CODE DOCUMENTATION

# SEM\_5\_proj\_back

|  |  |
| --- | --- |
| **Project:** | SEM\_5\_proj\_back |
| **Path:** | C:\Users\Suyog\Downloads\PROJECT\_WEBSITE\SEM\_5\_proj\_back |
| **Generated:** | October 21, 2025 at 09:09 PM |
| **Document Type:** | Complete Source Code Documentation |

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **File Path** | **File Type** |
| 1 | index.js | .JS |
| 2 | package.json | .JSON |
| 3 | README.md | .MD |

# SOURCE CODE FILES

## FILE 1: index.js

|  |  |  |  |
| --- | --- | --- | --- |
| **Language: JavaScript** | Size: 14.07 KB | Lines: 445 | Extension: .JS |

**CODE CONTENT:**

import express from "express";  
import mysql from "mysql2";  
import { SerialPort } from "serialport";  
import { ReadlineParser } from "@serialport/parser-readline";  
import dotenv from "dotenv";  
import cors from "cors";  
import bcrypt from "bcryptjs";  
import jwt from "jsonwebtoken";  
  
dotenv.config();  
  
const app = express();  
const port = process.env.PORT || 3000;  
  
// ✅ Allow frontend requests  
app.use(  
 cors({  
 origin: "http://localhost:5173",  
 methods: ["GET", "POST", "PUT", "DELETE"],  
 credentials: true,  
 })  
);  
app.use(express.json());  
  
// MySQL Database Connection  
const db = mysql.createConnection({  
 host: process.env.DB\_HOST,  
 user: process.env.DB\_USER,  
 password: process.env.DB\_PASS,  
 database: process.env.DB\_NAME,  
});  
  
db.connect((err) => {  
 if (err) {  
 console.error("Database connection failed: " + err.stack);  
 return;  
 }  
 console.log("Connected to database as id " + db.threadId);  
});  
  
// ---------------- ESP32 Serial Port Code (UNCHANGED) ----------------  
let serialPort;  
try {  
 serialPort = new SerialPort({  
 path: process.env.SERIAL\_PORT,  
 baudRate: parseInt(process.env.SERIAL\_BAUD) || 115200,  
 });  
  
 const parser = serialPort.pipe(new ReadlineParser({ delimiter: "\r\n" }));  
  
 parser.on("data", (data) => {  
 const cleanData = data.replace(/[^\x20-\x7E\r\n]/g, "").trim();  
 if (!cleanData) return;  
  
 console.log("Cleaned data from ESP32:", cleanData);  
  
 if (  
 cleanData.includes("Received:") ||  
 cleanData.includes("Backend Response:")  
 )  
 return;  
  
 try {  
 const parts = cleanData.split(":");  
 if (parts.length < 2) {  
 console.error("Invalid data format:", cleanData);  
 sendSerialResponse("ERROR:Invalid format");  
 return;  
 }  
  
 const command = parts[0];  
  
 if (command === "REG") {  
 if (parts.length < 4) {  
 console.error("Invalid registration data:", cleanData);  
 sendSerialResponse("ERROR:Invalid registration data");  
 return;  
 }  
 const userId = parseInt(parts[1]);  
 const password = parts[2];  
 const fingerId = parseInt(parts[3]);  
 handleRegistration(userId, password, fingerId);  
 } else if (command === "LOGIN") {  
 if (parts.length < 3) {  
 console.error("Invalid login data:", cleanData);  
 sendSerialResponse("ERROR:Invalid login data");  
 return;  
 }  
 const password = parts[1];  
 const fingerId = parseInt(parts[2]);  
 const confidence = parts.length > 3 ? parseFloat(parts[3]) : 0;  
 handleLogin(password, fingerId, confidence);  
 } else {  
 console.error("Unknown command:", command);  
 sendSerialResponse("ERROR:Unknown command");  
 }  
 } catch (error) {  
 console.error("Error parsing data:", error);  
 sendSerialResponse("ERROR:Parse error");  
 }  
 });  
  
 serialPort.on("error", (err) =>  
 console.error("Serial port error:", err.message)  
 );  
 serialPort.on("open", () => console.log("Serial port opened successfully"));  
} catch (error) {  
 console.error("Failed to initialize serial port:", error.message);  
}  
  
function sendSerialResponse(message) {  
 if (serialPort && serialPort.isOpen) {  
 console.log("Sent to ESP32:", message);  
 serialPort.write(message + "\n");  
 }  
}  
  
function handleRegistration(userId, password, fingerId) {  
 const checkUserQuery = "SELECT \* FROM users WHERE id = ?";  
 db.query(checkUserQuery, [userId], (err, results) => {  
 if (err) return sendSerialResponse("ERROR:Database error");  
 if (results.length > 0) {  
 sendSerialResponse("ERROR:User ID exists");  
 return;  
 }  
  
 const checkFingerQuery = "SELECT \* FROM users WHERE finger\_id = ?";  
 db.query(checkFingerQuery, [fingerId], (err, results) => {  
 if (err) return sendSerialResponse("ERROR:Database error");  
 if (results.length > 0) {  
 sendSerialResponse("ERROR:Fingerprint exists");  
 return;  
 }  
  
 const insertQuery =  
 "INSERT INTO users (id, finger\_id, password) VALUES (?, ?, ?)";  
 db.query(insertQuery, [userId, fingerId, password], (err) => {  
 if (err) return sendSerialResponse("ERROR:Insert failed");  
 sendSerialResponse("SUCCESS:User registered");  
 });  
 });  
 });  
}  
  
function handleLogin(password, fingerId, confidence) {  
 const joinQuery = `  
 SELECT u.id AS id, u.password AS esp\_password, p.password\_hash AS profile\_hash, p.email, p.mobile  
 FROM users u  
 LEFT JOIN user\_profiles p ON u.id = p.user\_id  
 WHERE u.finger\_id = ?  
 LIMIT 1  
 `;  
  
 db.query(joinQuery, [fingerId], (err, results) => {  
 if (err) {  
 console.error("DB error (login join):", err);  
 return sendSerialResponse("ERROR:Database error");  
 }  
 if (results.length === 0) {  
 return sendSerialResponse("ERROR:Fingerprint not registered");  
 }  
  
 const row = results[0];  
  
 if (row.profile\_hash) {  
 bcrypt.compare(password, row.profile\_hash, (cmpErr, same) => {  
 if (cmpErr) {  
 console.error("Bcrypt compare error:", cmpErr);  
 return sendSerialResponse("ERROR:Server error");  
 }  
 if (same) {  
 sendSerialResponse("SUCCESS:Login granted");  
 } else {  
 sendSerialResponse("ERROR:Invalid password");  
 }  
 });  
 } else {  
 if (row.esp\_password === password) {  
 sendSerialResponse("SUCCESS:Login granted");  
 } else {  
 sendSerialResponse("ERROR:Invalid password");  
 }  
 }  
 });  
}  
  
// ---------------- API Routes ----------------  
app.get("/api/logs", (req, res) => {  
 const query = "SELECT \* FROM access\_logs ORDER BY created\_at DESC LIMIT 50";  
 db.query(query, (err, results) => {  
 if (err) return res.status(500).json({ error: err.message });  
 res.json(results);  
 });  
});  
  
app.get("/api/users", (req, res) => {  
 const query = "SELECT \* FROM users";  
 db.query(query, (err, results) => {  
 if (err) return res.status(500).json({ error: err.message });  
 res.json(results);  
 });  
});  
  
app.get("/api/test", (req, res) => {  
 res.json({ message: "Backend Connected ✅" });  
});  
  
// ---------------- Frontend Registration Route (UPDATED) ----------------  
// ---------------- REGISTER ROUTE ----------------  
app.post("/api/register", async (req, res) => {  
 const { user\_id, email, mobile, password, role } = req.body; // Added role field  
  
 if (!user\_id || !email || !password) {  
 return res.status(400).json({ error: "Missing required fields" });  
 }  
  
 try {  
 // Step 1: Check if user exists in `users` table  
 db.query(  
 "SELECT \* FROM users WHERE id = ?",  
 [user\_id],  
 async (err, results) => {  
 if (err) {  
 console.error("Database error (users check):", err);  
 return res.status(500).json({ error: "Database error" });  
 }  
  
 // Step 2: If user does not exist (e.g., not yet registered by ESP32)  
 if (results.length === 0) {  
 const dummyPassword = "frontend\_dummy";  
 const dummyFingerId = 0; // placeholder until ESP32 registers it  
 db.query(  
 "INSERT INTO users (id, finger\_id, password) VALUES (?, ?, ?)",  
 [user\_id, dummyFingerId, dummyPassword],  
 (err) => {  
 if (err) {  
 console.error("Error creating dummy user:", err);  
 return res  
 .status(500)  
 .json({ error: "Failed to create base user record" });  
 }  
 saveUserProfile(); // proceed to create user profile  
 }  
 );  
 } else {  
 // Step 3: If already exists in `users`, just create profile (if not exists)  
 saveUserProfile();  
 }  
  
 // Step 4: Create or update user profile  
 async function saveUserProfile() {  
 try {  
 const hashed = await bcrypt.hash(password, 10);  
  
 // Check if user\_profile already exists  
 db.query(  
 "SELECT \* FROM user\_profiles WHERE user\_id = ?",  
 [user\_id],  
 (err, profileResults) => {  
 if (err) {  
 console.error("Database error (profile check):", err);  
 return res.status(500).json({ error: "Database error" });  
 }  
  
 if (profileResults.length > 0) {  
 return res.status(400).json({  
 error: "User profile already exists. Please login.",  
 });  
 }  
  
 const userRole = role && role === "admin" ? "admin" : "user"; // Default role=user  
  
 db.query(  
 "INSERT INTO user\_profiles (user\_id, email, mobile, password\_hash, role) VALUES (?, ?, ?, ?, ?)",  
 [user\_id, email, mobile, hashed, userRole],  
 (err) => {  
 if (err) {  
 console.error(  
 "Database insert error (user\_profiles):",  
 err  
 );  
 return res  
 .status(500)  
 .json({ error: "Failed to create user profile" });  
 }  
  
 res.json({  
 success: true,  
 message: `User registered successfully as ${userRole}`,  
 });  
 }  
 );  
 }  
 );  
 } catch (error) {  
 console.error("Registration error:", error);  
 res  
 .status(500)  
 .json({ error: "Server error during profile creation" });  
 }  
 }  
 }  
 );  
 } catch (error) {  
 console.error("Unexpected registration error:", error);  
 res.status(500).json({ error: "Unexpected server error" });  
 }  
});  
  
// Test backend route  
app.get("/api/test", (req, res) => {  
 res.json({ message: "Backend Connected ✅" });  
});  
  
// ------------------- ADD THIS ROUTE HERE -------------------  
app.get("/api/user-by-email", (req, res) => {  
 const { email } = req.query;  
 if (!email) return res.status(400).json({ error: "Email is required" });  
  
 db.query(  
 "SELECT user\_id FROM user\_profiles WHERE email = ? LIMIT 1",  
 [email],  
 (err, results) => {  
 if (err) return res.status(500).json({ error: "Database error" });  
 if (results.length === 0) return res.json({ user: null });  
 res.json({ user: results[0] });  
 }  
 );  
});  
// -------------------------------------------------------------  
  
app.post("/api/login", async (req, res) => {  
 try {  
 const { user\_id, password } = req.body;  
 if (!user\_id || !password)  
 return res.status(400).json({ error: "Missing required fields" });  
  
 const query = `  
 SELECT u.id AS id, u.password AS esp\_password, p.password\_hash AS profile\_hash, p.role AS role  
 FROM users u  
 LEFT JOIN user\_profiles p ON u.id = p.user\_id  
 WHERE u.id = ?  
 LIMIT 1  
 `;  
  
 db.query(query, [user\_id], async (err, results) => {  
 if (err) {  
 console.error("Database error during login:", err);  
 return res.status(500).json({ error: "Database error" });  
 }  
  
 if (!results || results.length === 0) {  
 return res.status(404).json({ error: "User not found" });  
 }  
  
 const row = results[0];  
 if (!row) return res.status(404).json({ error: "User row missing" });  
  
 // Helper function to send success  
 const handleSuccess = (roleType) => {  
 if (!process.env.JWT\_SECRET) {  
 console.error("JWT\_SECRET missing in .env");  
 return res.status(500).json({ error: "Server configuration error" });  
 }  
 const token = jwt.sign(  
 { id: row.id, role: roleType },  
 process.env.JWT\_SECRET,  
 { expiresIn: "2h" }  
 );  
 return res.json({  
 success: true,  
 message: "Login successful",  
 role: roleType,  
 token,  
 });  
 };  
  
 // Compare passwords safely  
 if (row.profile\_hash) {  
 try {  
 const same = await bcrypt.compare(password, row.profile\_hash);  
 if (same) handleSuccess(row.role || "user");  
 else return res.status(401).json({ error: "Invalid password" });  
 } catch (bcryptErr) {  
 console.error("Bcrypt error:", bcryptErr);  
 return res.status(500).json({ error: "Server error" });  
 }  
 } else {  
 // Fallback: compare with ESP32 plain password  
 if (row.esp\_password === password) handleSuccess(row.role || "user");  
 else return res.status(401).json({ error: "Invalid password" });  
 }  
 });  
 } catch (error) {  
 console.error("Unhandled login error:", error);  
 return res.status(500).json({ error: "Server error" });  
 }  
});  
  
app.get("/api/logs/user/:user\_id", (req, res) => {  
 const { user\_id } = req.params;  
  
 const query = `  
 SELECT a.id, a.user\_id, a.result, a.note, a.created\_at,  
 p.email, p.mobile  
 FROM access\_logs a  
 JOIN user\_profiles p ON a.user\_id = p.user\_id  
 WHERE a.user\_id = ?  
 ORDER BY a.created\_at DESC  
 LIMIT 50  
 `;  
  
 db.query(query, [user\_id], (err, results) => {  
 if (err) return res.status(500).json({ error: err.message });  
 res.json(results);  
 });  
});  
  
app.get("/api/logs/all", (req, res) => {  
 const query = `  
 SELECT a.id, a.user\_id, a.result, a.note, a.created\_at,  
 p.email, p.mobile  
 FROM access\_logs a  
 JOIN user\_profiles p ON a.user\_id = p.user\_id  
 ORDER BY a.created\_at DESC  
 LIMIT 500  
 `;  
  
 db.query(query, (err, results) => {  
 if (err) return res.status(500).json({ error: err.message });  
 res.json(results);  
 });  
});  
  
// ---------------- Start Server ----------------  
app.listen(port, () =>  
 console.log(`✅ Backend running on http://localhost:${port}`)  
);  
  
process.on("SIGINT", () => {  
 console.log("Shutting down server...");  
 if (serialPort && serialPort.isOpen) serialPort.close();  
 db.end();  
 process.exit(0);  
});

▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬

## FILE 2: package.json

|  |  |  |  |
| --- | --- | --- | --- |
| **Language: JSON** | Size: 0.53 KB | Lines: 24 | Extension: .JSON |

**CODE CONTENT:**

{  
 "name": "project\_1",  
 "version": "1.0.0",  
 "main": "index.js",  
 "type": "module",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "keywords": [],  
 "author": "",  
 "license": "ISC",  
 "description": "",  
 "dependencies": {  
 "@serialport/parser-readline": "^13.0.0",  
 "bcryptjs": "^3.0.2",  
 "cors": "^2.8.5",  
 "dotenv": "^17.2.3",  
 "express": "^5.1.0",  
 "jsonwebtoken": "^9.0.2",  
 "mysql": "^2.18.1",  
 "mysql2": "^3.15.2",  
 "serialport": "^13.0.0"  
 }  
}

▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬

## FILE 3: README.md

|  |  |  |  |
| --- | --- | --- | --- |
| **Language: Text** | Size: 0.04 KB | Lines: 0 | Extension: .MD |

**CODE CONTENT:**

# PROJECT SUMMARY

|  |  |
| --- | --- |
| **Project Name:** | SEM\_5\_proj\_back |
| **Total Files Documented:** | 3 files |
| **Total Lines of Code:** | 469 lines |
| **Approximate Project Size:** | 0.12 MB |
| **Document Generated:** | October 21, 2025 |

This document contains the complete source code documentation for the project. All code files have been extracted and formatted for better readability.