VNU HCMC – University of Science
Faculty of Information Technology
Introduction to IoT

Final project

PPlanet

Automatic pet feeder

Instructor: MSc. Cao Xuan Nam

Group 02
19127067 – Hoang Nhu Thanh
19127142 – Tran Thai Duc Hieu
19127242 – Do Vuong Phuc

1 Table of Contents

2 Group information

Group 02	
Student ID	**Full name**
19127067	Hoang Nhu Thanh
19127142	Tran Thai Duc Hieu
19127242	Do Vuong Phuc

3 Product description

3.1 Introduction

Product name: PPlanet (an automatic pet feeder)

Let you know if your pet is suffering from anorexia.

3.2 Description

PPlanet automatic pet feeder supports

1. Schedule feeding within your device on website.
2. Config wi-fi via soft access point.
3. Log the feeding history on server.
4. Ring and led light to notify that your pets are being feed.
5. Notify via email when the food is running out.
6. Notify via email if your pets leave the food untouched.

4 Input and Output

4.1 Input

- Ultrasonic sensor to measure food remaining in the container
- Pressure sensor to measure how much food the pet has eaten

4.2 Output

- Buzzer to alert when it is feeding
- Led to notify wi-fi status and feeding status
- Servo to open and close the food container

4.3 View input

- View the amount of the rest food in container
- View the last feeding time

4.4 Control output

- User device to schedule feeding and feed manually

4.5 Cloud

- View eating history through website
- Load current schedule setting

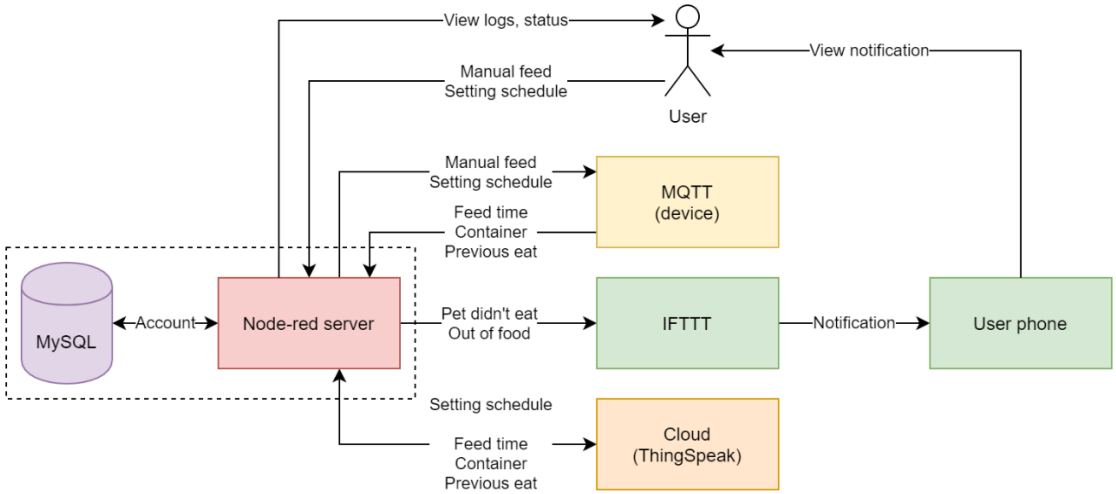
4.6 Notification

- Notification sent to device

4.7 Config wi-fi

- Using soft access point to config wi-fi from smart phone or laptop

5 Data transfer flow



1. Users can get access/log-in to the web via NodeRED server. Users' account will be stored on MySQL database.
2. User can feed manually or set a feeding schedule via NodeRED. Then these data (time config) will be sent from NodeRED to MQTT (our device).
3. When the time is due, or when user press the feeding button manually, MQTT will send log to NodeRED server. The log includes 3 field of information which are the feeding time, how much food remain in the container and did the pet eat its previous meal.
4. Afterwards:
    - The log that had just been sent to NodeRED will be store to cloud (ThingSpeak).
    - If the pet did not eat its previous meal, or if the food is running out, user will have the corresponding notification sent to their mobile via IFTTT.
5. When user config the feeding time on NodeRED, the settings will not only be sent to device (MQTT) but will also be stored on cloud (ThingSpeak).
6. Because feeding logs and settings are stored on cloud (described at 4 and 5), whenever user visit the website, logs and settings can be loaded back on NodeRED so that user can view them.

6 3D design

6.1 Overall

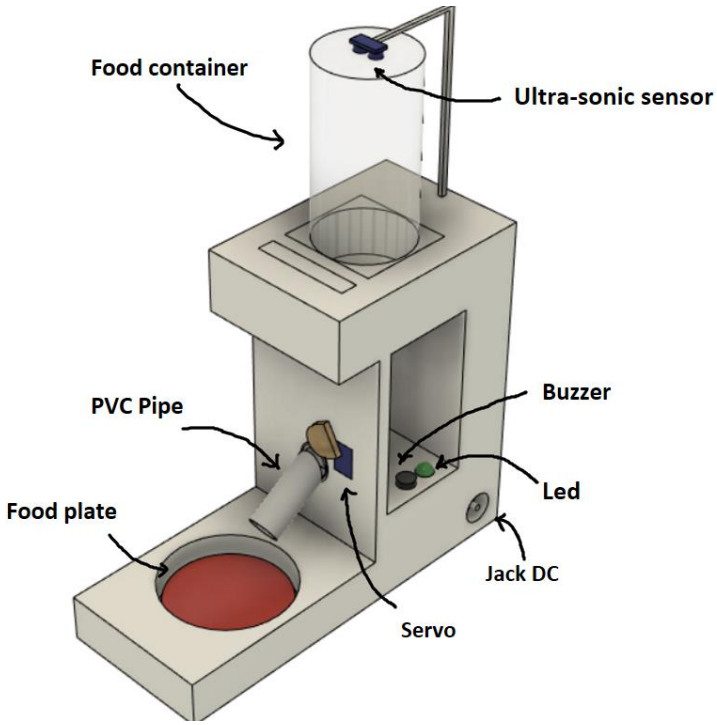


Figure 1. Full 3D sketch

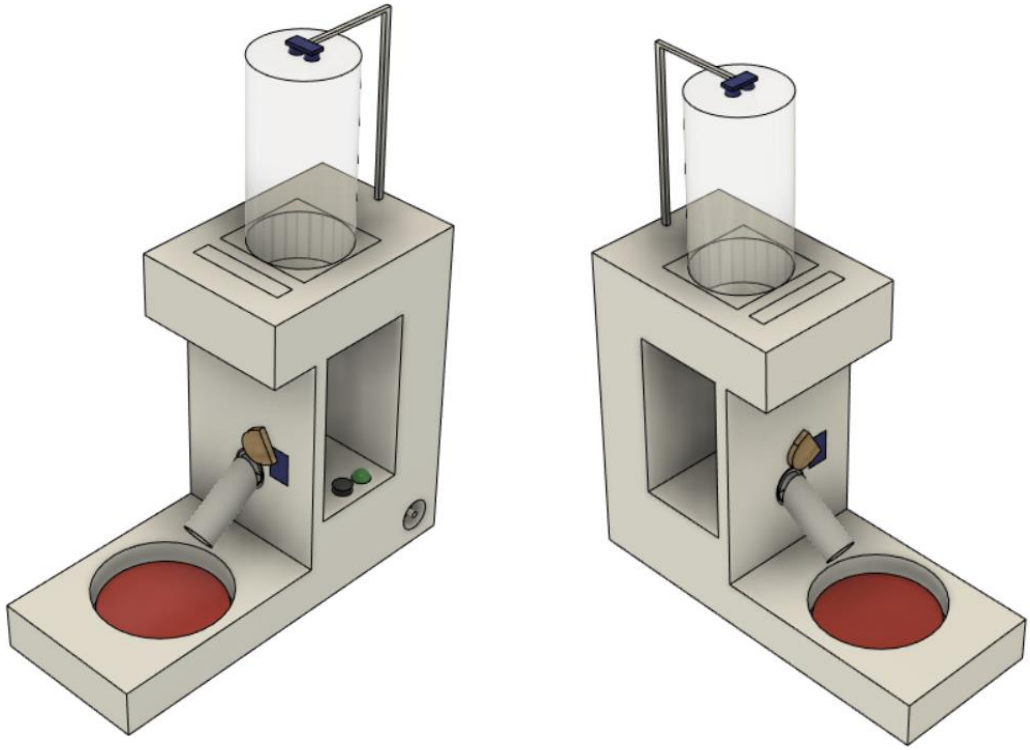


Figure 2. Top-left and top-right view respectively form left to right

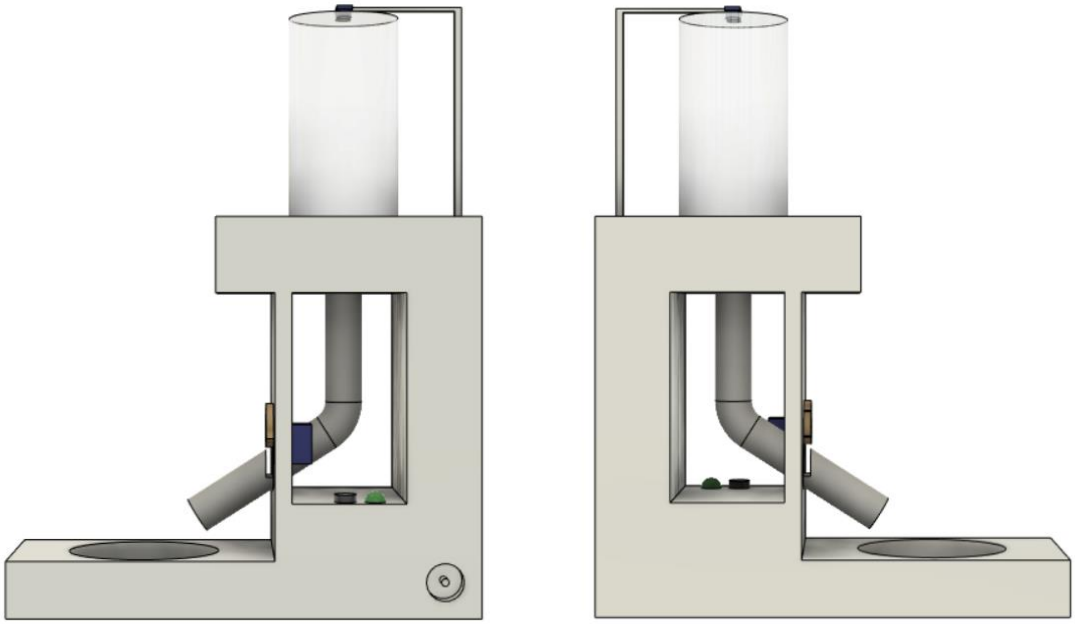


Figure 3. Left and right view respectively from left to right

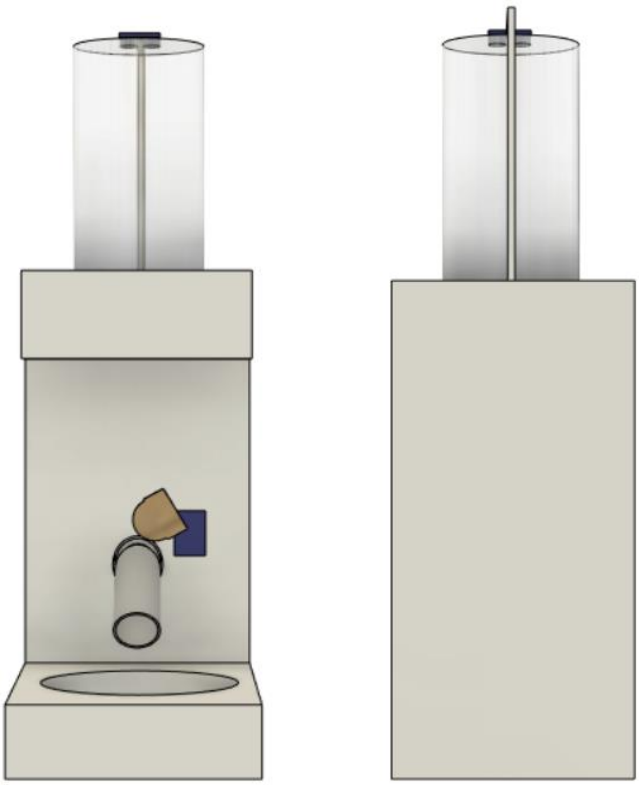


Figure 4. Front and back view respectively from left to right

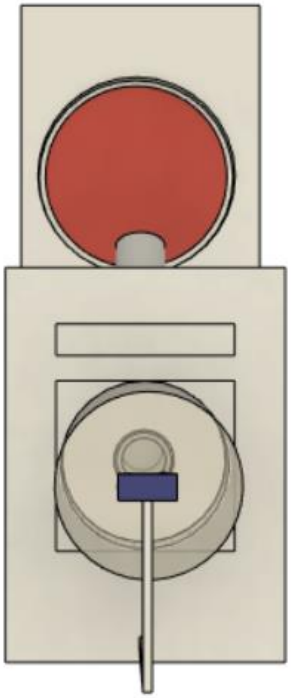


Figure 5. Top view

6.2 Detail 3D sketch of the base of the feeder

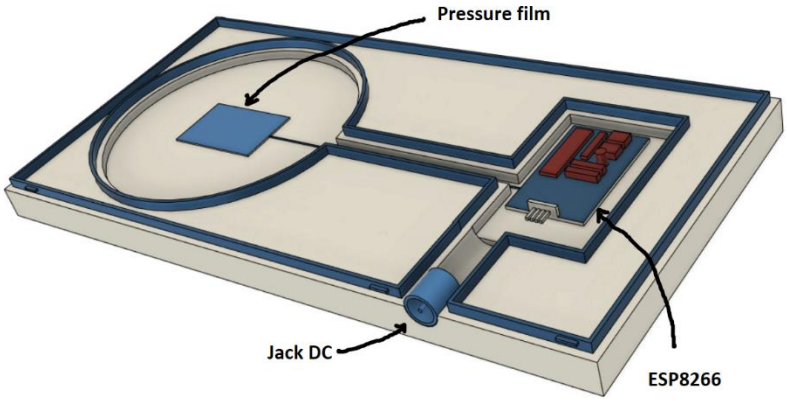


Figure 6. Top-left view

Figure 7. Left view

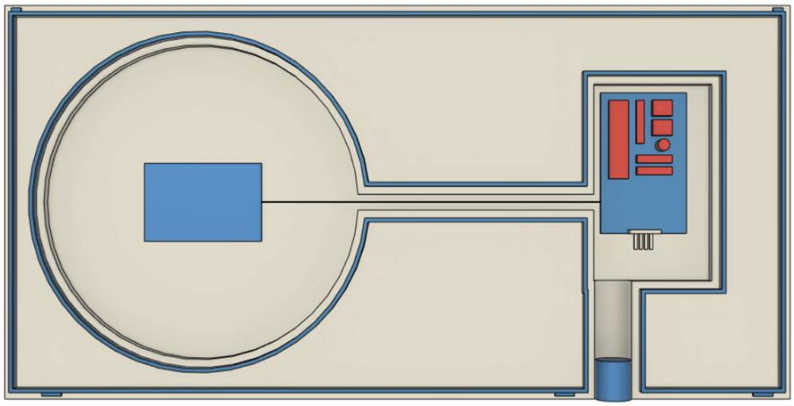
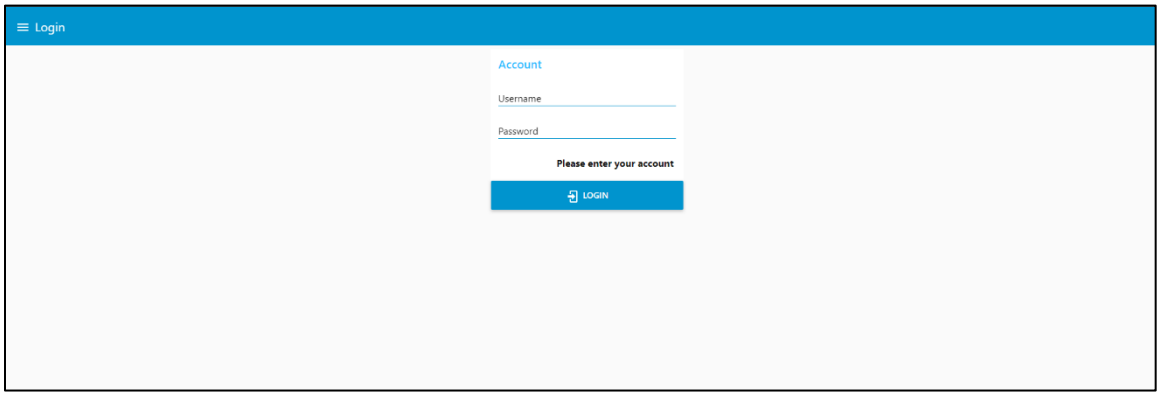


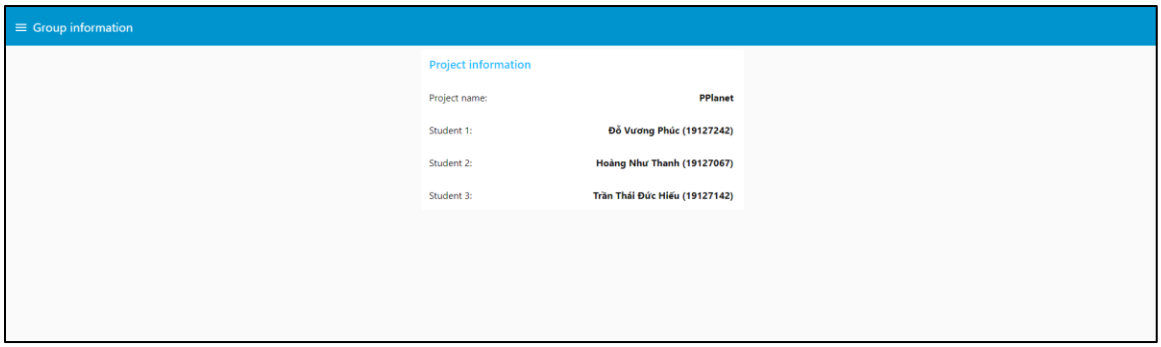
Figure 8. Top view

7 Web design

When user first access to the site, they will see the login screen.



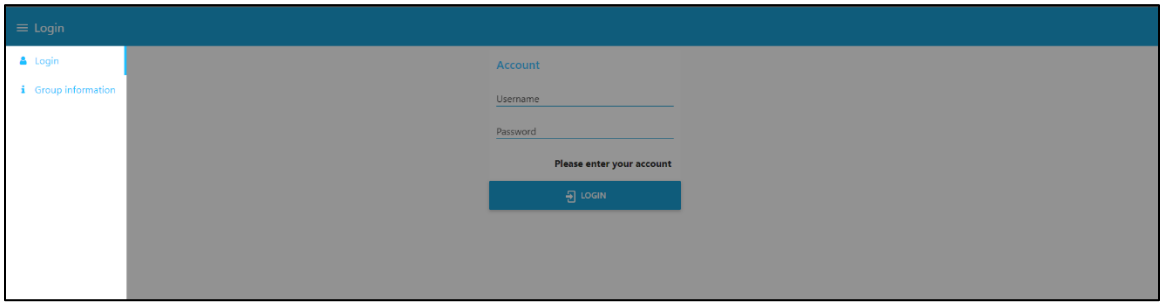
If the user does not login, what they can do is only to view the Group information.



There are **2 kinds** of account: **Admin** and **User.**

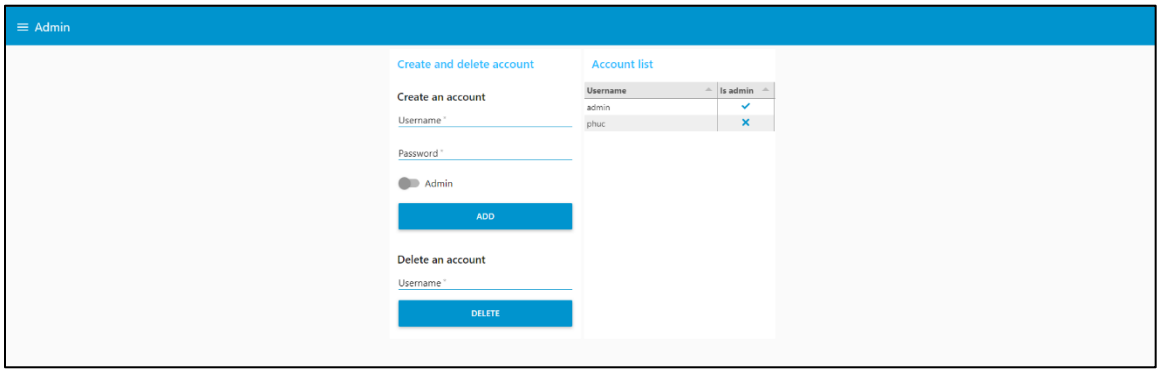
7.1 As a guest

A guest can only view login tab and group information tab.

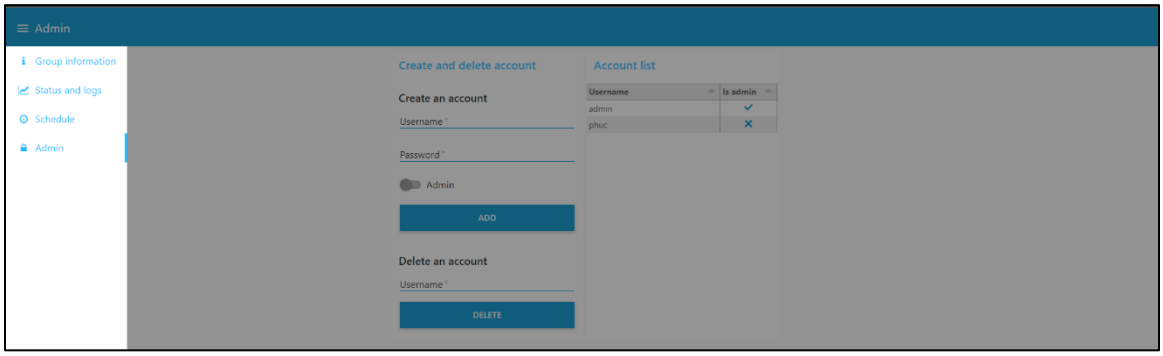


7.2 Login as admin

As an admin, we can add or delete any user. We can also view the account list.

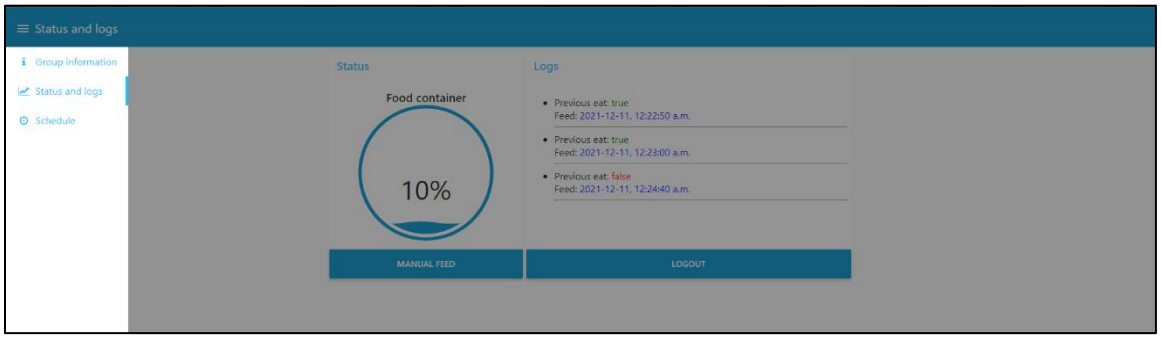


In addition, an admin has all the rights to view and do **everything** a user can, including feeding manually, view status and logs and schedule feeding.

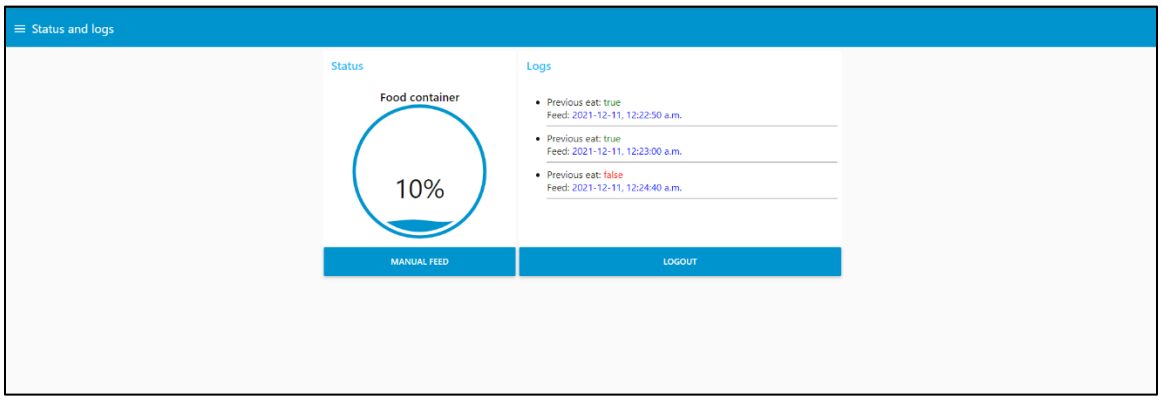


7.3 Log in as user

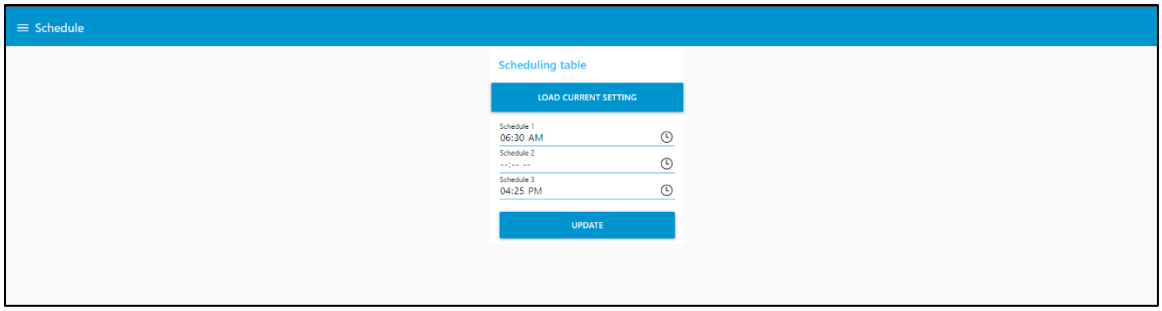
A user can feed manually, view status and logs and schedule feeding.



User can view how much food remaining in the container, feed manually by hitting the "Manual feed" button. Additionally, the logs will be presented as below.

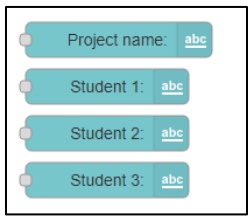


Moreover, as has been introduced, a user can schedule the feeding and also view their previous settings.



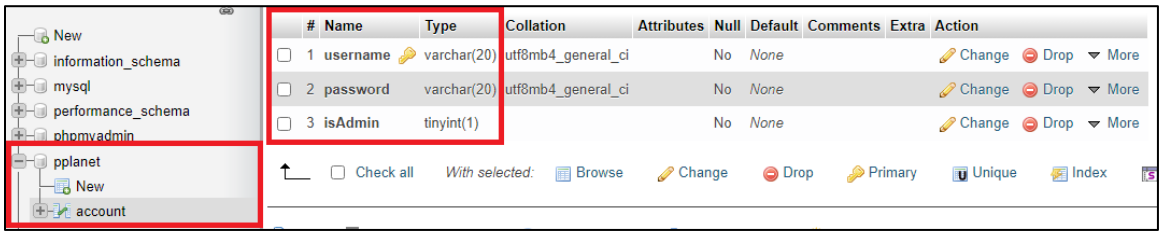
8 NodeRED flow

8.1 Group information tab

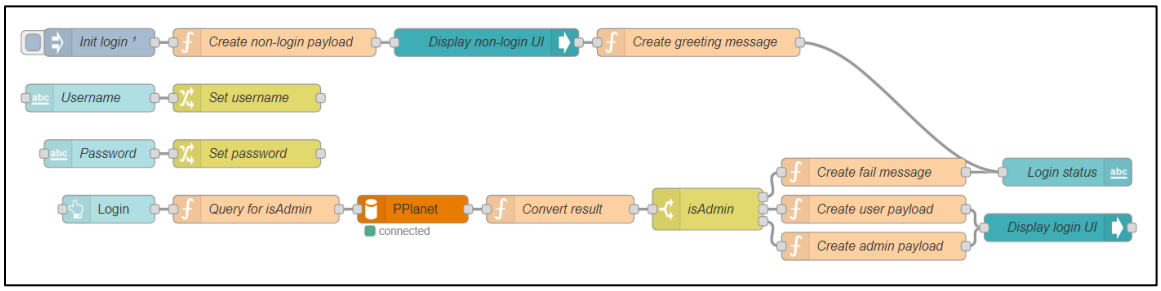


The flow of group information is quite simple, it only contains three text displays in order to show information of group.

8.2 Login tab



As we already mentioned, users' login information are stored in a centralized database system which is MySQL. Here, we create a table contains three fields including username, password and a boolean to check whether user is admin.



For the node-red flow, whenever user open the website, we need to direct to the "non-login UI" which allows them to see only the login tab, and group information only. This is the way use "Control UI" node to redirect:

```
msg = {};
msg.payload = {
    "tab": "Login",
    "tabs": {
        "hide": ["Status and logs","Schedule","Event Logs","Admin"],
        "show": ["Group information","Login"],
    },
};
return msg;
```

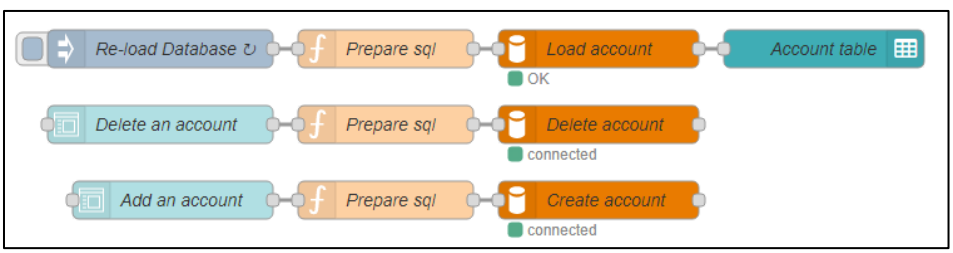
Secondly, when users interact with username text field, or password text field, we just only stored the data in a flow variable. Until they click login, the stored information (username, password) will be taken out, before sending to the MySQL database.

```
msg = {};

username = flow.get('username') || "";
password = flow.get('password') || "";

msg.payload = [username,password];
msg.topic = "SELECT isAdmin FROM ACCOUNT WHERE username=? AND password=?";

return msg;
```

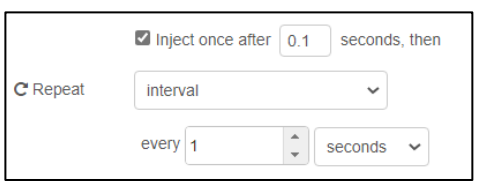
After received the result, we first convert it into $\begin{cases} -1 & (fail) \\ 0 & (user) \\ 1 & (admin) \end{cases}$. Since then, we can use switch case to change message on the screen, or redirect them to a new screen.

```
if (msg.payload[0]!=undefined) {
    msg.payload = msg.payload[0].isAdmin;
} else {
    msg.payload = -1;
}

return msg;
```

8.3 Admin tab



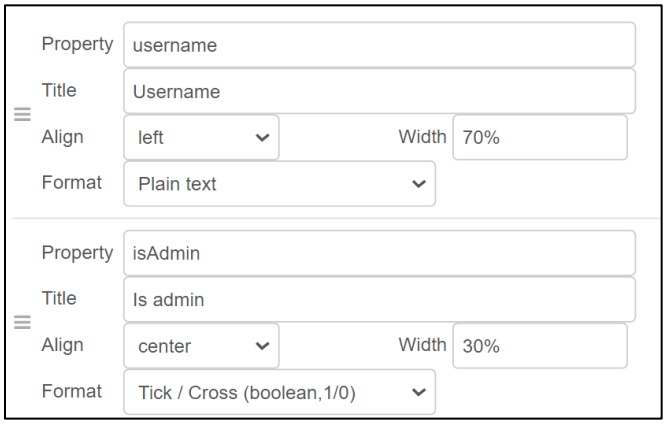
If the logined account is admin type, the website was redirected to admin tab. In this table, the account table is loaded repeatedly after every 1 second.



Instead of loading all users' information, we just only query for the username and the account type.

```
msg.topic = "SELECT username, isAdmin FROM ACCOUNT";
return msg;
```

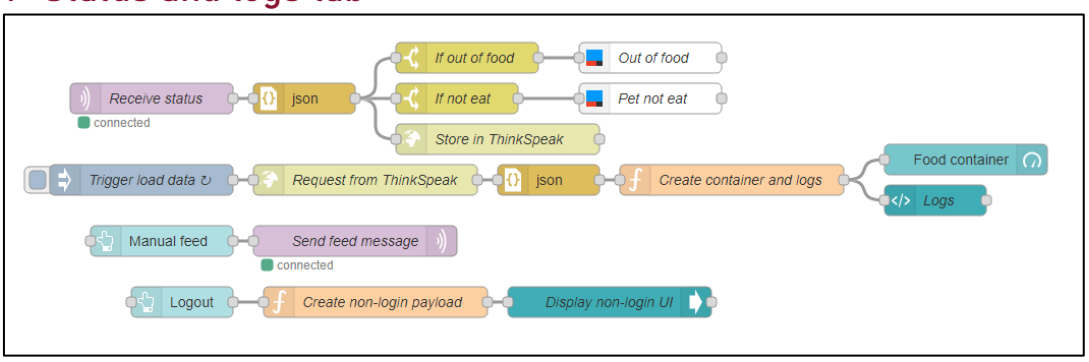
Then display all the receive records within two columns:



For create account and delete account features, we use two forms provided by Node-red UI dashboard. From those information, we create a query string. Although we can create query string by concatenating them, we parameterized the input in order to avoid being SQL injected.

```
payload = msg.payload;
username = payload.username;

msg.payload = [username];
msg.topic = "DELETE FROM ACCOUNT WHERE username=?"

return msg;
```

8.4 Status and logs tab



In this tab, we catch information packages from MQTT Explorer (or device) and convert them to json object. When package comes, we store it on cloud, and check whether do we need to notify users. If the system is out of food, or your pet did not eat previous meal, the IFTTT node will send a notification.

In the same way of loading account list on the above section, we load our log on cloud continuously after 1 second. We just only load 20 records instead of loading all of it. From the loaded data, we manipulate them into the suitable message.

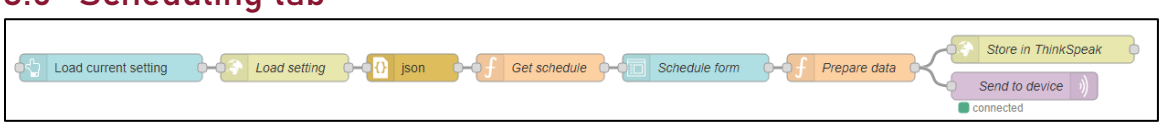
```
feeds = msg.payload.feeds;
nFeed = feeds.length;
msg = {}

if (nFeed>0) {
    msg.container = feeds[nFeed-1].field2;
} else {
    msg.container = 0;
}

log = [];
for (let i in feeds){
    date = new Date(parseInt(feeds[i].field1));
    eatValue = feeds[i].field3;
    if (eatValue=="true") {
        color = "green";
    } else {
        color = "red";
    }
    log.push({
        "time": date.toLocaleString(),
        "eat": eatValue,
        "color": color
    });
}

msg.log = log;
return msg;
```

After converted, we display logs by using template node:

```
<ul>
    <li ng-repeat="m in msg.log">
        Previous eat:
        <font color={{m.color}}>
            {{m.eat}}
        </font>
        <br>
        Feed:
        <font color="blue">
            {{m.time}}
        </font>
        <hr>
    </li>
</ul>
```

Subsequently, when user click “Manual feed”, we simply send a package to our MQTT explorer. Furthermore, when “Logout” button is clicked, we redirect to the login screen as the same way as we done on the login tab.

8.5 Scheduling tab



In this tab, I would like to split the two into two parts for easier to understand.

From “Load current setting” to “Schedule form”: we load and convert the previous setting of schedule to json object when user clicks the “Load setting” button. After that, we extract the selected schedule from our json object before putting them on the form.

```
payload = msg.payload;
schedule1 = payload.feeds[0].field1 || "";
schedule2 = payload.feeds[0].field2 || "";
schedule3 = payload.feeds[0].field3 || "";

if (schedule1!="") {
    schedule1 = new Date(parseInt(schedule1));
}

if (schedule2!="") {
    schedule2 = new Date(parseInt(schedule2));
}

if (schedule3!="") {
    schedule3 = new Date(parseInt(schedule3));
}

msg = {}
msg.payload = {
    "schedule1": schedule1,
    "schedule2": schedule2,
    "schedule3": schedule3
}

return msg;
```

From "Schdule form" to the end: When user click "Update" after they have inputted their schedule, we also extract schedule and construct the package to store in the cloud. In additional, the schedule is also sent to MQTT Explorer.

```
payload = msg.payload;

msg = {};

date1 = payload.schedule1;
date2 = payload.schedule2;
date3 = payload.schedule3;

if (date1!=null){
    date1 = new Date(date1);
    date1 = date1.getTime();
}

if (date2!=null){
    date2 = new Date(date2);
    date2 = date2.getTime();
}

if (date3!=null){
    date3 = new Date(date3);
    date3 = date3.getTime();
}

msg.payload = {
    "schedule1": date1,
    "schedule2": date2,
    "schedule3": date3
};

return msg;
```