D4 Individual Report

Benjamin Sanati
bes1g19@soton.ac.uk
Personal Tutor: Dr Yoshishige Tsuchiya

Abstract: This report outlines key areas of an IoT based fitness program I helped to design and produce, detailing my contribution to the project. The report includes a summary of my individual progress during the D4 design project, portraying my technical and team management accomplishments in the duration of the project, along with a reflection and critical analysis of my achievements.

1. Introduction

In this report, I will outline my contribution to the D4 design project, along with a detailed description of the specification of the system modules I was tasked with designing, developing and testing. I will also outline major communication protocols and security topology that were required such that I could make a fully functioning prototype of the sub-modules I was tasked with producing. I will then reflect on my team management and team working abilities during the project along with a critical evaluation of the project as a whole.

2. Contribution

In this project, I was tasked with multiple technical challenges including; designing and producing the server, producing the website GUI, JSON data handling, integration of the server into the system, digital processing of the Hall effect sensor. I also acted as the team leader and therefore had multiple team management and form/trade fair responsibilities alongside the technical challenges that I faced.

3. Specification

In this project, I was tasked with designing; a server, the basic GUI for the 'my tree' and 'our garden' pages, the data handling of client data over the server, the digital processing for the onboard hall effect sensor on the ESP32 which is used on the secondary device sub-system and the integration of the server to the clients and website.
The server had to interact with all devices on the network, such that the workout notification functionality could be realised. I, therefore, had to ensure that the secure server operated on an asynchronous, full-duplex, bidirectional communication system.
The basic GUI for the 'my tree' and 'our garden' pages that I was required to produce were identical. They included; an aesthetically pleasing navigation bar which was easily visible and interactive with the user, along with the background of the website, which had to give a pleasant experience.
The data handling had to ensure; that client data being transmitted over the server was secure, and that client data was transferred in a concise and legible (approved) method.
The onboard hall effect sensor had to work such that magnet proximity could be registered and result in 10 points being incremented for every 10 registered repetitions of close magnet proximity. It had to be designed such that it could be integrated into the secondary device sub-system with ease.

4. Design and Simulation

4.1. Server

I looked at two potential communication protocols to realise the server: MQTT and HTTP with web-sockets. HTTP uses a simple POST/GET function to update the website and could be easily coupled with a database to store the data being transferred, This could not realise the workout notification functionality due to its inability to supply a direct remote connection between all users in the system. MQTT uses full-duplex, bidirectional data transmission which can handle the workout notification functionality requirement between users and has a very easy to use data identification method, due to topics.
I then had to investigate MQTT brokers. I looked at mosquito, azure, google and AWS. I settled with AWS due to the Hornbill IoT open-source library[1] (press here to see library), which increases the ease of connectivity to AWS from the ESP32. AWS IoT's MQTT broker supplied a reliable connection to the ESP32's and allowed me to host the website over the cloud with the use of Node-Red.

Its simple pub/sub protocol allowed for the creation of a direct remote connection between all users and the website, enhancing the potential for expansion of our product to accommodate more users.

AWS was setup by producing a policy to which all the 'things' could subscribe to. This policy set-out the structure in which data would be transmitted from the ESP32 to the AWS MQTT broker. The things were then created which each had their own PEM encoded certificates and private keys for security. The contents of these certificates and private keys were placed within the 'aws_iot_certficates.can file in the AWS_IoT Hornbill library for the ESP32. The ESP32 then used my AWS endpoint to interact with the broker, along with calls to methods set up in the library to connect, disconnect to AWS or pub/sub to topics on the server. This integrates the ESP32s to the server and "views" each individual device as its own client (to the server, the primary and secondary devices which each user has been seen as individual clients). For the final product, there were a total of 5 topics which were used to identify the level user1 or user2 were on (example topic: 'User2PD/user2/level') or to identify the total amount of points the users had acquired (example: 'User2PD/user2/points'). All JSON data was transferred in the following format: {"state": {"user1": {"level": "4"}}} (an example of user 1 getting to level 4), with the 'level' being changed to 'points' for all of the points being transmitted.

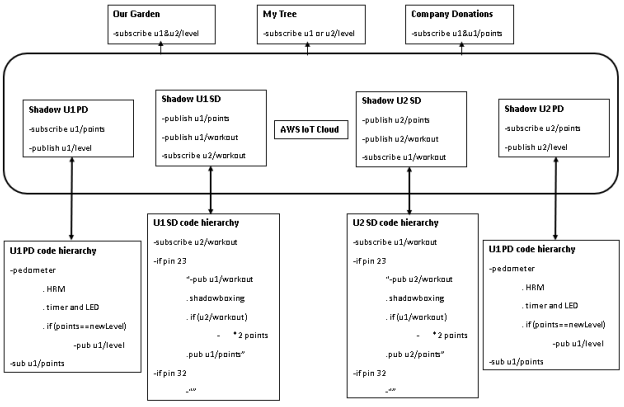

Figure 1: The full System Diagram I designed

I designed the layout for the U1PD, U1SD and U2SD .ino files such that they could all connect to AWS and pub/sub to the correct topics for the specific devices. The integrated sensor code could then just be copy and pasted into the .ino files to fully integrate the system.

4.2. Website GUI and Data Handling

The website was designed by a sub-team of two people (myself included). We had a series of team calls to discuss the layout of each website page and I used the Paint drawings created to create the basic GUI for the website. All images for the background were acquired on 'https://pixabay.com/' which is a copyright-free, image providing website. The background consists of 6 images which the animates like a slideshow. This was achieved with the 'keyframes' feature. The navigation bar was then implemented in white and the 'hover' feature provided user interactivity.

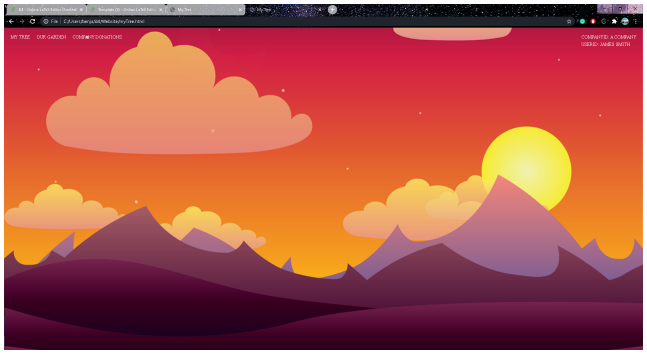

Figure 2: Basic GUI Website

Node-Red was used to handle the JSON data being transferred from the users to the website via function and template nodes. The functions parsed the JSON data into integers, saved the variables in context and set them as the message payload to the template nodes using JavaScript. The template nodes would then use JavaScript functions to determine which avatar image had to be displayed on the website.

```
if(msg.topic == "User1PD/user1/level") {
    p = JSON.parse(msg.payload);
    if(p == null) {
        level1=context.get('level1') || 0;
    } else {
        level1=p.state.user1.level;
    }

    msg.payload.level1=level1;
    context.set('level1', level1);
    return msg;

} else if(msg.topic == "User2PD/user2/level"){
    p = JSON.parse(msg.payload);
    if(p == null) {
        level2=context.get('level2') || 0;
    } else {
        level2=p.state.user2.level;
    }

    msg.payload.level2=level2;
    context.set('level2', level2);
    return msg;

} else {
    level1=context.get('level1') || 0;
    context.set('level1', level1);
    msg.payload.level1 = level1;

    level2=context.get('level2') || 0;
    context.set('level2', level2);
    msg.payload.level2 = level2;
    return msg;
}
```

Figure 3: Function Node Code

Node-Red was set up such that each node would execute

every time data was passed to the server. Different web-pages were subscribed to different topics, hence the MQTT input nodes are different for each web-page. This, therefore, integrates the ESP32s to the website via the server and thus, fully integrates the server to all clients.

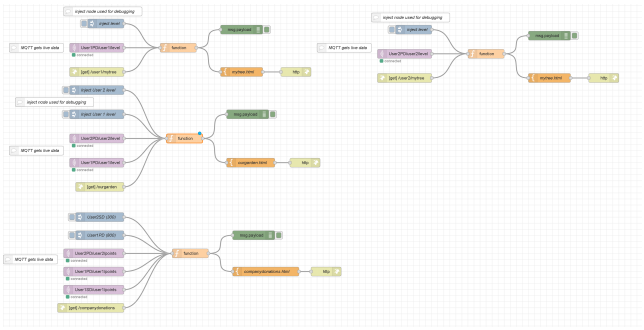

Figure 4: Nodes in Node-Red

To access the website, the IP for node red had to be used along with :1880. Then the user pages can be called by adding a '/user1/mytree'. A full example URL is 'https://18.132.48.233:1880/user1/mytree'.

4.3. Hall Effect Sensor

The ESP32 has a built-in Hall effect sensor. Arduino has a built-in function 'hallRead()' which displays the value from the Hall effect sensor. This can then be used to set limits, which indicate whether a magnet is in close proximity to the sensor or not. Every time a limit was breached, a 'count' variable was incremented and when the 'count' variable reached 10, a 'points' variable would increment by 10.
To debug the hall effect sensor, I used the serial monitor to ensure the count and points values were incremented when necessary, along with toggling the onboard ESP32 LED.

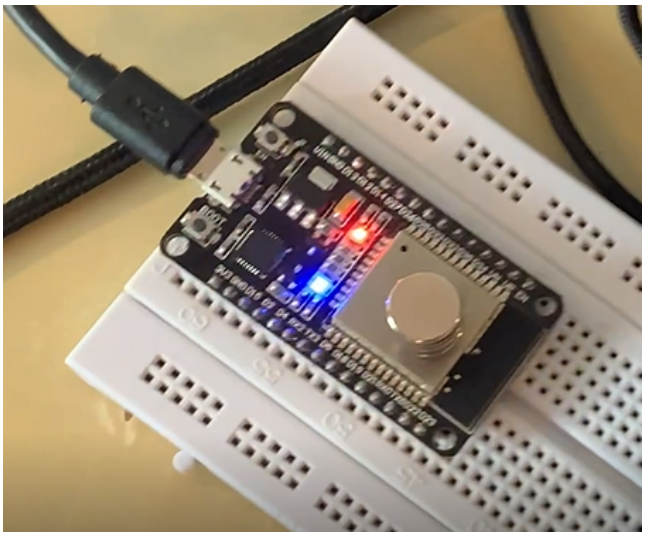

Figure 5: Hall Effect Sensor, Toggling on-board LED

5. Testing and Results

5.1. Server

When the server was initially connected to the ESP32s, I tested to see if data could be published to the server and data could be subscribed to by the ESP on the MQTT test client. When publishing to a topic, the test client would be subscribed to the same topic and I would check that data had been successfully passed from the ESP to the MQTT test client. When subscribing to a topic, the test client would publish to the same topic and I would check that data had been successfully passed from the MQTT test client to the ESP. Vice versa when publishing. Overall, the system worked well, and the transmission of data was ensured by sending all data 3 times.

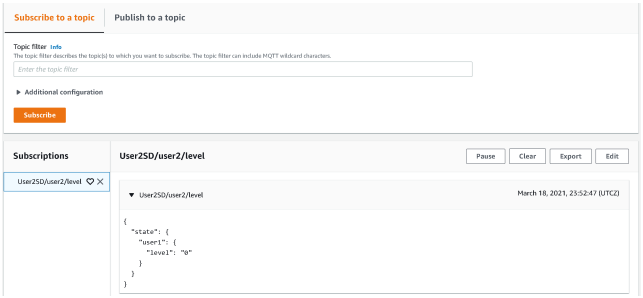

Figure 6: MQTT test Client

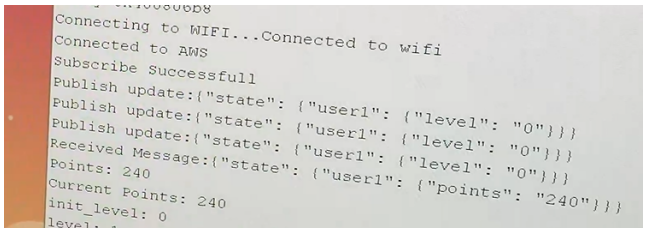

Figure 7: ESP32 Serial Monitor

Prior to producing the final server, I produced a mock server, with much less functionality than the completed product, however, still emulating a client-server-website topology. This was produced to ensure that the completed server could easily be implemented to produce the final product. This practice server consisted of two ESP32s. One was connected to a push button and published to 'LEDSTATE/user1/power' topic, toggling its value, and the other was connected to an LED which was subscribed to the 'LEDSTATE/user1/power' topic. The JSON data being transmitted was in the form {"state": {"user1": {"power": "1"}}} (1 if LED is on, 0 if LED is off). A website was produced, which would also be subscribed to the 'LEDSTATE/user1/power' topic, displaying the state of the LED. After this practice run had successfully been trialled, I moved onto producing the final completed server.

One test was the workout notification method. This ensured that ESP32's pub/sub was functioning correctly. The simulation was carried out by pressing the user 2 secondary device workout button and ensuring that the LED on the user 1 Primary device would toggle high. The same check would then be done but the other way around (user 1 presses workout button and user 2 gets notified on secondary device). This test showed high transmission reliability, with data, again, being transmitted and received over 90% of the time. Due to the increase in devices, the overall system did not transmit and receive data with as high an efficiency as with the individual ESP32. Overall, however, the data transmission was reliable and consistent, and was successfully presented live in the trade fair (working every time).

The next test was the level up method. This ensured that whenever a user would level up, the website would update by changing the avatar image to one of a larger tree. This simulation was carried out by reducing the levelling up points required value on both users devices and levelling up user 1's tree to level 1 and levelling up user 2's tree to level 3. This would then be checked on the 'my tree' and 'our garden' pages to ensure it worked. This test was highly successful, with all tests resulting in a positive result. The data transmission through the system (from the ESP to AWS to Node-Red and finally to the website) could be visualised using various debugging methods (serial monitor, MQTT test client and Node-Red debugging window from debug nodes), all of which showed reliable data transmission across the multiple platforms.

One change that was made to the server after testing was to the AWS_IOT.cpp file in the AWS IoT library. The yield and delay times were changed to 5 and 550 microseconds respectively to improve transmission efficiency.

```
void aws_iot_task(void *param) {

IoT_Error_t rc = SUCCESS;

    while(1)
    {
        //Max time the yield function will wait for read messages
        rc = aws_iot_mqtt_yield(&client, 5);

        if(NETWORK_ATTEMPTING_RECONNECT == rc)
        {
            // If the client is attempting to reconnect we will skip the rest of the loop.
            continue;
        }


        vTaskDelay(550 / portTICK_RATE_MS);
    }
}
```

Figure 8: AWS IoT yield and delay time changes

Overall, my server was efficient, reliable and could perform all requirements stated in the specification.

5.2. Node-Red

Inject and debug nodes were used extensively in the production of the website to ensure that data being transmitted by the server could correctly be processed into an integer. The inject node was used to inject a message payload in the same form as the MQTT client would transmit the data (same as stated above, e.g.: {"state": {"user1": {"level": "4"}}}) on the same topic that the website would acquire the data (using the same example 'User1PD/user1/level') to fully simulate and debug the data that would be transmitted to AWS by the users.

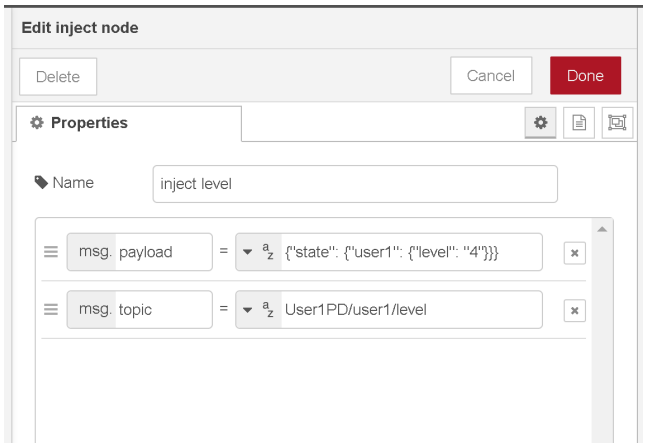

Figure 9: Inject node example

The debug nodes were then used after the function nodes such that I could debug the parsing of JSON values into integers. These same values were passed into the template nodes which display the website, therefore the debug nodes are integral to understanding what data is being passed to the template nodes. The debug nodes solely output the value stored in the message payload. A simple test flow is represented below. This includes the inject node, the function node, the template and http in/out nodes, along with the debug node.

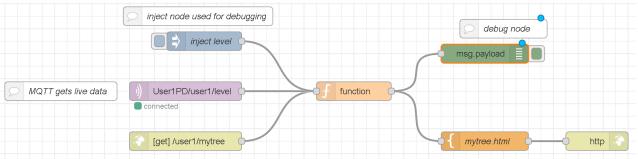

Figure 10: User1 'my tree' Web page flow including testing

These nodes allowed me to efficiently test and debug my function and template node code, such that data could be transferred from the server to the website correctly.

6. Management and Team Working

As team leader, I had the responsibility to ensure the final design met the specification. To ensure a constant progression during the short time period, I ensured that our team had at least one meeting everyday on teams. This quickly made the team comfortable with each other and allowed us to both progress faster and to keep me up to date with progress. In order to effectively manage the team, I always checked in on each sector at end of day to ensure that progression was on track.

I acted as a mediator, organiser and form manager in the team. This was to ensure that I was seen as an equal while also ensuring that progression of the project was constant. Alongside this, I ensured that the team were aware, that any technical disagreements between me and another team member would be discussed as equals and that the entire team would have to decide on a final outcome. I continuously planned ahead so that deadlines were met and constantly had the handbook at hand to ensure that our work was being done above the set standard. As a team, we ensured that each hardware member had their own hardware along with backups with the integration team, to minimise the likelihood of faulty products becoming a major issue in development.

To Partition the work between my teammates, I produced an excel document and let people choose what area of design they wished to go into, depending on their strengths. Projects were therefore assigned in a way to maximise the efficiency and utilise skills of each team member. This ensured all people had there say on what they wanted to do and also ensured that team members believed that they had the ability to accomplish the task being asked of them. Alongside this if teammates told me additional work was required, team rearranging occurred. We partitioned the workload between 4 sub-teams that would all report back to me, updating me on there progress. We stuck to the timetable we had produced in form 2. Rearranging of team members was based on whether I believed that the sub-team was keeping to form 2 or whether they needed an extra hand.

To ensure that the team had a clear view of what had to be created, I produced a full system diagram and code layout file which broke down all of the requirements of each sub-team, such as to avoid confusion later in the project. I also setup a GitLab repository so that code was shared between the team. I ensured that the integration team stated the requirements in terms of sensor interfacing code clearly, so that merging individual pieces of work wouldn't be an issue. Every time a meeting was held, one person in the meeting was tasked with filling in the meetings form on a shared google document that was created to track progress. In order to minimise risks, we started early to find issues with initial design and poured hours of work into research such that development could be as clear as possible.

The entire design specification of our system was met, therefore, I believe I did well as a team leader, however, there is still room for improvement within my leadership and technical abilities.

7. Critical Evaluation and Reflection

Delegation of workload and importance of timing are two of the biggest things that I learned.I may have taken on too many technical responsibilities, considering I was the team leader. In the future, I would delegate more of the technical workload to other team members. I was good at delegating the workload for others, however, I did not delegate the work well enough for myself. Although all of my work was still completed within the timeline set out in form 2, meaning that this did not affect the project as a whole, this caused an excessive burden on myself which can be avoided.

Aside from this, I believe I was a good team leader, who actively listened to the teams input and organised the work to be done by the team well. The fact that we finished, however, could indicate that we were not ambitious enough. This was something I was very wary of at the start. We could have been more ambitious with our choice of microcontroller, using the PSOC, or we could have increased the amount of sensors being used in the system to increase the functionality of the device.

Overall I am pleased with what I designed; I overcame hurdles to get the server working and integrated into the system and believe I managed my team very well. There were many issues that were encountered during the project research and development stages, which were resolved. The fact we managed to overcome all of the projects problems to produce a final working product that matched the specification is an achievement which I praise all of our team for.

8. References

References

[1] Hornbill, "Aws_iot," *Github*, 2017.

[2] Pixabay, "All background images on website," *Pixabay.com*, 2016.