IDG2001 - Cloud Technologies

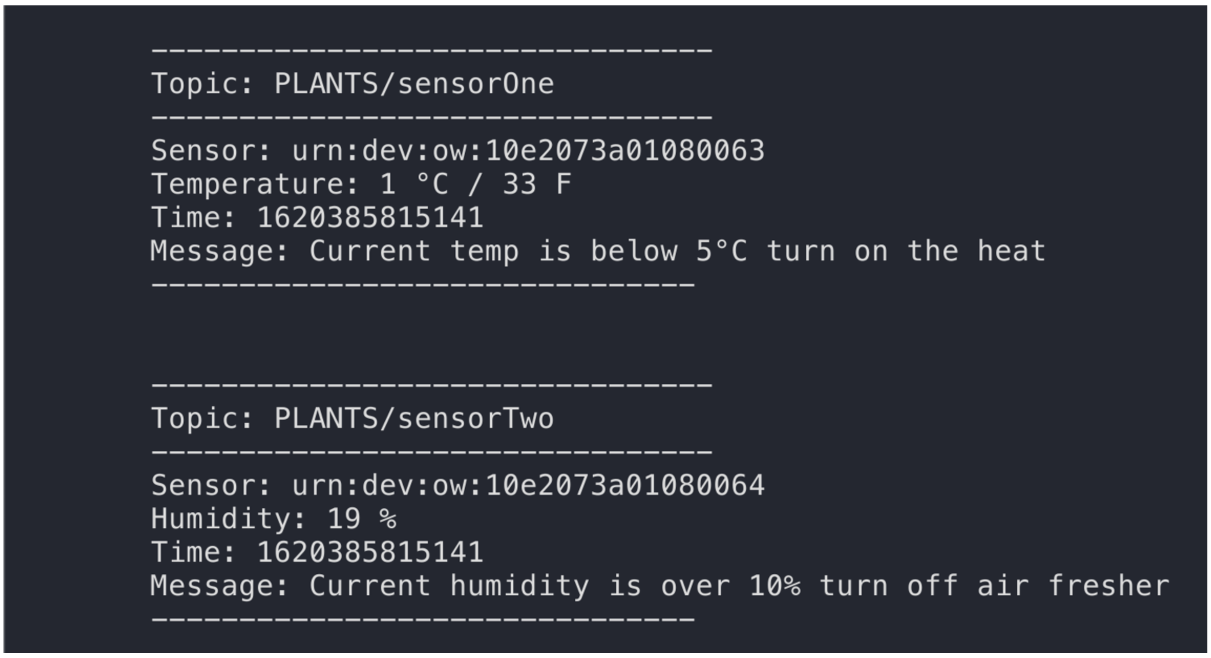
Possible marks to get

• IoT scenario– 5 points.
• Creativity of an IoT scenario and MQTT protocol – 4 points.
• Connection between Broker and publisher, Connection between publisher and subscriber (more than one subscribers) – 5 points.
• Database connection and table structure– 2 points.
• Insert/select data to Database – 3 points.
• Implementation of SenML/JSON and SenML/XML as payload - 6 points.
• Final presentation of project(10 minutes) – 5 points.
• Additional point: Implementation of SenML/EXI as payload - 5 points

IoT scenario

Since the covid-19 pandemic we have started to grow plants inside to prevent loneliness. The plants make us happier and help us do something else than schoolwork. Our IoT scenario is meant to make plant caring easier and sufficient. We have enough information on the internet on how to take care of plants, how much water does a plant need, what temperature does a plant need and how the humidity should be in the room. But with overflood of schoolwork, we don't have time to check plants or we even forget when the last time was. This solution will prevent this from ever happening again.

The subscriber sends a message if the temperature or humidity levels are bad, good or high. and will inform the user if the plant is not happy.



```
--------------------------------
Topic: PLANTS/sensorOne
--------------------------------
Sensor: urn:dev:ow:10e2073a01080063
Temperature: 1 °C / 33 F
Time: 1620385815141
Message: Current temp is below 5°C turn on the heat
-------------------------------


--------------------------------
Topic: PLANTS/sensorTwo
--------------------------------
Sensor: urn:dev:ow:10e2073a01080064
Humidity: 19 %
Time: 1620385815141
Message: Current humidity is over 10% turn off air fresher
-------------------------------
```

Scenario 1:
A notification or email is sent to the user with the message if the temperature or humidity of the plant(s) is not optimal for them. You can choose how frequently the user gets a notification but default is 4 times a day.

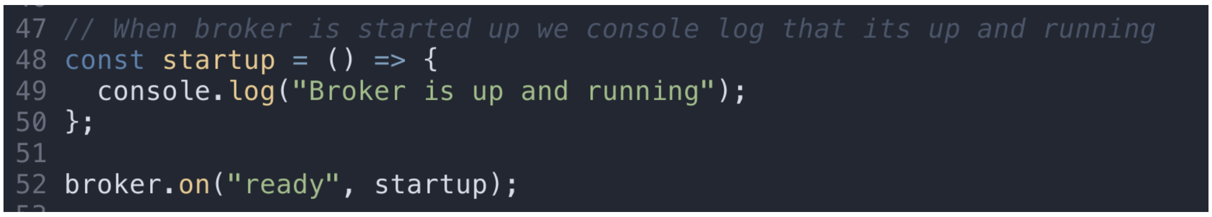
Scenario 2:
The user can set up a watering schedule with a calendar and be notified on email when the different waterings should be done.

Connection between Broker and publisher

We decided to use a config.js file containing client information and are importing this on pub.js and sub.js

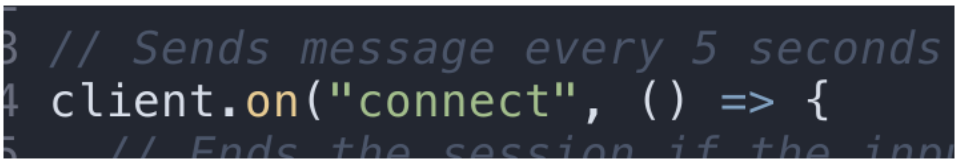
Mosca is used as a broker in this project and is configured to use port 1883.
For client we are using mqtt with configuration mqtt.connect("mqtt://localhost:1883")

index.js

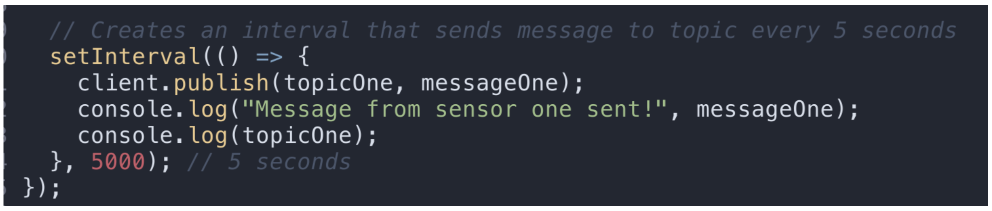


```javascript
// When broker is started up we console log that its up and running
const startup = () => {
  console.log("Broker is up and running");
};

broker.on("ready", startup);
```

pub.js

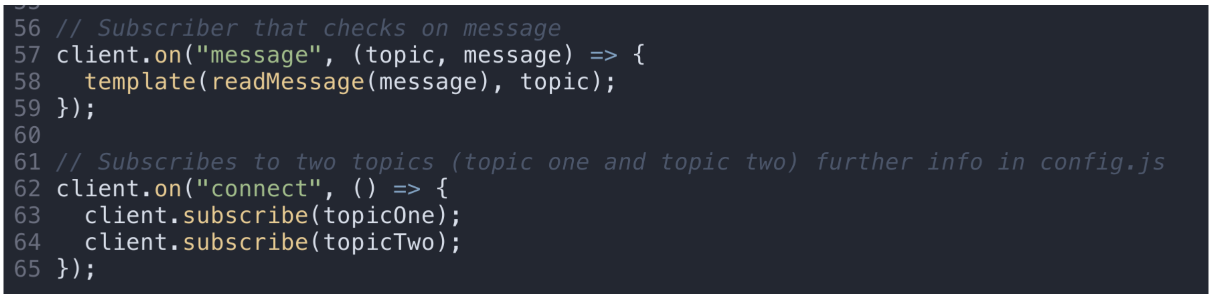


```javascript
// Sends message every 5 seconds
client.on("connect", () => {
   // Ends the session if the inn
```



```javascript
  // Creates an interval that sends message to topic every 5 seconds
  setInterval(() => {
    client.publish(topicOne, messageOne);
    console.log("Message from sensor one sent!", messageOne);
    console.log(topicOne);
  }, 5000); // 5 seconds
});
```

sub.js

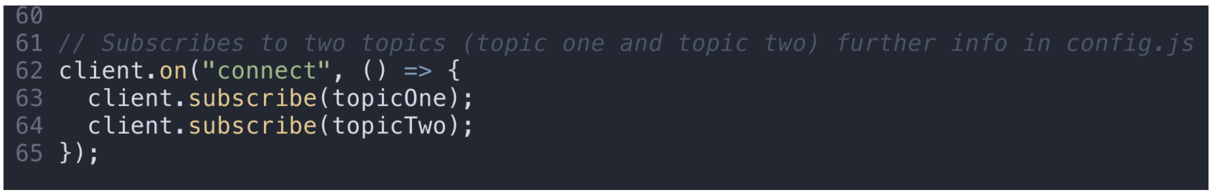


```
56 // Subscriber that checks on message
57 client.on("message", (topic, message) => {
58   template(readMessage(message), topic);
59 });
60
61 // Subscribes to two topics (topic one and topic two) further info in config.js
62 client.on("connect", () => {
63   client.subscribe(topicOne);
64   client.subscribe(topicTwo);
65 });
```

Connection between publisher and subscriber (more than one subscribers)

We have a line in config.js where 1 or 2 publishers could be invoked. This is for demo purposes and shows that we have multiple publishers and subscribers for different topics.

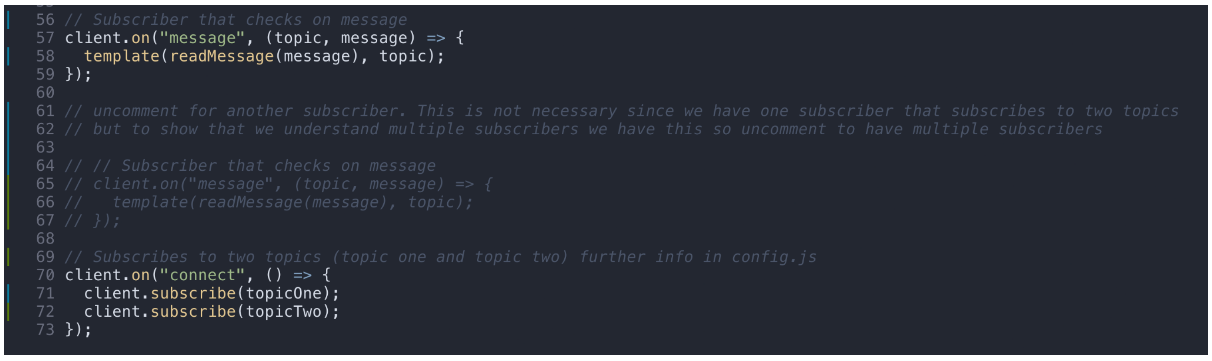
Inside sub.js we have



```
60
61 // Subscribes to two topics (topic one and topic two) further info in config.js
62 client.on("connect", () => {
63   client.subscribe(topicOne);
64   client.subscribe(topicTwo);
65 });
```

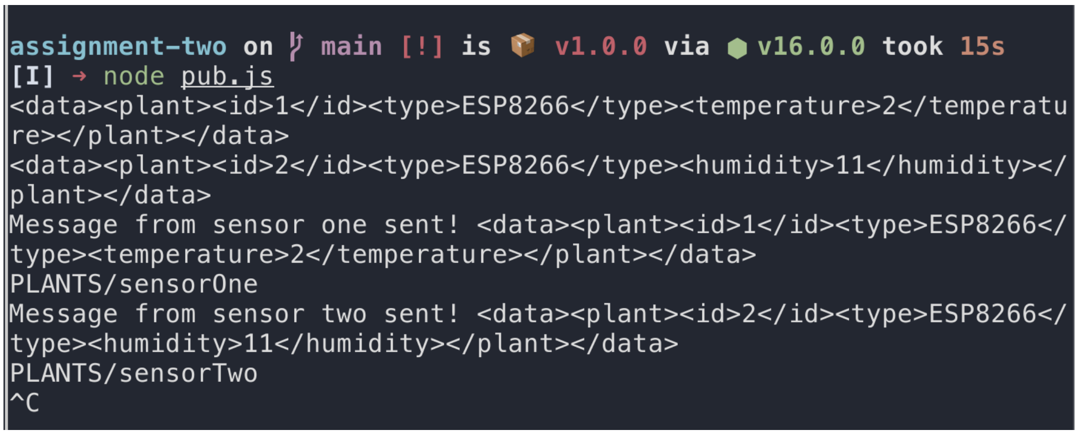
which is subscribing to two topics at the same time.

Uncomment from line 65 / 67 to get one more subscriber. As stated inside the code we have one subscriber with two topics but to show that we understand the task we have another subscriber but it's uncommented right now as it's not needed. But to show that we understand we have it inside the code.



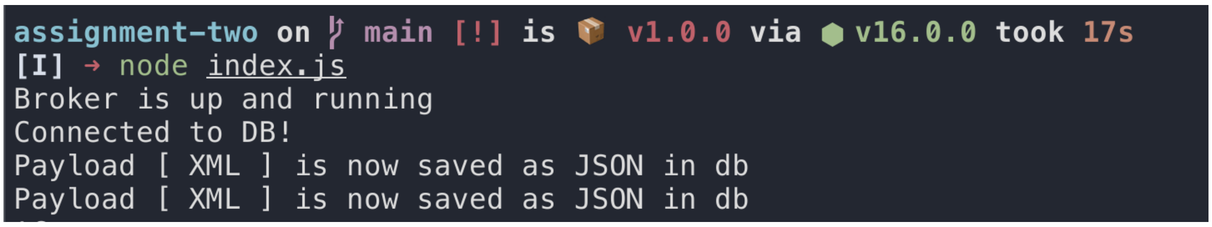
```
56 // Subscriber that checks on message
57 client.on("message", (topic, message) => {
58   template(readMessage(message), topic);
59 });
60
61 // uncomment for another subscriber. This is not necessary since we have one subscriber that subscribes to two topics
62 // but to show that we understand multiple subscribers we have this so uncomment to have multiple subscribers
63
64 // // Subscriber that checks on message
65 // client.on("message", (topic, message) => {
66 //   template(readMessage(message), topic);
67 // });
68
69 // Subscribes to two topics (topic one and topic two) further info in config.js
70 client.on("connect", () => {
71   client.subscribe(topicOne);
72   client.subscribe(topicTwo);
73 });
```

Example message sent with xml as payload
pub.js



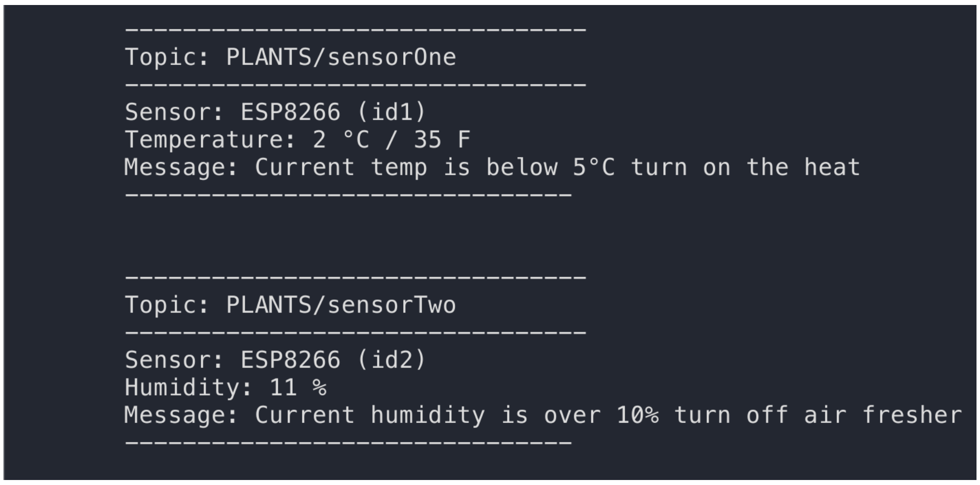
```
assignment-two on  main [!] is 📦 v1.0.0 via ⬢ v16.0.0 took 15s
[I] ➜ node pub.js
<data><plant><id>1</id><type>ESP8266</type><temperature>2</temperatu
re></plant></data>
<data><plant><id>2</id><type>ESP8266</type><humidity>11</humidity></
plant></data>
Message from sensor one sent! <data><plant><id>1</id><type>ESP8266</
type><temperature>2</temperature></plant></data>
PLANTS/sensorOne
Message from sensor two sent! <data><plant><id>2</id><type>ESP8266</
type><humidity>11</humidity></plant></data>
PLANTS/sensorTwo
^C
```

index.js



```
assignment-two on  main [!] is 📦 v1.0.0 via ⬢ v16.0.0 took 17s
[I] ➜ node index.js
Broker is up and running
Connected to DB!
Payload [ XML ] is now saved as JSON in db
Payload [ XML ] is now saved as JSON in db
```

sub.js



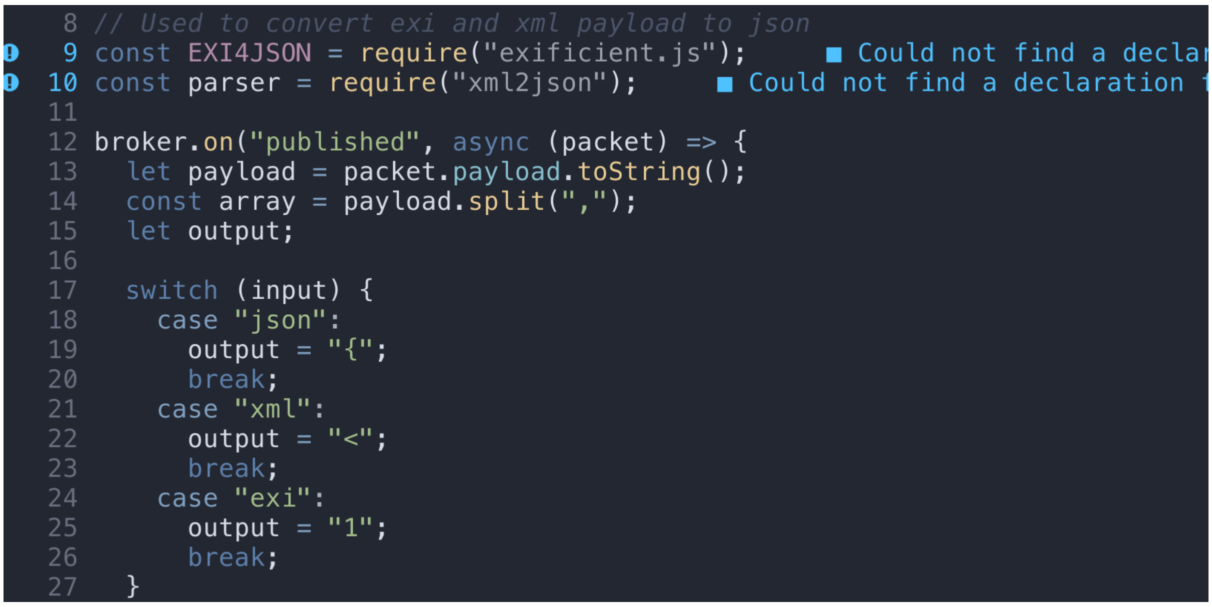
```
        --------------------------------
        Topic: PLANTS/sensorOne
        --------------------------------
        Sensor: ESP8266 (id1)
        Temperature: 2 °C / 35 F
        Message: Current temp is below 5°C turn on the heat
        -------------------------------



        --------------------------------
        Topic: PLANTS/sensorTwo
        --------------------------------
        Sensor: ESP8266 (id2)
        Humidity: 11 %
        Message: Current humidity is over 10% turn off air fresher
        -------------------------------
```

Database connection and table structure

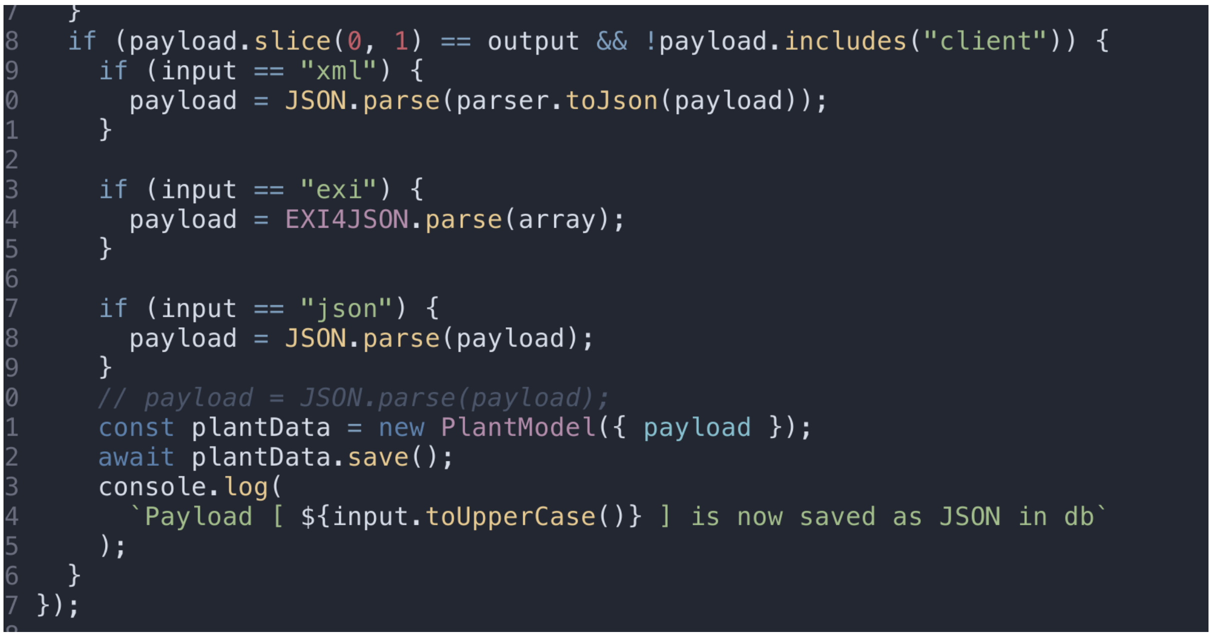
We decided to use a local database as this project is not going to be deployed and we have configured it to save in the database plants and are using plants as collection name.

We are storing the payload as a string in the database and could be easily extracted and parsed to display the object. Look at the images for saved database objects.

Since we can send three types of data in publisher [json, xml and exi] we need to convert it to a json object to store it in the database to avoid storing different types of data. To do this we have a switch statement that checks if the first character of the message we get as payload from the publisher. See code below.
If the payload is not from input == json we are converting it to json and stringify before saving to db

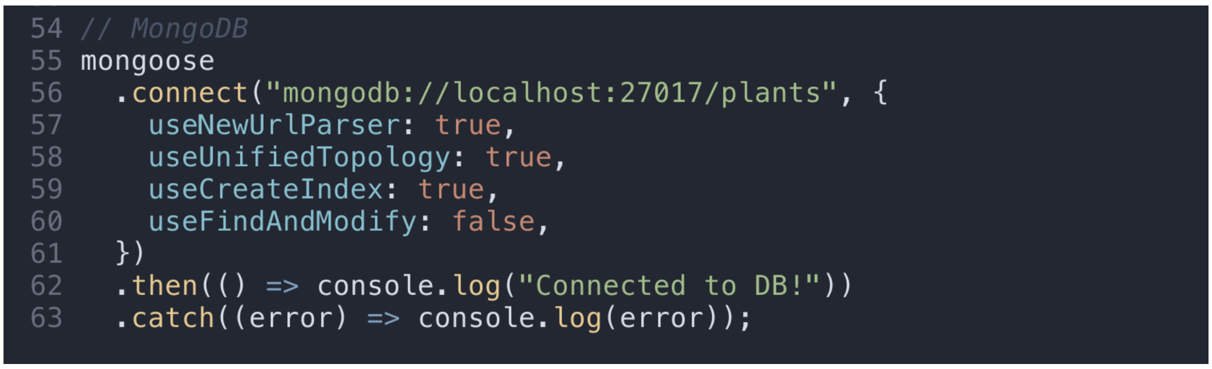


```
// Used to convert exi and xml payload to json
const EXI4JSON = require("exificient.js");     ■ Could not find a declar
const parser = require("xml2json");     ■ Could not find a declaration f

broker.on("published", async (packet) => {
  let payload = packet.payload.toString();
  const array = payload.split(",");
  let output;

  switch (input) {
    case "json":
      output = "{";
      break;
    case "xml":
      output = "<";
      break;
    case "exi":
      output = "1";
      break;
  }
```



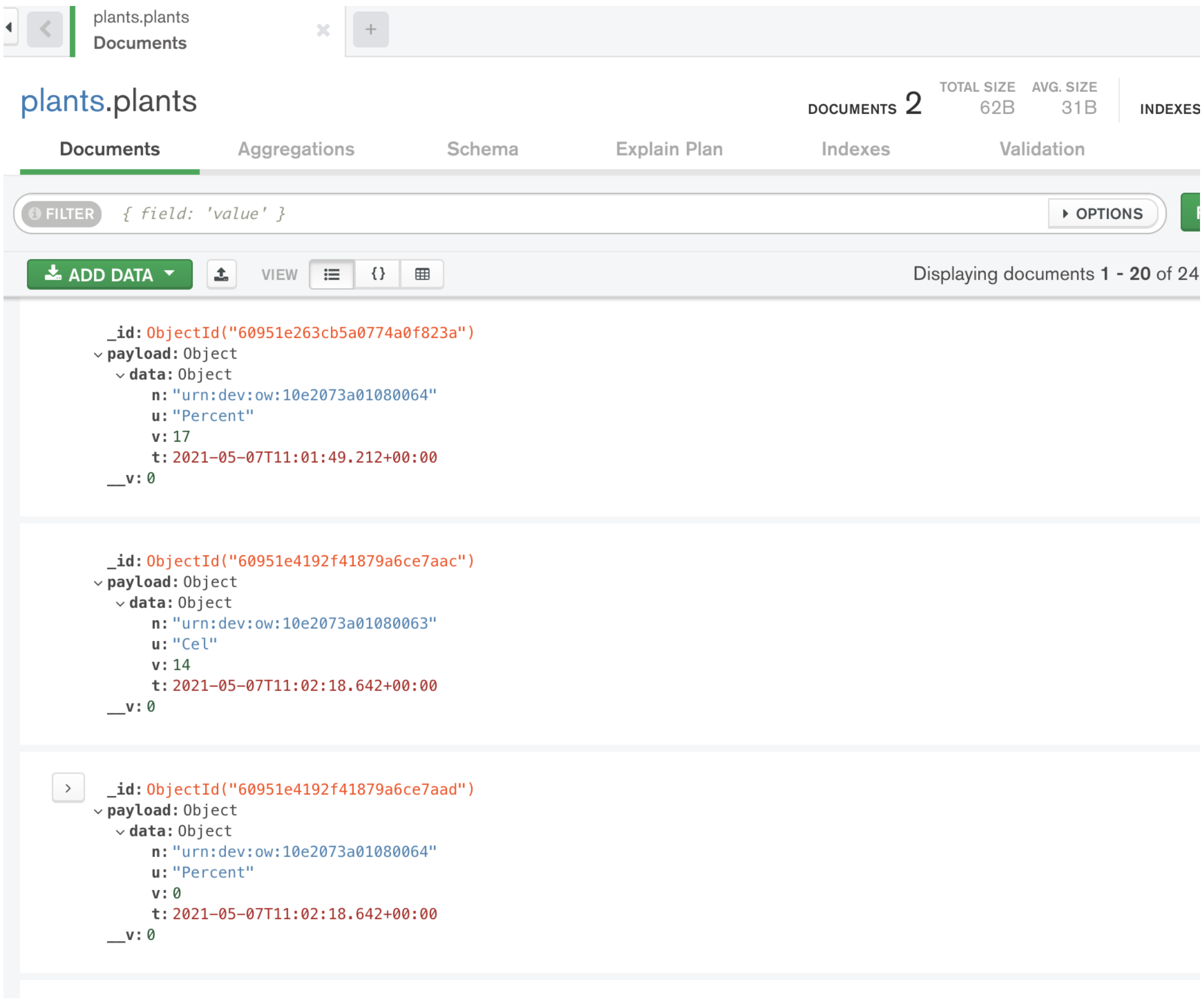
```
  }
  if (payload.slice(0, 1) == output && !payload.includes("client")) {
    if (input == "xml") {
      payload = JSON.parse(parser.toJson(payload));
    }

    if (input == "exi") {
      payload = EXI4JSON.parse(array);
    }

    if (input == "json") {
      payload = JSON.parse(payload);
    }
    // payload = JSON.parse(payload);
    const plantData = new PlantModel({ payload });
    await plantData.save();
    console.log(
      `Payload [ ${input.toUpperCase()} ] is now saved as JSON in db`
    );
  }
});
```

index.js



```
// MongoDB
mongoose
  .connect("mongodb://localhost:27017/plants", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    useFindAndModify: false,
  })
  .then(() => console.log("Connected to DB!"))
  .catch((error) => console.log(error));
```

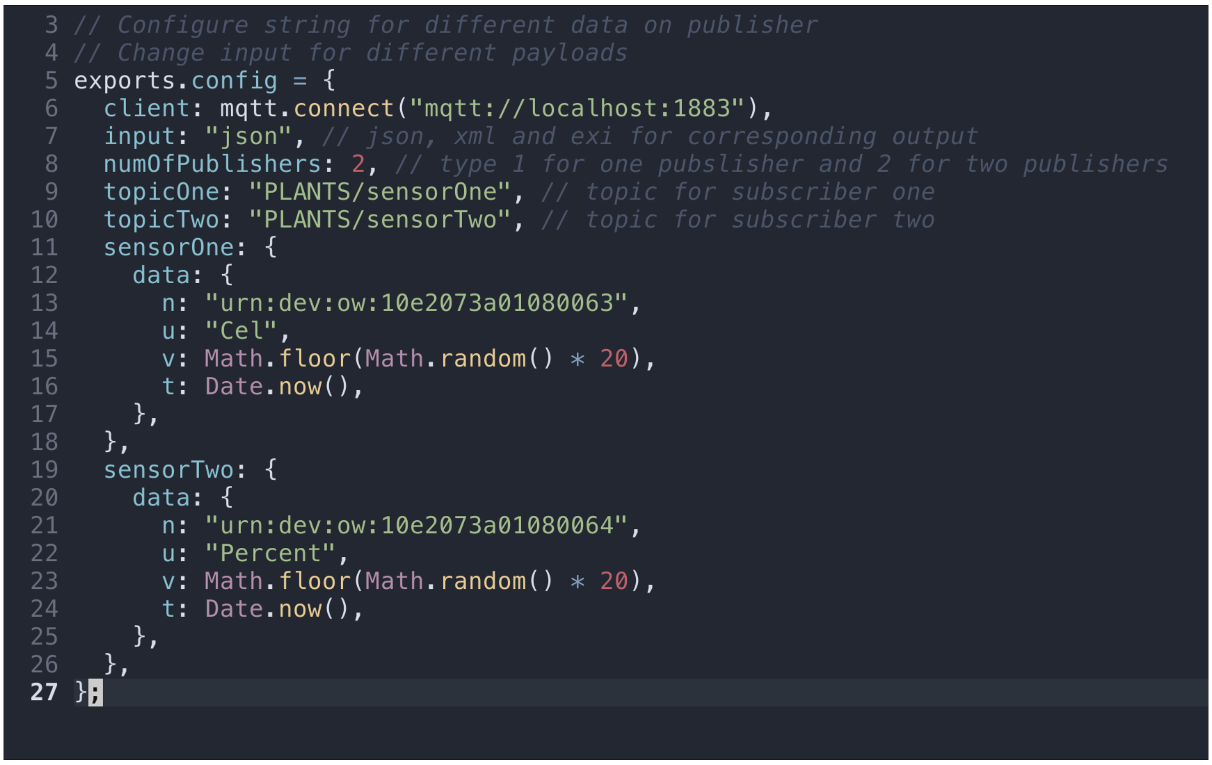
database



```
_id: ObjectId("60951e263cb5a0774a0f823a")
payload: Object
  data: Object
    n: "urn:dev:ow:10e2073a01080064"
    u: "Percent"
    v: 17
    t: 2021-05-07T11:01:49.212+00:00
__v: 0

_id: ObjectId("60951e4192f41879a6ce7aac")
payload: Object
  data: Object
    n: "urn:dev:ow:10e2073a01080063"
    u: "Cel"
    v: 14
    t: 2021-05-07T11:02:18.642+00:00
__v: 0

_id: ObjectId("60951e4192f41879a6ce7aad")
payload: Object
  data: Object
    n: "urn:dev:ow:10e2073a01080064"
    u: "Percent"
    v: 0
    t: 2021-05-07T11:02:18.642+00:00
__v: 0
```

Implementation of SenML/JSON and SenML/XML as payload and Additional point: Implementation of SenML/EXI as payload

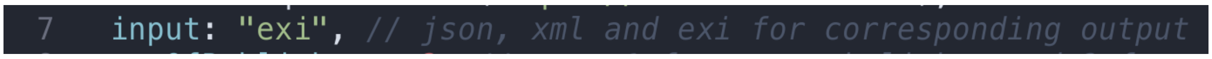
To emulate sensor data we have an object inside config.js that is being sent with random values from the publisher. Here we have two sensors with id, type, and humidity or temperature.



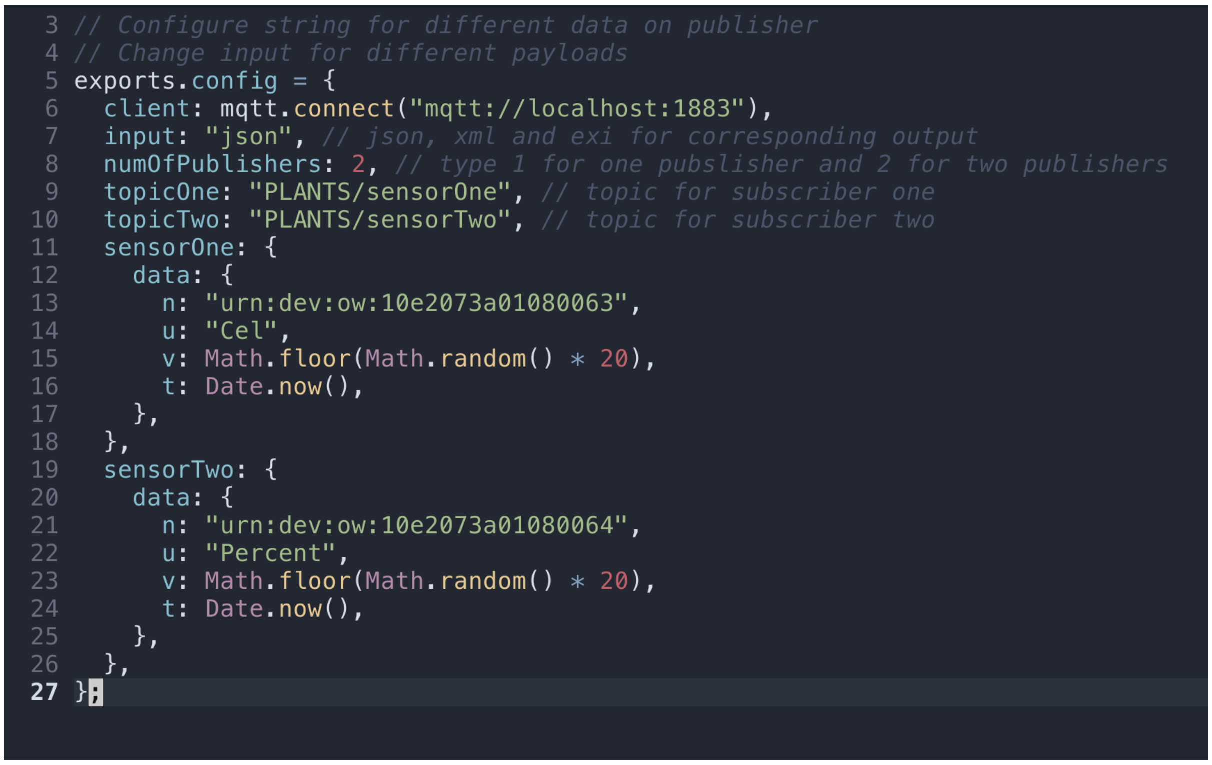
```
// Configure string for different data on publisher
// Change input for different payloads
exports.config = {
  client: mqtt.connect("mqtt://localhost:1883"),
  input: "json", // json, xml and exi for corresponding output
  numOfPublishers: 2, // type 1 for one pubslisher and 2 for two publishers
  topicOne: "PLANTS/sensorOne", // topic for subscriber one
  topicTwo: "PLANTS/sensorTwo", // topic for subscriber two
  sensorOne: {
    data: {
      n: "urn:dev:ow:10e2073a01080063",
      u: "Cel",
      v: Math.floor(Math.random() * 20),
      t: Date.now(),
    },
  },
  sensorTwo: {
    data: {
      n: "urn:dev:ow:10e2073a01080064",
      u: "Percent",
      v: Math.floor(Math.random() * 20),
      t: Date.now(),
    },
  },
};
```

inside the config.js we have a line that accepts either “json”, “xml” or “exi”. configuring the string we are sending different payloads from publisher, if json is picked the payload from publisher is json data, xml the payload is sent as xml and for exi the payload is sent as exi format.

Config.js

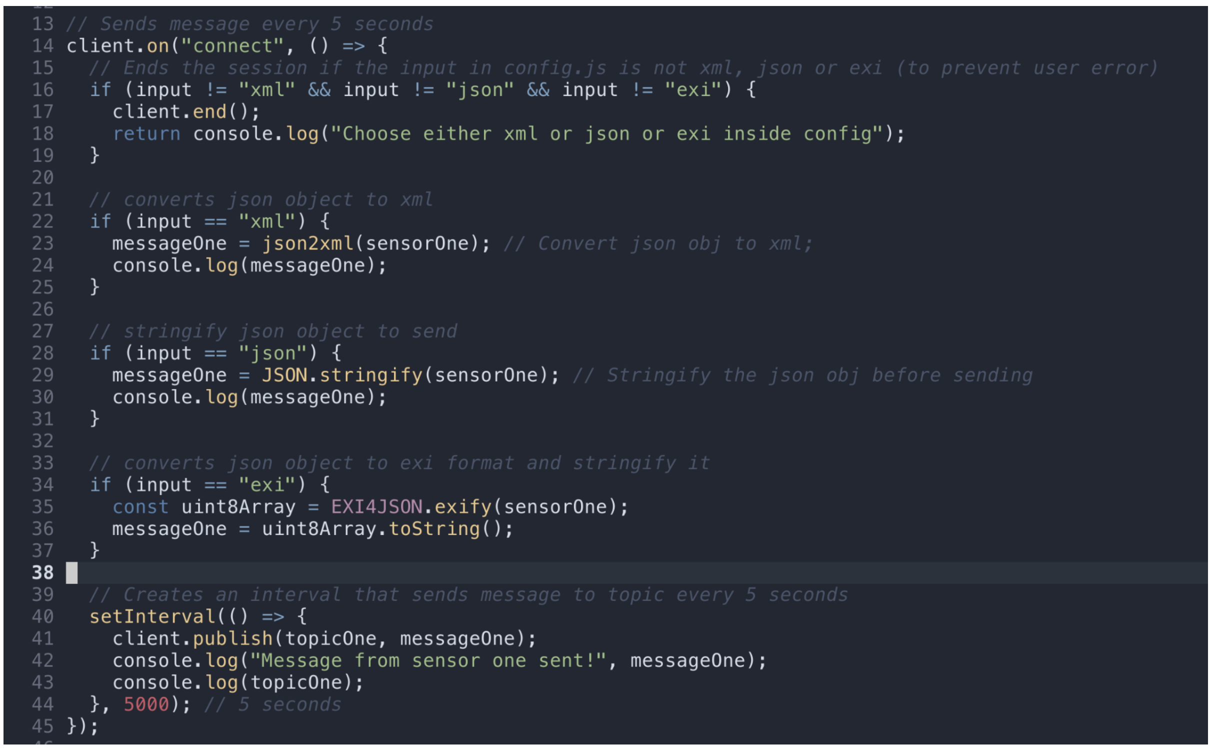


```
  input: "exi", // json, xml and exi for corresponding output
```



```
// Configure string for different data on publisher
// Change input for different payloads
exports.config = {
  client: mqtt.connect("mqtt://localhost:1883"),
  input: "json", // json, xml and exi for corresponding output
  numOfPublishers: 2, // type 1 for one pubslisher and 2 for two publishers
  topicOne: "PLANTS/sensorOne", // topic for subscriber one
  topicTwo: "PLANTS/sensorTwo", // topic for subscriber two
  sensorOne: {
    data: {
      n: "urn:dev:ow:10e2073a01080063",
      u: "Cel",
      v: Math.floor(Math.random() * 20),
      t: Date.now(),
    },
  },
  sensorTwo: {
    data: {
      n: "urn:dev:ow:10e2073a01080064",
      u: "Percent",
      v: Math.floor(Math.random() * 20),
      t: Date.now(),
    },
  },
};
```

pub.js



```
// Sends message every 5 seconds
client.on("connect", () => {
  // Ends the session if the input in config.js is not xml, json or exi (to prevent user error)
  if (input != "xml" && input != "json" && input != "exi") {
    client.end();
    return console.log("Choose either xml or json or exi inside config");
  }

  // converts json object to xml
  if (input == "xml") {
    messageOne = json2xml(sensorOne); // Convert json obj to xml;
    console.log(messageOne);
  }

  // stringify json object to send
  if (input == "json") {
    messageOne = JSON.stringify(sensorOne); // Stringify the json obj before sending
    console.log(messageOne);
  }

  // converts json object to exi format and stringify it
  if (input == "exi") {
    const uint8Array = EXI4JSON.exify(sensorOne);
    messageOne = uint8Array.toString();
  }

  // Creates an interval that sends message to topic every 5 seconds
  setInterval(() => {
    client.publish(topicOne, messageOne);
    console.log("Message from sensor one sent!", messageOne);
    console.log(topicOne);
  }, 5000); // 5 seconds
});
```