Mini Project Report for 18ECO109J(Embedded System Design Raspberry Pi)

Title	Auto-discovery of smart home devices
Batch-mates	None (solo project)
Register Number	RA1911030010014
Name	Gita Alekhya Paul
Faculty In-charge	Vijayakumar P

Contents

Background and Motivation:

How does an Alexa smart speaker automatically detect newly connected smart home devices connected to the same network? This is the question which has been a key motivation for this mini project.

According to a survey conducted in 2019, approximately 37 per cent of Indian respondents stated that they owned a smart speaker. In 2020, Amazon smart speakers had a market share of 79 per cent in India. Google followed it, a distant second at 11 per cent. Over a million smart speakers were shipped in India in that year, Echo Dot (3rd Gen) and Echo Dot (4th Gen) were the top two smart speakers. In 2017, Amazon was the first company to launch smart speakers in the country.

As per the statistics shown, we can clearly see that India's smart home devices market has been booming since its inception. But this technology is not understood by most of its users. It seems like magic to most of its consumers. From the auto-discovery of devices to the efficient pairing with other smart home devices like a smart LED bulb, most of this technology is proprietary code abstracted from the end-user.

The mini-project aims to demystify this technology behind the smart home devices like the Phillips Hue, Alexa and Google Home ecosystems. The project aims at demonstrating the working of the SSDP protocol, its uses and the working of UPnP protocol service descriptions. The demonstration also includes an example of how users can deploy this ecosystem to a Raspberry Pi and how smart LED bulbs can be created using NodeMCUs.

Key Problems Solved:

Although the mini project started out as a study experiment, I have managed to build a smart home device ecosystem that solves some of the following key problems:

- Created an open-source smart home device management ecosystem. These technologies and their implementations are often proprietary in nature, hidden from the consumers by the companies who manufacture them. This project serves as a transparency report to explain and demystify the smart home device technology.
- The project also serves as a study project for studying multiple web and IoT protocols used to implement smart home devices. Protocols like SSDP, UPnP and HTTP APIs have been used in this project with documentation on their working and implementation code.
- Anyone can easily deploy the smart home system as a software package on a Raspberry Pi. The smart LED bulbs showcased in this demonstration can be easily made by using NodeMCUs and some LEDs.

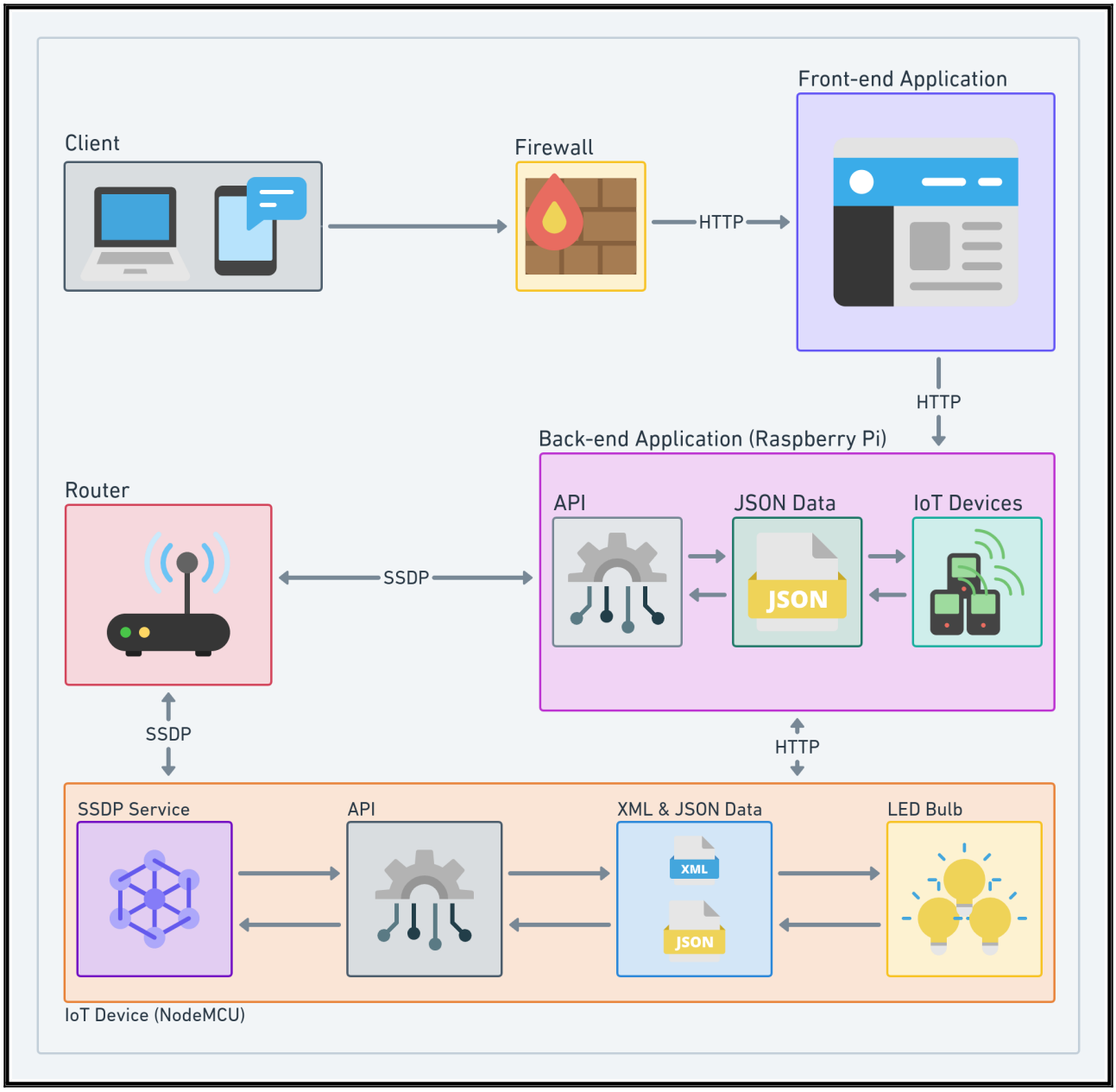
Objectives:

The project uses SSDP as a medium for recognising new devices connected to the network and getting their device descriptions. The Simple Service Discovery Protocol (SSDP) protocol is used by the Universal Plug and Play (UPnP) protocol. The protocol provides broadcast and discovery of network services on a local network. A central Node.js server deployed to a Raspberry Pi serves as

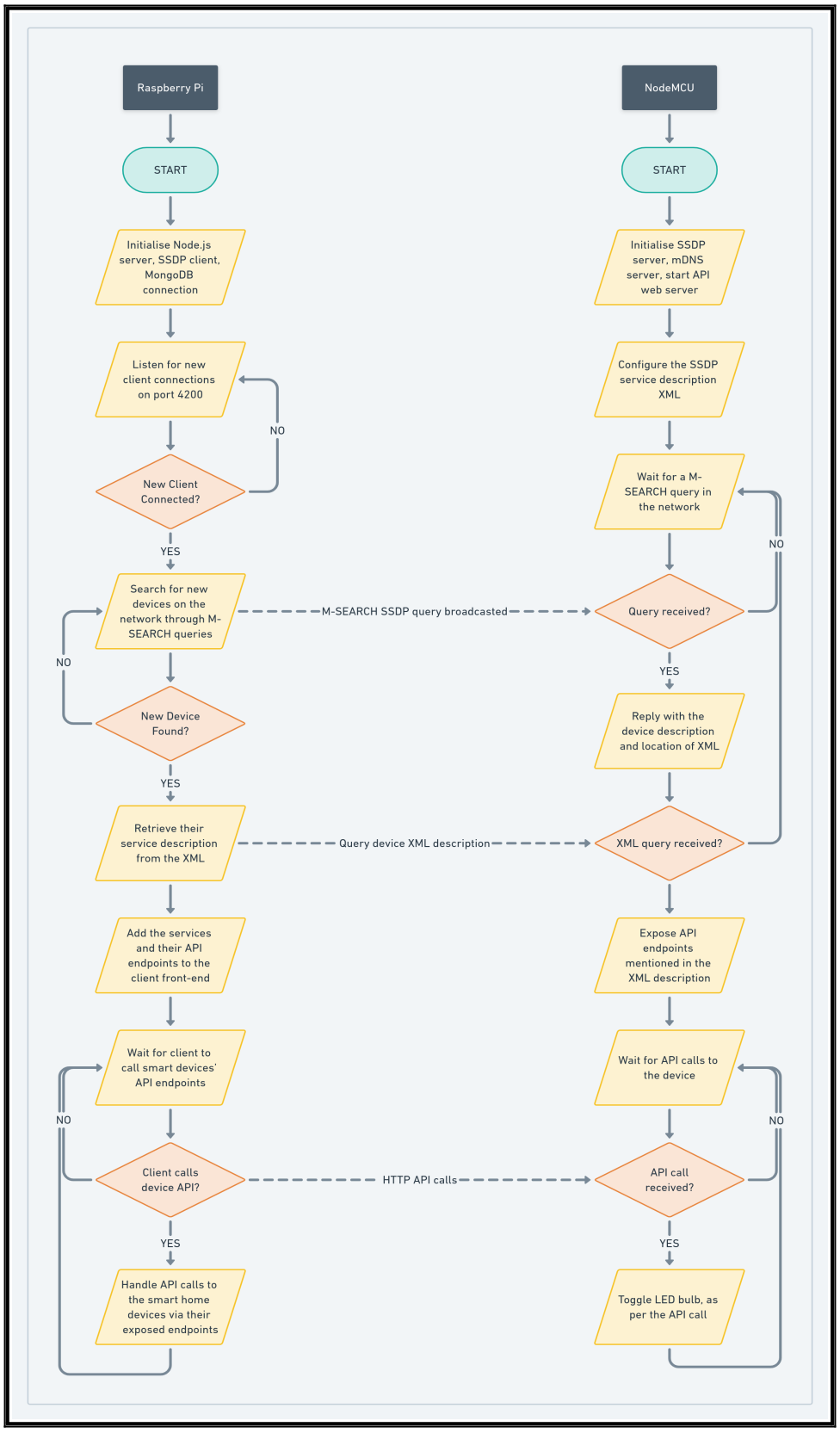
an SSDP client that notes every device connected to the network. Once a new device is detected, the server reads the UPnP protocol service description XML, and the exposed API endpoints are conveyed to the front-end client. The front-end client can then call these specific APIs to change the state of the connected smart home devices.

System Model / Block Diagram:

System Model Diagram:



System Block Diagram:



Methodology / Algorithm / Program:

Hardware Requirement:

- NodeMCU ESP8266
- LEDs
- 330 Ω Resistors
- Breadboard
- Jumper Wires
- Raspberry Pi

Software Requirement:

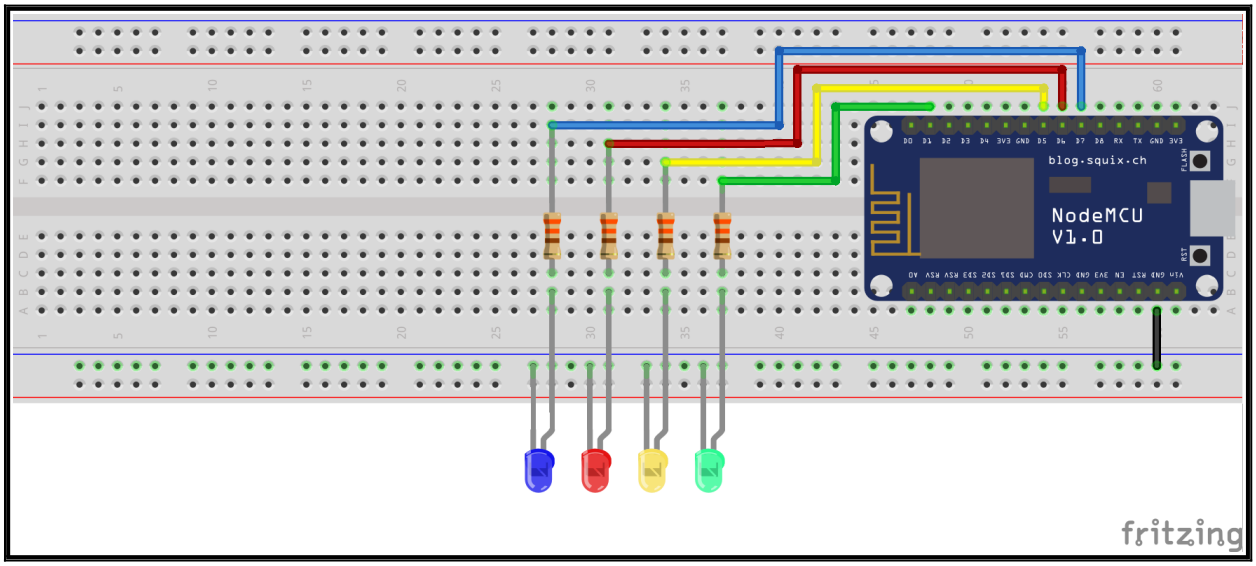
- Node.js 14.15.4 LTS
- Yarn 1.22.10

Application Code:

File Structure:

```
├── api
│   ├── app.ts
│   ├── auth
│   │   ├── auth.routes.ts
│   │   ├── auth.schema.ts
│   │   └── auth.service.ts
│   ├── device
│   │   ├── device.routes.ts
│   │   ├── device.schema.ts
│   │   └── device.service.ts
│   ├── error
│   │   ├── error.constants.ts
│   │   └── error.handler.ts
│   ├── middlewares
│   │   ├── validate-jwt.ts
│   │   └── validate-query.ts
│   └── services
│       ├── database.service.ts
│       ├── socket.service.ts
│       └── ssdp.service.ts
├── nodemcu
│   └── main.ino
├── public
│   ├── css
│   │   └── style.css
│   └── js
│       ├── dashboard.js
│       └── index.js
├── views
│   ├── dashboard.ejs
│   └── index.ejs
```

Circuit Diagram:



Programs:

/api/app.ts :

```
import { config } from "dotenv";
import express, { Express, Request, Response, NextFunction } from "express";
import cors from "cors";
import { join } from "path";
import { DatabaseService } from "./services/database.service";
import { SSDPService } from "./services/ssdp.service";
import { SocketService } from "./services/socket.service";
import authRoutes from "./auth/auth.routes";
import deviceRoutes from "./device/device.routes";
import { describeDevice } from "./device/device.service";
import { errorHandler } from "./error/error.handler";
import { Server } from "http";
import { Socket } from "socket.io";

const app: Express = express();
config();

app.set("view engine", "ejs");
app.set("views", join(__dirname, "..", "views"));

app.use(express.json());
app.use(cors());

app.use("/api/v1/auth", authRoutes);
app.use("/api/v1/device", deviceRoutes);

if (process.env.NODE_ENV === "production") {
  app.use(express.static(join(__dirname, "..", "public")));
  app.get("/", (req: Request, res: Response, next: NextFunction) => {
```

```
    res.render("index", {});
  });
  app.get("/dashboard", (req: Request, res: Response, next: NextFunction) => {
    res.render("dashboard", {});
  });
}

app.use((req: Request, res: Response, next: NextFunction) => {
  res.status(404).json({
    success: false,
    message: `Cannot ${req.method} ${req.url}`,
  });
});
app.use(errorHandler);

Promise.all([
  DatabaseService.getInstance().initalize(),
  SSDPService.getInstance().initalize(),
])
  .then(() => {
    return app.listen(process.env.PORT, () => {
      console.log(
        `Express:${process.env.NODE_ENV} listening on Port ${process.env.PORT}`
      );
    });
  })
  .then((expressServer: Server) => {
    return SocketService.getInstance().initalize(expressServer);
  })
  .then(async (socketServer) => {
    socketServer.on("connection", (socket: Socket) => {
      console.log(`${socket.id} has connected.`);
    });
    const client = await SSDPService.getInstance().getClient();
    client!.on("response", async (headers, statusCode, rinfo) => {
      console.dir(headers);
      console.dir(statusCode);
      console.dir(rinfo);
      const deviceDescription = await describeDevice(
        headers.ST!,
        headers.USN!,
        headers.LOCATION!
      );
      socketServer.sockets.emit("new-device", deviceDescription);
    });
  })
  .catch((err) => {
    console.error("%o", err);
    process.exit(1);
  });
```

/api/auth/auth.routes.ts :

```
import { Router, Request, Response, NextFunction } from "express";
const router: Router = Router();
import validateQuery from "../middlewares/validate-query";
import { user, userSchema } from "./auth.schema";
import { signup, login } from "./auth.service";

const handlePostSignup = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { email, password } = req.body as user;
    const result = await signup(email, password);
    res.status(201).json({
      success: true,
      message: "User successfully signed-up!",
    });
  } catch (err) {
    next(err);
  }
};

const handlePostLogin = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { email, password } = req.body as user;
    const authToken = await login(email, password);
    res.json({
      success: true,
      authToken,
    });
  } catch (err) {
    next(err);
  }
};

router.post("/signup", validateQuery("body", userSchema), handlePostSignup);
router.post("/login", validateQuery("body", userSchema), handlePostLogin);

export default router;
```

/api/auth/auth.schema.ts :

```
import * as yup from "yup";

export const userSchema = yup
  .object({
    email: yup
      .string()
      .trim()
      .min(1, "email cannot be null")
      .email("not a valid email")
      .required(),
    password: yup.string().trim().min(1, "password cannot be null").required(),
  })
  .required();

export type user = yup.InferType<typeof userSchema>;
```

/api/auth/auth.service.ts :

```
import { DatabaseService } from "../services/database.service";
import { errors } from "../error/error.constants";
import { hash, compare } from "bcrypt";
import { user } from "./auth.schema";
import { sign } from "jsonwebtoken";

export const signup = async (
  email: string,
  password: string
): Promise<boolean | undefined> => {
  const db = await DatabaseService.getInstance().getDb("home-automation");
  const exists = await db.countDocuments({ email: email });
  if (exists) throw errors.ACCOUNT_EXISTS;
  const hashedPassword = await hash(password, 12);
  const { result } = await db.insertOne({
    email: email,
    password: hashedPassword,
    devices: [],
  });
  if (result.ok) return true;
};

export const login = async (
  email: string,
  password: string
): Promise<string | undefined> => {
  const db = await DatabaseService.getInstance().getDb("home-automation");
  const exists = await db.countDocuments({ email: email });
  if (!exists) throw errors.WRONG_CREDS;
  const user = await db.findOne<user>({ email: email });
  const matchPassword = await compare(password, user!.password);
  if (!matchPassword) throw errors.WRONG_CREDS;
  const authToken = await sign({ email: email }, process.env.JWT_SECRET!, {
```

```
    issuer: "gitaalekhyapaul",
    expiresIn: "2h",
  });
  return authToken;
};
```

/api/device/device.routes.ts :

```
import { Router, Request, Response, NextFunction } from "express";
const router: Router = Router();
import validateQuery from "../middlewares/validate-query";
import { validateJwt } from "../middlewares/validate-jwt";
import { searchDevices, findDevice, addDevice } from "./device.service";
import { deviceSchema, listDeviceSchema } from "./device.schema";
import { errors } from "../error/error.constants";

const handleGetSearch = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    console.log("M-SEARCH QUERY");
    await searchDevices();
    res.json({
      success: true,
    });
  } catch (err) {
    next(err);
  }
};

const handlePostFind = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { link, urn, uuid } = req.body as deviceSchema;
    const exists = await findDevice(res.locals.user.email, uuid, urn);
    res.json({
      success: true,
      exists,
    });
  } catch (err) {
    next(err);
  }
};

const handlePostAdd = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
```

```
  try {
    const { link, urn, uuid } = req.body as deviceSchema;
    const success = await addDevice(res.locals.user.email, uuid, urn);
    if (!success) throw errors.BAD_REQUEST;
    res.json({
      success,
    });
  } catch (err) {
    next(err);
  }
};

router.get("/search", validateJwt(true), handleGetSearch);
router.post(
  "/find",
  validateJwt(true),
  validateQuery("body", listDeviceSchema),
  handlePostFind
);
router.post(
  "/add",
  validateJwt(true),
  validateQuery("body", listDeviceSchema),
  handlePostAdd
);

export default router;
```

/api/device/device.schema.ts :

```
import * as yup from "yup";

export const listDeviceSchema = yup
  .object({
    uuid: yup.string().trim().min(1, "uuid cannot be null").required(),
    urn: yup.string().trim().min(1, "urn cannot be null").required(),
    link: yup.string().trim().min(1, "description cannot be null").required(),
  })
  .required();

export type deviceSchema = yup.InferType<typeof listDeviceSchema>;
```

/api/device/device.service.ts :

```
import { SSDPService } from "../services/ssdp.service";
import { DatabaseService } from "../services/database.service";
import { deviceSchema } from "./device.schema";
import { errors } from "../error/error.constants";
import Axios from "axios";
import { parseStringPromise } from "xml2js";
import { userSchema } from "../auth/auth.schema";

export const searchDevices = async () => {
```

```
  const client = await SSDPService.getInstance().getClient();
  client!.search("urn:schemas-upnp-org:device:ESP8266:1");
};

export const findDevice = async (
  email: string,
  uuid: string,
  urn: string
): Promise<boolean> => {
  const db = await DatabaseService.getInstance().getDb("home-automation");
  const user = await db.findOne<{
    email: string;
    devices: Array<deviceSchema>;
  }>({ email: email });
  const deviceExists = user!.devices.findIndex(
    (p) => p.uuid === uuid && p.urn === urn
  );
  if (deviceExists === -1) return false;
  else return true;
};

export const addDevice = async (
  email: string,
  uuid: string,
  urn: string
): Promise<boolean> => {
  const db = await DatabaseService.getInstance().getDb("home-automation");
  const { result } = await db.updateOne(
    { email: email },
    {
      $push: {
        devices: { uuid, urn },
      },
    }
  );
  if (result.ok) return true;
  else return true;
};

export const describeDevice = async (
  urn: string,
  uuid: string,
  descriptionLink: string
) => {
  try {
    const { data } = await Axios.get(descriptionLink);
    const deviceDescription = await parseStringPromise(data);
    return {
      urn,
      uuid,
      link: deviceDescription.root.URLBase[0],
      name: deviceDescription.root.device[0].friendlyName[0],
      modelNumber: deviceDescription.root.device[0].modelNumber[0],
    };
```

```
  } catch (err) {
    console.dir(err);
  }
};
```

/api/error/error.constants.ts :

```
export const errors = {
  BAD_REQUEST: {
    httpStatus: 400,
    message: "Bad Request.",
  },
  INTERNAL_SERVER_ERROR: {
    httpStatus: 500,
    message: "Internal Server Error.",
  },
  UNAUTHORIZED: {
    httpStatus: 401,
    message: "Unauthorized.",
  },
  NOT_FOUND: {
    httpStatus: 404,
    message: "Resource Not Found.",
  },
  MONGODB_CONNECT_ERROR: {
    httpStatus: 500,
    message: "Could Not Connect to MongoDB.",
  },
  JWT_ERROR: {
    httpStatus: 403,
    message: "JWT Token Not Found.",
  },
  ACCOUNT_EXISTS: {
    httpStatus: 400,
    message: "Account already exists. Please login",
  },
  WRONG_CREDS: {
    httpStatus: 400,
    message: "Wrong email or password.",
  },
  SOCKETIO_CONNECT_ERROR: {
    httpStatus: 500,
    message: "Could Not Connect to Socket.IO.",
  },
};
```

/api/error/error.handler.ts :

```
import { Request, Response, NextFunction } from "express";

export interface ApiError extends Error {
  message: string;
  httpStatus?: number;
```

```
}

export const errorHandler = (
  err: ApiError,
  req: Request,
  res: Response,
  next: NextFunction
) => {
  console.error("%o", err);
  if (err.httpStatus) {
    return res.status(err.httpStatus).json({
      success: false,
      error: err.message,
    });
  }
  res.status(500).json({
    success: false,
    error: "Internal Server Error.",
  });
};
```

/api/middlewares/validate-jwt.ts :

```
import { verify } from "jsonwebtoken";
import * as yup from "yup";
import { Request, Response, NextFunction } from "express";
import { errors } from "../error/error.constants";

export const JwtRequestSchema = yup
  .object({
    authorization: yup
      .string()
      .trim()
      .min(1, "JWT cannot be null")
      .matches(/^Bearer .+$/, "JWT should be Bearer Token"),
  })
  .required();

export interface jwtPayload {
  email: string;
}

type JwtRequest = yup.InferType<typeof JwtRequestSchema>;

/**
 * JWT Validation Middleware
 * @param {boolean} checkRequired Is the JWT check mandatory?
 */
export const validateJwt = (checkRequired: boolean = true) => {
  return async (req: Request, res: Response, next: NextFunction) => {
    try {
      const { authorization } = req.headers as JwtRequest;
      if (!authorization) {
        if (!checkRequired) {
```

```
          return next();
        } else {
          return next(errors.JWT_ERROR);
        }
      }
      const authToken = authorization.split(" ")[1];
      const payload: jwtPayload = verify(authToken, process.env.JWT_SECRET!, {
        issuer: "gitaalekhyapaul",
      }) as jwtPayload;
      res.locals.user = payload;
      next();
    } catch (err) {
      next({
        httpStatus: 403,
        message: `${err.name}: ${err.message}`,
      });
    }
  };
};
```

/api/middlewares/validate-query.ts :

```
import * as yup from "yup";
import { Request, Response, NextFunction } from "express";
type RequestLocations = "query" | "body" | "params" | "headers";

/**
 * Generic Request Validator
 * @param {RequestLocations} location The parameter of the req object to be
validated.
 * @param {yup.ObjectSchema} schema The schema against which validation is to be
done.
 */
const validateQuery = (
  location: RequestLocations,
  schema: yup.AnyObjectSchema
) => {
  return async (req: Request, res: Response, next: NextFunction) => {
    let _location: any;
    switch (location) {
      case "query":
        _location = req.query;
        break;
      case "body":
        _location = req.body;
        break;
      case "params":
        _location = req.params;
        break;
      case "headers":
        _location = req.headers;
        break;
    }
```

```
    try {
      await schema.validate(_location, { abortEarly: false });
      next();
    } catch (error) {
      let message: string = "";
      error.errors.forEach((e: string) => {
        message += `${e}. `;
      });
      next({
        httpStatus: 400,
        message: message,
      });
    }
  };
};

export default validateQuery;
```

/api/services/database.service.ts :

```
import * as MongoDB from "mongodb";
import { errors } from "../error/error.constants";

export class DatabaseService {
  private static instance: DatabaseService;
  private dbClient: MongoDB.MongoClient = new MongoDB.MongoClient(
    process.env.MONGO_URI!,
    {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }
  );
  private constructor() {}

  /**
   * Initializes the database client instance
   * @returns {Promise<void>} Returns a promise which resolves when the database
is successfully connected.
   */
  public initalize = async (): Promise<void> => {
    try {
      await this.dbClient.connect();
      console.info("Connected to MongoDB");
    } catch (err) {
      console.error("%o", err);
      console.error("Could not connect to MongoDB");
      throw errors.MONGODB_CONNECT_ERROR;
    }
  };

  /**
   * Singleton function to get the database instance
   * @returns {DatabaseService} Returns the database instance
   */
```

```
  public static getInstance = (): DatabaseService => {
    if (!DatabaseService.instance) {
      DatabaseService.instance = new DatabaseService();
    }
    return DatabaseService.instance;
  };

  /**
   * Returns the Collection Instance of the given database
   * @param {string} collection The Collection Name
   * @returns {MongoDB.Collection} The instance
   */
  public getDb = async (collection: string): Promise<MongoDB.Collection> => {
    return this.dbClient.db().collection(collection);
  };
}
```

/api/services/socket.service.ts :

```
import { Server } from "http";
import { Server as socketIOServer } from "socket.io";
import { errors } from "../error/error.constants";

export class SocketService {
  private static instance: SocketService;
  private io: socketIOServer | null = null;
  private constructor() {}

  public initalize = async (expressServer: Server): Promise<socketIOServer> => {
    try {
      this.io = new socketIOServer(expressServer);
      console.info(`Connected to Socket.IO on Port ${process.env.PORT}`);
      return this.io;
    } catch (err) {
      console.error("Could not connect to Socket.IO Server");
      console.error("SocketIOError\n%o", { error: err });
      throw errors.SOCKETIO_CONNECT_ERROR;
    }
  };

  public static getInstance = (): SocketService => {
    if (!SocketService.instance) {
      SocketService.instance = new SocketService();
      return SocketService.instance!;
    }
    return SocketService.instance!;
  };
  public getIO = (): socketIOServer => {
    return this.io!;
  };
}
```

/api/services/ssdp.service.ts :

```
import { Client } from "node-ssdp";
import { errors } from "../error/error.constants";

export class SSDPService {
  private static instance: SSDPService;
  private client: Client | null = null;
  private constructor() {}

  public initalize = async (): Promise<void> => {
    try {
      this.client = new Client();
      console.info("Initialized SSDP Client");
    } catch (err) {
      console.error("%o", err);
      console.error("Could not initialize SSDP client");
      throw {
        httpStatus: 500,
        message: "Could not initialize SSDP client",
      };
    }
  };

  public static getInstance = (): SSDPService => {
    if (!SSDPService.instance) {
      SSDPService.instance = new SSDPService();
    }
    return SSDPService.instance;
  };

  public getClient = () => this.client;
}
```

/nodemcu/main.ino :

```
/**
 * @file main.ino
 * @author Gita Alekhya Paul
 * @brief A home automation system
 * @version 1.0.0
 * @date 2021-10-16
 *
 * @copyright Gita Alekhya Paul (c) 2021
 *
 */
//Importing Libraries
#include <ESP8266SSDP.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <WiFiManager.h>
```

```
//Defining pins
#define PORT 80
#define redPin D6
#define yellowPin D5
#define greenPin D1
#define statusLed D7

void wifiConnect();
String getHostname();
void initPins();
void startMDNS();
void handleDescription();
void startWebServer();
void configureSSDP();
void toggleLed();
void handle404NotFound();
void showStatus();

//Init server and wifiManager
ESP8266WebServer server(PORT);
WiFiManager wifiManager;

void setup()
{
    initPins();
    Serial.begin(115200);
    wifiConnect();
    startMDNS();
    startWebServer();
    configureSSDP();
    showStatus();
}

void loop()
{
    MDNS.update();
    server.handleClient();
}

void showStatus()
{
    digitalWrite(statusLed, HIGH);
    delay(250);
    digitalWrite(statusLed, LOW);
    delay(250);
    digitalWrite(statusLed, HIGH);
    delay(250);
    digitalWrite(statusLed, LOW);
    delay(250);
    digitalWrite(statusLed, HIGH);
}

void handle404NotFound()
{
```

```
    Serial.println("API_ERROR:: 404 RESOURCE NOT FOUND");
    DynamicJsonDocument res(1024);
    res["success"] = false;
    res["message"] = "Resource Not Found";
    char response[1024];
    serializeJson(res, response);
    server.send(404, "application/json", response);
}

void toggleLed()
{
    int ledType = -1;
    DynamicJsonDocument req(1024);
    deserializeJson(req, server.arg("plain"));
    const char *led = req["ledType"];
    if (strcmp(led, "red") == 0)
    {
        ledType = 1;
    }
    else if (strcmp(led, "yellow") == 0)
    {
        ledType = 2;
    }
    else if (strcmp(led, "green") == 0)
    {
        ledType = 3;
    }
    switch (ledType)
    {
    case 1:
    {
        Serial.print("RED LED Toggled!\n");
        digitalWrite(redPin, !digitalRead(redPin));
        break;
    }
    case 2:
    {
        Serial.print("YELLOW LED Toggled!\n");
        digitalWrite(yellowPin, !digitalRead(yellowPin));
        break;
    }
    case 3:
    {
        Serial.print("GREEN LED Toggled!\n");
        digitalWrite(greenPin, !digitalRead(greenPin));
        break;
    }
    default:
        break;
    }
    server.send(200, "text/html", "OK");
}

void configureSSDP()
```

```
{
    String hostname = getHostname();
    SSDP.setSchemaURL("description.xml");
    SSDP.setHTTPPort(PORT);
    SSDP.setName(hostname);
    SSDP.setSerialNumber(ESP.getChipId());
    SSDP.setURL("http://" + hostname + ".local");
    SSDP.setModelName("ESP8266");
    SSDP.setModelNumber(WiFi.macAddress());
    SSDP.setModelURL("http://" + hostname + ".local");
    SSDP.setManufacturer("Gita Alekhya Paul");
    SSDP.setManufacturerURL("https://esp8266.gitaalekhyapaul.com");
    SSDP.setDeviceType("urn:schemas-upnp-org:device:ESP8266:1");
    SSDP.setInterval(1000);
    SSDP.begin();
    Serial.println("SSDP Completed.");
}

void startWebServer()
{
    server.on("/description.xml", HTTP_GET, handleDescription);
    server.on("/api/v1/ledChange", HTTP_POST, toggleLed);
    server.onNotFound(handle404NotFound);
    server.begin();
    MDNS.addService("http", "tcp", PORT);
    Serial.print("Web server started on Port ");
    Serial.println(PORT);
}

void handleDescription()
{
    digitalWrite(statusLed, LOW);
    SSDP.schema(server.client());
}

void startMDNS()
{
    Serial.println("Starting mDNS...");
    String hostname = getHostname();
    if (!MDNS.begin(hostname))
    {
        Serial.println("Error starting up mDNS responder!");
        Serial.println("Restarting...");
        ESP.restart();
    }
    Serial.println("mDNS responder started.");
    Serial.println("Hostname: " + hostname + ".local");
}

void initPins()
{
    pinMode(redPin, OUTPUT);
    pinMode(yellowPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
```

```
    pinMode(statusLed, OUTPUT);
}

String getHostname()
{
    String hostname = WiFi.macAddress();
    hostname.replace(":", "");
    hostname = "ESP8266" + hostname;
    return hostname;
}

void wifiConnect()
{
    Serial.print("Connecting to WiFi");
    wifiManager.setDebugOutput(false);
    wifiManager.autoConnect();
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
    return;
}
```

/public/css/style.css :

```
@import url("https://fonts.googleapis.com/css2?family=Kanit&display=swap");
.container-fluid {
  height: 100vh;
}
* {
  font-family: "Kanit", sans-serif;
}
```

/public/js/dashboard.js :

```
(() => {
  let socket;
  let devices = [];

  const searchForDevices = async (ev) => {
    try {
      if (devices.length === 0) {
        searchSpinner.hidden = false;
      }
      const response = await fetch("/api/v1/device/search", {
        method: "GET",
        headers: {
          Authorization: `Bearer ${sessionStorage.getItem("authToken")}`,
```

```
      },
    });
    if (!response.ok) {
      if (response.status !== 403) throw await response.json();
      else {
        alert("Session expired! Please login again!");
        window.location.replace("/");
      }
    }
  } catch (err) {
    searchSpinner.hidden = true;
    console.dir(err);
    alert("APIError");
  }
};

const render = async () => {
  let html = "";
  if (devices.length > 0) {
    devices.forEach((device) => {
      html += generateCard(device);
    });
    const deviceCards = document.getElementById("device-cards");
    deviceCards.innerHTML = html;
    const connectButtons = document.querySelectorAll("#connect-new");
    connectButtons.forEach((e) =>
      e.addEventListener("click", async (ev) => {
        try {
          const btn = ev.target;
          let identity = {};
          btn.parentElement.parentElement.parentElement
            .querySelectorAll("input")
            .forEach((input) => {
              identity[input.name] = input.value;
            });
          console.log(identity);
          const response = await fetch("/api/v1/device/add", {
            method: "POST",
            headers: {
              Authorization: `Bearer ${sessionStorage.getItem("authToken")}`,
              "Content-Type": "application/json",
            },
            body: JSON.stringify(identity),
          });
          if (!response.ok) {
            if (response.status !== 403) throw await response.json();
            else {
              alert("Session expired! Please login again!");
              window.location.replace("/");
            }
          }
          devices.map((index) => {
            if (index.uuid === identity.uuid) {
              index.exists = true;
```

```
        }
      });
      await render();
    } catch (err) {
      console.dir(err);
      alert("APIError");
    }
  })
);
const redLedButtons = document.querySelectorAll("#red-led");
const yellowLedButtons = document.querySelectorAll("#yellow-led");
const greenLedButtons = document.querySelectorAll("#green-led");
redLedButtons.forEach((e) =>
  e.addEventListener("click", async (ev) => {
    try {
      const btn = ev.target;
      let identity = {};
      btn.parentElement.parentElement.parentElement
        .querySelectorAll("input")
        .forEach((input) => {
          identity[input.name] = input.value;
        });
      console.log(identity);
      await fetch(`${identity.link}api/v1/ledChange`, {
        method: "POST",
        mode: "no-cors",
        body: JSON.stringify({ ledType: "red" }),
      });
    } catch (err) {
      console.dir(err);
      alert("APIError");
    }
  })
);
yellowLedButtons.forEach((e) =>
  e.addEventListener("click", async (ev) => {
    try {
      const btn = ev.target;
      let identity = {};
      btn.parentElement.parentElement.parentElement
        .querySelectorAll("input")
        .forEach((input) => {
          identity[input.name] = input.value;
        });
      console.log(identity);
      await fetch(`${identity.link}api/v1/ledChange`, {
        method: "POST",
        mode: "no-cors",
        body: JSON.stringify({ ledType: "yellow" }),
      });
    } catch (err) {
      console.dir(err);
      alert("APIError");
    }
```

```
      })
    );
    greenLedButtons.forEach((e) =>
      e.addEventListener("click", async (ev) => {
        try {
          const btn = ev.target;
          let identity = {};
          btn.parentElement.parentElement.parentElement
            .querySelectorAll("input")
            .forEach((input) => {
              identity[input.name] = input.value;
            });
          console.log(identity);
          await fetch(`${identity.link}api/v1/ledChange`, {
            method: "POST",
            mode: "no-cors",
            body: JSON.stringify({ ledType: "green" }),
          });
        } catch (err) {
          console.dir(err);
          alert("APIError");
        }
      })
    );
  }
};

  const generateCard = (device) => {
    return `
    <div class="col-sm-3 py-2">
  <div class="card bg-dark text-white shadow-lg">
  <div class="card-header">${device.modelNumber}</div>
  <div class="card-body">
                <h5 class="card-title">NodeMCU </h5>
                <p class="card-text">
                ${device.name}
                </p>
                <input type="hidden" name="urn" value="${device.urn}"></input>
                <input type="hidden" name="uuid" value="${
                  device.uuid
                }"></input>
                <input type="hidden" name="link" value="${
                  device.link
                }"></input>
                <div class="btn-group w-100 d-flex justify-content-between
align-items-center" role="group">
                ${
                  device.exists === false
                    ? `<div id="connect-button mx-auto my-auto">
                    <button type="button" class="btn btn-primary shadow-lg"
id="connect-new">Add Device</button>
                      </div>`
                    : `
                      <div id="connected-buttons mx-auto my-auto">
```

```
                        <button type="button" class="btn btn-danger mx-1"
id="red-led">Red</button>
                        <button type="button" class="btn btn-warning mx-1"
id="yellow-led">
                    Yellow
                    </button>
                    <button type="button" class="btn btn-success mx-1"
id="green-led" >
                    Green
                    </button>
                    </div>
                    `
                  }
                  </div>
                  </div>
                  </div>
            </div>
            `;
  };

  const searchSpinner = document.getElementById("search-button-spinner");

  const searchButton = document.getElementById("search-button");
  searchButton.onclick = searchForDevices;

  const logoutButton = document.getElementById("logout-button");
  logoutButton.onclick = (ev) => {
    sessionStorage.removeItem("authToken");
    alert("Bye-bye!");
    window.location.replace("/");
  };

  if (!sessionStorage.getItem("authToken")) {
    alert("Session timeout! Please login!");
    window.location.replace("/");
  } else {
    alert("Welcome to dashboard!");
    socket = io();
    searchForDevices()
      .then(() => {
        render();
      })
      .catch((err) => {});
  }
  setInterval(searchForDevices, 5000);

  socket.on("new-device", async (data) => {
    try {
      const response = await fetch("/api/v1/device/find", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${sessionStorage.getItem("authToken")}`,
        },
        body: JSON.stringify({
```

```
          urn: data.urn,
          uuid: data.uuid,
          link: data.link,
        }),
      });
      if (!response.ok) throw await response.json();
      const res = await response.json();
      console.log(res.exists);
      if (res) {
        const device = {
          exists: res.exists,
          ...data,
        };
        console.log(device);
        if (
          devices.findIndex(
            (p) => p.uuid === device.uuid && p.urn === device.urn
          ) === -1
        )
          devices.push(device);
        searchSpinner.hidden = true;
        render();
      }
    } catch (err) {
      console.dir(err);
      alert("APIError");
    }
  });
})();
```

/public/js/index.js :

```
const loginButton = document.getElementById("login-button");
const signupButton = document.getElementById("signup-button");

loginButton.onclick = (ev) => {
  alert("Login");
};

signupButton.onclick = async (ev) => {
  try {
    const email = document.getElementById("email");
    const password = document.getElementById("password");
    if (email.value && password.value) {
      const response = await fetch("/api/v1/auth/signup", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email: email.value,
          password: password.value,
        }),
      });
```

```
      if (!response.ok) throw await response.json();
      alert("Successful Sign-up! Please login to continue!");
    } else alert("Email or password is blank.");
  } catch (err) {
    console.dir(err);
    alert(`APIError: ${err.error}`);
  }
};

loginButton.onclick = async (ev) => {
  try {
    const email = document.getElementById("email");
    const password = document.getElementById("password");
    if (email.value && password.value) {
      const response = await fetch("/api/v1/auth/login", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email: email.value,
          password: password.value,
        }),
      });
      if (!response.ok) throw await response.json();
      const res = await response.json();
      sessionStorage.setItem("authToken", res.authToken);
      alert("Welcome User!");
      window.location.replace("/dashboard");
    } else alert("Email or password is blank.");
  } catch (err) {
    console.dir(err);
    alert(`APIError: ${err.error}`);
  }
};
```

/views/dashboard.ejs :

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/
bootstrap.min.css" rel="stylesheet"
    integrity="sha384-
giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"
crossorigin="anonymous" />
  <link rel="stylesheet" href="/css/style.css" />
  <title>Home Automation</title>
</head>

<body class="bg-dark bg-gradient text-white">
```

```
  <div class="container-fluid">
    <div class="row h-100">
      <div class="col-10-md col-12 mx-auto my-3">
        <div class="row d-flex justify-content-between align-items-center my-5">
          <div class="col-3-md col-8">
            <button type="button" class="btn btn-info shadow-lg btn-lg d-flex"
id="search-button">
              <div class="d-flex justify-content-center align-items-center">
                <span class="mx-1">
                  Search for new devices
                </span>
                <span class="spinner-border spinner-border-sm mx-1" id="search-
button-spinner" hidden>
                </span>
              </div>
            </button>
          </div>
          <div class="col-3-md col-4 text-end">
            <button type="button" class="btn btn-secondary shadow-lg btn-lg"
id="logout-button">
              Logout
            </button>
          </div>
        </div>
        <div class="row" id="device-cards"></div>
      </div>
    </div>
  </div>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle
.min.js"

integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroa
fW"
    crossorigin="anonymous"></script>
  <script src="/socket.io/socket.io.js"></script>
  <script src="/js/dashboard.js"></script>
</body>

</html>
```

/views/index.ejs :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/
bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-
giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"
```

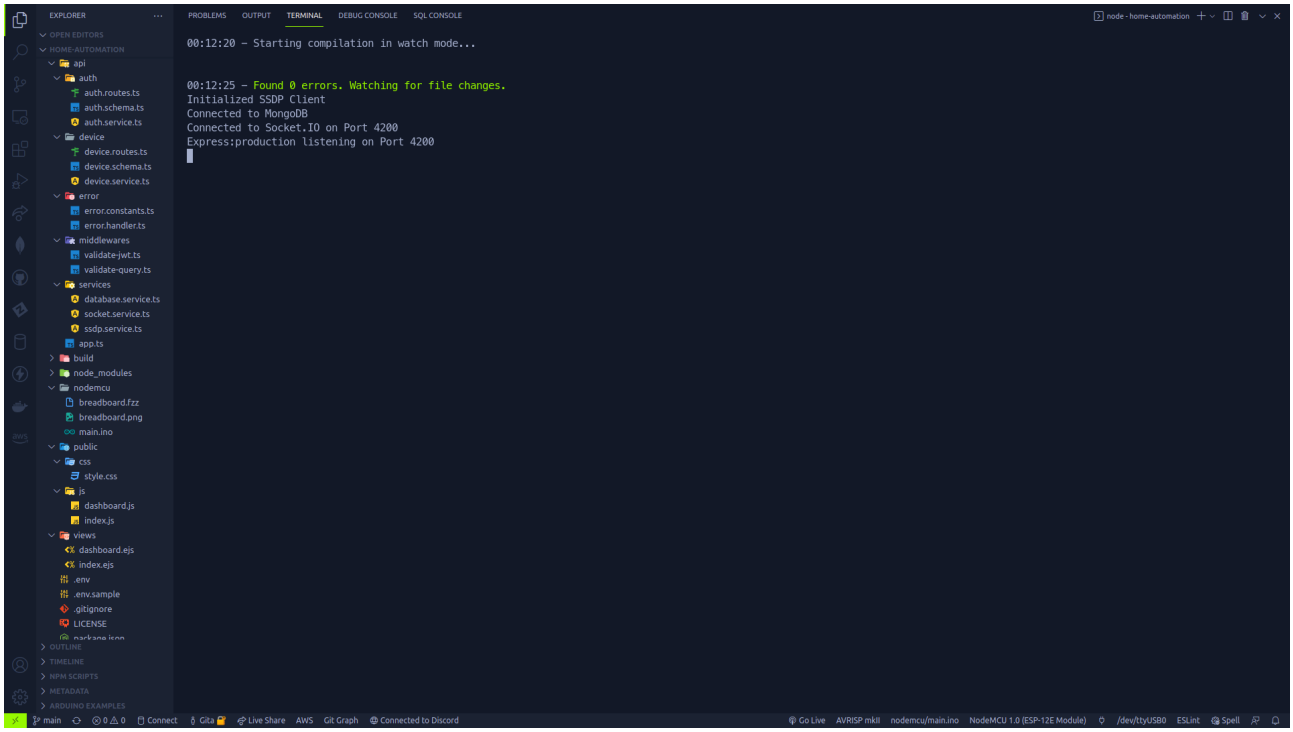
```
    crossorigin="anonymous"
  />
  <link rel="stylesheet" href="/css/style.css" />
  <title>Home Automation</title>
</head>
<body class="bg-dark bg-gradient text-white">
  <div class="container-fluid">
    <div class="row h-100">
      <div class="col-10 col-md-4 mx-auto my-auto">
        <div class="text-center pb-5">
          <h1 class="display-1">Dashboard</h1>
        </div>
        <div class="mt-5 pb-5 form-floating">
          <input
            type="email"
            class="form-control bg-dark form-control-lg text-white shadow-lg"
            id="email"
            placeholder="name@example.com"
          />
          <label for="email" class="form-label">Email</label>
        </div>
        <div class="pb-3 form-floating">
          <input
            type="password"
            class="form-control bg-dark form-control-lg text-white shadow-lg"
            id="password"
            placeholder="Enter your password"
          />
          <label for="password" class="form-label">Password</label>
        </div>
        <div class="pt-5 btn-group d-flex w-100" role="group">
          <div class="row d-flex w-100 my-4">
            <div class="col-6 text-center">
              <button
                type="button"
                class="btn btn-success bg-gradient btn-lg"
                id="login-button"
              >
                Login
              </button>
            </div>
            <div class="col-6 text-center">
              <button
                type="button"
                class="btn btn-primary bg-gradient btn-lg"
                id="signup-button"
              >
                Signup
              </button>
            </div>
          </div>
        </div>
      </div>
    </div>
```

```
    </div>
    <script

src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle
.min.js"

integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroa
fW"
      crossorigin="anonymous"
    ></script>
    <script src="/js/index.js"></script>
  </body>
</html>
```

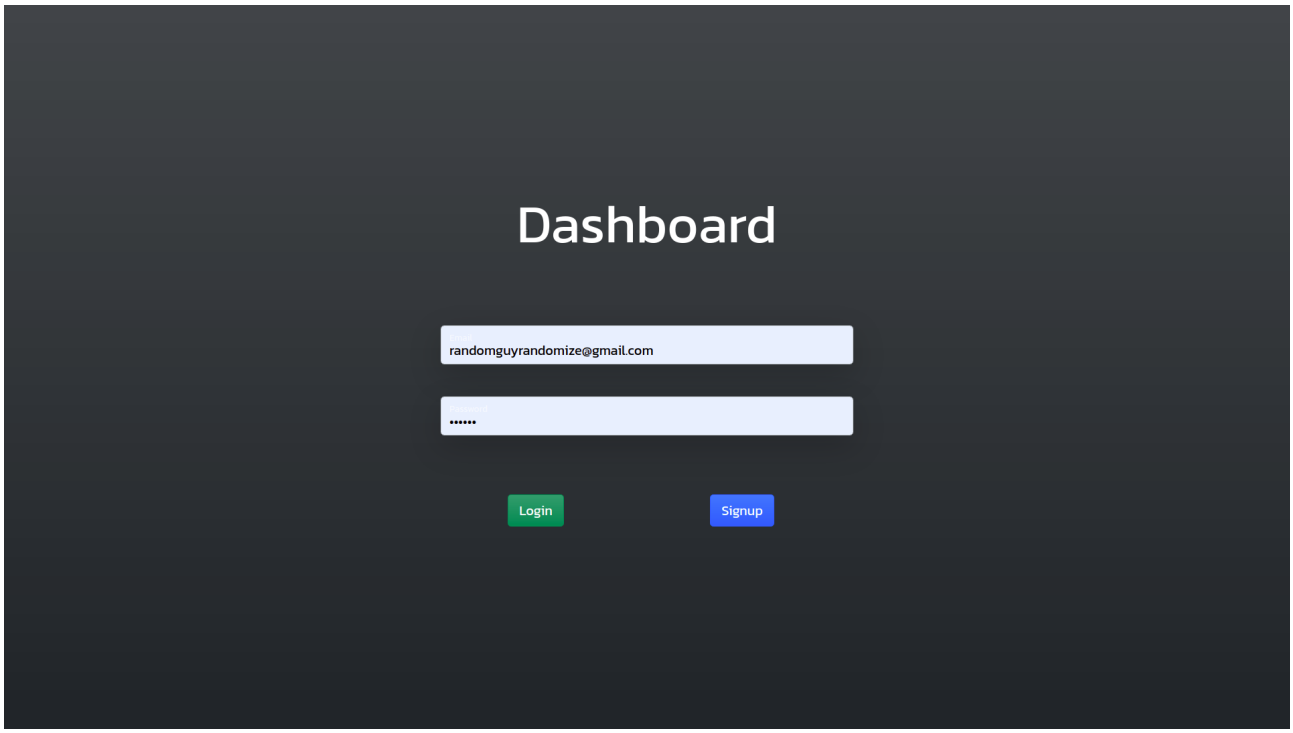
Result and Discussion:

The devices are set up as per specification, with LEDs connected to the NodeMCU as per the circuit diagram. The above web server code is saved and executed on a Raspberry Pi to act as a smart home device management dashboard. The screenshots and video demonstration linked below accurately proves the points mentioned above.

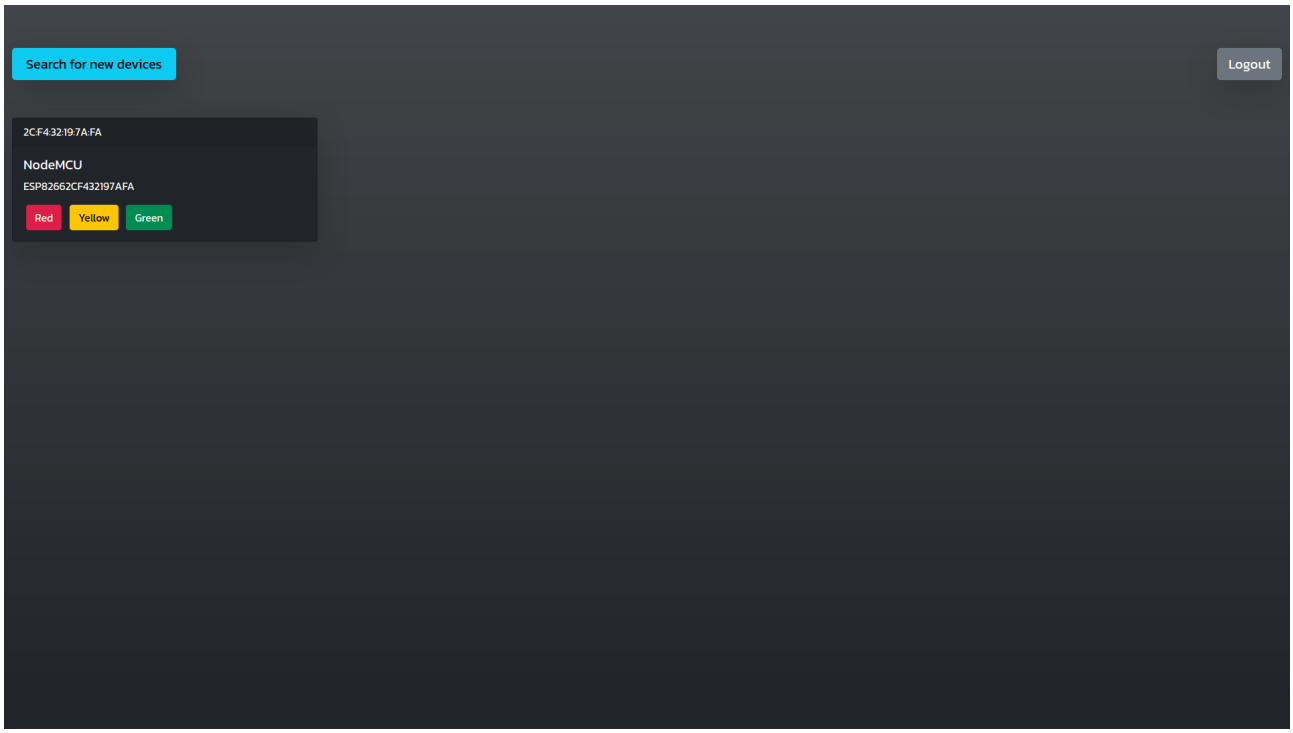
Running the server:



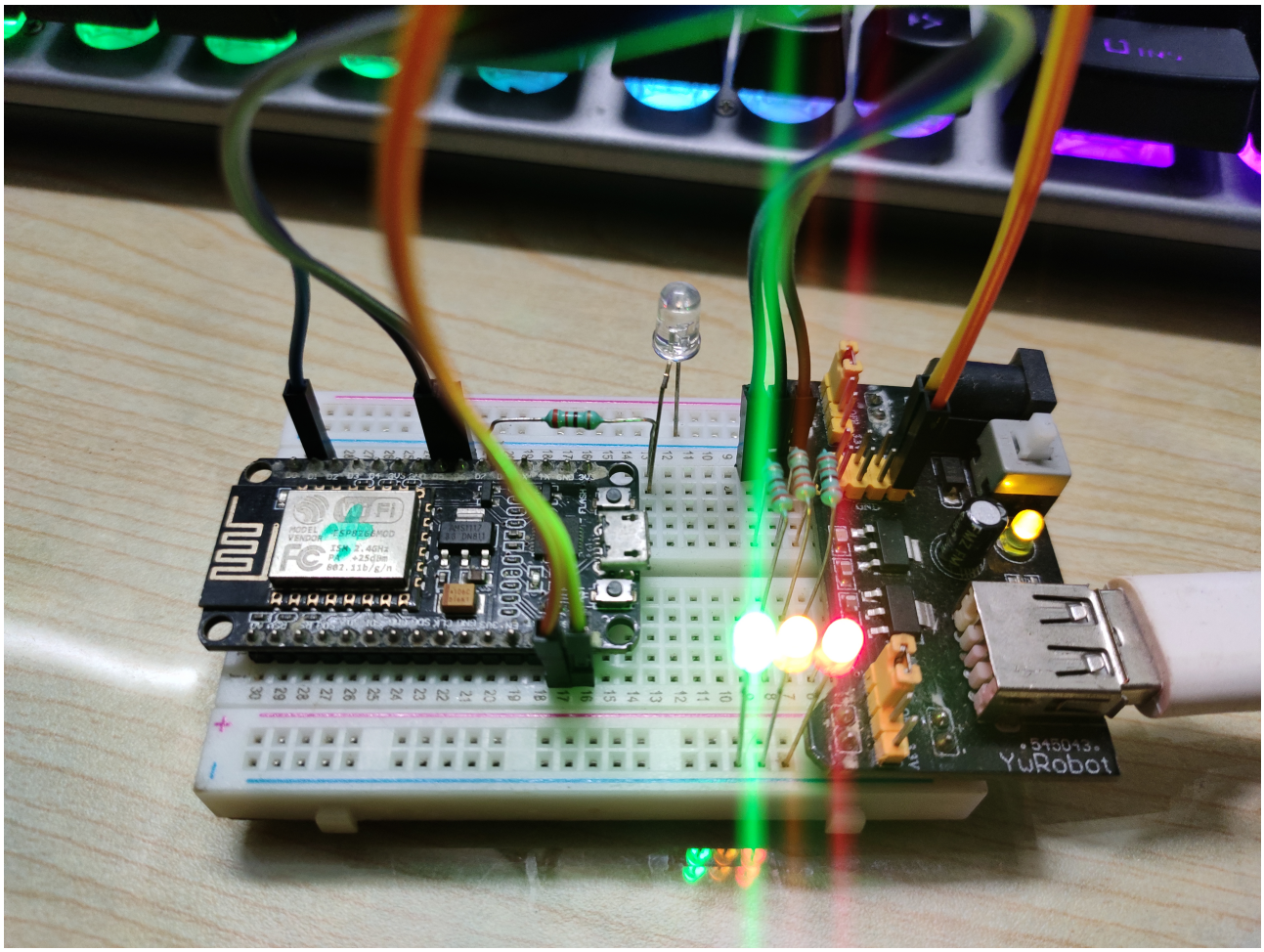
Login Dashboard:



Device Discovery:



NodeMCU Smart Bulb in Action:



Video Demo Link (Uploaded in GCR):

https://drive.google.com/file/d/14tkL6RdwSxeuvMXI7HUFLW7gij5yU2SW/view

Future Improvements and Discussions:

One of the significant future improvements that I can make in the project is modifying and enhancing the smart home dashboard code and accepting and adapting to various smart devices rather than only smart LED bulbs. This can be successfully done by researching and implementing multiple UPnP device descriptions and making the API capable of handling numerous types of device descriptions.

Conclusion:

To conclude the project report, I would like to highlight some of the key features of the product as well as the requirement of the same:

- India has a dominating smart home device market, and it is ever increasing. Day by day, more Indians are buying devices like Alexa, Google Home and Philips Hue. Although the consumers are purchasing such devices, most of them are unaware of how they work and what software and internet protocols they use to communicate. This project aims to demystify the technology behind the auto-detection of smart home devices in your network.
- The project helps in demonstrating the use of SSDP, UPnP and HTTP protocols to auto-discover smart home devices as soon as they connect to the same network as the device management hub, which is being deployed on a Raspberry Pi. The project consists of a smart LED bulb using NodeMCU ESP8266 and a central hub deployed via a Node.js server. The central hub listens for new devices connected to the network via SSDP self-discovery and then extracts their XML device descriptions to know their API endpoints. The clients can use the dashboard to trigger the device APIs to change their state, which is to change their LEDs.

References:

1. https://www.statista.com/statistics/1182543/india-smart-speakers-market-share-by-company/
2. https://www.statista.com/statistics/1098396/india-ownership-of-smart-speakers/
3. https://www.netify.ai/resources/protocols/ssdp
4. https://blog.cloudflare.com/ssdp-100gbps/
5. https://www.npmjs.com/package/node-ssdp
6. https://datatracker.ietf.org/doc/html/draft-cai-ssdp-v1-03
7. https://stormwall.network/knowledge-base/protocol/ssdp