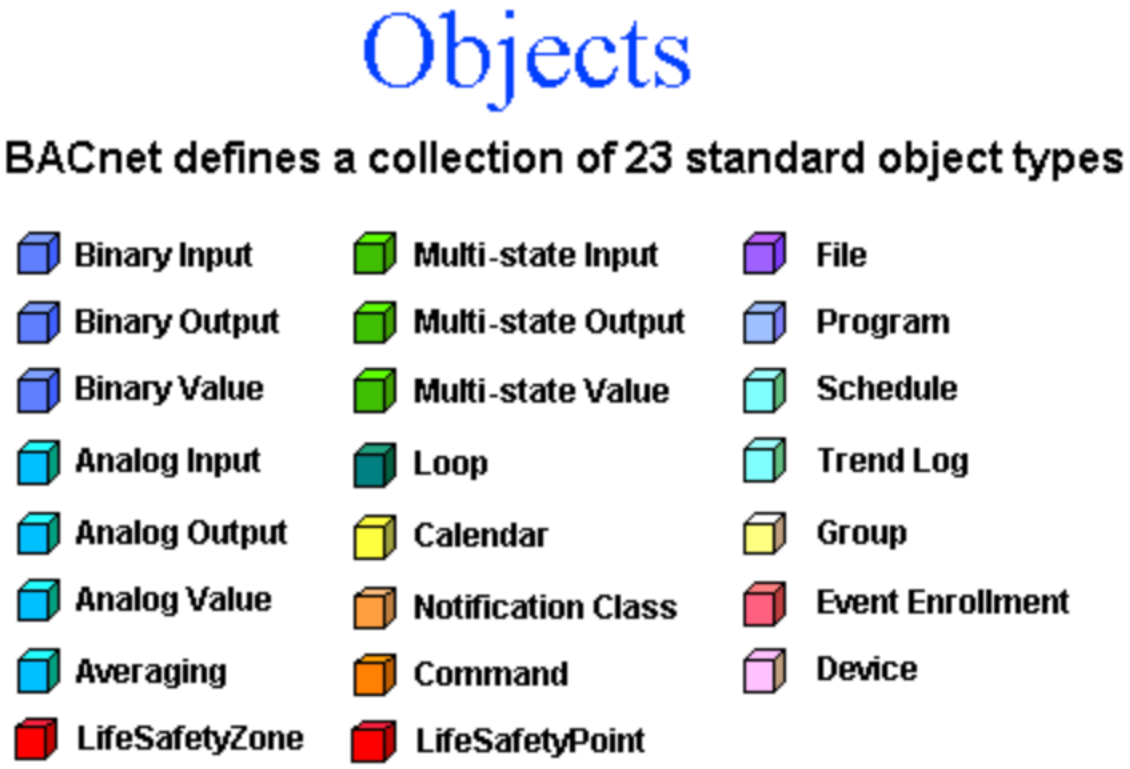

Fig 4. BACnet standard object types (Phil Zito, 2017) [i1]

When Shewei Wang and Junlong Xie studied BMS integration between different types of management systems in 2002 [1] such as Heating/Ventilation and Air Conditioning (HVAC) combined with access control to the building, fire systems, lighting etc… They concluded that there was no common method for distributing information among BMS and other facilities management networks and accessing their databases that was convenient to use and develop at that time. Control networks of BMS like BACnet and LonWorks use local area network (LAN) which utilizes the Transmission Control Protocol/Internet Protocol (TCP/IP) and HyperText Transfer Protocol (HTTP), to integrate a local area control network with the Internet, an interface needed to be developed between these protocols to help with that a number of different technologies were used such as Component Object Model (COM) or Distributed Component Object Model (DCOM), Extensible Markup Language (XML), Simple Object Access Protocol (SOAP) and Object Process Control (OPC). These technologies acted as middleware that helped with translation of code and data in real-time between local networks and the Internet. Release of Active Server Pages (ASP) made it possible for the system developers to build dynamic web functionality using server-side scripts. ASP also allowed developers to ingrain server components into HTML pages that turned business logic scripts into reusable COM components for a web application. XML allowed devices to exchange data efficiently. Usually, a building is constructed with an integrated system that provides lots of control modules and monitoring devices. This traditional building integrated system cannot provide proper services when the building is repurposed. [6]

Fast forward to 2022 and the quickly emerging Internet of Things (IoT) technologies are redefining the way embedded systems can be designed and developed. Along with an increasing number of sustainability regulations, government pressure and green energy movement, IoT is pushing BMS business architecture to evolve and achieve its goals.
A building IoT system has the capability of integrating all resources including sensors and physical objects as well as people and their surroundings. Modern buildings use numerous automatic control systems. Governments particularly in China, are encouraging the use of control systems and energy consumption monitoring platforms in public and commercial buildings. All functions and building management services need to be modularized in cloud platform based IoT control systems. [6]

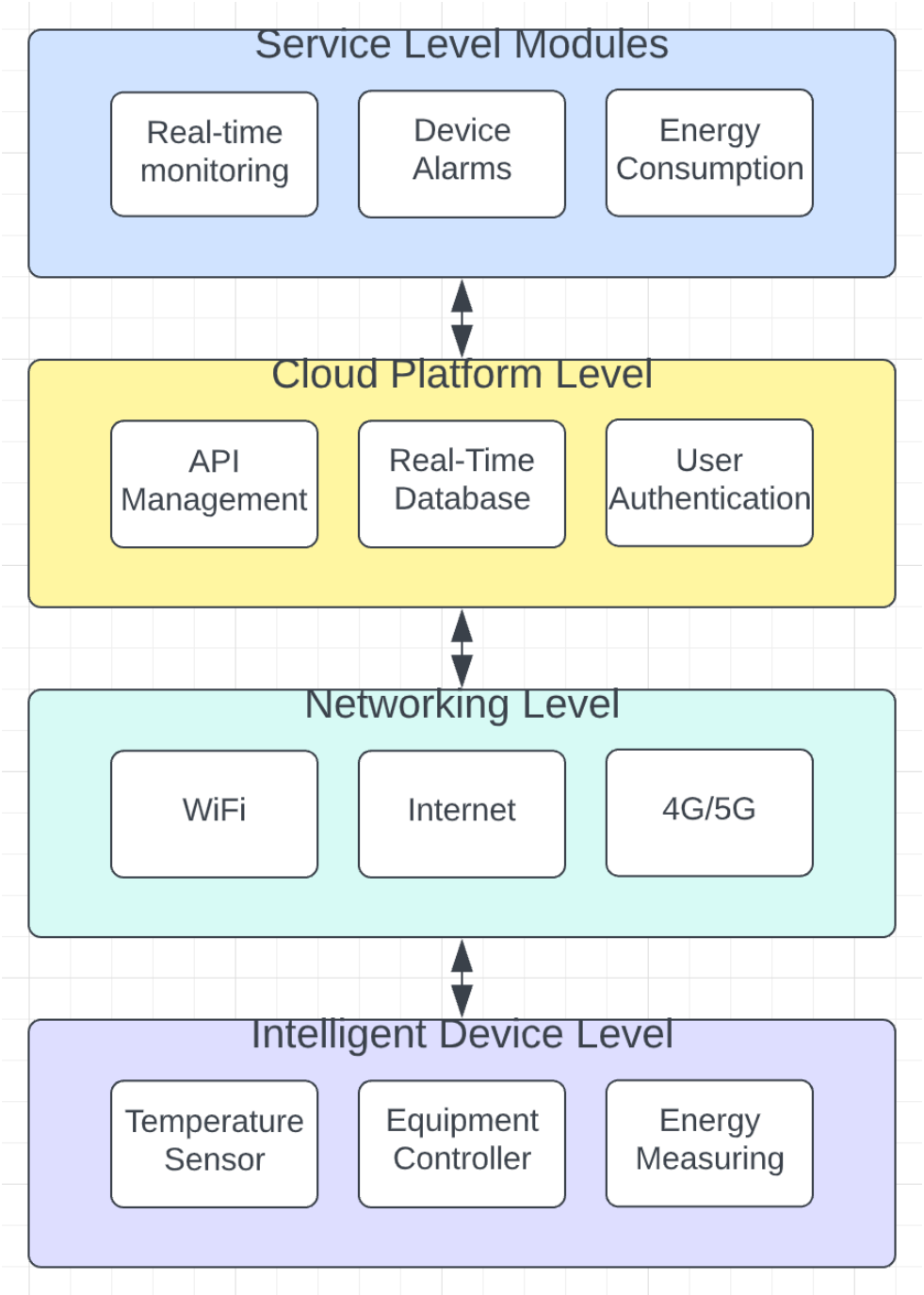

Fig 5. Building Management System on a cloud platform.

There are four levels of building management systems on a cloud platform as illustrated in Fig 5. Including intelligent device level, networking level, cloud platform level and service level. The first and bottom level is the intelligent device (IoT device) which could be an environmental node, equipment control node, energy consumption measuring node or network relay. These nodes cooperate with a network module to allow interaction with the cloud platform.

Creating IoT based complex networking for a cloud system of BMS, a series of intelligent devices need to be built. These devices can have custom chipsets or utilise some of the existing chipsets as core controllers like Arduino ESP8266 or a Raspberry Pi Pico/3B/4B. An example of an Arduino ESP8266 is demonstrated in Fig 6.

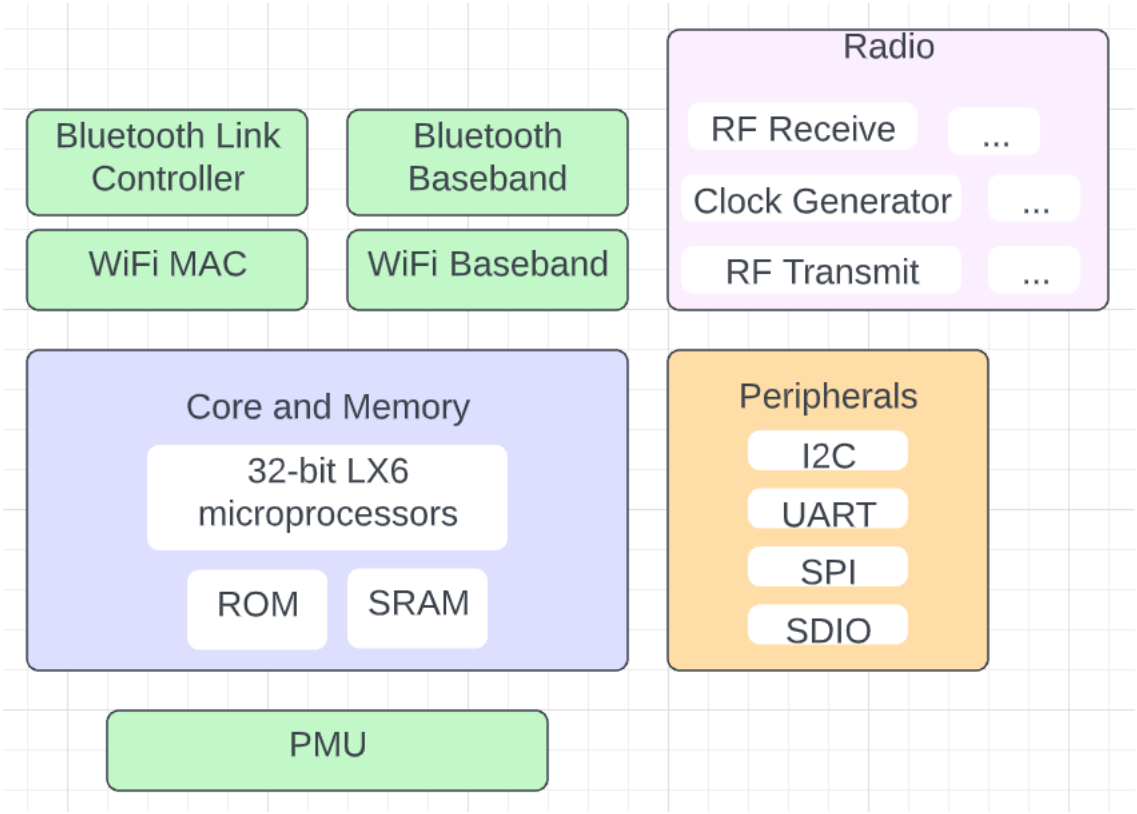

Fig 6. Arduino ESP8266 Controller Diagram

The ESP8266 can function as a standalone application or as a slave to a host microcontroller due to its complete and self-contained Wi-Fi networking capabilities [7]. When the controller is used to host the application, it starts up right away from the flash. The embedded high-speed cache aids in improving system performance and memory optimization. It can be used as a Wi-Fi adaptor in any microcontroller architecture with SPI/SDIO or I2C/UART interfaces. ESP8266 comes with antenna switches, RF power amplifiers, low noise receive amplifiers, filters and power management modules [8], making it ideal for constructing intelligent nodes.

The BMS cloud platform implements functions that allow data exchange through the Internet between intelligent nodes, a web server and a database making data accessible to the end-users through views that invoke API interfaces and Create, Read, Update, and Delete (CRUD) operations. The common way of integrating software cloud platforms with intelligent devices is demonstrated in Fig 7. Where the intelligent device communicates with the webserver which then sends and receives data from the Cloud database attached to cloud services associated with the views. This integration can further be simplified where the intelligent device is a standalone application that communicates with a serverless backend cloud platform directly as demonstrated in Fig 8.

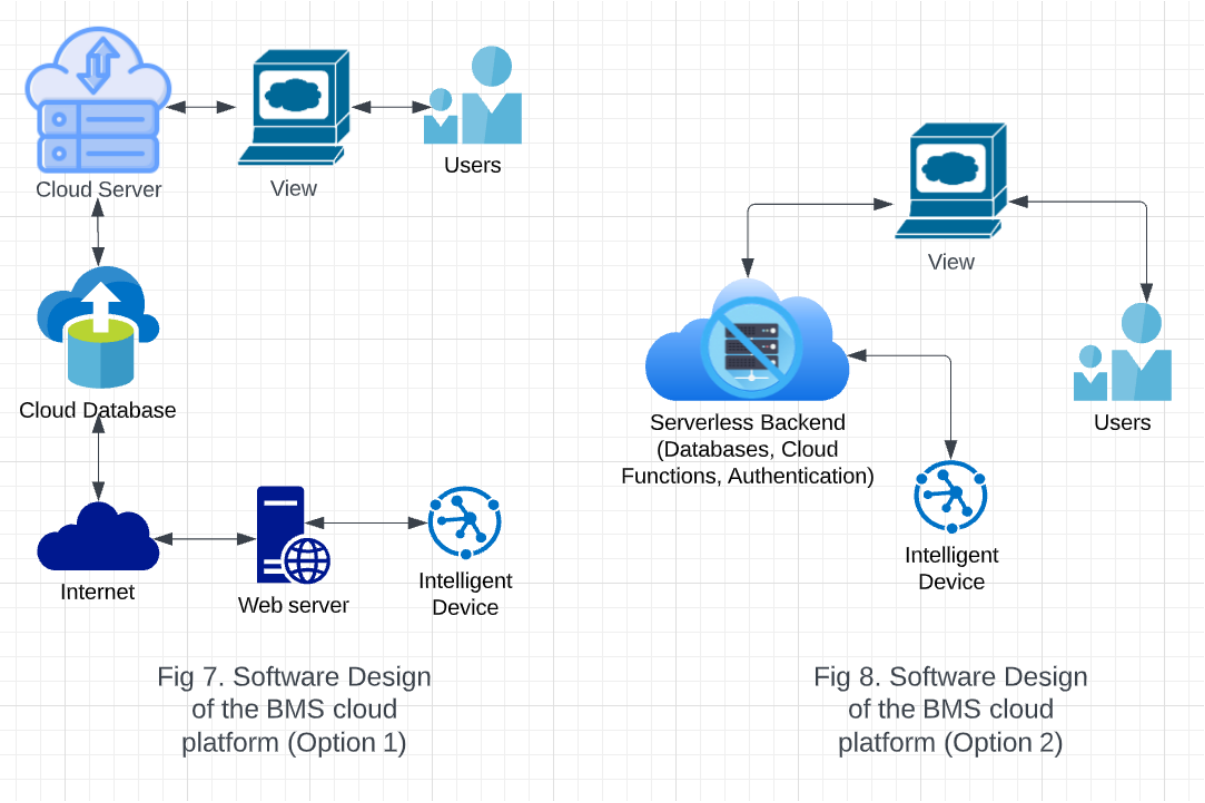

Fig 7, and Fig 8. Software Design of the BMS cloud platform.

Backend as a Service (BaaS) like Firebase provides a real-time database in JavaScript Object Notation (JSON) format that automatically stays in sync with a frontend application meaning that any changes in the real-time database are effectively viewable in real-time on the frontend application [9]. Ever since Firebase was acquired by Google in 2014 it became a suite for making applications on top of the Google Cloud Platform. Firebase was further developed by Google and now also provides user authentication, hosting, and serverless computing using cloud functions and cloud messaging. Firebase provides Software Development Kits (SDK) that support a number of different programming languages and technologies that allow developers to build frontend applications on top of the serverless backend services provided by Firebase that easily integrate with intelligent devices.

Chapter 3. Development Method

After a thorough research into how control systems like BMS can be made using current technologies, languages, and frameworks available in 2022. I decided to create *MyBMS* using serverless Backend as a Service (BaaS) - Firebase, intelligent devices as standalone applications which are prototyped using Python programming language to imitate real world devices on a BMS network, and view created with React library for JavaScript programming language that allows control and monitoring of the device prototypes. This is the most suitable way for development of a building management system prototype as a sole developer with cost and time effectiveness in mind. Because on the market there isn't anything else like Firebase that supports IoT applications and provides powerful tools in terms of data communications, analysis, and access [10], it became clear that this is the technology to go with.

Requirements for Building Management System Prototype

The first thing to consider was to decide on how the user is going to interact with the intelligent devices and how are the intelligent devices going to communicate with each other to perform their correct function. For this BMS prototype, we assume that the hypothetical building requires two air handling units that condition and circulate the air in and out of the building, two chillers that make sure the coolant used in air conditioning is of a low temperature and two pumps that circulate the coolant which is just water in a closed system with added chemicals that ensure its quality against corrosion and wear of the system [11]. These devices need to be communicating with their relevant counterparts and other intelligent devices that are responsible for providing data about the building environment such as temperature setpoint, current indoor temperature, outside temperature, indoor and outdoor humidity, indoor CO2 level and current occupancy. To do this the system requires means of simulating values that in the real world would be provided by sensors attached to intelligent device/s. This data is necessary to ensure the correct function of the HVAC system regarding high oxygen quality, desired temperature, and optimal humidity levels [12].

Instead of using Arduino or Raspberry Pi as controllers, Python programs are used to simulate hardware and intelligent devices. Python classes that correspond to a unique intelligent device are connected directly to the real-time database in Firebase using its SDK and every 2 seconds query the database for changes. If changes are made and if these changes should affect the behaviour or state of the device, then its internal state changes and information about the new state are also updated in the real-time database. The database is the main means of communication for all controllers on the *MyBMS* network meaning controllers aren't directly connected to one another, they only see the state of each other as it is in the real-time database, therefore it is necessary that the data in the database is always synced with the real state of the simulated device. If the user is viewing data in real-time and the device disconnects and stops updating the database, then the user will see the old information about its state in their view, therefore when an intelligent device disconnects the system must alarm the user that this event has taken place and that

the currently visible data is the last known, and not necessarily the current. The advantage of creating the system architecture this way allows to take full advantage of the real-time database and eliminate the need for connecting the intelligent device with each other using Internet protocol sockets which greatly simplifies the network architecture for an IoT building management system as demonstrated in Fig 9.

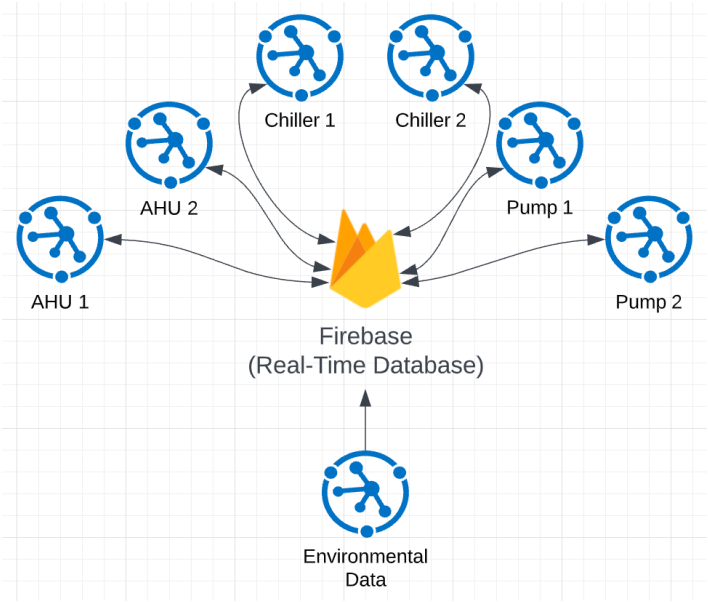

Fig 9. Intelligent devices communicating through a real-time database

Each device on the *MyBMS* network handles its own logic based on the mode they are set to. A device can be in ON/OFF/AUTO mode where mode ON overrides any logic and turns the device on to maximum power like an Air Handling Unit and allows the user to override motors and actuators inside the unit. OFF turns off all the motors and closes actuators and AUTO mode makes the device consider the current state of other associated intelligent devices and environmental data to deliver optimal settings for itself to achieve environmental goals in an energy-efficient and sustainable way.

Environmental data is manually entered into a data simulation form on the web application of the BMS. Users can specify different weather and atmospheric conditions inside and outside of the building and intelligent devices set to an AUTO mode read these values and respond to them in a manner that allows connected devices to achieve indoor temperature setpoint, optimal indoor humidity level of 40% and low CO2 levels while reusing indoor air as much as possible which helps with lowering energy consumption. The system uses Firebase Authentication and only an authorised user can log in to the system's interface using email and password credentials which are previously set up by an admin of the system in the Firebase authentication console. Because of this, there is no need to develop a signup page or manually assign tokens using something like OAuth which would require us to store user passwords and other user information somewhere else adding unnecessary complexity. Only a login page is required, ensuring the system's robustness against unauthorised access.

Chapter 4. System Design and Architecture

The project is split into three parts: Intelligent devices, Firebase (BaaS) and a Web Application that implements a secure user interface allowing verified users to interact with the database's storage bucket. Values in the database that correspond to the simulated environment or device state are read by intelligent devices which using a built-in logic can alter their own internal state as well as other devices that are necessary to provide correct functionality of a cooling and air handling plant.

Intelligent Devices Design

Raspberry Pi as an Intelligent Device.

The first prototype of an intelligent device that was created for *MyBMS* consisted of a Raspberry Pi as a controller to which a wired temperature sensor was connected via its data bus to Raspberry Pi's General-Purpose Input/Output (GPIO) pin. The temperature probe used was an *AZ-Delivery DS18B20 Temperature Sensor Sonde as illustrated in Fig 10.* Raspberry Pi runs Linux and Python which has special libraries for interacting with the GPIO pins from the Raspberry Pi meaning that the sensor data can be captured in a Python script and assigned to a variable for further processing.

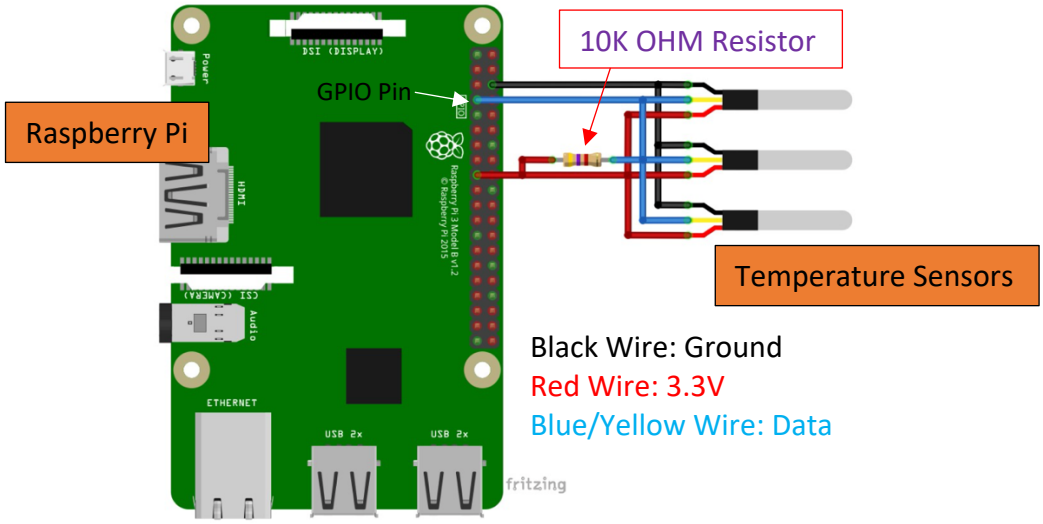

Fig 10. Raspberry Pi with connected DS18B20 sensors.

Az-Delivery DS18B20 Temperature Sonde comes with a boilerplate Python script from the manufacturers that allows for making temperature readings every second or any other interval greater than 0.1 of a second [Appendix A]. Now that the reading is being taken every second and each time, we overwrite the value of the temperature variable we can push it to the real-time database on every iteration of the while loop.

To do that, we first must connect to the database with a Firebase configuration file that points to our real-time database. The configuration file includes an API key, Authentication Domain, Database URL and Storage Bucket these details are unique to each Firebase or

Google Cloud Platform (GCP) project and act as a key when we invoke any of the CRUD APIs available for a real-time database. Now that we're able to read sensor data in real-time and that we're connected to the storage bucket of our database we can invoke different functions like push(), update() or delete() that allow us to control the state and existence of our data in the Cloud using Python library *Pyrebase* that was developed for the purpose of controlling Firebase real-time database.

The problem.
Creating a BMS prototype system using Raspberry Pi or Arduino device would require a substantial amount of physical input/output devices connected to the limited number of GPIO pins on the board as well as the ranges of input would be hard to simulate without real-world building infrastructure where values are realistic. Also testing the system against different value ranges wouldn't be possible using real sensors without having to recreate real weather and temperature conditions to test the devices' correct behaviour.

The solution.
When we abstract ourselves from the physical components that we want to interact with, it is easy to realise that the physical components like an air handling unit, chiller, pump, or temperature sensor can be represented as a digital device in form of a class that consists of properties and methods that describe its state and behaviour. This approach allows for creating objects based on these classes like AHU-1, and AHU-2 to simulate any sort of building environment. We can use client-side triggers that will update values inside the database and then inside the objects of classes that prototype physical components to imitate the state of real machinery. We can also simulate real-time temperature values using a random number generating code within desired ranges to simulate real-world temperature for a given scenario. We can then update these values in equal time intervals in the database if the values are different from the ones currently there.

List of intelligent devices simulators required for imaginary Building 1, Air Handling Plant.

- *Air Handling Unit 1 (AHU-1, Primary)*
- *Air Handling Unit 2 (AHU-2, Secondary)*
- *Chiller-1 (Primary)*
- *Chiller-2 (Secondary)*
- *Pump-1 (Primary)*
- *Pump-2 (Secondary)*
- *Environmental Sensors Simulator*

Air Handling Units Controller Simulator Design

As mentioned in the introduction, there can be two types of air handling units in a BMS, there are primary and secondary types. Primary AHU is also known as a Leading unit meaning that when the automatic mode is active this unit reacts to environmental data without a dependency on another AHU. The secondary unit works differently, it is dependent on environmental data and the state of a primary AHU that dictates its function. Both Primary and Secondary AHU share the same properties and only slightly differ in their behaviour. Each AHU has two parts – Extract and Supply and as the names suggest extract is responsible for extracting the air from inside the building which is then passed through filters and depending on whether a damper separating Supply and Extract is open or closed the air is pulled back

into the building if a damper is open allowing that air to re-circulate which is dependent on indoor CO2 level and if it's closed then the air is pulled out of the building. There are two fans, one in the Supply section of an AHU and one in the Extract. They can work at different speeds that can be set from 0 to 100%. When the Supply pulls air from the outside it is passed through air filters and depending on the outside humidity unit can either dry the air using a Thermal Wheel that aids dehumidification of the supplied air or add moisture to the air with a humidifier that works like a sprinkler which creates a water mist that is mixed with supplied air to add humidity if the supplied air is too dry. Air handling units use coolant supplied by the cooling plant (Chillers and Pumps) to lower supplied air temperature if the temperature setpoint is lower than the indoor or outdoor temperature. If the air doesn't need to be cooled, then the coolant bypass actuator is closed. All the properties of an air handling unit and its function are visually illustrated in Fig 11.

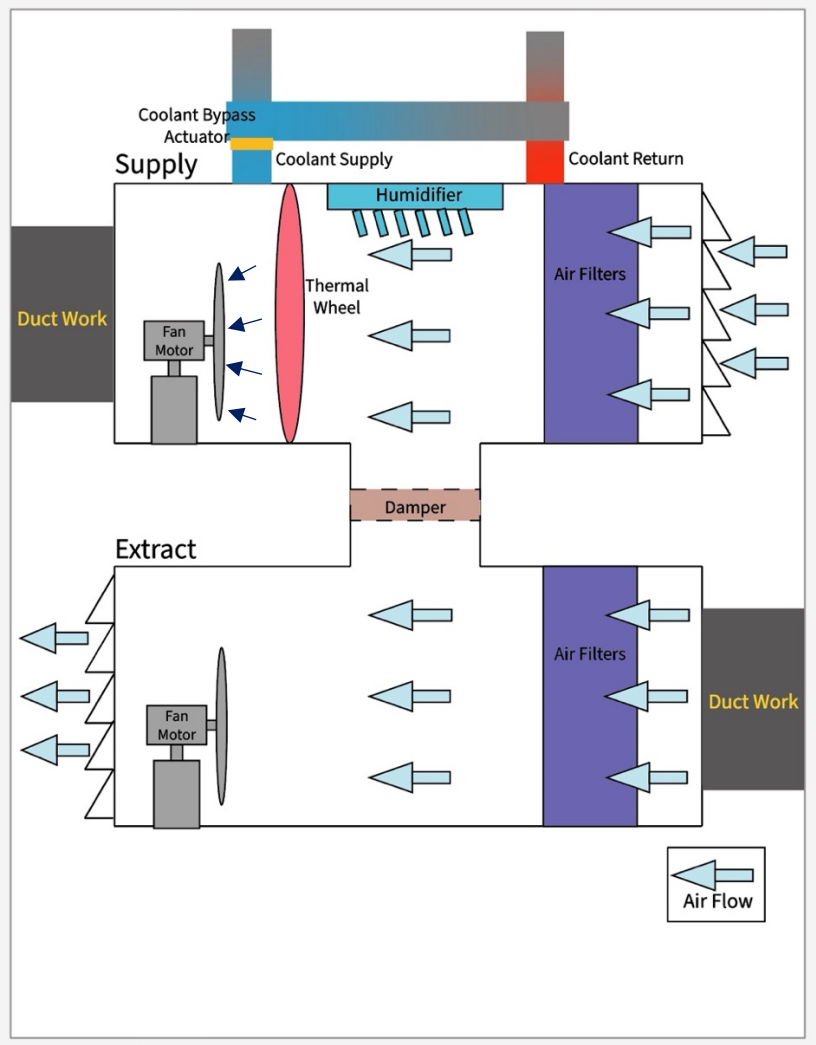

Fig 11. Air Handling Unit, Supply and Extract

Both *primary* and *secondary* AHU have the same attributes and can be represented as respective objects of an AHU class where its class is a model of an AHU controller device with built-in state and logic. Fig 12.

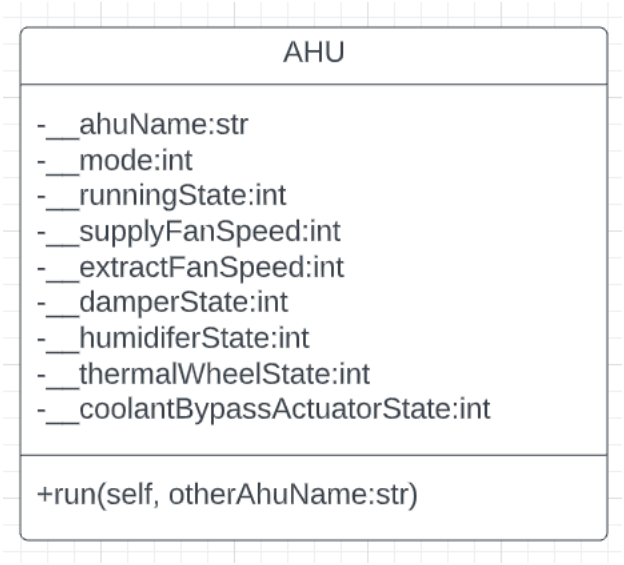

Fig 12. Class Diagram of an Air Handling Unit.

Chillers and Pumps Controllers Simulator Design

For every chiller, there is one pump that circulates coolant around the closed-circuit system. There can be two types of chillers primary and secondary where the primary chiller is the main worker on the cooling system and the only time a secondary chiller works along a primary is when the coolant's temperature exceeds a specified threshold, in our case, it's if the flow temperature is above six degrees Celsius or the return temperature is above eleven degrees Celsius. Their multiplicities, attributes and methods are shown in Fig 13.

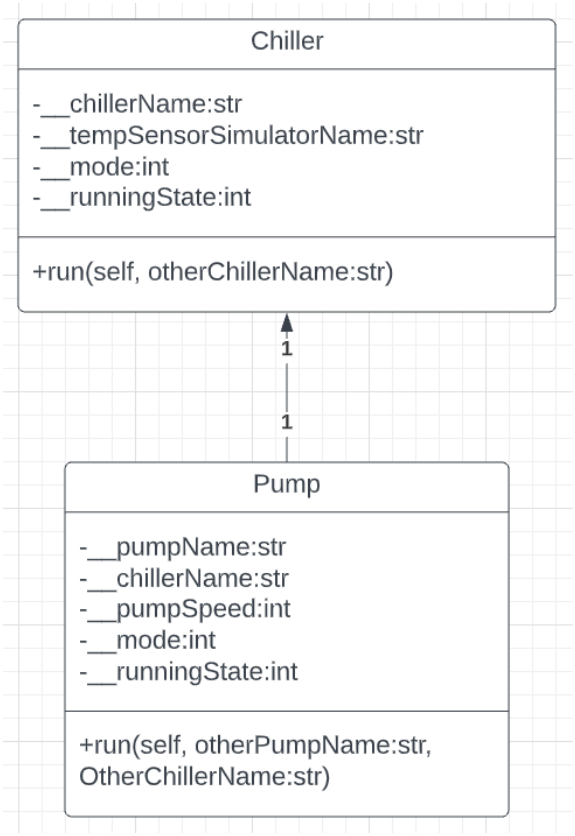

Fig 13. Chiller and Pumps Class and their association.

A simple diagram illustrating what a pump and a chiller look like is shown in Fig 14. Controller classes of Chiller and Pump have a run method that handles its logic according to the current state of other devices associated with it.

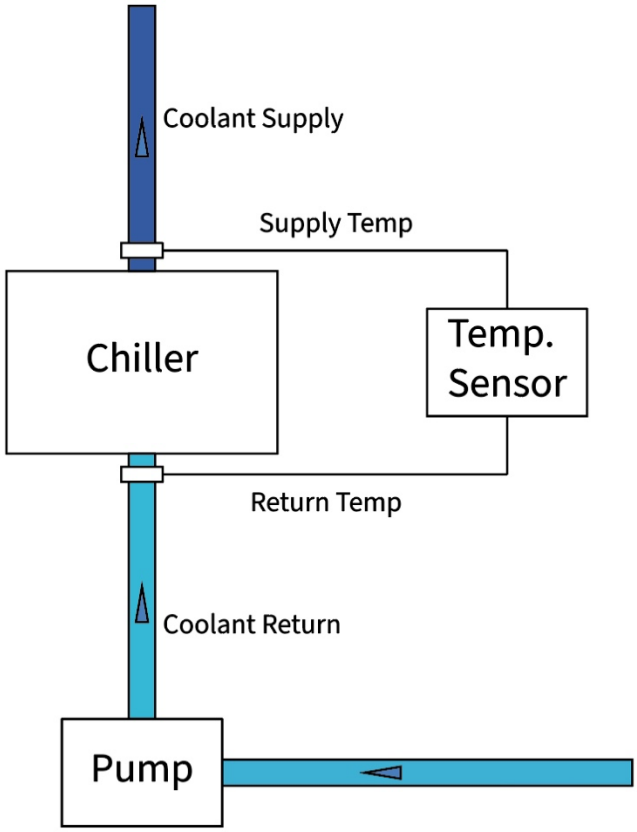

Fig 14. Chiller, Pump and Temperature Sensor.

Intelligent Devices Integration

Each class is a standalone application with in-direct dependencies on its primary and secondary units as well as in a case of a chiller also a pump and a temperature sensor, by in-direct meaning that their respective values in the database are taken into consideration by intelligent devices instead of a direct connection with the associated device. It is possible for the device objects to be created in separate environments and even have their own simulation scripts that call each one by one because of the database communication architecture but for the simplicity of running intelligent devices for MyBMS, a simulator script was created that initializes all the objects and runs them in continuously in specified time intervals. The full class diagram is illustrated in Fig 15.

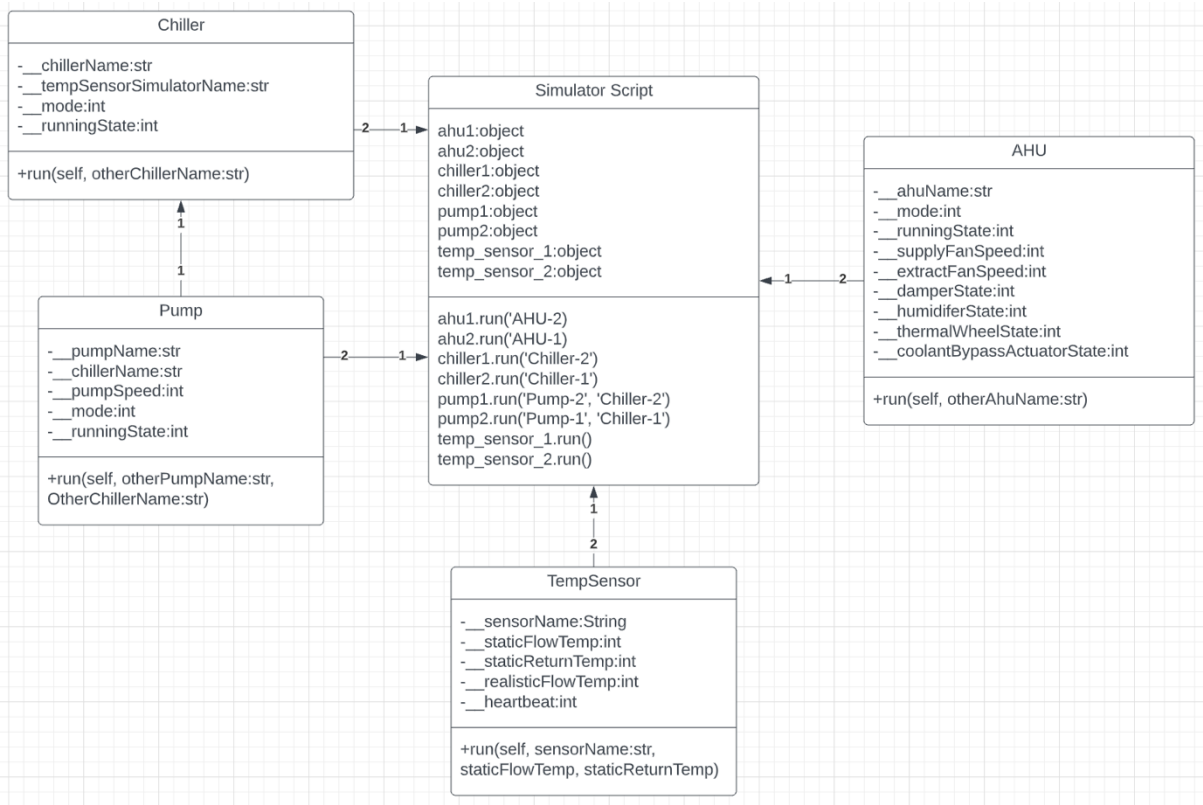

Fig 15. Class Diagram of MyBMS IoT Device Simulator

Firebase (Auth and Real-Time Database)

Database

JSON objects are used to store all data in the Firebase Realtime Database. Consider the database to be a cloud-hosted JSON tree. There are no tables or records in this database, unlike a SQL database. When you add data to the JSON tree, it becomes a node with an associated key in the existing JSON structure. You can either offer your own keys, such as user IDs or semantic names or have them provided via the push() function. Firebase also specifies several other functions that come with its SDK for interacting with a storage bucket.

Authentication

MyBMS takes advantage of Firebase Authentication SDK to authenticate users using their email and password credentials. Firebase backend services take care of data verification and send a response back to the client. After successful verification, a logged-in user can read and write data to the database this method of user authentication provides a high level of system robustness encrypting all messaging, automatically distributing public keys and obscuring all backend operations from the user. Firebase provides a console for setting user permissions these permissions can give access to different users to grant access to only services that we want them to access in our case we only have one type of user that has full rights to the system so there is no need of specifying permissions in the Firebase console. User account creation happens directly in the Firebase control panel where an admin or the owner of the database creates users and their passwords. There is no need for a

Signup page which in any case is extremely simple to implement and the code looks almost the same as the login page.

Web Application Design

The web application of MyBMS provides access and interface to data corresponding to the current state of intelligent devices. It uses a Pub/Sub messaging service from the Google Cloud Platform. This service supports an event-driven application design pattern allowing integration with Google's systems like Firebase (Real-Time Database) that exports events to Pub/Sub.

Pub/Sub, let's developers create event procedures and consumers called *publishers* and *subscribers.* Publishers broadcast events (data) to subscribers asynchronously instead of synchronous remote procedure calls (RPCs). Publishers in our case the intelligent devices send events to Pub/Sub service (Firebase) without paying attention to how we process these events. Pub/Sub delivers these events to its subscribers in our case it's the web application components that update their state on value changes in the database.

The application is built using JavaScript programming language with React library which is an open-source front-end technology for building user interfaces using UI components. When Pub/Sub service delivers data, React can render this data to the web page in real-time. *MyBMS* web application is an example of a single page application (SPA) with the ability to manipulate DOM components that translate JavaScript code to HTML tags using JSX which is a syntax extension of JavaScript it is like a template language that comes with the full power of JavaScript, it allows data to be displayed on the web page.

Components of the web application that provide an interface to the devices are organised in hierarchical composition where one component calls on the service of another to create a super component like the Air Handling Plant in MyBMS which consists of AHU, Chillers and Pumps. This composition is displayed in Fig 16.

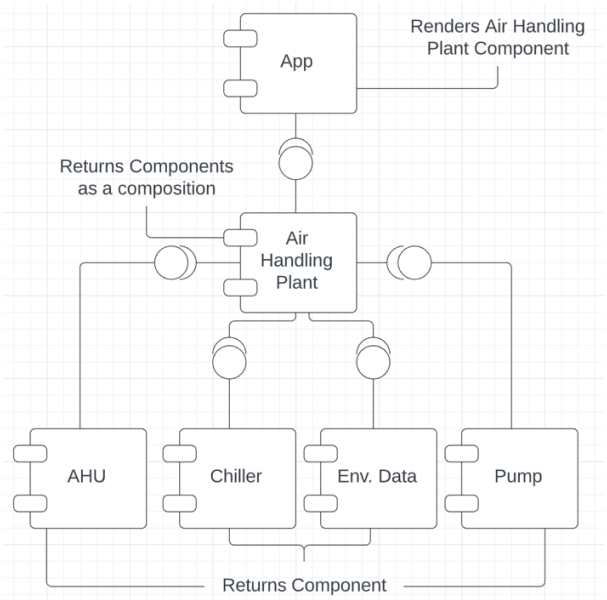

Fig 16. Component Composition of Air Handling Plant

An important component of the web application to consider further is the Environmental Data Simulator, this component lets a logged-in user create real-world building conditions that impact the function of air handling units in their automatic mode. It only exists in the database as JSON object and as an interactive component in the web application but in reality, this data would be provided by various sensors.

Environmental Data Variables include:

- Temperature Setpoint
    - The number between 15 and 30 degrees Celsius, it affects a coolant bypass actuator, if the setpoint is lower than the current indoor temperature, coolant bypass helps the air supplied from the outside to be cooled down to ensure lowering of the air temperature inside the building.

- Indoor Temperature
    - This number value corresponds to the simulated current indoor temperature. Acceptable values are between 0 and 35 Celsius degrees.

- Outdoor Temperature
    - This number value simulates the current outside temperature. Acceptable values are between -50 and 50 Celsius degrees.

- Indoor Humidity
    - The value of indoor humidity affects a thermal wheel (dehumidifier) when the value is greater than 40% and a humidifier if that value is lower. 40% humidity is considered an optimal value for human wellbeing and technology infrastructure. Acceptable values are between 0 and 100 per cent.

- Outdoor Humidity
    - The principle is the same as indoor humidity because supplied air can be conditioned further to speed up the process of humidifying or dehumidifying the indoor atmosphere. Acceptable values are between 0 and 100 per cent.

- Occupancy
    - This metric affects the fan speed of the air handling units which essentially affects the supplied air turnover, if the occupancy levels are low we don't have to condition the air too quickly as CO2 levels will stay lower than in a heavily occupied building which will save energy and lower cost. Acceptable values are between 0 and 100 per cent.

- CO2 Level
    - CO2 Level affects a damper that divides an air handling unit's supply and extract chambers. When the CO2 levels are medium or low the damper state can be open to reuse the indoor air as much as possible to preserve the indoor atmosphere and reduce the energy cost of humidifying/dehumidifying supplied air as well as heating it up.

IoT Devices, Firebase and Web Application Integration

The devices can be written in any language, work on any hardware and if they are connected to the internet anywhere in the world and are capable of reading and writing data in JSON format that corresponds to their state in real-time then they can easily integrate with our frontend services using glue code. A full diagram of the system architecture is shown in Fig 17. The management block in figure 16 corresponds to all of the IoT devices on the network and means that we have to manage their logic there before we send data about their state to Cloud Firestore for storage. Each of the IoT devices imports its own Firebase configuration file which includes an API key and storage bucket information. When the configuration file includes the correct information, the device is then able to read and write data to the real-time database of Firestore. This data is then available to be viewed in the React web application to authenticated users.

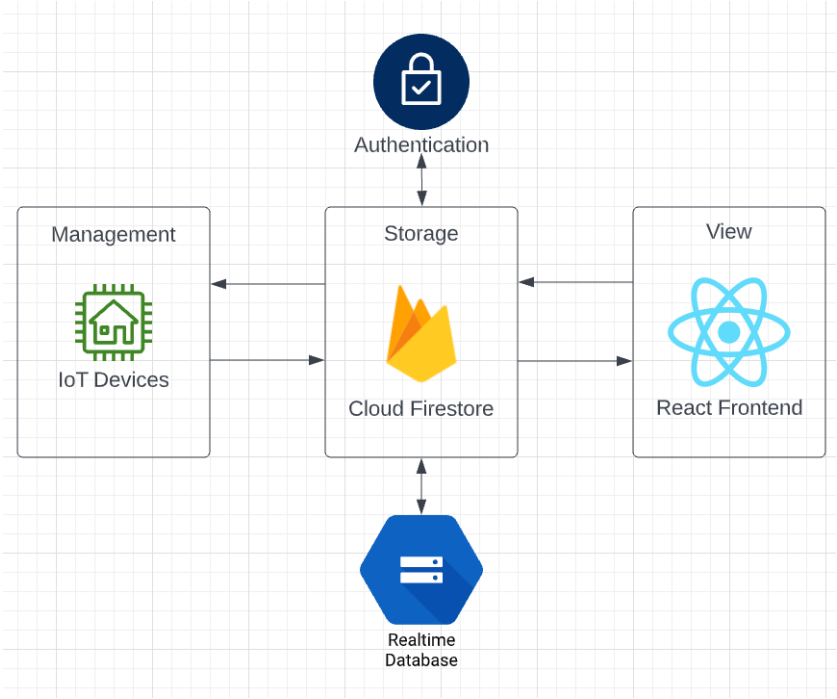

Fig 17. System Integration

Security Rules

Security rules are an important part of Firebase, they allow clients to invoke CRUD operations in the database. Firebase Realtime Database specifies a special location in their console that allows the developer to create these rules for specific types of users and give them access to chosen parts of the system, for example – if the system was expanded to having more than one type of users like a regular user and a super user then the super user could specify a different sets of controls that the regular user is allowed to perform. For this prototype there was no need of specifying these rules as the focus in the this prototype was on the interactions and real-time data processing but I thought it was important to highlight the possibility of setting user permissions in the future.

Chapter 5. System Evaluation and Testing

System in Operation

We now have our intelligent devices created and running independently, querying the database for their respective values and reacting to that data in time intervals specified by the user, at the same time the web application can be running on a different server. When a user logs in with valid credentials, once he/she is authenticated by the Firebase OAuth server they are redirected to the Dashboard page that allows them to choose which building they want to access. Depending on the size of the company using MyBMS they could have access to multiple buildings that they have a contract with for maintenance. Each building could be connected to a completely different Firebase Realtime Database storage bucket and authentication server and include totally different components and intelligent devices. In our prototype, we have only one option, our imaginary Building 1 in Fig 18.

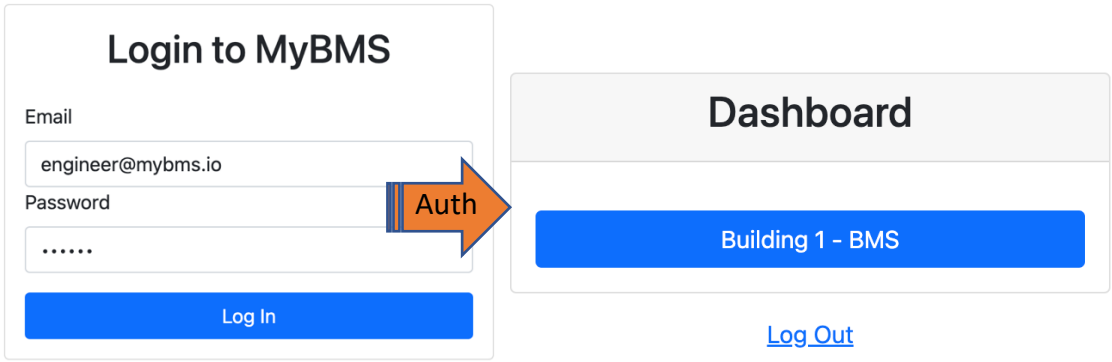

Fig 18. Login component redirecting to the dashboard of MyBMS.

Once the users have made a choice on which building they'd like to access, on click they are redirected to the Building Overview page of the web application, this page could include information from analytics, if we decided to store the real-time data and generate data visualisation components like charts but also any overview data of the components that indicate a part of their current state rather than an entire detail.

In MyBMS Building 1 a decision was made to display the connection status of each intelligent device that is currently running. When the devices are running, within each iteration of their run method they send a timestamp to the database that corresponds to their heartbeat information. The web application reads this heartbeat value when it changes and saves it internally and does that check again on the next iteration to compare the new value of the timestamp with the last value they saved, if the value is different that means a device is sending data about its state, therefore, it is connected. When the value is the same after ten seconds when the next check is being made it will indicate that the heartbeat information is not being updated in the database concluding the device is disconnected/not running. A screenshot in Fig 19 shows a moment simulator.py file is run that includes all the intelligent devices as objects of their classes turning on.

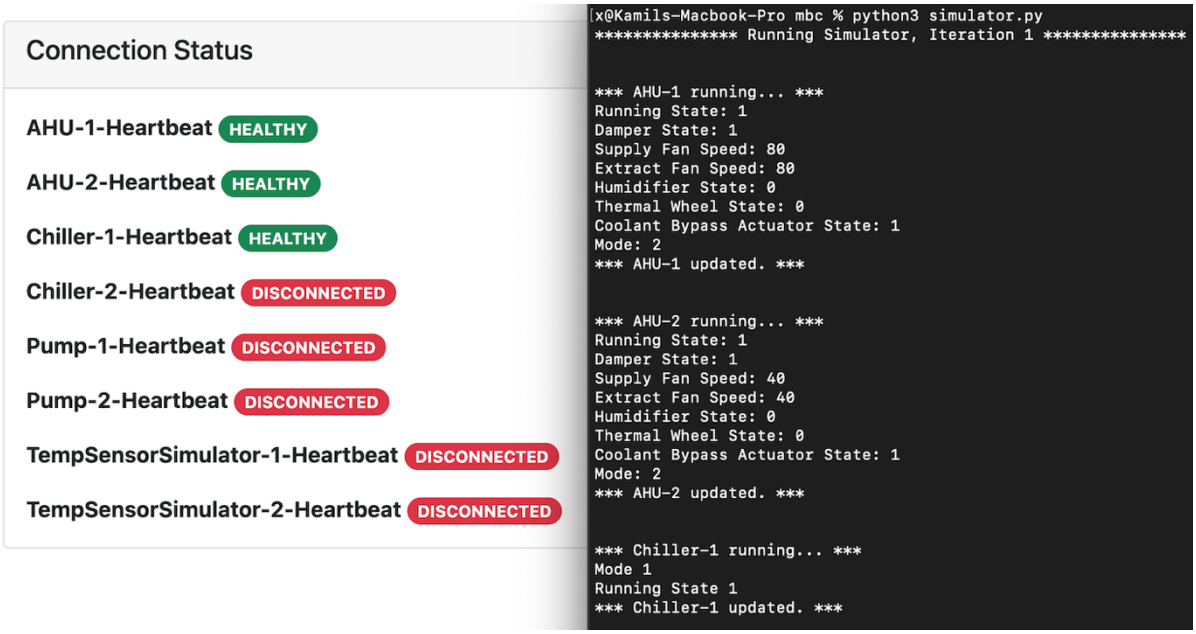

Fig 19. Connection Status component reacting to heartbeat changes of the running devices.

Another component that can be accessed in the building overview is an environmental data simulator that allows for simulating realistic scenarios for testing intelligent devices' behaviour. In reality, this data would be provided by several different sensors but for feasibility, the component has a form where values can be manually entered those trigger events in the Pub/Sub service of the web application. Environmental data is vital to intelligent devices and ensures their correct operations.

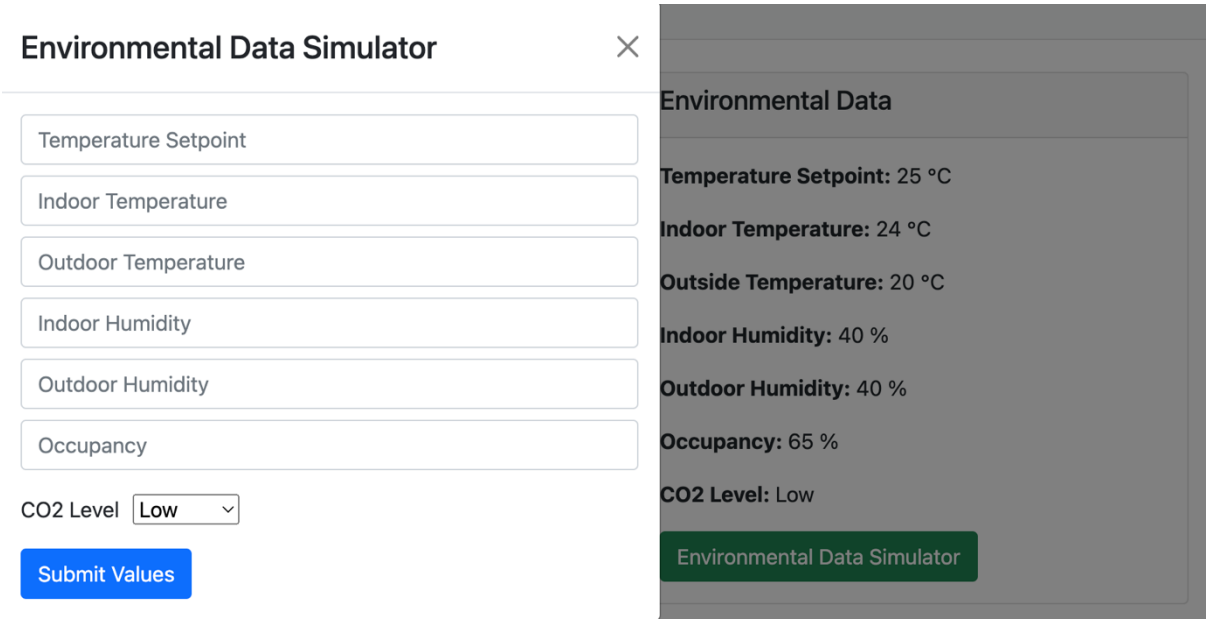

Fig 20. Environmental Data Simulator Component

When the status of all devices is "HEALTHY" we can start interacting with them by accessing the Air Handling Plant component that allows us to control and display detailed information about the state of the devices. It is a super component that includes all the assets on the network of Building 1. Fig 21.

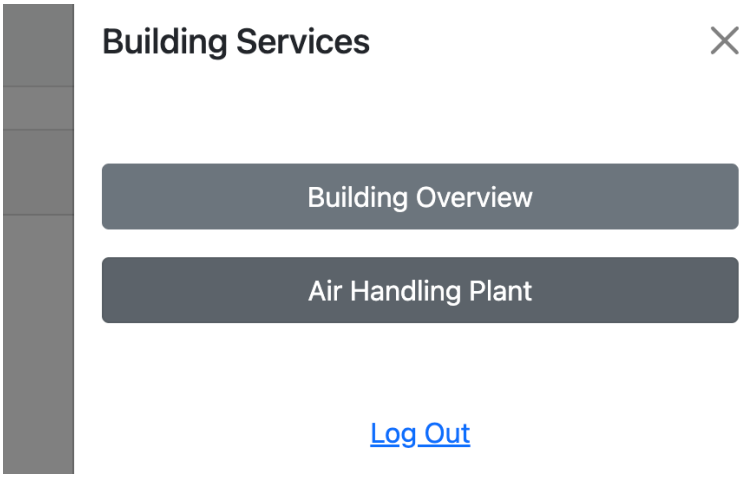

Fig 21. Opened Burger Menu from the Navigation Bar

The first two assets displayed are Air Handling Unit 1 (AHU-1 (Primary)) and Air Handling Unit 2 (AHU-2 (Secondary)) in Fig 22. They include buttons and switches that allow for controlling these units, the first is mode – the user can decide what the AHU should be doing, if the mode is ON device works at 100% supply and extract speed and toggle switches are available for overriding the state of its attributes. AUTO mode makes device work automatically by reading simulated data from the environmental data simulator that was previously specified by the user. OFF state switches the device off. Both AHUs are created from a single component since their attributes are the same and JSON data in the database has the same structure component is reused to create them.

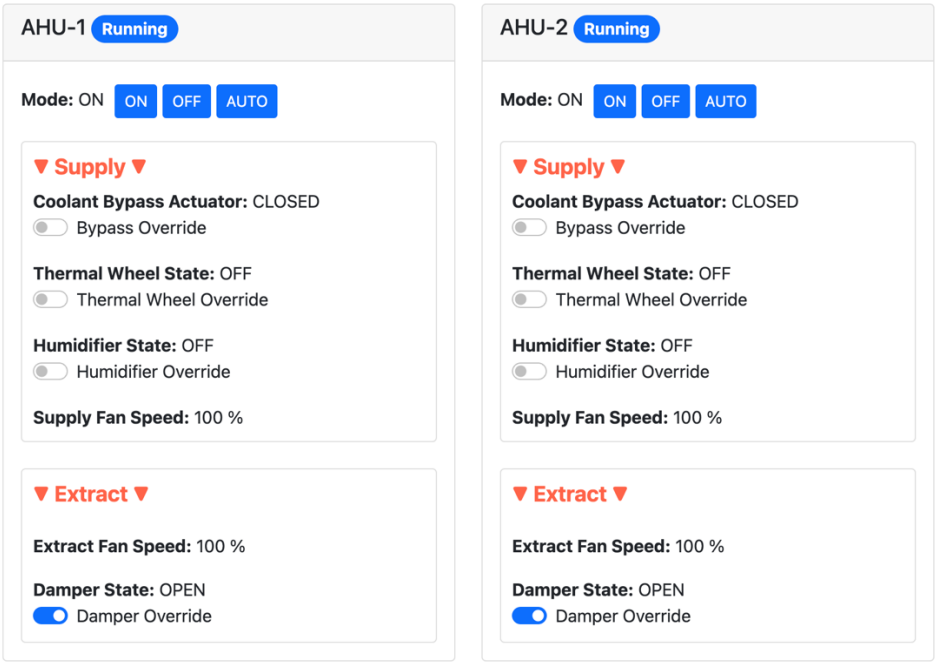

Fig 22. Air Handling Unit Components

The principle is the same for Chiller and Pump, they are also assets with primary and secondary units that are generated from reusable components of chiller and pump. The main difference between the chiller and AHU components is the different data structures and AHU having toggle switches and the chiller not having them. The chiller also displays

information collected from the temperature sensors that correspond to them as illustrated in Fig 23.

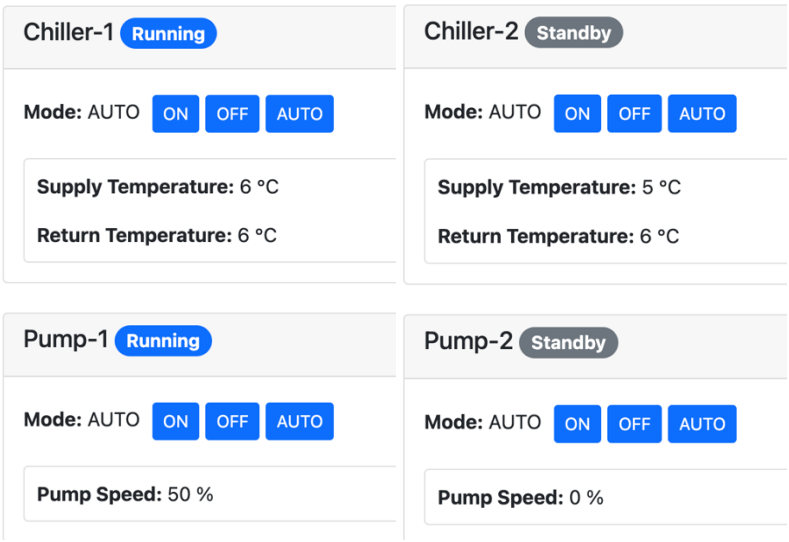

Fig 23. Chillers (Primary and Secondary) and Pumps (Primary and Secondary) Components.

Unit Testing

Unit testing was applied to the children's components of the web application corresponding to data regarding the state of the IoT devices. For unit testing these components, I used a method of shallow rendering that comes with a library called Jest test runner for JavaScript. It uses Enzyme library's shallow rendering that allows for rendering parent components exclusively without their children, which is perfect for unit testing. This way, we don't test any other components other than the one we passed to the shallow function.

I initialized a dummy version of tested components state with values that correspond to the html tags and text on the website to make sure that the data passed to the component from its state storage object transforms into a user friendly text and displays on the web page. I used the shallow function and a wrapper to isolate a component in a test block and then using find function we can locate those elements and compare them with expected text.

The unit tests can be found inside the "src" directory of "my-bms" folder in the GitHub repository in which another directory called "__test__" is located that consists of all the unit tests made with Jest, they can be run using "npm test" command after installing node modules.

Functional and Integration Testing

The emphasis on testing the prototype was to design the tests in such a way that can quickly troubleshoot many functions that indicates correct system responses. For example: when the user tries to log in, he/she will go to the URL of the application and enter their credentials, if they are correct and after a login button is pressed, the application redirects the user to the private route of the application's interface and if those login credentials were incorrect then the application displays a login error to the user, this means that the user authentication service works as expected, the component responsible for handling user input and displaying a button is loaded on the page and works correctly. This is where automation tools like Selenium come into play that can simulate human-machine interaction where we can automate an opening of the browser, entering a URL, logging in, inputting data or making sure that the data is visible.

I decided to use Selenium to test the prototype of MyBMS where I created a test suite inside of a single file. It is responsible for running several tests that ensure user credentials and the authentication system work correctly, once the user is logged in an environmental data simulator component is tested to make sure data can be entered using correct ranges and sets this data to dummy values at the end of the test. The next test was to ensure that the intelligent devices simulators are running and for this test to pass a simulator.py file of the MyBMS Controller part of the project must be running. After that, the Selenium test moves on the navigation bar to the burger menu click it and enter the air handling plant component where it checks if its children's components have successfully loaded to the screen. Once all the tests have passed, the Selenium test suite stops and gives control back to the user for further manual testing.

Running Selenium tests before the first attempt at manual testing of the application is a great way to ensure integration of the application with the database and intelligent devices but also authentication server and components existence all at the same time. To run the Selenium tests, the web application from the main branch of the repository must be running first, then the selenium application which also uses Node.js to work with the WebDriver for a specific browser to carry out the browser tests must be executed separately by running a command *npm install* to create node modules and then another command *node testSuite.js*. to run the tests. An important dependency for the test is a WebDriver that must be included in the same directory as the testSuite.js, for MyBMS, I mainly used the *geckodriver* that runs the tests in Mozilla Firefox web browser.

Chapter 6. Conclusion

To summarise this particular project, I researched control networks for a building management use case and how they evolved over the years of continuous improvements then I took this knowledge and imagined a building in which all of the devices can be separately connected to the Internet having their own controller that computes data and logic and then I thought about how these controllers can interact with each other and after further research into IoT technologies currently on the market to avoid having to reinvent the wheel, I proposed a new approach to control networks for building management systems use case where the devices can communicate with each other through triggers in the database. To check the feasibility of this approach I developed a system prototype for just one of the potential parts of the building's infrastructure called the air handling plant, which is responsible for supplying, extracting, and conditioning the air inside the building.

Overall, this approach to a real-time system development has been successful thanks to the asynchronous function invocation capabilities of the React framework that utilizes engines of the most popular Internet browsers like Google Chrome, Safari, and Mozilla Firefox to display changes in the HTML tags without a need to refresh the page. Due to ease of access to high-speed internet, data can travel across the world and back in a matter of milliseconds making it a highly efficient method of centralizing data corresponding to a real-time state of machinery represented as JSON objects making them easy to create, read, update, and delete on a serverless backend Firebase Realtime Database.

Over the months of learning the required stack of technologies and developing MyBMS alongside, the project has evolved from an untestable and unstable version that kind of worked but scored low on performance due to a bad initial design of the React application that would open way too many sockets than necessary, crash on rendering when too many objects were called or application having memory leaks that eventually were all eliminated as I increased my understanding of React, this led to the application becoming stable with adapted good code of practice that improved its performance, testability and code reusability that eventually became a first version prototype which is now available on GitHub.

Chapter 7. Reflections

Over the development lifecycle of this project I learned many new skills surrounding modern frontend development, backend integration, working with analogue input from real life sensors and cloud based data storage, there were many ups and downs throughout the development of the project the downs were mainly caused by lack of experience in project design and development from start to finish which required a substantial amount of hours in learning the technologies that were used in MyBMS project to essentially bring the idea of this integrated system to life.

Some of the biggest challenges were posted on the frontend side of the system because I never had an opportunity of using or learning an industry standard framework for client-side application development I felt lost and frustrated trying to solve relatively trivial tasks when I couldn't understand things like hooks and state handling in React which does have a steep learning curve and it was challenging to use. However, after persistent practice and learning process I managed to grasp the ins and outs of this framework and over time transformed my code from a previously redundant and coupled codebase to a more agile and reusable application components that are easy to test and use, I now feel comfortable using this framework and understand in a much deeper way how APIs and software systems are built in the real world using these freely available tools and technologies created by the giants of silicon valley to improve software development workflow.

When I ask myself if I would change anything or do anything differently if I had a chance to start over, I would probably choose a different use case that is more practical to me and focused on creating a RESTful API from scratch which is something I didn't know how to do before I started this project. It was an amazing learning experience about the real-time database, React, Node.js and good old fashioned Python problem solving. Even though I am happy with the results of the MyBMS prototype, it does work as expected but ever since I had to design the IoT devices as simulators from scratch using only code, the results are hard to visualise because of the intangibility of this approach. If I could start over I would probably go for a physical sensor for example and then I would focus on different things that we can actually do with that data such as visualising in form of graphs or manipulating it in different ways on the frontend. I would also use some other IoT technologies that I haven't had a chance to try out yet like the IoT core from Google Cloud which can handle the device management like retry logic and Pub/Sub events asynchronously.

In the future I will keep learning React and Node.js, I will keep building RESTful APIs, I will be implementing user authentication using OAuth v2 and Firebase and I consider the experience building this project invaluable, it was my first contact with a real project from scratch that I imagined, designed and developed. The mistakes I've made and victories I had have contributed enormously to the success of my future career as a software engineer.

Chapter 8. References

[1] Wang, S. and Xie, J., 2002. Integrating Building Management System and facilities management on the Internet. Automation in construction, 11(6), pp.707-715.

[2] R. Wilkinson, Become a BAS master, Engineered Systems (Troy) 17 (2) (2000) 44–49.

[3] ANSI/ASHRAE Standard 135-1995, A data communication protocol for building automation and control networks, ASH- RAE, 1995.

[4] ANSI/EIA 709.1-A-1999, Control network protocol specification, EIA, 1999.

[5] S.W.Wang,X.Q.Jin, Model-based optimal control of VAV air- conditioning system using genetic algorithm, Building and Environment 35 (6) (2000) 471–487.

[6] Wang, M., Qiu, S., Dong, H. and Wang, Y., 2017, October. Design an IoT-based building management cloud platform for green buildings. In 2017 Chinese Automation Congress (CAC) (pp. 5663-5667). IEEE.

[7] K. Chooruang, P. Mangkalakeeree, "Wireless heart rate monitoring system using MQTT," Procedia Computer Science, vol. 86, pp. 160-163, 2016.

[8] T.A.Abdulrahman,O.H.Isiwekpeni,N.T.SurajudeenBakindeandA.O. Otuoze, "Design, specification and implementation of a distributed home automation system," Procedia Computer Science, vol. 94, pp. 473- 478, 2016.

[9] Moroney, L., 2017. The firebase realtime database. In *The Definitive Guide to Firebase* (pp. 51-71). Apress, Berkeley, CA

[10] Li, W.J., Yen, C., Lin, Y.S., Tung, S.C. and Huang, S., 2018, February. JustIoT Internet of Things based on the Firebase real-time database. In *2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE)* (pp. 43-47). IEEE.

[11] Boffardi, B.P., 1999. Water treatment for HVAC & R systems. *ASHRAE Journal*, *41*(5), pp.52-56.

[12] Shen, W., Newsham, G. and Gunay, B., 2017. Leveraging existing occupancy-related data for optimal control of commercial office buildings: A review. *Advanced Engineering Informatics*, *33*, pp.230-242.

[i1] Phil Zito, 2015. BACnet Standard Objects on March 2017, The Definitive Guide to BACnet. Available from https://guides.smartbuildingsacademy.com/definitive-guide-bacnet [Accessed 24 January 2022]

Chapter 9. Appendices

Appendix A

Temperature Sensor

This appendix contains the boilerplate code of an AZ-Delivery DS18B20 Temperature Sensor Sonde. This script reads analogue input data from a GPIO PIN 28 on the Raspberry Pi using Python programming language.

Temperature Sensor Class:

```
import os
import glob
import time

class DS18B20:

    def __init__(self):
        os.system('modprobe w1-gpio')
        os.system('modprobe w1-therm')
        base_dir = '/sys/bus/w1/devices/'
        device_folder = glob.glob(base_dir + '28*')
        self._count_devices = len(device_folder)
        self._devices = list()
        i = 0
        while i < self._count_devices:
            self._devices.append(device_folder[i] + '/w1_slave')
            i += 1

    def device_names(self):
        names = list()
        for i in range(self._count_devices):
            names.append(self._devices[i])
            temp = names[i][20:35]
            names[i] = temp

        return names
```

```
# (one tab)
    def _read_temp(self, index):
        f = open(self._devices[index], 'r')
        lines = f.readlines()
        f.close()
        return lines

    def tempC(self, index = 0):
        lines = self._read_temp(index)
        retries = 5
        while (lines[0].strip()[-3:] != 'YES') and (retries > 0):
            time.sleep(0.1)
            lines = self._read_temp(index)
            retries -= 1

        if retries == 0:
            return 998

        equals_pos = lines[1].find('t=')
        if equals_pos != -1:
            temp = lines[1][equals_pos + 2:]
            return float(temp) / 1000
        else:
            return 999 # error

    def device_count(self):
        return self._count_devices
```

(most of the code in the script is modified from a script on the adafruit site)

Save the script by the name “*DS18B20classfile.py*”.

Main Script:

The following is a code for the main script:

```
import time
from DS18B20classFile import DS18B20

degree_sign = u'\xb0' # degree sign
devices = DS18B20()
count = devices.device_count()
names = devices.device_names()

print('[press ctrl+c to end the script]')
try: # Main program loop
   while True:
      i = 0
      print('\nReading temperature, number of sensors: {}'
                                                .format(count))
      while i < count:
         container = devices.tempC(i)
         print('{}. Temp: {:.3f}{}C, {:.3f}{}F of the device {}'
                           .format(i+1, container, degree_sign,
                           container * 9.0 / 5.0 + 32.0, degree_sign,
                           names[i]))
         i = i + 1

      time.sleep(1)

# Scavenging work after the end of the program
except KeyboardInterrupt:
   print('Script end!')
```