Yarmouk Private University

Faculty of Informatics and Communications Engineering

Department of Communication and Information Engineering

Building Distributed System to Handle Data in Internet of Things Applications

Senior Project Report

Prepared by:

Ammar Al-Madi Mohamed Abdullah

MHD. Manar Buz Aljedy

Supervisor

Dr. Mohamed Khaled Chahine Dr. Wassim Al-Juneidi

Eng. Rami Abbas

Second Semester
2020-2021

Abstract

The advent of Internet of Things (IoT) has kindled the possibility of umpteen number of challenges. One of the major challenges in the realization of IoT applications is interoperability among various IoT entities. Thus, the need for a new architecture comprising of smart control and a number of common systems have been identified by researchers. Our goal is in this project design a collaborative model and an architecture to take advantage of the available computing resources. The main challenge is to manage and maintain large number of devices and react smartly according to the data generated by them. So, we suggest the following a system based on IoT, with Internet Information Services (IIS) for setting up web servers, an ASP.NET model - view - controller (MVC) for establishing a remote, monitoring and control system by using web browser or mobile application and a Microsoft SQL Server as the database with the web browser connected to the Internet, where the sensor devices acquire the data and a send it to the server that performs the hard processing, the sensing data sent to the server by using the TCP/IP protocol, the users can provide instructions immediately without being present to check the conditions, which considerably reduces labor and time costs. This approach can be used in a diversity of real applications running in different environments under different conditions where a set of computing systems are available.

Arabic Abstract

ظهور إنترنت الأشياء أثار إمكانية وجود عدد لايحصى من التحديات، أحد هذه التحديات الرئيسية هو تحقيق التوافق بين مختلف تطبيقات ومنظومات إنترنت الأشياء وهذا يحتاج الى بنية جديدة تتألف من متحكم ذكي وعدد من الانظمة المشتركة يتم تحديدها من قبل الباحثين وهدفنا في هذا المشروع هو تصميم نموذج وبنية مشتركة للاستفادة من جميع موارد الحوسبة المتاحة ويكون الهدف الرئيسي هو إدارة المعلومات القادمة من الاجهزة العديدة، لذالك نقترح النظام التالي الذي يعتمد على إنترنت الأشياء مع خادمات معلومات الإنترنت لبناء مخدمات ويب ونموذج صفحات الخادم النشط من اجل أنشاء نظام تحكم عن بعد ومراقبة باستخدام متصفح الويب أو الهاتف المحمول باضافة الى قاعدة البيانات حيث تحصل أجهزة الاستشعار على البيانات وتقوم بإرسالها الى الخادم الذي يقوم بعمليات المعالجة الصعبة، المعلومات المرسلة الى الخادم تستخدم بروتوكول التحكم بالنقل، ويستطيع المستخدم التحكم بشكل مباشر دون الحاجة للتواجد قرب النظام مما يقلل بشكل كبير من التكاليف والوقت، يمكن استخدام هذا المنهج في مجموعة متنوعة من التطبيقات التي تعمل في بيئات مختلفة في ظل ظروف مختلفة حيث تتوافر مجموعة من انظمة الحوسبة.

Acknowledgment

It has been a great opportunity to gain lots of experience in real time projects, followed by the knowledge of how to actually design and analyze real projects. For that we want to thank all the people who made it possible for students like us. Special thanks to the graduation Project Unit for the efforts they did to provide us with all useful information and making the path clear for the students to implement all the education periods in real-time project design and analysis. Furthermore, we all the professors and visiting industry for the interesting lectures they presented which had great benefit for all of us. We would like to express our deepest gratitude to our graduation project supervisor Dr. Wassim Juneidi for his patience and guidance along the semester. In addition, we would like to express our sincere appreciations to Our department head and graduation project coordinator Dr. Mohamed Khaled Chahine for his guidance, continuous encouragement and support. Moreover, it is our duty to thank all the testing committee members for their generous discussions and encouragement.

List of Contents

List of Figures:

List of Tables:

List of Abbreviations

ACLs	Access Control List
AJAX	Asynchronous JavaScript
API	Application Programming Interface
ASP	Active Server Methods
ASMX	Active Server Pages
CSS	Cascading Style Sheets
DLL	Dynamic-link library
DOC	Distributed Object Computing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
IoT	Internet of Things
IIS	Internet Information Services
IP	Internet Protocol
IPsec	IP Security
JSON	JavaScript Object Notation
LPWAN	Lower-Power Wide Area Network
MACs	Message Authentication Codes
MVC	Model View Controller
MSMQ	Microsoft Message Queuing
NAT	Network Address Translation
OBDC	Open Database Connectivity
PHP	Hypertext Preprocessor
PRC	Galileo Public Regulated Service
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSI	Remote Service Invocation
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
SQL	Structured Query Language
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
TCP	Transmission Control Protocol
URL	World Wide Web
URI	Uniform Resource Identic
WAN	Wide Area Network
WCF	Windows Communication Foundation
WEB	World Wide Web
WSNs	Wireless sensor networks
XML	Extensible Markup Language

Chapter One

Theoretical Study

1.1 Problem Description and Formulation

Recent applications have been developed around the aforementioned concepts where sensing and processing capabilities of the devices play an important role. These devices are usually embedded systems and/or mobile devices such as smart phones, wearables, laptops, tablet PCs, etc. To deploy complex artificial intelligence applications in IoT environments provides a powerful driver for increased edge computing capabilities. Real-world use cases of artificial intelligence combining with the Internet of Things. This evolution promotes a digital transformation of the society by providing the citizens and professionals with advanced applications for sensing and analyzing data on the ground. Due to the recent successes and the anticipated breakthroughs in different fields, it has now become one of the most promising research areas. Indeed, this fact is largely accelerated by new smartphones and communication capabilities. However, the design of advanced IoT-based applications remains a challenge. Handling simultaneous data flows, data processing and/or complex mathematical function execution could overflow the computing capabilities of the embedded systems and mobile devices. One approach to overcome this drawback consists in designing a distributed system where the sensor devices are the distributed part to acquire the data and a centralized infrastructure that performs the hard processing. The classical client/server architecture has been designed for that purpose. Currently, this centralized infrastructure is usually deployed in the Cloud. However, this shift introduces several new risks, and some bottlenecks and delays may result from the communications among the devices and the centralized system. In particular, the latter drawback is strongest for multimedia data, for example, in applications that use video and image acquisition devices. For that reason, it is difficult to implement a centralized multimedia analysis system in the cloud. To overcome these bottlenecks and delays, this work extends our recent proposal of a distributed architecture to perform collaborative work for IoT-based environments and sharing the application workload among the available devices. This improved architecture takes into account the different network layers and their computing

platforms involved, from the remote Cloud servers to connected smart sensors and “things” The approach aims at optimizing the use of computational resources of an IoT environment while providing a framework able to obtain data from sensors, perform complex computational tasks and run advanced applications.

1.2 Project Summary

With the rapid development of the Internet in recent years, Internet-based applications such as remote monitoring systems are becoming increasingly popular in industry. Through the boundless Internet, a remote monitoring system allows the user to have remote real-time control of the situation in a factory by using a smartphone or computer. In the future, factories will have different types of devices, which will need to be integrated in an intelligent manner. Wireless sensor networks (WSNs) are the base technology of the Internet of Things (IoT). A WSN is a network that uses intelligent sensors to transmit and receive data. The applications of WSN include health, environment, industrial, and traffic monitoring. WSNs have not only contributed to the development of IoT but also led to the development of devices and technologies that support the growth of the Internet, such as QR codes, intelligent phones, social networks, and cloud computing. The open database connectivity (OBDC) method can be used for connecting and transferring the data, for example through Microsoft SQL Server. The web server must be established in order to provide a response to the client. Microsoft Internet Information Services (IIS) is a service for setting up the web server and delivering data to the client in Extensible Markup Language (XML) format. In recent years, developers have replaced XML with JavaScript Object Notation (JSON) format to transfer data because JSON is a lightweight data interchange format that is easy for humans to read and write as well as easy for machines to parse and generate. JSON increases the decoding speed of a browser and improves the efficiency of a website. In addition, developers use a model–view–controller (MVC) architectural pattern to design a website because it provides a way to divide a given application into three interconnected parts. The MVC design pattern decouples these major components,

enabling efficient code reuse and parallel development. A simple application for a remote monitoring system is a web oscilloscope. The web oscilloscope delivers acquisition data, which is stored in the database via TCP/IP and simulates a real oscilloscope to enable the user to easily observe the signal plot of an electrical circuit. Other applications of remote monitoring include video surveillance, appliance testing, and ocean monitoring. Because several studies have proposed that the graphical interfaces are more acceptable for users than using numerical tables as the interface, all these applications are designed with a friendly human machine interface to enable users to supervise a situation. According to market research of browsers used in intelligent devices, people use browsers for browsing websites on devices such as smartphones, tablets, and laptops.

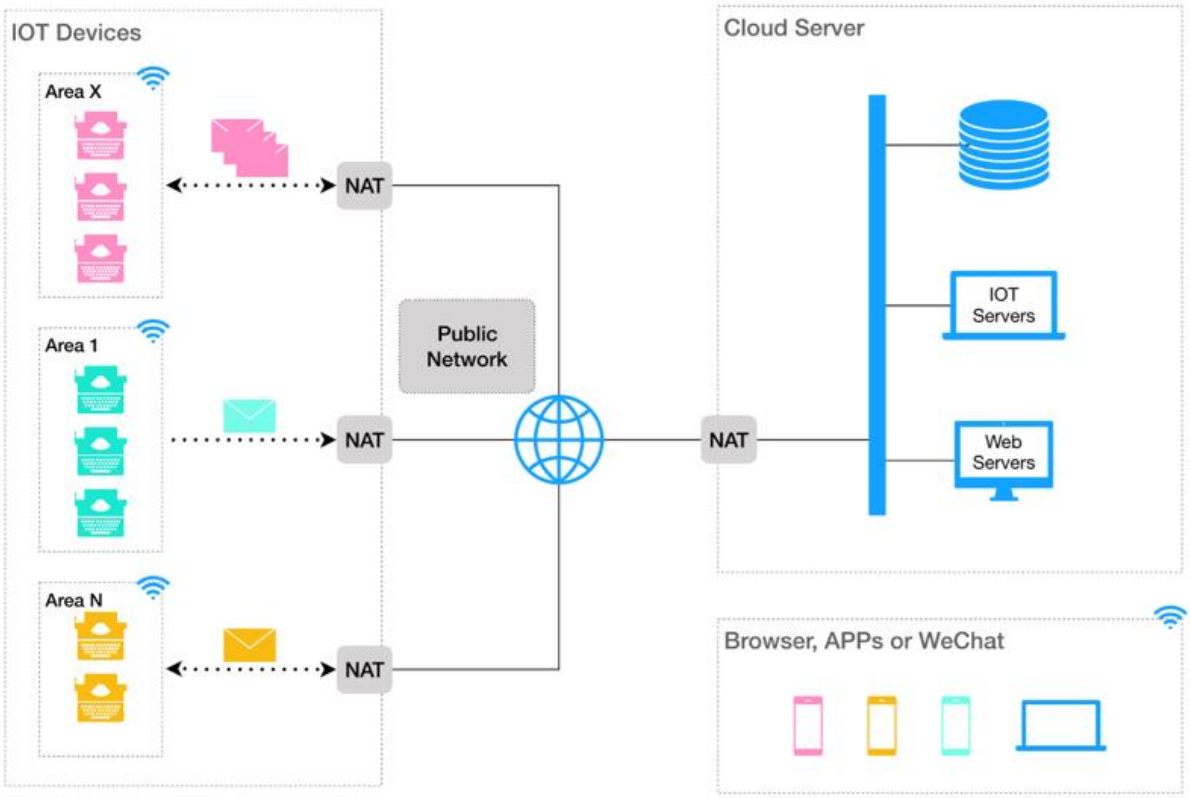

Figure 1.1 Architecture of remote monitoring system. [1]

1.3 Concept of the System and Related Technologies

This project is a web application design based on the ASP.NET framework using Visual Studio 2019. The coding languages used to establish the bridge between the server and client in the MVC structure were C#, SQL, JavaScript, HTML, and CSS. Some popular libraries, such as Razor and jQuery, and a common format for transferring data JSON.

1.3.1 ASP.NET MVC architecture

In this study, an MVC architecture was adopted to design the web application for a remote the IoT system. As shown in figure 2, the developer must divide an application into three types of components:

- A model stores data that is then retrieved according to commands from the controllers. It includes business logic and all data used in a project.
- A view presents the data from the controller based on changes in the model.
- A controller decides the data flow and sends commands to the model and view.

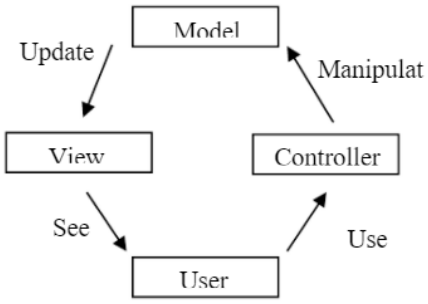

Figure 1.2 MVC architecture.[2].

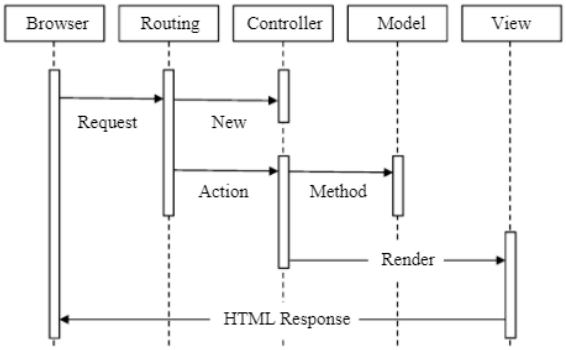

Figure 1.3 Life cycle of an MVC. [2].

Programming languages such as Java, C#, and PHP have popular MVC frameworks that are used in web application development. In this study, the ASP.NET MVC framework was applied to a remote the IoT system. The life cycle of an MVC architecture is presented in figure 3. The entry point for every MVC application is URL routing. After the ASP.NET platform receives

a request from a browser, a controller determines how it should be handled. The controller then determines the view and presents the view through URL routing. [2].

1.3.2 Relational Database

Microsoft SQL Server was adopted as a relational database management system to execute four basic functions of a database. Using a database to store data acquired with an analog-to-digital converter and establishing relations among each type of data is a common approach in industrial systems. To connect the web application to an SQL Server, Entity Framework should be used. Entity Framework can help a developer to relate the object in their code with a table in the database. [2].

1.4 Distributed Systems

A distributed system is an information-processing system that contains a number of independent computers that cooperate with one another over a communications network in order to achieve a specific objective. This definition pinpoints a number of aspects of distributed systems. Although the elementary unit of a distributed system is a computer that is networked with other computers, the computer is autonomous in the way it carries out its actions. Computers are linked to one another over a communications network that enables an exchange of messages between computers. The objective of this message exchange is to achieve a cooperation between computers for the purpose of attaining a common goal. [3].

1.4.1 Distribution of State and Behavior

A physical view of a distributed system describes this technology. It includes computers as nodes of the communications network along with details about the communications network itself. In contrast, a logical view of a distributed system highlights the applications aspects. can

therefore also be interpreted as a set of cooperating processes. The distribution aspect refers to the distribution of state (data) and behavior (code) of an application. The process encapsulates part of the state and part of the behavior of an application, and the application's semantics are achieved through the cooperation of several processes. The logical distribution is independent of the physical one. For example, processes do not necessarily have to be linked over a network but instead can all be found on one computer. [3].

Advantages

Distributed systems offer a variety of advantages compared to centrally or ganized mainframes. Decentralization is a more economic option because networked computing systems offer a better price/performance ratio than mainframe systems. The introduction of redundancy increases availability when parts of a system fail. Applications that can easily be run simultaneously also offer benefits in terms of faster performance in relation to centralized solutions. Distributed systems can be extended through the addition of components, thereby providing better scalability compared to centralized systems. [3].

Table 1.1 Advantages and disadvantages of centralized versus distributed systems. [3].

Criteria	Centralized system	Distributed system
Economics	low	high
Availability	low	high
Complexity	low	high
Consistency	simple	difficult
Scalability	poor	good
Technology	homogenous	heterogenous
Security	high	low

Disadvantages

The advantages offered by distributed systems are also countered by some disadvantages. The more components in a system, the greater the risk that the rest of the system will suffer unless

special measures are taken in the event that one of the components fails. Special mechanisms are needed to avert these failures and make them transparent to the user. Moreover, the many components that make up a distributed system are potential sources of failures. Due to the physical and time separation, consistency (for example, with distributed databases) is more of a problem than with centralized systems. Leslie Lampert presents a (cynical) alternative characterization that highlights the complexity of distributed systems. [3].

1.4.2 Client/Server Model

The client/server model introduces two roles that can be assumed by processes: the role of service (client) and the role of service provider (server). The distribution of roles implies an asymmetry in the distributed execution of an application. The server offers a service to which one or more clients has access. Here processes act as natural units in the distribution. In the context of distributed systems, the communication between client and server can be based on one of the mechanisms of Remote Procedure Call (RPC) is an example of synchronous request-oriented communication. The sender sends a request to the receiver and is passive until the receiver delivers the results of the request. An example of asynchronous request-oriented communication is Remote Service Invocation (RSI). During this type of communication, the sender remains active while the receiver is processing the request. Although RSI makes better use of the parallelism offered in distributed systems, RPC is based on popular programming paradigms and is therefore intuitive in its interpretation. The RPC introduced in the last section offers a fundamental communication mechanism for client/server interaction. The client is the initiator of an RPC, and the server provides the implementation of the remotely executed procedure. The request message contains all current input parameters for the procedure call. Conversely, the response message contains all results for the corresponding request produced by the server. The advantage of using remote procedure call as a communication mechanism for the client/server model is that it incorporates procedural programming paradigms and is

therefore easily understood. The implementation of the procedure is an integral part of the server, and the invocation of the procedure is part of the application running in the client. [3].

Advantages

An advantage of the client/server model is the intuitive splitting of applications into a client part and a server part. Based on conventional abstractions such as procedural programming, it simplifies the design and the development of distributed applications. Over and above this, it makes it easy to migrate or integrate existing applications into a distributed environment. The client/server model also makes effective use of resources when a large number of clients are accessing a high-performance server. Another advantage of the client/server model is the potential for concurrency. [3].

Disadvantages

From a different point of view, all these advantages could also be considered disadvantages. For example, the restriction to procedural programming paradigms excludes other approaches such as functional or declarative programming. Furthermore, even procedural paradigms cannot always ensure that transparency is maintained between local and remote procedure calls since transparency can no longer be achieved in the case of radical system failure. The concurrency mentioned earlier as an advantage can also lead to problems because of its requirement that processes be synchronized. [3].

1.5 Remoting System

1.5.1 Overview

Remoting is the process of programs or components interacting across certain boundaries. These contexts will normally resemble either different processes or machines. In the .NET Framework, this technology provides the foundation for distributed applications. The framework .NET includes .NET Remoting API that support the development of distributed applications. [1]. This is an extensible Distributed Object Computing (DOC) middleware infrastructure comparable to the Java Remote Method Invocation (RMI) although the latter adopts an entirely different internal architecture. Both frameworks allow objects on a client machine to communicate with remote objects on a server. [4].

1.5.2 .NET Remoting

.NET Remoting simplifies the development of distributed systems by offering an extensible infrastructure that permits objects that do not reside in the same memory space (or even on the same host) to communicate with one another in a transparent fashion. This implies that every message sent to a remote object will have to be delivered through an alternative mechanism. Therefore, each message from a local (client) object to a remote (server) object will be intercepted using a (double) proxy pattern. In addition, .NET Remoting gives you a flexible and extensible framework that allows for different transfer mechanisms (HTTP and TCP are supported by default), encodings Simple Object Access Protocol (SOAP) and binary come with the framework), and security settings (Internet Information Services (IIS) Security and SSL come out of the box). With these options, and the possibility of extending all of them or providing completely new implementations, .NET Remoting is well suited to distributed

applications. You can choose between HTTP-based transport for the Internet or a faster TCP-based one for LAN applications by literally changing a single line in a configuration file. [5]

1.5.3 Advantages of .NET Remoting

Several different architectures for the development of distributed applications already exist. You might therefore wonder why .NET introduces another, quite different way of developing those kinds of applications. One of the major benefits of .NET Remoting is that it's centralized around well-known and well-defined standards like HTTP and that it is directly tied to the .NET Framework and has not been retrofitted later.

Ease of Implementation

With .NET this concept of absolute ease of implementation has been extended to the development of distributed applications. There are no proxy/stub-compilation cycles. You don't have to define your interfaces in a different programming. A unique feature is that you don't have to decide up front on the encoding format of remoting requests; instead, you can switch from a fast TCP transport to HTTP by changing one word in a configuration file. You can even provide both communication channels for the same objects by adding another line to the configuration. [4]

Extensible Architecture

.NET Remoting offers the developer and administrator a vastly greater choice of protocols and formats than any of the former remoting mechanisms. In Figure 8, you can see a simplified view of the .NET Remoting architecture. Whenever a client application holds a reference to a remote object, it will be represented by a Transparent Proxy object, which "masquerades" as the destination object. This proxy will allow all of the target object's instance methods to be

called upon it. Whenever a method call is placed to the proxy, it will be converted into a message, and the message will pass various layers.

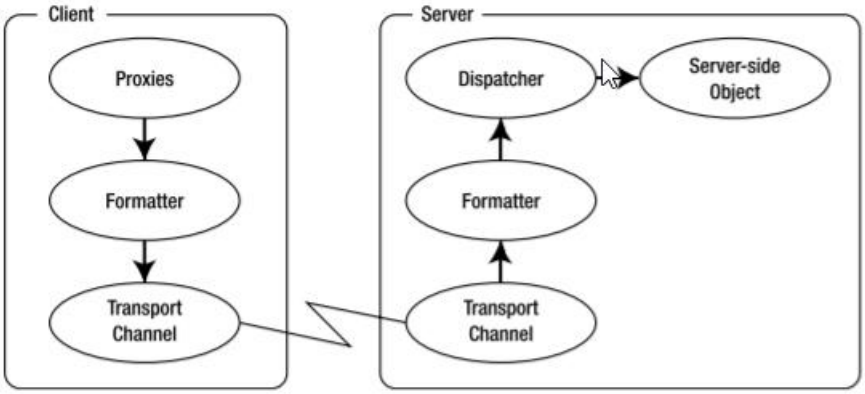

Figure 1.4 The NET Remoting architecture. [4].

The message will pass a serialization layer, the formatter which converts it into a specific transfer format such as SOAP. The serialized message later reaches a transport channel, which transfers it to a remote process via a specific protocol like HTTP or TCP. On the server side, the message also passes a formatting layer, which converts the serialized format back into the original message and forwards it to the dispatcher. Finally, the dispatcher calls the target object's method and passes back the response values through all tiers. [4].

Interface Definitions

.NET Remoting provides several different ways of defining those interfaces, as discussed in the following sections.

- Shared Assembly

In this case, the server-side object's implementation exists on the client as well. Only during instantiation is it determined whether a local object or an object on the remote server will be created. This method allows for a semitransparent switch between invoking the local implementation (for example, when working offline) and invoking server-side objects (for

example, to make calculations on better-performing servers when connected to the network).[4].

- Shared Interfaces or Base Objects

When creating a distributed application, you define the base classes or interfaces to your remote objects in a separated assembly. This assembly is used on both the client and the server. The real implementation is placed only on the server and is a class that extends the base class or implements the interface. The advantage is that you have a distinct boundary between the server and the client application. [4].

- Generated Metadata Assembly

This approach seems to be the most elegant one at first glance. You develop the server in the same way as when using the shared assemblies method. Instead of really sharing the Dynamic-link library (DLL) or EXE, you later extract the necessary metadata, which contains the interface information, using SoapSuds that will either need the URL to a running server or the name of an assembly as a parameter, and will extract the necessary information (interfaces, base classes, objects passed by value, and so on). It will put this data into a new assembly, which can be referenced from the client application. You can then continue to work as if you'd separated your interfaces right from the beginning. [4].

Marshalling Data

Marshalling is the process of converting a data field, or an entire set of related structures, into a serialized string that can be sent in a message. To marshall a binary number, one might convert it to hexadecimal digit string, if the message format must be text. If the message will carry binary data, the binary number might be converted into 4 little-endian normalized binary bytes and sent that way. Pointers are harder; one often has to convert them into an abstract reference (e.g., a "node number") that is independent of the actual memory locations. [5].

- Marshal-by-value objects:

  These objects are copied and passed by value out of the application domain. When client calls a method on marshal-by-value-object, the remoting system creates a copy of this object and passes the copy to the client application domain. The copy hence received can handle any method call in client domain. Using Marshal-by-value-object reduces resource consuming trip across network.

- Marshal-by-reference objects:

  The clients that use these objects need a proxy to access the object remotely. When client calls a method on Marshal by reference object, the remoting system create proxy object in the caller application that contains the reference of all method and properties of the object.

Serialization of Data

In .NET Remoting the encoding/decoding of objects is natively supported. You just need to mark such objects with the [Serializable] attribute or implement the interface ISerializable and the rest will be taken care of by the framework.

The underlying .NET runtime formatting mechanism marshals simple data types and subobjects (which have to be serializable or exist as remote objects), and even ensures that circular references will be tracked and transferred correctly. [5].

Multiserver/Multiclient

When you use remote objects (as opposed to using copies of remotely generated objects that are passed by value), .NET automatically keeps track of where they originated. So a client can ask one server to create an object and safely pass this as a method's parameter to another server. The second server will then directly execute its methods on the first server, without a round trip through the client. Nevertheless, this also means there has to be a direct way of communication

from the second server to the first one—that is, there must not be a firewall in between, or at least the necessary ports should be opened. [5].

1.6 Windows Communication Foundation

1.6.1 WCF ─ Overview

Windows Communication Foundation (WCF). The elementary feature of WCF is interoperability. It is one of the latest technologies of Microsoft that is used to build service-oriented applications. Based on the concept of message-based communication, in which a Hypertext Transfer Protocol (HTTP) request is represented uniformly, WCF makes it possible to have a unified Application programming interface (API) irrespective of diverse transport mechanisms. WCF was released for the first time in 2006 as a part of the .NET framework with Windows Vista, and then got updated several times. 4.8 (2019) is the most recent version that is now widely used. A WCF application consists of three components. [6]:

- WCF service.
- WCF service host.
- WCF service client.

1.6.2 Fundamental Concepts of WCF

- Message ─ This is a communication unit that comprises of several parts apart from the body. Message instances are sent as well as received for all types of communication between the client and the service.
- Endpoint ─ It defines the address where a message is to be sent or received. It also specifies the communication mechanism to describe how the messages will be sent along

with defining the set of messages. A structure of an endpoint comprises of the following parts:

  - Address ─ Address specifies the exact location to receive the messages and is specified as a Uniform Resource Identifier (URI). It is expressed as scheme://domain[:port]/[path]
  - Binding ─ It defines the way an endpoint communicates. It comprises of some binding elements that make the infrastructure for communication. For example, a binding states the protocols used for transport like TCP, HTTP, etc., the format of message encoding, and the protocols related to security as well as reliability.
  - Contracts ─ It is a collection of operations that specifies what functionality the endpoint exposes to the client. It generally consists of an interface name.

- Hosting ─ Hosting from the viewpoint of WCF refers to the WCF service hosting which can be done through many available options like self-hosting, Internet Information Services (IIS) hosting, and WAS hosting.
- Metadata ─ This is a significant concept of WCF, as it facilitates easy interaction between a client application and a WCF service. Normally, metadata for a WCF service is generated automatically when enabled, and this is done by inspection of service and its endpoints.
- WCF Client ─ A client application that gets created for exposing the service operations in the form of methods is known as a WCF client. This can be hosted by any application, even the one that does service hosting.
- Channel ─ It is a medium through which a client communicates with a service. Different types of channels get stacked and are known as Channel Stacks.
- Simple Object Access Protocol (SOAP) ─ It is an Extensible Markup Language (XML) document comprising of a header and body section. [7].

1.6.3 WCF Features

- Service Orientation: one consequence of using Web services (WS) standards is the WCF enables you to create service-oriented application. service-oriented architectures (SOA) are the reliance on WEB services to send and receive data.
- Interoperability: WCF implements modern industry standards for Web service interoperability.
- Multiple Message Patterns: Messages are exchanged in one of several patterns. The most common pattern is the request/reply pattern, where one endpoint requests data from a second endpoint such as a one-way message in which a signal endpoint sends a message without any expectation of reply.
- Data Contracts: Because WCF is built using .Net Framework, it also includes code-friendly methods of supplying the contracts you want to enforce.
- Security: Messages can be encrypted to protect and you can require users to authentication themselves before being allowed to receive messages. Security can be implemented using well-know standers such as Secure Sockets Layer (SSL) or WS-Secure Conversation.
- Multiple Transports and Encoding: Messages can be sent in any of several built-in transport protocols and encodings. The most common protocols and encoding is to send text encoded SOAP messages using is the HTTP for use on the World Wide Web. Alternatively, WCF allow you to send message over Transmission Control Protocol (TCP), Named Pipes or Microsoft Message Queuing (MSMQ). [7].

1.6.4 Advantages of WCF

- It is interoperable with respect to other services. This is in sharp contrast to .NET Remoting in which both the client and the service must have .Net.

- WCF services offer enhanced reliability as well as security in comparison to Active Server Methods (ASMX) web services.
- Implementing the security model and binding change in WCF do not require a major change in coding. Just a few configuration changes are required to meet the constraints.
- WCF has built-in logging mechanism whereas in other technologies, it is essential to do the requisite coding.
- WCF has integrated Asynchronous JavaScript (AJAX) and support for JavaScript Object Notation (JSON).
- It offers scalability and support for up-coming web service standards.
- It has a default security mechanism which is extremely robust. [6].

1.6.5 WCF – Architecture

WCF has a layered architecture that offers ample support for developing various distributed applications.

Contracts

The contracts layer is just next to the application layer and contains information similar to that of a real-world contract that specifies the operation of a service and the kind of accessible information it will make. Contracts are basically of four types discussed below in brief:

Figure 1.5 WCF Contracts architecture. [6].

- Service contract: This contract provides information to the client as well as to the outer world about the offerings of the endpoint, and the protocols to be used in the communication process.
- Data contract: The data exchanged by a service is defined by a data contract. Both the client and the service has to be in agreement with the data contract.
- Message contract: A data contract is controlled by a message contract. It primarily does the customization of the type formatting of the SOAP message parameters. Here, it should be mentioned that WCF employs SOAP format for the purpose of communication. SOAP stands for Simple Object Access Protocol.
- Policy and Binding: There are certain pre-conditions for communication with a service and such conditions are defined by policy and binding contract. A client needs to follow this contract. [6].

Service Runtime

The service runtime layer is just below the contracts layer. It specifies the various service behaviors that occur during runtime. There are many types of behaviors that can undergo configuration and come under the service runtime.

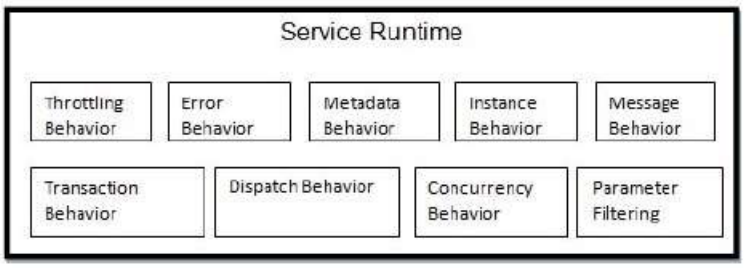

Figure 1.6 WCF Service Runtime architecture. [6].

- Throttling Behavior ─ Manages the number of messages processed.
- Error Behavior ─ Defines the result of any internal service error occurrence.

- Metadata Behavior ─ Specifies the availability of metadata to the outside world.
- Instance Behavior ─ Defines the number of instances that needs to be created to make them available for the client.
- Transaction Behavior ─ Enables a change in transaction state in case of any failure.
- Dispatch Behavior ─ Controls the way by which a message gets processed by the infrastructure of WCF.
- Concurrency Behavior ─ Controls the functions that run parallel during a client-server communication.
- Parameter Filtering ─ Features the process of validation of parameters to a method before it gets invoked. [6].

Messaging

This layer, composed of several channels, mainly deals with the message content to be communicated between two endpoints. A set of channels form a channel stack and the two major types of channels that comprise the channel stack are the following ones:

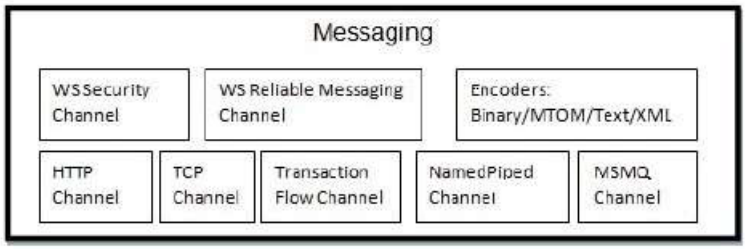

Figure 1.7 WCF Messaging architecture. [6].

- Transport Channels ─ These channels are present at the bottom of a stack and are accountable for sending and receiving messages using transport protocols like HTTP, TCP, Peer-to-Peer, Named Pipes, and MSMQ.
- Protocol Channels ─ Present at the top of a stack, these channels also known as layered channels, implement wire-level protocols by modifying messages. [6].

Activation and Hosting

The last layer of WCF architecture is the place where services are actually hosted or can be executed for easy access by the client. This is done by various mechanisms discussed below in brief.

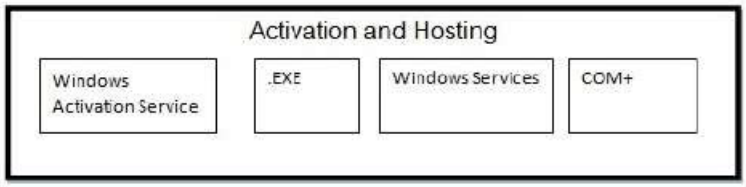

Figure 1.8 WCF Activation and Hosting architecture. [6].

- IIS ─ IIS stands for Internet Information Service. It offers a myriad of advantages using the HTTP protocol by a service. Here, it is not required to have the host code for activating the service code; instead, the service code gets activated automatically.
- Windows Activation Service ─ This is popularly known as WAS and comes with IIS 7.0. Both HTTP and non-HTTP based communication is possible here by using TCP or Named Pipe protocols.
- Self-hosting ─ This is a mechanism by which a WCF service gets self-hosted as a console application. This mechanism offers amazing flexibility in terms of choosing the desired protocols and setting own addressing scheme.
- Windows Service ─ Hosting a WCF service with this mechanism is advantageous, as the services then remain activated and accessible to the client due to no runtime activation. [6].

1.7 Internet of Things

IoT (Internet of Things) is an advanced automation and analytics system which exploits networking, sensing, big data, and artificial intelligence technology to deliver complete systems for a product or service. These systems allow greater transparency, control, and performance when applied to any industry or system. IoT systems have applications across industries through their unique flexibility and ability to be suitable in any environment. They enhance data

collection, automation, operations, and much more through smart devices and powerful enabling technology. [8].

1.7.1 IoT-Key Features

The most important features of IoT include artificial intelligence, connectivity, sensors, active engagement, and small device use. A brief review of these features is given below:

- AI – IoT essentially makes virtually anything “smart”, meaning it enhances every aspect of life with the power of data collection, artificial intelligence algorithms, and networks.
- Connectivity – New enabling technologies for networking, and specifically IoT networking, mean networks are no longer exclusively tied to major providers. Networks

  can exist on a much smaller and cheaper scale while still being practical. IoT creates these small networks between its system devices.
- Sensors – IoT loses its distinction without sensors. They act as defining instruments which transform IoT from a standard passive network of devices into an active system capable of real-world integration.
- Active Engagement – Much of today's interaction with connected technology happens through passive engagement. IoT introduces a new paradigm for active content, product, or service engagement.
- Small Devices – Devices, as predicted, have become smaller, cheaper, and more powerful over time. IoT exploits purpose-built small devices to deliver its precision, scalability, and versatility. [8].

1.7.2 IoT-Advantages

The advantages of IoT span across every area of lifestyle and business. Here is a list of some of the advantages that IoT has to offer:

- Improved Customer Engagement – Current analytics suffer from blind-spots and significant flaws in accuracy and as noted, engagement remains passive. IoT completely transforms this to achieve richer and more effective engagement with audiences.
- Technology Optimization – The same technologies and data which improve the customer experience also improves device use, and aid in more potent improvements to technology. IoT unlocks a world of critical functional and field data.
- Reduced Waste – IoT makes areas of improvement clear. Current analytics give us superficial insight, but IoT provides real-world information leading to more effective management of resources.
- Enhanced Data Collection – Modern data collection suffers from its limitations and its design for passive use. IoT breaks it out of those spaces, and places it exactly where humans really want to go to analyze our world. It allows an accurate picture of everything. [8].

1.7.4 IoT-Disadvantages

Though IoT delivers an impressive set of benefits, it also presents a significant set of challenges. Here is a list of some its major issues:

- Security – IoT creates an ecosystem of constantly connected devices communicating over networks. The system offers little control despite any security measures. These leaves users exposed to various kinds of attackers.
- Privacy – The sophistication of IoT provides substantial personal data in extreme detail without the user's active participation.

- Complexity – Some find IoT systems complicated in terms of design, deployment, and maintenance given their use of multiple technologies and a large set of new enabling technologies.
- Flexibility – Many are concerned about the flexibility of an IoT system to integrate easily with another. They worry about finding themselves with several conflicting or locked systems.
- Compliance – IoT, like any other technology in the realm of business, must comply with regulations. Its complexity makes the issue of compliance seem incredibly challenging when many consider standard software compliance a battle. [8].

1.7.5 IoT Networking

The definition of cloud is quite accurately listing characteristics, service models and deployment models, however it does not refer to networks. Networks in IoT are not in fact a characteristic but they are enablers. One key-contributor factor for the success widespread of IoT technology is in fact due to the raise of modern, fast, reliable, low-latency and low-cost networks. Specifically, for IoT the most common network types range between Bluetooth, traditional Wireless Local Area Network (WAN), cellular and a new generation of Lower-Power Wide Area Network (LPWAN). Few network technologies have a clear advantage compared to other. WLAN and Bluetooth technology are without any doubt the most common type of consumer network in the market at the moment. They both work in a license-free radio frequency band; they both ensure a good bandwidth transfer rate and they both requires fairly inexpensive receivers. Limitation comes however from the fact that they have evident range limitation that precludes them to be the main choice for being used in extensive IoT applications. Range in fact is limited to few tens of meters in WLAN and few meters for Bluetooth connection. As IoT industrial applications are intended to work mostly with devices distributed in a wide area, often with bad cellular coverage, and that would require a strict power

management to extend the battery lifetime, a new technology of Lower-Power Wide Area Networking (LPWAN) is raising in IoT.

LPWAN are networks that combine technologies in order to achieve long distance, robust and low-bit rate communications with battery operated sensors geographically located in a wide area. [9].

Chapter Two

Practical Implementation

In this chapter we will study the main part to form the Distributed Applications that we will use for this project.

2.1 Introduction

Developing and deploying applications are critical aspects of providing modern organizations with new and innovative services while helping them maintain and operate their existing capabilities. Although there are an increasingly diverse set of application development technologies, .NET has been the de facto standard for Windows since it was first released by Microsoft, and with a growing ecosystem of alternative .NET implementations, it is increasingly being chosen for a variety of cross platform workloads. [10]

Nonetheless, no application is an island, and .NET applications not only depend on environments to execute in, but also require a plethora of additional services, including, but not limited to, relational databases, queuing middleware, authentication and authorization services, file storage, networking, caching, and a variety of operational monitoring and logging services. ASP.NET adds to the .NET platform:

- **Base framework for processing web requests in C# or F#**
- **Web-page templating syntax**, for building dynamic web pages using C#
- **Libraries for common web patterns**, such as Model View Controller (MVC)
- **Authentication system** that includes libraries, a database, and template pages for handling logins, including multi-factor authentication and external authentication with Google, Twitter, and more.
- **Editor extensions** to provide syntax highlighting, code completion, and other functionality specifically for developing web pages. [10].

2.2 System Overview

The entire system is designed following to the principle of modularity. As shown below (figure 2.1), the system is modularized into 3 parts: field data collection module (IOT System), service-oriented communication module (Web Server), and user application module (ASP.NET MVC). Each part communicates with others following rigorous use of well-defined data interfaces. Both the field system and remote system provide welding monitoring function and data query function. The detailed function for each part will be shown in later sections.

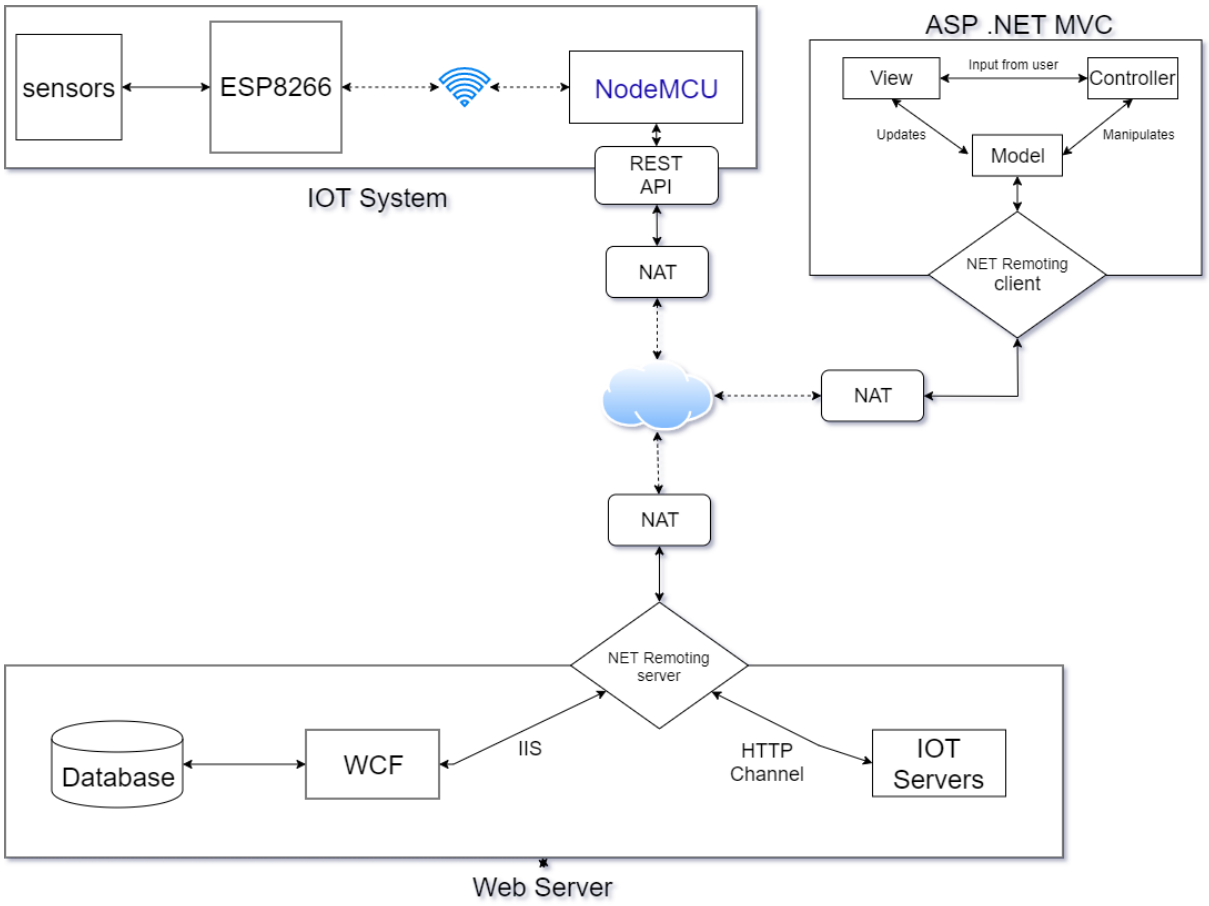

Figure 2.1 System Block Diagrams

2.3 Internet of Things System

The Smart House is the full controlled automated system designed efficiently to fit end user requirements. The technical progress all around the universe assigned the buildings and the houses to be equipped by Building Management System BMS that controls and monitors the electrical devices in various environments based on Internet of Things (IoT) approach. The electrical devices such as Air conditions, TVs, Ventilations, House Lights, and Irrigation systems etc., need to be controlled to guaranty security developments, the benefits of the smart house technology are colossal and can be specified as follows:

1. Energy enhancement: The house lights and the devices can be controlled and monitored permanently such that the lights or the devices can be turned OFF depending on the proposed website. Hence, the electrical energy would be reserved efficiently.

2. Security enhancement: The system is monitored based on webpage monitoring system and the house can be surrounded by cameras to capture the events with respect to motion sensors. In addition, the smart house system can contain more features like fingerprints and key cards that maximize system security for a little bit more.

3. Accessibility: the voice commands can help the incompetent persons to control house lights and the devices using their voice.

4. The convenience: The availability of control system designed to dominate house gadgets and simplify the life based on this system such that all appliances such as air conditions, TVs, multimedia players, etc. are controlled anytime throughout the house via appropriate measurement.

5. Life time: The efficiency and the life time are expanded due to the reasons presented above. [10].

2.3.1 Smart System Components

The proposed system is constructed of several elements identified as follows:

ESP8266 Node MCU

Arduino based ESP8266 Node MCU is a new microcontroller aspect that is created in corporation with Arduino Company. This microcontroller works somewhat according to Arduino microcontroller specifications regardless AVR processors that lead the entire module to be compiled by Arduino IDE C++ compiler. The module is considered a complete kit due to the specification that was added the ESP board to reduce the individual sectors that needed to be attached to the board in order to perform specific roles. The new ESP MCU module was configured with respect to Arduino Uno board manager and SAM core. The term 'Core' was given to the group of software units that are needed to compile the Arduino C++ headers by using MCU language. The creativity of ESP8266 module leads to build robust and complete systems due to the design methodology that developed Arduino core under the domination of ESP8266 Wi – Fi based on GitHub ESP8266 core webpage. This module is learning software platform that combines between ESP8266 and Node MCU firmware. The MCU module that is shown in Figure 2.2 works under the supervision of 802.11n and 802.11b networks. This means that it can serve as an Access Point AP, Wi – Fi station or both station and AP at the same time [10].

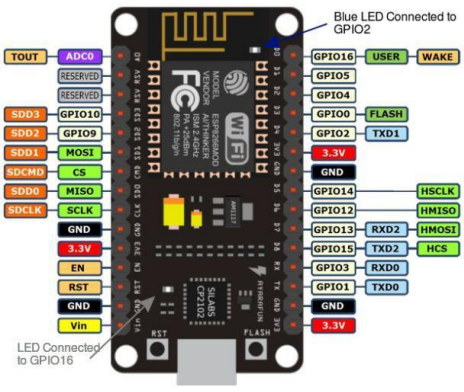

Figure 2.2 ESP8266 Node MCU Module

Node MCU Specifications & Features

- Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- Digital I/O Pins (DIO): 16
- Analog Input Pins (ADC): 1
- Flash Memory: 4 MB
- SRAM: 64 KB
- Clock Speed: 80 MHz
- Small Sized module to fit smartly inside your IoT projects.

ESP8266 ESP-01 WiFi Module

The ESP8266 ESP-01 is a serial to WiFi breakout module with a built in ARM microprocessor that has 1MB of memory and 2 GPIOs brought out to the header for connecting to peripherals. It can be used as a serial to WiFi bridge to add WiFi capability to a project or it can even be programmed directly and used as a little stand-alone processor. It has full TCP/IP capability built-in. [10].

Key Features of ESP8266 ESP-01 WIFI Module

- 32-bit RISC Tensilica Xtensa LX Processor running at 80MHz
- 1MB Flash Memory
- IEEE 802.11 b/g/n WiFi
- 2 GPIO
- 3.3V Operation

Temperature – Humidity Sensor

The Temperature – Humidity sensor that is known by DHT11, reads and measures the temperature and humidity degrees in a single distinctive model. Temperature (T) and Humidity (H) Sensor are treated in a complex way with a calibration of digital signal output. The sensor

guarantees extraordinary reliability and exceptional long term stability due to the private digital signal acquirement in the sensing technology. This module contains resistive humidity component and an NTC temperature component, connected to a high performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost effectiveness. DHT sensor measures both (T) and (H) which hands the readings through ESP8266 module with respect to Net Pie website. The module is constructed of three terminals identified by Vcc, Data, and Gnd. The sensor acts well if linked with the digital pins of a microcontroller. As the schematic connection demonstrated in Figure 2.3, VCC pin must be provided by 5 V from ESP8266 MCU, the data is chosen to be connected to the digital pin D5 of ESP8266, and the Gnd terminal of the sensor is connected to the Gnd pin of ESP8266 board.[10].

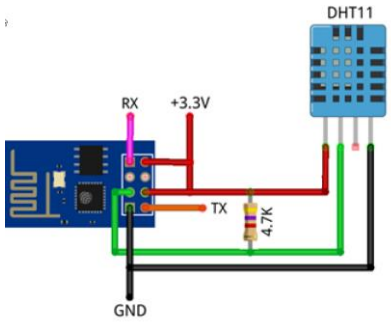

Figure 2.3 Temperature – Humidity (DHT11) Module

Port Diagram

Since the proposed system is a distributed system, and for ease of connection with the main controller, we used the ESP-01 model, which allows to connect all sensors wirelessly

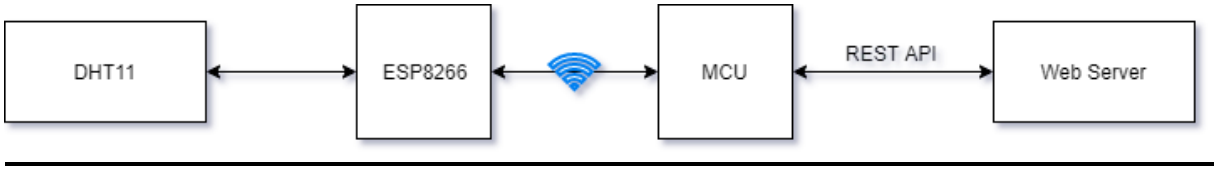

Figure 2.4 Port Diagram for IoT system

The sensors are connected to the main controller (Node MCU) via REST API, A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services, when we request the IP address assigned to each sensor, the information is sent via esp-01 in Json to the Node MCU to the webserver

```
{"sensor":"TemperatureSensor”,"DataT":'23.100000381469727','DataH":'68.20607'}
```

In the Node MCU, there is a function to read the information and put it into variables to deal with it.

```
pt::read_json("example.json", loadPtreeRoot);
   pt::ptree temp ;
   std::string name ;
   std::string DateT ;
   std::string DateH ;
   name = temp.get_name ("Sensor");
   DataT = temp.get_time("DataT");
   DataH = temp.get_time("DataH");
```

HTTP Request Methods: GET vs POST

The Hypertext Transfer Protocol (HTTP) works as a request-response protocol between a client and server:

- The server submits an HTTP request to the ESP8266
- The ESP returns a response to the server
- Finally, the response contains status information about the request and may also contain the requested content.

HTTP GET

GET is used to request data from a specified resource. It is often used to get values from APIs.[].

For example, you can have:

```
GET /update-sensor?temperature=value1&humidity=value2
```

HTTP POST

POST is used to send data to a server to create/update a resource. For example, publish sensor readings to a server. [].

The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST /update-sensor HTTP/1.1 Host: example.com

api_key=api&sensor_name=name&temperature=value1&humidity=value2
```

ESP8266 HTTP GET: JSON Data

This method is requested via the following link https://esp8266.local/Data and return the data in JSON format

```
void getData() {
  String mydataT = JSON.stringify(Temperature);
    String mydataH = JSON.stringify(Humidity);
  String input =
"{\"sensor\":\"Temperature\",\"Time\":"'+GetTime()+'",\"Date\":"'+GetData()+'",\"Data\":"'
+mydataT+","+mydataH+"'}";
   server.send(200, "text/json", ""+input+"");
}
```

In addition to fetching information from sensors, there are also functions to control the ports of the ESP via getSettings()

```
Void getSettings() {
    String response = "{";
    if (server.arg("LED_BUILTIN")== "HIGH"){
       digitalWrite(LED_BUILTIN, HIGH);
    }
    else if (server.arg("LED_BUILTIN")== "LOW"){
        digitalWrite(LED_BUILTIN, LOW);
    }
    response+="}";
    server.send(200, "text/json");
}
```

The user can control the home lighting and other applications through the dashboard page

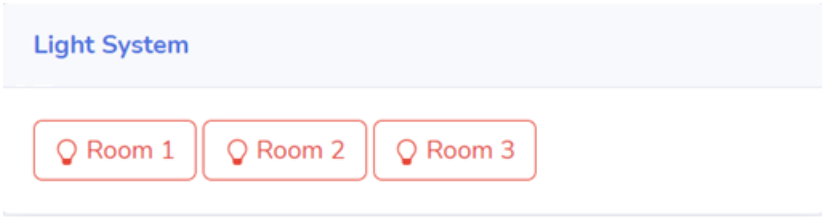

Figure 2.5 Light System

In addition, there are other functions for calculating the time and date when the event occurs, the functions of communicating with the wireless network, and the functions of building a REST server. The entire code will be included in the appendix and explained

2.3.2 CCTV System

For the monitoring system , we use Raspberry PI 4 and the Raspberry Pi camera, first we need to connect the ribbon to the CSI connector to the Raspberry PI board and turn the raspberry on Once it's on, open a terminal window. And we execute:

```
$ sudo raspi-config
```

And chooes interfacing option-p1 camera

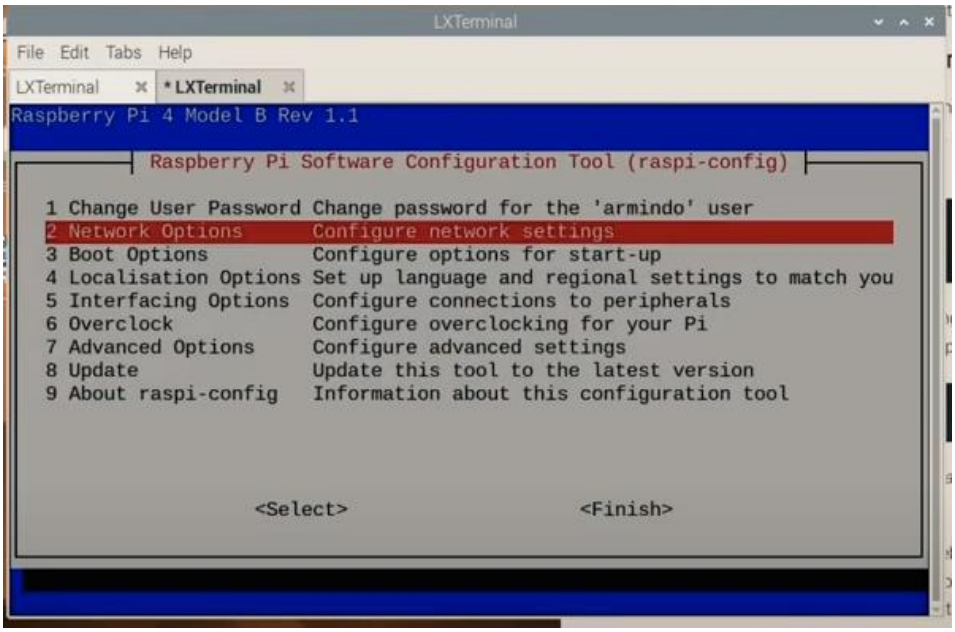

Figure 2.6 Raspberry Pi Configuration tool

Creating a Livestream

Open a terminal window. And execute:

```
raspivid -o – -t 0 -n | cvlc -vvv stream:///dev/stdin –sout ‘#rtp{sdp=rtsp://:8554/}’
:demux=h264
```

This creates an RTSP stream from the Raspberry PI camera that is accessible from the local network, we can access it through the dashboard page

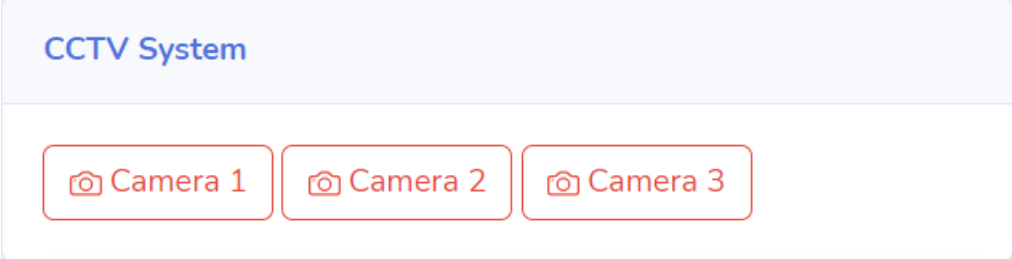

Figure 2.7 CCTV system

Each camera has its own link for example camera 1 link: http://192.168.1.14:5000

2.4 MVC Application

To test the practical performance of this system, we also develop a Web MVC application based on C# language. The website provides basic data query service and remote monitoring function. To make sure that the monitoring function is real-time, the website establishes a remoting connection with the web server-based .NET Remoting library. MVC model been explained in the theoretical section and here will be the steps to build the web application.

In this project we use Visual Studio 2019 So we downloaded and installed it on the device whose characteristics are mentioned in the following (table 2.1)

Table 2.1 Computer Characteristic

OS Name	System Type	Processor	Total Physical Memory	Local Fixed Disk
Microsoft Windows 10 Pro	x64-based PC	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core(s), 8 Logical Processor(s)	15.8 GB	SSD 117.7 GB (126,406,950,912 bytes)

In Visual Studio, from the **File** menu, select **New** -> **Project**. You will be presented to the New Project dialog, where you select the project type, name and location. For this tutorial, we'll use an ASP.NET Web Application

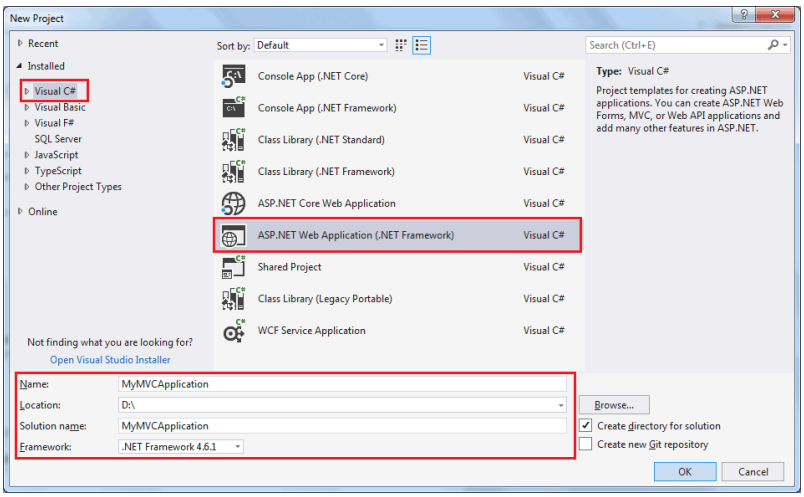

Figure 2.8 Create new project

From the **New ASP.NET Web Application** dialog, select MVC

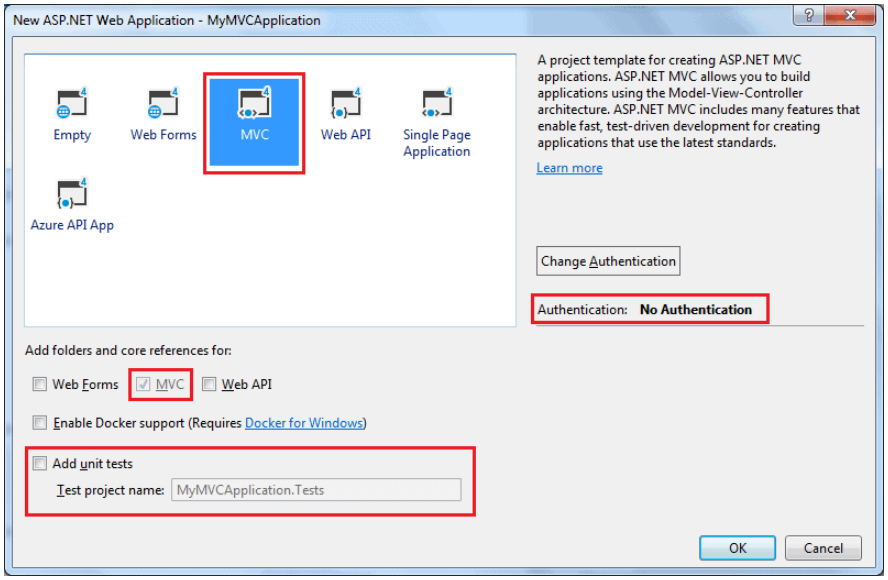

Figure 2.9 create MVC project

After some time, Visual Studio will create a simple MVC project using the default template, as shown below.

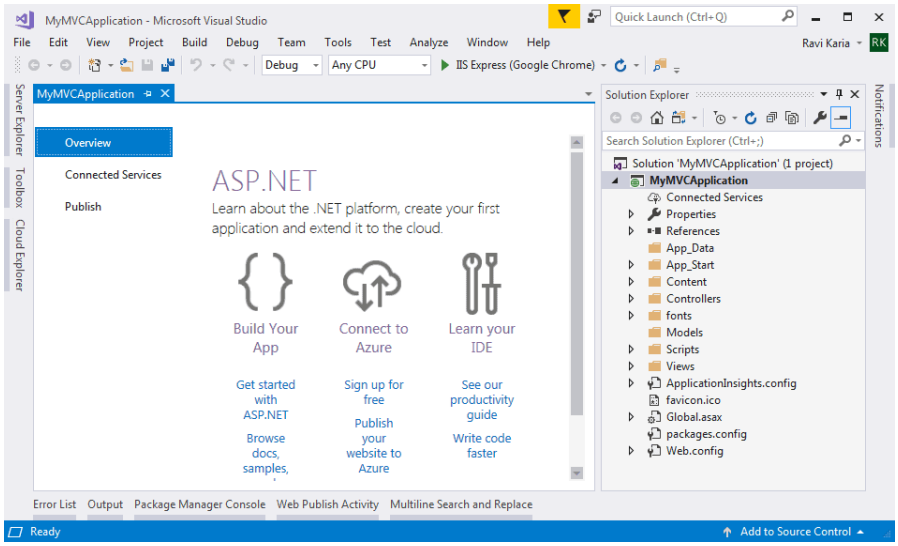

Figure 2.10 simple MVC project

As we talked briefly about earlier, the **Controller** acts as the middleman - it will combine your **Model** with a **View** and serve the result to the end-user. However, neither a Model nor a View is required - the Controller can act on its own for the most basic operations, e.g. delivering a simple text message or redirecting the user to somewhere else.

However, there are a few things we need to do before adding a new controller to our project.

2.4.1 Adding MVC support to a Web project

We need to add MVC support to it, to let the .NET framework and the web server know how to process incoming requests etc. to that let's open the **Startup.cs** file in Visual Studio and look for the **ConfigureServices** method. It's currently empty, but let's change that by adding the following line to it:

```
services.AddMvc();
```

The method should now look like this:

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddMvc();
}
```

We also need to modify the **Configure()** method to make our web application handle how to map incoming requests to your controllers so let's modify **Configure()** method in the **Startup.cs** file so that it looks like this:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   if(env.IsDevelopment())
   {
   app.UseDeveloperExceptionPage();

   app.UseRouting();
   app.UseEndpoints(endpoints =>
   {
   endpoints.MapDefaultControllerRoute();
   });
}
```

Now we're finally ready to add our first Controller!

2.4.2 Add a Controller

Add our very first Controller to the project. Just right-click the new folder and select Add -> **New Item**..., like this:

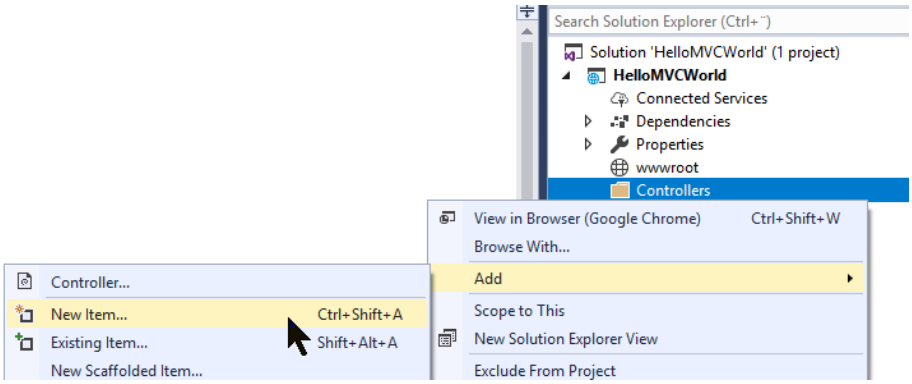

Figure 2.11 Add new controller

Visual Studio will offer to apply scaffolding to your new controller, which basically means that it can be created with a range of methods for doing various stuff. However, this tutorial is all about doing things from scratch, so you should select the one called something like "MVC Controller - Empty" and then click the Add button.

A new Controller will be generated for you and it will look like this:

```
namespace HelloMVCWorld.Controllers
{
  public class HomeController : Controller
 {
  public ActionResult Index()
  {
     return View();
  }
 }
}
```

This is how a default project is built. For our project, we need to modify these default codes, so we need two controllers one for home page and other for the Login page.

For the **Home Controller** at first, we include the library that we need for the work

The default library and extra for the model, remoting, database, etc.

```
using System.Web.Mvc;
using System.Net;
using ADPproject.Services.Remoting;
using ADPproject.Model;
using Newtonsoft.Json;
using IotLibrary;
```

and in the class, there is one **ActionResult** method.

ActionResult is a return type of a controller method, also called an action method, and serves as the base class for result classes. Action methods return models to views, file streams, redirect to other controllers, or whatever is necessary for the task at hand. The controller takes on this responsibility to connect system components, acting as a traffic cop. [11]

The Dashboard method that handles the view for the IOT web page.

```
public ActionResult Dashboard(sensor_v_m model)
    {
        readjson cust = remoting.Getsensor(model);
        string TmpData = cust.Data;
        Viv bewBag.tmp = TmpData;
      return View();
    }
```

We notice a new type of struct (readjson) it's called models and it is a class that contains parameters to transfer a specific type of information that the user creates within the project specifications.

The model classes represent domain-specific data and business logic in the MVC application. It represents the shape of the data as public properties and business logic as methods.

A controller can have one or more action methods, and each action method can return a different view, a view is used to display data using the model class object. The Views folder contains all the view files in the ASP.NET MVC application.

For the Dashboard view the code is included in the appendix due to its length but we will mention the sections related to the project, The code consists of interfaces written in several languages HTML5, CSS, JavaScript. This three-language use to make front-end framework called Bootstrap used to create modern websites and web apps.

The most important sections in the Dashboard view are the model part:

```
@model mvc_test.Models.sensor_v_m
@using (Html.BeginForm("remotingtest", "test"))
{
   @Html.TextBoxFor(m => m.Id)

   <input type="submit" value="Search" />
}
```

When we enter ID of the sensor in the textbox the ID pass through the model to the controller then to the web server over the remoting function `remoting.Getsensor` Which fetches the required data of the sensor

2.5 Remoting System Component

To build the remoting system we must have the following three structures

1- (.dll) shared library that contain the remote class interface and the mobile object

2- Remoting server which builds a communication channel on a specific port and determines the basic settings, which are the server activate object (SAO) and the client activate object (COA)

3- Client In which a channel is defined and registered, a proxy is built and used

Now let's talk about **remoting.Getsensor** struct, remoting is a class we create to connect the ASP.NET with the Web server to exchange the data, .NET Remoting was explain in the theoretical part.

```
public class remoting
   {
      static void Main(string[] args)
      {
         int i = 1;
         i++;
         HttpChannel chnl = new HttpChannel();
         ChannelServices.RegisterChannel(chnl, false);
         ICustomerManger prox =
(ICustomerManger)Activator.GetObject(typeof(ICustomerManger),
            "http://localhost:443/CustomerManger.soap");
      }
```

```
    }
```

In the **remoting** class in the clint side there is two section the void main, his mission creates remote object name **prox** and the Http channel to the server (http://localhost:443/CustomerManger.soap)

SOAP is a protocol to transport data in XML format from the client to the server and back.[11].

The second section is **Getsensor** that connects with the clint to fetch information from the **getDB** method in the server side who connect to the database and show it to the user in the view page

```
public static readjson Getsensor(sensor_v_m i)
    {
       readjson cust = new readjson();
       cust = prox.getDB (i.Id);
       return cust;
    }
```

The `getDB` methode:

```
public readjson getDB (int i)
        {
           mvctestEntities2 db = new mvctestEntities2();
           TEMP TEMP = new TEMP();
           readjson TempDATA = new readjson();
           TEMP = db.TEMPs.SingleOrDefault(x => x.Id == i);
           try
           {
              TempDATA.Data = TEMP.Data.ToString();
              TempDATA.Date = TEMP.Time;
           }
           catch(NullReferenceException)
           {
              Console.WriteLine("wait");
           }
           return TempDATA;
        }
```

In the server side the remote class inherited from the interface and inherit from MarshalByRefObjects.

MarshalByRefObject is the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy. [11]

And the void main to Initializing the connection, we use port 443 because the connection is secure uses HTTPS powered by Transport Layer Security (TLS) we'll talk about it in the other section. And the remote and we choose the SAO as a single tone that is, create a single bandwidth for all users

```
static void Main(string[] args)
      {
         Console.WriteLine("{0}:{1}:{2}:{3}", DateTime.Now.Hour.ToString(),
            DateTime.Now.Minute.ToString(), DateTime.Now.Second.ToString(),
            DateTime.Now.Millisecond.ToString());
         Console.WriteLine("Server.Main: Server Started");

         HttpChannel chnl = new HttpChannel(443);
         ChannelServices.RegisterChannel(chnl, false);

         Console.WriteLine("{0}:{1}:{2}:{3}", DateTime.Now.Hour.ToString(),
            DateTime.Now.Minute.ToString(), DateTime.Now.Second.ToString(),
            DateTime.Now.Millisecond.ToString());
         Console.WriteLine("Server.Main: Server is listenning to port 443");


RemotingConfiguration.RegisterWellKnownServiceType(typeof(CustomerManger),
"customerManger.soap", WellKnownObjectMode.SingleCall);


         Console.WriteLine("{0}:{1}:{2}:{3}", DateTime.Now.Hour.ToString(),
            DateTime.Now.Minute.ToString(), DateTime.Now.Second.ToString(),
            DateTime.Now.Millisecond.ToString());

         Console.WriteLine("Server.Main: Server is ready to be used");

         API_to_DB.InsertData();
         Console.ReadKey();
      }
```

When we the project start the next screen shows that the server is running waiting for a request from the clint

```
12:37:51:207
Server.Main: Server Started
12:37:51:360
Server.Main: Server is listenning to port 443
12:37:51:361
Server.Main: Server is ready to be used
```

Figure 2.12 Server Start Screen

The clint call for request, the server create a remote object contains the data from the Database and send it to the clint

```
12:48:40:955
CustomerManger.Constructor: Object Created
wait
```

Figure 2.13 Server Screen

The communication between the clint and server it is done through a library called **IoTLibrary.dll** that we have added to the project, in this library we need to define the remote class interface that we use which is used by the server as inheritance and used by the client to build the activator

activator class is containing methods to create types of objects locally or remotely, or obtain references to existing remote objects. [11]

```
public interface ICustomerManger
    {
        readjson getDB(int i);
    }
```

In the library we add the mobile object called readjson but with Serializable attribute so we can carry data between server and clint and the getDB method from the clint server

```
[Serializable]
    public class readjson
    {
        public int sensor;
        public string Data;
        public string Time;
        public string Date;
    }
```

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file.

2.6 Creating a View

You can create a view for an action method directly from it by right clicking inside an action method and select Add View...

The following creates a view from the **Dashboerd()** action method of the Home controller, as shown below.

Figure 2.14 Add View

This will open the Add View dialogue box. It's good practice to keep the view name the same as the action method name so that you don't have to explicitly specify the view name in the action method while returning the view.

Select the scaffolding template. Template dropdown will show default templates available for Create, Delete, Details, Edit, List, or Empty view. Select "Empty" template because we want to build our own interface.

After building our interface, the final look is shown below

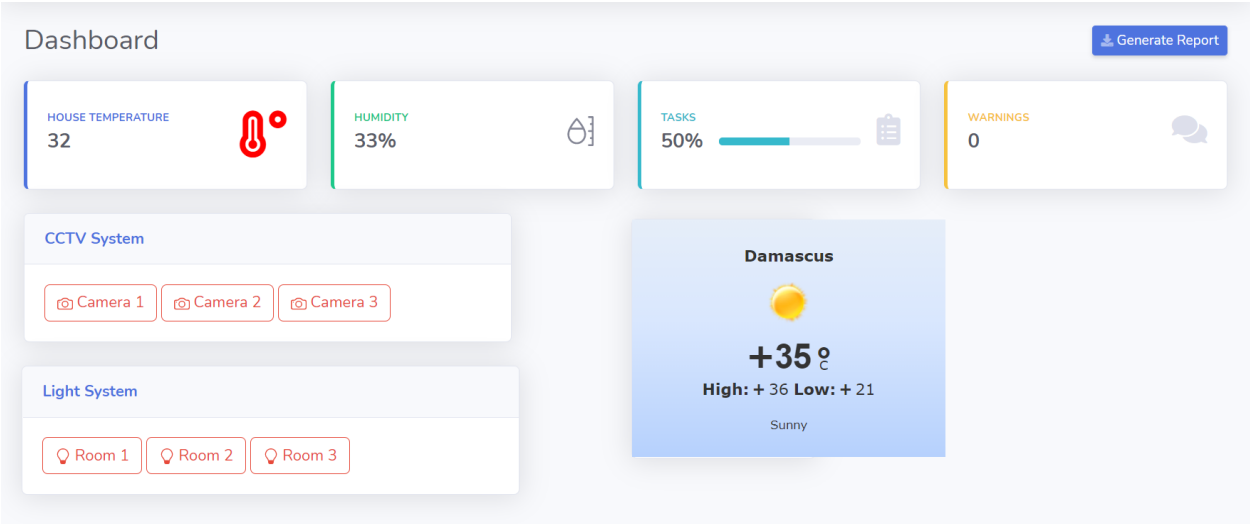

Figure 2.15 Dashboard Page

It shows the house temperature and humidity, the lighting control buttons, the CCTV system buttons, the weather forecast, and other features, including household tasks, a generate report page and various warning

2.7 WCF Architecture

There are four major layers that provide developers with a new service-oriented programming model. The WCF architecture consists of the four layers explained in the theoretical section

Contracts (Layer 1): The contract layer contains various types of contracts and policy and binding used in WCF. The various types of contracts present in the contract layer are as follows:

Service Contract: Service contract includes the operations performed by the service and exposes them as a single unit through an interface. [12]

```
namespace WcfServiceLibrary
{
      [ServiceContract]
}
```

Data Contract: Exposes the user defined data types and classes in a WCF service. [12]

```
[DataContract]
   public class CompositeType
   {
      [DataMember]
      public int Id { get; set; }
      [DataMember]
      public double Data { get; set; }
      [DataMember]
      public string Data { get; set; }
      [DataMember]
      public string Time { get; set; }
   }
}
```

Operation Contract: Operation contract is used to expose the operations that a service can perform. It defines the methods of a WCF service along with the parameters and return type of the methods it call data service. [12]

```
[ServiceContract]
   public interface IDataService
   {
      [OperationContract]
      sensor_activity GetData(int custID);
      [OperationContract]
      List<sensor_activity> GetDataL();
      [OperationContract]
      string InsertData(sensor_activity cust);
```

```
    [OperationContract]
    void UpdateData(sensor_activity cust);
    [OperationContract]
    void DeleteData(int custID);
}
```

Service Runtime (Layer 2): The service runtime layer contains the behavior of the service that occurs during the execution of the service. [12]

```
public class DataService : IDataService
{
    public void DeleteData(int custID) {}
    public sensor_activity GetData(int custID) {}
    public List<sensor_activity> GetDataL() {}
    public string InsertData(sensor_activity cust) {}
    public void UpdateData(sensor_activity cust1) {}
}
```

The full code will be included in the appendix

Messaging (Layer 3): Using the channels, the messaging layer processes the message that is processed and transported to a client accessing the service. [12]

Activation and Hosting (Layer 4): This layer supports the execution of services in various environments, such as Windows Services, IIS and Windows Activation Services (WAS). A service can either be self-hosted or hosted in the context of another application and we use the IIS for hosting the WCF server. [12]

IIS: When a WCF service is hosted in IIS the client can access the service over the internet. When a service is hosted in IIS, it acquires the benefits of IIS such as process lifetime management and automatic update after configuration changes. [12]

To add website to the IIS

open IIS on your system. Or you can directly open IIS by typing **inetmgr** in run window like below.

This will open a popup to input new website details. Input the following details in pop-up box.

- **Site name:** Name of website to be appeared in IIS listing.
- **Application pool:** Select an application pool or keep is the default to create new application pool same name as site name.
- **Physical path:** Enter the location of website pages on system.
- **Binding:**
- **Type:** Select protocol to configure (eg: http or https)
- **IP address:** Select ip address from drop list to set dedicated ip for site or keep is the default to use shared Ip.
- **Port:** Enter port on which site will be accessible for users.
- **Host name:** Enter you actual domain name you want to use.
- **Start Website immediately:** keep this box checked to start site.

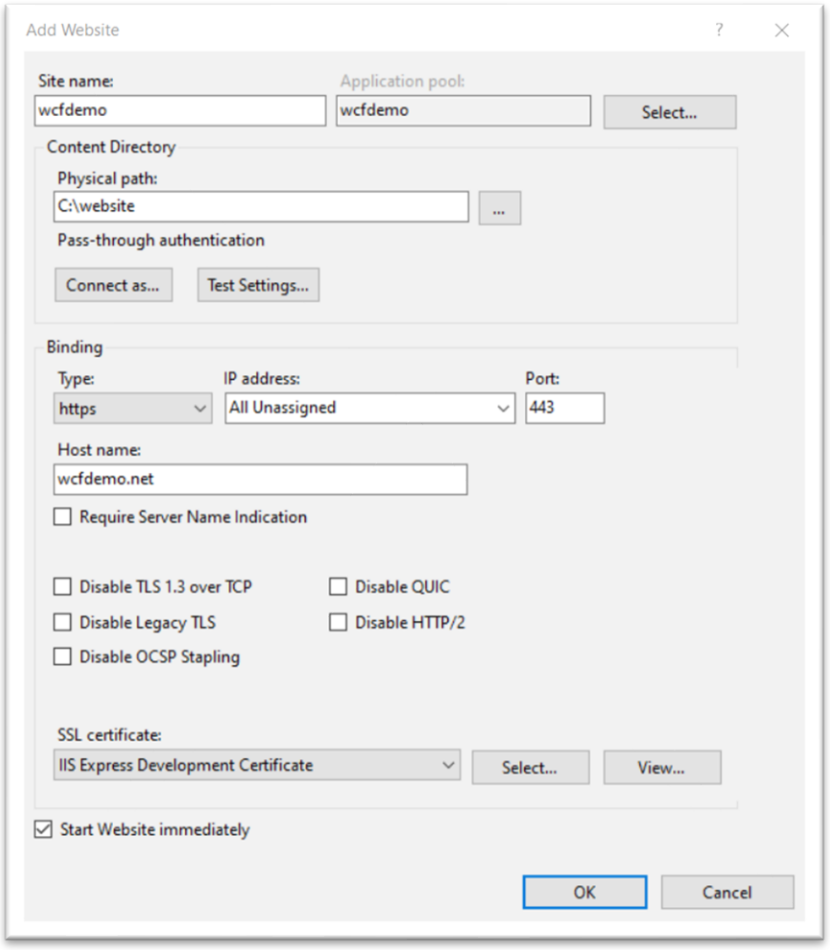

Figure 2.16 Add Website Page

To verify configuration, you can simply access the site in a web browser.

Figure 2.17 WCFdemo Website

Next, we must run visual studio and publish the WCF on the IIS site

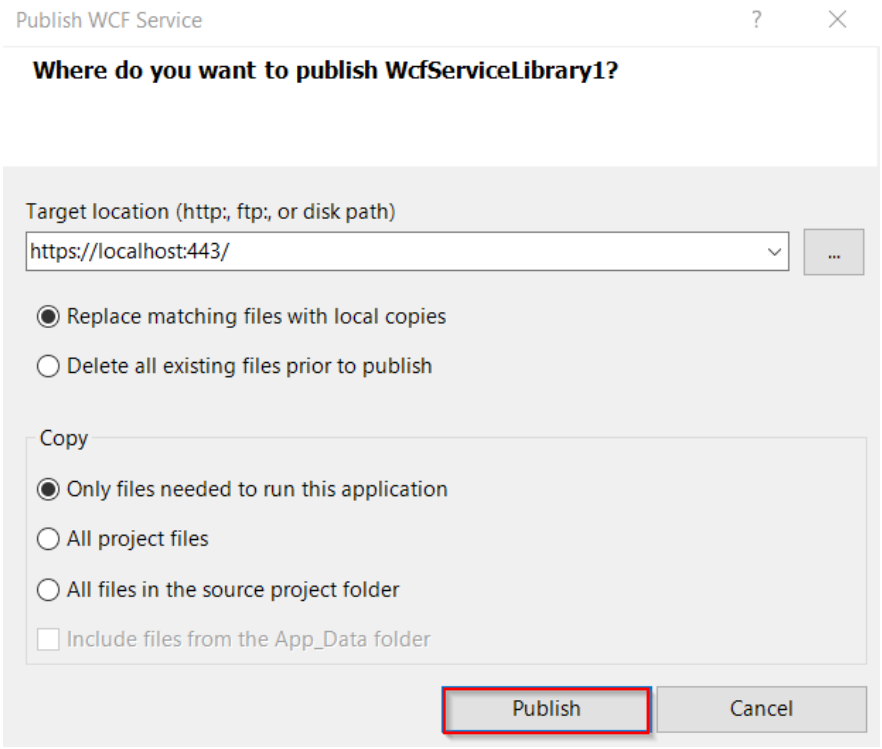

Figure 2.18 Publish WCF Service Page

After we publish the WCF service we can add our services to the web server

as a service reference from the project solution we choose add service reference and the following tap will show

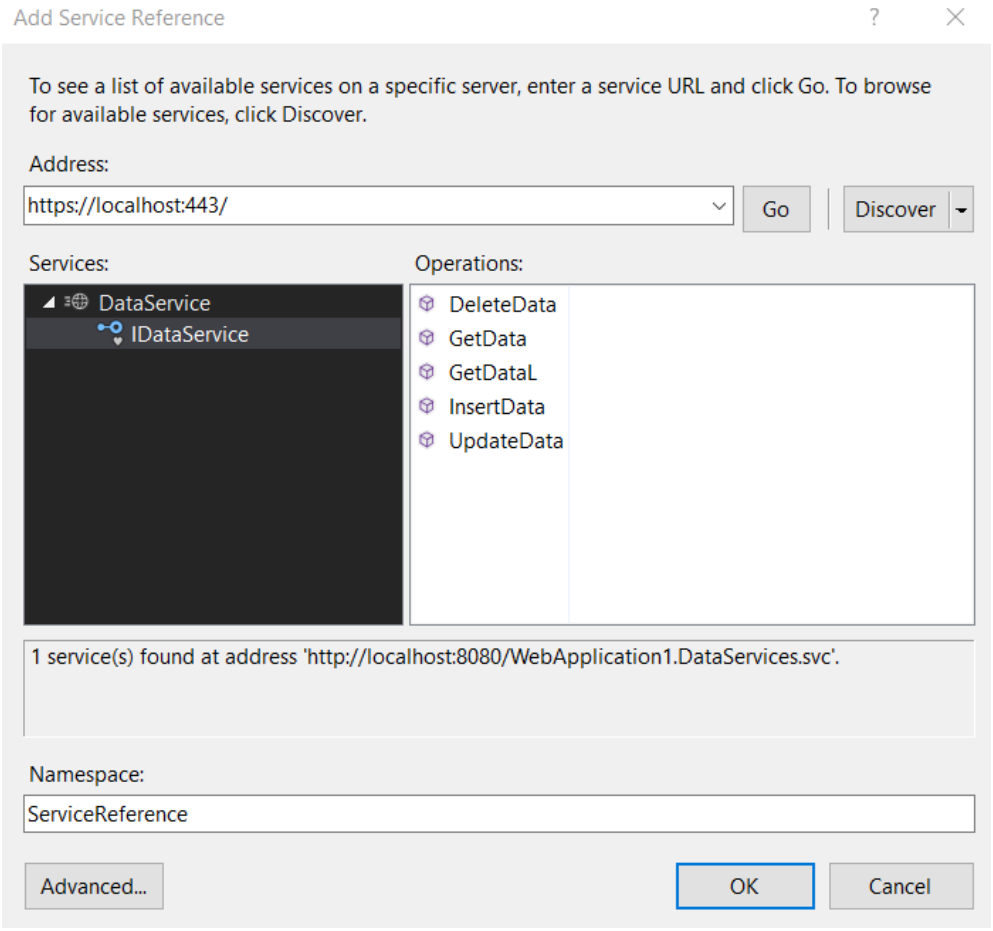

Figure 2.19 Service Reference

Press ok this will result in a proxy class created in the web server

2.8 Create a Database

SQL Server databases are some of the most common databases in use, to create a database and start entering your information first we need to **Install the SQL Server Management Studio software** than **Start up SQL Server Management Studio.** If you already have a server up and running, and have the permissions necessary to connect to it, you can enter the server address and authentication information. If you want to create a local database, set the Database Name to . and the authentication type to "Windows Authentication".

Next, we need to **Locate the Database folder** after the connection to the server, either local or remote, is made, the Object Explorer window will open on the left side of the screen. At the top of the Object Explorer tree will be the server you are connected to. if it is not expanded, click the "+" icon next to it. Located the Databases folder.

Create a new database. Right-click on the Databases folder and select "New Database...". A window will appear, allowing you to configure the database before creating it.

Create a table. A database can only store data if you create a structure for that data. A table holds the information that you enter into your database, and you will need to create it before you can proceed. Expand the new database in your Database folder, and right-click on the Tables folder and select "New Table...".

Windows will open on the rest of the screen which will allow you to manipulate your new table.

The database for the project simple we need two table one for the sensors data and other for the users, in the sensors data it is highly recommended that you create a Primary Key as the first column on your table. This acts as an ID number, or record number, that will allow you to easily recall these entries later. In the sensors data table there will be five field (ID, Data, Time, Date, User ID) for each sensor, for every user has a ID and User ID to fetch the correct information of each user, time and date to record the timestamp of an event and data field so the database diagram will be in the following (figure 2.20)

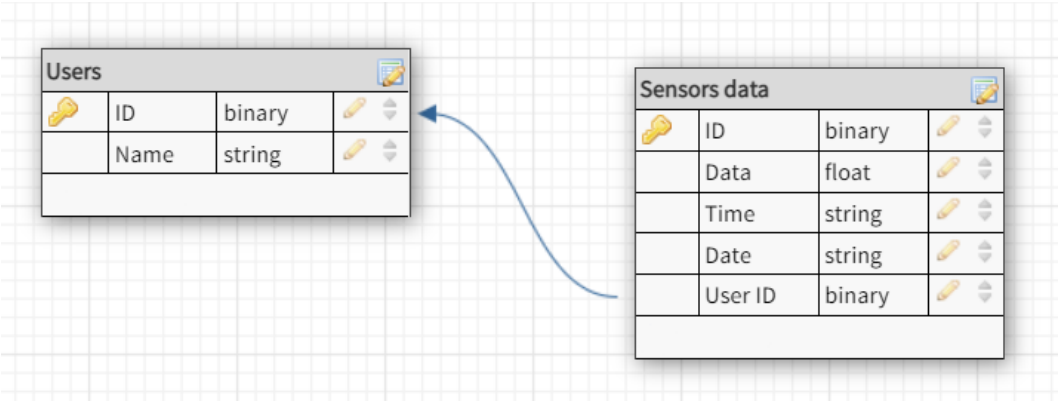

Figure 2.2 Database Diagram

as notes in the figure there is two primary keys for the two table and one foreign key, a foreign key is **a column or group of columns in a relational database table** that provides a link between data in two tables So we can connect each data to their user.

The process of recalling the information from the database will be done through a method called Database_Entities() and it is added in several steps

1- add new item to the project called **ADO.NET entity** we will choose **EF designer** from database the after name it and press next the following tap will show

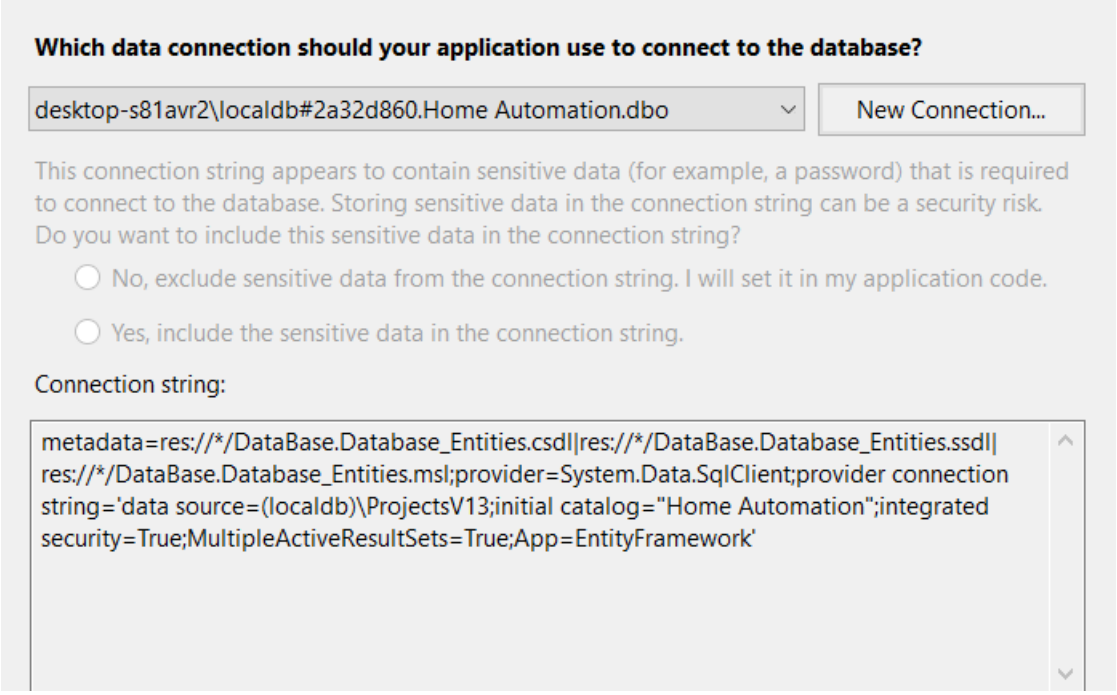

Figure 2.21 Database Connection string

This tap Contains connection string and database server

2- the next tap for choosing the tables from the database for the project

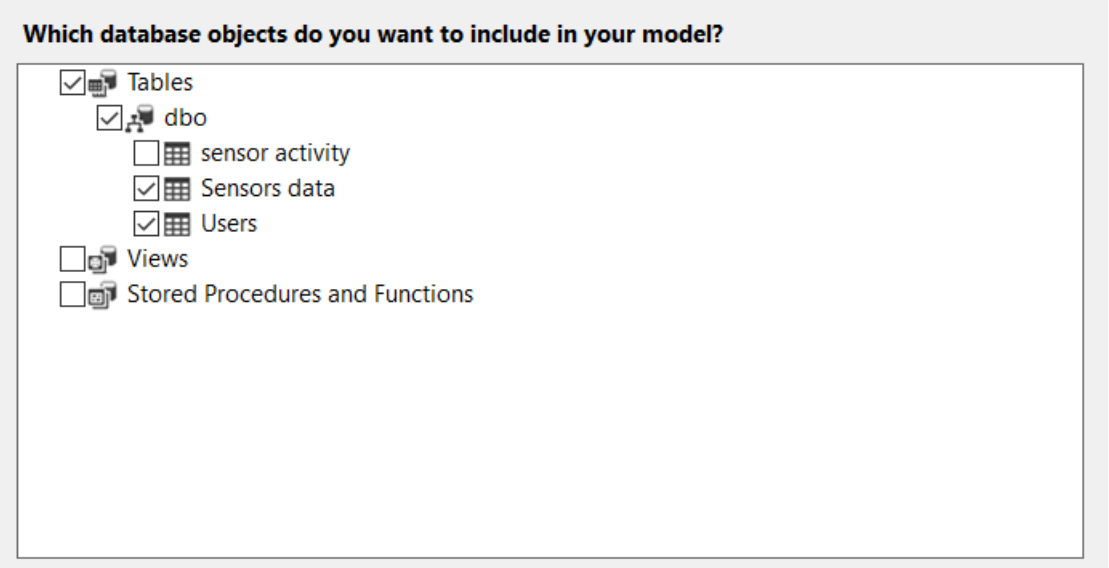

Figure 3 Database Tables

3- choose out the tables then press finish and the database will appear as follows

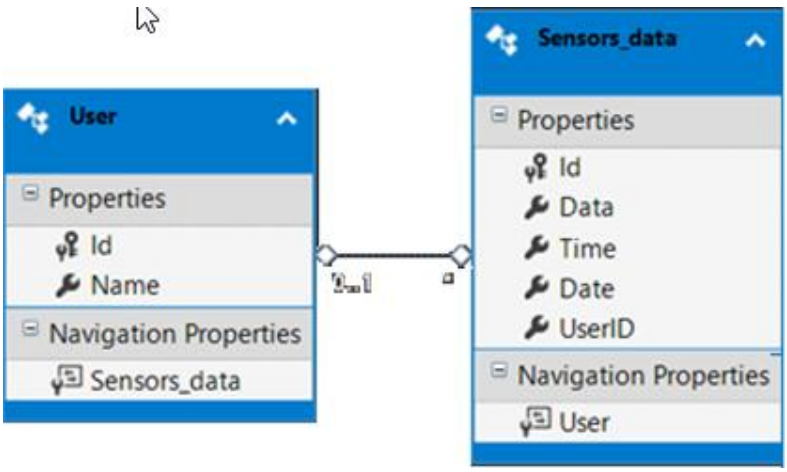

Figure 2.4 The Database

The information is called from the database using the following code:

```
var ctx = new Database_Entities()
var data = (from cust in ctx.sensor_data
                    where cust.Id == custID
                    select cust).SingleOrDefault();
          return data;
```

2.9 Login Page

In order to have a page dedicated to each user through which he controls the different applications, we need to have a username and password for each user to be entered through a page to connect to his dashboard panel therefore, a special controller was created for this page and a ActionResult method name login

```
public ActionResult Login(UserModel userModel)
        {
            SecurityService securityService = new SecurityService();
            Boolean success = securityService.Authenticate(userModel);
            if (success)
            {
                return RedirectToAction("Dashboard", "Home");
            }
            else
            {
                return View();
            }
        }
```

In this method we have if statement prevent any user from entering the dashboard page without a password and username through the method securityService.Authenticate(); that connect to the database of users and find the name and password than compare them

Login page view

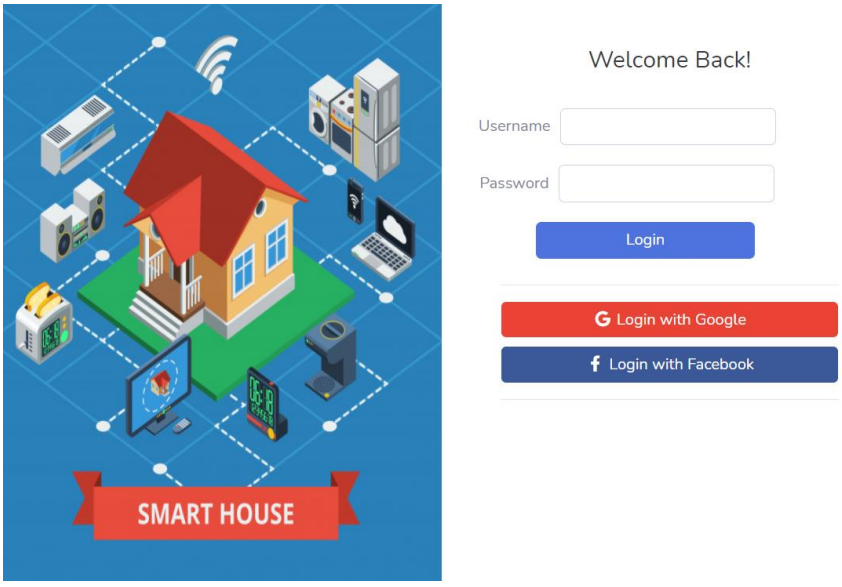

Figure 2.24 Login Page View

Chapter Three

Secure ASP.NET

3.1 Introduction

Security is a broad topic. Research has shown that early design of authentication and authorization eliminates a high percentage of application vulnerabilities. Secure communication is an integral part of securing your distributed application to protect sensitive data, including credentials, passed to and from your application, and between application tiers.

There are many technologies used to build .NET Web applications. To build effective application-level authentication and authorization strategies, you need to understand how to fine-tune the various security features within each product and technology area, and how to make them work together to provide an effective, defense-in-depth security strategy. [13].

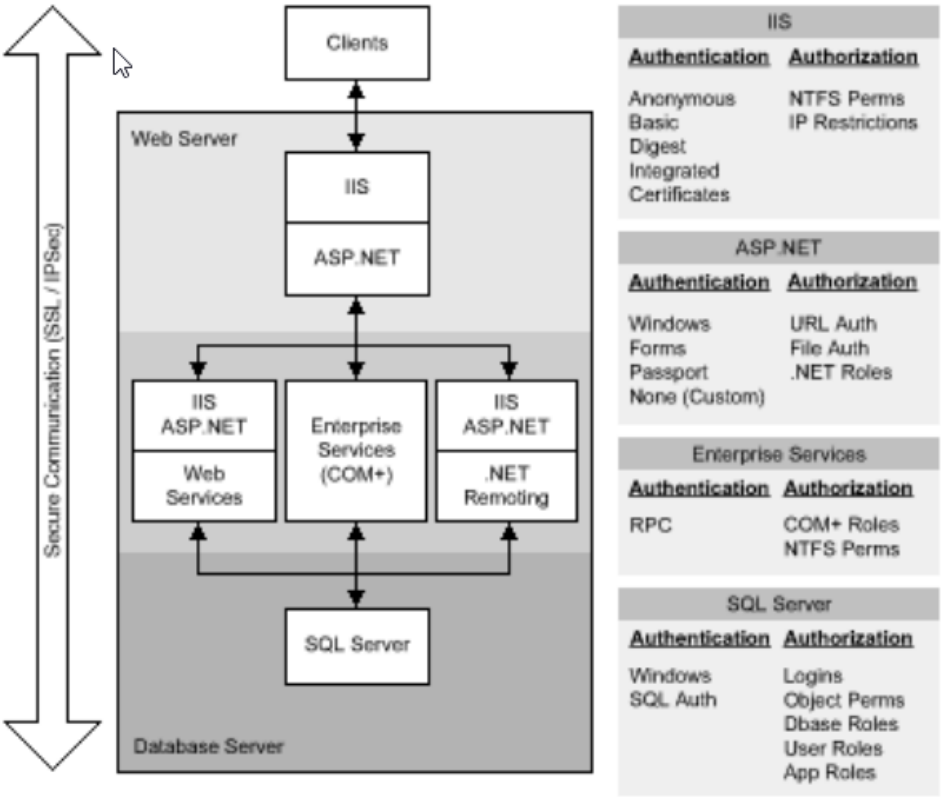

Figure 3.1 Various technologies to secure ASP.NET

3.2 Designing an Authentication and Authorization Strategy

Designing an authentication and authorization strategy for distributed Web applications is a challenging task. The good news is that proper authentication and authorization design during the early phases of your application development helps to mitigate many top security risks. [13].

The following steps identify a process that will help you develop an authentication and authorization strategy for your application:

1. Identify resources
2. Choose an Authorization Strategy
3. Choose the identities used for resource access

Identify Resources

Identify resources that your application needs to expose to clients. Typical resources include:

- Web Server resources such as Web pages, Web services, static resources (HTML pages and images).
- Database resources such as per-user data or application-wide data.
- Network resources such as remote file system resources and data from directory stores such as Active Directory. [13].

Choose an Authorization Strategy

The two basic authorization strategies are:

- **Role based**. Access to operations (typically methods) is secured based on the role membership of the caller. Roles are used to partition your application's user base into sets of users that share the same security privileges within the application; for example, Senior Managers, Managers and Employees.
- **Resource based**. Individual resources are secured using Windows ACLs. [13].

Choose the Identities Used for Resource Access

Choose the identity or identities that should be used to access resources across the layers of your application. This includes resources accessed from Web-based applications, and optionally Web services, Enterprise Services and .NET Remoting components.

For the project we use SecurityDAO and SecurityService method

```
public class SecurityDAO
   {
      public bool FindByUser(UserModel user)
      {
         var ctx = new Database_Entities();
         if (user.Username == ctx.users.username && user.Password ==   ctx.users.password)
         {
            return true;
         }
         else
         {
            return false;
         }
      }
   }
}
```

This method validates the password and username from the database

```
public class SecurityService
   {
      SecurityDAO daoService = new SecurityDAO();
      public bool Authenticate(UserModel user)
      {
         return daoService.FindByUser(user);
      }
   }
```

Here, it verifies that the information matches the user's information In the event of a match the page redirect from the login page to the dashboard page

3.3 Secure Communication

Many applications pass security sensitive data across networks to and from end users and between intermediate application nodes. Sensitive data might include credentials used for authentication, or data such as credit card numbers or bank transaction details. To guard against unwanted information disclosure and to protect the data from unauthorized modification while in transit, the channel between communication end points must be secured.

Secure communication provides the following two features:

Privacy. Privacy is concerned with ensuring that data remains private and confidential, and cannot be viewed by eavesdroppers who may be armed with network monitoring software. Privacy is usually provided by means of encryption.

Integrity. Secure communication channels must also ensure that data is protected from accidental or deliberate (malicious) modification while in transit. Integrity is usually provided by using Message Authentication Codes (MACs). [13].

In this typical deployment model, a request passes through three distinct channels. The client-to-Web server link may be over the Internet or corporate intranet and typically uses HTTP. The remaining two links are between internal servers within your corporate domain. Nonetheless, all three links represent potential security concerns. Many purely intranet-based applications convey security sensitive data between tiers; for example, HR and payroll applications that deal with sensitive employee data.

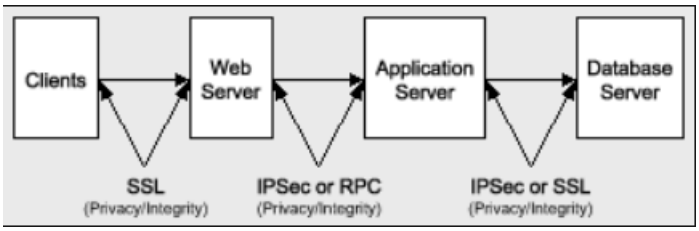

Figure 3.2 shows how each channel can be secured by using a combination of SSL, IPsec and RPC encryption. [13].

The choice of technology depends on a number of factors including the transport protocol, end point technologies, and environmental considerations (such as hardware, operating system versions, firewalls, and so on).

3.3.1 Browser to Web Server

To secure sensitive data sent between a browser and Web server, use SSL/TLS is used to establish an encrypted communication channel between client and server.

in the following situations:

You are using Forms authentication and need to secure the clear text credentials submitted to a Web server from a logon form.

You are using Forms authentication and need to secure the clear text credentials submitted to a Web server from a logon form.

Your application passes sensitive data between the browser and Web server (and vice-versa); for example, credit card numbers or bank account details. [13].

Enabling and Setting Up the SSL

1. In the Solution Explorer, select the application and press the F4 key.
2. Enable the SSL Enabled option and copy the SSL URL.

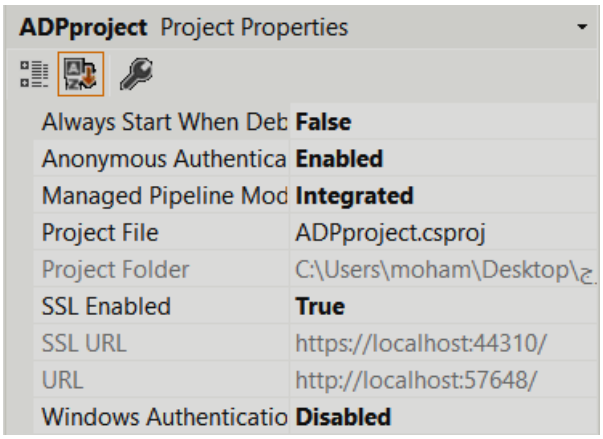

Figure 3.3 Enabling SSL

Now just right-click on the application and select the Properties option.

Select the Web tab from the left pane, and paste the SSL URL into the Project URL box.

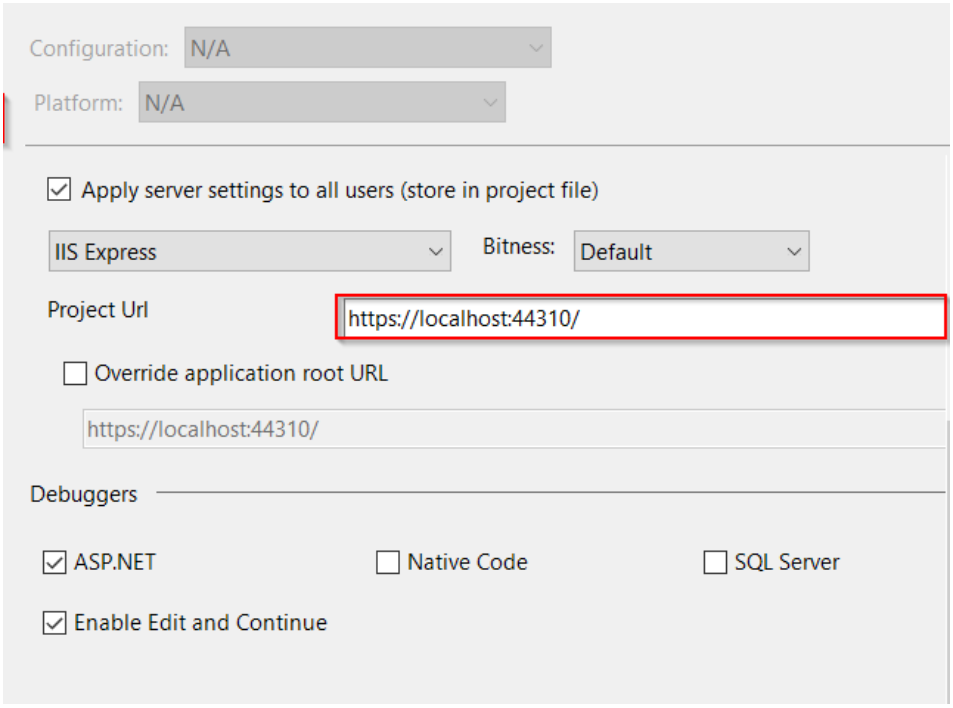

Figure 3.4 Project Url

Select the HomeController.cs from the Controllers folder and add the following highlighted code to edit:

```
namespace WebApplication1.Controllers
{
    [RequireHttps]
    public class HomeController : Controller
    {
        public ActionResult Dashboard()
        {
            return View();
        }
}
```

Now press Ctrl+F5 to run the application and follow the instructions to trust the self-signed certificate generated by IIS Express.

After clicking on Yes, the Security Warning wizard opens and click Yes to install the certificate representing the localhost.

Now, when you run the application using Internet Explorer (IE), it shows the Home Page of the application and there is no SSL warning.

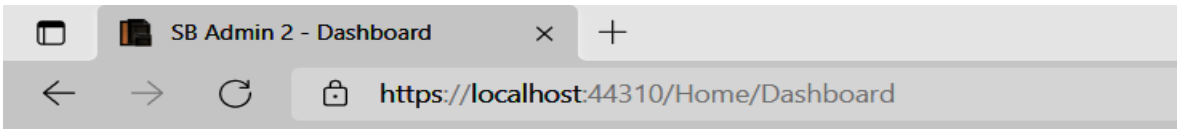

Figure 3.5 Project Home page

3.3.2 Web Server to Remote Application Server

The transport channel between a Web server and a remote application server should be secured by using IPsec, SSL or RPC Encryption. The choice depends on the transport protocols, environmental factors (operating system versions, firewalls and so on).

Enterprise Services. If your remote server hosts one or more serviced components (in an Enterprise Services server application) and you are communicating directly with them (and as a result using DCOM), use RPC Packet Privacy encryption.

Web Services. If your remote server hosts a Web service, you can choose between IPSec and SSL.

You should generally use SSL because the Web service already uses the HTTP transport. SSL also allows you to only encrypt the data sent to and from the Web service (and not all traffic sent between the two computers). IPSec results in the encryption of all traffic sent between the two computers.

.NET Components (using .NET Remoting). If your remote server hosts one or more .NET components and you connect to them over the TCP channel, you can use IPSec to provide a secure communication link. If you host the .NET components within ASP.NET, you can use SSL (configured using IIS). [13].

Enabling Security Exception

For the server and clint side we have to change the code for the register channel

```
HttpChannel chnl = new HttpChannel(443);
        ChannelServices.RegisterChannel(chnl, bool ensureSecurity);
```

If the ensureSecurity parameter is set to true, the remoting system determines whether the channel implements ISecurableChannel, and if so, enables encryption and digital signatures. An exception is thrown if the channel does not implement ISecurableChannel.

3.3.3 Application Server to Database Server

To secure the data sent between an application server and database server, you can use SSL. Or Windows authentication. One of the key benefits of using Windows authentication to SQL Server is that it means that the credentials are never passed across the network.[13].

Enabling SQL Server Authentication through SQL Management Studio

To enable SQL Server Authentication

Open SQL Server Management Studio. And connect to the SQL Server.

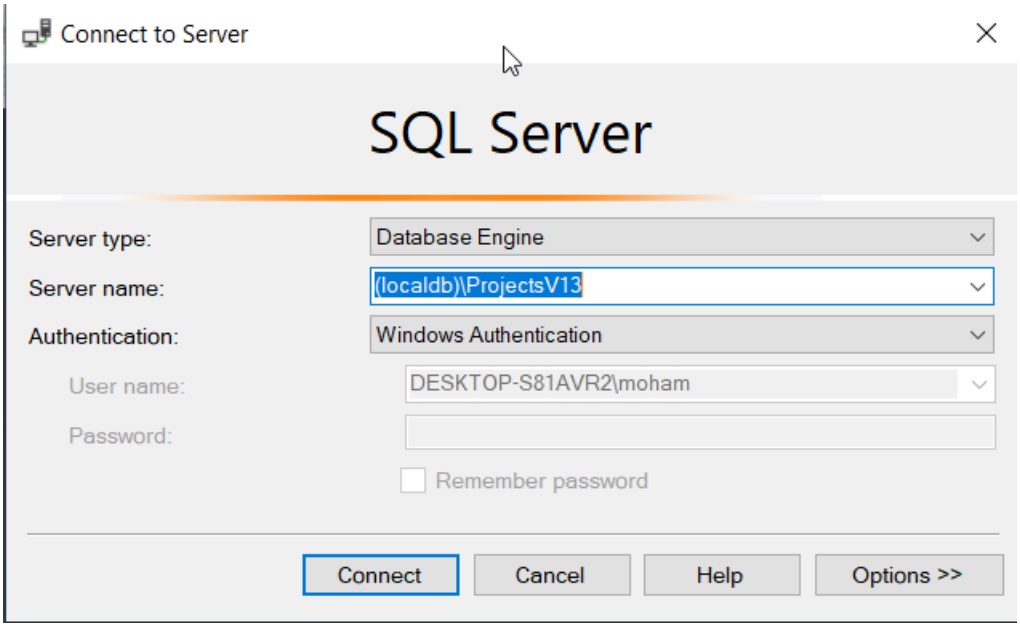

Figure 3.6 SQL Server Management Studio

In the Object Explorer, right-click the server and click Properties.

On the Security page under Server authentication, select SQL Server and Windows Authentication mode and then click OK.

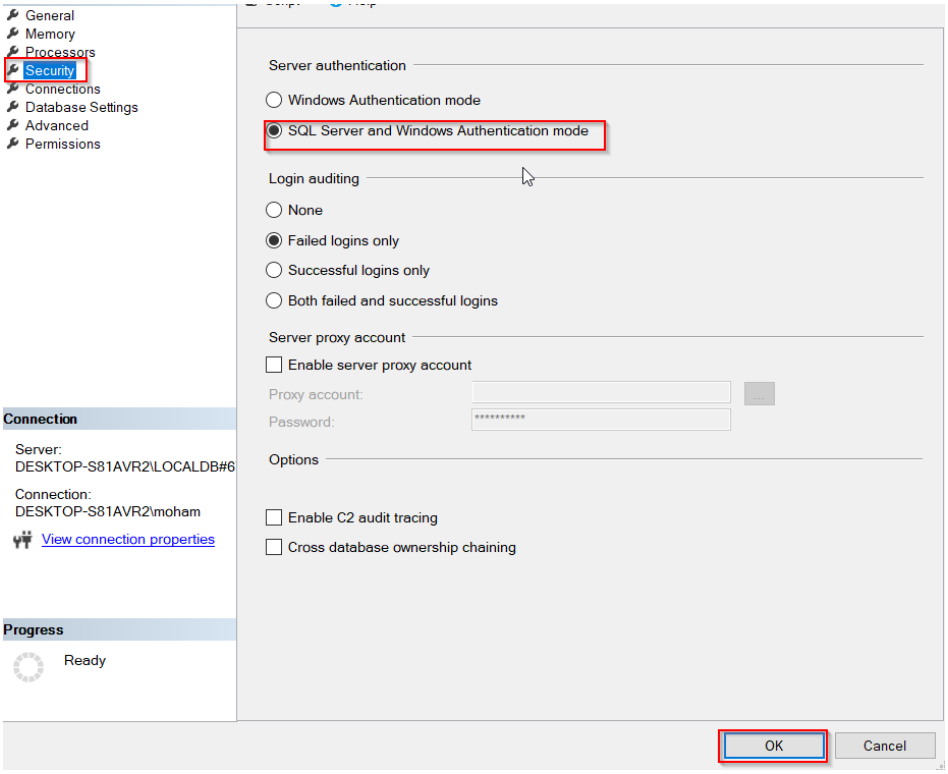

Figure 3.7 Enable windows Authentication

And Windows authenticate now active.

3.4 HTTPS Server on the ESP8266 NodeMCU

To offer secured content, a server greets the client with a trusted certificate, issued by a known authority. The certificate has a limited time validity and must be renewed from time to time. we'll generate a SSL certificate and use it on ESP8266 web server. [13].

Certificate and key

For ESP8266 compatibility, the certificate must use SHA256 and the key length must be either 512 or 1024 bits. A 512 bits RSA key will make ESP8266 respond faster, but it is considered weak by modern browsers. For better security, use 1024 bits RSA key. The trusted CA should give you both the certificate and the private RSA key. We intend to use OpenSSL to generate the certificate and the private RSA key. Getting OpenSSL on Linux is easy since most distributions already have it installed and you can find it in software repositories

Launch openssl on the command line, from the folder where you want certificate and key to be generated. It is possible to generate both key and certificate using a single command:

```
req -x509 -newkey rsa:1024 -sha256 -keyout key.txt -out cert.txt -days 365 -nodes -subj "/C=RO/ST=B/L=Bucharest/O=OneTransistor [RO]/OU=OneTransistor/CN=esp8266.local" -addext subjectAltName=DNS:esp8266.local
```

the rsa:1024 specifies key length in bits, while in the second approach, last argument of genrsa is used for this. The -days parameter specifies certificate validity starting from the generation time. [13].

the parameters in the key.txt file and the certificate in cert.txt.

- **C** - country, short name
- **ST** - state or province
- **L** - locality or city
- **O** - organization
- **OU** - organizational unit
- **CN** - common name (domain name)

The subjectAltName parameter must contain the domain name(s) where your server is accessible. It can specify also IP addresses like this: subjectAltName=DNS:esp8266.local,IP:192.168.1.184.

The Web Server

The key and certificate must be pasted into the sketch:

```
namespace BearSSL {

using ESP8266WebServerSecure = esp8266webserver::ESP8266WebServerTemplate<WiFiServerSecure>;

static const char serverCert[] PROGMEM = R"EOF(

-----BEGIN CERTIFICATE-----

-----END CERTIFICATE-----
```

```
)EOF";

static const char serverKey[] PROGMEM =  R"EOF(

-----BEGIN PRIVATE KEY-----

-----END PRIVATE KEY-----

)EOF";
```

The certificate must be imported into the web server system.

Chapter Four

Tests and Results

4.1 Introduction

Centralized systems may have helped build the internet, but they have important disadvantages. That's what decentralized and distributed systems try to address.

4.2 The Importance of Different Systems

The centralized vs decentralized vs distributed systems debate is relevant to both individuals and organizations. It affects almost everyone who uses the web. It's at the core of the development and evolution of networks, financial systems, companies, apps, web services, and more.

While all these systems can function effectively, some are more stable and secure than others by design. Systems can be very small, interconnecting only a few devices and a handful of users. Or they can be immense and span countries and continents. Either way, they face the same challenges: fault tolerance, maintenance costs, and scalability.[14].

4.3 Distributed Systems and Internet of Things

The topic of "Distributed Systems and Internet of Things" represents a vision in which the Internet extends into the real world embracing everyday objects. Physical items are no longer disconnected from the virtual world, but can be remotely controlled and can act as physical access points to Internet services. The back end of these physical access points is a geographically distributed system which integrates such diverse technologies as pervasive, mobile and Cloud computing. This topic arises from synergically merging IoT and distributed computing.[14].

4.4 Why use .NET?

A few benefits of using .NET and WCF Web API instead of other web services frameworks. using .NET and WCF for the web services framework can ease many pains, such as having to create the clients and authorizing the clients that can communicate.

The following table describes the major features of each technology.

Table 4.1 features of ASP.NET and WCF.[15].

WCF	ASP.NET
Enables building services that support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them.	HTTP only. First-class programming model for HTTP. More suitable for access from various browsers, mobile devices etc enabling wide reach.
Enables building services that support multiple encodings (Text, MTOM, and Binary) of the same message type and allows switching between them.	Enables building Web APIs that support wide variety of media types including XML, JSON etc.
Supports building services with WS-* standards like Reliable Messaging, Transactions, Message Security.	Uses basic protocol and formats such as HTTP, WebSockets, SSL, JSON, and XML. There is no support for higher level protocols such as Reliable Messaging or Transactions.
Supports Request-Reply, One Way, and Duplex message exchange patterns.	HTTP is request/response but additional patterns can be supported through SignalR and WebSockets integration.
WCF SOAP services can be described in WSDL allowing automated tools to	There is a variety of ways to describe a Web API ranging from auto-generated HTML help

generate client proxies even for services with complex schemas.	page describing snippets to structured metadata for OData integrated APIs.

4.5 Performance

The performance of WCF and ASP.NET is good. Neither framework causes any problems in this case. Let's consider a simple API call.

```
public string Get() {
   // Return the list of the data
   return dataList;
 }
```

Performance depends on certain factors:

- CPU and RAM resources available for the programs to run; ASP.NET Web API can be hosted in a program.
- Network bandwidth and throughput.
- Other services running in the background.
- Time required by data engines to return the data, or to load the data from the memory or files.

The rest of the stuff is performed by the ASP.NET Web API to serialize the data to JSON format and send the data over the network. Most of the times, serialization and deserialization of data can also take time; Luckily, ASP.NET Web API uses the Newtonsoft.Json API to perform data deserialization and serialization actions and which are notably fast.

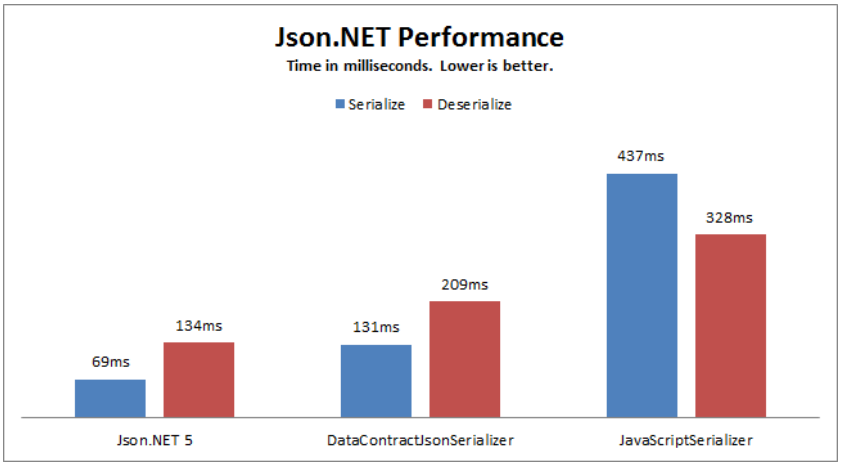

Figure 4.1 Json.NET Performance. [15].

Even the entry level and outgoing level doors of ASP.NET Web API are fine tuned to not give any performance factor a hard time. This is what makes ASP.NET Web API a very powerful framework to create web services on the top of HTTP protocol. [15].

4.6 Conclusion

In this work, the up-to-date web technologies were utilized to render the whole home automation system a distributed type with the processes as services. The cloud portion of the distributed system involves the web applications integrated with data management and repositories as well as communication interfaces. We induced great flexibility in the automation operations through HTML5 based web applications and services development for intuitive GUI web applications. Similarly, modular design concept was adopted in the embedded hardware development for better functionality and greater reliability. A robust data communication protocol to ensure seamless communication between the individual applications and systems was deployed. Relatively, a high level of security by the virtue of the robust web service security protocol deployed was realized. Overall, the system provides a cost-effective solution to home automation as the costs of a dedicated public IP address and a high-end computer, as present mostly in other solutions, are removed.

4.7 Future Work

We intend to further improve the performance by incorporating a higher layer communication protocol, Message Queuing Telemetry Transport (MQTT), an extremely simple and lightweight machine-to-machine, messaging protocol built on WebSocket for constrained devices and low-bandwidth, high-latency or unreliable networks

Reference:

[1]-Higinio Mora, &María Teresa, &David Gil 2018, Collaborative Working Architecture for IoT-Based Applications', Department of Computer Science Technology and Computation, University of Alicante,

[2]-Kuang-Chi Kao. 2018. Design and development of an IoT-based web application for an intelligent remote SCADA system in of Electrical and Electronic Engineering, Hsinchu, Taiwan.

[3]-Richard, M. A. & others (2005). DISTRIBUTED SYSTEMS ARCHITECTURE
A Middleware Approach. San Francisco. [accessed 29 March 2021]

[4]-Mario Szpuszta, & Ingo Rammer. Advanced .NET Remoting, Second Edition, Retrieved from https://vdocuments.site/advanced-net-remoting.html.

[5]-Bert Vanhooff, & Davy Preuveneers, Yolande Berbers. (2006). '.NET Remoting and Web Services:' A Lightweight Bridge between the .NET Compact and Full Framework. Department of Computer Science, Belgium. [accessed 17 March 2021]

[6]-Tutorials Point. (2018). LEARN WCF Windows Communication Foundation. Tutorials Point (I) Pvt. Ltd.

[7]-Shad Sluiter (2020). What is WCF Windows Communication Foundation. Available at: https://youtu.be/azfA_n_2wXo (Accessed: 16 March 2021).

[8]-Tutorials Point 2020, India, accessed 20 March 2021, <https://www.tutorialspoint.com/internet_of_things.htm>.

[9]-Andrea Finardi (4 June 2018). IoT Simulations with Cisco Packet Tracer

[10]-Dlnya Abdulahad Aziz 2019, ‘Design of Smart House System based on Webserver Architecture Control, accessed 17 July 2021.

[11]-C# Corner, Ksasikumar, accessed 15 Jun 2021, < https://www.c-sharpcorner.com/article/net-remoting/>

[12]-Gray, L 2018, 'Advances in Computer Science Research', International Symposium on Communication Engineering & Computer Science, volume 86, no. 2.

[13]-One Transistor, Cornelius, accessed 20 July 2021, < https://www.onetransistor.eu/2019/04/https-server-on-esp8266-nodemcu.html>

[14]-Berty, Manfred Touron, accessed 20 July 2021, < https://berty.tech/blog/decentralized-distributed-centralized/>

[15]-Hack.guides, Afzaal Ahmad Zeeshan, accessed 22 July 2021, Using ASP.NET Web API for web services for IoT, <pskb-prod.herokuapp.com)>

جامعة اليرموك الخاصة
كلية هندسة المعلوماتية والاتصالات
قسم هندسة المعلومات والاتصالات

بناء نظام موزع للتعامل مع البيانات في تطبيقات إنترنت الأشياء

اعداد الطلاب

عمار الماضي محمد عبد الله

محمد منار بوز الجدي

المشرفين

د. محمد خالد شاهين د. وسيم الجنيدي

م. رامي عباس

الفصل الثاني

2020-2021