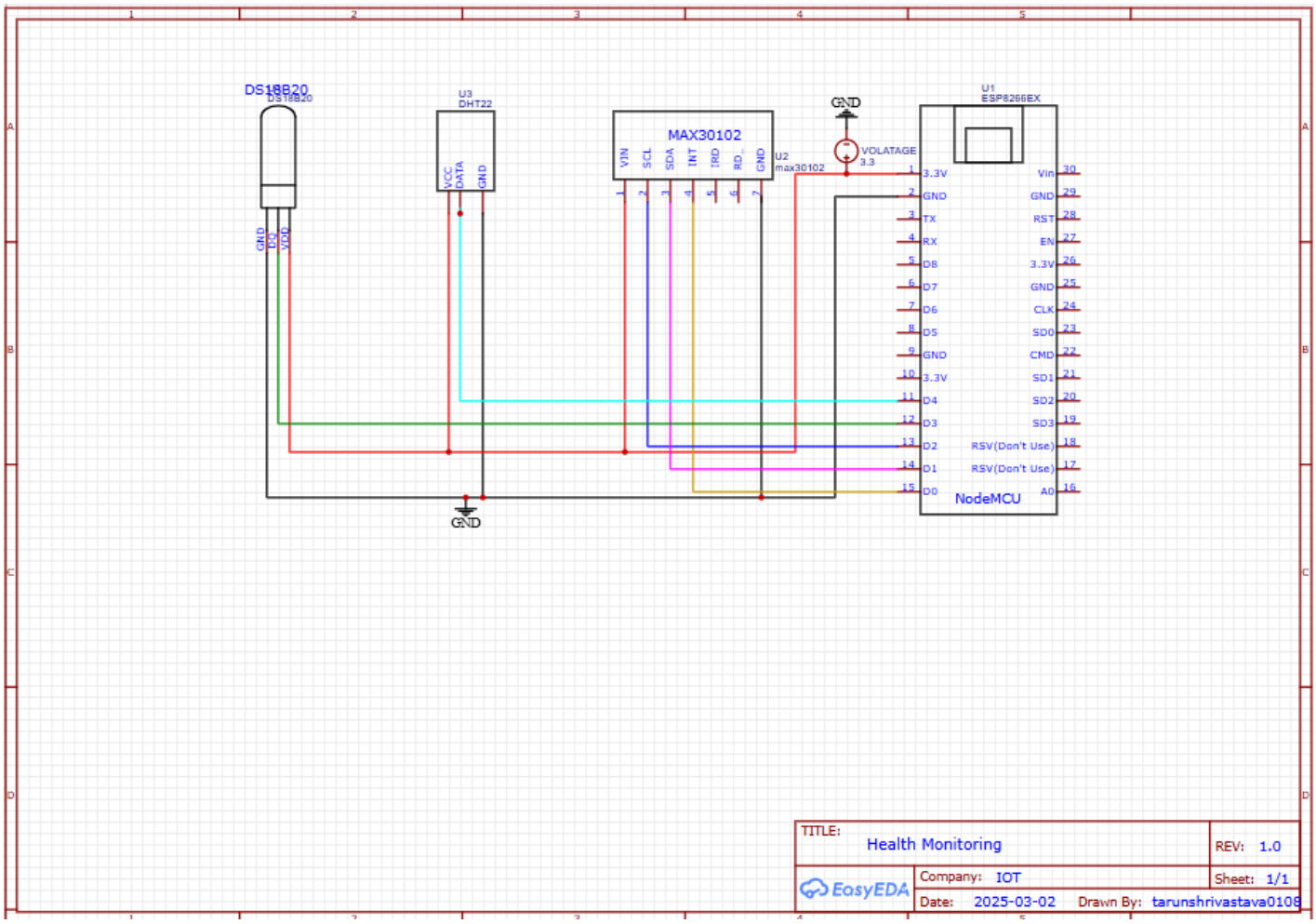
🩺 IoT-Based Health Monitoring System

A complete IoT-based health tracking ecosystem using NodeMCU (ESP8266), a FastAPI server for data handling, and a responsive web frontend built with Next.js. It is designed to collect patient body temperature and heart rate along with environment temperature and humidity data, log it to a backend database, and display it through intuitive visualizations. The system uses a proxy server to securely handle HTTP requests and is scalable for future mobile app support.

Circuit Diagram

The circuit uses the following components:

- NodeMCU (ESP8266) – core controller
- DHT11/DHT22 – for measuring temperature and humidity
- Power source (USB/adapter)
- Wires and breadboard



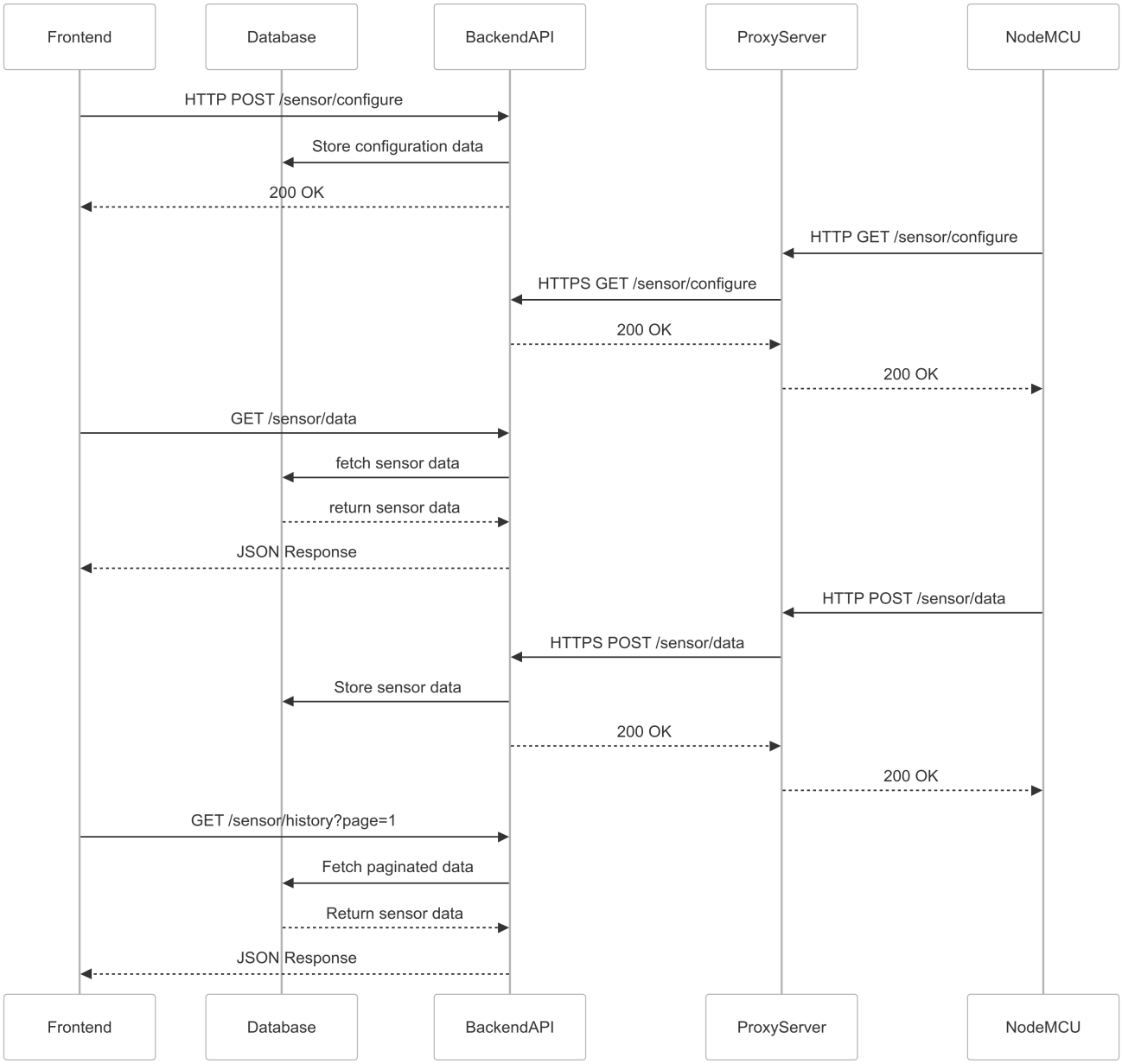
Worked with EasyEDA to design and simulate.

For more details refer to /resources/ folder and check out Info file.

🏗️ Architecture Overview

The architecture of the IoT Health Monitoring System is designed to ensure seamless data flow from sensor to screen, combining hardware, backend, and frontend components efficiently. At the heart of the system, a NodeMCU (ESP8266) collects vital health data from connected sensors and transmits it to a local proxy server, which converts insecure HTTP requests into HTTPS to securely interact with a FastAPI-based backend.

This backend handles data processing, storage using SQLAlchemy, and exposes robust RESTful APIs consumed by a Next.js frontend for real-time visualization and monitoring. The system is modular and scalable, with future plans to integrate a Flutter mobile app, extending accessibility and real-time alerting to smartphones while maintaining a consistent API-driven architecture.



Data Flow Summary

1. NodeMCU reads sensor data.
2. Sends HTTP data to Proxy Server on the local machine.
3. Proxy Server forwards it to the secure FastAPI backend.
4. FastAPI stores the data in the database.
5. Frontend fetches data using paginated and filtered GET requests.
6. Data is visualized in charts and tables with alerts.

NodeMCU Codebase

The main.ino file located in /codes/main/ is responsible for:

- Connecting to WiFi
- Receiving configuration data
- Reading sensor data (DHT11, DS18B20, MAX30102)
- Sending data to the proxy/local server at intervals
- Handling certificate validation or optionally bypassing HTTPS via a local proxy

Features:

- Basic retry mechanism
- Configurable server URL
- Continous vital montitoring
- Sending
- Easy debugging through serial monitor

Proxy Server

The proxy_server/ folder contains an HTTP proxy written in Node.js using http-proxy-middleware. This server:

- Accepts HTTP requests from the NodeMCU
- Forwards them securely to the HTTPS backend
- Solves SSL/certificate issues on the microcontroller side

Start with node proxy.js to run the server on localhost.

⚙ Backend - FastAPI Server

The server (in /server/) handles:

- Receiving sensor data via REST API
- Storing it in a relational database
- Providing endpoints to fetch paginated and filtered history
- Running a local proxy server to redirect HTTP requests from the ESP to HTTPS

Key Endpoints:

- POST /sensor/configure - Accepts initial configurations for nodemcu
- GET /sensor/configure - Fetches initial configurations
- POST /sensor/data – Accepts sensor readings
- GET /sensor/data – Fetches last sensor reading
- GET /sensor/history – Returns paginated, timestamp-filtered data

Includes custom pagination without using third-party libraries.

Documentation: https://healthmonitoring-production.up.railway.app/docs

Frontend - Next.js Dashboard

The web frontend (/frontend/) is built using:

- Next.js + Tailwind CSS
- Recharts for graphs
- Axios for API calls
- OpenRouter for ai diagnostics

Features:

- Live dashboard with updated metrics
- History section with 3 charts and logs
- Filter sensor history by date and time
- Pagination controls
- Ai Diagnostics based on recent sensor value and symptoms by user
- Modular and scalable codebase

Setup Instructions

1. Clone the repository:

```
git clone https://github.com/your-username/health-monitoring.git
```

2. Flash main.ino to your NodeMCU using Arduino IDE.
3. Run the FastAPI backend:

```
cd server
docker-compose up --build
```

4. Run the proxy server:

```
cd proxy_server
node proxy.js
```

5. Launch frontend:

```
cd frontend
npm install
npm run dev
```

Additional Resourecs

Ardunio Code: https://app.arduino.cc/sketches/d4ad097d-9feb-43bd-90d3-ff0d64bffc8c?view-mode=preview

NodeMCU ESP8266 package: http://arduino.esp8266.com/stable/package_esp8266com_index.json

MAX3010x library : https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library