CTIS-411 SENIOR PROJECT-I

SOFTWARE DESIGN DESCRIPTION
(SDD)

Team 5 - Members

Berk Özdoruk
Erdoğan Yağız Şahin
Oğuzhan Özkan
Ömer Levent Durdalı

Project Supervisor

Dr. Cüneyt Sevgi

CTIS
Information Systems
and Technologies

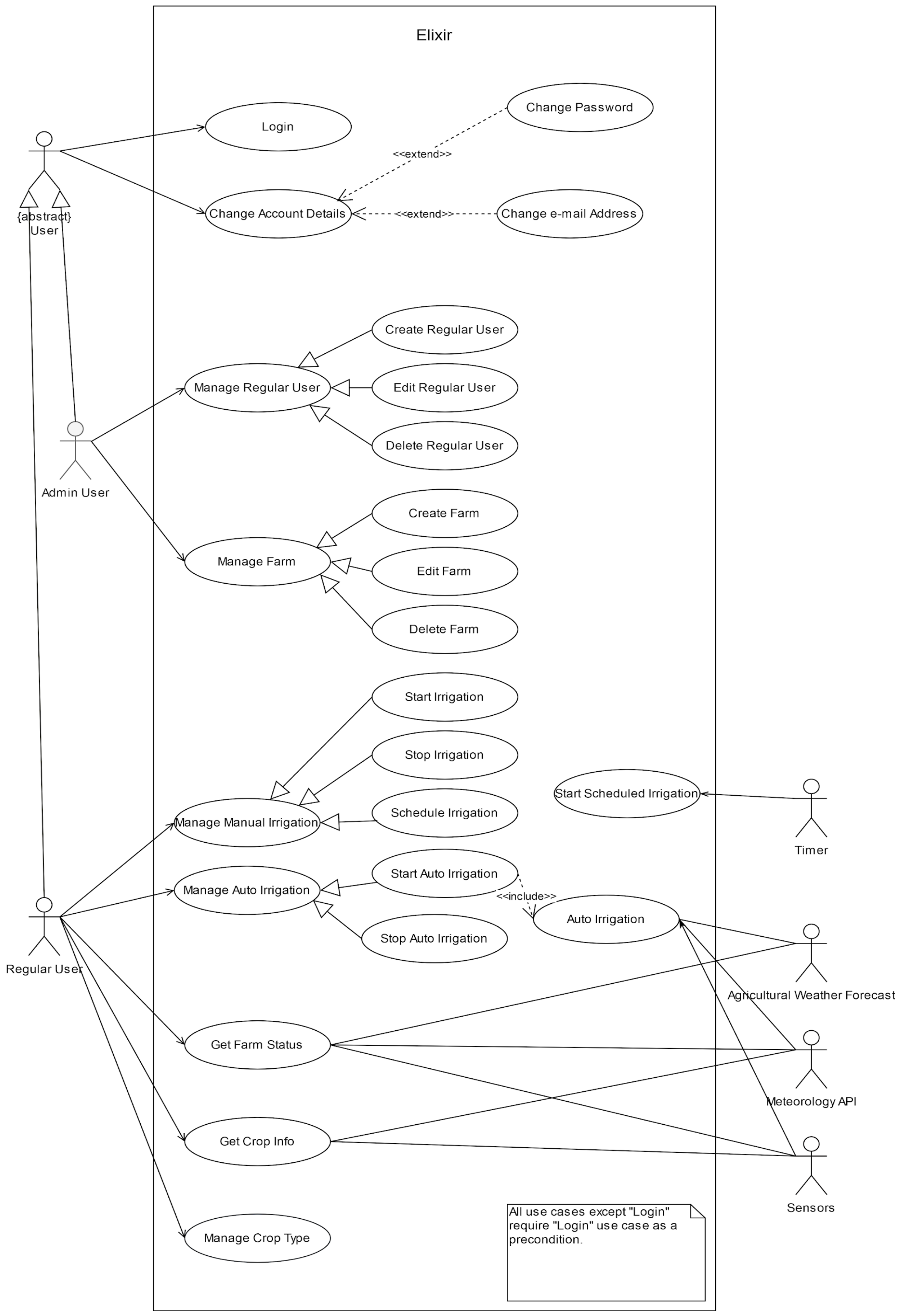
Table of Contents

List of Figures

List of Tables

1. ANALYSIS MODEL & PLANNING

1.1. FUNCTIONAL REQUIREMENTS (Use Case Diagram)



All use cases except "Login" require "Login" use case as a precondition.

1.2. NON-FUNCTIONAL REQUIREMENTS

1.2.1. Performance

- The system shall display the visualized output of farm and crop information to the user in less than 3 seconds.

1.2.2. Security

- The system shall communicate over encrypted channels for both the web application and the MQTT devices.
- All the sensitive data shall be encrypted both in rest and in transit.

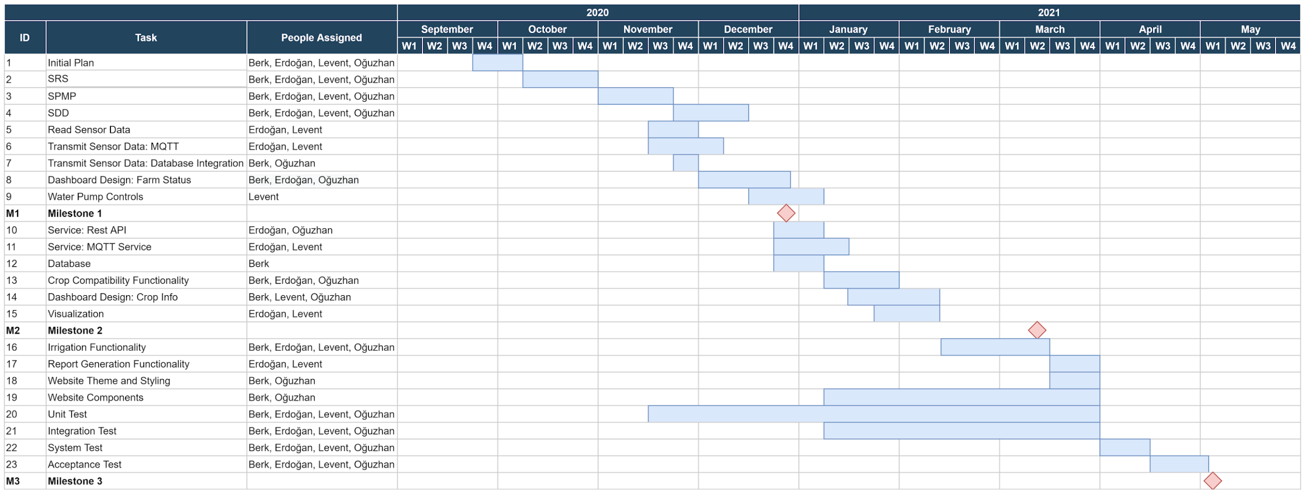
1.2.3. Reliability

- The devices within the MQTT network shall have the fault tolerance mechanisms to restart both the devices and the services when they stop working.

1.2.4. Availability

- The system shall have an availability rate of 99.99%.

1.3. SOFTWARE INCREMENTS



ID	Task	People Assigned
1	Initial Plan	Berk, Erdoğan, Levent, Oğuzhan
2	SRS	Berk, Erdoğan, Levent, Oğuzhan
3	SPMP	Berk, Erdoğan, Levent, Oğuzhan
4	SDD	Berk, Erdoğan, Levent, Oğuzhan
5	Read Sensor Data	Erdoğan, Levent
6	Transmit Sensor Data: MQTT	Erdoğan, Levent
7	Transmit Sensor Data: Database Integration	Berk, Oğuzhan
8	Dashboard Design: Farm Status	Berk, Erdoğan, Oğuzhan
9	Water Pump Controls	Levent
M1	**Milestone 1**	
10	Service: Rest API	Erdoğan, Oğuzhan
11	Service: MQTT Service	Erdoğan, Levent
12	Database	Berk
13	Crop Compatibility Functionality	Berk, Erdoğan, Oğuzhan
14	Dashboard Design: Crop Info	Berk, Levent, Oğuzhan
15	Visualization	Erdoğan, Levent
M2	**Milestone 2**	
16	Irrigation Functionality	Berk, Erdoğan, Levent, Oğuzhan
17	Report Generation Functionality	Erdoğan, Levent
18	Website Theme and Styling	Berk, Oğuzhan
19	Website Components	Berk, Oğuzhan
20	Unit Test	Berk, Erdoğan, Levent, Oğuzhan
21	Integration Test	Berk, Erdoğan, Levent, Oğuzhan
22	System Test	Berk, Erdoğan, Levent, Oğuzhan
23	Acceptance Test	Berk, Erdoğan, Levent, Oğuzhan
M3	**Milestone 3**	

Figure x: Gantt Chart

Milestone-1: (Wednesday December 23, 2020) - 1 st Increment

- Tasks
  - **Read Sensor Data**
  - **Transmit Sensor Data (MQTT)**
  - **Transmit Sensor Data: Database Integration**
  - **Dashboard Design: Farm Status**
  - **Website Dashboard for visualization of sensor and API data**
- Use Cases
  - **Get Farm Status**

Milestone-2: (Wednesday March 17, 2021) - 2 nd Increment

- Tasks
    - **Water Pump Controls**
    - **Service: Rest API**
    - **Service: MQTT Service**
    - **Database**
    - Crop Compatibility Functionality
    - Dashboard Design: Crop Info
    - Visualization
- Use Cases
    - **Login**
    - **Change Account Details**
    - **Change Password**
    - **Change E-Mail Address**
    - **~~Manage Crop Type~~ Upload Crop Signature File**
    - **Create Regular User**
    - **Create Farm**
    - **Auto Irrigation**
    - **Start Irrigation**
    - **Stop Irrigation**
    - **Manage Regular User**
    - **Edit Regular User**
    - **Delete Regular User**
    - **Manage Farm**
    - **Edit Farm**
    - **Delete Farm**

Milestone-3: (Wednesday May 5, 2021) – Final Increment

- Tasks
    - Irrigation Functionality
    - Report Generation Functionality
    - Website Theme and Styling
    - Website Components
- Use Cases
    - Manage Manual Irrigation
    - Schedule Irrigation
    - Manage Auto Irrigation
    - Start Auto Irrigation
    - Stop Auto Irrigation
    - Start Scheduled Irrigation
    - ~~Get Crop Info~~ Provide Advice for Crop Suitability
    - Auto Irrigation Extensions

1.4. FRAMEWORKS, LIBRARIES, SERVICES, DATABASES, APPLICATION PROGRAMMING INTERFACES (APIs)

1.4.1. Frameworks & Runtime Environments

- Jest: Jest is a testing framework for JavaScript which will be used for validation & verification activities of our backend (Node.js) and frontend (React.js) codebase of our web application.
  - Integration Date: 1st Increment
- Node.js: Node.js is a runtime environment for JavaScript which will be used to run JavaScript for the backend of our web application.
  - Integration Date: 1st Increment
- React.js: React.js is a web framework for JavaScript which will be used to develop the frontend of our web application.
  - Integration Date: 1st Increment
- unittest: unittest is Python's built-in unit testing framework which will be used to run the unit tests for our Python codebase.
  - Integration Date: 1st Increment

1.4.2. Libraries

- Plotly.js: Plotly.jsis a data visualization library for JavaScript. Plotly.js will be used for the visualization of farm and crop information.
  - Integration Date: 1st Increment
- Eclipse Paho MQTT Python client library: Paho is an MQTT client library. It will be used for our pub/sub infrastructure to send sensor data to the Mosquitto broker from a Raspberry Pi.
  - Integration Date: 1st Increment

1.4.3. Services

- Azure Virtual Machines: It will be used to create a virtual machine to host our web application.
  - Integration Date: 2nd Increment
- Eclipse Mosquitto: Mosquitto is a broker service for the MQTT protocol. It will be used to implement a publish/subscribe infrastructure.
  - Integration Date: 1st Increment
- Git: It will be used for version control for our repositories.
  - Integration Date: 1st Increment
- Google Cloud Platform Compute Engine: It will be used to create a virtual machine to host our Mosquitto Broker and Telegraf agent.
  - Integration Date: 1st Increment
- SonarQube: SonarQube is a continuous inspection tool. It will be used for the inspection of bugs, vulnerabilities, and code quality.
  - Integration Date: 2nd Increment
- Telegraf: Telegraf is an agent for collecting and processing metrics. Telegraf will be used along with its MQTT consumer plugin in order to subscribe to MQTT topics (sensor values) and write them to InfluxDB.
  - Integration Date: 1st Increment

1.4.4. Databases

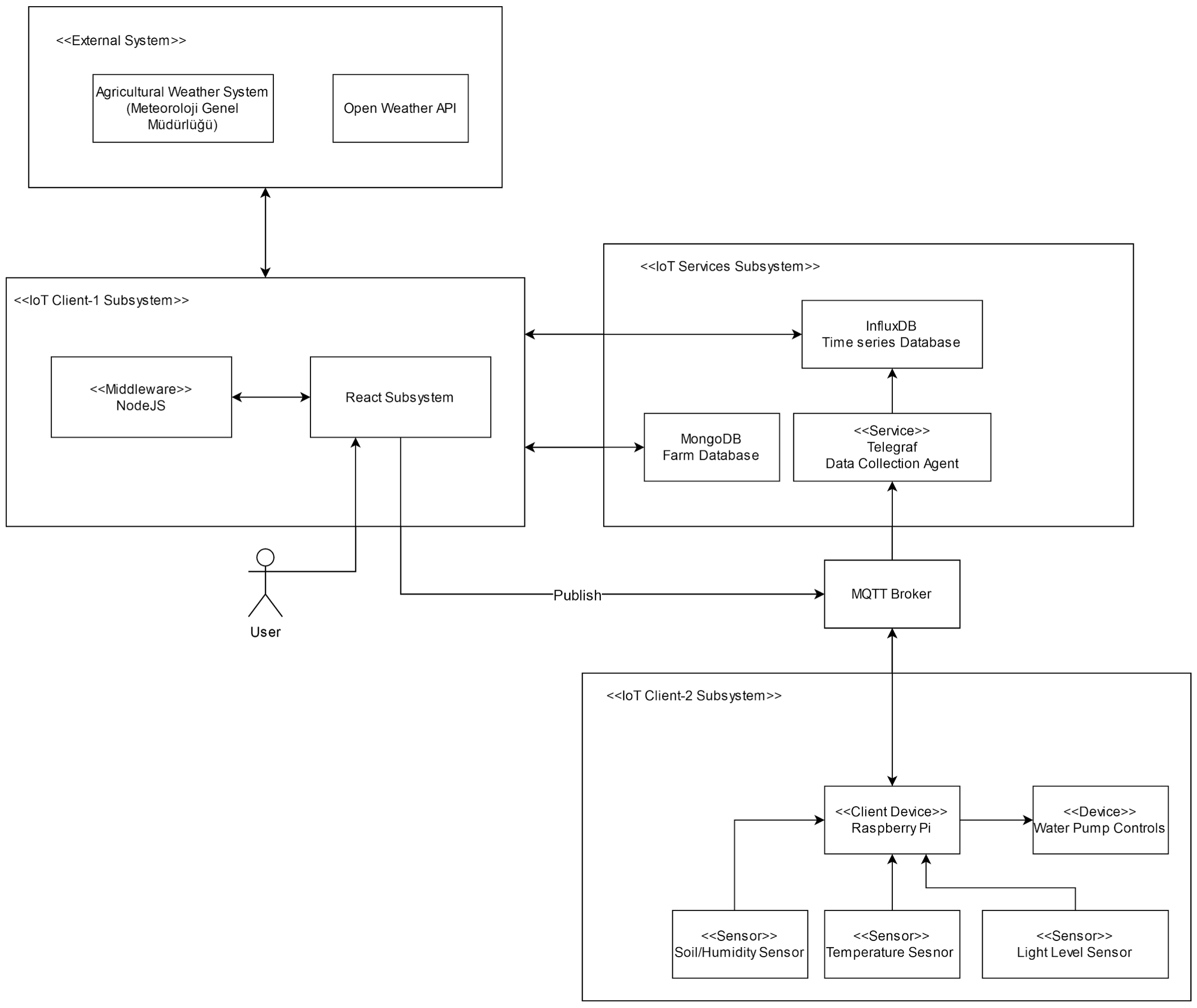
- InfluxDB: InfluxDB is a time-series database. It will be used as the database for our sensor data.
    - Integration Date: 1st Increment
- MongoDB: MongoDB is a NoSQL database. It will be used as the database for our user data.
    - Integration Date: 2nd Increment

1.4.5. APIs

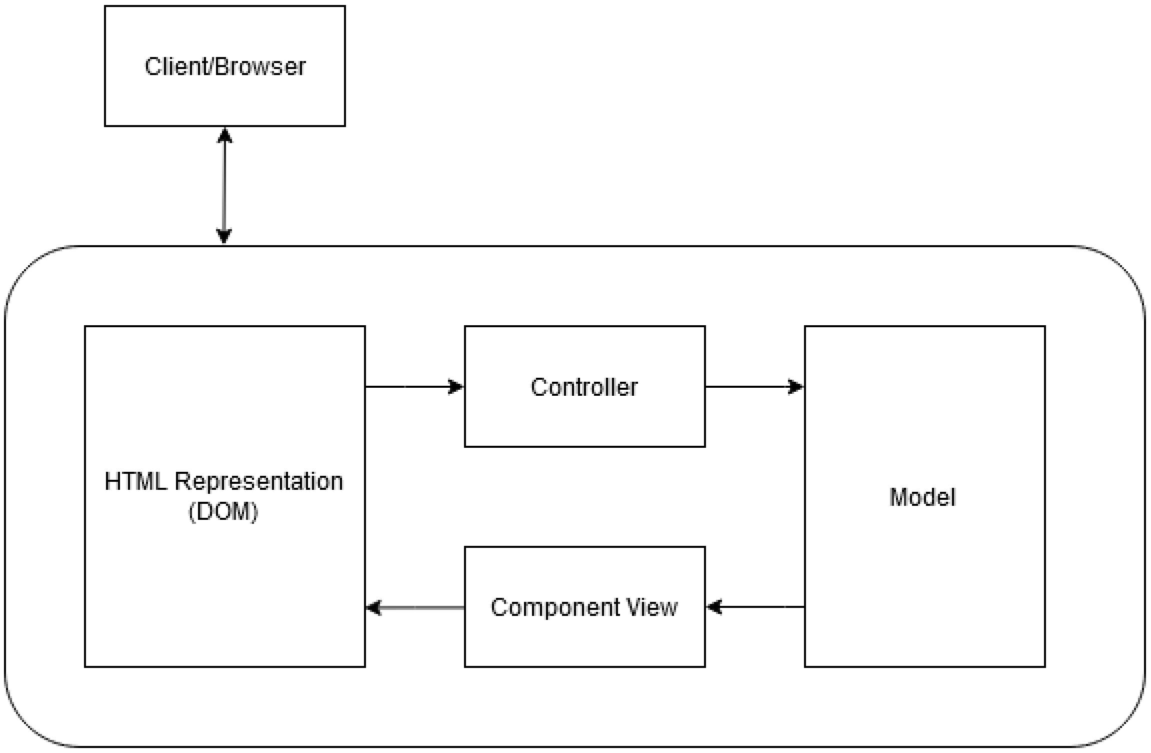
- InfluxDB API: It will be used to query and aggregate the sensor data stored in the InfluxDB.
    - Integration Date: 1st Increment
- OpenWeatherMap API: It will be used to query local weather and precipitation information.
    - Integration Date: 1st Increment

2. HIGH-LEVEL (ARCHITECTURE) DESIGN

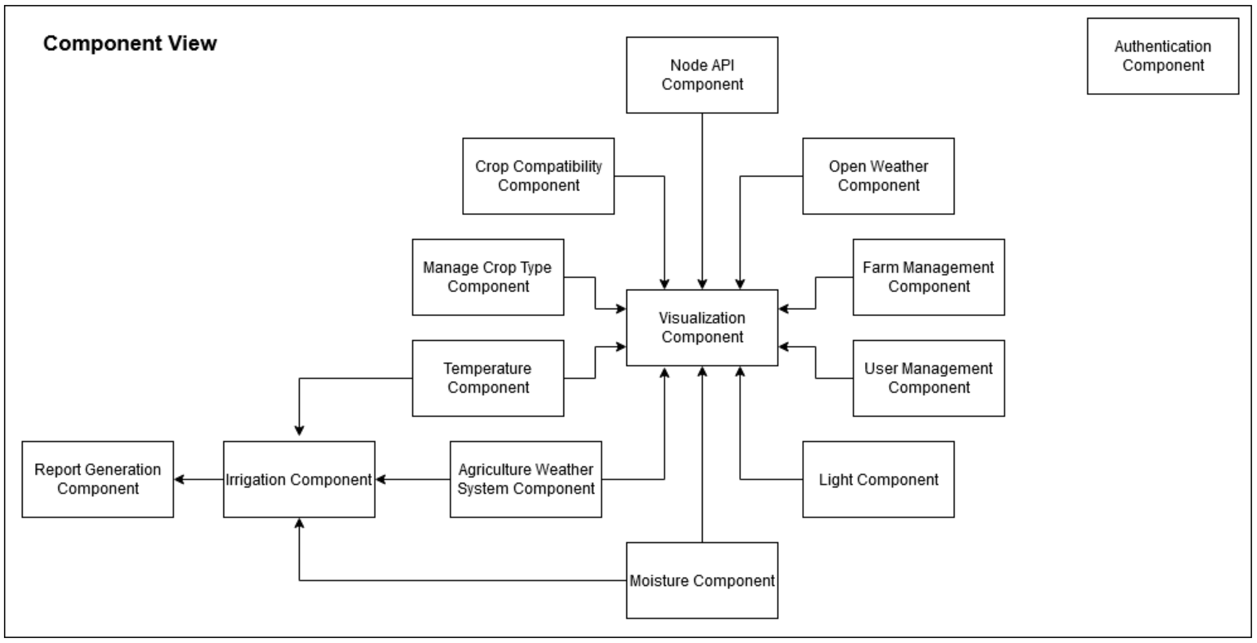
2.1. LOGICAL VIEW


<<External System>>
Agricultural Weather System (Meteoroloji Genel Müdürlüğü)
Open Weather API
<<IoT Client-1 Subsystem>>
<<Middleware>> NodeJS
React Subsystem
<<IoT Services Subsystem>>
InfluxDB Time series Database
MongoDB Farm Database
<<Service>> Telegraf Data Collection Agent
User
Publish
MQTT Broker
<<IoT Client-2 Subsystem>>
<<Client Device>> Raspberry Pi
<<Device>> Water Pump Controls
<<Sensor>> Soil/Humidity Sensor
<<Sensor>> Temperature Sesnor
<<Sensor>> Light Level Sensor

Software System Architecture Block Diagram



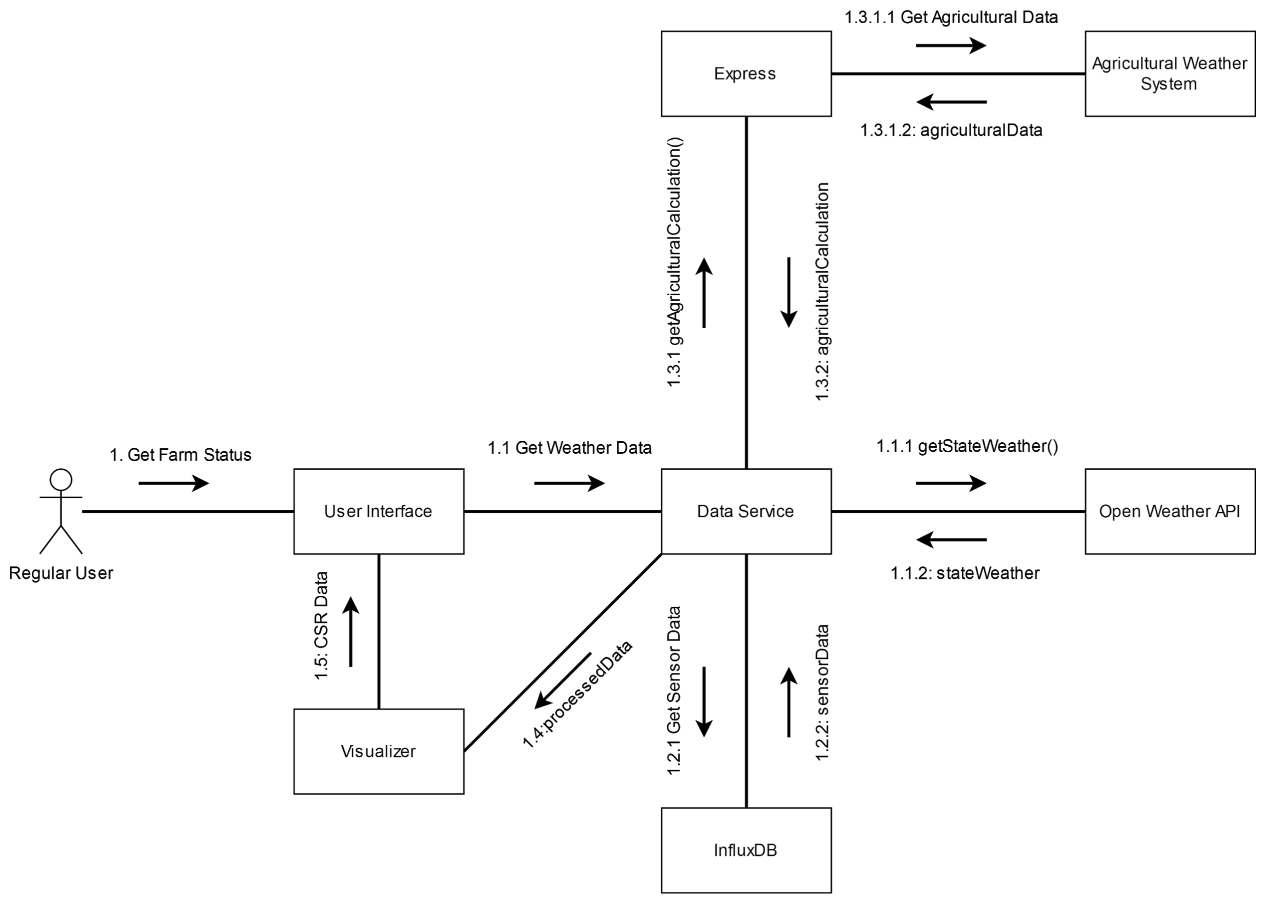
React System Architecture Diagram



React System Component View Diagram

2.2. PROCESS VIEW

Get Farm Status:



UML Communication Diagram

1: Get Farm Status: The regular user requests to get farm status using the user interface. The user simply clicks on the ‘Status’ button.
1.1: A request is made to get weather data to the Data Service.
1.1.1: A request to get state weather is made to the Open Weather API.
1.1.2: The Open Weather API returns the stateWeather object to the Data Service containing the weather information
1.2.1: A request to get sensor data is made to InfluxDB.
1.2.2: InfluxDB returns the sensorData object to the Data Service containing the sensor data.
1.3.1: A request to get agricultural calculation is made to Express.
1.3.1.1: Express requests to get agricultural data from the Agricultural Weather System.
1.3.1.2: Agricultural Weather System returns the agriculturalData object to Express containing the agricultural data.
1.3.2: Express returns the agriculturalCalculation object containing the agricultural calculations derived from the agricultural data.
1.4: The processdata object containing the stateWeather object containing open Weather API information, a sensorData object containing sensor data and an agriculturalCalculation object containing the agricultural data is sent to the visualizer.

1.5: Visualizer serves the client side rendering data to the user interface, showing the visualized output of the farm status.

2.3. PHYSICAL VIEW

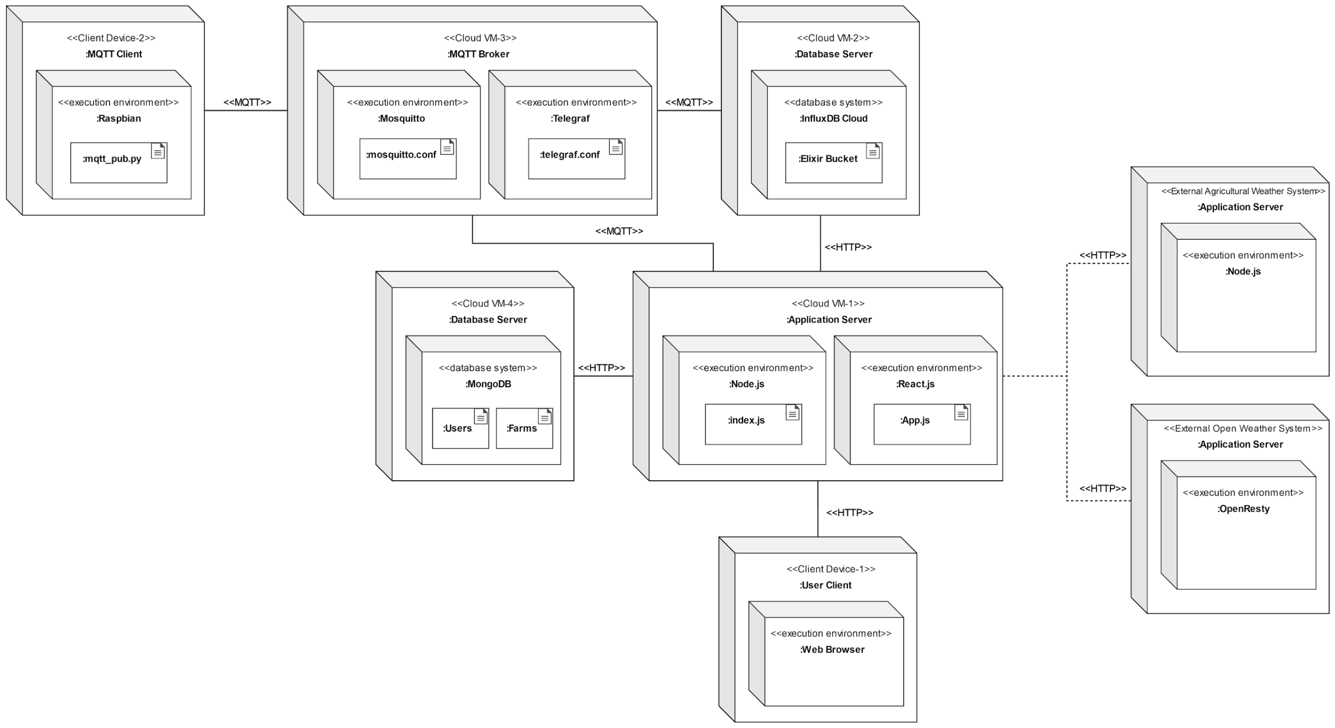


Figure x: UML Deployment diagram (Physical View)

2.4. DESIGN QUALITY

2.4.1. Performance

- The system shall display the visualized output of farm and crop information to the user in less than 3 seconds.
  - In order to provide low latency, virtual machine instances will be deployed to geographically closer locations.
  - For IoT communication, the MQTT (over SSL) protocol will be used, which can publish a message up to 20 times faster with about 50 times less traffic compared to an HTTP(S) POST message (Bartnitsky).
  - Gzip compression will be enabled for the web application server in order to provide smaller file sizes.
  - Caching will be enabled for the web application in order to fetch non-dynamic resources with improved performance.
  - Queries to be used with the APIs will be optimized and minimized in order to return only relevant data.
  - For validation and verification, Chrome and Firefox developer tools' performance utilities will be used to measure the response time and detect problematic components. Additionally, **sitespeed.io** tools will be used in order to conduct a thorough measurement of web application response time.

2.4.2. Security

- The system shall communicate over encrypted channels for both the web application and the MQTT devices.
    - For the web application, a Let's Encrypt TLS certificate will be used in order to use TLS with a CA-signed certificate for free, which will allow encrypted communication between the client and the server. Additionally, only port 443 will be allowed for incoming connections for the virtual machine hosting the web application.
    - For the InfluxDB service, an InfluxDB Cloud instance will be used. InfluxDB Cloud allows communication via HTTPS by default
    - For the MQTT communication, a self-signed certificate will be created via OpenSSL. Both the broker and the clients will be configured to use this self-signed certificate for encrypted communication.
    - For the MongoDB service, a Let's Encrypt TLS certificate will be used in order to provide encrypted communication between the MongoDB database server and the application server.
    - For validation and verification, incoming packets will be inspected with Wireshark to ensure that they are encrypted.
- All the sensitive data shall be encrypted both in rest and in transit.
    - For in transit encryption, see above explanations.
    - For in rest encryption, user passwords will be salted and hashed with bcrypt. Additionally, all the virtual machines will employ full-disk encryption against potential physical intrusions.
    - For validation and verification, the documents within the database will be inspected to ensure users passwords and salted and hashed. To make sure that virtual machines have full-disk encryption enabled, the 'disk encryption' state of the virtual machines will be inspected in the Azure Security Center. No verification is required for GCP VMs as full-disk encryption cannot be turned off for them.

2.4.3. Reliability

- The devices within the MQTT network shall have the fault tolerance mechanisms to restart both the devices and the services when they stop working.
    - The mosquito and telegraph services will use custom unit files with auto-restart capabilities in case of crash or failure.
    - The Python scripts for publishing sensor data and subscribing to control messages will be run as custom services with auto-restart capabilities in their unit files.
    - For validation and verification, MQTT services will be intentionally crashed, killed, and hung to ensure that the auto-restart capabilities are working fine.

2.4.4. Availability

- The system shall have an availability rate of 99.99%:
    - Azure's Virtual Machines service provides an SLA of 99.99% for virtual machines that have at least two instances deployed in two

availability zones in a region (Microsoft). Considering that Azure's availability zone pricing is based on bandwidth usage, this solution is affordable for us. Thus, we will deploy two instances in two availability zones for our Azure VMs, which will host our web application and NoSQL database.

- Google Cloud Platform's Compute Engine service provides an SLA of 99.99% for instances in multiple zones (Google). As we're running the cheapest VM available, f1.micro, this option is affordable for us. Thus, we will deploy our Compute Engine instance in two zones, which will host our MQTT broker and Telegraf agent.
- For validation and verification, uptime checks provided with Azure and GCP will be used.

3. LOW-LEVEL DESIGN

- React Dashboard Implementation
  - https://github.com/IoTFarming-CTISTeam3/reactWithNodeBack
  - https://github.com/IoTFarming-CTISTeam3/react_dash

- Hardware Programs, Sensors and Diagrams
  - https://github.com/IoTFarming-CTISTeam3/Project-Prototype

- The following implementations will be show in section 3.1
  - Temperature reading and MQTT Client publishing for Raspberry Pi.
  - LDR reading and MQTT Client publishing for Raspberry Pi.
  - Soil Humidity reading and MQTT Client publishing for Raspberry Pi.
  - OpenWeatherAPI Precipitation Calculation
  - InfluxDB Temperature Query
  - InfluxDB Soil Moisture Query
  - InfluxDB Light Query
  - Plotly Visualizations
  - Agricultural Weather System Frost Risk Calculation
  - Agricultural Weather System Heat Stress Calculation
  - Agricultural Weather System Pesticide Eligibility Calculation

3.1. Elixir Implementation

Temperature reading and MQTT Client publishing for Raspberry Pi.

```
import paho.mqtt.client as mqtt
import time
from w1thermsensor import W1ThermSensor #DS18B20 Library
import random

USERNAME="ank-polatli-01"
PASSWORD="<PASSWORD>"

sensor = W1ThermSensor() # Without specifying GPIO pins the library auto
detects the DS18B20

broker_address="35.231.170.180"
client = mqtt.Client("ANK-POLATLI-01")
client.username_pw_set(username=USERNAME, password=PASSWORD)
client.connect(broker_address)
client.loop_start()
try:
    while True:
        temp = sensor.get_temperature()
        print("Temp:"+"{:.2f}".format(temp)+" C")
```

```python
        client.publish("/farm/ank-polatli-01/temperature/", temp)
        time.sleep(1800) # Publish the temperature message every 30 minutes
except KeyboardInterrupt: #Ctrl-C
    print("\nExitting Program")
    pass
```

LDR reading and MQTT Client publishing for Raspberry Pi.

```python
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time
#MQTT Part
USERNAME="ank-polatli-01"
PASSWORD="<PASSWORD>"
broker_address="35.231.170.180"
client = mqtt.Client("ANK-POLATLI-01")
client.username_pw_set(username=USERNAME, password=PASSWORD)
client.connect(broker_address)
client.loop_start()
# Raspberry Pi GPIO Pin Number = 5
GPIO.setmode(GPIO.BCM) #pin number of LDR
pin_to_circuit = 5
def rc_time (pin_to_circuit):
    count = 0   #LDR value counter
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    time.sleep(0.1)   #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN) #Count until the pin goes high
    while (GPIO.input(pin_to_circuit) == GPIO.LOW):
        count += 1
    return count   #Catch on change - Dark - Light

try: # Main loop for sending data
    while True:
        #print(rc_time(pin_to_circuit))
        value=rc_time(pin_to_circuit)
        if (value <= 25000 ):
                client.publish("/farm/ank-polatli-01/lightlevel/", 1)
        if (value > 25000):
                client.publish("/farm/ank-polatli-01/lightlevel/", 0)

        time.sleep(1800) # Publish the  message every 30 minutes
        #time.sleep(10) # Publish the  message every 10 seconds
except KeyboardInterrupt: #Ctrl-C
```

```
    print("\nExitting Program")
    pass
```

Soil Humidity reading and MQTT Client publishing for Raspberry Pi.

```python
import paho.mqtt.client as mqtt
import time
import time
import random

USERNAME="ank-polatli-01"
PASSWORD="<PASSWORD>"

# Raspberry Pi GPIO Pin Number

broker_address="35.231.170.180"
client = mqtt.Client("ANK-POLATLI-01")
client.username_pw_set(username=USERNAME, password=PASSWORD)
client.connect(broker_address)
client.loop_start()

try:
    while True:
        #client.publish("/farm/ank-polatli-01/humidity/", hum_sensor)
        time.sleep(1800) # Publish the  message every 30 minutes
except KeyboardInterrupt: #Ctrl-C
    print("\nExitting Program")
    pass
```

OpenWeatherAPI precipitation calculation

```javascript
getStateWeather = () => {
    let WetTemp = [];
    let one =
"https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=36.8486072
&lon=28.25418&dt=" + this.state.date[0] +
"&appid=79cfd14676df6fdf7ca1f503cd406ce1&units=metric";
    let two =
"https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=36.8486072
```

```
&lon=28.25418&dt=" + this.state.date[1] +
"&appid=79cfd14676df6fdf7ca1f503cd406ce1&units=metric";
    let three =
"https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=36.8486072
&lon=28.25418&dt=" + this.state.date[2] +
"&appid=79cfd14676df6fdf7ca1f503cd406ce1&units=metric";
    let four =
"https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=36.8486072
&lon=28.25418&dt=" + this.state.date[3] +
"&appid=79cfd14676df6fdf7ca1f503cd406ce1&units=metric";
    let five =
"https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=36.8486072
&lon=28.25418&dt=" + this.state.date[4] +
"&appid=79cfd14676df6fdf7ca1f503cd406ce1&units=metric";
    axios
      .all([
        axios.get(one),
        axios.get(two),
        axios.get(three),
        axios.get(four),
        axios.get(five)
      ])
      .then(axios.spread((req0, req1, req2, req3, req4) => {
        this.state.date.forEach((item, i) => {
          // Number of hours in result
          let hours = eval(`req${i}.data.hourly`).length
          let dailyrain = 0;
          for (var j = 0; j < hours; j++) {
            if (typeof eval(`req${i}.data.hourly[j].rain`) == 'object') {
              dailyrain = dailyrain +
eval(`req${i}.data.hourly[j].rain`)[Object.keys(eval(`req${i}.data.hourly[j
].rain`))[0]]
            }
          }
          WetTemp.push(dailyrain);
          this.setState({
            raiy: WetTemp,

            day: this.timeFrom(5),
            weather: [...this.state.weather, {
              //humidity : eval(`req${i}.data.current.humidity`)
              rain: dailyrain,
            }]
          });
        })
      }))
  }
```

InfluxDB Temperature Query

```
getStateTemp = () => {
    var axios = require('axios');
    var data = 'from(bucket: "erdoganys\'s Bucket")\r\n  |> range(start:
-1w)\r\n  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")\r\n
|> filter(fn: (r) => r["_field"] == "value")\r\n  |> filter(fn: (r) =>
r["host"] == "broker")\r\n  |> filter(fn: (r) => r["topic"] ==
"/farm/ank-polatli-01/temperature/")\r\n  |> aggregateWindow(every: 24h,
fn: mean, createEmpty: false)\r\n  |> drop(columns: ["topic", "host",
"_measurement"])';
    let temperatureArr = [];
    var config = {
      method: 'post',
      url:
'https://us-central1-1.gcp.cloud2.influxdata.com/api/v2/query?org=2b0234c7b
fa505d4',
      headers: {
        'Authorization': 'Token
GA6zK8qSTBEqzAjqaKIX_5VrKjHkLDmDWCNwIqs-_al3ZhUuYVFxqn95WOrw2rYxT4QTTpegYIL
eGUdz7MXH9g==',
        'Content-Type': 'application/vnd.flux',
        'Accept': 'application/csv'
      },
      data: data
    };
    axios(config)
      .then((response) => {
        let templen = readString(response.data);
        console.log("check", templen);
        for (let i = 1; i < templen.data.length; i++) {
          if (templen.data[i][6] !== undefined) {
            temperatureArr.push(templen.data[i][6]);
          }
        }
        temperatureArr.shift();
        console.log("temp", temperatureArr);
        this.setState({
          temp: temperatureArr,
          day: this.timeFrom(temperatureArr.length),
        }, () => {
          console.log(this.timeFrom(temperatureArr.length));
        });
      })
      .catch(function (error) {
        console.log(error);
      });
  }
```

InfluxDB Soil Moisture Query

```
 getStateMoisture = () => {
    let timesfour = [];
    let soil = [];
    var axios = require('axios');
    var data = 'from(bucket: "erdoganys\'s Bucket")\r\n  |> range(start:
-24h)\r\n  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")\r\n
|> filter(fn: (r) => r["_field"] == "value")\r\n  |> filter(fn: (r) =>
r["host"] == "broker")\r\n  |> filter(fn: (r) => r["topic"] ==
"/farm/ank-polatli-01/humidity/")\r\n  |> aggregateWindow(every: 4h, fn:
mean, createEmpty: false)\r\n  |> drop(columns: ["topic", "host",
"_measurement"])';
    var config = {
      method: 'post',
      url:
'https://us-central1-1.gcp.cloud2.influxdata.com/api/v2/query?org=2b0234c7b
fa505d4',
      headers: {
        'Authorization': 'Token
GA6zK8qSTBEqzAjqaKIX_5VrKjHkLDmDWCNwIqs-_al3ZhUuYVFxqn95WOrw2rYxT4QTTpegYIL
eGUdz7MXH9g==',
        'Content-Type': 'application/vnd.flux',
        'Accept': 'application/csv'
      },
      data: data
    };
    axios(config)
      .then((response) => {
        var soilmoisture = readString(response.data);
        for (let i = 1; i < soilmoisture.data.length; i++) {
          var soilmoistures = parseInt(soilmoisture.data[i][6]);
          var timefo = soilmoisture.data[i][5];
          if (!Number.isNaN(soilmoistures)) {
            soil.push(soilmoistures);
            timesfour.push(timefo);
          }
        }
        this.setState({
          soiltemp: soil,
          date: timesfour,
        });
      })
      .catch(function (error) {
        console.log(error);
      });
  }
```

InfluxDB Light Query

```
getStateLight = () => {
    var data = 'from(bucket: "erdoganys\'s Bucket")\r\n  |> range(start:
-24h)\r\n  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")\r\n
|> filter(fn: (r) => r["_field"] == "value")\r\n  |> filter(fn: (r) =>
r["host"] == "broker")\r\n  |> filter(fn: (r) => r["topic"] ==
"/farm/ank-polatli-01/lightlevel/")\r\n  |> last()\r\n  |> drop(columns:
["topic", "host", "_measurement"])';
    let lightdates = [];
    var config = {
      method: 'post',
      url:
'https://us-central1-1.gcp.cloud2.influxdata.com/api/v2/query?org=2b0234c7b
fa505d4',
      headers: {
        'Authorization': 'Token
GA6zK8qSTBEqzAjqaKIX_5VrKjHkLDmDWCNwIqs-_al3ZhUuYVFxqn95WOrw2rYxT4QTTpegYIL
eGUdz7MXH9g==',
        'Content-Type': 'application/vnd.flux',
        'Accept': 'application/csv'
      },
      data: data
    };
    axios(config)
      .then((response) => {
        console.log("csv", response.data);
        var light = readString(response.data, config);
        lightdates = parseInt(light.data[1][6]);

        this.setState({
          datalight: lightdates,
        });
      })
      .catch(function (error) {
        console.log(error);
      });
  }
```

Plotly visualization example for Soil Moisture

```
render() {
  return (
    <div>
      <div >
        <Plot
          data={[
            {
              x: this.state.date,
              y: this.state.soiltemp,
              type: 'bar',
              mode: 'lines,markers',
              marker: { color: 'cyan' },
            },
            // {type: 'bar', x: ["Monday", "Thursday"], y: [1, 2, 3]},
          ]}
          layout={{
            width: 540,
            height: 330,
            title: '<b>Daily Soil Moisture<b>',
            /*plot_bgcolor: "rgba(0,0,0,0)",
            paper_bgcolor: "rgba(0,0,0,0)" ,*/
            margin: {
              l: 50,
              r: 50,
              b: 50,
              t: 100,
              pad: 4
            }
          }}
        />
      </div>
      {/*
          {this.state.soiltemp.map(soiltemp => {
               return <h1>{soiltemp}</h1>;
            })}
        */}
      </div>
    );
  }
```

Calculate heat stress based on Agricultural Weather System data

```
async function riskHesap(data) {
    let arr = [];
    let saatlikTahmin = data;
    let status;
    console.log("saatlik tahmin", saatlikTahmin[0].tahmin.length)
    for (var i = 0; i < saatlikTahmin[0].tahmin.length; i++) {

        var thi = (1.8 * saatlikTahmin[0].tahmin[i].sicaklik + 32) - [(0.55
- 0.0055 * saatlikTahmin[0].tahmin[i].nem) * (1.8 *
saatlikTahmin[0].tahmin[i].sicaklik - 26.8)];
        if (i == saatlikTahmin[0].tahmin.length - 1) break;

        if (thi < 66) {
            status = "No Stres";
        }
        else if (thi >= 66 && thi <= 71) {
            status = "Middle Stres";
        }
        else if (thi >= 72 && thi <= 79) {
            status = "Stres";
        }
        else if (thi >= 80 && thi <= 89) {
            status = "Intense Stres";
        } else {
            status = "Strong Stres";
        }
        arr.push(status);
        var myJsonString = JSON.stringify(arr);
    }
    console.log('sd', arr);
    console.log(myJsonString);
    return (myJsonString);
}
```

Calculate frost risk and pesticidation suitability based on Agricultural Weather System data

```
    //Zirai tahmin İşlemleri
    for (var i = 0; i < 5; i++) {
        if (sicMin[i] <= 0 && sicMin[i] >= -2.2) {
            donRisk.push("Remarkable");
        }
        else if (sicMin[i] <= -2.2 && sicMin[i] >= -4.4) {
            donRisk.push("Middle");
        }
        else if (sicMin[i] <= -4.4) {
            donRisk.push("Strong");
        } else {

            donRisk.push("No Risk");
        }
        //İlaçlama Kontrolü
        ilac[i] = "";
        var uygun = 0;
        if (hds[i].indexOf('Y') == -1 && hds[i] != 'K') //Tahminler -
Günlük
            uygun++;
        if (sicMin[i] < 25) //Tahminler - Günlük
            uygun++;
        if (sicMax[i] > 10) //Tahminler - Günlük
            uygun++;
        if (rzg[i] >= 0 && rzg[i] <= 19) //Tahminler - Günlük
            uygun++;
        if (nem[i] >= 30 && nem[i] <= 85) //Tahminler - Günlük
            uygun++;

        if (uygun == 5) ilac[i] = "Pesticides";

        if (uygun != 5) {
            ilac[i] = "No Pesticides";
        }
    }
```

Bibliography

Bartnitsky, Jan. “HTTP vs MQTT performance tests.” *flespi — accelerated telematics & IoT development*, 23 January 2018, https://flespi.com/blog/http-vs-mqtt-performance-tests. Accessed 8 December 2020.

Google. “Compute Engine Service Level Agreement (SLA).” *Google Cloud Platform*, 7 April 2020, https://cloud.google.com/compute/sla. Accessed 8 December 2020.

Microsoft. “SLA for Virtual Machines.” *Microsoft Azure*, July 2020, https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_9/. Accessed 8 December 2020.