Projet final — Plateforme de monitoring IoT

Contexte

Vous allez développer une **API backend** pour une plateforme de gestion d'objets connectés dans un contexte de maison intelligente (smart home).

L'API permet à des **devices** (capteurs) de s'enregistrer, transmettre des données de télémétrie, et à un **administrateur** de gérer la flotte et consulter les mesures.

Modalités

Aspect	Détail
Équipes	Binômes (2 étudiants)
Travail	Sur votre temps personnel (hors cours)
Rendu	Repository GitHub fonctionnel
Évaluation	Soutenance devant jury externe
Date limite	*À confirmer*

Historique Git

L'historique de votre repository doit montrer :

- Des commits réguliers (pas un seul commit final)
- Des contributions des **deux membres** du binôme
- Des messages de commit clairs

Un historique suspect sera questionné en soutenance.

Utilisation d'IA

L'utilisation d'outils d'IA (Claude, ChatGPT, Copilot...) est **autorisée**, mais vous devez être capables d'**expliquer chaque ligne de code** lors de la soutenance.

Stack technique

Technologie	Usage
Node.js 24 LTS	Runtime
pnpm	Gestionnaire de packages
Express	Framework backend
TypeScript	Typage statique

Technologie	Usage
MongoDB	Base de données (via Docker Compose)
Zod	Validation des entrées

Acteurs et authentification

Device

Un capteur IoT qui s'enregistre et envoie des mesures.

Authentification : header `x-device-key` contenant la clé générée lors de l'enregistrement.

Admin

L'administrateur qui gère les devices et consulte les données.

Authentification : header `x-api-key` contenant une clé statique définie dans `.env`.

Types de devices

Deux types de capteurs sont supportés :

Type	Données de télémétrie
`climate`	`temperature` (number, °C), `humidity` (number, %)
`presence`	`motion` (boolean)

Tous les devices peuvent optionnellement transmettre leur niveau de `battery` (number, %).

Flux d'enregistrement

1. Le device envoie `POST /devices/register` avec son `deviceId` (uuid généré par le device), `name` et `type`
2. L'API génère une `deviceAccessKey` et retourne le tout avec `status: "pending"`
3. Le device stocke sa `deviceAccessKey` et poll (= appelle à intervalle régulier) `GET /devices/me` pour connaître son status
4. L'admin consulte `GET /admin/devices?status=pending` et approuve via `POST /admin/devices/:id/approve`
5. Le device voit son status passer à `active` et peut envoyer de la télémétrie

---

Règles métier

Situation	Comportement attendu

Situation	Comportement attendu
Device déjà enregistré refait `POST /devices/register`	Ancienne `deviceAccessKey` révoquée, nouvelle générée, status → `pending`
Device `pending` tente `POST /telemetry`	Erreur `403 Forbidden`
Device `revoked` tente n'importe quelle action	Erreur `403 Forbidden`
`deviceAccessKey` invalide ou manquante	Erreur `401 Unauthorized`
Admin approve/revoke sur device inexistant	Erreur `404 Not Found`
Admin approuve un device `pending`	`200 OK` (status → active)
Admin approuve un device `active`	`200 OK` (idempotent)
Admin approuve un device `revoked`	`200 OK` (status → active, réactivation)
Admin révoque un device `pending`	`200 OK` (status → revoked)
Admin révoque un device `active`	`200 OK` (status → revoked)
Admin révoque un device `revoked`	`200 OK` (idempotent)

Endpoints API

Public

Méthode	Route	Description
`GET`	`/ping`	Health check → `{ "ok": true }`

Device (auth: `x-device-key`)

Méthode	Route	Description
`POST`	`/devices/register`	Demande d'accès (sans auth x-device-key)
`GET`	`/devices/me`	Consulter son status
`POST`	`/telemetry`	Envoyer une mesure

Admin (auth: `x-api-key`)

Méthode	Route	Description
`GET`	`/admin/devices`	Liste des devices (filtre `?status=`)
`GET`	`/admin/devices/:id`	Détail d'un device
`POST`	`/admin/devices/:id/approve`	Approuver un device

Méthode	Route	Description
POST	/admin/devices/:id/revoke	Révoquer un device
GET	/admin/devices/:id/telemetry	Mesures paginées (?limit=&offset=)
GET	/admin/devices/:id/telemetry/latest	Dernière mesure
GET	/admin/devices/:id/stats	Stats agrégées (?from=&to=)

Spécifications détaillées

POST /devices/register

Requête (sans authentification) :

```
{
  "deviceId": "550e8400-e29b-41d4-a716-446655440000",
  "name": "Salon - Température",
  "type": "climate"
}
```

Réponse `201 Created` :

```
{
  "deviceId": "550e8400-e29b-41d4-a716-446655440000",
  "deviceAccessKey": "7c9e6679-7425-40de-944b-e07fc1f90ae7",
  "status": "pending"
}
```

Erreurs :

- `400 Bad Request` : champs manquants ou invalides

GET /devices/me

Header : `x-device-key: <deviceAccessKey>`

Réponse `200 OK` :

```
{
  "deviceId": "550e8400-e29b-41d4-a716-446655440000",
  "name": "Salon - Température",
  "type": "climate",
  "status": "active"
}
```

Erreurs :

- `401 Unauthorized` : clé manquante ou invalide
- `403 Forbidden` : device révoqué

`POST /telemetry`

Header : `x-device-key: <deviceAccessKey>`

Requête (climate) :

```
{
  "timestamp": "2026-01-10T10:00:00Z",
  "temperature": 22.5,
  "humidity": 45,
  "battery": 98
}
```

Requête (presence) :

```
{
  "timestamp": "2026-01-10T10:00:00Z",
  "motion": true,
  "battery": 72
}
```

Réponse `201 Created` : `{ "ok": true }`

Erreurs :

- `400 Bad Request` : données invalides (mauvais type, champs manquants)
- `401 Unauthorized` : clé manquante ou invalide
- `403 Forbidden` : device non actif (pending ou revoked)

`GET /admin/devices`

Header : `x-api-key: <apiKey>`

Query params :

Param	Type	Description
`status`	string (optionnel)	Filtrer par status (`pending`, `active`, `revoked`)

Réponse `200 OK` :

```
[
  {
    "deviceId": "550e8400-...",
    "name": "Salon - Température",
    "type": "climate",
    "status": "active",
    "createdAt": "2026-01-10T09:00:00Z"
  }
]
```

GET /admin/devices/:id/telemetry

Header : `x-api-key: <apiKey>`

Query params :

Param	Type	Défaut	Description
`limit`	number	20	Nombre de résultats (max 100)
`offset`	number	0	Décalage

Réponse `200 OK` :

```
{
  "data": [
    {
      "timestamp": "2026-01-10T10:00:00Z",
      "temperature": 22.5,
      "humidity": 45,
      "battery": 98
    }
  ],
  "pagination": {
    "total": 248,
    "limit": 20,
    "offset": 0
  }
}
```

Tri par défaut : `timestamp` descendant (plus récent en premier).

GET /admin/devices/:id/stats

Header : `x-api-key: <apiKey>`

Query params :

Param	Type	Description
`from`	ISO date	Date de début
`to`	ISO date	Date de fin

Réponse `200 OK` (climate) :

```
{
  "from": "2026-01-01T00:00:00Z",
  "to": "2026-01-31T23:59:59Z",
  "count": 248,
  "temperature": { "min": 18.2, "max": 26.5, "avg": 21.8 },
  "humidity": { "min": 35, "max": 72, "avg": 48.3 }
}
```

Réponse `200 OK` (presence) :

```
{
  "from": "2026-01-01T00:00:00Z",
  "to": "2026-01-31T23:59:59Z",
  "count": 156,
  "motionDetected": 42
}
```

Modèle de données

Collection `devices`

Champ	Type	Description
`_id`	ObjectId	Généré par MongoDB
`deviceId`	string	UUID fourni par le device
`name`	string	Nom lisible
`type`	string	`"climate"` ou `"presence"`
`deviceAccessKey`	string	UUID généré par l'API
`status`	string	`"pending"`, `"active"`, `"revoked"`
`createdAt`	Date	Date d'enregistrement

Collection `telemetry`

Champ	Type	Description

Champ	Type	Description
`_id`	ObjectId	Généré par MongoDB
`deviceId`	string	Référence au device
`timestamp`	Date	Timestamp de la mesure
`temperature`	number	(climate uniquement)
`humidity`	number	(climate uniquement)
`motion`	boolean	(presence uniquement)
`battery`	number	Optionnel

Index recommandé

```
db.telemetry.createIndex({ deviceId: 1, timestamp: -1 });
```

Scripts fournis

Des scripts sont fournis pour tester votre API :

Script	Usage
`pnpm simulate:device`	Simule un device (register → poll → telemetry)
`pnpm admin:approve-device <deviceId>`	Approuve un device pending
`pnpm admin:revoke-device <deviceId>`	Révoque un device

Ces scripts doivent fonctionner avec votre API une fois celle-ci implémentée.

Livrables

Repository GitHub

- [ ] Code source complet
- [ ] Historique Git avec contributions des 2 membres
- [ ] Pas de `node_modules` commité
- [ ] Pas de secrets (`.env`) commités

Fichiers requis

- [ ] `README.md` — Instructions de lancement
- [ ] `docker-compose.yml` — MongoDB
- [ ] `.env.example` — Variables d'environnement requises

README.md

Votre README doit contenir :

1. **Prérequis** (Node.js, pnpm, Docker)
2. **Installation** (commandes exactes)
3. **Configuration** (variables d'environnement)
4. **Lancement** (commandes exactes)

Reproductibilité

Critère essentiel : le jury doit pouvoir lancer votre projet avec :

```
git clone <votre-repo>
cp .env.example .env
pnpm install
docker compose up -d
pnpm start:dev
```

Ensuite, `curl http://localhost:3000/ping` doit retourner `{ "ok": true }`.

Si le projet ne démarre pas → pénalité sur la note.

Barème (20 points)

Catégorie	Points
Fonctionnel — Core	12
Qualité technique	6
Reproductibilité	2

Détail "Fonctionnel — Core" (12 pts)

Élément	Points
`POST /devices/register` + règle ré-enregistrement	2
`GET /devices/me`	0.5
`POST /telemetry` (2 types validés)	2.5
Auth device (`x-device-key`, gestion 401/403)	1
`GET /admin/devices` avec filtre status	1
`GET /admin/devices/:id`	0.5
`POST .../approve` + `.../revoke` (idempotent)	2
`GET .../telemetry` (paginé)	1

Élément	Points
`GET .../telemetry/latest` + `.../stats`	1
Auth admin (`x-api-key`)	0.5

Détail "Qualité technique" (6 pts)

Critère	Attendu
Validation Zod (ou autre)	Toutes les entrées API validées
Gestion d'erreurs	Format cohérent, codes HTTP appropriés
Architecture	Routes, controllers, repositories séparés
Code lisible	Nommage clair, pas de code mort
TypeScript strict	Types explicites, pas de `any`

Détail "Reproductibilité" (2 pts)

Critère	Points
Lancement en une commande	1
README clair et complet	1

Structure recommandée

```
src/
├── app.ts                # Config Express
├── server.ts             # Point d'entrée
├── config.ts             # Variables d'env validées
├── db.ts                 # Connexion MongoDB
├── errors/
│   └── http-error.ts     # Classe HttpError
├── middlewares/
│   ├── auth.middleware.ts # Auth device/admin
│   ├── validate.middleware.ts
│   └── error.middleware.ts
├── routes/
│   ├── index.ts
│   ├── ping.routes.ts
│   ├── devices.routes.ts
│   ├── telemetry.routes.ts
│   └── admin.routes.ts
├── controllers/
│   ├── ping.controller.ts
│   ├── devices.controller.ts
│   ├── telemetry.controller.ts
│   └── admin.controller.ts
```

```
├── repositories/
│   ├── devices.repository.ts
│   └── telemetry.repository.ts
├── schemas/                # Schémas Zod
│   ├── device.schema.ts
│   └── telemetry.schema.ts
└── types.ts
```

Checklist avant rendu

Fonctionnel

- [ ] `GET /ping` → `{ "ok": true }`
- [ ] `POST /devices/register` crée un device pending
- [ ] Ré-enregistrement révoque l'ancienne clé
- [ ] `GET /devices/me` retourne le status
- [ ] `POST /telemetry` fonctionne pour climate ET presence
- [ ] Device pending/revoked → 403 sur telemetry
- [ ] `GET /admin/devices` liste les devices
- [ ] Filtre `?status=` fonctionne
- [ ] `POST .../approve` et `.../revoke` fonctionnent
- [ ] Comportement idempotent
- [ ] `GET .../telemetry` avec pagination
- [ ] `GET .../telemetry/latest` retourne la dernière mesure
- [ ] `GET .../stats` retourne les agrégations

Technique

- [ ] Validation Zod sur toutes les entrées
- [ ] Gestion cohérente des erreurs (401, 403, 400, 404)
- [ ] Architecture propre (routes/controllers/repositories)
- [ ] Pas de `any` dans le code TypeScript
- [ ] ESLint passe sans erreur

Livrables

- [ ] README.md complet
- [ ] .env.example présent
- [ ] docker-compose.yml fonctionnel
- [ ] Pas de node_modules ni .env commités
- [ ] Historique Git avec contributions des 2 membres

Ressources

- **Base du projet** : https://github.com/stephane-ruhlmann/esgi-moc-projet-final-node-mongodb
- **Documentation Express** : https://expressjs.com
- **Documentation MongoDB Driver** : https://mongodb.github.io/node-mongodb-native/

- **Documentation Zod** : https://zod.dev

Bon courage ! 🚀