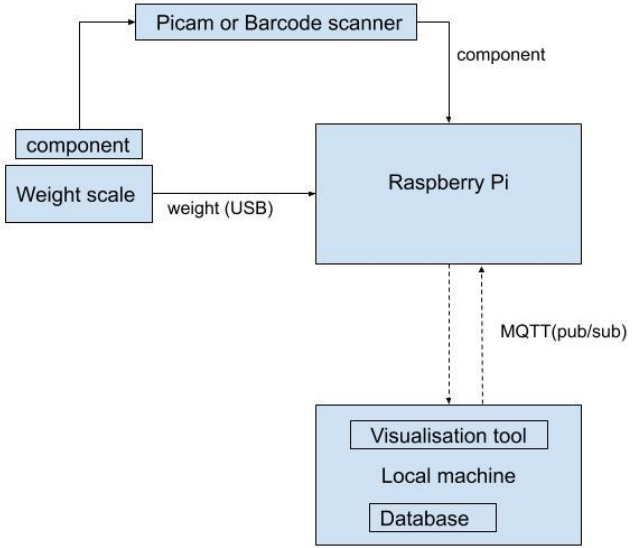
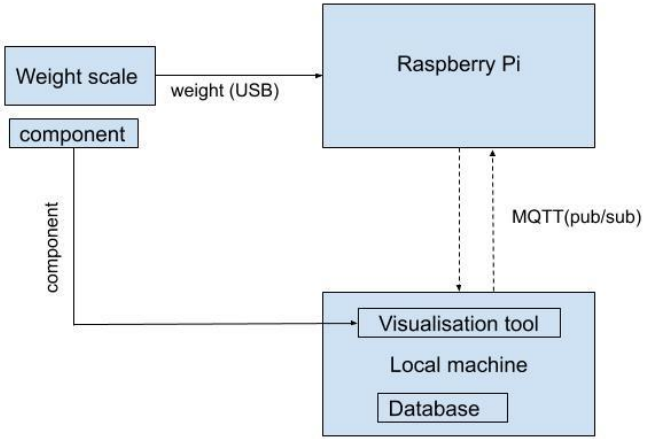
DISTRIBUTED STORAGE SYSTEM

Guide :
Prof. Laura Thiele
Prof. Sascha Bosse

Abhijith Remesh - 221424
Baizil Mulakkampilly - 221544

Block diagram





Hardware and Software used

Node-RED

MYSQL

Tasks

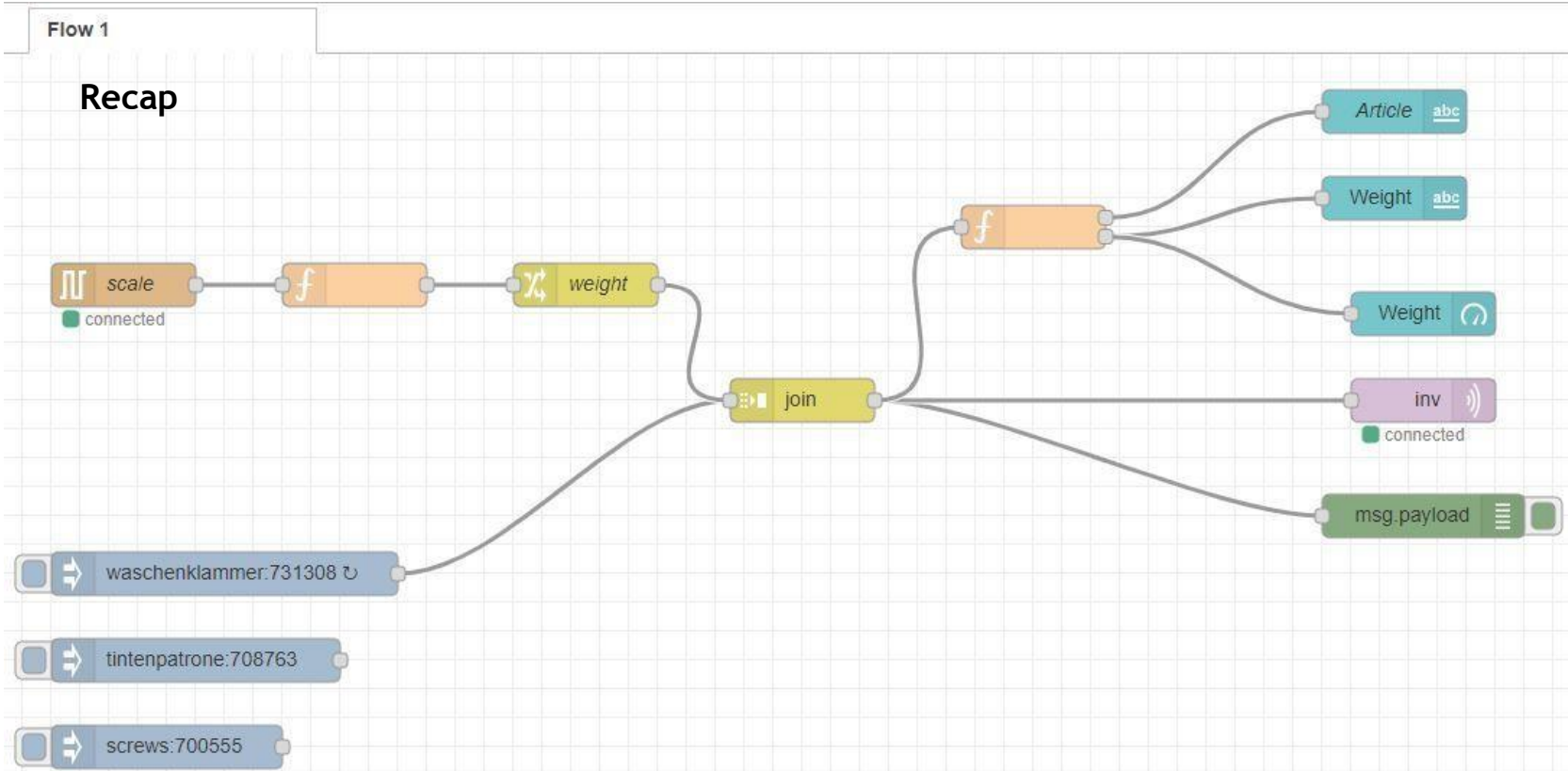
- Interfacing the scaling device with raspberry pi and parsing the data in specific data formats.
- Development of database containing the feature information of warehouse components (name, type, unit weight)
- Developing the logic to identify the counts of warehouse components accurately based on weight comparison.
- Setting up MQTT client on local machine and MQTT broker on pi.
- Implementing the dashboard to view the inventory data levels of each warehouse component.
- Developing the user interface facility to add new warehouse component, select a component and delete an existing component.
- Development of a database for storing the inventory log
- Storing the inventory log as per user input.
- Utility to delete and download the inventory log locally via user input
- Approaches to identify article weight mismatch.

Recap


```python
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate=9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1)

while 1:
    x=ser.readline()
    print (x)
```

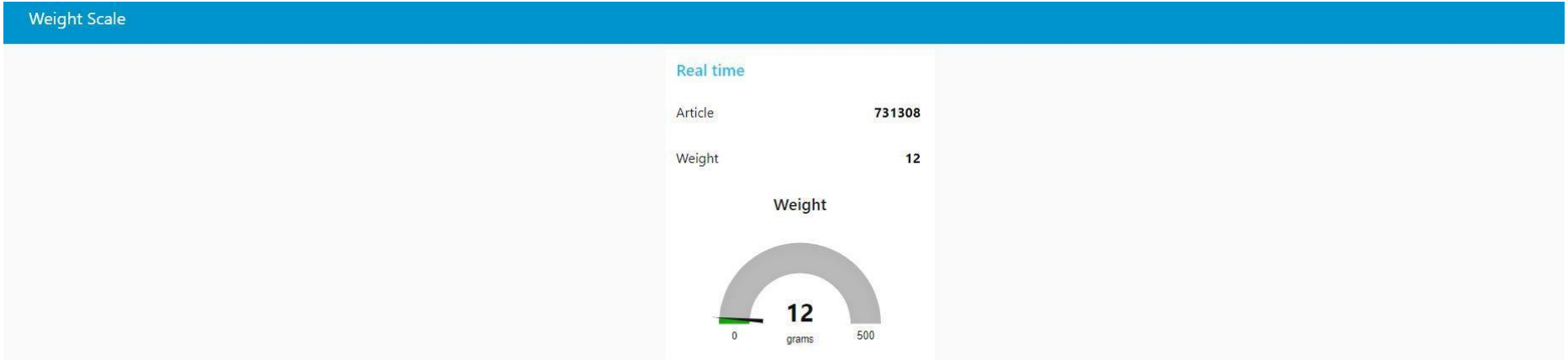
```
Coming from First device
W:+     188.8g

Coming from First device
W:+     188.8g

Coming from First device
W:+     188.8g
```

Interfacing weight scale with pi



Node-red flow at pi end

Recap



Node-red UI at pi end

Articles we bought...

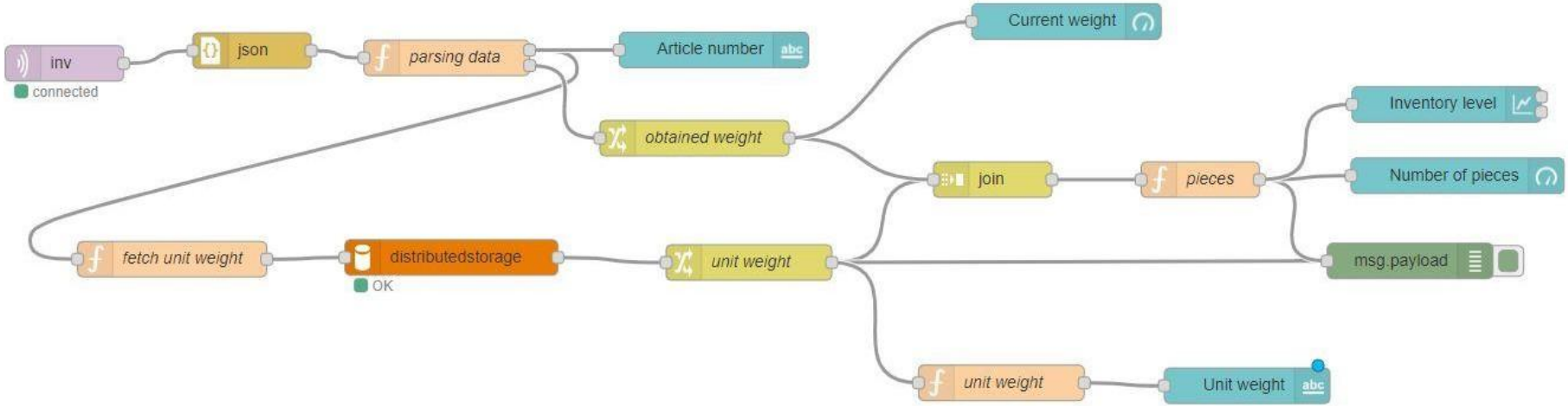
id	Article number	Article name	Unit Weight
1	731308	Waschenklammer	4
2	708763	Tintenpatrone	1.4
3	700555	Screws	3

Distributed Storage system database
Table - inventorydb

```
CREATE DATABASE `distributedstorage`;

CREATE TABLE `distributedstorage`.`inventorydb` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `article_number` INT NOT NULL,
  `article_name` VARCHAR(45) NOT NULL,
  `unit_weight` FLOAT NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  UNIQUE INDEX `article_number_UNIQUE` (`article_number` ASC) VISIBLE,
  UNIQUE INDEX `article_name_UNIQUE` (`article_name` ASC) VISIBLE);

INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('1','731308','waschеklammer', 4);
INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('2','708763','tintenpatrone', 1.4);
INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('3','700555','screw', 3);
```

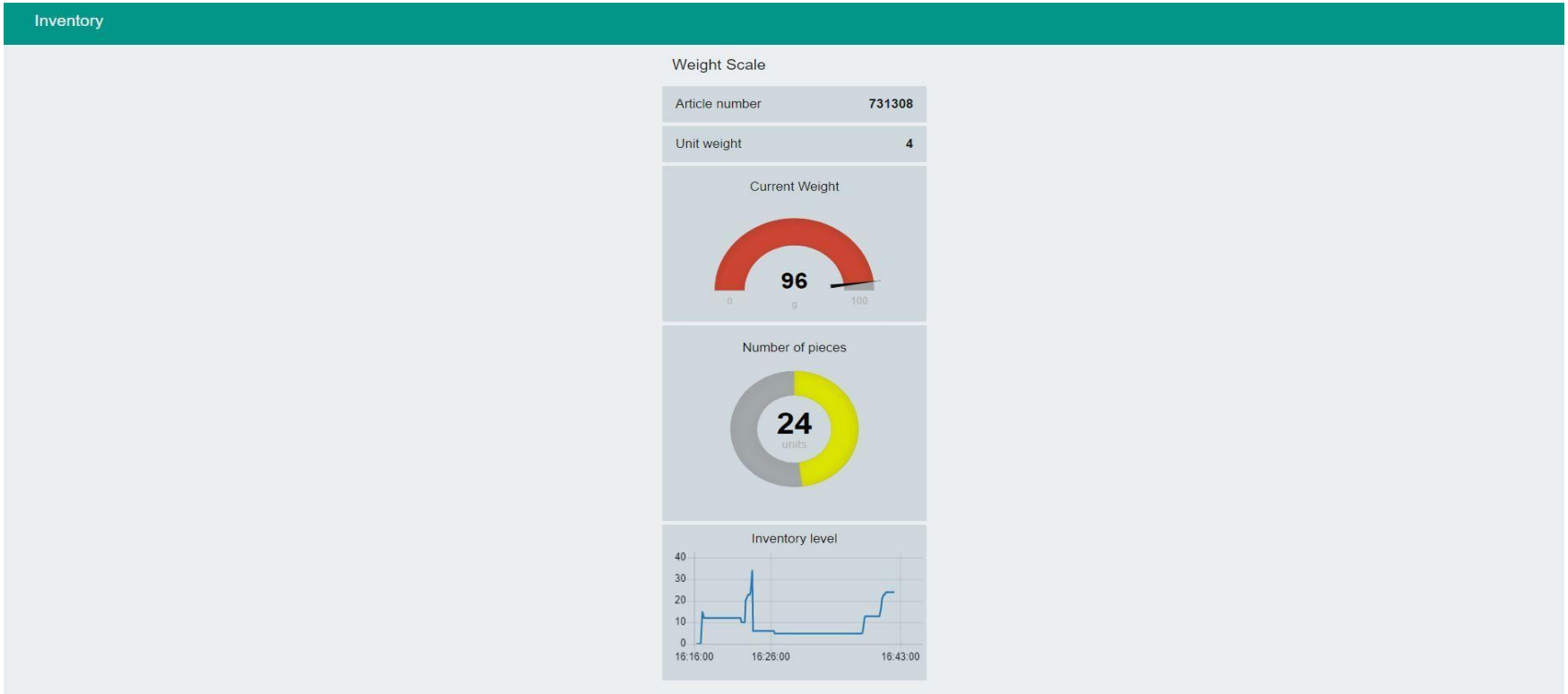
id	article_number	article_name	unit_weight
1	731308	waschеklammer	4
2	708763	tintenpatrone	1.4
3	700555	screw	3

Recap



Node-red flow at client end

Recap



Node-red UI at client end

‘distributedstoragesystem’ database table ‘inventorydb’

```
CREATE DATABASE `distributedstorage`;

CREATE TABLE `distributedstorage`.`inventorydb` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `article_number` INT NOT NULL,
  `article_name` VARCHAR(45) NOT NULL,
  `unit_weight` FLOAT NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  UNIQUE INDEX `article_number_UNIQUE` (`article_number` ASC) VISIBLE,
  UNIQUE INDEX `article_name_UNIQUE` (`article_name` ASC) VISIBLE);

INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('1','731308','wascheklammer', 4);
INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('2','708763','tintenpatrone', 1.4);
INSERT INTO `distributedstorage`.`inventorydb` (`id`,`article_number`,`article_name`,`unit_weight`)
VALUES ('3','700555','screw', 3);
```

id	article_number	article_name	unit_weight
1	731308	wascheklammer	4
2	708763	tintenpatrone	1.4
3	700555	screw	3

master data table to store the feature info of inventory warehouse components

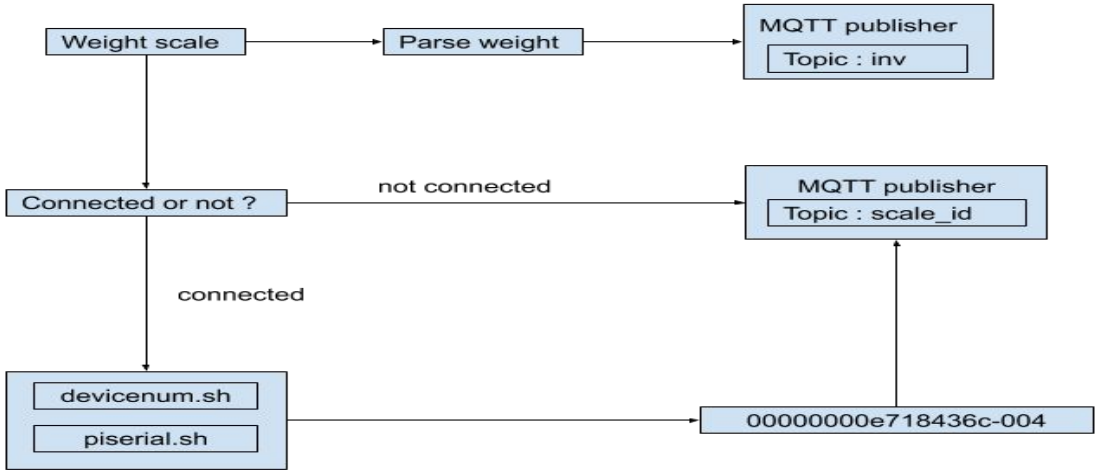
‘inventorylog’ database
table ‘datalog’

```
CREATE DATABASE `inventorylog`;

CREATE TABLE `inventorylog`. `datalog` (
   `time` varchar(45) NOT NULL,
   `scale_id` varchar(45) NOT NULL,
   `article_number` int(11) NOT NULL,
   `unit_weight` float NOT NULL,
   `obtained_weight` float DEFAULT NULL,
   `count` int(11) DEFAULT NULL
) ;
```

datalog table to store the inventory log over time

time	scale_id	article_number	unit_weight	obtained_weight	count
30-11-2019 23:27:58	00000000e718436c-007	731308	4	0	0
30-11-2019 23:28:03	00000000e718436c-007	731308	4	4	1
30-11-2019 23:28:07	00000000e718436c-007	731308	4	8	2
30-11-2019 23:28:11	00000000e718436c-007	731308	4	12	3
30-11-2019 23:28:15	00000000e718436c-007	731308	4	16.2	4
30-11-2019 23:28:26	00000000e718436c-007	731308	4	20.2	5
30-11-2019 23:28:30	00000000e718436c-007	731308	4	24.2	6
30-11-2019 23:28:34	00000000e718436c-007	731308	4	28.2	7
30-11-2019 23:28:39	00000000e718436c-007	731308	4	32.2	8
30-11-2019 23:28:50	00000000e718436c-007	731308	4	36.2	9
30-11-2019 23:28:55	00000000e718436c-007	731308	4	40.2	10

Flow logic at the pi end



devicenum.sh gets the device number allocated by the pi to the connected USB device.
piserial.sh gets the serial number of the raspberry pi.
Combining both gives the scale id.

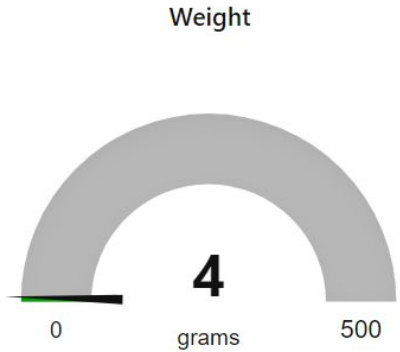
node red flow at the pi end

Dashboard at the pi end

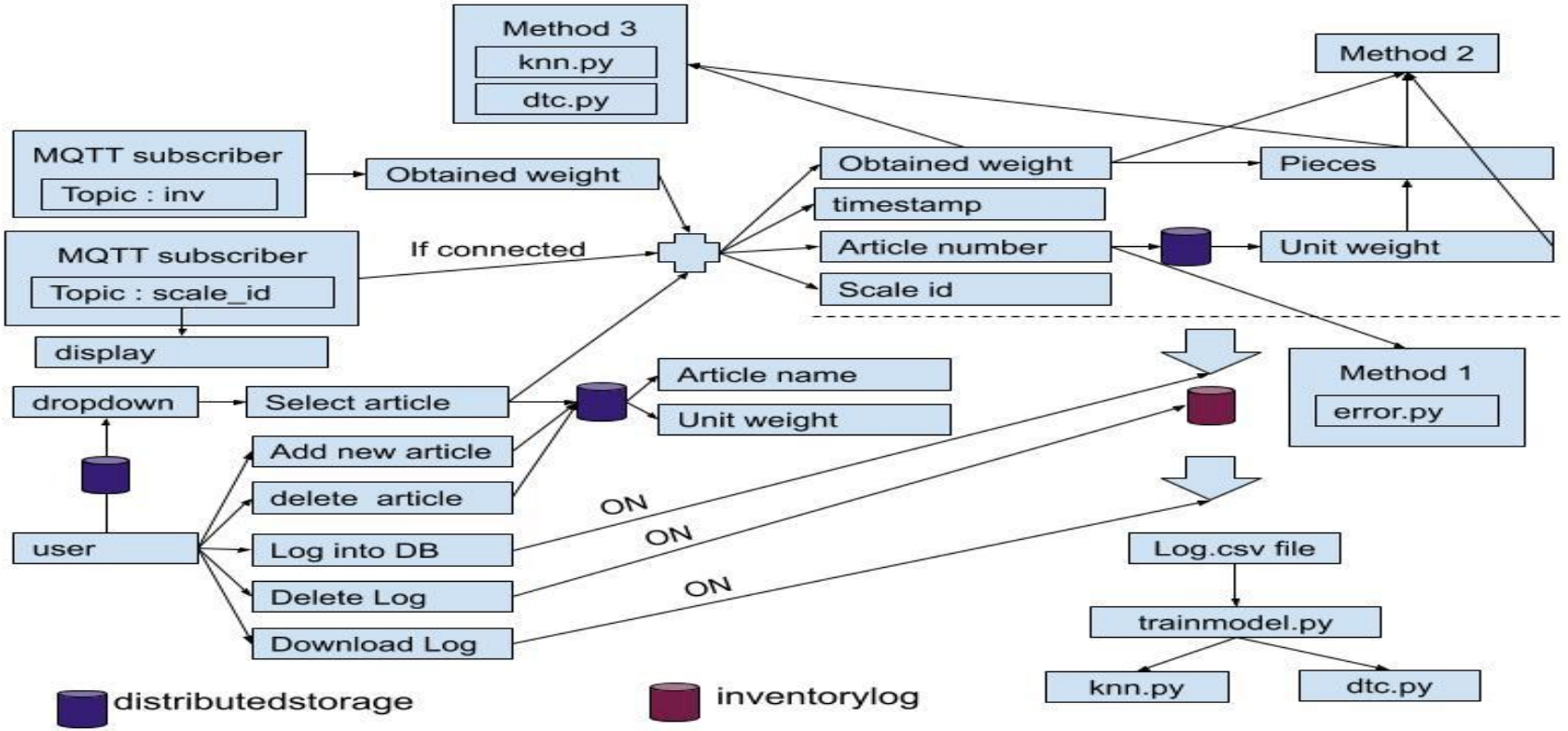
Weight Scale

Real time

Weight 4

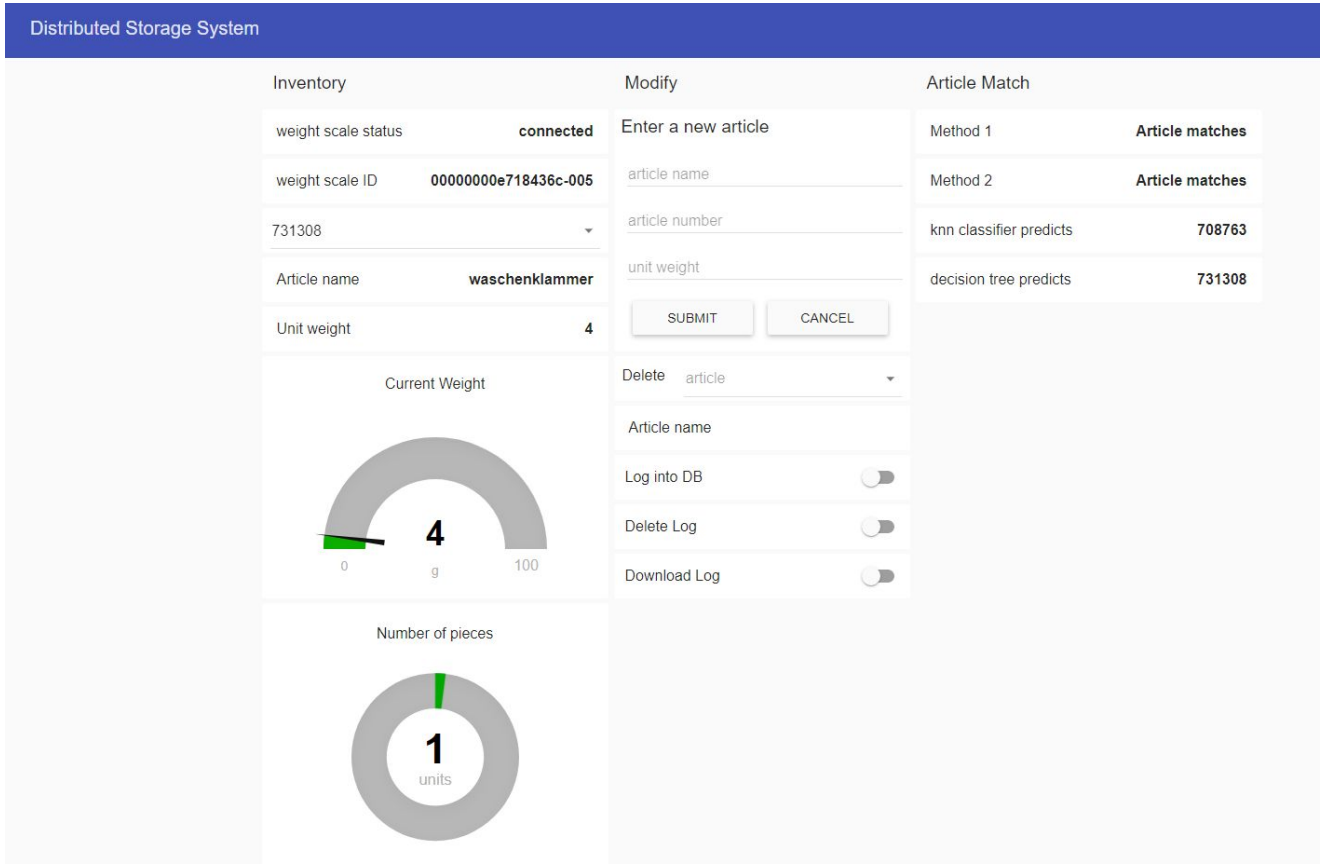


Flow logic at the client end



node red flow at the client end

Dashboard at client side



Scalability approaches

1 weight scale - 1 raspberry Pi

- weight scale + raspberry pi can be identified using scale id.

- for each new pi, a new duplicate flow must be deployed at both pi and client end.
  - for each new flow, the MQTT node must be specified with it’s respective ip address.
  - Each flow has it’s own dashboard.

- n weight scales > n raspberry pi > n flows at pi end > n flows at client end> n client dashboards > 1 distributed storage system

4 weight scale - 1 raspberry Pi

- For each weight scales connected to pi, pi randomly allocates device numbers to it like **e718436c-005, e718436c-006, e718436c-00X.**

- Can identify from which pi, the readings are being received but cannot identify from which connected weight scales, it is being sent.

- Since a pi can have four weight scales connected to it, a single pi will have four node red flows for each of the weight scale
    - for each flow, the “serial in” node must be specified with corresponding serial port.
    - for all the four flows, the “MQTT” node should be having the same ip address.
    - A single flow at the client end for one pi with four weight scales sending weight readings.
    - One single dashboard at the client end.

- For every new pi, the third step is to be repeated.
- 4n weight scales > n raspberry pi > 4 flows at each (1..n) pi > n flows at client end > n dashboards > 1 distributed storage system.

Note - Since there is only one dashboard at the client end for one raspberry pi which is connected to four weight scales, it may lead to synchronisation issues.