Project Report: Smart Home App

Student Name: Barry McCabe

Student Number: x21156131

Contents

[1. Introduction 2](#_Toc155457964)

[2. Service Definitions 2](#_Toc155457965)

[3. Service Implementations 3](#_Toc155457966)

[4. Naming Services 4](#_Toc155457967)

[5. Remote Error Handling & Advanced Features 4](#_Toc155457968)

[6. Client - Graphical User Interface (GUI) 5](#_Toc155457969)

[7. GitHub Integration 5](#_Toc155457970)

[8. Appendix 6](#_Toc155457971)

[Github link 7](#_Toc155457972)

[Smart Home App Code 7](#_Toc155457973)

# Introduction

The project was to develop a Smart home system (smart devices, smart thermostats). This report outlines the development of Smart Home App, which creates an interconnected environment where smart devices collaborate to enhance the end users living experience within the home.

# The System will consist of three primary services:

1. Lighting Control: This allows the user to turn lighting on/off and the adjust the brightness of two zones independently.
2. Thermostat Control: This provides the user the current temperature and allows the user to the set the temperature.
3. Security Control: This will allow the user to activate/deactivate the security and provide the end user real-time updates and alerts.

It will also have a system status/check service, implemented in the console.

# Service Definitions

The system comprises several services, each fulfilling a specific functionality:

**LightService:** Manages the on/off state of lights. Uses ToggleLight for toggling light status and GetStatus to retrieve current light settings.

* **Message Formats:**

ToggleRequest: Contains a boolean status indicating the desired light state.

ToggleResponse: Returns the result of the toggle action and a message (e.g., "Lights are now ON").

**BrightnessService**: Adjusts the brightness of lights in real-time. Utilises client-side streaming RPC.

* **Message Formats:**

BrightnessRequest: Includes zone ID and brightness level.

BrightnessResponse: Provides feedback or confirmation of the brightness adjustment.

**ThermostatService:** Controls the home's temperature. Implements a simple RPC for setting the temperature.

* **Message Formats:**

TemperatureRequest: Contains the desired temperature setting.

TemperatureResponse: Confirms the action and returns a related message (e.g., "Temperature set to 22°C").

**TemperatureService:** Provides the current temperature. Uses a simple RPC.

* **Message Format:**

CurrentTemperatureResponse: Returns the current temperature value.

**AlertService:** Delivers security alerts to the client using server-side streaming RPC.

* **Message Format:**

Alert: Contains the alert message and a timestamp.

**SecurityService:** Manages security zones activation through client-side streaming RPC.

* **Message Formats:**

ZoneActivationRequest: Carries the ID of the zone and a boolean indicating activation status.

ActivationResponse: Confirms the status of zone activation.

**SystemStatusService:** Checks overall system status through bidirectional streaming RPC.

* **Message Formats:**

StatusCheckRequest: Contains a command or query for system status.

StatusCheckResponse: Returns a status message.

Message formats conform to proto3 syntax, ensuring structured and standardised data exchange.

# Service Implementations

**LightService Implementation**

Server-Side: Handles requests to toggle lights on/off. When a ToggleLight request is received, the server changes the light's state and returns the new status.

Client-Side: Sends requests to the server to toggle the light's state and displays the updated status on the GUI.

**BrightnessService Implementation**

Server-Side: Listens for a stream of brightness adjustment requests. As requests are received, the brightness for the specified zones is adjusted, and a confirmation response is sent.

Client-Side: Allows users to adjust brightness through a slider interface. These adjustments are continuously streamed to the server.

**ThermostatService Implementation**

Server-Side: Processes requests to set a specific temperature. The server updates the thermostat settings and confirms the action back to the client.

Client-Side: Provides an interface for users to set the desired temperature, sending this data to the server.

**TemperatureService Implementation**

Server-Side: Periodically measures (random) and provides the current temperature upon request.

Client-Side: Requests current temperature information from the server and displays it on the GUI, updating at regular intervals.

**AlertService Implementation**

Server-Side: Continuously monitors for alerts (random alerts from a list) and streams them to the client.

Client-Side: Receives and displays the stream of alerts in real-time in the GUI.

**SecurityService Implementation**

Server-Side: Receives streams of zone activation requests. Each request is processed to activate or deactivate the specified security zones.

Client-Side: Users can activate or deactivate security zones through the GUI. The state of each zone is streamed to the server for processing.

**SystemStatusService Implementation**

Server-Side: Engages in a bidirectional stream with the client for system status checks. It processes incoming status check requests and responds with the current status.

Client-Side: Initiates and maintains a bidirectional communication channel with the server. Sends status check requests and displays the received responses.

The SystemStatusService was not implemented within the GUI but in the console. It could not be added to the GUI as web-gRPC or websockets would need to be added.

# Naming Services

The naming services in the Smart Home App are designed to be intuitive and descriptive, ensuring that each component's purpose is immediately clear to both developers and end users.

The service names are descriptive and functional, with each service named according to its function e.g., LightService clearly indicates its purpose in controlling lights.

The names are kept as simple as possible, ensuring they are free from technical jargon.

# Remote Error Handling & Advanced Features

# The system handles remote errors. It detects failures (e.g., network issues), logs them, and communicates these errors back to the client html.

# Examples of the error handling, it the server is disconnected, the set temperature, activate/deactivate security, and lighting brightness adjustment return an error message to the end user.

# 

# Client - Graphical User Interface (GUI)

The Graphical User Interface is developed with HTML, CSS and JavaScript. Bootstrap was used to give a responsive HTML page that will adjust on various screen sizes.

The HTML page consists of a container for the image and header text, and three Bootstrap cards for Lighting, Thermostat and Security control. The switch buttons for security zone activation are from a bootstrap plugin. Node.js web client and ejs is utilised to interact with the server.

# 

# GitHub Integration

The GitHub file layout is as follows. The smart-home-app folder contains the final code.

Other files contained within the GitHub folder are initial GUI, revised GUI and project documentation.

The smart-home-app was built in stages and there are regular commits.

# 

# Appendix

## GitHub link

All files used in the project can be found at the following GitHub.

<https://github.com/bmac01/Distributed-Systems-Project>

Folder smart-home-app contains the working code.

## 8.2 Smart Home App Code

**Proto file code**

syntax = "proto3";

package smarthomecontrol;

// Service definitions for Smart Home Control System

//LightService to handle the light toggle controls

//Simple RPC

service LightService {

rpc ToggleLight (ToggleRequest) returns (ToggleResponse);

rpc GetStatus (Empty) returns (LightStatus);

}

//Client Side rpc streaming to control brightness setting

service BrightnessService {

rpc AdjustBrightness (stream BrightnessRequest) returns (BrightnessResponse);

}

//Thermostat Service

//Simple rpc to set temperature

service ThermostatService {

rpc SetTemperature (TemperatureRequest) returns (TemperatureResponse);

}

//Simple rpc to get current temperature

service TemperatureService {

rpc GetCurrentTemperature (Empty) returns (CurrentTemperatureResponse);

}

//Security System

//AlertService - server side rpc to provide alerts to the client

service AlertService {

rpc GetAlerts (Empty) returns (stream Alert);

}

//SecurityService - client side rpc to activate zones

service SecurityService {

rpc ActivateZones (stream ZoneActivationRequest) returns (ActivationResponse);

}

// Message definitions

message Empty {}

message Light {

int32 light\_id = 1;

float brightness = 2;

bool power\_status = 3;

}

message LightStatus {

repeated Light lights = 1;

}

message ToggleRequest {

bool status = 1;

}

message ToggleResponse {

bool result = 1;

string message = 2;

}

message Alert {

string message = 1;

string timestamp = 2;

}

message ZoneActivationRequest {

int32 zone\_id = 1;

bool activate = 2;

}

message ActivationResponse {

string message = 1;

}

message TemperatureRequest {

float temperature = 1;

}

message TemperatureResponse {

string message = 1;

}

message BrightnessRequest {

int32 zone\_id = 1;

int32 brightness = 2;

}

message BrightnessResponse {

string message = 1;

}

message CurrentTemperatureResponse {

float temperature = 1;

}

//Bidirectional rpc - to check system status

service SystemStatusService {

rpc CheckStatus (stream StatusCheckRequest) returns (stream StatusCheckResponse);

}

//Message definitions

message StatusCheckRequest {

string checkCommand = 1;

}

message StatusCheckResponse {

string statusMessage = 1;

}

**server.js code**

//import gRPC and proto loader modules

var grpc = require('@grpc/grpc-js');

var protoLoader = require('@grpc/proto-loader');

//Load gRPC Service Definitions

var packageDef = protoLoader.loadSync('smarthome.proto', {});

var grpcObject = grpc.loadPackageDefinition(packageDef);

var smarthomecontrolPackage = grpcObject.smarthomecontrol;

//Create gRPC Server Instance and Initialize Light Status:

var server = new grpc.Server();

let lightStatus = false; // needs to be updated so that it is random ?? - leave as false for now, update later

//Add light service to gRPC Server:

server.addService(smarthomecontrolPackage.LightService.service, {

    ToggleLight: (call, callback) => {

        lightStatus = call.request.status;

        callback(null, {

            result: lightStatus,

            message: `Lights are now ${lightStatus ? 'ON' : 'OFF'}`

        });

    }

});

//Add Security Service to gRPC Server:

server.addService(smarthomecontrolPackage.SecurityService.service, {

    ActivateZones: (call, callback) => {

        call.on('data', (zoneActivationRequest) => {

            console.log(`Zone ${zoneActivationRequest.zone\_id} activation status: ${zoneActivationRequest.activate}`);

            // Process zone activation

        });

        call.on('end', () => {

            callback(null, { message: "Zones activation updated" });

        });

    },

});

//Add Alert Service to gRPC Server:

server.addService(smarthomecontrolPackage.AlertService.service, {

    GetAlerts: (call) => {

        var alerts = ["Intruder detected", "Window opened", "Motion detected", "Door unlocked"];

        let count = 0;

        var maxAlerts = 3; // Set the maximum number of alerts

        var intervalId = setInterval(() => {

            if (count >= maxAlerts) {

                clearInterval(intervalId); // Stop the interval

                call.end(); // End the call

                return; // Exit the function to prevent further execution

            }

            var alertIndex = Math.floor(Math.random() \* alerts.length);

            call.write({

                message: alerts[alertIndex],

                timestamp: new Date().toISOString()

            });

            count++; // Increment the count

        }, 500); // Sending a random alert every .5 seconds

    }

});

//Add Thermostat Service to gRPC Server

server.addService(smarthomecontrolPackage.ThermostatService.service, {

    SetTemperature: (call, callback) => {

        var temperature = call.request.temperature;

        // Logic to set the temperature

        callback(null, { message: `Temperature set to ${temperature}°C` });

    },

});

//Add Brightness Service to gRPC Server:

server.addService(smarthomecontrolPackage.BrightnessService.service, {

    AdjustBrightness: (call, callback) => {

        call.on('data', (request) => {

            console.log(`Zone ${request.zone\_id} brightness: ${request.brightness}`);

            // Logic to adjust the brightness

        });

        call.on('end', () => {

            callback(null, { message: "Brightness adjusted" });

        });

    },

});

// Function to generate a random temperature

function getRandomTemperature() {

    return Math.random() \* (25-18) + 18; // Random temperature between 18°C and 25°C

  }

//Add Temperature Service to gRPC Server:

server.addService(smarthomecontrolPackage.TemperatureService.service, {

    GetCurrentTemperature: (\_, callback) => {

      callback(null, { temperature: getRandomTemperature() });

}});

//Add System Status Service to gRPC Server:

server.addService(smarthomecontrolPackage.SystemStatusService.service, {

    CheckStatus: (call) => {

      call.on('data', (request) => {

        console.log(`Received check command: ${request.checkCommand}`);

        // Implement your logic to check system status

        // For example, respond with a status message

        let statusMessage = `Status checked for command: ${request.checkCommand}`;

        call.write({ statusMessage: statusMessage });

      });

      call.on('end', () => {

        call.end();

      });

    }

  });

//Start the gRPC Server

server.bindAsync('0.0.0.0:4000', grpc.ServerCredentials.createInsecure(), (error, port) => {

    if (error) {

        console.error(error);

        return;

    }

    server.start();

    console.log(`gRPC server running on port ${port}`);

});

**Client App js code**

// Importing required modules

var express = require('express');

var grpc = require('@grpc/grpc-js');

var protoLoader = require('@grpc/proto-loader');

// Initializing Express app

var app = express();

app.set('view engine', 'ejs');

app.use(express.static('public'));

app.use(express.json()); // for parsing application/json

// Load the gPRC package

var packageDef = protoLoader.loadSync('smarthome.proto', {});

var grpcObject = grpc.loadPackageDefinition(packageDef);

var smarthomecontrolPackage = grpcObject.smarthomecontrol;

// Initialize gRPC for LightService

var lightClient = new smarthomecontrolPackage.LightService('localhost:4000', grpc.credentials.createInsecure());

// Initialize gRPC for AlertService

var alertStatusClient = new smarthomecontrolPackage.AlertService('localhost:4000', grpc.credentials.createInsecure());

// Initialize gRPC for SecurityService

var securityClient = new smarthomecontrolPackage.SecurityService('localhost:4000', grpc.credentials.createInsecure());

// Initialize gRPC for ThermostatService

var thermostatClient = new smarthomecontrolPackage.ThermostatService('localhost:4000', grpc.credentials.createInsecure());

// Initialize gRPC for BrightnessService

var brightnessclient = new smarthomecontrolPackage.BrightnessService('localhost:4000', grpc.credentials.createInsecure());

// Initialize gRPC for TemperatureService

var temperatureClient = new smarthomecontrolPackage.TemperatureService('localhost:4000', grpc.credentials.createInsecure());

//Initialize gRPC for SystemStatus

var client = new smarthomecontrolPackage.SystemStatusService('localhost:4000', grpc.credentials.createInsecure());

// Route to render the home page

app.get('/', (req, res) => {

    res.render('index', { lightStatus: false });

});

// Route to toggle light status

app.get('/toggle', (req, res) => {

    var lightStatus = req.query.status === 'true';

    lightClient.ToggleLight({ status: lightStatus }, (error, response) => {

        if (!error) {

            res.json(response);

        } else {

            res.status(500).json({ message: "Error toggling light" });

        }

    });

});

// Route to fetch security alerts

app.get('/alerts', (req, res) => {

    var call = alertStatusClient.GetAlerts({});

    let alerts = [];

    call.on('data', (alert) => {

        alerts.push(alert);

    });

    call.on('end', () => {

        // Stream ended by the server

        res.json(alerts); // Send the collected alerts to the client

    });

    call.on('error', (error) => {

        console.error('Error on GetAlerts stream:', error);

        if (!res.headersSent) {

            res.status(500).json({ message: "Error fetching alerts" });

        }

    });

});

// Route to activate security zones

// Assume a default state for each zone

let zonesState = {

    1: false, // Zone 1

    2: false, // Zone 2

    3: false  // Zone 3

};

app.post('/activate-zones', (req, res) => {

    const zones = req.body.zones;

    zones.forEach(zone => {

        // Update the state of each zone

        zonesState[zone.zone\_id] = zone.activate;

    });

    // response indicating which zones are active and which are not

    let activeZones = [];

    let inactiveZones = [];

    for (const [zoneId, isActive] of Object.entries(zonesState)) {

        if (isActive) {

            activeZones.push(zoneId);

        } else {

            inactiveZones.push(zoneId);

        }

    }

    res.json({

        message: "Zones updated",

        activeZones: activeZones.join(', '),

        inactiveZones: inactiveZones.join(', ')

    });

});

//Route to set thermostat temperature

app.post('/set-temperature', (req, res) => {

    var { temperature } = req.body;

        thermostatClient.SetTemperature({ temperature }, (error, response) => {

        if (!error) {

            res.json(response);

        } else {

            res.status(500).send('Error setting temperature');

        }

    });

});

//Route to adjust brightness

app.post('/adjust-brightness', (req, res) => {

    const { zone\_id, brightness } = req.body;

    let call = brightnessclient.AdjustBrightness((error, response) => {

        if (!error) {

            res.json(response);

        } else {

            res.status(500).send('Error adjusting brightness');

        }

    });

    call.write({ zone\_id, brightness });

    call.end();

});

//Route to get current temperature

app.get('/current-temperature', (req, res) => {

  temperatureClient.GetCurrentTemperature({}, (error, response) => {

    if (!error) {

      res.json({ temperature: response.temperature });

    } else {

      res.status(500).json({ message: "Error fetching current temperature" });

    }

  });

});

// Creating and handling System Status check stream

const statusCheckCall = client.checkStatus();

statusCheckCall.on('data', (response) => {

  console.log('Received from server:', response.statusMessage);

});

statusCheckCall.on('end', () => {

  console.log('Server has completed sending messages');

});

// Send a check command

statusCheckCall.write({ checkCommand: "initial check" });

// Send another check command after 5 seconds

setTimeout(() => {

  statusCheckCall.write({ checkCommand: "follow-up check" });

}, 5000);

// End the call after 10 seconds

setTimeout(() => {

  statusCheckCall.end();

}, 10000);

module.exports = app; // Export the app instance for use in www

**index.ejs code**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Smart Home Control Panel</title>

    <!-- Bootstrap CSS -->

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

    <!-- Bootstrap CSS for Swicth buttons -->

    <link href="https://cdn.jsdelivr.net/gh/gitbrent/bootstrap-switch-button@1.1.0/css/bootstrap-switch-button.min.css" rel="stylesheet">

    <link rel="stylesheet" href="/stylesheets/style.css">

</head>

<body>

    <!-- Main Container with Image and Heading -->

    <div class="container my-5">

        <div class="text-center mb-5">

            <h1>Smart Home Control Panel</h1>

            <img src="/images/image1.jpeg" alt="Home Image" class="img-fluid rounded mb-4" style="max-width: 400px;">

            <p class="lead">Manage your home's lighting, temperature, and security system</p>

        </div>

        <div class="row g-4">

 <!-- Lighting Control Control Card -->

    <div class="col-md-4">

        <div class="card shadow h-100">

            <div class="card-header bg-primary text-white">Lighting Control</div>

            <div class="card-body">

                <button id="lightToggle" class="btn btn-lg w-100" onclick="toggleLight()">Turn On/Off</button>

                <h6 id="LightStatus">Lights: Off</h6><br>

                <h6>Adjust Brightness</h6>

                <div class="row align-items-center">

                    <div class="col">

                        <input type="range" class="form-range" id="brightnessSlider1" min="0" max="10" onchange="adjustBrightness(1, this.value)">

                    </div>

                    <div class="col-auto">

                        <label for="brightnessSlider1" class="form-label">Zone 1</label>

                    </div>

                </div>

                <h6 id="brightnessLevel1">Brightness: 5</h6>

                <div class="row align-items-center">

                    <div class="col">

                        <input type="range" class="form-range" id="brightnessSlider2" min="0" max="10" onchange="adjustBrightness(2, this.value)">

                    </div>

                    <div class="col-auto">

                        <label for="brightnessSlider2" class="form-label">Zone 2</label>

                    </div>

                </div>

                <h6 id="brightnessLevel2">Brightness: 5</h6>

                <p id="brightnessStatus" class="text-danger fw-bold" style="display: none;">Brightness</p>

            </div>

        </div>

    </div>

    <!-- Thermostat Control Card -->

    <div class="col-md-4">

        <div class="card shadow h-100">

            <div class="card-header bg-success text-white">Thermostat Control</div>

            <div class="card-body">

                <h5 class="current-temperature mb-3">Current Temperature: <span id="currentTemp">--</span>°C</h5>

                <input type="number" id="temperature" class="form-control mb-2" name="temperature" min="15" max="30" placeholder="--°C">

                <button class="btn btn-success w-100" onclick="setTemperature()">Set Temperature</button>

                <p id="temperatureStatus" class="text-danger fw-bold">Temperature set to</p>

            </div>

        </div>

    </div>

    <!-- Security Control Card -->

    <div class="col-md-4">

        <div class="card shadow h-100">

            <div class="card-header bg-danger text-white">Security Control</div>

            <div class="card-body">

                <button id="securityToggle" class="btn btn-lg w-100 mb-3" onclick="toggleSecuritySystem()">Activate/Deactivate</button>

                <input type="checkbox" id="zone1" data-toggle="switchbutton" checked data-onstyle="primary" data-size="sm">

                <label for="zone1" class="form-check-label">Zone 1</label>

                <input type="checkbox" id="zone2" data-toggle="switchbutton" checked data-onstyle="primary" data-size="sm">

                <label for="zone2" class="form-check-label">Zone 2</label>

                <input type="checkbox" id="zone3" data-toggle="switchbutton" checked data-onstyle="primary" data-size="sm">

                <label for="zone3" class="form-check-label">Zone 3</label>

                <h6>Select Zones and Activate alarm</h6>

                <p id="securityStatus" class="text-danger fw-bold">Current Status: Deactivated</p>

                <h6>Recent Activity</h6>

                <ul id="recentActivity" class="list-unstyled">

                    <!-- Recent activities will be listed here -->

                </ul>

            </div>

        </div>

    </div>

        </div>

    </div>

    <!-- Bootstrap Bundle with Popper -->

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>

    <!-- Bootstrap Custom Switch Buttons for Security Activation -->

    <script src="https://cdn.jsdelivr.net/gh/gitbrent/bootstrap-switch-button@1.1.0/dist/bootstrap-switch-button.min.js"></script>

    <script src="/javascripts/script.js"></script>

</body>

</html>

**Html script code**

//keep track o light status

let lightIsOn = false;

//Function to toggle light status

async function toggleLight() {

    const response = await fetch(`/toggle?status=${!lightIsOn}`);

    const data = await response.json();

    lightIsOn = data.result;

    document.getElementById('LightStatus').innerText = data.message;

}

// Function to fetch and display alerts

async function fetchAlerts() {

    try {

        const response = await fetch('/alerts');

        if (!response.ok) {

            throw new Error('Network response was not ok');

        }

        const alerts = await response.json();

        const ul = document.getElementById('recentActivity');

        ul.innerHTML = ''; // Clear existing alerts

        alerts.forEach(alert => {

            const li = document.createElement('li');

            li.textContent = `${alert.message} at ${alert.timestamp}`;

            ul.appendChild(li);

        });

    } catch (error) {

        console.error('Error fetching alerts:', error);

    }

}

// alerts every 3 seconds

setInterval(fetchAlerts, 3000);

// Function to toggle security system state

async function toggleSecuritySystem() {

    // Gather state of each zone from the Web GUI

    const zones = [

        { zone\_id: 1, activate: document.getElementById('zone1').checked },

        { zone\_id: 2, activate: document.getElementById('zone2').checked },

        { zone\_id: 3, activate: document.getElementById('zone3').checked }

    ];

    try {

        const response = await fetch('/activate-zones', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({ zones }),

        });

        if (!response.ok) {

            throw new Error('Network response was not ok');

        }

        const data = await response.json();

        document.getElementById('securityStatus').innerText = `Current Status: ${data.message}. Active Zones: ${data.activeZones}. Inactive Zones: ${data.inactiveZones}.`;

    } catch (error) {

        console.error('Error activating zones:', error);

        document.getElementById('securityStatus').innerText = `Error: ${error.message}`;

    }

}

// Function to set the temperature

async function setTemperature() {

    var temperature = document.getElementById('temperature').value;

    try {

        var response = await fetch('/set-temperature', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({ temperature: parseFloat(temperature) }),

        });

        if (!response.ok) {

            throw new Error('Network response was not ok');

        }

        var data = await response.json();

        document.getElementById('temperatureStatus').innerText = data.message;

    } catch (error) {

        console.error('Error setting temperature:', error);

        document.getElementById('temperatureStatus').innerText = `Error: ${error.message}`;

    }

}

// Function to adjust brightness

async function adjustBrightness(zoneId, brightness) {

    try {

        var response = await fetch('/adjust-brightness', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({ zone\_id: zoneId, brightness: parseInt(brightness) }),

        });

        if (!response.ok) {

            throw new Error('Network response was not ok');

        }

        var data = await response.json();

        document.getElementById(`brightnessLevel${zoneId}`).innerText = `Brightness: ${brightness}`;

        // Update and show the status message

        var statusElement = document.getElementById('brightnessStatus');

        statusElement.innerText = data.message;

        statusElement.style.display = 'block'; // Make it visible

    } catch (error) {

        var statusElement = document.getElementById('brightnessStatus');

        statusElement.innerText = `Error: ${error.message}`;

        statusElement.style.display = 'block'; // Make it visible

    }

}

//  Event listener setup for DOMContentLoaded event for the brightness sliders

document.addEventListener('DOMContentLoaded', (event) => {

    var slider1 = document.getElementById('brightnessSlider1');

    var slider2 = document.getElementById('brightnessSlider2');

 // Add change event listeners to brightness sliders

    slider1.addEventListener('change', () => adjustBrightness(1, slider1.value));

    slider2.addEventListener('change', () => adjustBrightness(2, slider2.value));

});

document.addEventListener('DOMContentLoaded', () => {

    // Set up other event listeners...

});

// Function to fetch current temperature

async function fetchCurrentTemperature() {

    try {

      const response = await fetch('/current-temperature');

      if (!response.ok) {

        throw new Error('Network response was not ok');

      }

      const data = await response.json();

      document.getElementById('currentTemp').innerText = data.temperature.toFixed(0);

    } catch (error) {

      console.error('Error fetching current temperature:', error);

    }

  }

  // Fetch the temperature immediately when the page loads

  fetchCurrentTemperature();

  // Then update the temperature every 5 seconds

setInterval(fetchCurrentTemperature, 5000);