1. **Module Imports (Lines 1-5):**

Import required modules including express, http, path, socket.io, and cors.

These modules are helpful for setting up the node environment, server setup and creation of websocket connection.

const express = require('express');

const http = require('http');

const path = require('path');

const { Server } = require('socket.io');

const cors = require('cors');

1. **Middleware Setup (Lines 8-13):**

In the below lines, we have middleware that acts kind of like a middleman. To be more precise it helps to pass through the initial layer first before entering the main logic or hitting the main API. Middlewares are helpful for a number of purposes like authentication, authorization or IP restriction etc. For this application we have used it to ensure basic security features like making the public folder as static so that users can only be restricted to index.html file and the stylesheet file via the GET API.

app.use(express.static(path.resolve('./public')));

// Serve static files from the current directory

app.use(express.static(\_\_dirname));

// Set specific headers for JavaScript files

app.use(express.static(\_\_dirname, {

setHeaders: (res, path, stat) => {

if (path.endsWith('.js')) {

res.set('Content-Type', 'text/javascript');

}

},

}));

//Explanation of the Code used

Create an Express application (app).

Serve static files from the 'public' folder.

Serve static files from the current directory (\_\_dirname).

Enable CORS middleware.

1. **HTTP Server and Socket.IO (Lines 15-21):**

On line 15 to 21 http server has been created using express and after that an object/instance of socket.io is created and stored in the constant named io.

const server = http.createServer(app);

// Create a new instance of Socket.IO and attach it to the HTTP server

const io = new Server(server);

//Explanation of code

Create an HTTP server using the Express application.

Create a new instance of Socket.IO (io) and attach it to the HTTP server.

1. **Initial Bulb States (Line 31):**

After that an initial array is created with initial values set as false. These values define the current state of the bulb or IOT Device and get updated later on click events that get received from the client side.

const initialBulbStates = [false, false, false, false];

//Algorithmic explanation

Initialise an array initialBulbStates to store the current states of the bulbs. Set initial values to false.

1. **Socket.IO Connection Handling (Lines 33-54):**

On connection with the websocket the server will emit/broadcast the code to all clients on receiving the trigger event from any particular client. This function is used to send updated date of initialBulbstates and console logging of received toggle events and broadcasting of events to all clients.

io.on('connection', (socket) => {

console.log('A user connected');

// Send initial bulb states to the new client

socket.emit('initialBulbStates', initialBulbStates);

// Handle 'toggle' event from clients

socket.on('toggle', (data) => {

console.log(`Bulb ${data.index} toggled to ${data.state}`);

// Update the initial bulb states array

initialBulbStates[data.index - 1] = data.state === 'on';

// Broadcast the 'toggle' event to all connected clients

io.emit('toggle', data);

});

//Algorithmic explanation

Handle the 'connection' event when a client connects.

Emit the initial bulb states to the newly connected client.

Handle the 'toggle' event from clients, update the initial states, and broadcast the event to all clients.

1. **Socket on Disconnect (Lines 56-59):**

This websocket’s builtin function will handle client’s connection being terminated and console logging of it.

socket.on('disconnect', () => {

console.log('User disconnected');

});

});

1. **Root Path Route (Lines 56-59):**

Here in this section a simple Restful API is used to handle GET requests. In response, a path to index.html would be sent.

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

// Send the path to index.html as the response

res.send("/index.html");

});

//Code explanation

Define a route for the root path ("/") that sends the path to 'index.html' as the response.

1. **Server Listening (Lines 61-64):**

At last Port constant has been defined with the value of 5500 which can be changed as per requirements. After that, the server would start listening to this port. A detailed explanation is written as follows.

server.listen(PORT, () => {

console.log(`Server is up and running on http://localhost:${PORT}`);

});

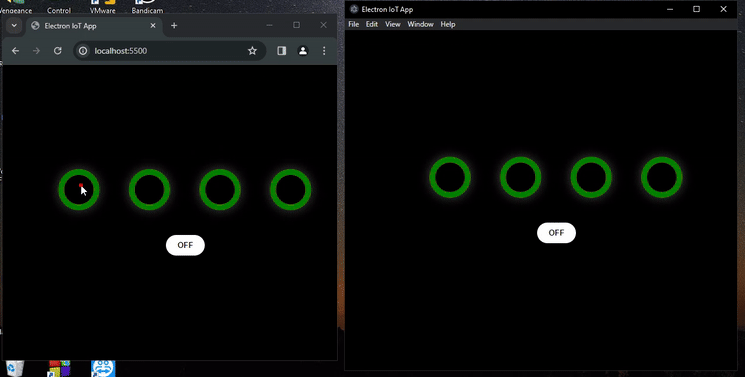
//Algorithmic Explanation

Define the port (PORT) for the server to listen on (5500).

Start the server and log a message indicating the server is up and running.

This server code sets up a basic Express server, serves static files, initialises WebSocket connections using Socket.IO, and handles events such as client connection, disconnection, and bulb state toggling. The server listens on port 5500.

**Application Preview**



Here’s the working of the application. Each trigger gets updated in real time across all devices with the help of underlying technology called Websockets which is a bi-directional communication protocol. On click events the event trigger message would be sent to the other client IOT Devices whether it be a smart bulb or a smart fan. Not only would it turn on or off the device it’ll also synchronously update the state of the buttons across all other client members using the app for better and consistent communication among all devices.