API Documentation for Arduino Sensor Data and Device Management

This document describes the available API endpoints for managing device registrations, device logins, and handling sensor data from Arduino devices. These APIs are designed to be secure and efficient, utilizing JWTs for authentication.

Base URL

```
bash
http://localhost:3001/api
```

Authentication

Most endpoints require authentication. You must provide a valid JWT in the `Authorization` header of your request.

Example Header:

```
makefile
Authorization: Bearer YOUR_JWT_TOKEN
```

Endpoints

---

1. Arduino Sensor Data

POST /arduino/data

This endpoint receives sensor data from Arduino devices.

Authorization: Bearer Token

Request Body:

```
json
{
  "temperature": "float",
  "humidity": "float"
}
```

Responses:

- **201 Created**

```
    json
 {
  "message": "Sensor data received successfully",
  "data": {
    "temperature": "float",
    "humidity": "float"
  }
}
```

- **401 Unauthorized**

```
json
•  {
•    "message": "Invalid token"
•  }
•
```

2. Device Registration

POST /devices/register

Register a new device in the system.

Request Body:

```
json
{
  "deviceId": "string",
  "model": "string",
  "owner": "string",
  "secret": "string"
}
```

Responses:

- **201 Created**

```
      json
•  {
  "message": "Device registered successfully",
  "device": {
    "deviceId": "string",
    "model": "string",
    "owner": "string"
  }
}
```

- **409 Conflict**

```
json
•  {
  "message": "Device already registered"
}
```

- **500 Internal Server Error**

json
- {
-   "message": "Error registering device"
- }
-

---

3. Device Login

POST /devices/device-login

Authenticate a device and receive a JWT for subsequent requests.

Request Body:

```
json
{
  "deviceId": "string",
  "secret": "string"
}
```

Responses:

- **200 OK**

  json
- {
  "token": "YOUR_JWT_TOKEN"
}

- **401 Unauthorized**

json
- {
  "message": "Invalid credentials"
}

- **500 Internal Server Error**

json
- {
-   "message": "Login error"
- }
-

Usage Examples

To interact with these APIs, you can use tools like curl, Postman, or write scripts using HTTP client libraries such as Axios.

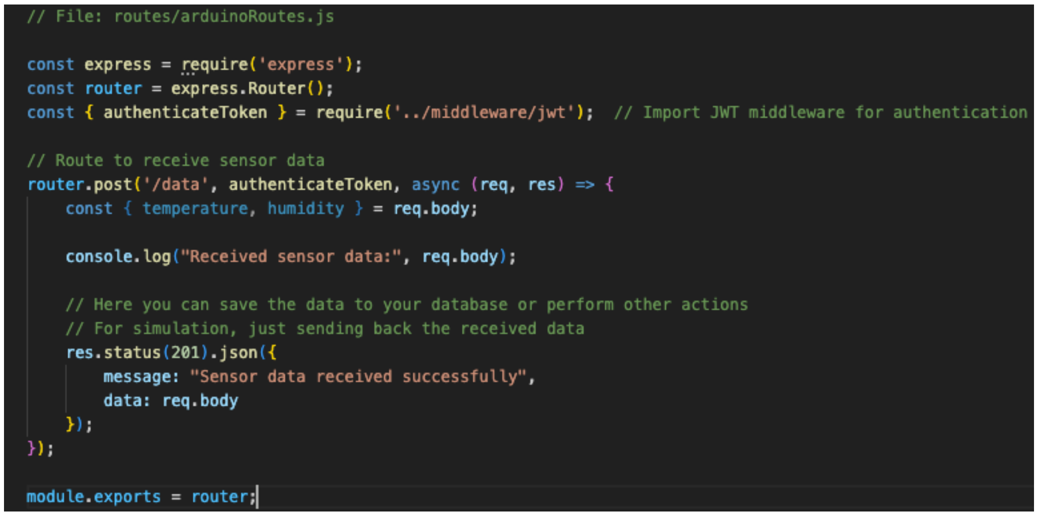
Example with curl:

Device Login:

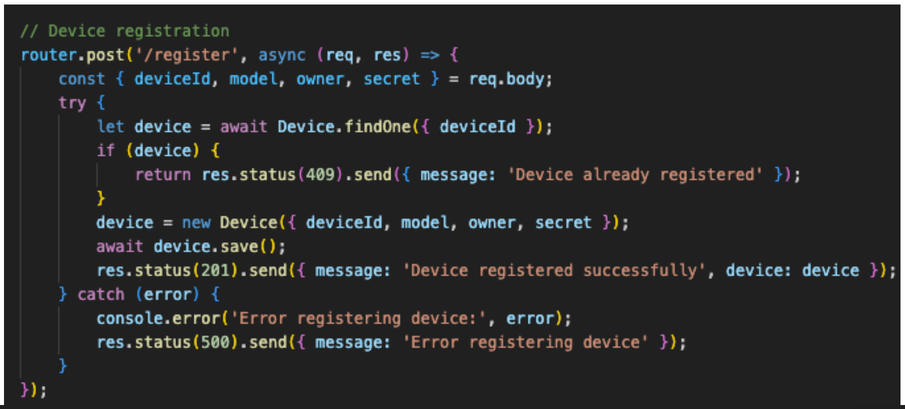
```
bash
curl -X POST http://localhost:3001/api/devices/device-login \
-H "Content-Type: application/json" \
-d '{"deviceId": "yourDeviceId", "secret": "yourSecret"}'
```

Handling Errors

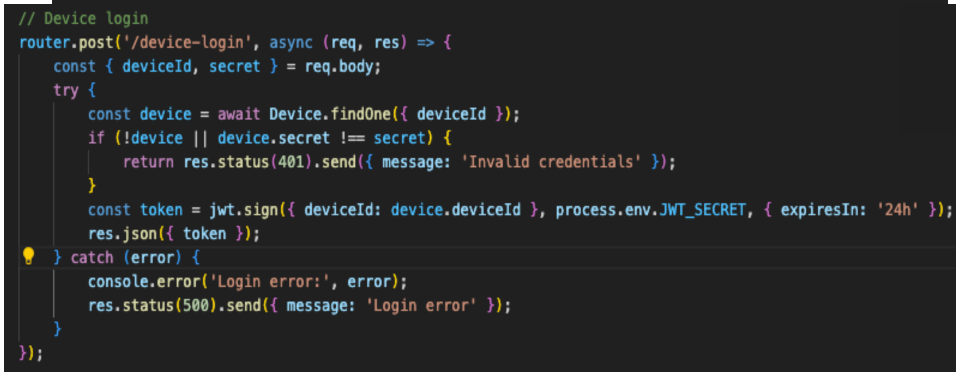
All endpoints are designed to return meaningful error messages and HTTP status codes to help diagnose issues.



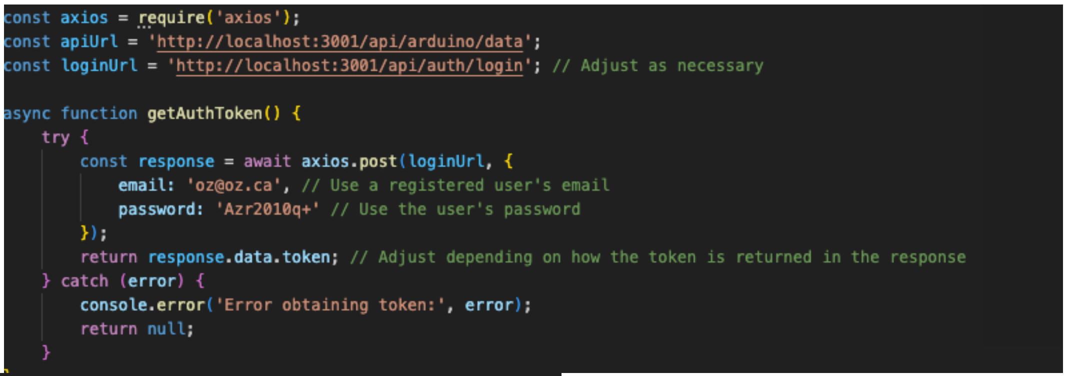
```
// File: routes/arduinoRoutes.js

const express = require('express');
const router = express.Router();
const { authenticateToken } = require('../middleware/jwt');  // Import JWT middleware for authentication

// Route to receive sensor data
router.post('/data', authenticateToken, async (req, res) => {
    const { temperature, humidity } = req.body;

    console.log("Received sensor data:", req.body);

    // Here you can save the data to your database or perform other actions
    // For simulation, just sending back the received data
    res.status(201).json({
        message: "Sensor data received successfully",
        data: req.body
    });
});

module.exports = router;
```



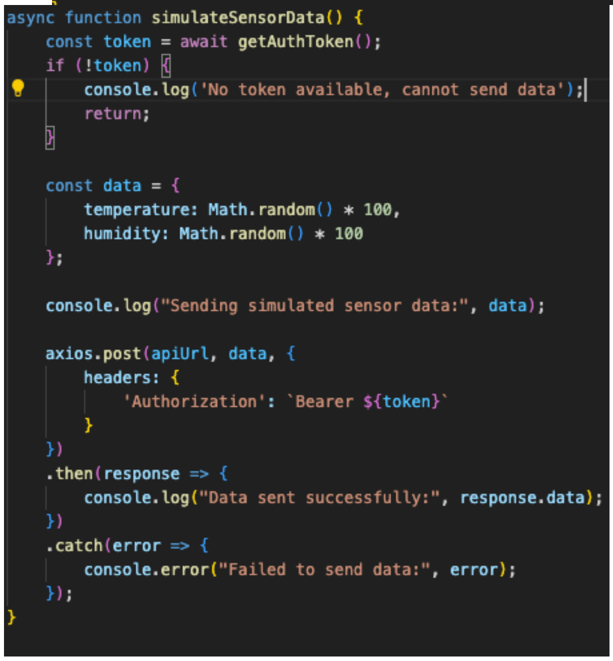
```
// Device registration
router.post('/register', async (req, res) => {
    const { deviceId, model, owner, secret } = req.body;
    try {
        let device = await Device.findOne({ deviceId });
        if (device) {
            return res.status(409).send({ message: 'Device already registered' });
        }
        device = new Device({ deviceId, model, owner, secret });
        await device.save();
        res.status(201).send({ message: 'Device registered successfully', device: device });
    } catch (error) {
        console.error('Error registering device:', error);
        res.status(500).send({ message: 'Error registering device' });
    }
});
```



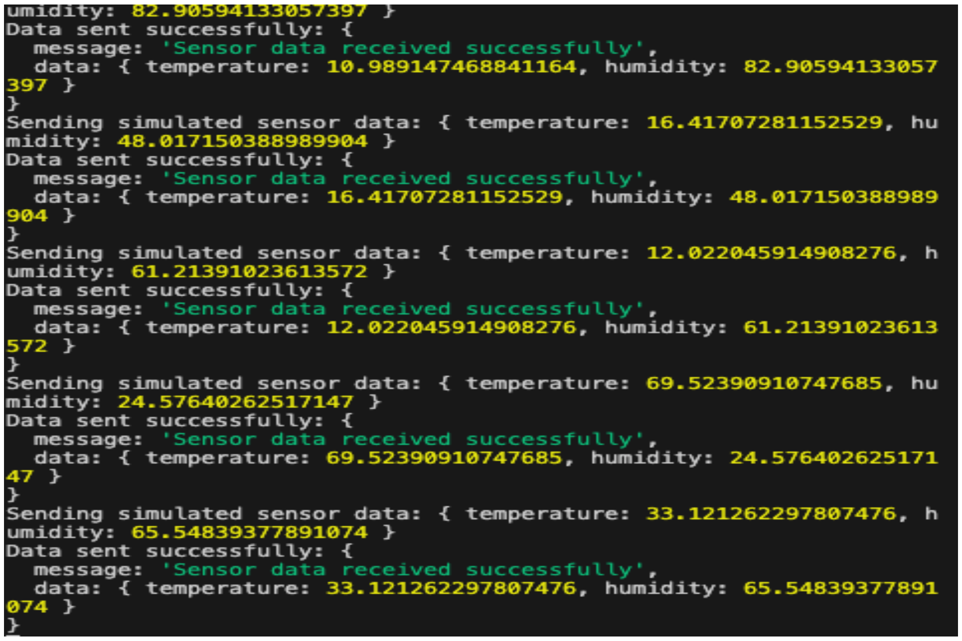
```
// Device login
router.post('/device-login', async (req, res) => {
    const { deviceId, secret } = req.body;
    try {
        const device = await Device.findOne({ deviceId });
        if (!device || device.secret !== secret) {
            return res.status(401).send({ message: 'Invalid credentials' });
        }
        const token = jwt.sign({ deviceId: device.deviceId }, process.env.JWT_SECRET, { expiresIn: '24h' });
        res.json({ token });
    } catch (error) {
        console.error('Login error:', error);
        res.status(500).send({ message: 'Login error' });
    }
});
```



```
const axios = require('axios');
const apiUrl = 'http://localhost:3001/api/arduino/data';
const loginUrl = 'http://localhost:3001/api/auth/login'; // Adjust as necessary

async function getAuthToken() {
    try {
        const response = await axios.post(loginUrl, {
            email: 'oz@oz.ca', // Use a registered user's email
            password: 'Azr2010q+' // Use the user's password
        });
        return response.data.token; // Adjust depending on how the token is returned in the response
    } catch (error) {
        console.error('Error obtaining token:', error);
        return null;
    }
```



```
async function simulateSensorData() {
    const token = await getAuthToken();
    if (!token) {
        console.log('No token available, cannot send data');
        return;
    }

    const data = {
        temperature: Math.random() * 100,
        humidity: Math.random() * 100
    };

    console.log("Sending simulated sensor data:", data);

    axios.post(apiUrl, data, {
        headers: {
            'Authorization': `Bearer ${token}`
        }
    })
    .then(response => {
        console.log("Data sent successfully:", response.data);
    })
    .catch(error => {
        console.error("Failed to send data:", error);
    });
}
```



```
umidity: 82.90594133057397 }
Data sent successfully: {
  message: 'Sensor data received successfully',
  data: { temperature: 10.989147468841164, humidity: 82.90594133057
397 }
}
Sending simulated sensor data: { temperature: 16.41707281152529, hu
midity: 48.017150388989904 }
Data sent successfully: {
  message: 'Sensor data received successfully',
  data: { temperature: 16.41707281152529, humidity: 48.017150388989
904 }
}
Sending simulated sensor data: { temperature: 12.022045914908276, h
umidity: 61.21391023613572 }
Data sent successfully: {
  message: 'Sensor data received successfully',
  data: { temperature: 12.022045914908276, humidity: 61.21391023613
572 }
}
Sending simulated sensor data: { temperature: 69.52390910747685, hu
midity: 24.57640262517147 }
Data sent successfully: {
  message: 'Sensor data received successfully',
  data: { temperature: 69.52390910747685, humidity: 24.576402625171
47 }
}
Sending simulated sensor data: { temperature: 33.121262297807476, h
umidity: 65.54839377891074 }
Data sent successfully: {
  message: 'Sensor data received successfully',
  data: { temperature: 33.121262297807476, humidity: 65.54839377891
074 }
}
```

Timer API Documentation

Base URL: `http://localhost:3001/api/timer`

This API allows you to start, stop, reset, and adjust a timer on the server.

---

1. Start Timer

Endpoint: `/start`

Method: `POST`

Description: Starts or restarts the timer with a specified duration.

Request Body:

```
json
{
  "duration": int // Duration in milliseconds
}
```

Responses:

- **200 OK**

  Body:

  ```
  json
  ```

-
  ```
   {
    "message": "Timer started"
  }
  ```

- **400 Bad Request**

Body:

json

```
•  {
•    "error": "Invalid duration"
•  }
•
```

Example:

```bash
curl -X POST http://localhost:3001/api/timer/start \
-H "Content-Type: application/json" \
-d '{"duration": 5000}'
```

---

2. Stop Timer

Endpoint: `/stop`

Method: `POST`

Description: Stops the currently active timer.

Request Body: None

Responses:

- **200 OK**

  Body:

  json

-
  ```
   {
    "message": "Timer stopped"
  }
  ```

- **404 Not Found**

Body:

json

```
•  {
```

- `"error": "No active timer to stop"`
- `}`
-

Example:

```
bash
curl -X POST http://localhost:3001/api/timer/stop
```

---

3. Reset Timer

Endpoint: `/reset`

Method: `POST`

Description: Resets the currently active timer, starting it over with the original duration.

Request Body: None

Responses:

- **200 OK**

  Body:

  ```
  json
  ```
- `{`
```
  "message": "Timer reset with the remaining time"
}
```

- **404 Not Found**

Body:

```
json
```
- `{`
- `"error": "No active timer to reset"`
- `}`
-

Example:

```
bash
curl -X POST http://localhost:3001/api/timer/reset
```

---

4. Adjust Timer's Duration

Endpoint: `/adjust`

Method: `POST`

Description: Adjusts the duration of the currently active timer to a new duration.

Request Body:

```
json
{
  "newDuration": int // New duration in milliseconds
}
```

Responses:

- **200 OK**

    Body:

    ```
    json
    ```

-
  ```
   {
  "message": "Timer duration adjusted"
}
```

- **400 Bad Request**

Body:

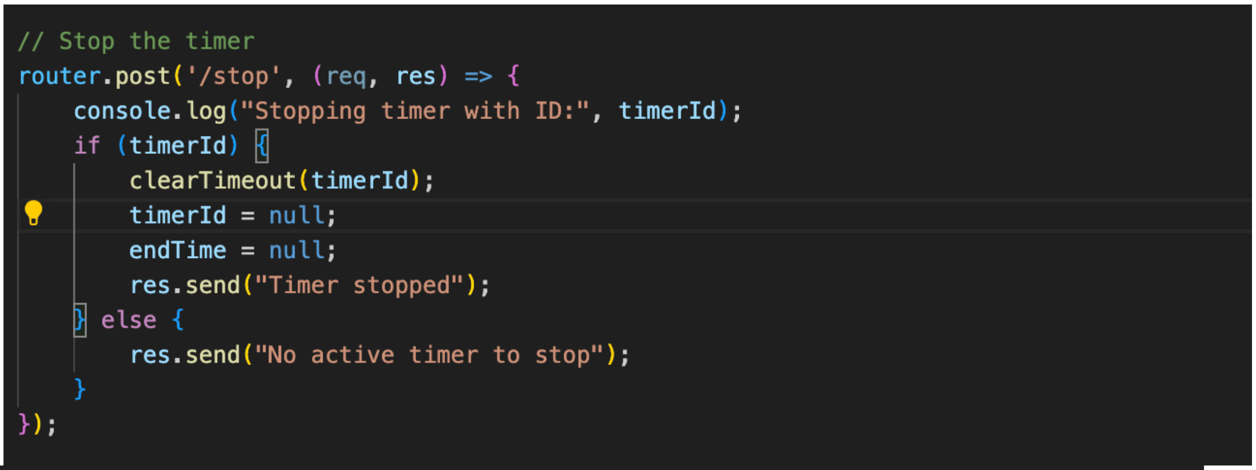
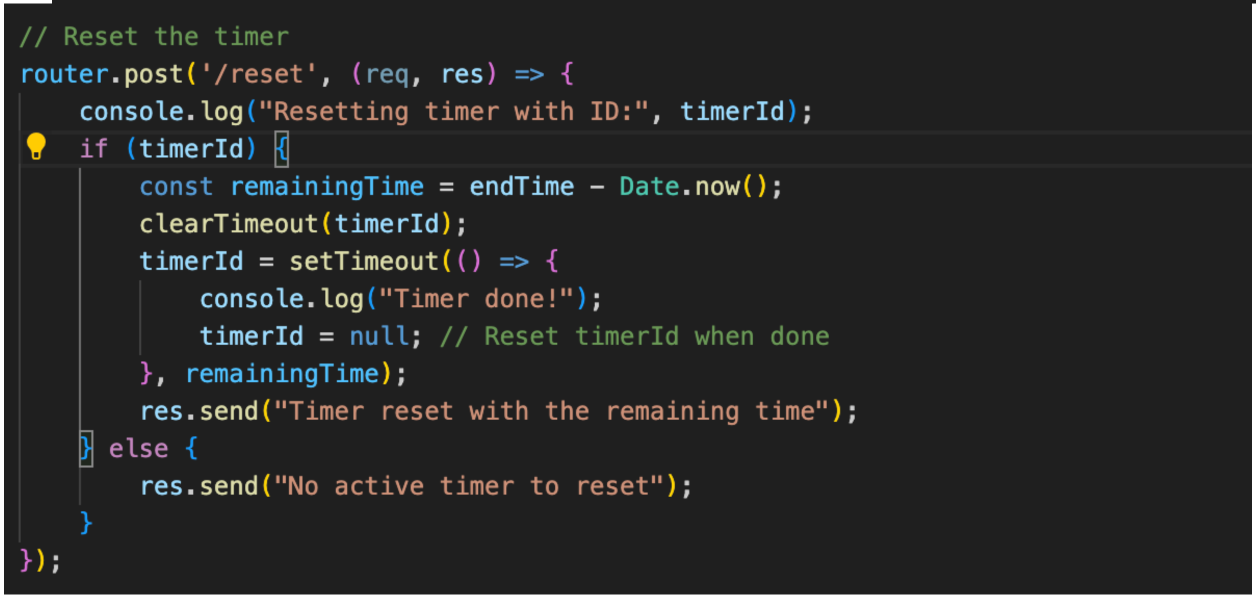
```
json
```
- `{`
- `  "error": "No active timer to adjust or no new duration provided"`
- `}`
-

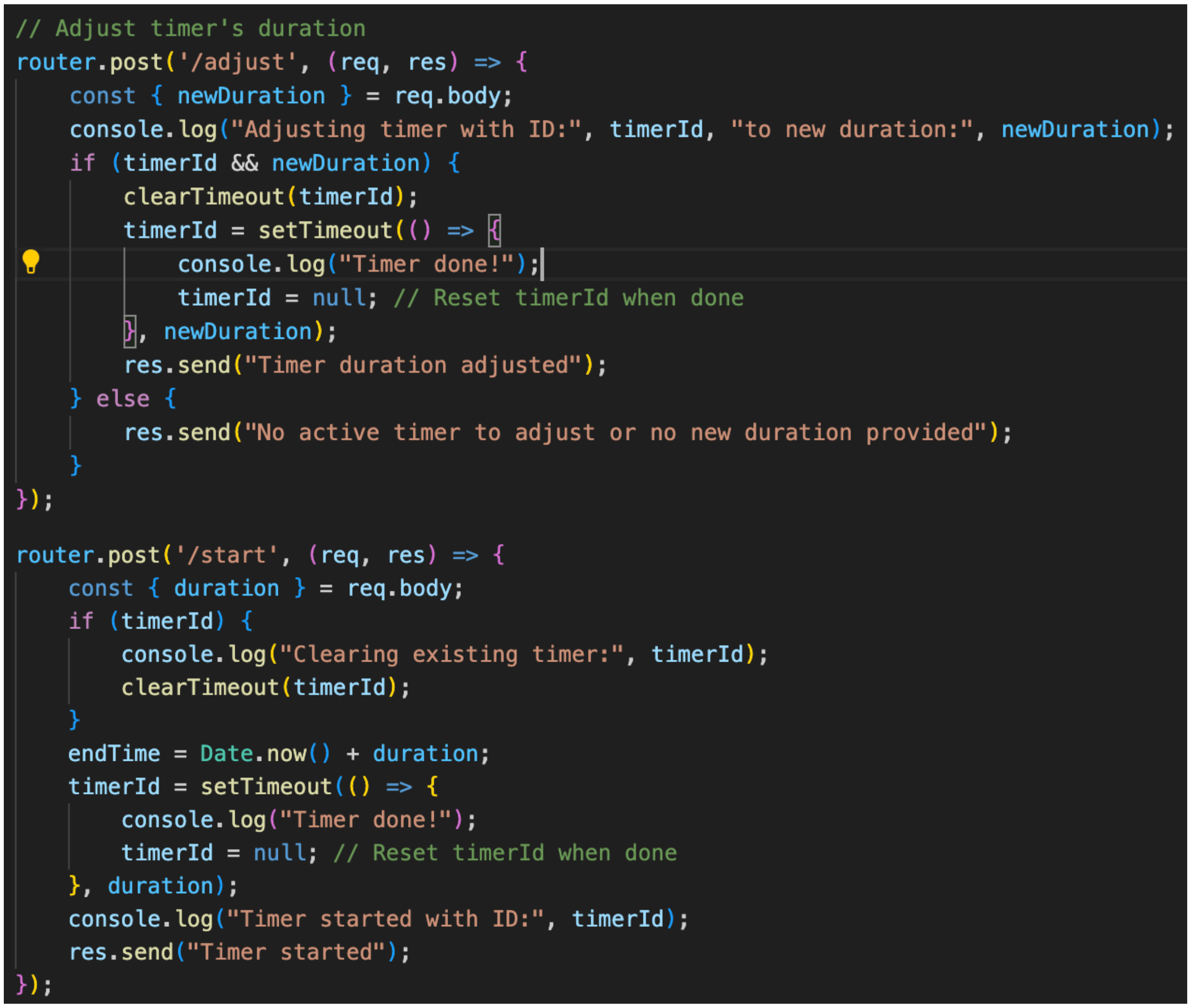
Example:

```
bash
curl -X POST http://localhost:3001/api/timer/adjust \
-H "Content-Type: application/json" \
-d '{"newDuration": 10000}'
```

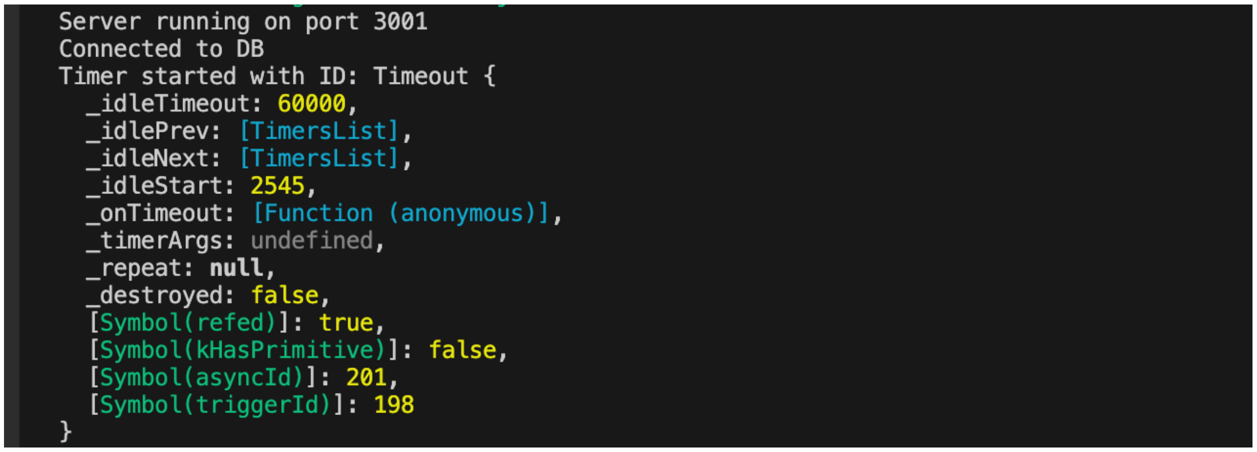
Notes:

- All endpoints must be accessed by making HTTP POST requests.
- The base URL may differ based on your server's configuration or deployment settings.
- Ensure the server is running before attempting to call these APIs.

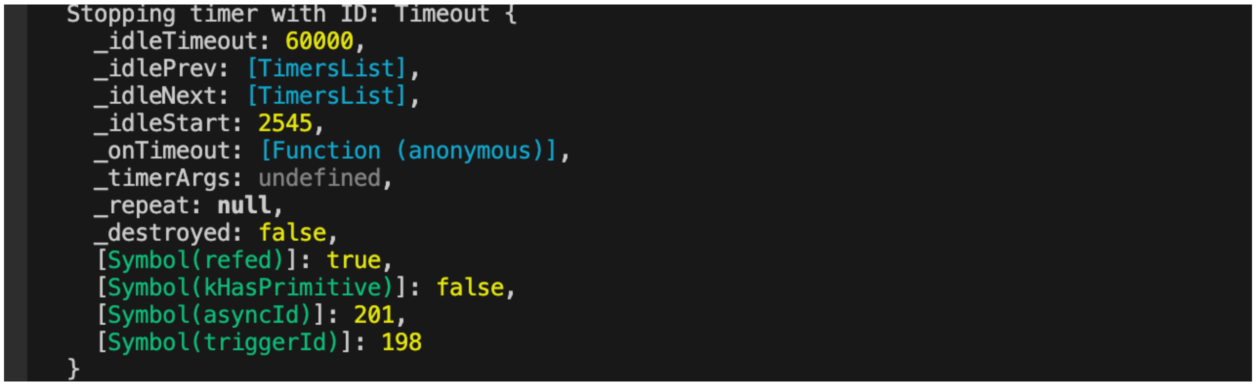

```javascript
// Stop the timer
router.post('/stop', (req, res) => {
    console.log("Stopping timer with ID:", timerId);
    if (timerId) {
        clearTimeout(timerId);
        timerId = null;
        endTime = null;
        res.send("Timer stopped");
    } else {
        res.send("No active timer to stop");
    }
});
```

```javascript
// Reset the timer
router.post('/reset', (req, res) => {
    console.log("Resetting timer with ID:", timerId);
    if (timerId) {
        const remainingTime = endTime - Date.now();
        clearTimeout(timerId);
        timerId = setTimeout(() => {
            console.log("Timer done!");
            timerId = null; // Reset timerId when done
        }, remainingTime);
        res.send("Timer reset with the remaining time");
    } else {
        res.send("No active timer to reset");
    }
});
```



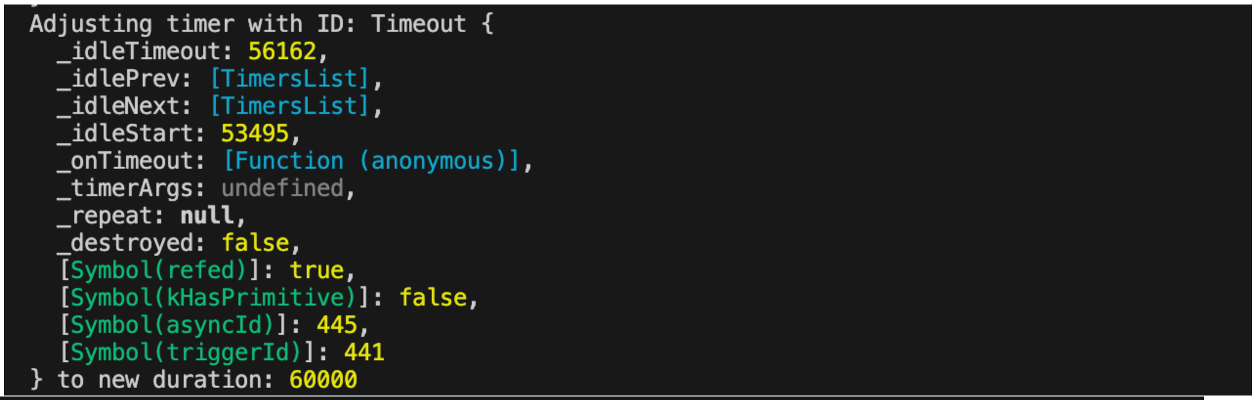
```
// Adjust timer's duration
router.post('/adjust', (req, res) => {
    const { newDuration } = req.body;
    console.log("Adjusting timer with ID:", timerId, "to new duration:", newDuration);
    if (timerId && newDuration) {
        clearTimeout(timerId);
        timerId = setTimeout(() => {
            console.log("Timer done!");
            timerId = null; // Reset timerId when done
        }, newDuration);
        res.send("Timer duration adjusted");
    } else {
        res.send("No active timer to adjust or no new duration provided");
    }
});

router.post('/start', (req, res) => {
    const { duration } = req.body;
    if (timerId) {
        console.log("Clearing existing timer:", timerId);
        clearTimeout(timerId);
    }
    endTime = Date.now() + duration;
    timerId = setTimeout(() => {
        console.log("Timer done!");
        timerId = null; // Reset timerId when done
    }, duration);
    console.log("Timer started with ID:", timerId);
    res.send("Timer started");
});
```



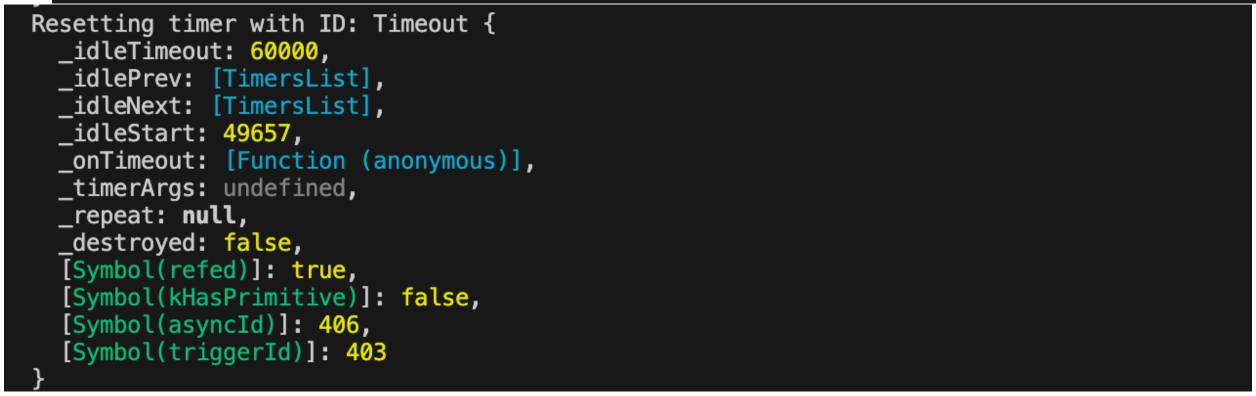
```
Server running on port 3001
Connected to DB
Timer started with ID: Timeout {
  _idleTimeout: 60000,
  _idlePrev: [TimersList],
  _idleNext: [TimersList],
  _idleStart: 2545,
  _onTimeout: [Function (anonymous)],
  _timerArgs: undefined,
  _repeat: null,
  _destroyed: false,
  [Symbol(refed)]: true,
  [Symbol(kHasPrimitive)]: false,
  [Symbol(asyncId)]: 201,
  [Symbol(triggerId)]: 198
}
```



```
  Stopping timer with ID: Timeout {
    _idleTimeout: 60000,
    _idlePrev: [TimersList],
    _idleNext: [TimersList],
    _idleStart: 2545,
    _onTimeout: [Function (anonymous)],
    _timerArgs: undefined,
    _repeat: null,
    _destroyed: false,
    [Symbol(refed)]: true,
    [Symbol(kHasPrimitive)]: false,
    [Symbol(asyncId)]: 201,
    [Symbol(triggerId)]: 198
  }
```



```
Adjusting timer with ID: Timeout {
  _idleTimeout: 56162,
  _idlePrev: [TimersList],
  _idleNext: [TimersList],
  _idleStart: 53495,
  _onTimeout: [Function (anonymous)],
  _timerArgs: undefined,
  _repeat: null,
  _destroyed: false,
  [Symbol(refed)]: true,
  [Symbol(kHasPrimitive)]: false,
  [Symbol(asyncId)]: 445,
  [Symbol(triggerId)]: 441
} to new duration: 60000
```



```
Resetting timer with ID: Timeout {
  _idleTimeout: 60000,
  _idlePrev: [TimersList],
  _idleNext: [TimersList],
  _idleStart: 49657,
  _onTimeout: [Function (anonymous)],
  _timerArgs: undefined,
  _repeat: null,
  _destroyed: false,
  [Symbol(refed)]: true,
  [Symbol(kHasPrimitive)]: false,
  [Symbol(asyncId)]: 406,
  [Symbol(triggerId)]: 403
}
```