# **COEN-396 WEB INFRASTRUCTURE AND RELATED TECHNOLOGY**

# **WINTER 2016**

**IOT HYDROPONICS**

**BALAKRISHNAN RAMDOSS (W1178330)**

**PARTH KALAVADIA (W1168539)**

**PRATEEK KHATRI (W1189416)**

**SAI SRINIVAS REDDY ERLA (W1182111)**

**SHRUTHI MURALI (W1186354)**

**VARUN RAPARLA (W1186352)**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **CHAPTER** | **PAGE NO** |
|  | **List of Figures**  **Abstract** |  |
| **1.** | **INTRODUCTION** |  |
| **2.** | **BACKGROUND**  **2.1 Theoretical Basis**  **2.2 Architecture** |  |
| **3.** | **MODULES**  **3.1 Embedded Control System**  **3.2 Fog computing**  **3.3 Web API’s**  **3.4 Web Application** |  |
| **4.**  **5.**  **6.** | **RESULTS**  **CONCLUSIONS**  **FUTURE WORK** |  |
|  | **REFERENCES**  **APPENDIX** |  |

**List of Figures**

|  |  |
| --- | --- |
| **Fig 3.3.0** | **WEB API Architecture** |
| **Fig 3.3.1** | **Login UI for Web Application with redirectUri** |
| **Fig 3.4.1** | **User dashboard** |
| **Fig 3.4.2** | **Plant details** |
| **Fig 3.4.3** | **Custom Plant** |
| **Fig 3.4.4** | **New System** |
| **Fig 3.4.5** | **Statistics** |

***ABSTRACT***

*We are civilized, modernized, globalized. But our biggest failure in civilization is “Soil Erosion” where we are making soil unable to grow crops effectively as it used to be. But we are modernized, we discovered various soil- less cultivation methods which we are sure that they are the future. One of such kind is “HYDROPONICS”- the method of growing crops without soil, within a mineral treated water solution in a controlled environment. Since they are grown in a controlled environment system, they regularly need environmental support for sustainability, sufficient nutrient proportion in water, also roots not only absorb fluids but they return a portion of their peculiar secretion back into it there by changing Ph. value of water since all of them excrete either acidic or basic solution. So every time there should be someone to look over these conditions. Why waste human power if we have sensors to look over it. Where everything can be controlled by data received from sensors and let actuators work accordingly which together will be “Internet of Things”. When all this can be controlled from a mobile location through a centralized and a well-structured system where a user can login into his controlled environment and check the condition of the crops with just a single click, anything can be grown without climatic and locational limitations since everything here is controlled accordingly.*

**1. INTRODUCTION**

Hydroponics is a unique way of crop cultivation where a substitute of soil is used for harvesting and all the nutritions, required by plants is provided through water in an controlled environment. The key aspect of hydroponics is the yield produced by the plants, which is much more than traditional technique and time required for overall growth of plants is less. Secondly, it saves upto 95% of water required than in traditional technique. Some of the benefits of hydroponics are as followed:

* Pest free cultivation of crops.
* Less surface area required for cultivation.
* Can be grown at any place irrespective of the soil quality.
* Less labour required.
* Can grow in any climate.
* Nutrition level in crops is much higher than crops grown in traditional technique.

**2. BACKGROUND**

**2.1 Theoretical Basis**

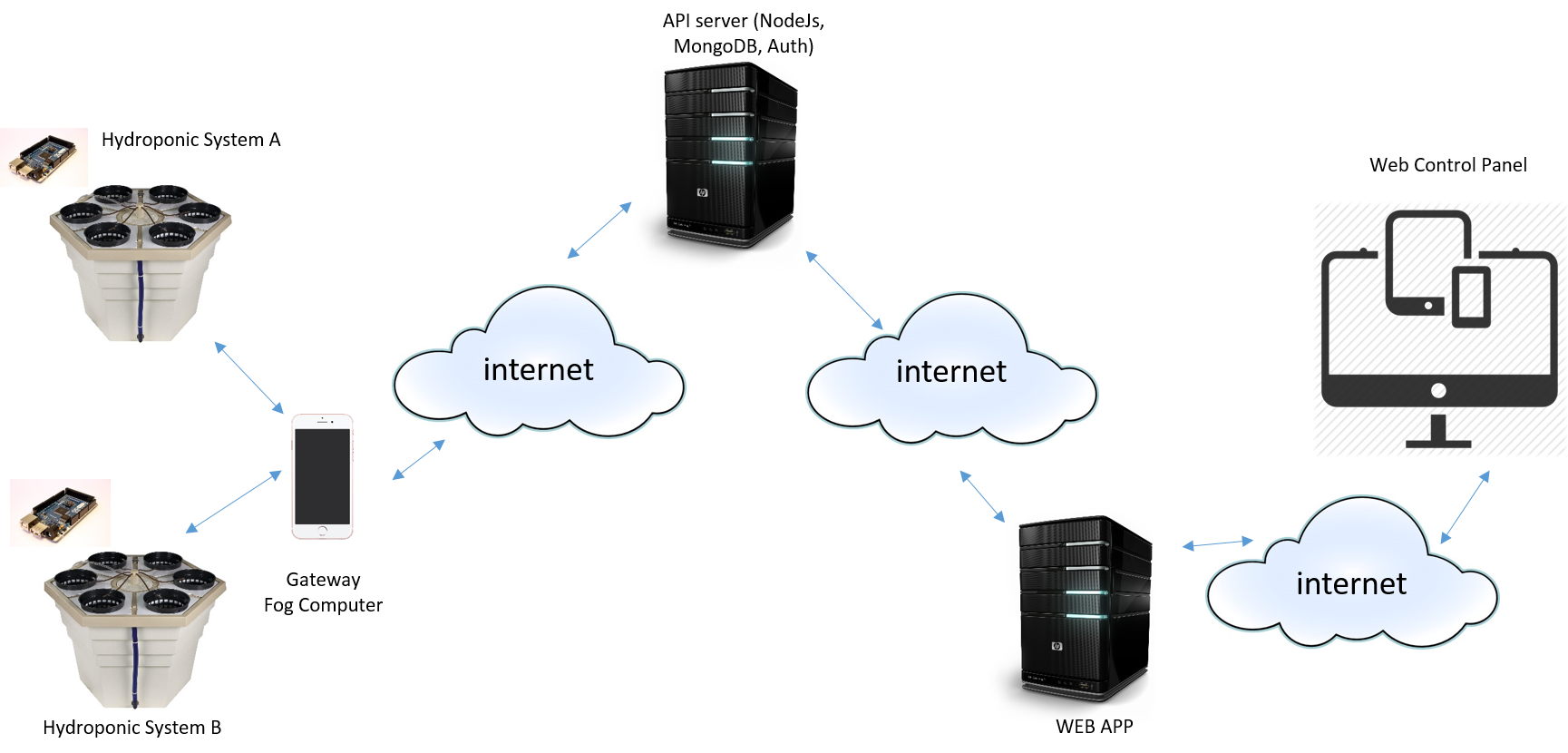
Hydroponics is a soilless technique of agriculture where multiple environmental parameters required by plants are controlled artificially. The control parameter basically includes light, moisture, water pH and nutrition level. If parameters are regularly monitored and controlled by control system, an efficient way to implement hydroponics can be implemented. To do so, we studied the way in which each of these parameters can be controlled individually and then put together in one system.

* Light control system: We have photon light sensor that helps to find the current light intensity. If the intensity level falls below a certain threshold, the external light should be turned on.
* Moisture control system: A minimum moisture level is required to maintain a constant moisture level to the roots. To accomplish this, we have moisture sensor which senses the moisture level in substrate and switches on the water pumps.
* pH control system: pH level needs to be maintained between certain range, so ph level is measured through pH sensor and acid or base is pumped in water according to pH value.
* TDS control system: Total dissolve solids is used to measure the nutrition level in water. This is measured through EC meter(EC sensor) and nutrition or water is pumped in reservoir to maintain nutrition level.

We did the study of which sensors and actuators required for each control system.

We even did study as to what are the parameters each individual plant require and accordingly, the values were set on server. Later we decided to keep this dynamic as it can be useful for scientists to experiment on the hydroponic system.

**2.2 Architecture**



**3. MODULES**

**3.1 Embedded Control System**

The main control system of the project consists of various controllers and actuators which form the Hydroponic System. The main objective of this control system is to provide sustenance to the plants and keep them healthy. However, this can only be accomplished by some initial information provided by the gardner via the server to the device.

**3.1.1 The flow of Information**

The ingredients which the control system needs to provide the plants are as follows:

* Acid and Base (for pH Balancing)
* Nutrients (for growth)
* Water
* Light

To decide which ingredient is provided to the plant at what time some initial information is required from the server for the control system to instantiate. The communication happens between the system and a mobile device using WiFi. The system sends HTTP get request with the values of the sensors or actuators depending on the type of request and always receives the latest thresholds in response. A sample of the get request sent by the system is as follows:

*http.begin("172.20.10.1", 8080,"?&fn=getSensorDetails&sensorId=esp001&ph=7.04&tds=50.23&light=91&moisture=64&boot=true");*

The service “getSensorDetails” is used to send latest sensor values to the mobile device which are then also forwarded to the cloud server. In response the system receives threshold values of all the actuators which would be activated in case the sensor readings go out of the threshold range. A sample of the response to the HTTP request received by the system is as follows:

*{*

*“Valid”: 1,*

*“phUpperThreshold”: 8,*

*“phLowerThreshold”: 6,*

*“tdsLowerThreshold”: 40,*

*“tdsUpperThreshold”: 70,*

*“soilMoistureMin”: 60,*

*“lightIntensityMin”: 80*

*}*

This JSON object received by the embedded control system is enough to activate it and make it running. The first field of “valid” states that the thresholds in the JSON object are validated by the Mobile server and are ready for use by the system. On boot the system waits for the valid thresholds to be received and only after that the system is activated. Also, the system processes pH and TDS levels as float values and Soil Moisture Level and Light Intensity on percentage scales.

**3.1.2 Multi-Master vs. Master-Slave Configuration**

On the initial design phase the embedded system was designed to be a multi-master collaborative system between the ESP8266 (NodeMCU) and an Arduino UNO controller. After implementing the firmware on both the devices it was found that communication was only possible from Arduino UNO to ESP8266 and not the other way round. A more electronic level research is required so as to figure out why this happened. The communication was tested directly and with a bi-directional TTL logic level shifter. The voltage conversions were verified but Arduino UNO would fail to recognize any input via serial or digital given to it from the ESP8266.

The idea of Multi-Master configuration was that the Arduino UNO and ESP8266 would collaborate and share information to successfully run the system. A pseudo protocol was even designed for this communication via message codes and data fragmentation but due to the hardware limitations it had to be scratched at the last minute.

Because of the failure of the Multi-Master configuration the architecture of the embedded system had to be shifted to a Master-Slave configuration in which ESP8266 was the Master and collaborating Arduino UNO would be the slave. Instead of the ESP266 asking for the sensor values the Arduino UNO connected to the sensors was programmed to broadcast the information containing reading of all the sensors on the serial port and whenever the ESP8266 would need it would read from the Serial Buffer.

The initial design of the system considered Arduino to be interacting with the Sensors and actuators since it has the standard 5 volts operation whereas the ESP8266 works on 3.3 volts. The DC Motor based pumps were connected and wired to the relay board along with the light and a test run with the ESP8266 controlling the relay directly and with a logic level shifter was conducted. In both the cases ESP8266 failed to provide enough current for the relay board to detect a change in the logic level for the control input.

The biggest problem which arises here was that ESP8266 was not able to provide any information to the Arduino UNO so as to use Arduino to control the relay board for controlling the actuators. This required some kind of a hack for ESP8266 to be able to relay some information to Arduino UNO somehow. For this to work 6 pins of the Arduino UNO were designated as INPUT by using the following command:

*pinMode(2,INPUT);*

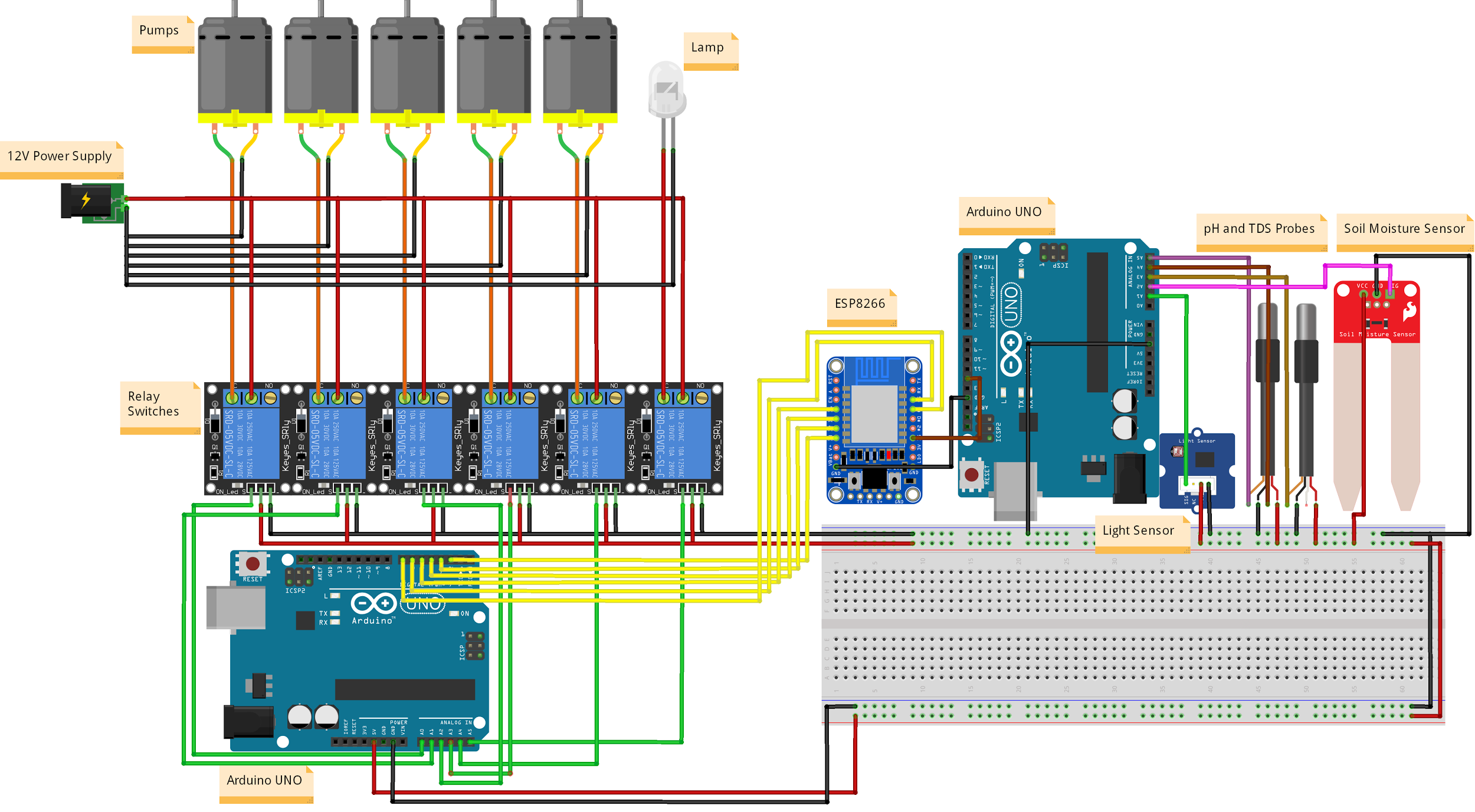
Even after configuring an input pin, and having the ESP8266 interface with it, the logic level would always be read as low on the Arduino. Interestingly, after configuring the pin mode of the I/O pin as input, we can also activate the internal pull-up resistors on the Arduino and have the input pin’s default state set to HIGH by simple:

*digitalWrite(2,HIGH);*

Now whenever a HIGH signal of 3.3v was sent to the Arduino on the pin with this kind of configuration the state of the pin would be read as HIGH, but when a LOW signal of 0 v would be sent to the Arduino the ESP8266 which is 5v tolerant would sink this voltage and bring it down to 0v reading a valid state of LOW on the Arduino.

This worked perfectly and we were able to assign another Arduino UNO controller for controlling the relay boards, completing the Master-Slave configuration successfully.

**3.1.3 Embedded System Design**



**Figure 3.1.1 The Circuit Design**

The basic layout of the circuit is shown in Figure 3.1.1 where we can see the ESP8266 in the center connected to two Arduino UNO boards, one connected to the sensors and other connected to the relay boards. The power to relay board and sensors is provided by the Arduino UNO boards and it is made sure that the whole system shares a common Ground. Please note that the breadboard is depicted in this diagram to show the common rails of 5 volts and ground for all the peripherals in this system. A more commercial approach would be to eventually design a PCB containing all these components on board, making it easier for installation.

**3.1.4 The Flow of Control**

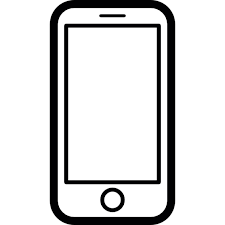
Untitled Diagram.png

**Figure 3.1.2**

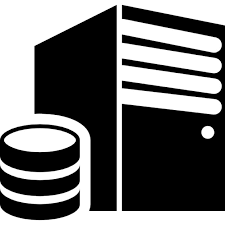
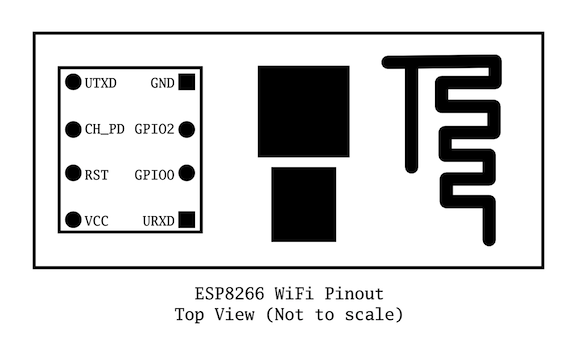
The overview of the System Flow of Control is depicted in the Figure 3.1.2 in a flowchart format. When the System boots, it connects to the Mobile Device’s hotspot and send in a HTTP GET request with the boot field set as true, for the server to know that the system has just started up. If the server has valid thresholds available for the system depending on the ID, a set of valid threshold values with valid field set as “1” in the JSON object are relayed to the embedded system via the mobile device. Once the thresholds are received by the system the ESP8266 starts accepting sensor readings from the Arduino UNO and matches them with the threshold limits. If there is any value which goes outside the boundary of the threshold limits, the relevant actuator for balancing pH, TDS, water or light are activated and HTTP GET request using the “getActuatorUpdate” service is sent to the server denoting which actuators were activated. If there is no threshold violation the system checks for a deflection in sensor readings by a delta value, after which the system updates the sensor values on the server. The Delta deflection is checked after any of the actuators are changed as well, since that would certainly become true and send updates to the server automatically. This is more of a design decision depending on the requirements and the reusability of a service.

**3.2 Fog Computing**

To control the control system of the hydroponic setup, we are using the architecture of fog computing to compute the data produced by the embedded system in the real time scenario, and log that data on the server.







**2.3.2 Mobile Device as a server to embedded system**

The mobile device is used as a mediocre computing device, which connects multiple plantations system(esp8266) through hotspot and try to provide service through http protocol. Following are few the services provided to embedded module:

* Receive the identity of of the embedded system and forward that information to the server so that it can be mapped to a particular user.
* Provide the thresholds of different parameters of the control system and update it whenever new values are updated on the server for that particular plant. This information is received from server through apple push notification, so that continuous pulling is not required and it saves the unnecessary.
* If the controlling parameters goes out of the threshold range, then mobile device provides the information as to what necessary steps should be taken, so that that particular parameter is within the threshold range.

HTTP request api’s created for ESP 8266:

* **getSensorDetails**:

Sample:  
http.begin("172.20.10.1", 8080,"?&fn=getSensorDetails&sensorId=esp001&ph=7.2&tds=50000000000&light=15.3&moisture=15.99&boot=true");

Sample response:

{

"tdsUpperThresholds" : 40,

"lightLowerThresholds" : 50,

"tdsLowerThresholds" : 10,

"moistureLowerThresholds" : 10,

"phLowerThresholds" : 5,

"phUpperThresholds" : 7

}

* **updateActuatorData**

Sample:

http.begin("172.20.10.1", 8080,"?&fn= updateActuatorData&sensorId=esp001&isPhBalance=7.2&isTdsBalance=50000000000&isLightBalance=15.3& isMoistureBalance=15.99&boot=true");

Sample response:

{

"tdsUpperThresholds" : 40,

"lightLowerThresholds" : 50,

"tdsLowerThresholds" : 10,

"moistureLowerThresholds" : 10,

"phLowerThresholds" : 5,

"phUpperThresholds" : 7

}

**2.3.2 Mobile Device as a client to the middleware server**

Mobile device is also responsible to maintain the required data for control system, according to the plant specific. These data is initialized on the server, who is even responsible to map the plant category, and is pushed to the mobile device. These data eventually becomes the governing parameter for embedded systems and try to balance the control system based on that.

Secondly, whatever data produced by the embedded system, is logged on the server, so that current status of the hydroponic system can be presented to the user.

**3.3 Web API’s**

The data for UI and the Fog Computer is provided by the middleware API server. The server is built using NodeJs[1] for handling load. The data is stored in MongoDB hosted by MongoLab[]. Both the API server(hosted in AWS) and MongoLab[2] are basic free accounts. The system allows requests from any domain after verifying user identity. The NodeJs platform is very simple and robust. This API server is made with ExpressJs[3] framework with ParrpostJS[4] for Authentication, MongooseJs[5] for Data modelling, AsyncJS[6] for making parallel and serial function calls, Node APN[7] for Apple push Notification service and more packages for Logging, Cookie Parsing, Body parsing etc. Different APIs are written for Fog computer and the Front end web application.

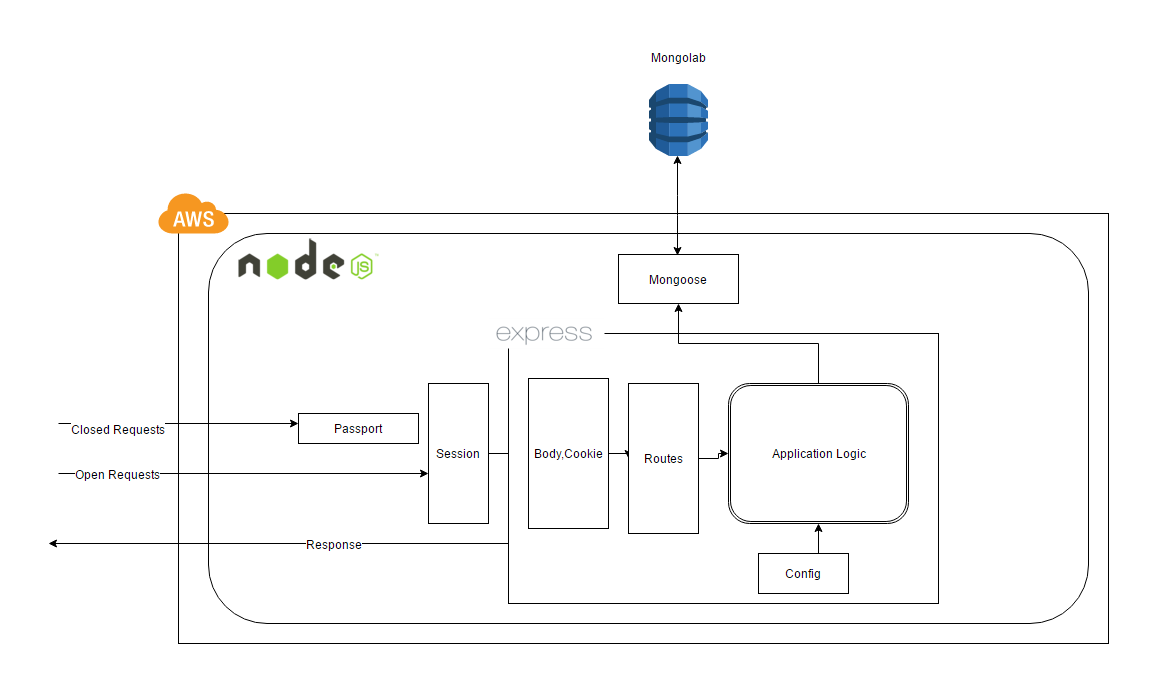
**3.3.0 Introcution to NodeJS[1] [[1]](#footnote-0)**

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](https://developers.google.com/v8/). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](https://www.npmjs.com/), is the largest ecosystem of open source libraries in the world.

As an asynchronous event driven framework, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently.

const http = require('http');  
const hostname = '127.0.0.1';  
const port = 1337;  
http.createServer((req, res) => {  
 res.writeHead(200, { 'Content-Type': 'text/plain' });  
 res.end('Hello World\n');  
}).listen(port, hostname, () => {  
 console.log(`Server running at http://${hostname}:${port}/`);  
});

Upon each connection the callback is fired, but if there is no work to be done Node is sleeping.This is in contrast to today's more common concurrency model where OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process—there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, less-than-expert programmers are able to develop scalable systems.

Fig 3.3.0 WEB API Architecture

**3.3.1 APIs for Front-End Web Application:**

The Web application makes the browser to get data needed using Ajax calls. The biggest part was to enable the APIs in a different domain from the web application and to enable user authentication. The web application itself cannot identify the user, instead the webapplication redirects the user to <http://api.humandroid.us/login> with making itself as a referrer. The API server gives a simple UI for login. Once the user enters proper credentials, the API server will set the session id in the form of cookie( incase of other REST clients, no cookie is set) and redirect back to web application. The data from the browser are sent to API server only using POST requests. Certain API requests will do more than just updating the data in db. For example, the API /api/updateandnotifyplant/ will update the thresholds for a given plant and sends a push notification to the Fog computer which is holding the plant so that it will take care of the rest.

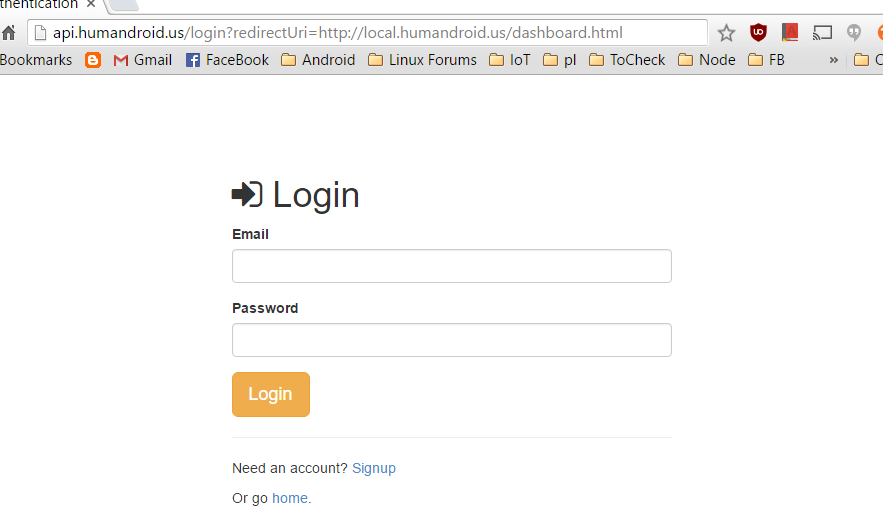


Fig 3.3.1 Login UI for Web Application with redirectUri

The open APIs are :

|  |  |
| --- | --- |
| /api | Welcomes the user |
| /api/login | Login API for other REST clients |
| /api/checkme | API tells whether the user is logged in or not |
| / | Welcome Page UI |
| /login | Login UI |

Table 3.3.1 The Oen APIs exposed by the server

The Closed APIs are:

|  |  |
| --- | --- |
| /api/requestdevice | Requests API to create a device for the user |
| /api/updatedevice | API for Updating device |
| /api/createchildplant | To Create a child plant |
| /api/updateandnotifyplant | To update the thresholds of the plant and |
| /api/logplantupdate | To update the Plant condition to the server |
| /api/logactuatorupdate | To update the Actuator status to server |
| /api/getalldevices | To get the list of devices for the user |
| /api/getrootplants | To get the list of Root (Default) plants |
| /api/getallplants | To get all the plants( Root + custom) |
| /api/getplantforarduino/<ARDID> | To get a plant using Arduino ID |
| /api/getuserdata | To get the details of the user |
| /api/getdevice/<id> | To get the details of the device |
| /api/getplant/<id> | To get the plant using its id |
| /api/getplantsforuser | To get all the plants for that user |
| /api/getlatestplantupdate | To get the recent status of a plant |
| /api/getplantupdates | To get the list of updates for a plant (customizable) |
| /api/getallarduinos | To get the list of all arduinos for the user |

Table 3.3.2 Secured APIs for all type of REST clients

The Table 3.3.2 doesn’t list all the APIs but the ones which are mostly used. All other utility APIs and Examples for all the APIs is available in the provided API doc.

**3.3.2 APIs for the Fog Computer**

The Fog computer gets all the data for the user by calling the APIs specific to Fog computer. It needs to know what are the thresholds for a plant, and to update the conditions of the plant and actuator to the server frequently. All the updates given by the Fog computer is saved in the Mongo database. The server is capable of handling multiple client requests from multiple devices. All the requests from the Fog computer are authenticated.

**3.3.3 API Server Configuration**

var express = require('express');  
var app = express({name:'IoT Server'});  
var port = process.env.PORT || 80;  
var vhost= require('vhost');  
var morgan = require('morgan');  
var cookieParser = require('cookie-parser');  
var bodyParser = require('body-parser');

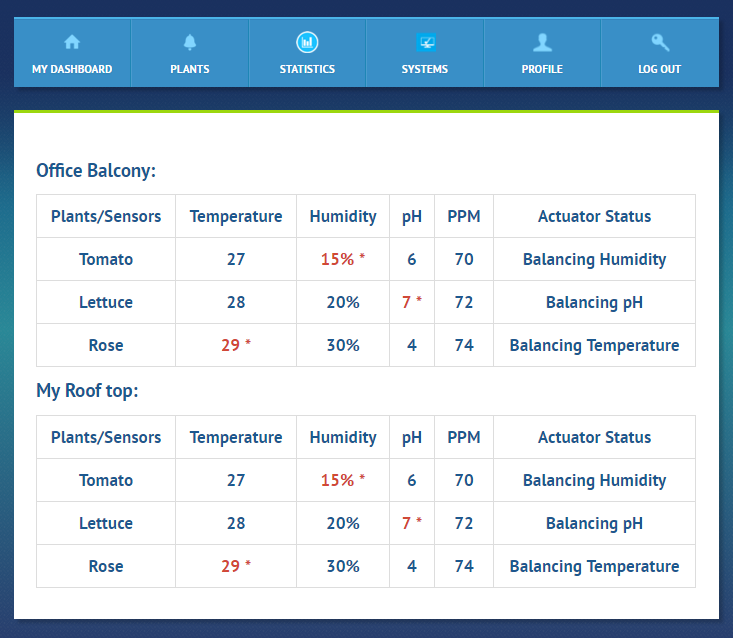
app.use(function(req, res, next) {  
res.header('Access-Control-Allow-Credentials', true);  
res.header('Access-Control-Allow-Origin', req.headers.origin || 'http://local.humandroid.us');  
res.header('Access-Control-Allow-Methods', req.headers['access-control-request-headers']||'GET, POST');  
res.header('Access-Control-Allow-Headers', req.headers['access-control-request-headers']||   
 '\*');  
res.header('X-Powered-By','IoT Hydrophonics');  
next();  
});

The above source shows the overall configuration of the API server for the cross-domain requests and the middlewares used for Cookie parsing, Body parsing, logging, and the API server itself (express).

**3.4 Web Application**

The application was developed using Bootstrap, HTML5, jQuery to create a responsive website and could be used in devices with varying screen resolutions.

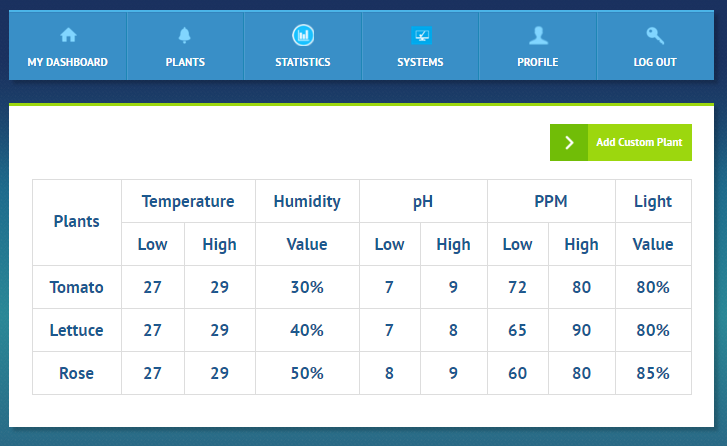
The user controls and monitors his/her system using the application. After the device is setup at the location specified by the user, all that the user has to do is go to the website register himself/herself and login to her profile. The user is presented with a user dashboard as soon as he/she enters his profile. The user dashboard presents the user with the current information about the various systems. According to the UI design the user dashboard would look like as it's shown in Fig.3.4.1.



**Fig.3.4.1 User dashboard**

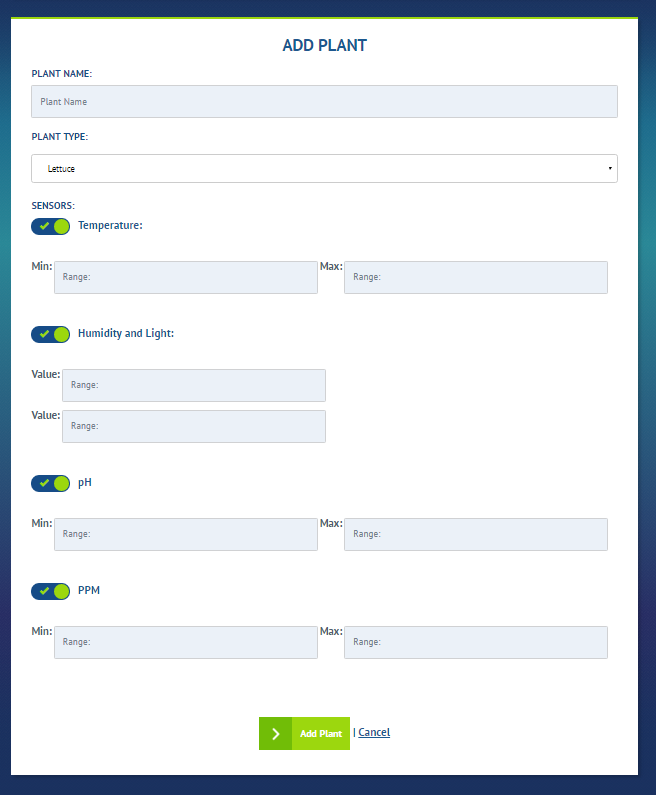
The user dashboard presents the user with all the systems he/she is controlling. The exact sensor details are being retrieved from the server where the data is automatically updated when there is a change in the threshold values set for the plant. When the threshold value goes above or below the threshold values set by the user for that particular plant then the value that is not within the range is changed to red color, this is to indicate to the user that the value is not within the range and the Actuator status is also displayed in the table so the user is aware that the threshold value is being taken care of or that the system is under control. If the actuator does not turn on or the balance of threshold value is not being taken care due to some mechanic failure of the motor or empty nutrition,acid,base or water tanks then that data is displayed so that the user is aware of the problem and takes care of it to maintain the system. The red color for those values and the \* was given after the feedback taken from real time users. We spoke some people who did gardening explained to them the concept of our system and asked them what they would like to see, It was told that they wanted to know if everything was okay and if there was a problem then what it was and how it could be solved. Keeping these points in mind we designed the dashboard in a way so that the user understands the system status in a glance. The user also wanted get notifications when the actuator turns on or off or when a sensor or any mechanical device fails in the system. We tried to implement this as well. We were successful in sending messages about the status if the actuator was turned on/off but we did not have enough time to figure out a way to detect mechanical failure and send information about it as well. This is one important part we are willing to implement in the future work regarding this project.

The user profile is exclusive for each user and other users cannot view the data regarding the systems and the custom plants created by the user. The user can view all the root plants and custom plant details in the plants tab. The fig 3.4.2 shows the table with the list of all the root plants and the custom plants added by the user and the add custom plant button. An import condition added here is that the user cannot change the values of the root plants but can refer to the values to customize his/her plants.



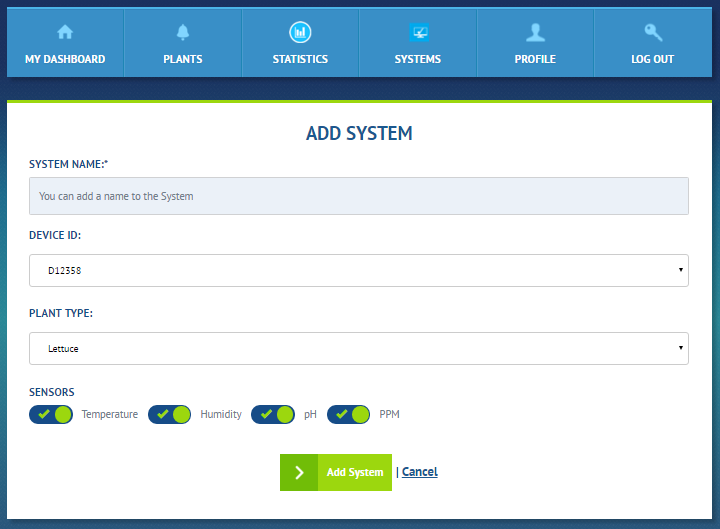
**Fig:3.4.2 Plant Details**

Clicking on the Add Custom plant would redirect the user to another page as shown in fig 3.4.3 where the user is allowed to create custom plant of his/her choice. The user is asked to name the custom plant and then the user has to select the plant type to view the values of the root plant and customize the plant accordingly. This is a page where he had trouble designing as we had to present the user with two details and get a completely different value from the user to save the details of the custom plant. The two values we had to present the user was the range within which he/she has to set the sensor values. For eg: The pH should be in the range between 0 to 14 and should not have any value above or below those values and the values of the root plants for their reference when they select the plant type dynamically.



**Fig: 3.4.3 Custom Plant**

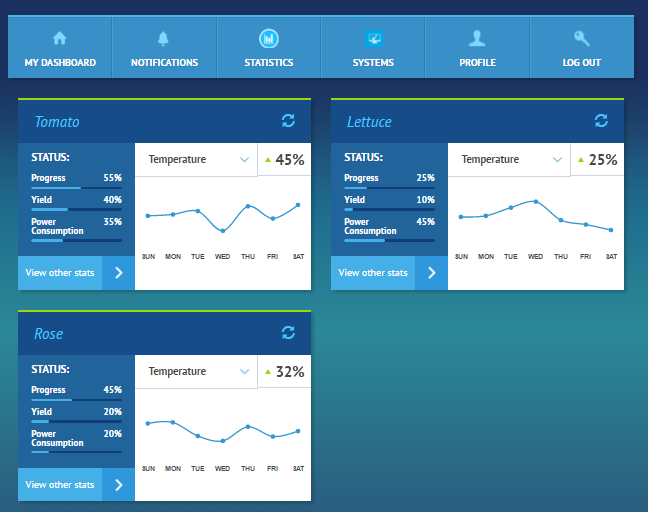
The data what we wanted to receive from the user is the value he/she wanted to set for the plant. First we thought of using dual sliders for this purpose which solved two of the parts the range to limit the users within which they had to set their custom plant values and the values to receive from the user. But we were were not able to change the dual slider value after the plant type was selected as the dual sliders value were retrieved on page load and was not updated on selection of plant type and changed accordingly. Due to time constraints we had to change that idea and use text fields to display the values of the root plants and add inter conditions to not accept values above or below the accepted range. But this is another change we are planning to do in future.



**Fig 3.4.4 New System**

Adding a new system is another important task a user would want to accomplish in this project from his/her end to control the system. In Order to complete this particular task the users are asked to name their system and select the device id that particular location is linked to, the device id is nothing but the Fog computing device as explained in [section 3.2]. The user is given the list of all the devices which his/her system locations connect to communicate sensor data and receive threshold values. Then he can select a Root plant or a custom plant added by him/her the full list of root plants and the user specific custom plants are given to the user in the plant type drop down list. After all this when the user clicks the add system the server is updated with all the details about the user and the new system details. Immediately the system is setup and the user can view it in the dashboard and the systems tab is populated where he can view the existing system data and details the screenshot is shown in the appendix.

The Fig 3.4.5 below shows the statistical data and the progress and yield of the various plants with respect to the existing systems.



**Fig: 3.4.5 Statistics**

The user can view the sensor value variation over a period of time and the progress of the plant growth the yield status and power consumption. All these values would stored and collected from the various hydroponic systems to form a network of hydroponic systems and this data could be used in the future to analyse and find out under which controlled environment the system gives a higher yield. This does not only help the user know the statistics of his one system but some day a pattern could be formed out of all the data that has been stored from the various systems.

The user can view the existing plants in a system by going to the system which gets populated under the system tab and can also edit the plant details and can even delete the plant and add another plant. The user is also given the option of deleting the system. All these features were included in website to replicate a real world requirements of the users so they can be extended and refined in the future. The api’s were also created to support this features. The systems and the plants are both generic in nature. New systems and plants can be added and deleted into the user profiles at his/her mercy. The user has full control and freedom to control and monitor his/her systems from anywhere and anytime. The web application was designed first and then developed. A paper prototype was prepared which was taken to users for their feedback and ideas and also understand their requirements and then changes where made and implemented. The users are prompted at every point to prevent them from making any mistakes. The Nielsen's heuristic evaluation was done on the application so that it follows at least most of his 10 principles. A lot of reach and understanding was done designing the user interface so that it’s easy for the users to use and also have everything they would need to control and monitor their systems at ease. Both the utility and usability of the system was kept in mind while designing the system.

**Result**

As we had our entire setup ready, we were successful in getting values from sensors and able to communicate with fog computing device and when values were above or below threshold range, we were able to turn on actuators accordingly and turn off them when values are back to threshold.

On other hand, we were able to communicate values from fog to server and web end is able to retrieve live data from server and also able to update the threshold values for custom plants. Whenever this update is made a push notification is successfully sent to fog device.Meanwhile the server was load tested with 100k requests with 1000 concurrency and the result was successful.

**Conclusion**

With increasing soil erosion and scarcity of resources, it is time to protect our natural resources and with this mechanism there is no need to substitute our food or luxury to meet these needs. With substantial decrease in the overall prices of hardware, it becomes more feasible and economical to apply this method for a large setup too.

Also with this type of cultivation, it is possible to grow acres and acres of crops indoors in a very limited area there by saving a lot of space to grow more crops and plants.

With this kind of multi-tier architecture and communication, this decreases internet dependency and limits data traffic required to communicate with server. And also user experience is much more interactive with plants and a user can control conditions according to plant type and size of the system.

**Future Work**

As of this phase we are using mobile for Fog computing, this can be replaced with upcoming hardware devices like Raspberry pi-3 which also contains an inbuilt Wi-Fi module and will be much cheaper. We can include some more parameters like water temperature control system to get more precious and better yield of crop.

There is a potential chance for scientists and agriculturists to experiment on plant growth and sustainability by dynamically varying conditions around the plant.

**References**

[1] - <https://nodejs.org>

[2] - <https://mlab.com/>

[3] - <http://expressjs.com/>

[4] - <http://passportjs.org/>

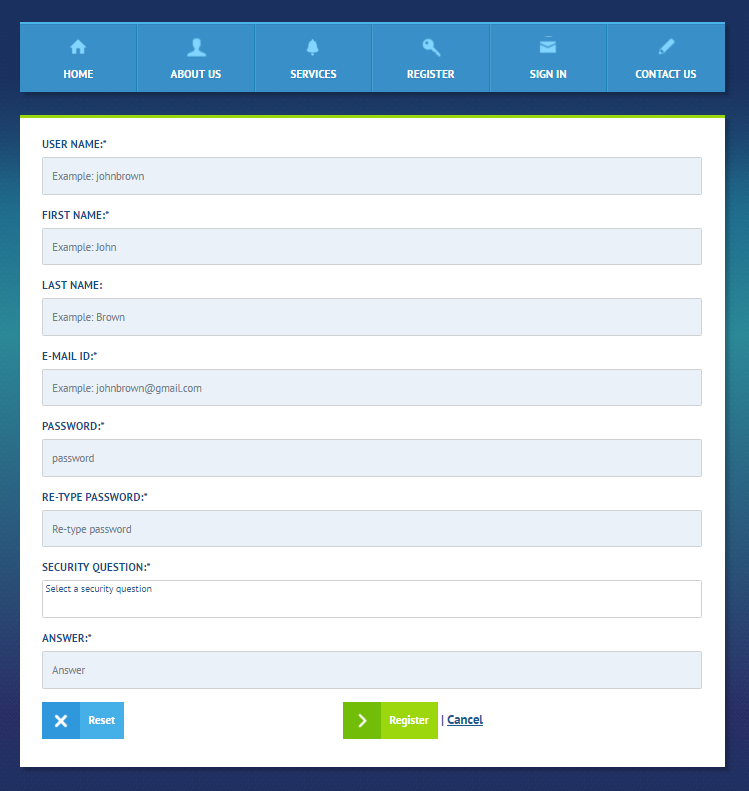
[5] - <http://mongoosejs.com/>

[6] - <https://github.com/caolan/async>

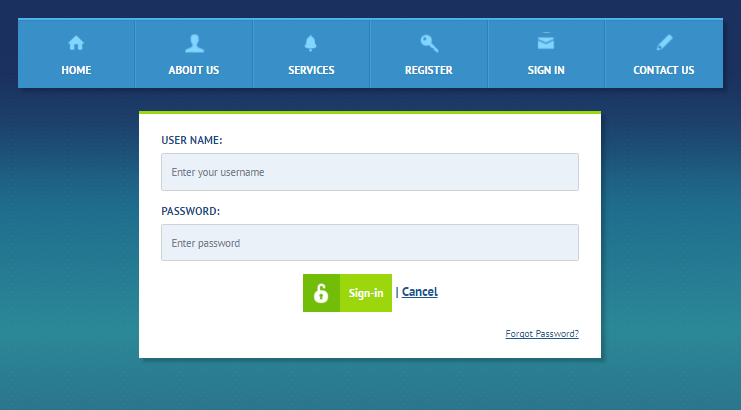
[7] - <https://github.com/argon/node-apn>

**APPENDIX:**

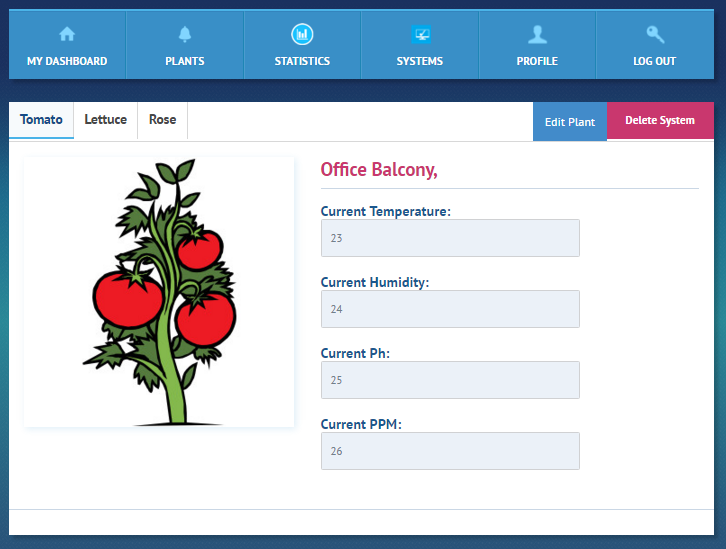
1. **SCREENSHOTS**



**Registration Form**

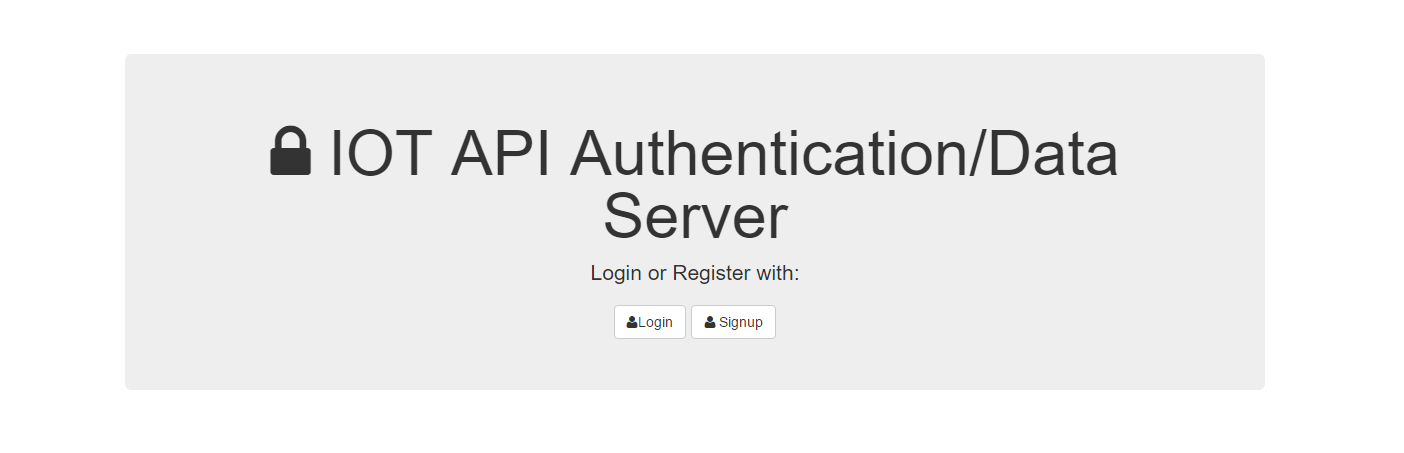


**Sign in**

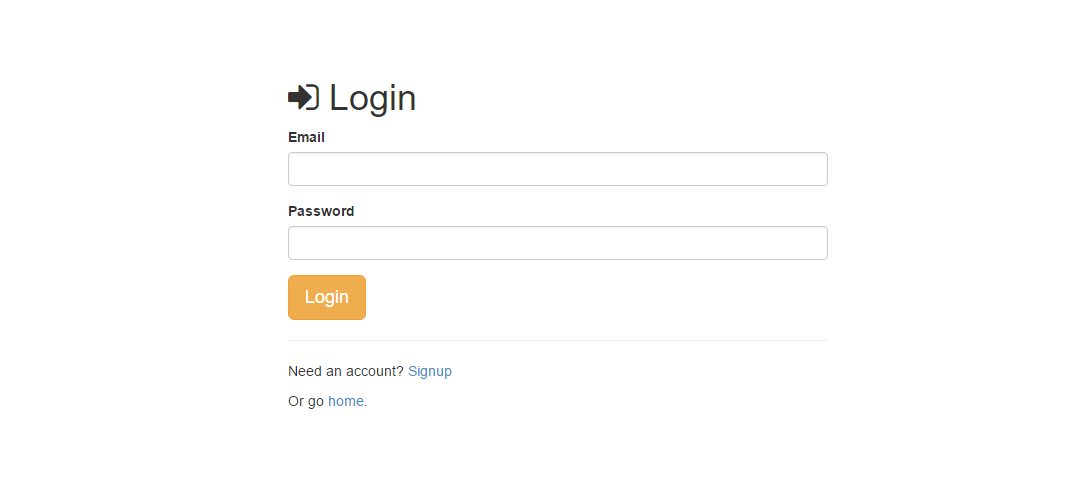


**Existing System**

**Auth Server home**



**Auth Server login from referrer:**



1. The entire content for this section is taken from the Nodejs.org web site for better explantation. [↑](#footnote-ref-0)