Expertise
and insight
for the future

Duy Le-Dinh

Multiplatform Architecture, Protocols and Technologies for Smart Systems

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information Technology

Bachelor's Thesis

01 June 2020

Author Title Number of Pages Date	Duy Le-Dinh Multi-platform architecture and technology for smart systems 35 pages + 1 appendix 01 June 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart Systems
Instructors	Sami Sainio, Senior Lecturer

Low-cost connectivity microprocessors have enabled ordinary electronic devices to be smarter with internet-connected capabilities. Synergized with smartphones and cloud computing, smart devices have formed the internet of things (IoT), which consists of a wide range of hardware, mobile apps, and cloud services.

This diversity in smart systems offers great choices to the end-users. Yet, the same variousness generates new problems for developers: a huge amount of time is required to develop mobile applications and cloud services. Worse yet, this challenge exacerbates when applications need to be maintained or integrated with the third-party service.

This project explores the state-of-the-art of Cross-Platform Technologies (CPT), i.e., React Native, Flutter, Xamarin, and Qt: frameworks that have potential in alleviating the above issues. Another purpose of this project is to study the IoT protocols and architectures and how they can interface with CPT frameworks. Finally, this project aims to find the good designs that empowers engineers to build excellent smart systems while limiting the maintenance efforts.

In this project, information has been gathered from different sources such as official documentation, research papers, review articles, and actual experiments. The study shows that a solid cross-platform smart system should consist of four basic components: devices, gateways, engine controller, and user-space applications. The combination of lightweight IoT protocols such as MQTT, heavy-protocol like HTTP, and cross-platform mobile software empower the system to work seamlessly and smartly

As a result of this final year project, a prototype of the smart garden eco-system has been implemented at Lien Tam Buddhist Monastery in Turku, Finland. The system will be developed further in the future as an open-source project.

Keywords	smart systems, architecture, protocols, cross-flatform, flutter, react, native, mobile, mqtt, coap, iot, rest api

Contents

List of Abbreviations

ADC	Analog to Digital Converter.
CPT	Cross-Application Technologies.
IoT	Internet of Things.
RTOS	Real-Time Operating System
V	Voltage, Electrical energy level
PWA	Progressive Web Apps
HWA	Hybrid Web Apps
HTTP	Hyper Text Transfer Protocol
UDP	User Datagram Protocol
MQTT	MQ Telemetry Transport
CoAP	Constrained Application Protocol
SSL	Secure Sockets Layer
TTL	Transport Layer Security
OS	Operating Systems
UI	User Interface

1 Introduction

As the processing power grows continuously and the cost of electronic components drops, new smart connectivity devices emerge to market. Today, there are approximately 26,66 billion internet-connected devices, while the world population is 7,8 billion [1 1]. In other words, on average, every individual owns more than 3 devices; smart devices are present nearly everywhere. It is believed that this growth trend is likely to continue in the following years, as shown in figure 1.

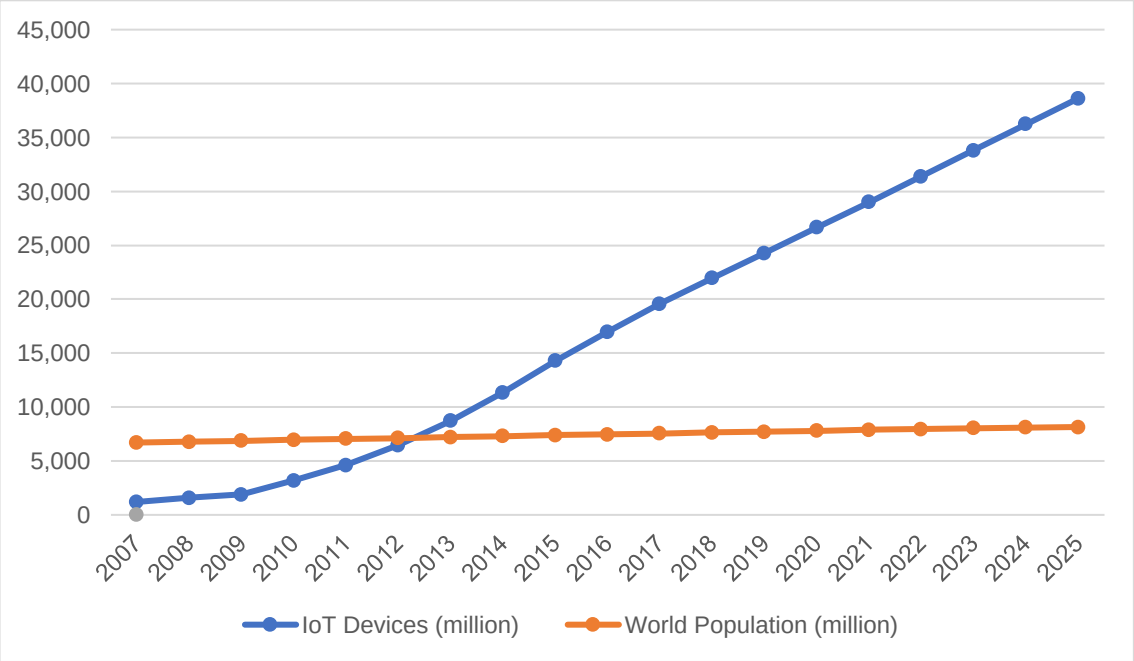


Figure 1. Global Connected and IoT Device Installed Based Forecast between 2007-2025 [1]

The blossom of the smartphone market has changed the course of smart systems dramatically. In fact, a decade ago, the only well-known commodity operating system was Microsoft Windows; whereas, nowadays, Android is the most popular operating system, as shown in figure 2. [3]

A smartphone engages in many aspects of smart systems, e.g., as main controllers, the devices managers, and data collectors. In fact, smartphone applications are important parts of the smart systems nowadays.

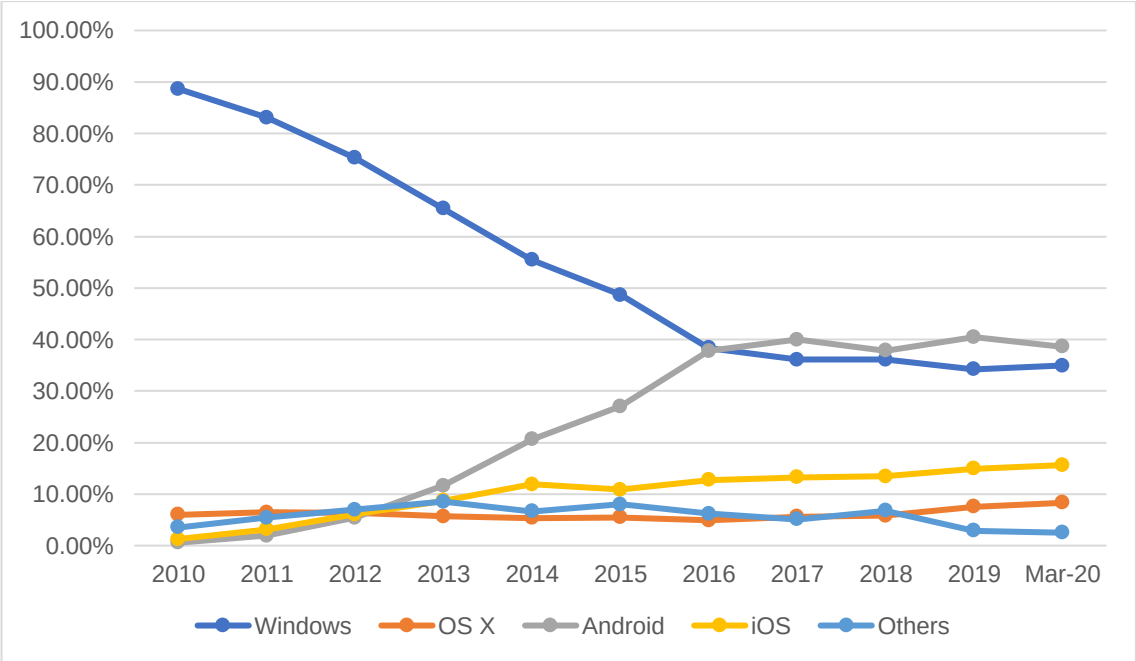


Figure 2. Market share comparison between different Operating System. [3]

The rise of cloud computing enables smart systems to unite multiple technologies and application sectors. For instance, knowledge of how a plant grows can be applied, thereby empowering the system to take care of the plant automatically or give users meaningful insights. Not to mention, cloud connection allows user's input to be stored, processed, and analyzed; hence, the meaningful output could be generated and utilized to enhance system performance. Smart System has evolved to be a knowledge-rich ecosystem, as illustrated in figure 3.

Thanks to these changes, service, and software now take priority over devices, the system's complexity is growing fast with the ability to manipulate data, integrate with third parties, and update continuously. Furthermore, enterprises are shifting their business model from products sales to digital propositions sales. They are not only selling devices but also the eco-system, which includes software, services, contents, community connection, and supports. [4, 5, 6, 7]

This project explores how smart system engineers can cope with the new situations; it examines what architectures, protocols, and technologies permit software developers to build the entitle smart system ecosystem. Any solution for this system imposes several requirements. First, it has to be compatible with the diversity of platforms and devices.

Second, it has to be easy to maintain. Finally, it should have the ability to integrate with the cloud seamlessly.

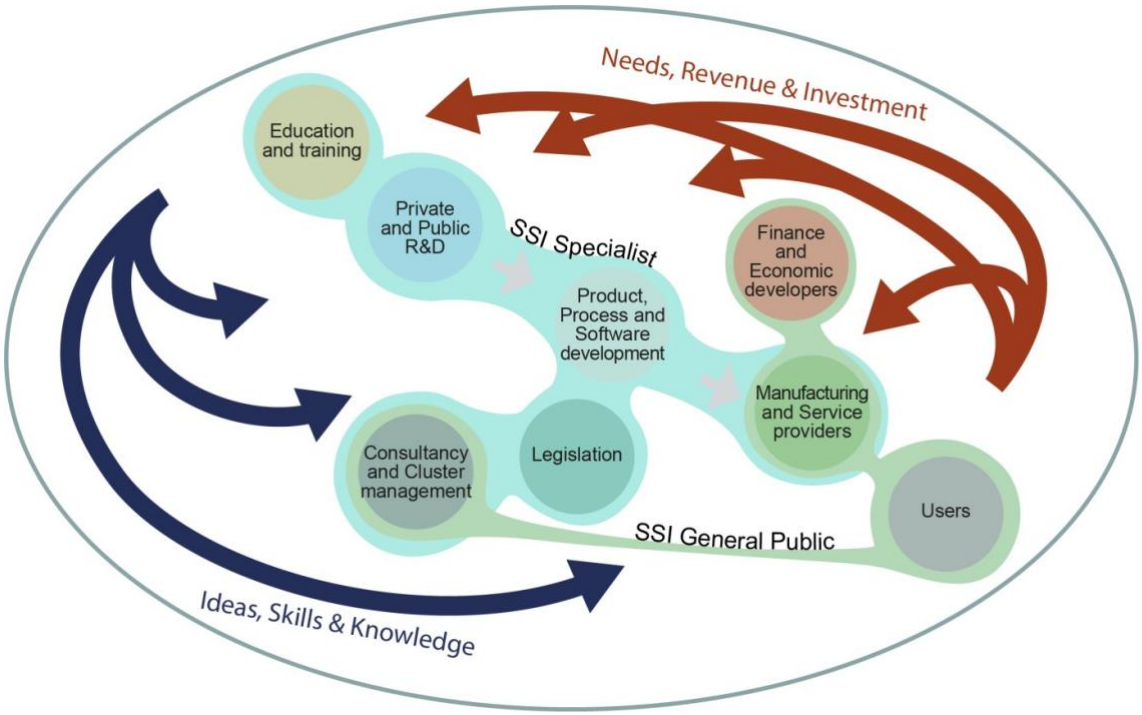


Figure 3. Smart System Ecosystem illustration.

In the next section, this report describes the history and characteristics of a smart system and discusses the basic layers in Smart System Architectures; it shows how devices could interface with the others and with the network. Furthermore, this report shows the CPT in mobile application development; and it shows the state-of-the-art CPT framework such as React Native, Flutter, Xamarin, and Qt.

The use case has been taken for Lien Tam Monastery in Turku: Smart Garden Ecosystem. The prototype utilizes the architecture and protocol mentioned in this report. The user application is built with the Flutter Framework. The implementation chapter explains how decisions have been made.

Finally, a short conclusion will end this thesis and attempt to predict the future of smart systems.

2 What is a Smart System?

2.1 Brief History of Smart System

Since the dawn of civilization, humankind has been building machines that could reduce physical work. Examples of the ancient machines include wedge, inclined plane, wheel. They lay the foundation for the modern industrial revolutions.

From the first industrial revolution started in England to the second industrial revolution, mankind has invented sophisticated machines for mass production, utilized the power of electricity, built better engines, etc. The world has witnessed the significant development in technology and economy.

In 1946, the first fully functional digital computer – The ENIAC (Electronic Numerical Integrator and Computer), was completed [8]. Ten years later, the Massachusetts Institute of Technology inventors invented the transistor and opened the new era of technology [9]. The third industrial revolution began with the development of electronic infrastructure, computers, and digital technology.

In 1971, Texas Instruments developed the first microprocessor: the TMS 1000 series, (figure 4) [10]. In the same year, Intel 4004 – the first single-chip microprocessor was released [11]. During this period, the integrated circuit's price dropped while the usage surged.

Figure 4. The TMS 1000 Microcontroller.

In 1990, the Internet was invented, marked the important milestone in the development history of smart systems [12]. People started finding themselves living the parallel worlds: i.e., the physical world and the digital world. One with machines and everyday objects, one with no physical form, but a wide range of social media and digital contents.

Smart system has combined these two worlds, connected the digital world with the physical world, and transformed everyday objects into smart things. With the rise of smart system, home appliances become a system of smart home, cars become smart cars, cities become smart cities, etc. The popularity of smart systems has increased significantly from the early 2000s.

Cisco IBSG estimated that between 2008 and 2009, the Internet of Thing was “born”. In 2012, the number of internets connected devices surpassed the number of the world population. The trend keeps going up. Today, there are about 26.6 billion connectivity devices. [13]

2.2 Characteristics of Smart Systems

Smart Systems, also well-known as Embedded Systems, or Internet of Things are systems that incorporate sensors, actuators, and controllers with the purpose of analyzing and processing situation’s data, in order to decide predictively or adaptively on which actions should be performed. These systems are operated autonomously, with low energy consumption and networking capabilities. [14]

The key factor of smart systems is connectivity capabilities. Devices within the systems can interact with users, other devices, and cloud services. Connectivity enables smart systems to adapt to changes. Furthermore, thanks to this interaction, smart systems could identify, understand, and extract information, e.g., meaning, time, place, to make better decisions and get smarter.

Well-known as the precision system, smart systems could eliminate human error in a variety of fields, such as healthcare, aerospace, manufacturing, etc. Besides, the characteristics of having low energy consumption and lightweight make smart systems an important role in areas that are sophisticated and safety-critical.

A wide range of application sectors can benefit from applying smart system technology. Not only they are benefiting from smart systems, but smart systems could also benefit from them, with the ability to unite knowledge from different areas and create a single smart system ecosystem.

2.3 Trend and Challenges in Smart Systems

In 2014, Gartner, Inc – a research and advisory company, predicts that a typical house could host several hundred smart devices by 2022; the rise will start in 2020 [15]. It is expected that the smart home industry will offer a variety of opportunities for innovative organizations. The world is expected to have nearly 40 billion IoT devices by 2025.

New challenges of user expectation and demand are waiting for engineers to solve. First, as the price of microcontrollers drops, all devices could have an embedded controller inside, which enables them to have additional capabilities such as connectivity, performance monitoring, etc. At the same time, users expect more from smart devices, meaning that it has to perform real smart actions. Smart systems should be able to take care of their own and interact with users.

Second, thanks to the growth of smartphones, mobile apps will be an indispensable part of smart systems, which will be responsible for a wide range of functionalities, such as controlling, managing, even entertaining. Users expect devices and applications to be updated frequently, with new features.

Finally, with the development of cloud computing, services will take priority over devices, not only more ecosystem will be introduced but also the larger unified ecosystem will emerge; smart devices should be able to interact cross-ecosystem, they should be integrated with cloud services that provide knowledge for the system, collect user data, process and make good use of them.

How are engineers going to cope with these changes? This thesis discusses possible solutions in the following sections.

3 Smart System Architectures and Protocols

3.1 Four Basic Layers of Smart Systems

The fundamental layers of every smart system are identical to each other; even though each of them is designed or advertised differently; even though standards in the smart system industry, i.e., Alexa standard, IFFTT, Apple Home Kit are different; even though different devices and applications have different core technologies. [15, 16, 17]

Components of all smart systems could be categorized under 4 layers: Smart Devices, Gateway, Engine Processor, and Application Layer (figure 5). In other documents, this layer can become a variation of 3 layers or 5 layers. However, the principle of all is similar.

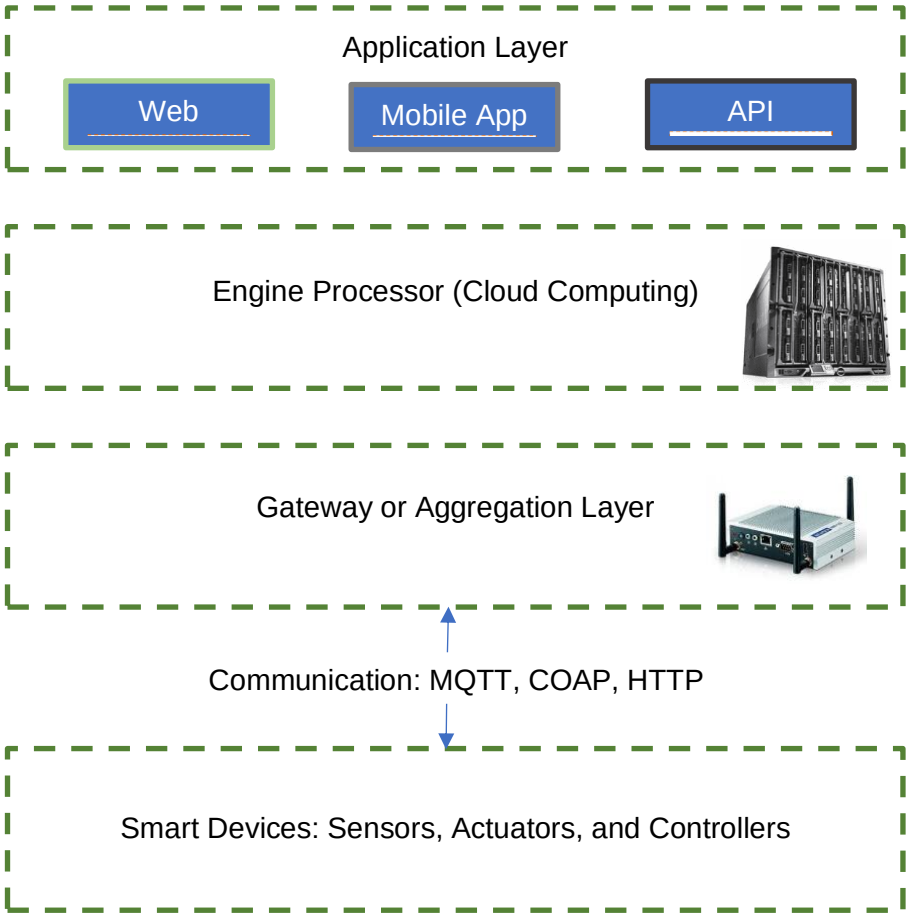


Figure 5. Basic Smart System Architecture.

Firstly, the most important part of the smart system is smart devices, which consist of sensors, actuators, and controllers. Sensors are the small electronic components that respond to a physical stimulus and transmit the results as impulses. Collected data, after being passed to the controllers, will then be passed to the gateway. Also, actuators execute the command given by controllers. Sensors can be embedded into a controller or can be a standalone device.

There is a large range of communication protocols that enable smart devices to interface with the gateway, such as MQTT and COAP in home appliances, or Fieldbus and IO-Link in the industrial systems. Other designs that consider protocols as a transportation layer. Protocols are critical in smart systems as they allow devices to communicate efficiently, thereby affecting significantly energy usage and response time.

The next one is the gateway layer. It contains software that collects, filters, and digitally converts raw data, and pre-processes them for the main processors. The gateway is the bridge between devices and processor. It ensures that data is transferred efficiently and securely.

Next, cloud computing is the brain in the architecture of smart systems. Without this engine processor, a smart system is just a system of embedded devices. A cloud platform can obtain and provide the knowledge to the smart systems; it is the engine that can unite other technologies and sectors, combine all of these valuable data, and form the smart ecosystem. Besides, in scalable systems, it can help reduce energy consumption and downtime.

Finally, the last part is the application layer, which has become the indispensable element of today's smart systems. Applications interact directly with users to control and sets up the system, collect user's data, and provide useful feedback. It can provide the API to give external systems a portal to access. Furthermore, it is a part of the device management system, which will be described in the next section.

3.2 Devices Manager in Smart Systems

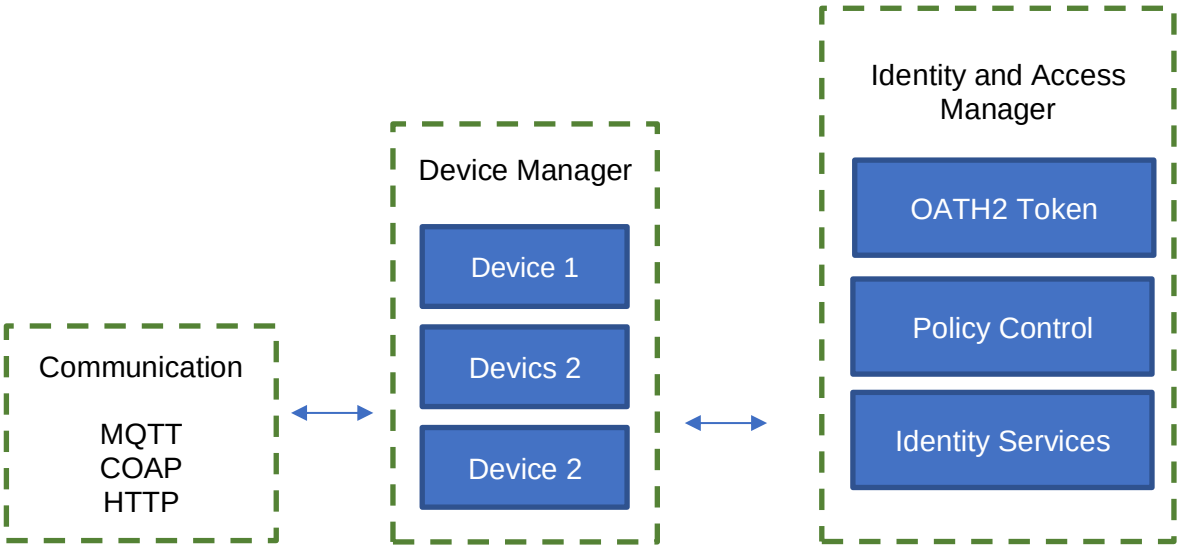


Figure 6. Device Management in Smart System

The Device Manager System is designed to authenticate, insure, configure, monitor, and maintain the firmware and software of the smart system. This software plays a critical role in ensuring that the system functions healthily with full connectivity and security feature. The efficient device manager is one of the important answers to the question of how a smart system can be maintained well. Figure 6 describes the architecture of the devices manager system. [18]

There are three important components in the device management system. First, the Identity and Access Manager function as the security layer. This component controls the policy, allows users to be identified, and provides access to users. Second, the device manager is a software that manages different agents in the smart system. This software can be a standalone software in large smart systems or part user applications in small systems.

Device manager can interact with devices and perform necessary tasks such as configure, monitor, or update the software through many different protocols, similar to the way components in the smart system communicate with the others. Examples of these protocols are MQTT, CoAP, and HTTP, etc. In the next section, popular protocols will be described.

3.3 MQTT

MQTT stands for “Message Queue Telemetry Transport”. It is defined as “Publish-subscribe-based “lightweight” messaging protocol, which is used on top of the TCP/IP protocol”. [19]

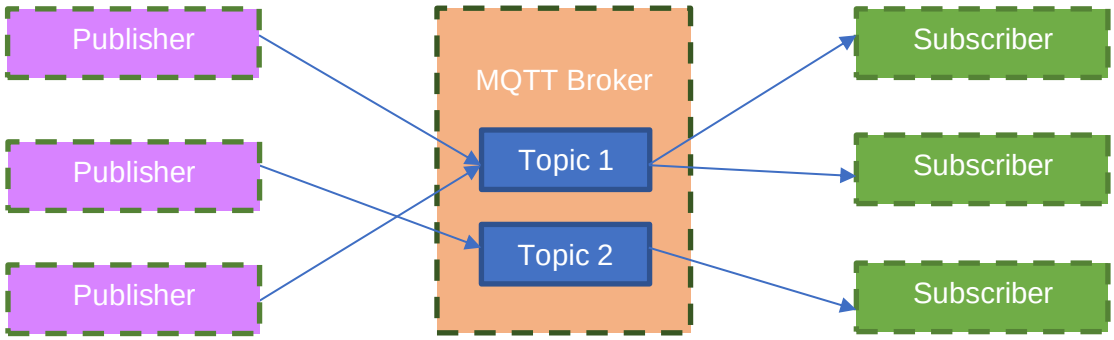


Figure 7. How MQTT protocol works.

MQTT is an ideal protocol for M2M applications, such as smart system applications. One popular example of MQTT is Facebook Messenger. Figure 7 shows how MQTT works. An MQTT Broker is a server that has different topics. Different client nodes subscribe and publish to the topic. When a message is broadcasted, messages are sent to the subscribers.

There are three Qualities of services (QoS) in MQTT:

- QoS0: Broker and client send data only once, the process is confirmed by TCP/IP protocols
- QoS1: Broker and client send data with at least one confirmation from the other end. In other words, there can be more than one confirmation that data has been received.
- QoS2: Broker and client ensure that data is received exactly once, this process requires the four-part handshake: the client publishes QoS2, broker replies, the client confirms, broker confirms.

3.4 COAP and HTTP

Unlike MQTT, CoAP (Constrained Applications Protocol) and HTTP (Hypertext Transfer Protocol) are one-to-one protocols. While the HTTP is the most popular protocol of the world wide web, which is used to power the web and other activities on the internet, Co CoAP is designed for the constrained environments. CoAP is similar to HTTP but it supports smaller data packets. [20]

While HTTP supports TCP, CoAP utilizes UDP (User Datagram Protocol). Compare to UDP, TCP is slower as it is a heavy-weight protocol, with a tracking system, big header size, connection-oriented, and extensive error checking. In term of security, SSL/TSL is not available for CoAP. Fortunately, there is an alternative tool, namely Datagram Transport Layer Security (DTLS). Similar to HTTP, CoAP follows the client/server model; it could send the GET, PUT, POST, and DELETE requests. CoAP is a RESTful protocol.

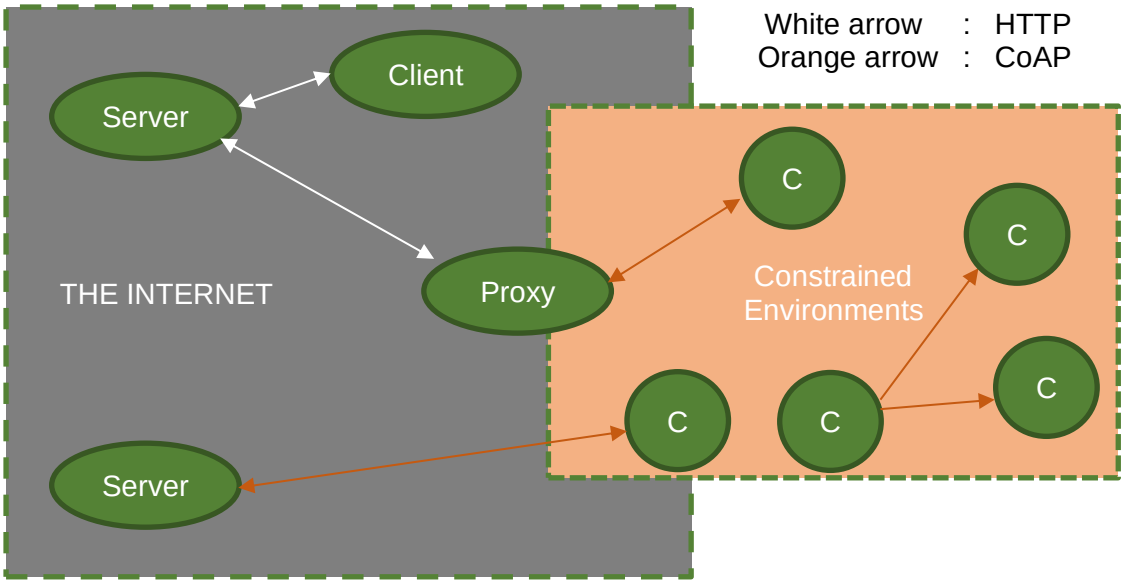


Figure 8. HTTP and CoAP protocols.

Along with MQTT, CoAP is one of the most popular protocols for smart system devices as they are lightweight and energy friendly. Especially, CoAP has a big advantage over other protocols: it is easy to be translated to HTTP. Using CoAP and HTTP altogether allows system to communicate seamlessly, as shown in figure 8.

3.5 WebSocket

WebSocket is two ways communication protocols between client and server by using a TCP socket. This protocol is very efficient, compared to HTTP. WebSocket initial the handshake as the HTTP request. After the connection has been established, it maintains the persistence with bidirectional communication. Figure 9 illustrates this method. The strategy used in the WebSocket protocol is push-based, meaning the client only receives the message when the new data is available. In the case of HTTP, to maintain the nearly real-time connection, the client needs to proceed with the long-polling or polling technique. [21]

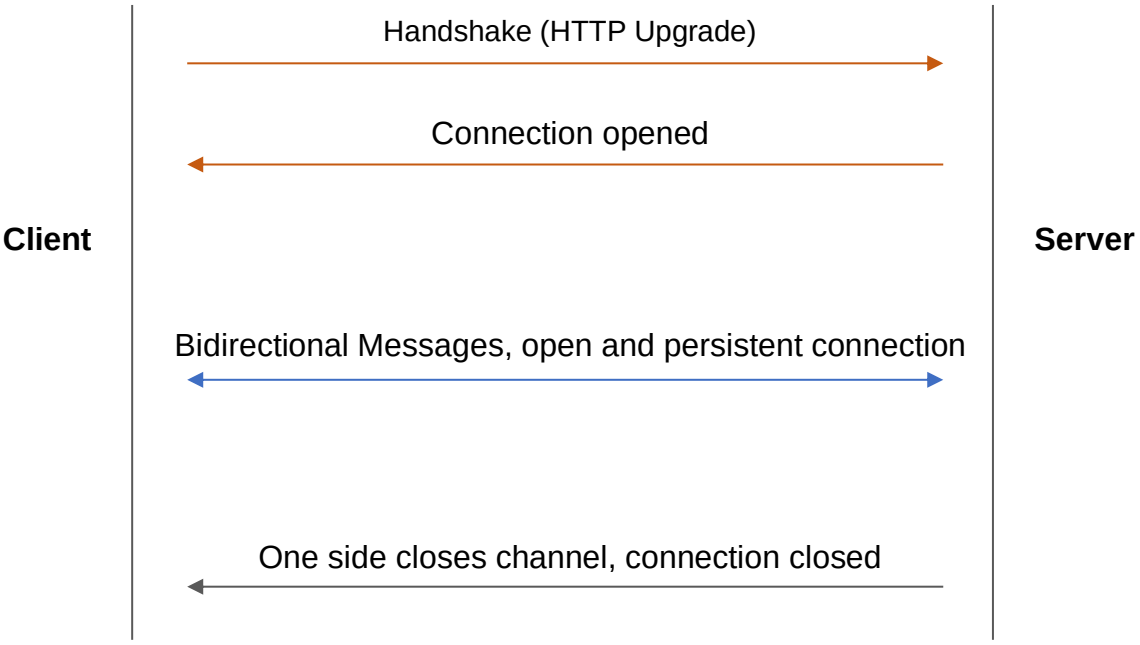


Figure 9. WebSocket handshake and communication.

Web Socket is not the greatest option for communication between smart devices. However, it has the potential in the communication between server to server, server to the edge processor. The following table compares the described protocols.

Table 1. Protocols Comparisons.

	MQTT	CoAP	HTTP	WebSocket
Year	1999	2010	1997	2008
Architecture	Client/Broker	Client/Server Client/Broker	Client/Server	Client/Server
Header Size	2 Byte	4 Byte	Undefined	Undefined
Packet	small, up to 256 MB)	small,	large	large
QoS	QoS 0, QoS 1, QoS 2	Confirmable message	Limited	Unreliable
Transportation protocols	TCP	UDP	TCP	TCP
Power Consumption	Low	Lowest	High	Highest
Security	TLS/SSL	DTLS, IPSec	TLS/SSL	TLS/SSL
Connection	consistent	restful	restful	consistent
Usage	D2C/C2C	D2D	C2C/D2C	C2C

The lightweight protocols such as MQTT or CoAP are suitable to use in the constrained environment. For example, smart devices can be connected using MQTT if the opened connection is required or CoAP if it is a restful connection is preferred. MQTT could also be used to connect the device to the cloud if the consistency is needed. On the other hand, the connection between devices and the cloud can be done via either CoAP or HTTP protocol. Finally, WebSocket can be utilized to connect different clouds together.

4 Mobile Cross-Platform Technologies in Smart System

Application is an essential part of today's smart system architecture, as explained in the previous section. To build an app for both iOS and Android smartphones, developers can choose either native technologies or non-native technologies. The native mobile framework is the best choice for apps that require high performance, stability, and feature

access. However, in terms of the development speed and cost, native technologies show problems, since apps have to be built and maintained multiple times.

This section reviews the state-of-the-art in cross-application technologies, which shows great promise in solving the native app's development speed and cost problems.

4.1 Progressive Web Apps and Hybrid Web Apps.

Progressive web apps (PWA) are a website that look like a mobile app but it can work offline. Hybrid Web Apps (HWA) are also web apps but it is embedded inside Native apps. They both are built using web technologies, i.e., HTML, CSS and JavaScript. Example of this technologies used in these browser-based solutions includes React, Angular, Js Ionic, PhoneGap, etc. [22]

The advantage of web apps is the fast development. Since a website is a cross-platform app itself, using this technology, a developer needs to write code once only, and the app is going to work on all devices that support web, meaning all smartphones and personal computers. [23]

Another advantage is that a web can be deployed faster. A PWA app does not have to go through App Store or Play Store to be installed. Users can access app directly through a website or save it as an app. It is also linkable, which can be shared via a URL, without any complex installation. While native apps are updated using the stores, developer can update PWA directly. In cases of native apps, some users may miss the update and be vulnerable to attacks. [24]

HWA is installed through the App Store or Play Store like the native apps. Both HWA and PWA use HTTPS protocols, which is a secured connection.

Even though they have many advantages and sounds like an ideal choice to develop an application for Smart Systems, they have problems. Firstly, compared to native apps, the general performance and graphical performance is slower. Second, the biggest disadvantage is feature accessibility. There are many features that a PWA application cannot access [25]. Examples are shown in the below, table 2.

Table 2. Feature Assess of web apps.

Supported	Not Supported
Bluetooth	NFC
Local Notification	Proximity Sensors
Touch Gestures	Contacts
Geolocation	SMS/MMS
Camera	Geofencing
Device Motion	Others
Offline Storage	
File Access	

As application is very important in Smart System, the lack of some feature or the low performance may not be tolerant. Therefore, this thesis does not discuss further about the web app technologies. In the following section, this thesis explores the most popular native cross-platform framework: React Native, Flutter, Qt and Xamarin.

4.2 React Native Framework

Developed and introduced by Facebook, Inc in 2015, React Native is the combination between React and Native Development. React Native was born at a Facebook's internal Hackathon event. React Native solves the issues of both Hybrid apps and Native apps.

An application written using React Native can run on both iOS and Android. React Native source codes are intepreted to Javascript, which will then be converted to Native code by a series of elements known as the Bridge, as shown in figure 10. [26]

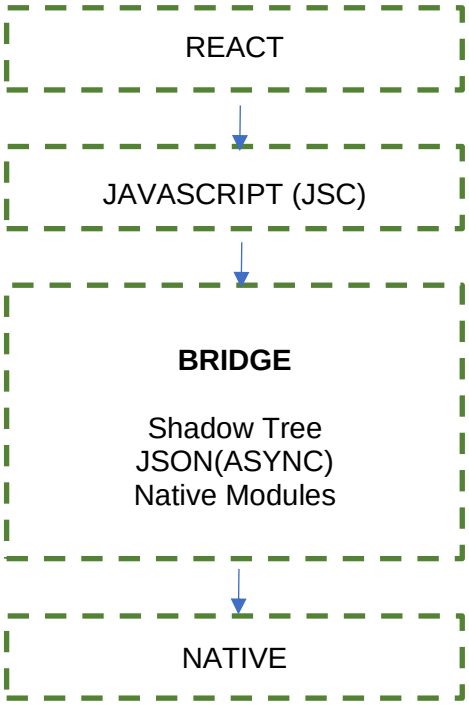


Figure 10. The Architecture of React Native

Since React Native utilizes JavaScript and React framework, companies which have other resources built in JavaScript can be beneficial with React Native. Shopify is an example, it has announced its change from Native apps to React Native, with one of the reasons is to leverage the JavaScript resource. [27]

In terms of smart system, React Native allows apps to be built quickly, and able to run on various platforms. Besides the two main smartphone operating systems: iOS and Android, React Native could be used to run on web server. In addition, the experimental feature enables React Native to work with Electron Js in a Desktop app, meaning, it can be used to build a Desktop software that can run on Windows, Linux and Mac OSX. [28]

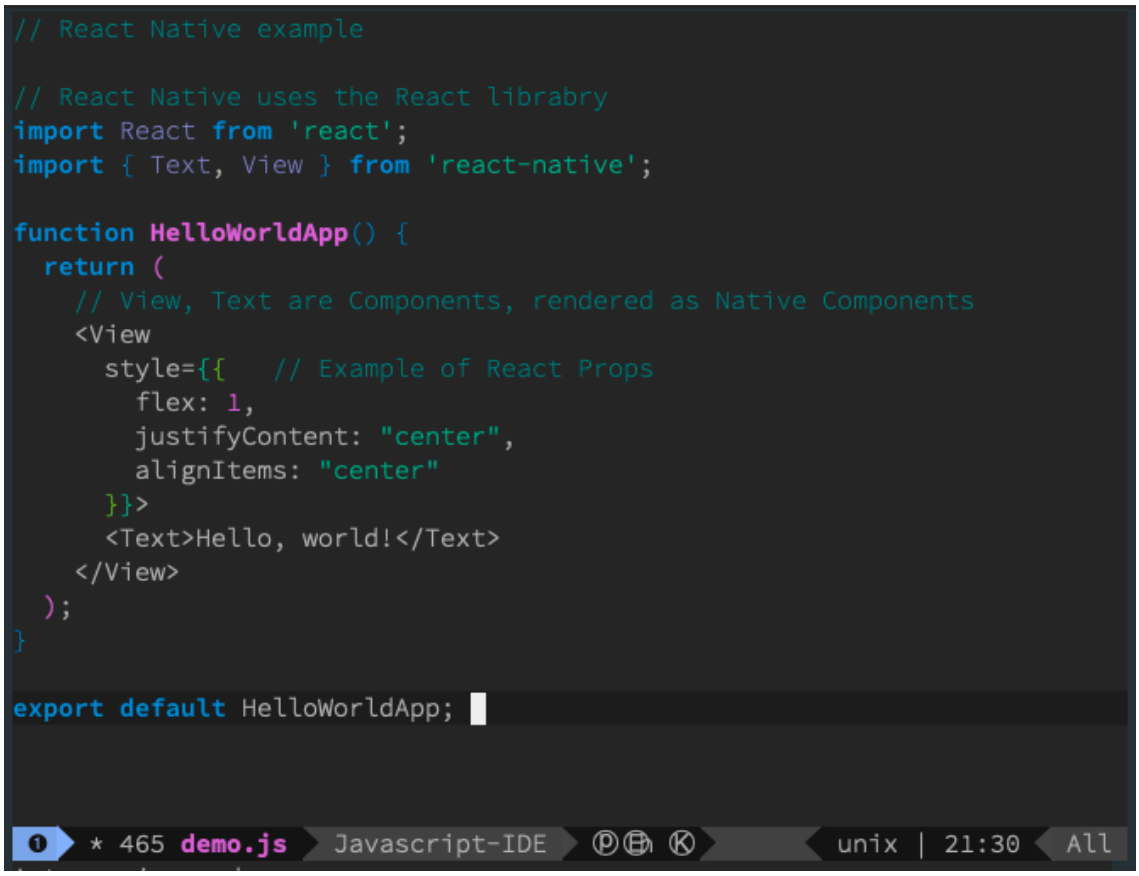


Figure 11. Example of Hello, World application written in React Native.

As demonstrated in figure 11, An app written using React Native consists of Components, which is a function that has props (properties). The bridge modules compile React components into the iOS/Android UI Components. Components are very useful; they allow the application to be developed quickly by reusing them. [29]

A compelling advantage of React Native is the performance, it can achieve 60 frames per second with the native look and feel. This is a powerful feature as it allows apps to run animation easily [30]. Unfortunately, since the interpreted layer is heavy, compared to the Native Apps, the performance is still much lower, even though, users usually cannot detect the difference.

4.3 Flutter Framework

With React Native, the app can be developed quickly. However, there is one more issue, which is ‘the-native-performance’. React Native app is still slow, compared to a native

app. Flutter not only can be developed fast but also it has native performance. Flutter can be considered as the state-of-the-art in cross-application technology. [31]

Flutter is written with Dart - a new programing language created by Google. Dart has excellent designs, it combines the best features of C++, C#, Java, ES7, with elegant and attractive syntax. The big success of Dart is the ability to be either a compiled or an interpreted language. For a web application, Dart can be interpreted to JavaScript, while on Flutter, it is compiled to the machine code.

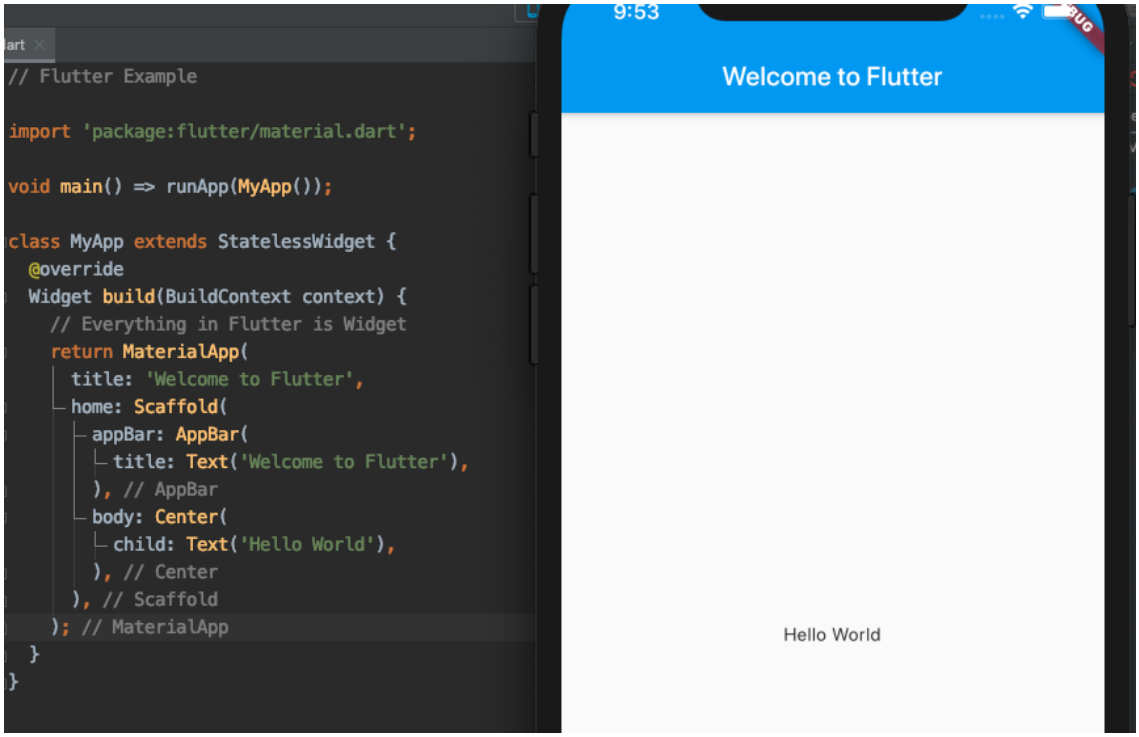


Figure 12. Example of Hello, World application written in Flutter.

Inspired by React, even though the language syntax is different, Flutter's UI is made of widgets, which similar to React Components [32]. Figure 11 and 12 shows how a Widget is implemented. In addition to widgets, Flutter's state management system is also similar to React-Redux. [33]

Flutter controls all screen's pixels by using its rendering engine. This is an advantage and also a disadvantage. On the one hand, it renders fast and keeps the same look on all devices that it supports. On the other hand, an app has to be shipped with this engine and become bigger. Besides, it does not render the Native Components. [34]

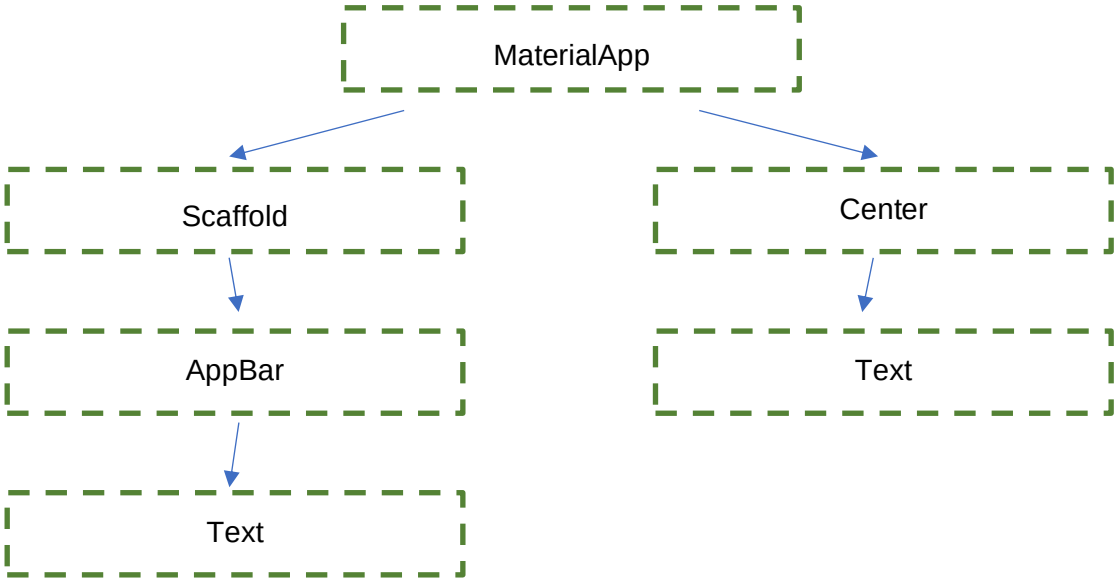


Figure 13. The widget tree of the above Hello, World application.

The first stable version of Flutter was launched on December 4, 2018. Even though it is young, it attracts a lot of attention from the community. On GitHub, it is now (5/2020) having more stars than React Native. Moreover, the features are growing fast, the current Flutter version not only supports mobile platforms but also Mac OSX and Web platform. The experimental version even supports Linux and Windows.

4.4 Xamarin Framework

Xamarin was a cross-platform framework developed by Xamarin, which now belongs to Microsoft. It is open-source and can be used to develop Android, iOS, macOS, watchOS, and tvOS apps. The language that Xamarin utilizes is C#.

Xamarin has an impressive performance. Even though the performance is still slower than Flutter, it is close to the performance of the native apps. Besides, another advantage of using Xamarin is that it uses native user interfaces, which allows apps to look native. Xamarin supports full native API access. [35]

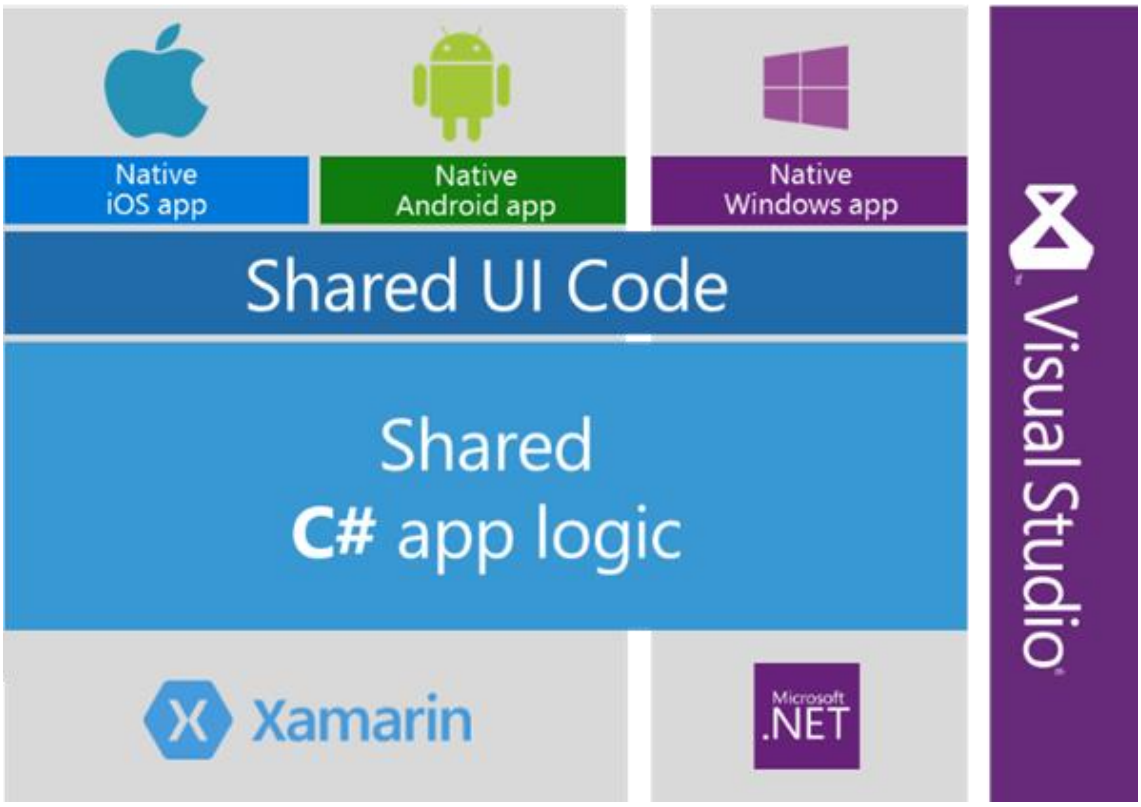


Figure 14. The architecture of Xamarin Framework

One downside of Xamarin is that although it is an open-source, in order to leverage all potential, Visual Studio IDE is required, and the Professional Visual Studio and Enterprise Visual Studio IDE are very expensive. Another disadvantage is that Xamarin is slightly delayed in updating new features. [35]

4.5 Qt Framework

Qt framework is a very famous open-source framework, it is an old framework with 24 years of history, with a community of approximately 1 million developers worldwide [36]. This is a powerful framework, as it can work on almost all platforms. It can work with embedded devices, microcontrollers, internet of things, desktop applications, multiple screens, and mobile devices. According to Qt, it is a “code once, deploy everywhere” framework.

Qt is built with C++, which is a language that most of the smart system developers know. Currently, as C++ is the fastest object-oriented programing language, and as a result,

there is no doubt about Qt performance. Examples of applications that are written in Qt are Spotify, Weather App, Atlas, and Math Graphica. For other platforms, Qt application can be seen in many different places such as cars' screen, smart home control panels, etc. [37]

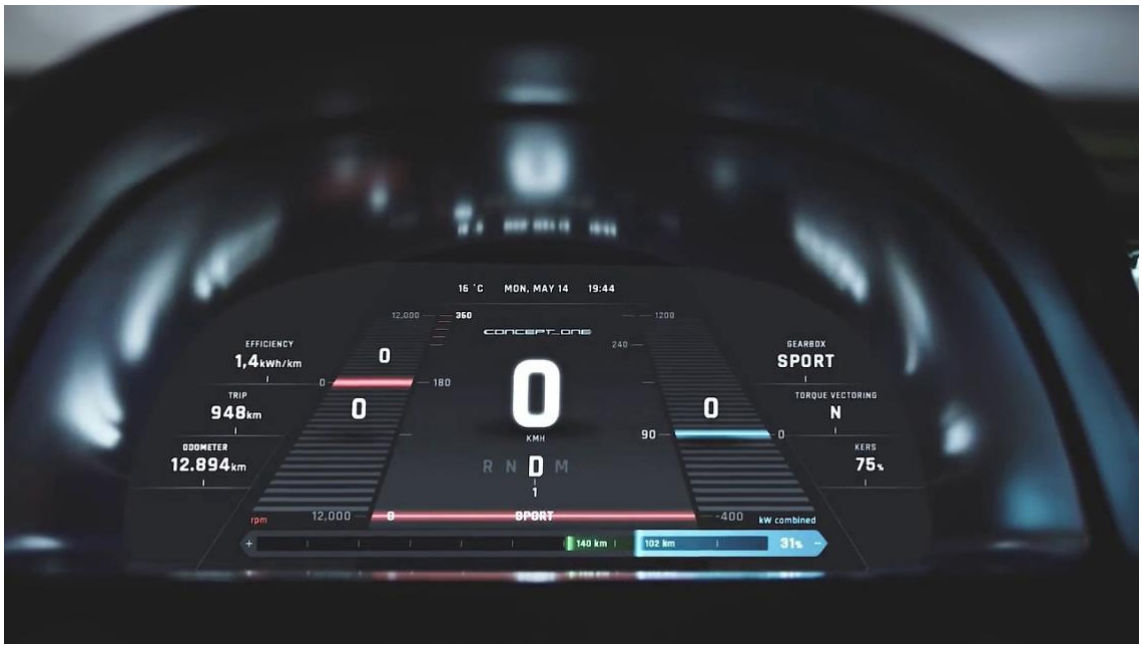


Figure 15. Qt is used in the Rimac Automobile electric super car info and entertainment system.

Similar to other frameworks, Qt has its disadvantages. Qt may be a good option for commercial uses. However, for a non-commercial version, there are constrains. For example, without the Qt Quick Compiler, the application startup time is long. Besides, Qt is under LGPL version 3, GPL version 2, and GPL version 3, meaning applications that developed by Qt have to be categorized within the same licenses. [38]

The above open-source problem is one of the reasons why Qt has not been tried during this project. Apart from that, Qt has the large potential to be applied in Smart System, because of it is mature, supports wide cross-platform, and utilize C++.

5 Use Case: Community Smart Garden Ecosystem

5.1 Project Description

Smart Garden is one of the most popular applications of smart system today. This system is available on a big scale, from a small single plant system to an industrial system. This project is carried out for a community garden at Lien Tam Buddhist Monastery. The goal is to build a smart ecosystem for Monastery's garden.

The first functionality of the system is to control the drip irrigation system autonomously. This system could turn the pump on, open the valves to supply water on a daily basis or when needed. It uses a series of moisture, light, and temperature to monitor the garden and make decision based on this information.

The second feature of the system is to monitor the garden. Since the garden is taken care of by infrequent volunteers, cooperation is sometimes disrupted. The system can monitor people who enters the garden, what they do, and notify the others. Besides, it can help the manager to manage the usage of farming equipment such as tools, machines, and gases.

Lastly, it is a learning platform. An application can provide necessary information about the tropical plants, how to take care of each plant, and how to harvest. Information is synchronized with the cloud. Volunteers can study, monitor, and control the garden via the mobile application that can run on both iOS and Android.

The prototype has been built and ready for this year's growing season (2020). This system can automatically monitor the soil moisture, give users the latest update of the garden's status. It can control the whole system automatically or be controlled by users over the mobile application through the internet. The learning section on a mobile app can teach users about vegetables that have been planted in the garden. Due to time constrain, it is not yet a fully functional system. However, this prototype is an excellent example to demonstrate the architecture, protocols that are reported in this thesis. Besides, the mobile app is a perfect example for the benefit of using cross-platform technology in nowadays smart system ecosystem

5.2 Design the Ecosystems

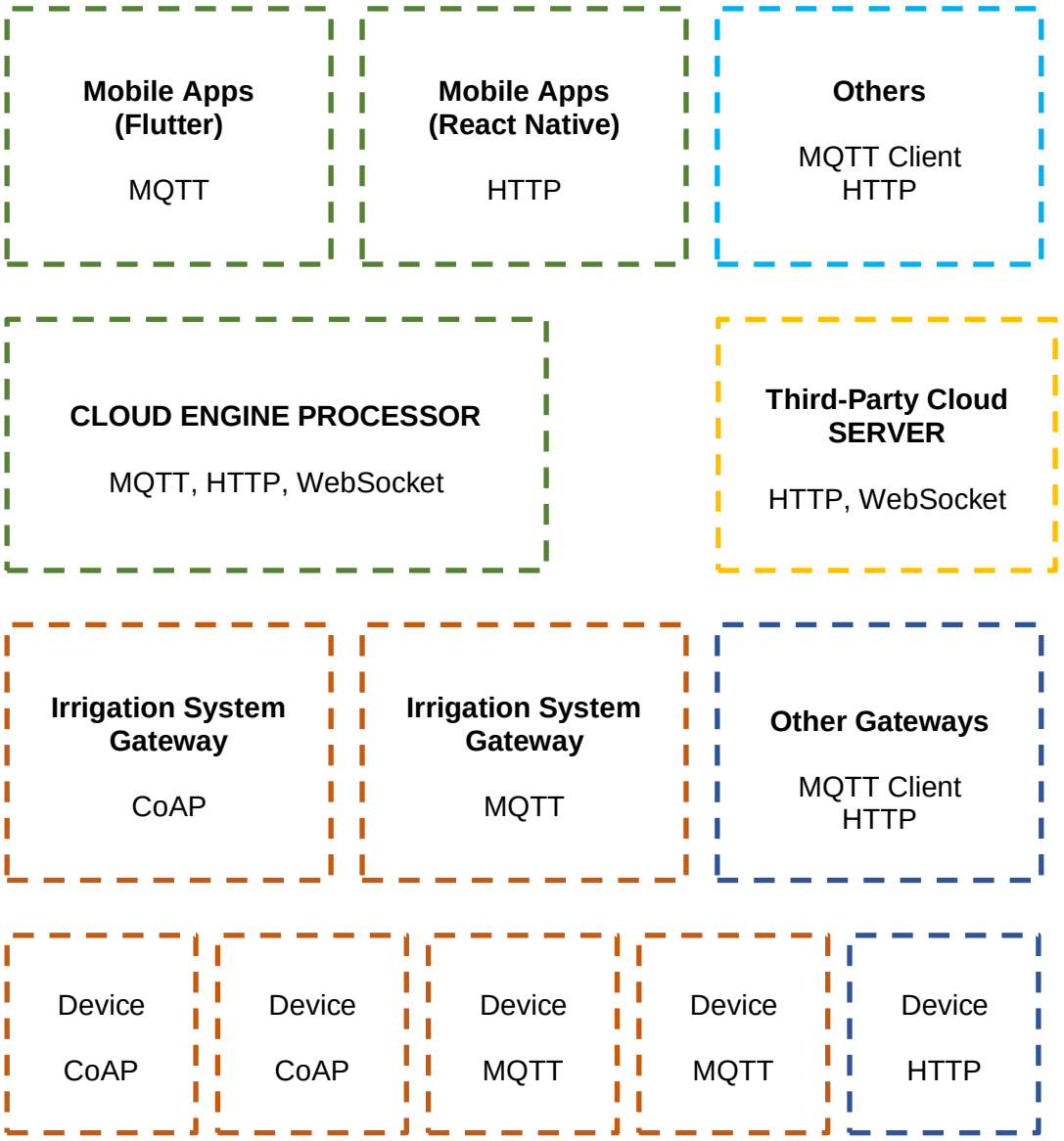


Figure 16. The design of Smart Garden Ecosystem

Figure 16 illustrates the basic design of the smart garden system. This design is very flexible. At the heart of the system, a cloud server responds to all requests from applications and devices gateway. The cloud interfaces with applications and gateways through various protocols. Devices with low-power consumption and consistent connection can use MQTT protocols to communicate with the cloud, while devices that do not require the consistency can send data to the server through HTTP requests.

The server could interact with the third-party clouds to provide extra services to the system. For example, if the client decides to buy a system from Gardena – a company that provides smart garden equipment, the integration between the client's cloud and Gardena's cloud can be implemented in order to allow users to control Gardena's equipment using the existing mobile app. Using the cloud server also enables the system to connect to a bigger cloud server that can provide services such as Big Data, Assistant, AI easily. The main task is to connect to the API provided by server providers.

On the application layer, there can be multiple applications connecting to the cloud. These apps are technology independent and platform-independent. For example, a volunteer's mobile application can get data from the server using an HTTP request, a manager controller app can maintain the MQTT connection to the server to receive "nearly-real-time" information.

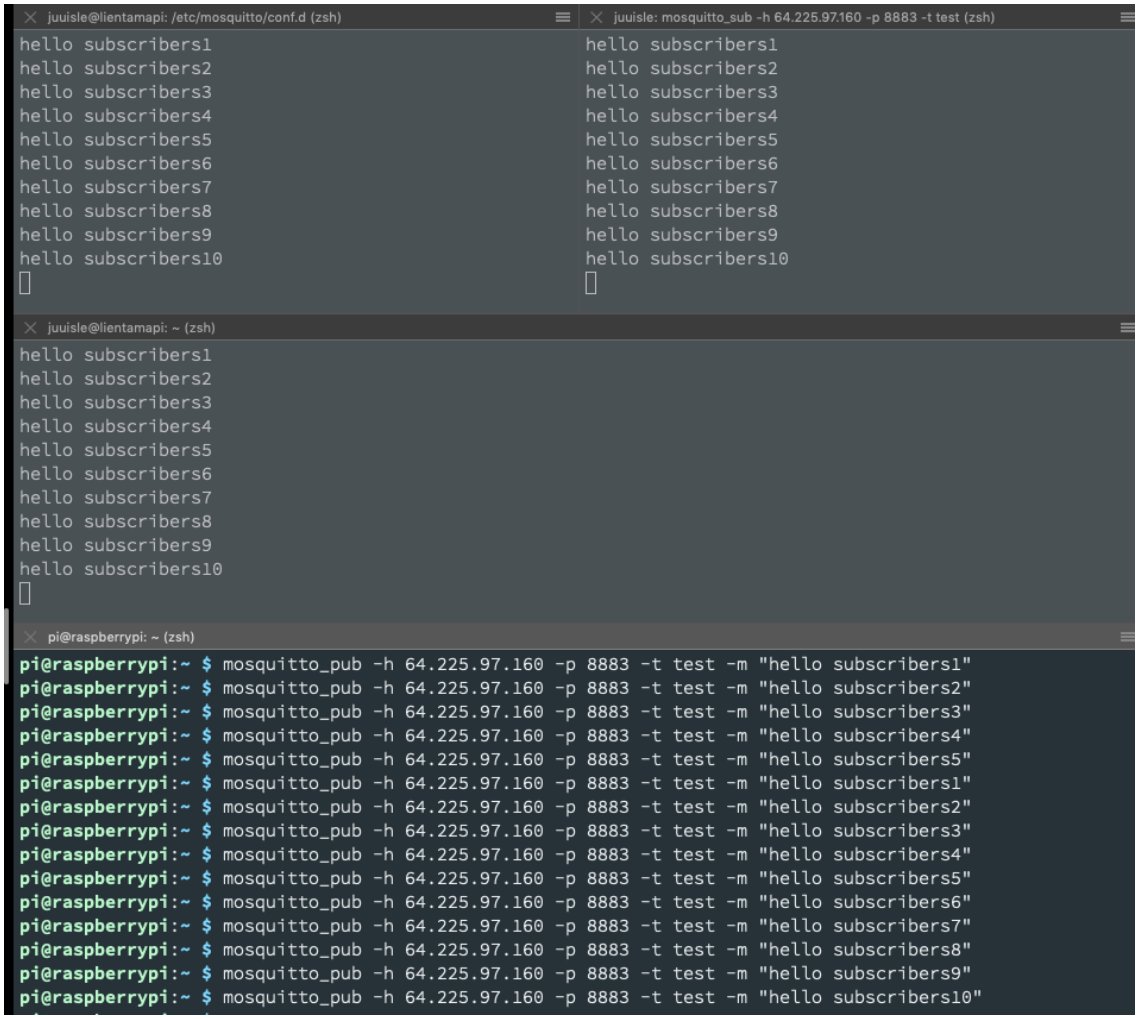


Figure 17. Test MQTT connection between several machines.

On the device layers, this design allows the system to be device-independent. If the device is broken, it can be replaced with other devices, as long as the new one provides the same protocol. The manager only needs to configure the new device in the management app.

One disadvantage of the system is if the internet is down, the whole system will corrupt. An edge computer can be a temporary solution; however, it cannot provide the same functionality as the cloud provides. Therefore, this system is internet dependent.

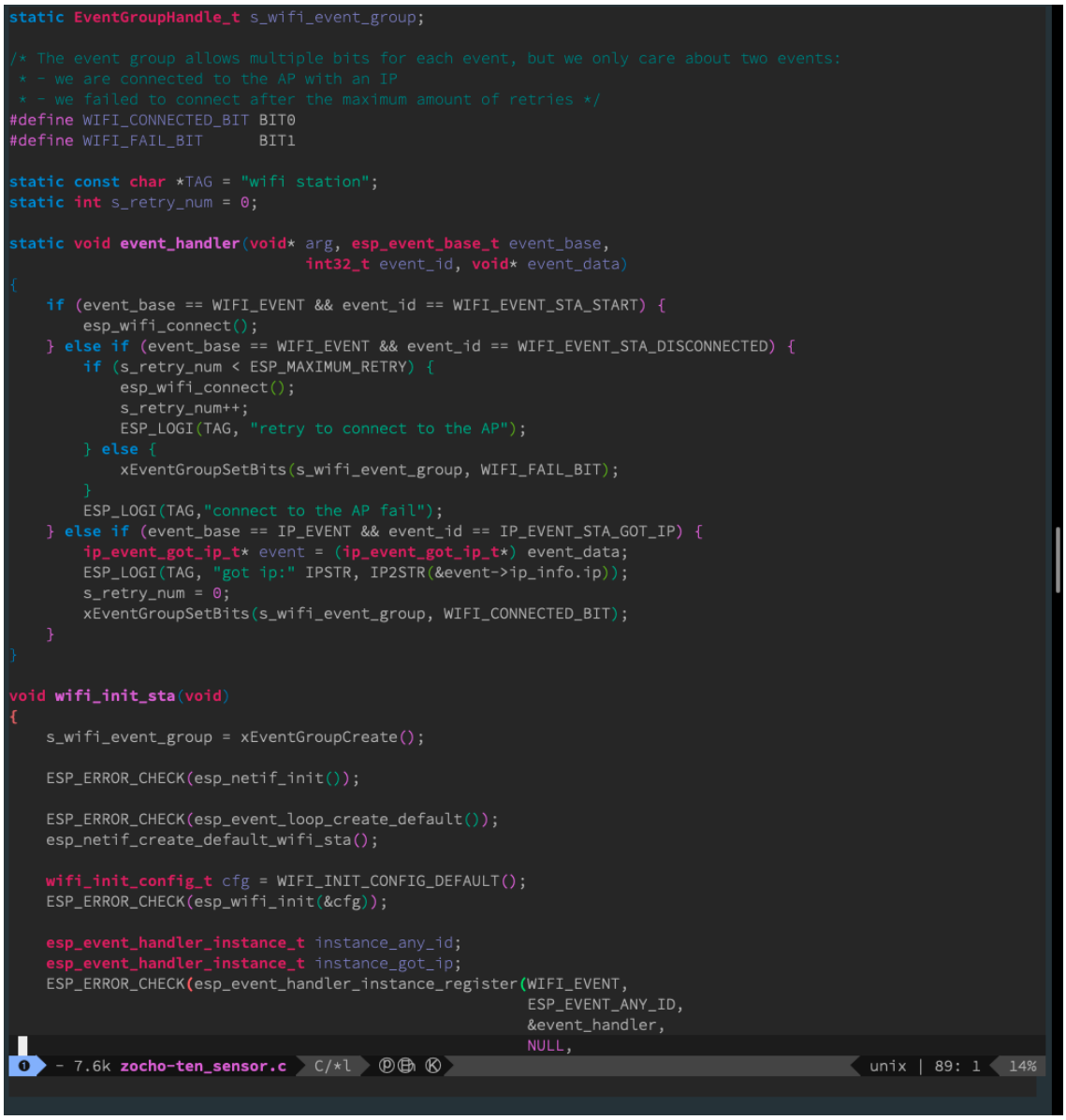
5.3 The System's Prototype

The system's design is sophisticated and requires time to build a complete solution. However, a simple prototype has been completed and can be used to demonstrate how the architecture, protocol, and especially the mobile application works.

The prototype consists of a wide range of components: a server hosted in Digital Ocean, Deco's mesh WLAN system, Capacitive Moisture Sensor, Solenoid, ESP32 Development Boards, and most importantly, a mobile app written in Flutter that can run on both iOS and Android smartphone.

A virtual server rented on the Digital Ocean is cheap and stable. It can be expanded easily if needed. In this project, the backend source code is hosted in the Digital Ocean. The Eclipse Mosquitto and Flask microframework have been utilized to power the central engine. The Mosquitto provides the MQTT broker for devices communication while Flask provides the REST API to enable, for example, data processing and third-party cloud integration.

For network infrastructure, the existing Deco Mesh network system of Lien Tam Monastery is re-designed to cover the whole garden. Mesh network is a perfect solution for smart systems because it allows the network to be self-configuring, self-healing, and provides great coverage.



```c
static EventGroupHandle_t s_wifi_event_group;

/* The event group allows multiple bits for each event, but we only care about two events:
 * - we are connected to the AP with an IP
 * - we failed to connect after the maximum amount of retries */
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static const char *TAG = "wifi station";
static int s_retry_num = 0;

static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        NULL,
 - 7.6k zocho-ten_sensor.c  C/*l                                  unix | 89: 1  14%
```

Figure 18. Sample FreeRTOS and ESP-IDF librabry used in the project to power the sensor.

The capacitive soil moisture sensor is selected to measure the moisture of small plant's soil. There are many different types of soil moisture sensors. However, the capacitive soil moisture sensor is more durable. A low-cost solenoid has been purchased for testing purposes; better options are being researched. ESP32 microcontrollers are being used as the controller to exchange data to the cloud via MQTT protocol. In this project, a DOIT Esp32 Devkit V1 has been used. (figure 19).

Figure 19. DOIT Esp32 DevKit v1 is used in this project.

ESP32 is a microcontroller manufactured by Espressif Systems Co., LTD. There are several reasons why ESP32 is chosen, instead of microcontrollers manufactured by NXP, STM, and Texas Instruments.

Firstly, ESP32 is popular among the DIY community, as the system is an open-source community project, utilizing this controller allows the system can be easily developed further, even by non-professional developers. The second reason is ESP32 is an affordable duo-core microcontroller with WLAN, Bluetooth, and Sensors built-in, which suitable for the requirement of this project. Finally, despite the popularity in the DIY community, ESP32 comes with the ESP-IDF framework built on top of FreeRTOS, which provides powerful tools for professional development. In other words, professional developers could work comfortably with this choice: i.e., having, for example, full control and debugging easily.

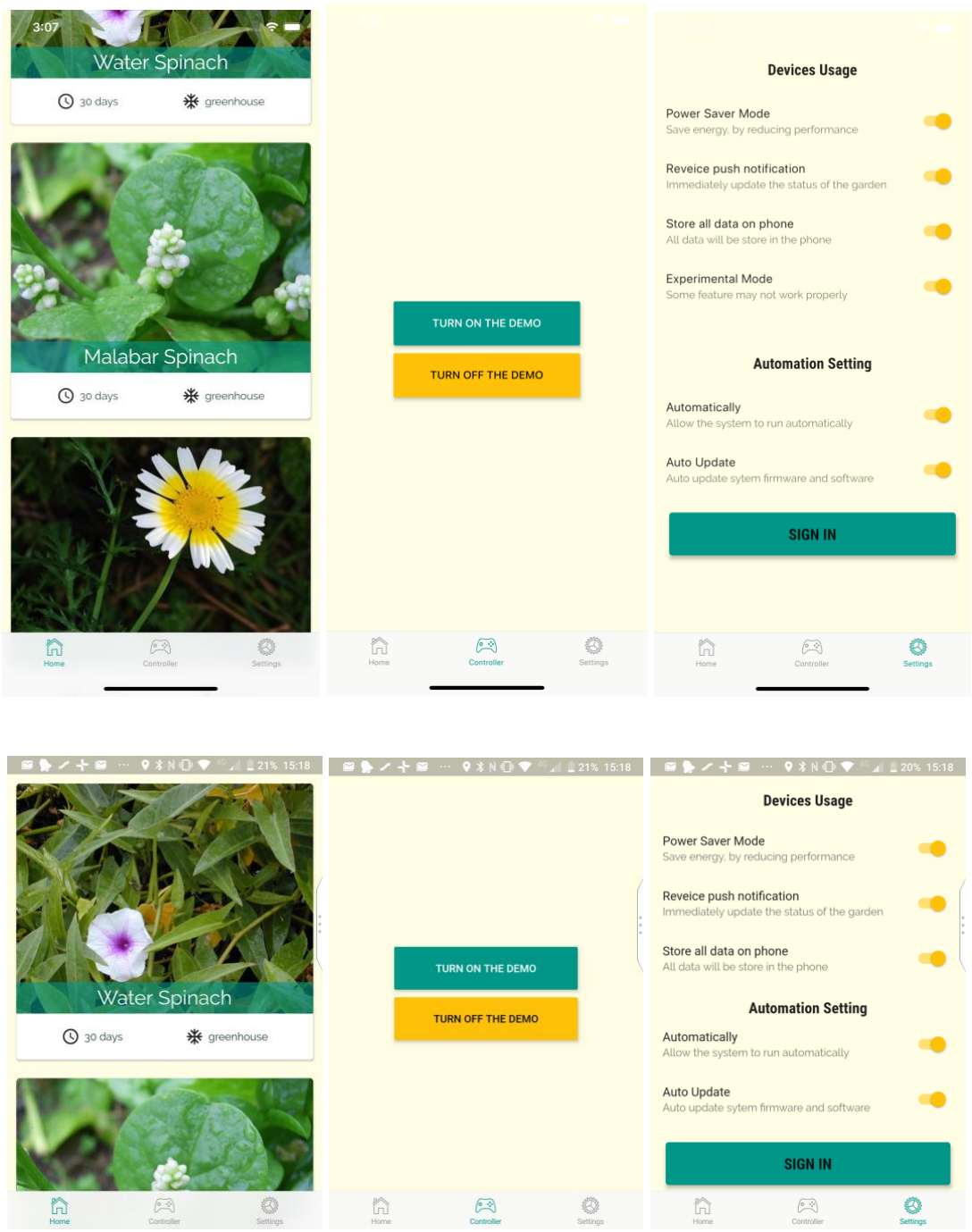


Figure 20. A Flutter App on iPhone 11 Pro Max (iOS) and Blackberry Key 2 (Android)

Most importantly, the whole system interfaces with users through a mobile application. At first, the React Native framework has been used to develop this application since the writer is familiar with React – which is a powerful library to build web interfaces. However, during the development, several issues occur. First, the performance is not good, even React Native is much better than Web App, the framework's complexity slows down the performance. Second, debugging in React Native is complicated, there are bugs that are

difficult to track. Finally, the most important reason is that a better solution, namely Flutter, already exists.

Flutter framework has React Native inspiration but does not have React Native's drawback. The framework's design is efficient and allow the app to be compiled to the machine code and run nearly as fast as the native apps. In addition, there are pre-built widgets that allow the application to be built quickly. Besides, the app provides good user-experience with stability, and easier to debug, compare to React Native. In short, with Flutter, not only an app can be easily built, but also it is also eye-catching and powerful. Figure 20 shows the UI of the Flutter app for both iOS and Android. This is an app that has been built in this project.

5.4 Implementation Prerequisites

Before starting this project, besides purchasing all necessary hardware components, a variety of prerequisites are required to be fulfilled before the development process can be started.

Working Integrated Development Environment (IDE) tools

Three different IDEs, i.e., Visual Studio, Android Studio, and Emacs has been chosen in this project. Unlike other projects which involve only one technology, this project combines different technologies, such as embedded system, web development, and mobile development. Different IDE is suitable for different technologies, besides, separate these working environments could reduce confusion during the development process.

These are personal choices. I chose the Android Studio for mobile app development since it supports Flutter SDK very well. Visual Studio is chosen to build the web application. In addition, Emacs is chosen for embedded programming on the ESP32 because it is convenient to use the terminal for writing, compiling, and debugging the C code. Besides, emacs is utilized to write code and config files on the server through SSH.

In addition to the IDE, the latest version of Mac OS and XCode are required to build the iOS application. The XCode version should be 2 versions ahead compared to the iPhone/iPad in order to deploy the app. During this project, I have made a mistake when

using the Mac OSX Mojave 10.14.6. This does not support the latest XCode. Since it is risky to update the system in the middle of the development, the app could not be deployed on the real iPhone using this machine. Fortunately, the virtual engine works, and app could be deployed using another Mac OS machine.

Installing Espressif IoT Development Framework (ESP-IDF) on Mac OS

- Different toolchains are required to be installed to support ESP-IDF. Depend on the OS, the requirements are different. On Mac OS, the installation can be done as follow:

```
$ sudo easy_install pip
$ pip install –user pyserial
$ sudo port install cmake ninja dfu-util
```

- After the tools have been installed, change directory to home and clone ESP-IDF framework from GitHub:

```
$ cd ~/esp
$ git clone --recursive https://github.com/espressif/esp-idf.git
```

- After the download completed, install the ESP-IDF:

```
$ cd ~/esp/esp-idf
$ ./install.sh
```

- Finally, add ESP-IDF to the PATH environment variable by using the automatic script. This script needs to be run every time the terminal restarts.

```
$ . $HOME/esp/esp-idf/export.sh
```

Installing Flutter Development Kit on Mac OS

- It is simple to install Flutter. First, download the latest stable release from the Flutter download page and extract the file to the desired location. After that, add the flutter tool to the path using the export command as follow:

```
$ export PATH="$PATH:`pwd`/flutter/bin"
```

- Download and install XCode from the App Store. XCode may requires users to update to the latest OS version. In the case installing the newest version is not possible, go to the Apple Developer page to download XCode. Please be mindful that Flutter may not work with the old XCode version. Check the documentation for more detail. After the installation, configure XCode as following in the terminal:

```
$ sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer

$ sudo xcodebuild -runFirstLaunch
```

- Finally, install Android Studio by simply download it from the official website and drag it to the Application folder on Mac OS. After installing, open Android Studio and follow the instruction to install Android SDK. By opening the Flutter source, Android Studio will automatically detect it and request for new packet install, accept the request.

- Check if all setup has been done successfully:

```
$ flutter doctor
```

Setting up Digital Ocean Server

To create a digital ocean account, open https://cloud.digitalocean.com/login, click Sign up, and follow the online instruction. After the registration, log in to the administration panel, create a new Droplet by select Create/Droplets and follow online instructions.

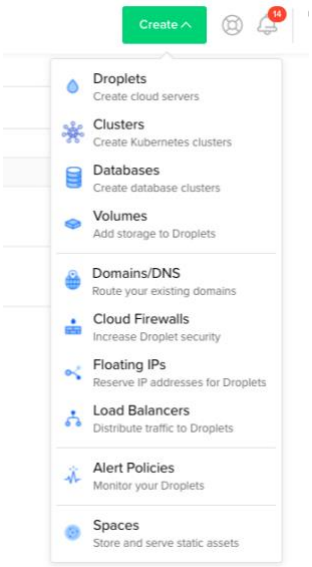


Figure 21. Create the Droplets on Digital Ocean

After that, the initial server setup is required. One needs to create a new user account and the user to the sudo group. The root account should not be used in ordinary tasks.

- Logging in as root using SSH and change the password:

  ```
  $ ssh root$droplets_ip_address
  ```

- Adding new Linux user, for security reason:

  ```
  $ adduser username
  ```

- Adding new user to the sudo group:

  ```
  $ usermod -aG sudo username
  ```

- Setting up the basic Firewall to allow OpenSSH:

  ```
  $ ufw allow OpenSSH
  ```

  ```
  $ ufw enable
  ```

- Check the ufw status:

  ```
  $ ufw status
  ```

  Output:

```
Status: active

To                         Action      From
-                          ------      -----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

Other Prerequisites

The working environment is ready now, however, other things should be set up as well. The working process involves soldering and measuring voltage. A good soldering station and a multimeter should be prepared. Besides, an oscillator is another tool that can bring benefits during the working process. Finally, one can prepare several smartphones for real tests.

5.5 Build and Deployment Process

Install MQTT broker on Digital Ocean Server

- Update Ubuntu's package and install Mosquitto:

```
$ sudo apt-get update
$ sudo apt-get install mosquitto mosquitto-clients
```

- Configure MQTT for testing, add the default config file:

```
$ sudo emacs /etc/mosquitto/conf.d/default.conf
```

- Since this is just a test server and a prototype, configure the default post and IP address simply by adding the following line to the end of the <default.conf> file and save it. This lets the moquitto server knows that it should listen for request that comes from port 1883, which is the default TCP/IP unencrypted port:

```
listener 1883 <server.ip.address>
```

- Allow port 1883 to be accessed. In production, a secured authentication with user and password must be set up, besides, SSL/TTL certificate need to be configured and port 8883 can be used:

```
$ sudo ufw allow 1883
```

Output:

```
Rule added
Rule added (v6)
```

- Restart the Mosquitto to update setting:

```
$ sudo systemctl restart mosquitto
```

- Test to make sure that it works:

```
$ mosquitto_pub -h <server.ip.add> -t test -m "hello, world" -p 1883
```

Build and Flashing the ESP32 Development Board

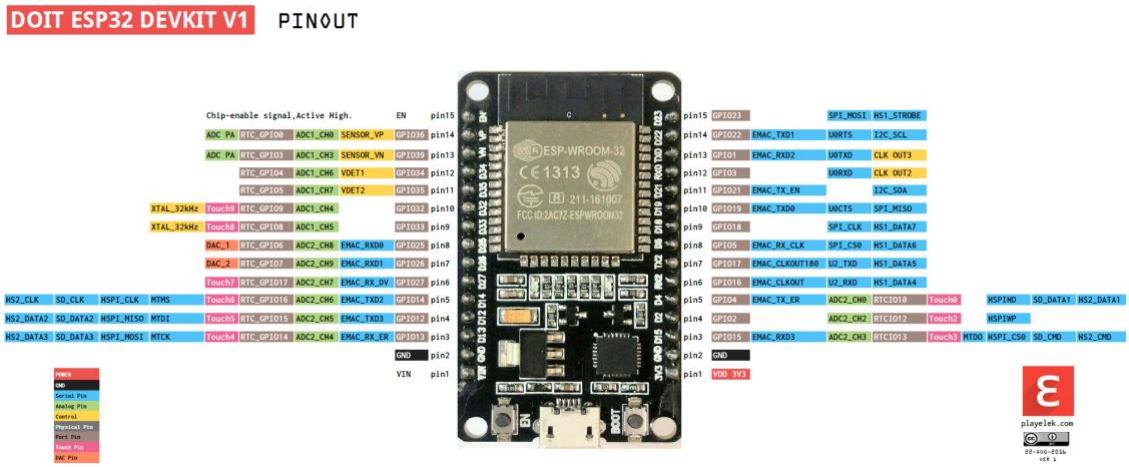


Figure 22. DOIT Esp32 DevKit v1 pinout.

- After connecting ESP32 with Sensor and Actuator. Go to the project root directory, for example:

```
$ cd ~/esp/zocho-ten-sensor
```

- Connect the ESP32 to the Mac OS through the USB connection, set the target with the following with the first command and open the configuration window:

```
$ idf.py set-target esp32
$ idf.py menuconfig
```

- A configuration window will be shown to allow the developer to config the program. This menu allows the developer to set up the project-specific variables, i.e. WLAN network, processor speed, etc. If all variables have been set up in the source code, simply skip the menu.

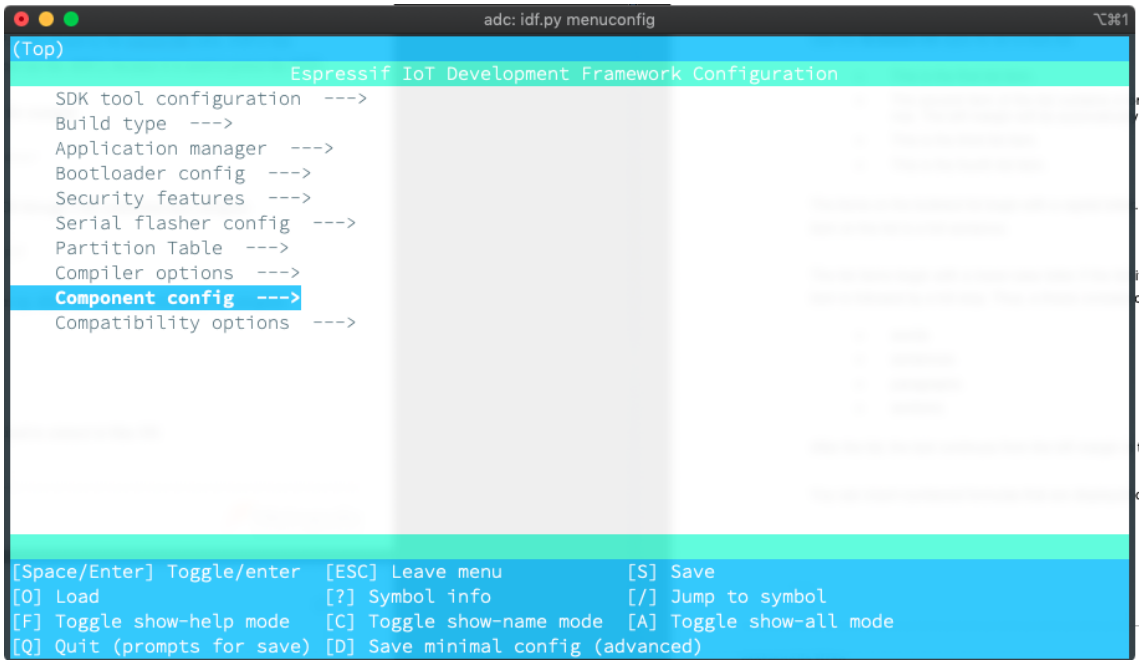


Figure 23. Menu Config Windows of ESP-IDF framework.

- Build the project:

```
$ idf.py build
```

- Identify the Port that ESP32 utilized to connect to Mac OS:

```
$ ls /dev*
```

- After identifying the port, save it to an environment variable:

```
$ ESP32_PORT='enter the above address'
```

- Flash the program to ESP32:

```
$ idf.py -p $ESP32_PORT flash
```

- Monitor the program:

```
$ idf.py -p $ESP32_PORT monitor
```

- Alternatively, the program can be monitor after flashing.

```
$ idf.py -p $ESP32_PORT flash monitor
```

Deploy Flask to Digital Ocean Server, setup MongoDB database, Nginx, uWSGI:

- Follow this instruction on how to deploy a Flask app to Digital Ocean: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgi-and-nginx-on-ubuntu-18-04

Build and deploy the mobile application

- The instruction can be found in the Deployment section in the Flutter documentation: https://flutter.dev/docs,
- Source codes can be downloaded from the link provided in the appendix.

6 Conclusion

Smart systems unite different fields of information technology such as embedded systems, networking, IoT, mobile solutions, and web development to form a smart ecosystem that leverages the advantages of these fields. Smart systems unite knowledge from other industries with the help of cloud computing, data mining, and artificial intelligence, to create a truly smart ecosystem.

This thesis presents the modern architecture of smart systems, consisting of four main layers: devices, gateways, engine processor, and application. This study also discusses how engineers could use this architecture to build a system that is platform-independent, powerful, and easy to maintain.

This study focuses on different mobile cross-platform technologies such as React Native, Flutter, Xamarin, and Qt. It studies the advantages and disadvantages of each technology. Flutter was chosen to be the mobile solution for the use case demonstration – the Community Smart Garden Ecosystem. Using frameworks such as Flutter could reduce half of the development time because it allows an application to be written once instead of multiple times.

At the end of the thesis, the implementation and decision-making process has been documented. The prototype is ready to be tested in the real environment, it shows great potential to be developed further and it will be developed further in the future. Due to the constraints of the thesis report, the source codes and more details are provided in the GitHub repositories mentioned in the appendix.

To conclude, this smart system design and technologies show great potential to solve the mentioned problems: how to build a smart system that is quick, powerful, and easy to maintain.

References

1 David Mercer. Global Connected and IoT Devices Forecast Update. Available from: https://www.strategyanalytics.com/access-services/devices/connected-home/consumer-electronics/reports/report-detail/global-connected-and-iot-device-forecast-update

2 Worldometer.World Population. Available from: https://www.worldometers.info/population/world/

3 Statcounter GlobalStats. Operating System Market Share Worldwide. Available from: https://gs.statcounter.com/os-market-share/all/worldwide/2019

4 Daniel Beverungen, Martin Matzner & Christian Janiesch. Information system for smart services. Available from: https://link.springer.com/article/10.1007/s10257-017-0365-8

5 Smart Systems primer. Available from: https://www.express-ca.eu/public/ecosystem-knowledge-gateway/references/references-smart-systems-primer/references-smart-systems-primar

6 Michael E.Porter and James E. Heppelmann. How Smart, Connected Products are Transforming Competition. Available from: https://hbr.org/2014/11/how-smart-connected-products-are-transforming-competition

7 6 Business Model shifts to explore. Available from: https://www.businessmodelsinc.com/business-model-shifts-blog/

8 Martin H. Weik. The ENIAC Story. Aavailable from: https://web.archive.org/web/20110814181522/http://ftp.arl.mil/~mike/comphist/eniac-story.html

9 San José State University, The History of Transistor. Available from: https://www.sjsu.edu/faculty/watkins/transist.htm

10 TMS 1000 One-Chip Microcomputers. Available from: https://web.archive.org/web/20050213130208/http://www.ti.com/corp/docs/company/history/tms.shtml, accessed 03-05-2020)

11 Intel's first microprocessor. Available from: https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html

12 Evan Andrews. Who Invented the Internet. Available at: https://www.history.com/news/who-invented-the-internet

13 Dave Evan. The Internet of Thing, How the Next Evolution of the Internet is changing everything. Available at: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

14 SSI Smart Systems Integration. Available from: https://www.smart-systems-integration.org/ssi-smart-systems-integration

15 Gartners says a typical family home could contain more than 500 smart devices by 2022: https://www.gartner.com/en/newsroom/press-releases/2014-09-08-gartner-says-a-typical-family-home-could-contain-more-than-500-smart-devices-by-2022

16 Multiple Authors. System Architecture for a Smart University Building. Available from: https://www.researchgate.net/publication/221078634_System_Architecture_for_a_Smart_University_Building

17 Multiple Authors. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165453/

18 Calum McClelland. IoT Device Management: What is it and why do you need it. Available from: https://www.iotforall.com/what-is-iot-device-management/

19 MQTT wiki page. Available from: https://github.com/mqtt/mqtt.github.io/wiki

20 Multiple Authors. The Constrained Application Protocol (CoAP). Available from: https://tools.ietf.org/html/rfc7252

21 The WebSocket API. Available from: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

22 Welcoming Progressive Web Apps to Microsoft Edge and Windows 10. Available from: https://blogs.windows.com/msedgedev/2018/02/06/welcoming-progressive-web-apps-edge-windows-10/

23 Introduction to Progressive Web App Architectures. Available from: https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures

24 A Guide to Mobile App Development: Web vs. Native vs. Hybrid. Available from: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/

25 What web can do today. Available from: https://whatwebcando.today/

26 Lorenzo Sciandra. The new React Native Achitecture Explained. Part Three. Available from: https://formidable.com/blog/2019/fabric-turbomodules-part-3/

27 Farhan Thawar. React Native is the Future of Mobile at Shopify. Available from: https://engineering.shopify.com/blogs/engineering/react-native-future-mobile-shopify

28 Evan Bacon. Making Desktop apps with Electron, React Native, and Expo. Available from: https://dev.to/evanbacon/making-desktop-apps-with-electron-react-native-and-expo-5e36

29 Core Components and Native Components. Available from: https://reactnative.dev/docs/intro-react-native-components

30 Performance Overview. Available from: https://reactnative.dev/docs/performance

31 Flutter vs Native vs React-Native: Examing performance. Available from: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

32 Introduction to widgets. Available from: https://flutter.dev/docs/development/ui/widgets-intro

33 State management. Available from: https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro

34 Tom Glider. Flutter's Key Difference: Owning Every Pixel. Available from: https://medium.com/flutter-community/flutters-key-difference-owning-every-pixel-e2135b44c8a

35 The Good and The Bad of Xamarin Mobile Development. Available from: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/

36 Stock Exchange Release April 12, 2017. Available from: https://investors.qt.io/releases/?release=A84F7B0247145FCF

37 Examples of Mobiles Apps in Qt. Available from: https://developex.com/blog/%F0%9F%86%92-examples-of-mobile-apps-in-qt/

38 5 types of Software Licenses you need to understand. Available from: https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/

Appendix: Source Code and Documentations

- **REST API, Flask:** https://github.com/juuisle/zocho-api
- **Mobile Application, Flutter:** https://github.com/juuisle/zocho-mobile
- **Actuator, ESP-IDF FreeRTOS:** https://github.com/juuisle/zocho-actuator
- **Sensor, ESP-IDF FreeRTOS:** https://github.com/juuisle/zocho-sensor