



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 .....	1
3	Product description.....	2
3.1	Introduction.....	2
3.2	Description .....	2
4	Input and Output.....	2
4.1	Input .....	2
4.2	Output .....	2
4.3	View input.....	2
4.4	Control output .....	2
4.5	Cloud .....	2
4.6	Notification .....	2
4.7	Config wi-fi.....	3
5	Data transfer flow.....	3
6	3D design.....	4
6.1	Overall .....	4
6.2	Detail 3D sketch of the base of the feeder.....	7
7	Web design.....	8
7.1	As a guest.....	8
7.2	Login as admin.....	9
7.3	Log in as user.....	10
8	NodeRED flow .....	11

## 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

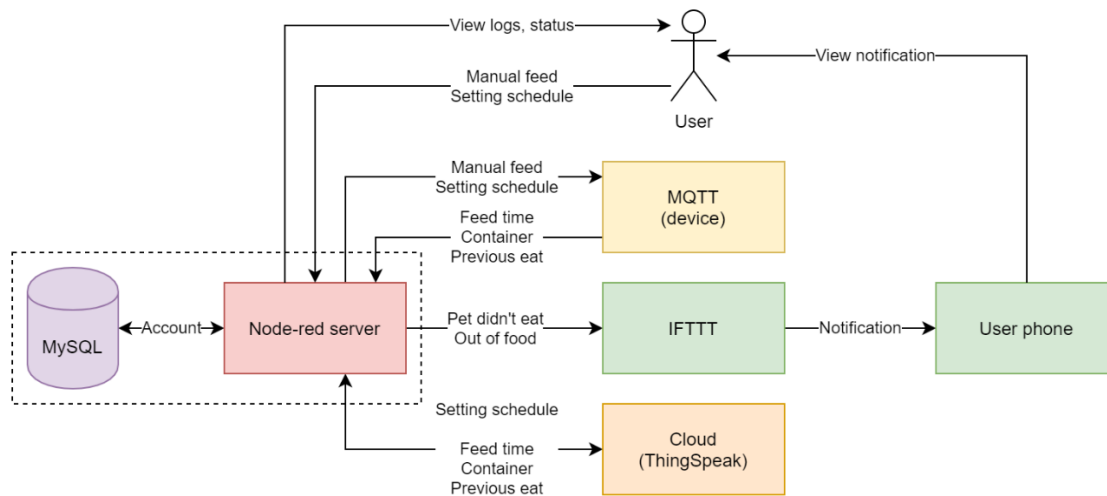
### 4.6 Notification

- 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

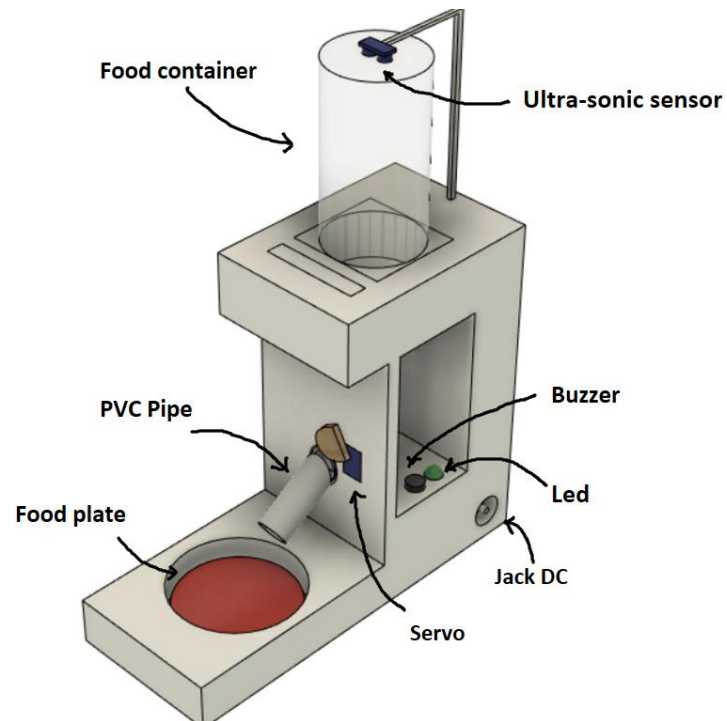


Figure 1. Full 3D sketch

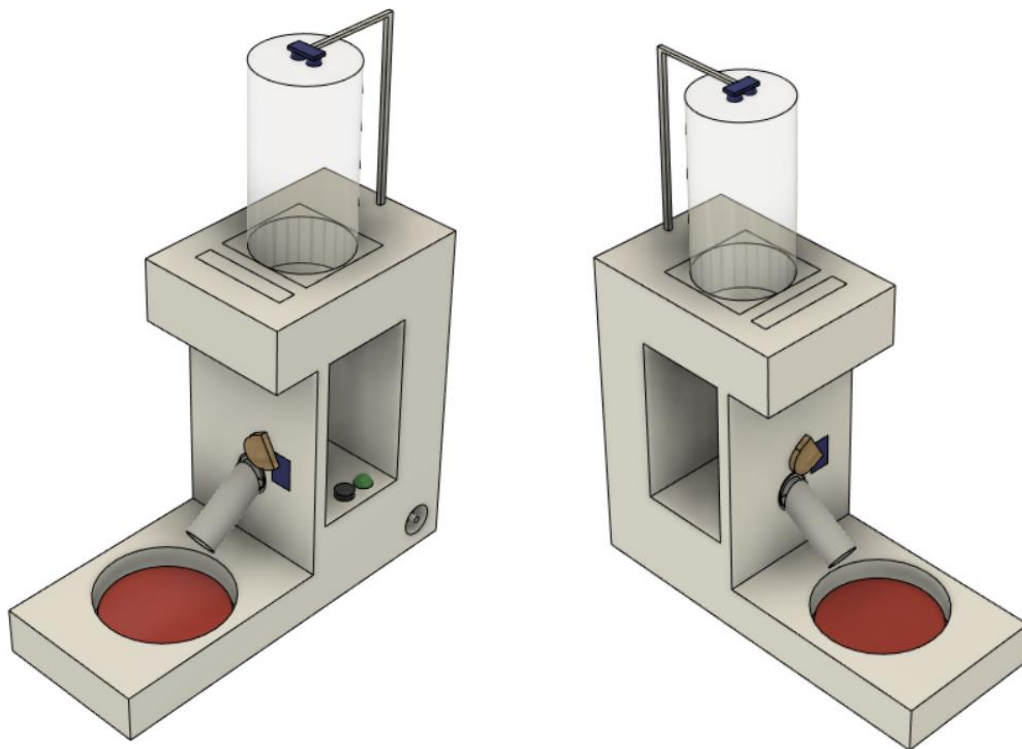
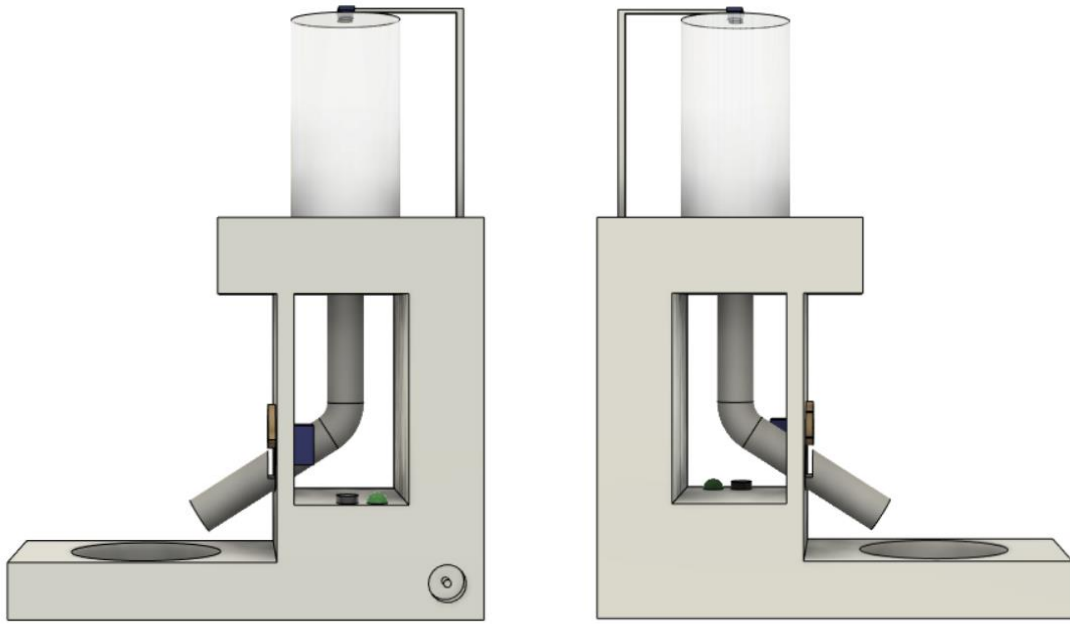
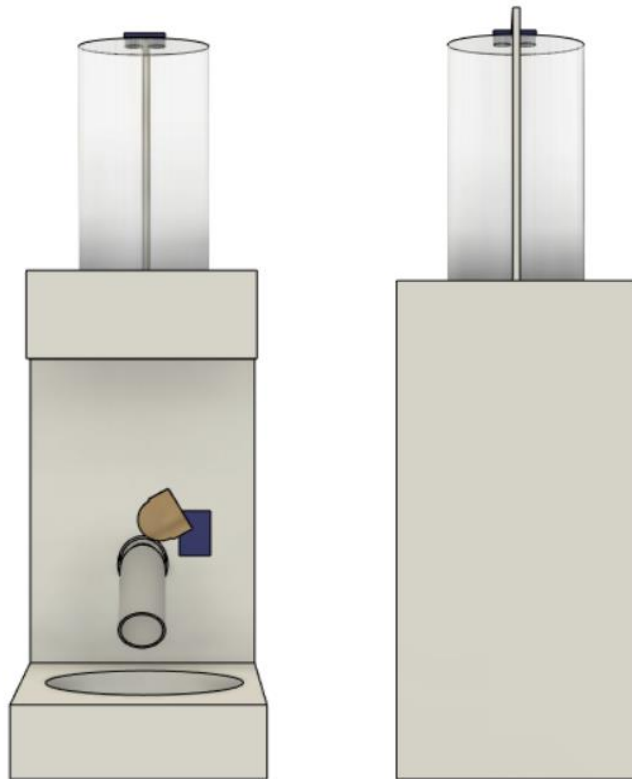


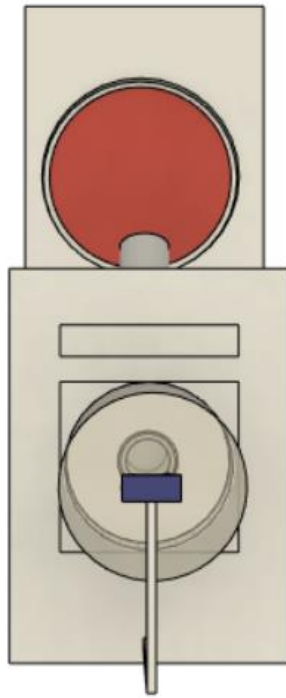
Figure 2. Top-left and top-right view respectively form left to right



*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

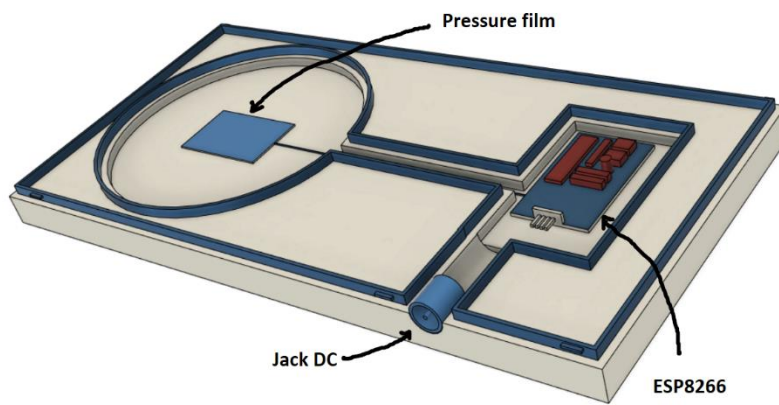


Figure 6. Top-left view

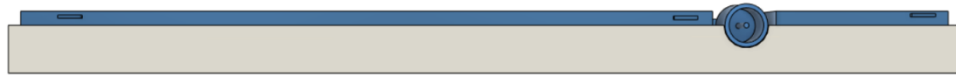


Figure 7. Left view

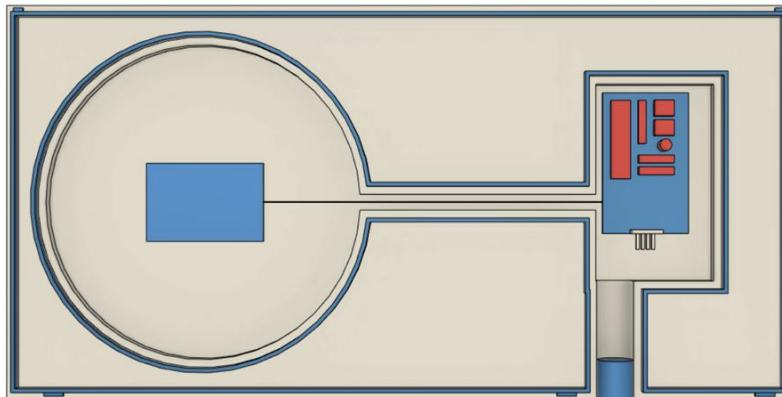
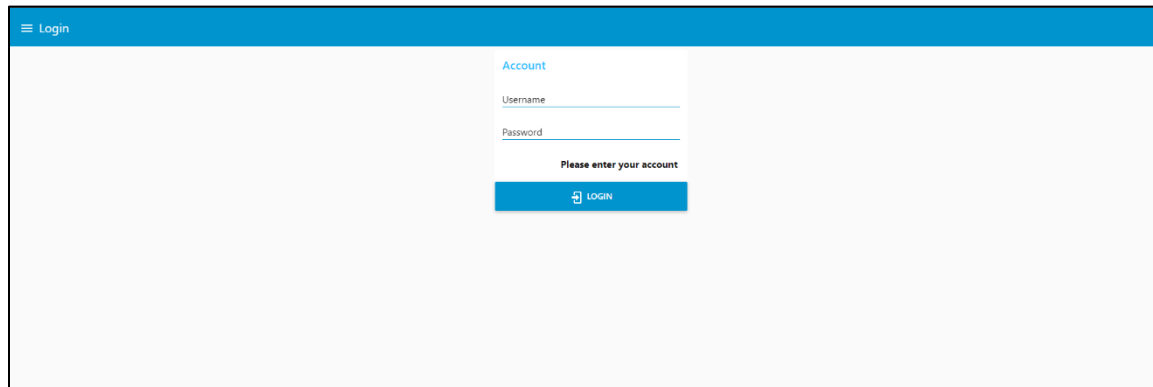


Figure 8. Top view



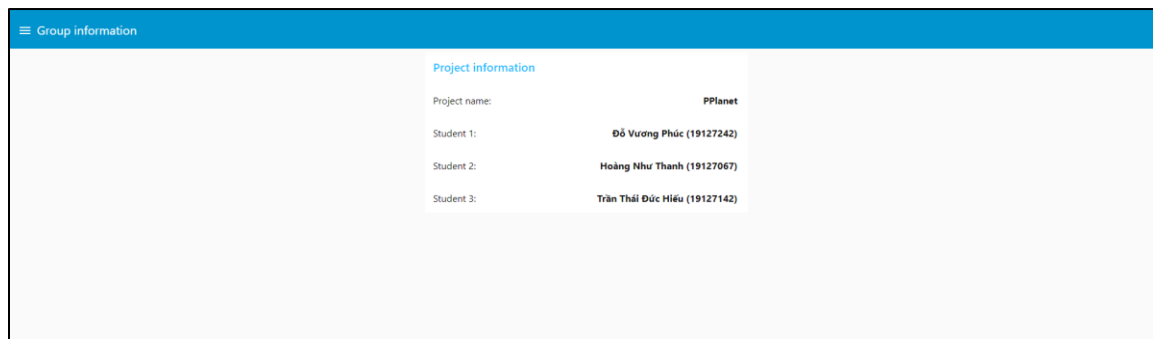
## 7 Web design

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



The login screen features a blue header bar with a hamburger menu icon and the text "Login". The main content area is light gray. In the center, there is a white box titled "Account" containing two input fields for "Username" and "Password". Below these fields is the text "Please enter your account" and a blue button with a white login icon and the text "LOGIN".

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



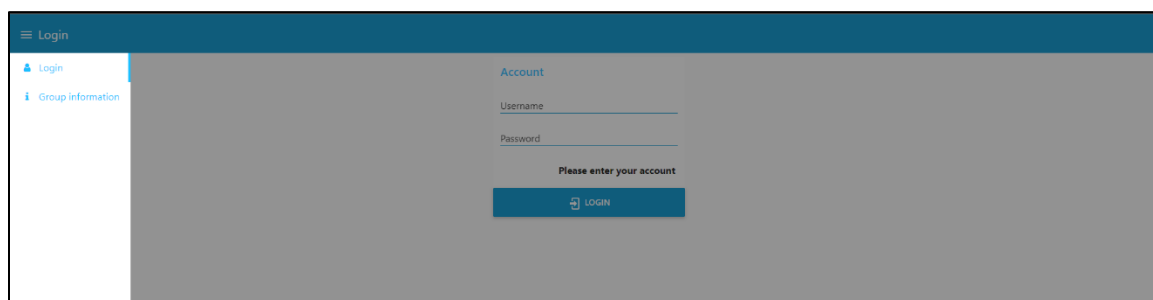
The group information screen has a blue header bar with a hamburger menu icon and the text "Group information". The main content area is light gray. In the center, there is a white box titled "Project information" containing a table with project details.

Project name:	PPlanet
Student 1:	Đỗ Vương Phúc (19127242)
Student 2:	Hoàng Như Thanh (19127067)
Student 3:	Trần Thái Đức Hiếu (19127142)

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

### 7.1 As a guest

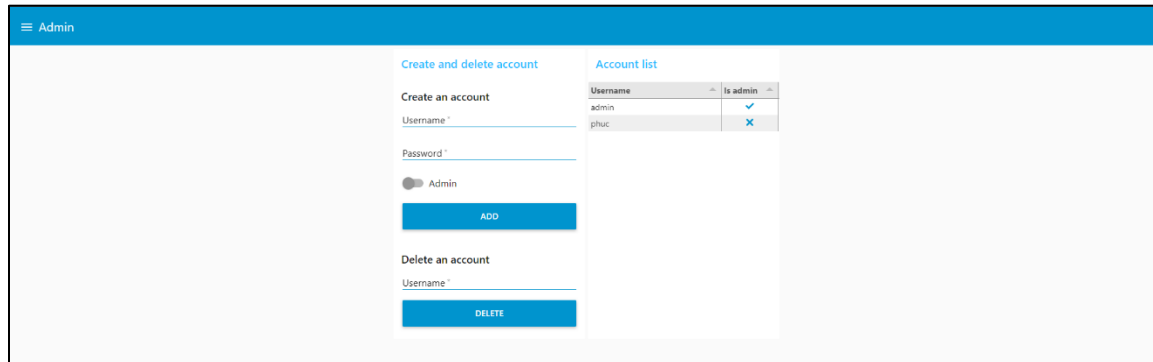
A guest can only view login tab and group information tab.



The guest view shows a dark blue header bar with a hamburger menu icon and the text "Login". On the left, there is a white sidebar with two tabs: "Login" (active, with a blue icon) and "Group information" (with an information icon). The main content area is dark gray. In the center, there is a white box titled "Account" containing two input fields for "Username" and "Password". Below these fields is the text "Please enter your account" and a dark blue button with a white login icon and the text "LOGIN".

## 7.2 Login as admin

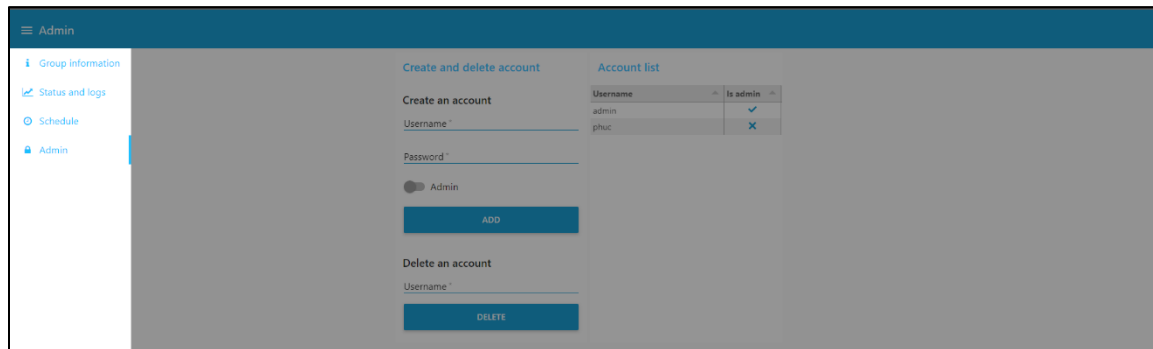
As an admin, we can add or delete any user. We can also view the account list.



The screenshot shows the Admin interface with a blue header bar containing a hamburger menu icon and the text "Admin". The main content area is divided into two columns. The left column, titled "Create and delete account", contains two sections: "Create an account" and "Delete an account". The "Create an account" section has a "Username" input field, a "Password" input field, a toggle switch labeled "Admin" (which is currently turned off), and a blue "ADD" button. The "Delete an account" section has a "Username" input field and a blue "DELETE" button. The right column, titled "Account list", contains a table with two columns: "Username" and "Is admin". The table has two rows: one with "admin" and a blue checkmark, and another with "phuc" and a blue X.

Username	Is admin
admin	✓
phuc	✗

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.

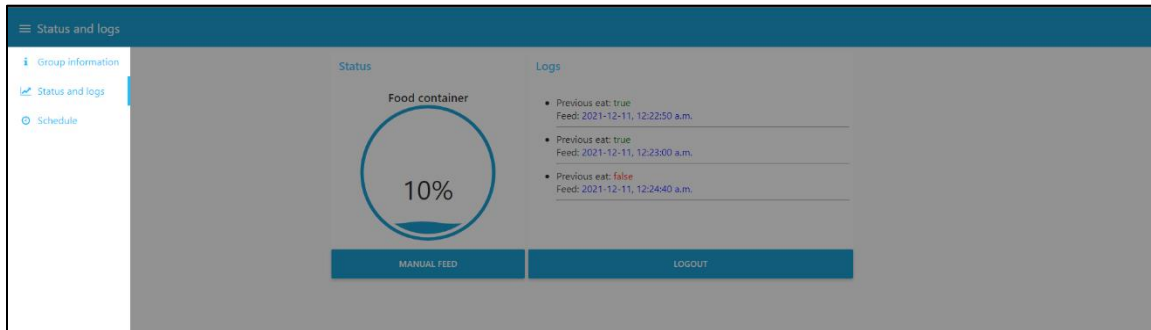


The screenshot shows the Admin interface with a dark blue header bar containing a hamburger menu icon and the text "Admin". A sidebar menu is visible on the left side, listing four items: "Group information", "Status and logs", "Schedule", and "Admin". The "Admin" item is highlighted with a blue bar. The main content area is divided into two columns, identical to the previous screenshot, showing the "Create and delete account" and "Account list" sections.

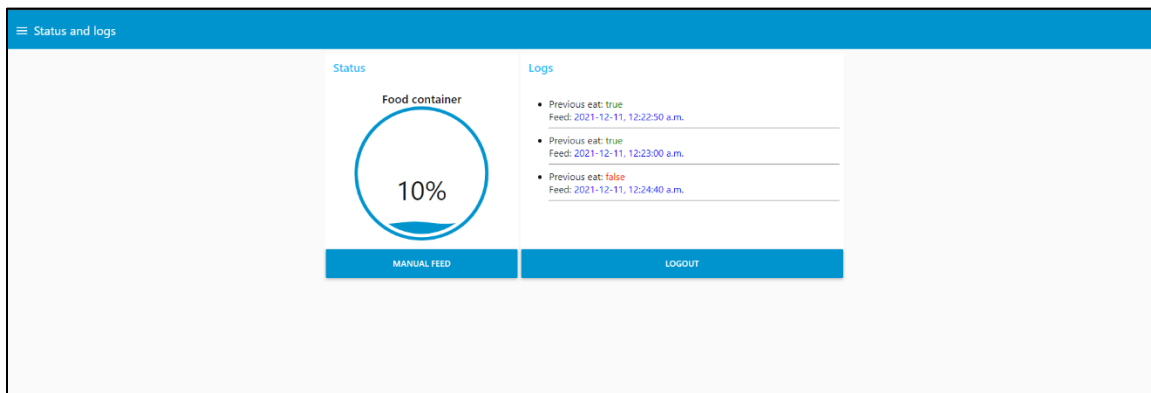
Username	Is admin
admin	✓
phuc	✗

## 7.3 Log in as user

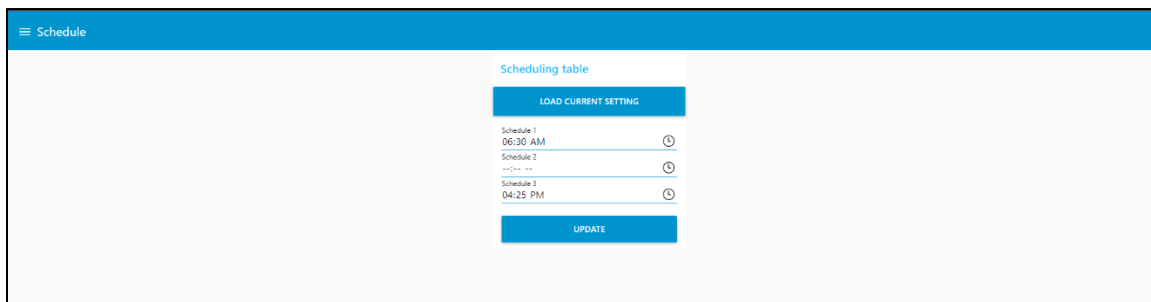
A user can feed manually, view status and logs and schedule feeding.



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

Project name:

Student 1:

Student 2:

Student 3:

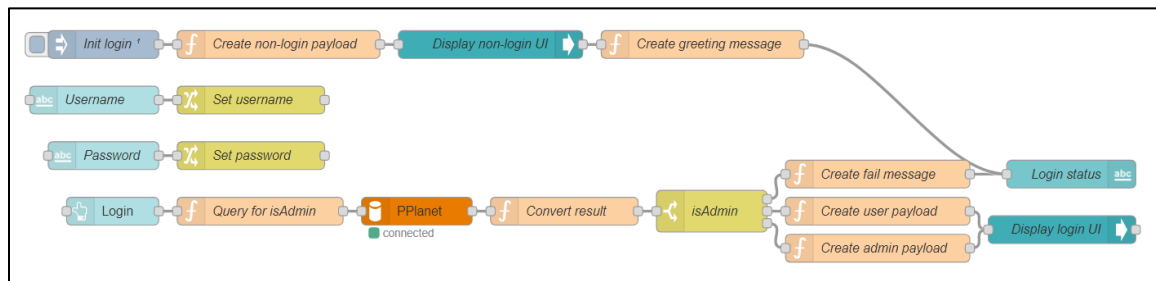
The flow of group information is quite simple, it only contains three text displays in order to show information of group.

### 8.2 Login tab

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	username	varchar(20)	utf8mb4_general_ci	No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/>	2	password	varchar(20)	utf8mb4_general_ci	No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/>	3	isAdmin	tinyint(1)		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

☐ Check all With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#)

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:

```
1 msg = {};
2 msg.payload = {
3   "tab": "Login",
4   "tabs": {
5     "hide": ["Status and logs", "Schedule", "Event Logs", "Admin"],
6     "show": ["Group information", "Login"],
7   },
8 };
9 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.

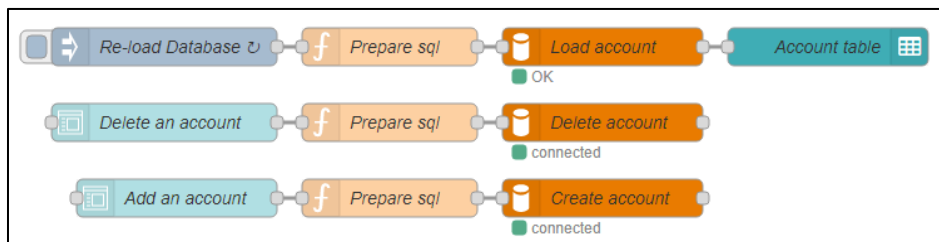
```
1 msg = {};
2
3 username = flow.get('username') || "";
4 password = flow.get('password') || "";
5
6 msg.payload = [username,password];
7 msg.topic = "SELECT isAdmin FROM ACCOUNT WHERE username=? AND password=?";
8
9 return msg;
```

After received the result, we first convert it into  $\begin{cases} -1 & (\text{fail}) \\ 0 & (\text{user}) \\ 1 & (\text{admin}) \end{cases}$ . Since then, we can use

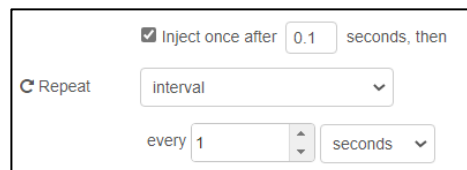
switch case to change message on the screen, or redirect them to a new screen.

```
1 if (msg.payload[0] !== undefined) {
2   msg.payload = msg.payload[0].isAdmin;
3 } else {
4   msg.payload = -1;
5 }
6
7 return msg;
```

### 8.3 Admin tab



If the logged 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.

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

Then display all the receive records within two columns:

Property	username		
Title	Username		
Align	left	Width	70%
Format	Plain text		

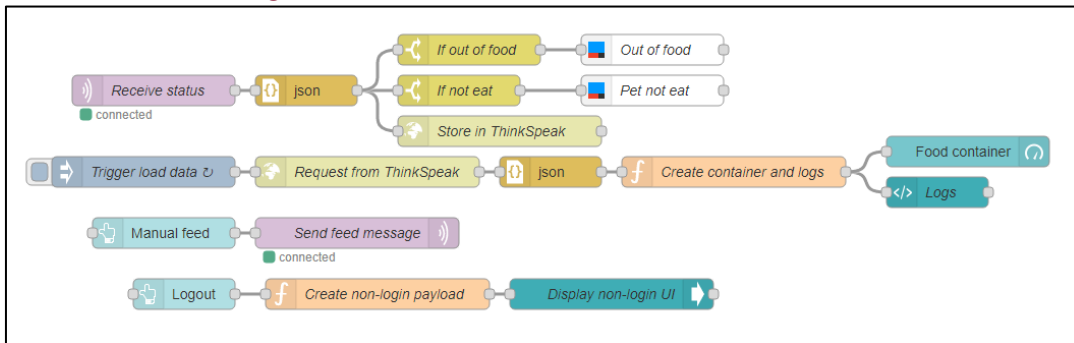
---

Property	isAdmin		
Title	Is admin		
Align	center	Width	30%
Format	Tick / Cross (boolean, 1/0)		

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.

```
1 payload = msg.payload;
2 username = payload.username;
3
4 msg.payload = [username];
5 msg.topic = "DELETE FROM ACCOUNT WHERE username=?"
6
7 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.

```

1 feeds = msg.payload.feeds;
2 nFeed = feeds.length;
3 msg = {}
4
5 if (nFeed>0) {
6     msg.container = feeds[nFeed-1].field2;
7 } else {
8     msg.container = 0;
9 }
10
11 log = [];
12 for (let i in feeds){
13     date = new Date(parseInt(feeds[i].field1));
14     eatValue = feeds[i].field3;
15     if (eatValue=="true") {
16         color = "green";
17     } else {
18         color = "red";
19     }
20     log.push({
21         "time": date.toLocaleString(),
22         "eat": eatValue,
23         "color": color
24     });
25 }
26
27 msg.log = log;
28 return msg;

```

After converted, we display logs by using template node:

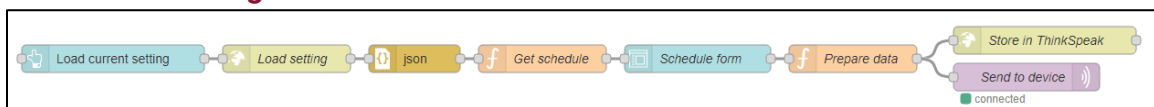
```

1 <ul>
2     <li ng-repeat="m in msg.log">
3         Previous eat:
4         <font color={{m.color}}>
5             {{m.eat}}
6         </font>
7         <br>
8         Feed:
9         <font color="blue">
10             {{m.time}}
11         </font>
12         <hr>
13     </li>
14 </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.

```

1 payload = msg.payload;
2 schedule1 = payload.feeds[0].field1 || "";
3 schedule2 = payload.feeds[0].field2 || "";
4 schedule3 = payload.feeds[0].field3 || "";
5
6 if (schedule1!="") {
7     schedule1 = new Date(parseInt(schedule1));
8 }
9
10 if (schedule2!="") {
11     schedule2 = new Date(parseInt(schedule2));
12 }
13
14 if (schedule3!="") {
15     schedule3 = new Date(parseInt(schedule3));
16 }
17
18 msg = {};
19 msg.payload = {
20     "schedule1": schedule1,
21     "schedule2": schedule2,
22     "schedule3": schedule3
23 };
24
25 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.

```

1 payload = msg.payload;
2
3 msg = {};
4
5 date1 = payload.schedule1;
6 date2 = payload.schedule2;
7 date3 = payload.schedule3;
8
9 if (date1!=null){
10     date1 = new Date(date1);
11     date1 = date1.getTime();
12 }
13
14 if (date2!=null){
15     date2 = new Date(date2);
16     date2 = date2.getTime();
17 }
18
19 if (date3!=null){
20     date3 = new Date(date3);
21     date3 = date3.getTime();
22 }
23
24 msg.payload = {
25     "schedule1": date1,
26     "schedule2": date2,
27     "schedule3": date3
28 };
29
30 return msg;

```