Intelligent Home Monitoring System (IHMS)

Project Documentation

Emre Akilli, Juan Carlos Gomez, David Palacio, Carlos Diaz, Henry Garcia

Dr. Masoud Sadjadi | CIS 4951 Capstone II Project | Clean Technology

Click Here for Live Demo

1. Executive Summary

The Intelligent Home Monitoring System (IHMS) is a full-stack capstone prototype designed to help homeowners and small business owners take back control of their energy usage. The system presents connected-device activity through a clean dashboard, device-level analytics, runtime tracking, alerts, timers, and a simulated emergency killswitch. The main purpose is not only to display energy data, but to make that data understandable enough for users to act on it.

IHMS addresses a common real-world problem: energy usage is often invisible until a monthly utility bill arrives. Users may know that costs increased, but they usually do not know which device caused it, how long devices were active, or whether a pattern is abnormal. IHMS uses a clean technology theme to connect financial savings with energy awareness and environmentally responsible behavior.

2. Project Goals and Scope

- Provide an intuitive way to view device-level energy data.
- Help users save money by identifying waste, overuse, and inefficient behavior.
- Use data visualization to make energy usage understandable rather than technical.
- Support a clean technology vision focused on awareness, efficiency, and sustainability.
- Demonstrate a working full-stack architecture using a React frontend, Express backend, and PostgreSQL database.
- Preserve demo reliability through fallback data if the API or database becomes temporarily unavailable.

Scope note: the current version is a capstone prototype. It demonstrates the software architecture and user experience for energy monitoring, but it does not yet connect to physical smart plugs or real IoT hardware.

3. System Architecture

IHMS uses a layered full-stack architecture. The frontend is responsible for user interaction and visualization, the backend exposes API endpoints, and PostgreSQL stores device state. The database setup is reproducible through a SQL file in the root-level Database directory.

3.1 High-Level Architecture Diagram

User Browser
React + TypeScript Frontend **Dashboard, Device Cards, Charts, Timers, Alerts, Killswitch**
VITE_API_BASE_URL / HTTP Requests

Node.js + Express Backend API GET /api/devices | POST /api/devices | PATCH /api/devices/:id | DELETE /api/devices/:id
PostgreSQL Database devices table seeded from database/ihms_database.sql

3.2 Data Flow Diagram

User Action	Frontend State	API Wrapper	Express Route	PostgreSQL devices table

Example flow: a user adds a device -> AddDeviceView sends a POST request -> Express validates and inserts the record -> PostgreSQL stores the device -> DashboardView refetches or updates state -> the new device appears on the dashboard.

4. Major Components

Component	Technology / File	Purpose
Frontend Client	React, TypeScript, Vite, Tailwind CSS, Recharts	Provides authentication screens, dashboard, device cards, modals, charts, timer controls, add-device flow, settings, and notification UI.
Backend API	Node.js and Express	Exposes REST endpoints for device operations and connects to PostgreSQL. It also includes safe demo initialization to keep the app runnable.
Database	PostgreSQL	Stores device records including name, type, status, power, Wi-Fi capability, alert flags, custom timer, runtime, and timestamps.
API Wrapper	client/src/app/deviceApi.ts	Centralizes frontend API calls and helps separate UI components from backend request details.
Fallback Data Layer	client/src/app/mockDatabase.ts	Keeps the demo functional if the backend is unavailable, preserving presentation reliability.
Documentation and SQL Setup	database/ and documentation/	Supports reproducibility, installation, and user operation.

5. Key Features

- Dashboard summary cards showing total power, active devices, and total devices.
- Device cards that display name, type, on/off status, live power usage, timer indicators, and alert state.
- Device detail modal with current wattage, runtime, usage trend message, and chart tabs for week, month, six months, and year.
- Add Device page with a simple device name field, device type selector, and simulated pairing flow.
- Timer controls for selected device types to encourage planned usage and prevent devices from running indefinitely.
- Runtime tracking to show how long a device has been active.
- Debug controls for demo scenarios, including spike usage, overuse alerts, gaming alerts, device-on simulation, reset, and killswitch.

- Emergency killswitch that turns all devices off and resets active alerts, high-usage states, timers, and runtime values.

6. User Interface Evidence

The screenshots below show the current dashboard, add-device flow, device analytics modal, and demo controls used during presentation. These screens support the design goal of making energy usage readable, visual, and easy to act on.

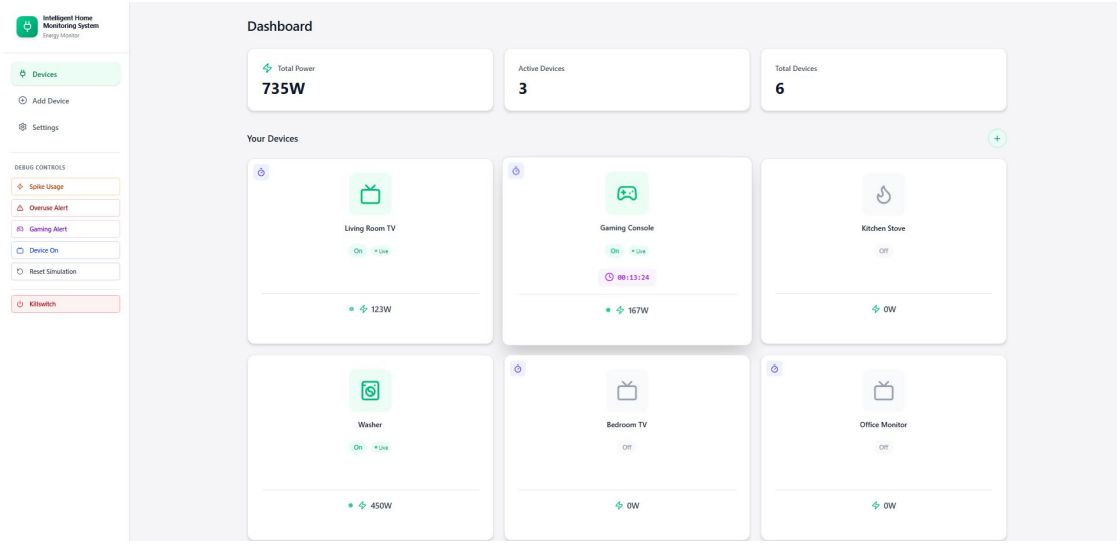

Figure 1. IHMS dashboard and device monitoring interface.

Living Room TV

On

120W

Current power consumption

00:55:57

Running time

Usage increased by 12% this week

Slightly higher than your typical usage

Energy Usage

+12%

Week | Month | 6 Months | Year

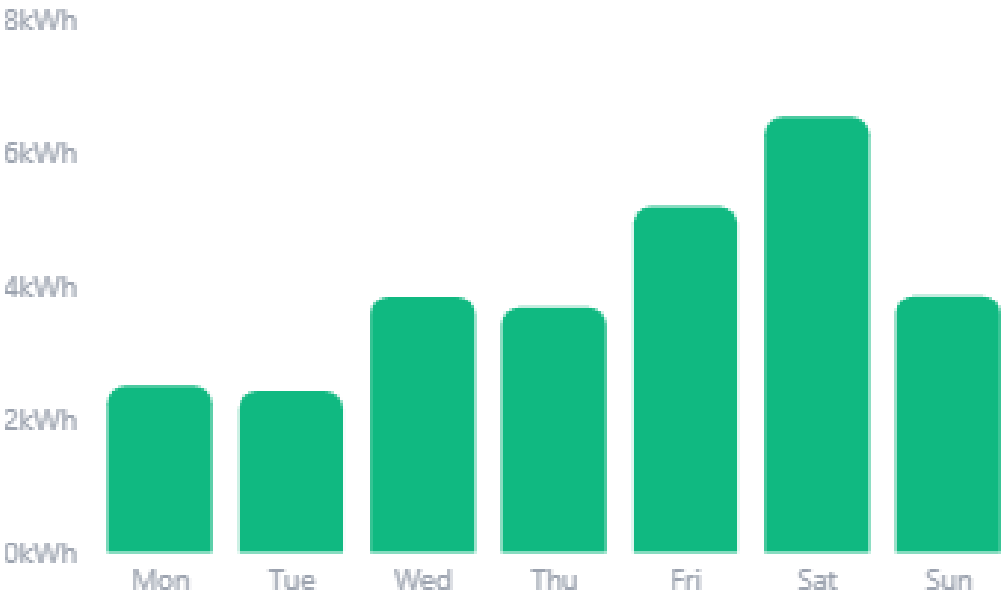

Figure 2. Device detail view with runtime and chart analytics.

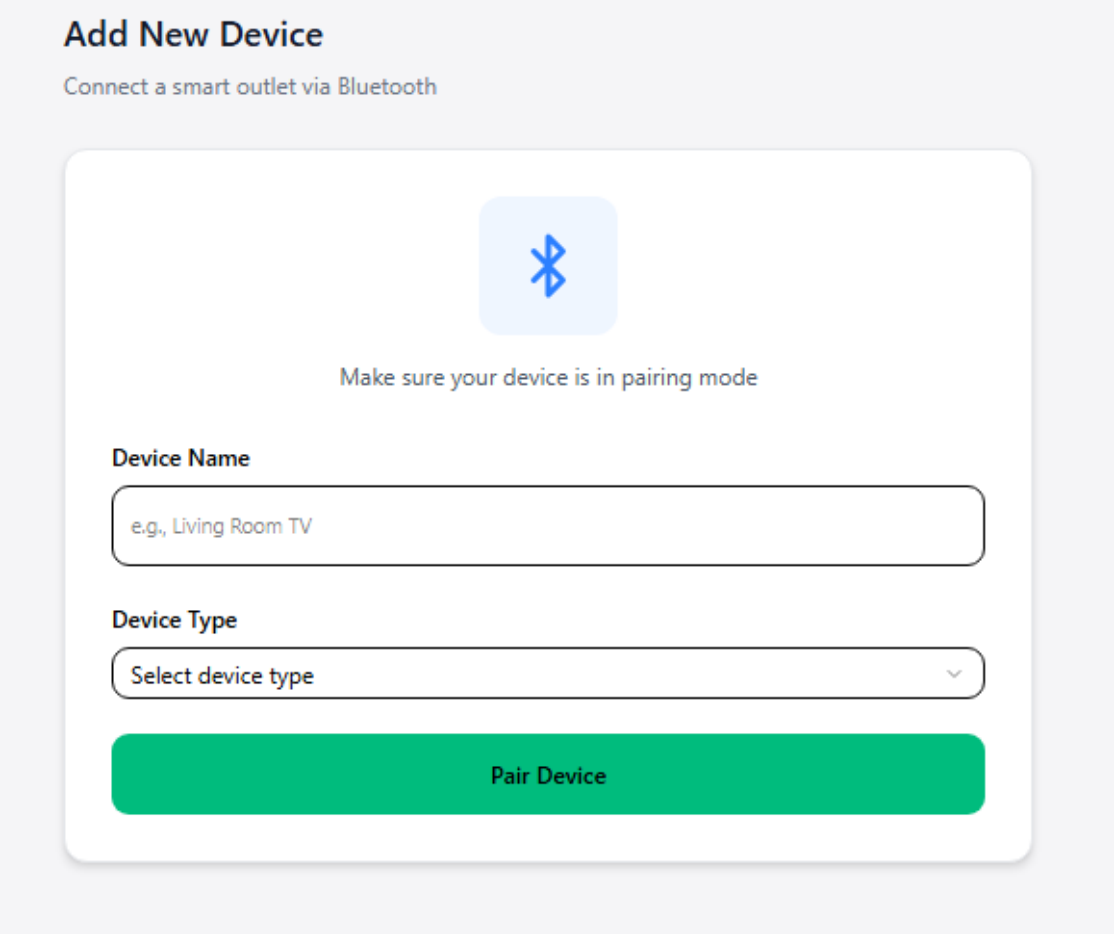

Figure 3. Add-device and pairing flow.

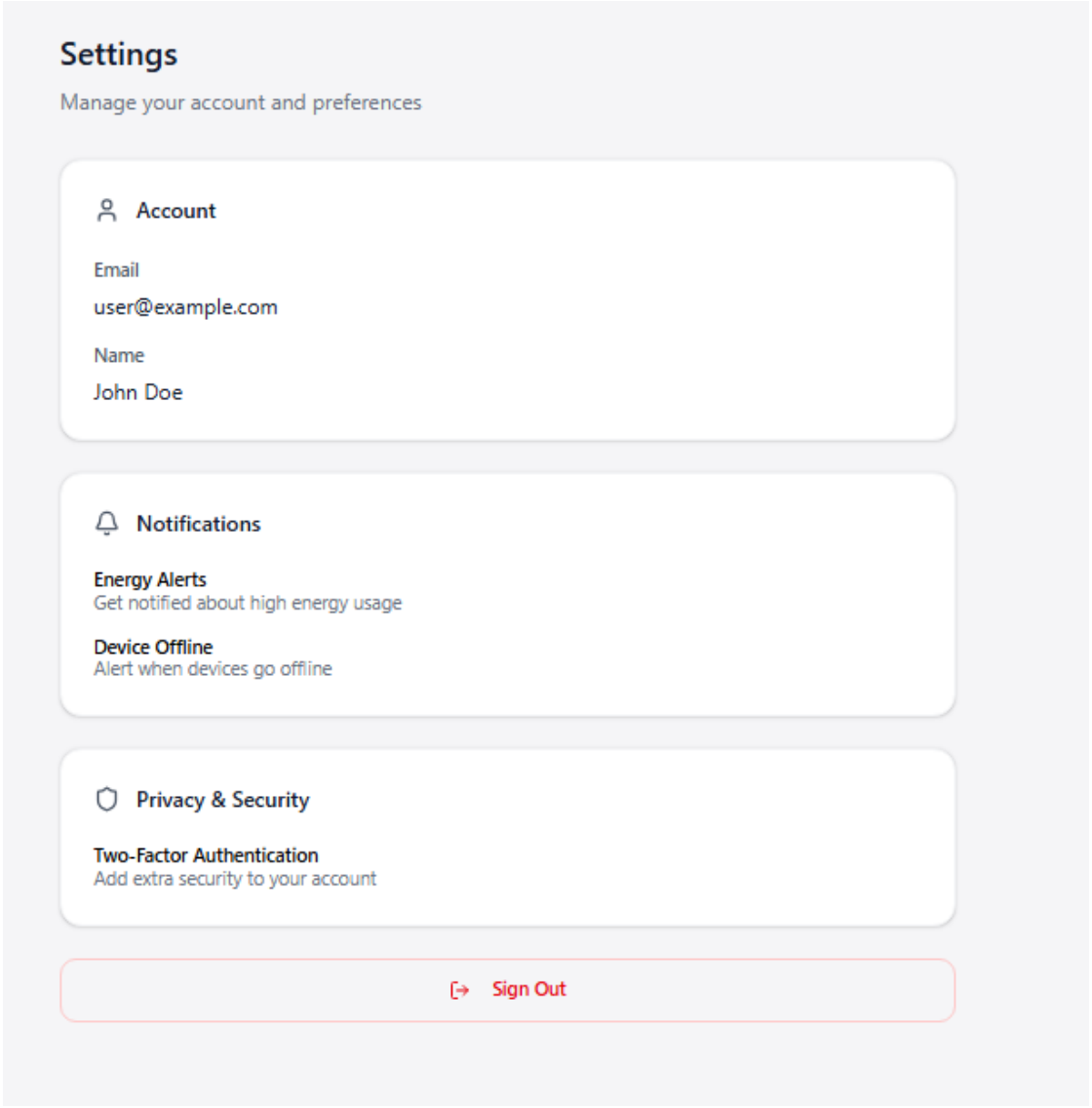

Figure 4. Timer, alert, and control-oriented interface elements.

7. Setup and Execution Instructions

7.1 Prerequisites

- Node.js 20 or newer
- npm 10 or newer
- Git
- PostgreSQL 14 or newer

7.2 Clone Repository

```
git clone https://github.com/eakillidev/ihms-demo
cd ihms
```

7.3 Install Backend Dependencies

```
cd server
npm install
```

7.4 Install Frontend Dependencies

```
cd client
npm install
```

7.5 Environment Configuration

Create server/.env for backend configuration:

```
DATABASE_URL=postgresql://username:password@host:port/database
PORT=5000
```

Create client/.env for frontend configuration:

```
VITE_API_BASE_URL=http://localhost:5000
```

7.6 Database Setup

Create a local PostgreSQL database named ihms, then run the SQL setup file from the project root:

```
createdb ihms
psql -U postgres -d ihms -f database/ihms_database.sql
```

The SQL file drops and recreates the devices table and inserts the current demo records. Runtime credentials are intentionally not included in source control.

7.7 Run Locally

Start backend:

```
cd server
npm start
```

Test API:

```
http://localhost:5000/api/devices
```

Start frontend in a second terminal:

```
cd client
npm run dev
```

8. Design Decisions and Trade-Offs

Decision	Rationale	Trade-Off
Frontend-first usability	The project prioritizes a polished dashboard and intuitive user flow because energy tools must be understandable to be useful.	Some advanced backend and hardware integrations are deferred.
Fallback demo data	The frontend can still show demo data if the backend is unavailable, protecting the live presentation.	This means some displayed data may be simulated rather than fully persistent.
React + TypeScript	TypeScript improves maintainability and helps catch mistakes during development.	Strict build errors required some pragmatic fixes to prioritize demo delivery.
Node.js + Express API	Express is lightweight and easy to extend for REST endpoints.	A larger production system may need WebSockets, queues, and stronger service separation.
PostgreSQL	Relational storage is reliable and	Long-term energy history may be

	easy to document with a reproducible SQL script.	better served by time-series storage or aggregation strategies.
Render deployment	Render keeps frontend, backend, and database deployment simple for a capstone demo.	Higher-scale production deployments may require more advanced infrastructure and observability.
Simulated device telemetry	Simulation allowed the team to focus on UX, architecture, and data flow without hardware delays.	Real hardware integration remains a future requirement for production readiness.

9. Database Design

The current database centers on the devices table. The schema mirrors the frontend device model and backend API fields so data can flow between layers with minimal transformation.

```
CREATE TABLE devices (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  type TEXT NOT NULL,
  status TEXT NOT NULL DEFAULT 'off' CHECK (status IN ('on', 'off')),
  power INTEGER NOT NULL DEFAULT 0,
  has_wifi BOOLEAN NOT NULL DEFAULT true,
  has_alert BOOLEAN NOT NULL DEFAULT false,
  is_high_usage BOOLEAN NOT NULL DEFAULT false,
  custom_timer INTEGER,
  runtime INTEGER NOT NULL DEFAULT 0,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

10. Security Considerations

Because IHMS may eventually involve household or business device data and control actions, security is a core future consideration. The current prototype demonstrates architecture and flow, while a production version would require stronger security controls.

- Authentication and authorization to ensure only approved users can view or control devices.
- Encrypted communication through HTTPS between frontend, backend, and deployed services.
- Input validation for all API requests, especially device creation and updates.
- Least-privilege access between users, devices, backend services, and database resources.
- Secret management through environment variables rather than committed files.
- Audit logging for sensitive control actions such as the killswitch.

11. User Manual Summary

1. Log in or sign up using the demo authentication screen.
2. Use the dashboard to review total power, active devices, and total devices.
3. Click a device card to open detailed analytics, including runtime and usage charts.
4. Use chart tabs to view week, month, six-month, or yearly usage patterns.
5. Add a new device from the Add Device page by entering a name and selecting a device type.

6. Set a timer for compatible devices to limit runtime.
7. Review alerts when unusual activity is detected or simulated.
8. Use the killswitch when all devices should be shut off quickly during a demo or emergency scenario.

12. Shortcomings and Wishlist

12.1 Current Shortcomings

- The project currently uses simulated or manually updated device readings.
- No physical smart plug, IoT sensor, or appliance hardware is connected yet.
- Authentication is demo-level and not production-ready.
- Historical usage charts may use randomized demo data rather than real long-term measurements.
- The killswitch updates application and database state but does not control physical electrical devices.
- Advanced user roles, households, organizations, and permissions are not implemented.
- Real-time streaming is not yet implemented with WebSockets or message queues.
- Cost estimates based on local electricity rates are not implemented yet.

12.2 Wishlist and Future Improvements

- Integrate real IoT smart plugs or energy sensors.
- Add secure production authentication and user account separation.
- Store long-term energy telemetry using time-series tables or aggregation strategies.
- Add customizable alert thresholds and energy budgets.
- Add cost estimates based on electricity rates and billing periods.
- Add AI-driven recommendations for reducing waste and identifying abnormal patterns.
- Support mobile notifications, email alerts, SMS alerts, or push notifications.
- Add room-based organization and multi-location support for business users.
- Provide real remote shutoff through compatible smart home hardware.
- Integrate with larger smart home ecosystems such as Amazon Alexa, Google Home, or Apple HomeKit.

13. Business and Clean Technology Perspective

IHMS has a practical business incentive: helping users save money. Energy awareness becomes more valuable when it directly connects to costs. By showing users which devices are running, how long they have been active, and where usage may be abnormal, IHMS gives users information they can act on immediately.

From a clean technology perspective, IHMS supports more conscious energy behavior. The system is not positioned as a replacement for the electrical grid or utility infrastructure; instead, it gives users visibility at the point where behavior can change. At scale, better awareness and lower waste can contribute to reduced demand and more efficient energy use.

The long-term business vision is to position IHMS as a platform that could integrate into a larger smart home ecosystem. Potential exit paths include acquisition or partnership with technology companies interested in expanding home automation, energy analytics, and consumer-facing sustainability tools.

14. Conclusion

The Intelligent Home Monitoring System demonstrates how software can make energy usage more transparent, understandable, and manageable. The project combines a polished user interface, practical backend API, reproducible database setup, and a clear clean-tech vision. While the current version is a prototype, it establishes a strong foundation for future hardware integration, real-time telemetry, security improvements, and scalable energy analytics.