# Prueba Técnica Full-Stack Smart City – Sensor de Temperatura (con Autenticación)

Frontend + Backend + Base de Datos + Auth básica

Stack Node:
React + TypeScript (preferible) o JavaScript. Valorables vite
Backend: API Routes. Valorables nestjs

Es posible utilizar otro stack basado en javascript tanto para backend como para frontend.

Base de datos: Mysql o PostgreSQL o mongodb

## 1) Objetivo

Construir una mini plataforma web para gestionar sensores de temperatura, ingerir lecturas desde fuentes heterogéneas y visualizar lecturas + histórico de ingestas. Además, implementar autenticación básica para proteger la aplicación.

## 2) Autenticación

Implementar un flujo simple de autenticación con sesión en cookie.

- Páginas: /login y /logout (opcional: /register).
- Protección de rutas: /sensors, /sensors/[id] y /ingestions deben requerir sesión.
- Sesión: cookie httpOnly (recomendado) si lo prefieres.
- Passwords: hashing jwt (recomendado). Si no se llega, documentar en README cómo se haría.
- Autorización mínima: cualquier usuario autenticado puede gestionar sensores e ingestas (bonus: solo el creador edita/borra).

Endpoints sugeridos (Auth):

- POST /api/auth/login (email, password)
- POST /api/auth/logout
- GET /api/auth/me (usuario actual o 401)
- Opcional: POST /api/auth/register

## 3) Gestión de Sensores

Pantalla/sensors con listado, creación, edición, activación/pausa y borrado de sensores.
Campos mínimos del sensor:

- name (requerido, ≥ 3)
- sensorCode (requerido, único, ≥ 3)
- type: HTTP_POLL | MANUAL_UPLOAD
- status: active | paused
- url (solo para HTTP_POLL, URL válida)

## 4) Ingesta de Temperatura

Pantalla /ingestions con histórico de ejecuciones. Debe existir una acción “Ingestar ahora”. Formatos soportados:

Formato A:
[{"sensorCode":"TEMP-001","ts":"2026-01-30T10:00:00Z","valueC":21.4}]

Formato B:
{"deviceId":"TEMP-001","data":[{"time":1706600000,"temp":21.4}]}

Reglas de mapeo:

- Guardar solo si coincide con el sensor configurado (sensorCode/deviceId).
- A: ts ISO → Date. B: time unix seconds → Date.
- Guardar temperatura en °C.
- Payload inválido → IngestionRun con status=error y mensaje claro.

## 5) Persistencia (Base de Datos)

Modelos mínimos esperados:

- User: id, email(unique), passwordHash, createdAt
- Sensor: id, name, sensorCode(unique), type, status, url?, createdAt, updatedAt
- IngestionRun: id, sensorId(FK), startedAt, finishedAt, status, recordsProcessed, errorMessage?
- TemperatureReading: id, sensorId(FK), timestamp, valueC Se valora: unique (sensorId, timestamp) o estrategia anti-duplicados.

## 6) Backend (API)

Endpoints sugeridos (no excluye a otros necesarios):

- GET /api/sensors
- POST /api/sensors
- GET /api/sensors/:id
- PATCH /api/sensors/:id
- DELETE /api/sensors/:id
- POST /api/sensors/:id/ingest
- GET /api/sensors/:id/readings?limit=20

Requisitos:

- Validación de entrada.
- Errores consistentes (400/401/404/500).
- Rutas API (excepto /api/auth/*) requieren sesión.

## 7) Mock endpoints internos (para HTTP_POLL)

- GET /api/mock/temp-format-a
- GET /api/mock/temp-format-b

## 8) Entregables

- Repositorio o ZIP.
- README: ejecución, env vars, migraciones/seed, decisiones, mejoras futuras.

Uso de herramientas de IA
Se permite el uso de herramientas de IA (por ejemplo, ChatGPT, Copilot, etc.) durante la realización de la prueba.
El objetivo no es evitar su uso, sino evaluar la capacidad del candidato para:

- Formular buenas preguntas.
- Entender las respuestas.
- Adaptar el resultado a su contexto
- Mantener coherencia y calidad en el código final.

El uso de IA no debe sustituir la comprensión del código entregado.

Cualquier decisión frente a arquitectura de la solución, stack utilizado y elementos empleados, se debe explicar en el README.

PAVAPARK

## Valorable:

- Clean architecture (inyección de dependencias)
- Orm (sequelize, prisma, typeorm…)
- Docker