



**Deakin University**

Kelp the World

**Project Handover**

27/09/2019

**Project Sponsor**

Kevin Lee

**Project Team**

KelptheWorld

**Bronte Jurgens, 217015344**

**Greg McIntyre, 218356779**

**Sean Pain, 218137385**

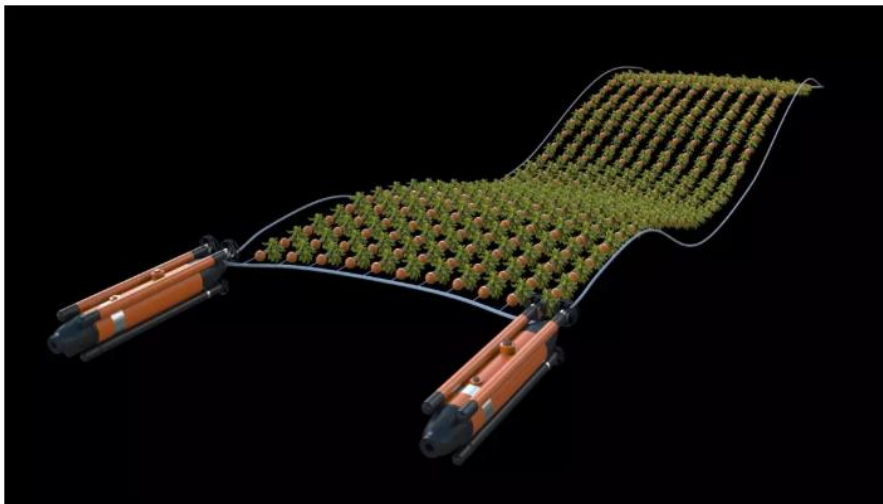
**Document Version 2019-09-27**

## Purpose

This document defines the transfer of all relevant information and artefacts produced during SIT209. With this document, a new member can identify all key aspects and artefacts of the project and have access to key systems or configurations.

## Project Description

- A floating kelp farm is a concept that would benefit from a real-life prototype.
- There is a concept to create frames of kelp farms that float just below the surface and constantly moved downward to accommodate the growth of the kelp, pressure changes at depth and the amount of buoyancy/water displacement will required will require constant management, the kelp requires a certain temperature of water, amount of light to remain healthy. An independent pump with its own buoyancy control will also be required to move cold water from the depths, mass monitoring will be useful for knowing when forests should be harvested/managed, location tracking will be important for management during storms and other emergencies.
- The 'ultimate goal' would be to create an opensource, robust, scalable, cost efficient, easily deployable, prototype of this frame, with mass sensors, light sensors, pressure sensors, gyroscopes, GPS. And have various interfaces for different roles to monitor huge numbers of frames; farmers, maintenance people, clients, environmental groups, press etc.
- This product is a concept only, this would be the first public prototype.



Fast Company Kelp Farm Concept [3]

## References

1. <https://oceana.org/blog/seaweed-could-be-scrubbing-way-more-carbon-atmosphere-we-expected>
2. <https://carboninstitute.org/kelp-and-carbon-sequestration-bringing-terrestrial-carbon-accounting-to-the-deep-sea/>

## Artefacts List

Artefact Name	Artefact Type	Revision Number	Notes
Source Code	GitHub Repository	2019-09-27	Live
MQTT Login	Image	2019-09-27	
MQTT Topic Format	Image	2019-09-27	
MongoDB Login	Image	2019-09-27	-
MongoDB Format	Image	2019-09-27	-
-	-	-	-

## Source Code

<https://github.com/gregorymcintyre/sit209-iot-app.git>

The source code for this project can be found at this git public repository, as this is a if you would like to contribute to this project or use it as a foundation please feel free to contact the git owner or credit the original authors.

## Control Device

sit209-iot-app/Kelp Farm Control Units/



The initial concept control device (CD) was created as a local Arduino Uno, an ethernet board and a data logger. This was a simple prototype used to transmit a data stream to cloudMQTT. Login details are below. As well as the topics transmitted by the device

customer.cloudmqtt.com

gmcintyre@deakin.edu.au

6IR00PuQK0K(Va\_d

Edit

Delete


Received messages	
Topic	Message
arduino_1/location	Brisbane
arduino_1/sensorData/time	1568330500000.00
arduino_1/sensorData/temperature_celsius	21.00
arduino_1/sensorData/light	5.00
arduino_1/sensorData/pressure	1.00
arduino_1/sensorData/bouyancy	1.10
arduino_1/sd	{"deviceId":"arduino_1","sensorData":{"ts":1568330529936,"loc":"Brisbane","li":23,"pres":8,"bouy":50}}

## Backend

sit209-iot-app/MQTT API/

The goal of the backend was the subscribe to the sensor data messages on the CloudMQTT platform and to log the data into MongoDB for historical data and to also interpret the sensor data values and send commands to the farm device if changes were required.

Below is the MongoDB login page. The password to login is 'SIT209MongoPass!'.

 mongoDB.

We've launched a unified login experience that gives you access to MongoDB Cloud, Support, and Jira with a single identity.

LEARN MORE

Login

span@deakin.edu.au

Usually, this is the email you used to register.

Forgot password?

Login

Register for a new account

Below is the object format within the Mongo Database.

```
_id: ObjectId("5d8809259ae3f325846c5d4f")
  sensorData: Array
    > 0: Object
    > 1: Object
    > 2: Object
    > 3: Object
    > 4: Object
    > 5: Object
    > 6: Object
    > 7: Object
    > 8: Object
    > 9: Object
    > 10: Object
    > 11: Object
    > 12: Object
    > 13: Object
    > 14: Object
    > 15: Object
    > 16: Object
    > 17: Object
      ts: 1568330529936
      temp: 21
      li: 23
      pres: 8
      bouy: 50
    > 18: Object
    > 19: Object
  deviceId: "arduino_1"
  __v: 19
```

## GUI

[sit209-iot-app/ui-interface/](#)

The end goal of the front-end was to create 3 interfaces. One as a farmer dashboard with their individual allocated devices. These would display the current status of the Arduino sensor readings. A second page would be a developer interface would display all the incoming live data and graphical dashboard to gain insights from. A third public page would have public insights into the weather and its effect on crops more generally.

### Kelp The World

Home  
KTW  
Devices

Device ID : 274452  
Light : 5.00 lux  
Temperature : 27.00 degrees  
Pressure : 1.00  
Bouyancy : 1.10

Light 5.00

## Business Features

Feature	Client Sign-Off State	State	Notes
API	09-09-2019	Complete	MQTT JSON
Arduino Concept Control Device	27-09-2019	In Progress	True sensors have not been implemented
Backend - Base	23-09-2019	Complete	
Backend – Handle sensor values	27-09-2019	In Progress	Method for sending commands back to device has not been created
GUI (Frontend)			

### 1. Create a service API for KelpTheWorld

A communication protocol was implemented using MQTT in a JSON format for passing information from control device to the backend. The CloudMQTT web service was used as a medium for this

### 2. Create an Arduino frame concept to receive sensor data and transmit to web server

A simple working Arduino control device was made, the device is currently only a publisher. The backend did not progress far enough to transmit buoyancy control information back to the control device. The Arduino CD has the capability to be connected to a light and pressure sensor quite simply, the mass sensor usable in an under-water variable pressure environment has not be found at this point. The buoyancy control has not been developed.

### 3. Create an Arduino control for water pump control

The Arduino pump controller was not progressed on. It could be developed from the control device using a water temperature sensor and the same buoyancy control methods as implemented in the control device

### 4. Create a web server to receive data from various farms

The backend webserver was setup so that it was able to handle any number of farms required. All the farmer needs to do is add the farm name to the approved device list in the backend and the backend will start processing the data from that device that it sees on the CloudMQTT platform. The data will then be processed and logged into MongoDB.

### 5. Extend backend webserver to process sensor data and send control commands

The backend extension which would allow for automated control of the farms was not progressed on. To create this there would need to be a command system to send commands back to the Arduino device.

### 6. Create a web interface to display data

- a. Maintenance
- b. Farmers
- c. Public

## Planned Work

Planned Feature	State	Sprint	Notes

## Control Device

Planned work for the control device would include, selection and addition of the appropriate hardware enabling the device to pass real life information. Other mediums such as Raspberry PI or other simple PCI controllers could also be programmed to work as control devices in the future as technology, price positions and hardware availability change.

The control device also needs to be developed to subscribe to a topic of buoyancy data in order to react to the backend logic, as the backend develops this will be more realistic. It is expected for data integrity that this would arrive as a JSON object possibly in a format such as below. For consistency it would be best to send an intended pressure value in atmospheres in order to reduce the amount of processing the CD would be required to perform, this would reduce the required processing ability of the device and possibly reduce costs.

Topic /aruduno\_1/control  
{ "deviceid": "arudino\_1" { "boyancy": 1.5 } }

Implementing a dual subscription would allow us to change the topic data format from;

Arudino\_1/sd  
{ "deviceid": "arduino\_1", "sensorData":  
{ "ts": 1568330529936, "loc": "Brisbane", "li": 23, "pres": 8, "bouy": 50 } }

To something more descriptive and detailed such as;

### Publish

Arudino\_1/details  
{ "deviceid": "arduino\_1", "loc": "Brisbane" }  
Arudino\_1/sensorData  
{ "time": 1568330529936, "light": 23, "pressure": 8, "bouyancy": 1.5 }  
Arudino\_1/pump  
{ "time": 1568330529936, "temperature": 10, "bouyancy": 1.5 }

### Subscribe

Arudino\_1/control/cd  
{ "deviceid": "arudino\_1" { "buoyancy": 1.6 } }

```
Arudino_1/control/pump  
{“deviceid”:”arudino_1”{“buoyancy”:1.8}}
```

## Backend

### 1. Automated sensor data handling

This would allow the backend to handle the sensor data coming from the farms via CloudMQTT, then interpret the data and send commands back to the respective Arduino device for changes to be made. In order to complete this a method of communication would need to be setup between the backend and the Arduinos.

### 2. Enhance existing backend to suit future improvements to Arduino devices

To allow for planned upgrades to the Arduino devices that were mentioned above, the backend needs to be enhanced and modified to handle the changes. This would involve adjusting the data structure for the incoming sensor data, along with the data structure for the data going into MongoDB.

## GUI

### 1. Create a react-web app with a local server to visually represent the application for the end user

A react application was created as a ui-interface folder within the kelp farm application.

### 2. Import router & links to different pages based url pathways

Router links were added to the app.js file. This would be the main access point of what would run at build time. Any components would be communicated with here by importing “react-router-dom” and using Link and Router functions. Within a <Router> fragment <Route/> would communicate links to components and <Link/> would communicate to the url pathway.

```
Import {BrowserRouter as Router, Route} from “react-router-dom”  
Import {Link} from ‘react-router-dom’
```

```
<Route path=”/” exact component={Devices}/>  
<Link to=”/” className = “navbar-brand”>KTW</Link>
```

### 3. Create a navbar to make each page visually accessible to the user

This can be done through <nav></nav> tags within the app.js file or through a navbar component. Planned work could include personalising the navbar file and adding extra functionality and filtering within a navbar component to clean up the app.js file. The <Navbar/> component could be then tagged in the app.js file and all navbar code be moved and altered within this component.



#### 4. Add components that will contain the functionality to call from MongoDB data

A device and user component were added to catch data from the Arduino through the back-end. To set this up propTypes were imported and set the state of each piece of data to be used as well as functions that altered the value of each data value when there was a change in reading.

Import propTypes from "prop-types";

```
this.onChangeTemperature = this.onChangeTemperature.bind(this);
this.onChangeLight = this.onChangeLight.bind(this);
"
onChangeTemperature(e) {
  this.setState({
    temperature: e.target.value
  });
}
```

The componentDidMount() function adds a set of initial values to work with for testing before any live data can come through.

'Axios' was the chosen library to call from the MongoDB database, however, although being able to see data coming through the terminal was unable to display from a browser. Testing with fetch also brought up the same issues. Planned work could include testing these libraries with different http requests with third party data to analyse the correct process of access points. This would be implemented within each individual component depending on the required data for that page.

For display purposes at this stage dummy data was input into the web page. This was displayed through the render() section. Through App-content div's heading tags were used to display the required data on the screen as per the following format.

```
<h1 style={styling}>Device ID: {this.state.deviceId}</h1>{"\n"}
```

By separating the containers in which this data is displayed any required data could be split and displayed on different pages or different areas of an individual page.

#### 5. Use CSS for styling and add embed styling where necessary

For positioning of individual features on the page container div's can be used to class specific visual representation to separate blocks.

[ Import './App.css' ] can be used at the beginning of the app.js file and will run for any children components as well. Within the app.css file planned work could include creating extra containers for other sections on a farmers visual dashboard for bespoke sizing, colouring and padding.

Eg.

```
.lightDashboard {
  display: flex;
  justify-content : space-around;
```

```
width: 200px;
height: 200px;
left: 480px;
margin: 40px 0;
padding-top: 20px;
padding-bottom: 40px;
background-color: #FFA500;
}
```

## **6. Add more page components**

To follow through with initial plans extra routing would need to be added with links to further components from the App.js file. Within the components folder a new component home.js (public page) and dashboard.js (developer management page). The home page could use an API to news feeds on current articles regarding climate change and agricultural trends. A fetch() function could be called from the public page to current weather data as well as to pages that have current articles on the topic. Future planning could also include the use of Plotly to display the fetched data from the Arduino in a way that will visually represent it for farmers to understand the readings properly in a dashboard format.

# Open Issues

## Control Device

The limitation of the MQTT data packet (128 Bytes) has been an issue since sprint 3. the backend required development to handle more topics than it currently does. this has already been elaborated on in the future work section.

## Backend

One of the main issues is the size limit of the MQTT data packet coming from the Arduino. This means that we will not be able to expand the amount of data being sent to the backend without creating a solution to handle separate pieces of data from the same device. As we will have to interpret which device the data belongs to when there are many devices running.

## GUI

Issues rose in the connection between the front end and the backend. While the data streamed through the terminal issues arose when viewing the json data in a browser. Once this could properly be posted onto the localhost server calling from the front will prove simpler.

# **Lessons Learned**

## **Control Device**

The Arduino was a great prototyping piece of hardware, but the limitations of the MQTT data format was an interesting hurdle to overcome.

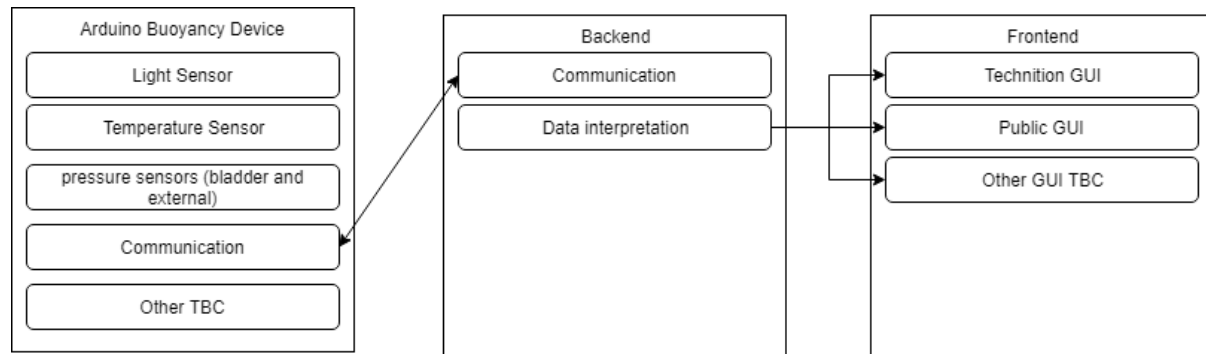
## **Backend**

Implementing the backend strengthened my knowledge on the MQTT messaging protocol, while also expanding my knowledge on the MongoDB platform and Node JS.

## **GUI**

Implementing the front-end with the React framework introduced multiple new concepts that can be associated with a visual interface. In the struggle to connect the front-end and back-end through research, I gained insight into the fetch API and the axios library in requesting data. I learned the importance of creating components not just for individual pages but for any repeated sections (i.e. Header) to clean up and reduce the amount of repeated code.

# High-level architecture of the product



## User Manual

### Control Device

[sit209-iot-app/Kelp Farm Control Units/Arduino/KelpFarmControlDevice/README.md](https://github.com/sit209-iot-app/Kelp_Farm_Control_Units/Arduino/KelpFarmControlDevice/README.md)

### KelpFarmControlDevice.ino

Kelp Farm Control Device

/\*----- Program: KelpFarmControlDevice

Description: Basic MQTT node using Arduino Uno

Hardware: Arduino Uno R3, Ethernet Shield (W5100), DHT11.

Software: Developed using Arduino 1.8.5 IDE

Libraries: -PubSubClient -SPI -Ethernet -DHT

Date: 2/9/2019

Author: Greg McIntyre -----\*/

### ##Directions for use

- make sure ip reflects an appropriate ip address for the device IPAddress ip(192, 168, 20, 7); <<<<<<< HEAD
- update

```

#define PUB_TIME "arduino_1/sensorData/time"          // MQTT topic for temperature [C]
#define PUB_TEMP "arduino_1/sensorData/temperature_celsius" // MQTT topic for temperature [C]
#define PUB_LIGHT "arduino_1/sensorData/light"        //
#define PUB_PRESSURE "arduino_1/sensorData/pressure"  //
#define PUB_BOUYANCY "arduino_1/sensorData/bouyancy"  //
=====
- update to reflect unique identifier for your control unit (eg. arduino_2)
>>>>>>> 0dd07d7ce39c84d69a0939757aceb684cb95774b

```#define PUB_LOC "arduino_1/location"
#define PUB_TIME "arduino_1/sensorData/time"
#define PUB_TEMP "arduino_1/sensorData/temperature_celsius"
#define PUB_LIGHT "arduino_1/sensorData/light"
#define PUB_PRESSURE "arduino_1/sensorData/pressure"
#define PUB_BOUYANCY "arduino_1/sensorData/bouyancy"

<<<<<<<< HEAD
#define SUB_LED "arduino_1/led"          // MQTT topic for LED```

to reflect unique identifier for your control unit (eg. arduino_2)

=====
#define SUB_LED "arduino_1/led"```
>>>>>>> 0dd07d7ce39c84d69a0939757aceb684cb95774b

```

## Backend

Prior to running the backend, you will need to install the following libraries:

- mqtt
- express
- body-parser
- mongoose
- random-int

You will then be able to start the server in the /MQTT API/ folder with the command:

*'npm start'*

If you need to add another device, you can insert the device name into the list object *'approved devices'*.

## GUI

Prior to running the application, the following libraries will need to be imported:

Axios, react-router-dom, semantic-ui-react, propTypes through

`'npm install {insert library}'`

Make sure the server is running with the live data feeding through the backend from the Arduino. Move into the ui-interface folder within the application folder and run :

`'npm start'` from the terminal.

This will open the browser with <http://localhost:3000>

## Sign-off

We KelptheWorld have included all relevant material which is agreed to be included in this handover. If an artefact is not included, it is stipulated in the Planned Work section or artefacts list.

Date:

1/10/2019

---

Signed



---