HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA AN TOÀN THÔNG TIN

Môn học: IOT và ứng dụng

Báo Cáo API

Họ và tên: Trần Trọng Mạnh

Mã sinh viên: B21DCAT127

Nhóm môn học: 05

Giảng viên: Nguyễn Quốc Uy

Hà Nội, 4/2024

MỤC LỤC

1. Cấu trúc project 3

2. API 3

a. Socket.io, gửi data từ hardware lên web theo thời gian thực 4

b. API để xử lý bật tắt thiết bị, cập nhật trạng thái thiết bị từ hardware lên web 5

- Update status to swich 5
- Xử lý bật tắt thiết bị theo topic/{device} 7

c. Lưu và lấy dữ liệu cảm biến vào MongoDB 9

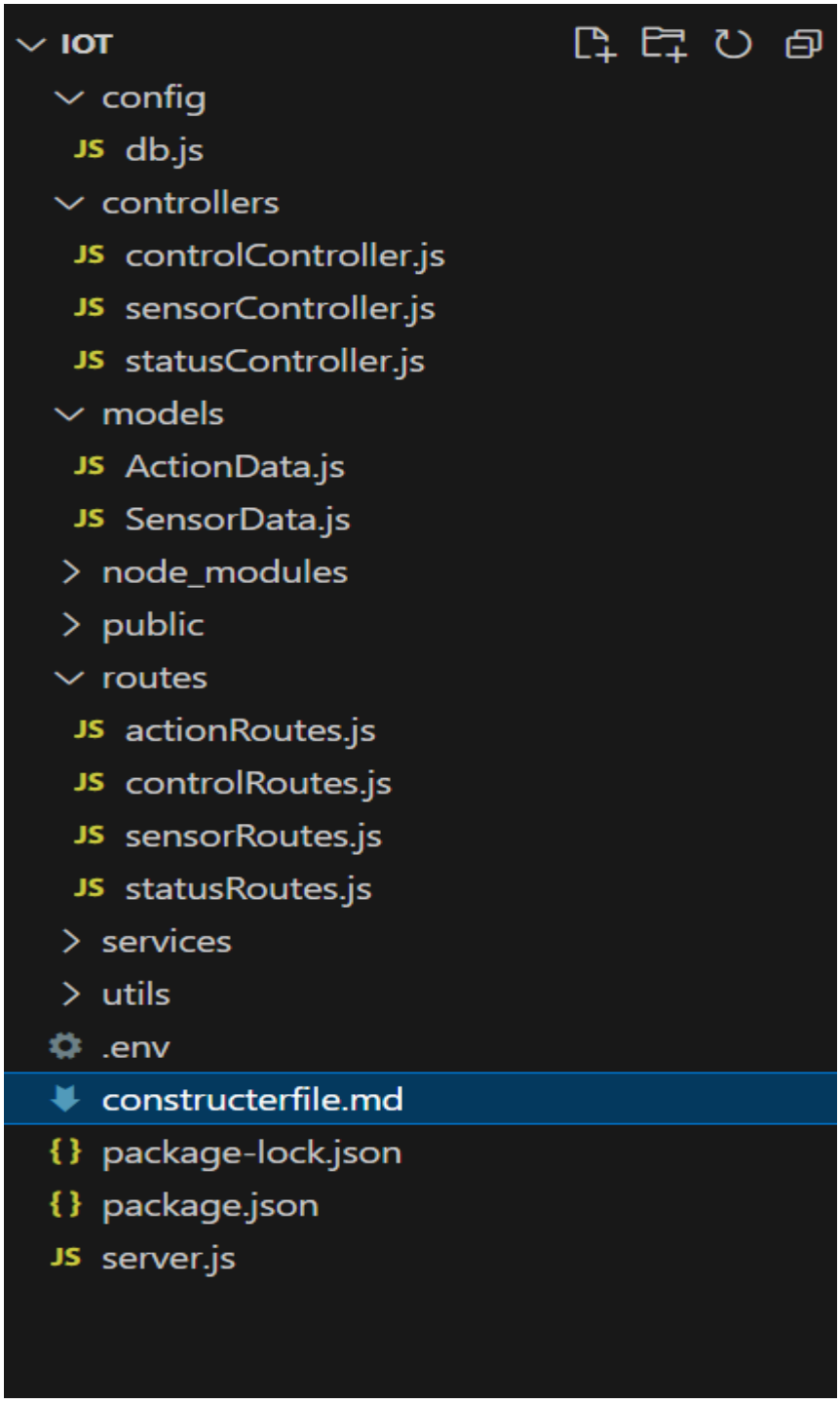
- Lưu dữ liệu cảm biến 9
- API: api/sensor-data/filter 10

d. Lưu và lấy data trạng thái của thiết bị vào MongoDB 12

- Thực hiện lưu data vào database mỗi khi bật tắt thiết bị 12

1. Cấu trúc project

Hardware: ESP8266, Arduino IDE 2.3.3
Backend: NoteJS
Database: MongoDB
FontEnd: Html, css, bootstrap



2. API

a. Socket.io, gửi data từ hardware lên web theo thời gian thực

Lấy data từ mqtt lên backend thông qua topic home/sensor

```
} else {
  // Gửi dữ liệu nhiệt độ, độ ẩm và ánh sáng qua MQTT (topic cũ)
  char sensorData[200];
  snprintf(sensorData, sizeof(sensorData),
    "{\"temperature\": %.2f, \"humidity\": %.2f, \"lighting\": %d}",
    temperature, humidity, (1030-lightLevel));
  mqttClient.publish("home/sensor", sensorData);  // Cập nhật với topic mong muốn của bạn
```

Mqtt.service.js

Thiết lập kết nối client

```
this.client.on('connect', () => {
  console.log('Kết nối MQTT thành công');
  // Đăng ký các topic cần thiết
  this.client.subscribe('home/sensor', (err) => {
    if (err) {
      console.error('Lỗi khi subscribe home/sensor:', err);
    }
  });
```

```
51        // Gửi dữ liệu về phía client
52        res.json(data);
53      } catch (error) {
54        console.error('Lỗi khi lấy dữ liệu sensor:', error);
55        res.status(500).json({ message: 'Server Error' });
56      }
57    };
```

Thiết lập mqtt gửi data bằng Socket.io để gửi data theo thời gian thực từ hardware lên backend. Sau đó dụng hàm: this.emit('newSensorData',formattedData);

```
this.client.on('message', (topic, message) => {
  try {
    const data = JSON.parse(message.toString());

    if (topic === 'home/sensor') {
      const formattedData = {
        temperature: data.temperature,
        humidity: data.humidity,
        light: data.lighting, // Đổi từ 'lighting' thành 'light'
        createdAt: new Date(),
      };

      // Phát dữ liệu tới frontend qua Socket.io nếu đã thiết lập
      if (this.ioInstance) {
        this.ioInstance.emit('newSensorData', formattedData);
      }

      // Phát sự kiện 'newSensorData' cho các listeners khác (như dataAggregator)
      this.emit('newSensorData', formattedData);
```

Phía frontend: Kết nối với socket.io, nhận data từ backend

```
// Kết nối với Socket.io
const socket = io(); // Kết nối tới cùng origin
```

```
// Nhận dữ liệu thời gian thực từ Socket.io
socket.on('newSensorData', (data) => {
  console.log('Received new sensor data:', data);
  updateChartTemperatureHumidity(data);
  updateChartLight(data);
  updatePanels(data);
});
```

sau đó dùng các hàm để cập nhật data cho nhiệt độ, độ ẩm, ảnh sáng và khối thay đổi màu theo giá trị.

b. API để xử lý bật tắt thiết bị, cập nhật trạng thái thiết bị từ hardware lên web

- Update status to swich

Khi load trang thì sẽ gọi api để cập nhật status cho thiết bị. Và sau đó nếu như có sự thay đổi gì từ hardware thì nó sẽ gửi lên client qua topic/status

```
// Kiểm tra xem có thay đổi trạng thái nào không
if (currentLedState != prevLedState || currentFanState != prevFanState || currentAcState != prevAcState) {
  // Nếu có thay đổi, gửi dữ liệu MQTT

  // Chuyển đổi trạng thái sang ON/OFF
  String ledStatus = (currentLedState == HIGH) ? "ON" : "OFF";
  String fanStatus = (currentFanState == HIGH) ? "ON" : "OFF";
  String acStatus = (currentAcState == HIGH) ? "ON" : "OFF";

  // Đóng gói JSON và gửi trạng thái thiết bị qua MQTT
  char statusData[150];
  snprintf(statusData, sizeof(statusData),
    "{\"led1\": \"%s\", \"fan\": \"%s\", \"ac\": \"%s\"}",
    ledStatus.c_str(), fanStatus.c_str(), acStatus.c_str());
  mqttClient.publish("home/status", statusData);  // Gửi lên topic "home/status"

  // Cập nhật trạng thái trước đó thành trạng thái hiện tại
  prevLedState = currentLedState;
  prevFanState = currentFanState;
  prevAcState = currentAcState;

  // Debug thông tin trạng thái đã gửi
  Serial.print("LED1: ");
  Serial.print(ledStatus);
  Serial.print(" | Fan: ");
  Serial.print(fanStatus);
  Serial.print(" | AC: ");
  Serial.println(acStatus);
}
```

Nếu có sự kiện sẽ được gửi theo topic/status

```
    this.client.on('message', (topic, message) => {
      try {
        const data = JSON.parse(message.toString());

        if (topic === 'home/sensor') {···
        } else if (topic === 'home/status') {
          // Cập nhật trạng thái thiết bị hiện tại
          this.currentDeviceStatus = {
            led1: data.led1,
            fan: data.fan,
            ac: data.ac
          };

          // Phát sự kiện tới frontend
          if (this.ioInstance) {
            this.ioInstance.emit('deviceStatusUpdate', this.currentDeviceStatus);
          }

          // Phát sự kiện 'deviceStatusUpdate' cho các listeners khác
          this.emit('deviceStatusUpdate', this.currentDeviceStatus);
        }

      } catch (e) {
        console.error('Lỗi phân tích JSON:', e);
      }
    });
  }
```

Sau đó được socket lắng nghe và cập nhật trên frontend

```
// Lấy trạng thái thiết bị khi trang tải
window.onload = () => {
  fetch('/api/status')
    .then(response => response.json())
    .then(data => {
      updateSwitches(data);
    })
    .catch(error => console.error('Error fetching status:', error));
};
```

```
// Lắng nghe sự kiện cập nhật trạng thái thiết bị từ Socket.io
socket.on('deviceStatusUpdate', (status) => {
  console.log('Device status update:', status);
  updateSwitches(status);
});
```

- Xử lý bật tắt thiết bị theo topic/{device}

```
79    // Kiểm tra và điều khiển thiết bị theo topic và payload
80    if (String(topic) == "home/led1") {···
87    } else if (String(topic) == "home/fan") {···
94    } else if (String(topic) == "home/ac") {···
101   } else if (String(topic) == "home/all") {
102     if (message == "ON") {···
106     } else if (message == "OFF") {···
110     }
111     digitalWrite(LED1, ledState1);
112     digitalWrite(FANPIN, fanState);
113     digitalWrite(ACIN, acState);
114   }
115 }
```

Xử lý khi frontend post API api/control

```
// Hàm gửi yêu cầu điều khiển thiết bị
function controlDevice(device, action) {
  fetch('/api/control', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ device, action })
  })
    .then(response => response.json())
    .then(data => {
      console.log(`Control ${device}:`, data.message);
    })
    .catch(error => console.error(`Error controlling ${device}:`, error));
}
```

Phía backend

routes > JS controlRoutes.js > ...

```
1  // routes/controlRoutes.js
2  const express = require('express');
3  const router = express.Router();
4  const controlController = require('../controllers/controlController');
5
6  // POST /api/control - Điều khiển thiết bị
7  router.post('/', controlController.controlDevice);
8
9  module.exports = router;
10
```

controllers > JS controlController.js > ...

```
exports.controlDevice = async (req, res) => {
  const { device, action } = req.body;

  let topic = '';
  let message = '';

  // Xác định topic và message dựa trên thiết bị và hành động
  switch (device) {
    case 'quạt':
      topic = 'home/fan';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    case 'điều hòa':
      topic = 'home/ac';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    case 'đèn':
      topic = 'home/led1';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    default:
      return res.status(400).json({ message: 'Thiết bị không hợp lệ' });
  }

  // Gửi lệnh tới MQTT broker sử dụng MQTT client đã kết nối
  mqttService.client.publish(topic, message, { qos: 1 }, (err) => {
    if (err) {
      console.error('Lỗi khi gửi lệnh MQTT:', err);
      return res.status(500).json({ message: 'Lỗi khi gửi lệnh đến MQTT' });
    }

    console.log(`Đã gửi lệnh tới ${topic}: ${message}`);
    res.json({ message: `${device} đã được ${action === 'on' ? 'bật' : 'tắt'}` });
  });
  const actionData = new ActionData({
    device,   // Sử dụng đúng tên biến
    status: action,
    // 'time' sẽ tự động được đặt bởi default trong schema
  });
```

Gửi lệnh tới mqtt thông qua mqttService.client.publish

c. Lưu và lấy dữ liệu cảm biến vào MongoDB

- Lưu dữ liệu cảm biến

Lắng nghe sự kiện newSensorData được khai báo ở trên để gửi data lên web thông qua socket.io

Xử lý data sau đó lưu vào MongoDB

utils > JS dataAggregator.js > ...

```
// utils/dataAggregator.js
const SensorData = require('../models/SensorData');

let latestData = null;

const startDataAggregation = (mqttService) => {
  // Lắng nghe sự kiện 'newSensorData'
  mqttService.on('newSensorData', (data) => {
    latestData = data;
  });

  // Thực hiện lưu dữ liệu mỗi 2 phút
  setInterval(async () => {
    if (latestData) {
      try {
        const dataToSave = {
          temperature: latestData.temperature,
          humidity: latestData.humidity,
          light: latestData.light,
          createdAt: latestData.createdAt,
        };
        await SensorData.create(dataToSave);
        console.log('Dữ liệu sensor đã được lưu vào MongoDB:', dataToSave);
      } catch (error) {
        console.error('Lỗi khi lưu dữ liệu vào MongoDB:', error);
      }
    } else {
      console.log('Không có dữ liệu sensor mới để lưu.');
    }
  }, 1000 * 1000); // Thời gian bao lâu thì lưu data 1 lần
};

module.exports = { startDataAggregation };

```

Sau đó lấy data từ database build lên trang datasensor.html

- API: api/sensor-data/filter

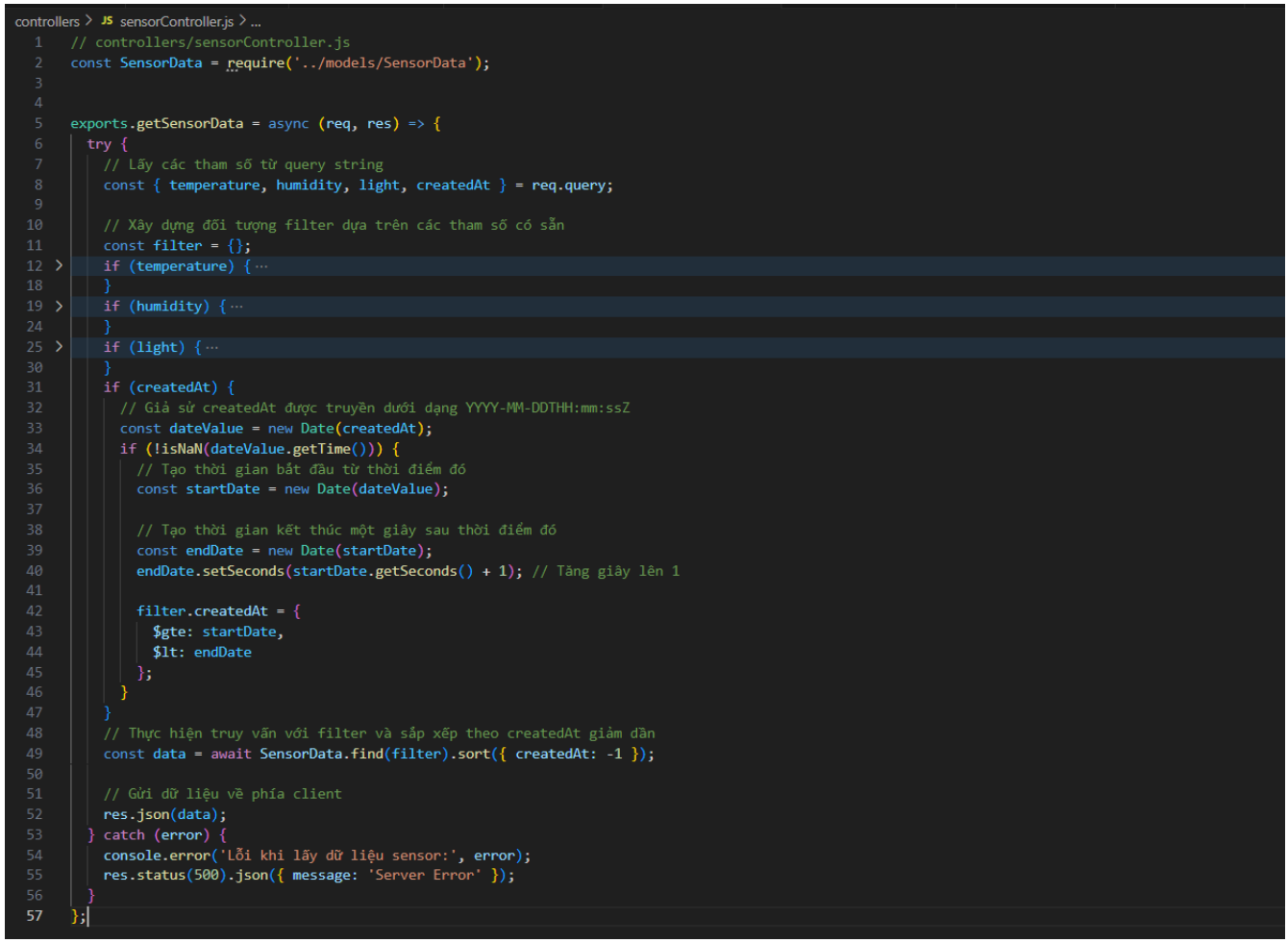
```
function fetchData(filters = {}) {
  $.ajax({
    url: 'http://localhost:3000/api/sensor-data',
    type: 'GET',
    data: filters,
    success: function (response) {
      const data = response.map((item, index) => ({
        id: index + 1,
        temperature: item.temperature,
        humidity: item.humidity,
        light: item.light,
        createdAt: formatDate(new Date(item.createdAt))
      }));
      table.clear().rows.add(data).draw();
    },
    error: function (error) {···
  });
}

fetchData();

$('#submitFilters').on('click', function () {
  const filterType = $('#filterType').val();
  let filterValue = $('#filterValue').val();

  const filters = {};
  if (filterType && filterValue) {
    if (filterType === 'createdAt') {
      // Chuyển đổi chuỗi thời gian thành ISO format
      filterValue = convertToISODate(filterValue);
    }
    filters[filterType] = filterValue;
  }

  fetchData(filters);
});

$('#resetFilters').on('click', function () {
  $('#filterType').val('temperature');
  $('#filterValue').val('');
  fetchData();
});
```

Xử lý backend để truy vấn dữ liệu vào database

routes > JS sensorRoutes.js > sensorController

```
1   // routes/sensorRoutes.js
2   const express = require('express');
3   const router = express.Router();
4   const sensorController = require('../controllers/sensorController');
5
6   // GET /api/sensor-data - Get sensor data history
7   router.get('/', sensorController.getSensorData);
8
9   module.exports = router;
10
```

Xử lý dữ liệu thời gian tìm kiếm chính xác đến giây
Sau đó thực hiện truy vấn



```
controllers > JS sensorController.js > ...
// controllers/sensorController.js
const SensorData = require('../models/SensorData');


exports.getSensorData = async (req, res) => {
  try {
    // Lấy các tham số từ query string
    const { temperature, humidity, light, createdAt } = req.query;

    // Xây dựng đối tượng filter dựa trên các tham số có sẵn
    const filter = {};
    if (temperature) {···
    }
    if (humidity) {···
    }
    if (light) {···
    }
    if (createdAt) {
      // Giả sử createdAt được truyền dưới dạng YYYY-MM-DDTHH:mm:ssZ
      const dateValue = new Date(createdAt);
      if (!isNaN(dateValue.getTime())) {
        // Tạo thời gian bắt đầu từ thời điểm đó
        const startDate = new Date(dateValue);

        // Tạo thời gian kết thúc một giây sau thời điểm đó
        const endDate = new Date(startDate);
        endDate.setSeconds(startDate.getSeconds() + 1); // Tăng giây lên 1

        filter.createdAt = {
          $gte: startDate,
          $lt: endDate
        };
      }
    }
    // Thực hiện truy vấn với filter và sắp xếp theo createdAt giảm dần
    const data = await SensorData.find(filter).sort({ createdAt: -1 });

    // Gửi dữ liệu về phía client
    res.json(data);
  } catch (error) {
    console.error('Lỗi khi lấy dữ liệu sensor:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};
```

d. Lưu và lấy data trạng thái của thiết bị vào MongoDB

- Thực hiện lưu data vào database mỗi khi bật tắt thiết bị

Lấy post API api/control. Mỗi khi có 1 post API thì thực hiện lưu vào database

controllers > controlController.js > controlDevice > controlDevice

```
// controllers/controlController.js
const mqttService = require('../services/mqttService');
const ActionData = require('../models/ActionData'); // Thêm dòng này

exports.controlDevice = async (req, res) => {
  const { device, action } = req.body;

  let topic = '';
  let message = '';

  // Xác định topic và message dựa trên thiết bị và hành động
  switch (device) {
    case 'quạt':
      topic = 'home/fan';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    case 'điều hòa':
      topic = 'home/ac';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    case 'đèn':
      topic = 'home/led1';
      message = action === 'on' ? 'ON' : 'OFF';
      break;
    default:
      return res.status(400).json({ message: 'Thiết bị không hợp lệ' });
  }

  // Gửi lệnh tới MQTT broker sử dụng MQTT client đã kết nối
  mqttService.client.publish(topic, message, { qos: 1 }, (err) => {
    if (err) {
      console.error('Lỗi khi gửi lệnh MQTT:', err);
      return res.status(500).json({ message: 'Lỗi khi gửi lệnh đến MQTT' });
    }

    console.log(`Đã gửi lệnh tới ${topic}: ${message}`);
    res.json({ message: `${device} đã được ${action === 'on' ? 'bật' : 'tắt'}` });
  });
  const actionData = new ActionData({
    device,   // Sử dụng đúng tên biến
    status: action,
    // 'time' sẽ tự động được đặt bởi default trong schema
  });
  await actionData.save();
};
```

thực hiện lưu ở dòng 44

- Lấy data history-action-device
  frontend lấy api/action

```
$(document).ready(function () {
  const table = $('#sensorTable').DataTable({
    columns: [
      { data: 'id' },
      { data: 'device' },
      { data: 'status' },
      { data: 'time' },
    ],
    language: {
      url: "https://cdn.datatables.net/plug-ins/1.13.4/i18n/vi.json"
    },
    searching: false
  });

  function fetchData(filters = {}) {
    $.ajax({
      url: 'http://localhost:3000/api/action',
      type: 'GET',
      data: filters,
      success: function (response) {
        const data = response.map((item, index) => ({
          id: index + 1,
          device: item.device,
          status: item.status,
          time: formatDate(new Date(item.time))
        }));
        table.clear().rows.add(data).draw();
      },
      error: function (error) {
        console.error('Lỗi khi gọi API:', error);
      }
    });
  }

  fetchData();

  $('#submitFilters').on('click', function () {
    const filterType = $('#filterType').val();
    let filterValue = $('#filterValue').val();

    const filters = {};
    if (filterType && filterValue) {
      if (filterType === 'time') {
        // Chuyển đổi chuỗi thời gian thành ISO format
```

Phía backend

routes > JS actionRoutes.js > ...

```
// routes/actionRoutes.js
const express = require('express');
const router = express.Router();
const controlController = require('../controllers/controlController');

// GET /api/history - Get action data history
router.get('/', controlController.getActionData);

module.exports = router;
```

```
exports.getActionData = async (req, res) => {
  try {
    //Lấy các tham số từ query string
    const { device, status, time } = req.query;
    //Xây dựng qua filter
    const filter = {};
    if (device) {
      filter.device = device;
    }
    if (status) {
      filter.status = status;
    }
    if (time) {
      // Giả sử time được truyền dưới dạng YYYY-MM-DDTHH:mm:ssZ
      const dateValue = new Date(time);
      if (!isNaN(dateValue.getTime())) {
        // Tạo thời gian bắt đầu từ thời điểm đó
        const startDate = new Date(dateValue);

        // Tạo thời gian kết thúc một giây sau thời điểm đó
        const endDate = new Date(startDate);
        endDate.setSeconds(startDate.getSeconds() + 1); // Tăng giây lên 1

        filter.time = {
          $gte: startDate,
          $lt: endDate
        };
      }
    }
    // Fetch the latest 100 sensor data entries
    const data = await ActionData.find(filter).sort({ createdAt: -1 });

    res.json(data);
  } catch (error) {
    console.error('Lỗi khi lấy dữ liệu ensor:', error);
    res.status(500).json({ message: 'Lỗi khsi lấy dữ liệu sensor' });
  }
};
```