**Note:- Encode the IP of the ESP32 during deployment on Render**

**🔐 ESP32-Based OTP Verification System using DHT11 and Timestamp**

**📌 Overview**

This project is an **IoT-integrated OTP authentication system** where a **unique 4-digit OTP** is generated by an **ESP32 microcontroller** using real-time entropy from the **DHT11 sensor (temperature & humidity)** and **timestamp data**. The OTP is then securely sent via **SMS to a user-provided mobile number**, and the system verifies the user-entered OTP through a web-based UI.

**🎯 Key Features**

* ✅ **Hardware-Generated OTP** using ESP32 with real-world sensor input (DHT11)
* ✅ **Stateless Web UI** with elegant UX for mobile number entry and OTP verification
* ✅ **SMS Delivery** via Fast2SMS, Twilio, or other configurable APIs
* ✅ **Node.js Backend** for OTP generation, storage (in-memory), and SMS integration
* ✅ **Secure Verification Logic** on the server-side
* ✅ **ESP32 HTTP Communication** with backend to report/generated OTP

**⚙️ Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| Microcontroller | ESP32 DevKit V1 |
| Sensor | DHT11 |
| Communication | HTTP POST to Node.js API |
| Backend | Node.js + Express.js |
| SMS API | Fast2SMS / Twilio / Textbelt |
| Frontend | HTML + JS (with vivid CSS) |
| Hosting | Render (for backend & frontend deployment) |

**🧠 How It Works**

1. **ESP32 reads DHT11** sensor values and combines them with a timestamp.
2. A **4-digit OTP** is generated from hashed/processed entropy.
3. ESP32 **sends the OTP to the backend server** via HTTP.
4. The backend **stores the OTP** temporarily and **sends it via SMS** to the user.
5. The user receives OTP, enters it into the frontend UI.
6. The backend **verifies the match** and responds with success/failure.

**🛡 Security Considerations**

* OTP is stored in memory and expires quickly.
* SMS API keys are secured via environment variables.
* Randomness is based on real-world sensor noise and time.