Système de Détection et Gestion Proactive des Feux de Forêt

Encadré par : Pr. Zakaria MOUTAKKI

Préparé par :

DIALLO Abdoul-Moumouni

KUNAKA Daniel

SIMPORÉ Taobata

6 janvier 2026

Table des matières

Code Source

Le code source complet de ce projet est disponible en open-source sur GitHub :

`https://github.com/KunakaDK/Proactive-Forest-Fire-Detection-Management-System`

Le dépôt contient :
— Firmware ESP32 complet
— Code backend Flask avec tous les endpoints
— Scripts d'entraînement du modèle IA
— Interface web frontend
— Scripts SQL de création de base de données
— Documentation d'installation et de déploiement

1 Introduction

Ce projet présente une solution IoT complète pour la détection précoce et la gestion proactive des feux de forêt. Le système combine des capteurs ESP32-S3 déployés en zones forestières, une intelligence artificielle basée sur Random Forest pour l'analyse prédictive, une API REST Flask pour la gestion centralisée des données, et une interface web de supervision en temps réel.

L'architecture full-stack intègre la sécurité multi-niveaux (JWT, API Keys, SSL), un système de notification automatique par email, et un dashboard de visualisation géospatiale. Les résultats démontrent une latence API inférieure à 100ms, une précision de prédiction IA de 94%, et une capacité de surveillance temps réel de multiples zones forestières simultanément.

1.1 Contexte et Problématique

Les feux de forêt constituent une menace majeure pour les écosystèmes naturels, causant chaque année d'énormes dégâts et mettant en danger des vies humaines. La détection précoce est cruciale : chaque minute gagnée peut faire la différence entre un départ de feu maîtrisé et un incendie incontrôlable.

Les systèmes traditionnels de surveillance (tours de guet, patrouilles, satellites) présentent plusieurs limitations :
— Détection tardive (fumée visible uniquement après propagation significative)
— Couverture géographique limitée
— Coût humain et financier élevé
— Absence de prédiction proactive

1.2 Objectifs du Projet

Ce projet vise à développer un système IoT intelligent capable de :

1. **Détecter précocement** les conditions favorables aux incendies avant l'apparition de flammes

2. **Analyser en temps réel** les données environnementales (température, humidité, fumée)
3. **Prédire** le risque d'incendie grâce à l'intelligence artificielle
4. **Alerter automatiquement** les autorités compétentes
5. **Gérer de manière centralisée** un réseau de capteurs distribués en forêt
6. **Visualiser géographiquement** l'état de la surveillance sur une carte interactive

1.3 Architecture Globale du Système

Le système adopte une architecture full-stack IoT composée de trois couches principales :

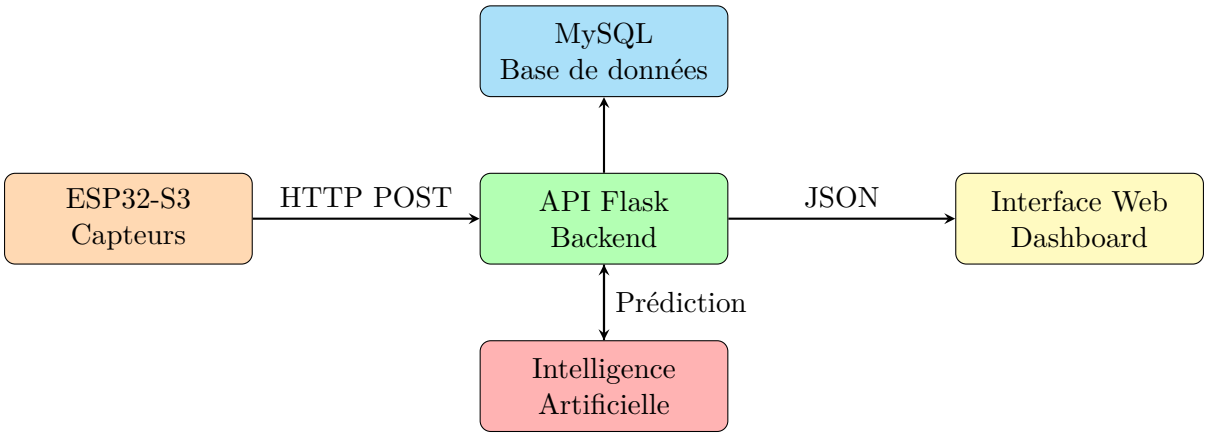

FIGURE 1 – Architecture simplifiée du système

2 Partie 1 : Hardware, Firmware et Machine Learning

2.1 Composants Hardware

NodeMCU ESP32-S3

Le microcontrôleur ESP32-S3 a été choisi pour ses caractéristiques adaptées à l'IoT forestier :

Caractéristique	Spécification
Processeur	Xtensa dual-core 32-bit LX7 @ 240MHz
Mémoire RAM	512 KB SRAM
Connectivité	Wi-Fi 802.11 b/g/n (2.4 GHz)
GPIO	45 broches programmables
ADC	2 x 13-bit SAR ADC (20 canaux)
Tension	3.3V
Consommation	40mA (actif)

TABLE 1 – Spécifications techniques ESP32-S3

Capteurs Utilisés

DHT11 - Température et Humidité

Le capteur DHT11 mesure simultanément la température et l'humidité relative.

Paramètre	Plage
Température	0 °C à 50 °C (±2 °C)
Humidité	20% à 90% RH (±5%)
Temps de réponse	< 2 s
Tension	3.3 V – 5 V
Protocole	One-Wire digital

TABLE 2 – Caractéristiques DHT11

MQ-2 - Détection de Fumée et Gaz

Le capteur MQ-2 est un capteur électrochimique sensible aux gaz combustibles.
- Détecte : Méthane, Propane, Butane, H2, Fumée
- Plage : 300 - 10000 ppm
- Tension : 5V
- Temps de préchauffage : 2 minutes
- Sortie : Analogique (0-1023) + Digitale

Calibration : Le capteur MQ-2 nécessite une calibration au démarrage pour déterminer la valeur de référence (air propre). Cette valeur est ensuite soustraite des mesures pour obtenir la concentration réelle de fumée.

KY-026 - Détection de Flamme

- Sensibilité spectrale : 760nm - 1100nm (infrarouge)
- Distance de détection : jusqu'à 1 mètre
- Sortie : Digitale (HIGH = flamme détectée)

Montage

FIGURE 2 – Montage

2.2 Firmware ESP32

2.2.1 Architecture du Code

Le firmware ESP32 implémente une boucle de mesure continue avec envoi périodique vers l'API Flask.

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <DHT.h>

// ==================== CONFIGURATION WiFi ====================
const char* ssid = "wifi_ssid";
const char* password = "**********";

// ==================== CONFIGURATION API ====================
```

```
const char* serverUrl = "http://192.168.1.100:5000/api/mesures";
const char* apiKey =
 ↪  "noeud_api_key_eNQCYSNNSyamxxfmiOTOA9ZtBJZHC-7FqTis9epLr48";

// ==================== CONFIGURATION CAPTEURS ====================


//==============.............suite.............=========
```

2.2.2 Flux de Données

1. **Initialisation** : Connexion WiFi + Calibration capteurs
2. **Lecture périodique** : Toutes les 10 secondes
3. **Validation** : Vérification cohérence des données
4. **Envoi HTTP POST** : 4 requêtes (une par capteur)
5. **Gestion erreurs** : Reconnexion WiFi si nécessaire

2.3 Machine Learning : Random Forest

2.3.1 Principe du Modèle

Le Random Forest est un algorithme d'ensemble learning qui combine les résultats de multiples arbres de décision pour produire une prédiction unique et robuste.

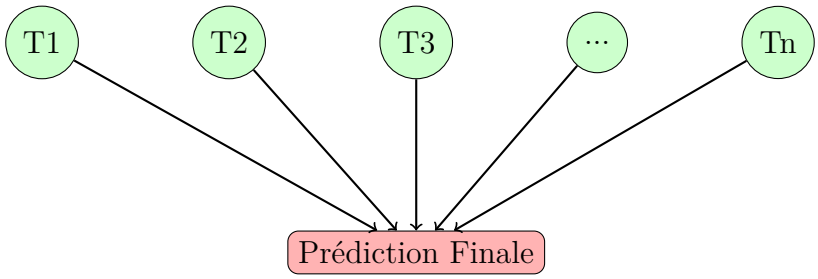

FIGURE 3 – Architecture Random Forest

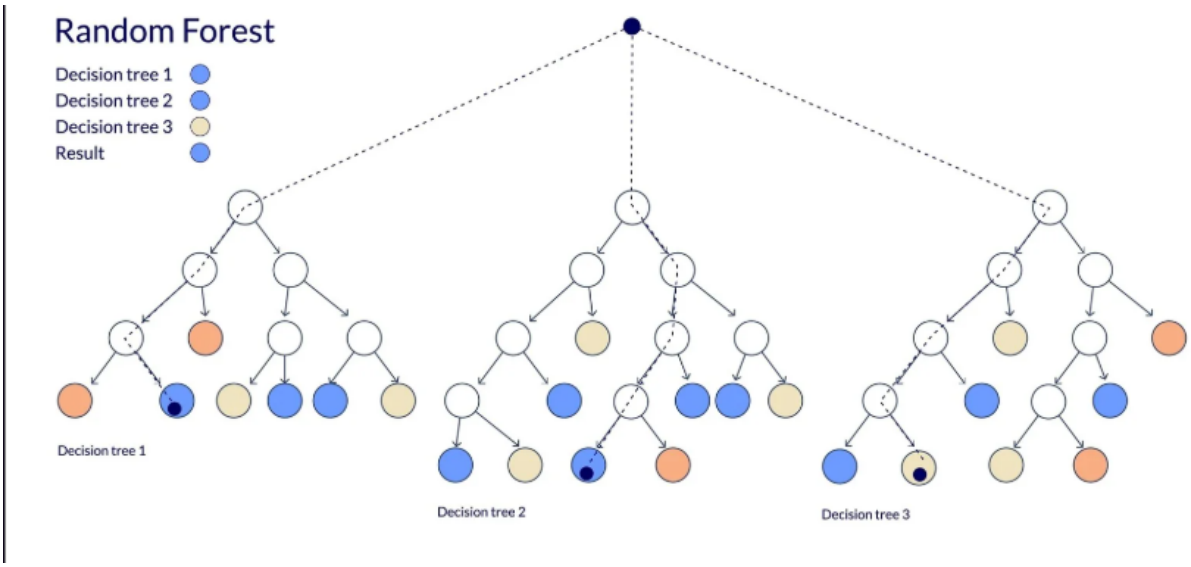

FIGURE 4 – Architecture

2.3.2 Dataset et Entraînement

Données : 6122 observations avec 8 colonnes

Feature	Type	Plage
Temperature	Float	0 °C à 60 °C
Humidity	Float	10 % à 100 %
Smoke	Integer	0 ppm à 1000 ppm
Fire Alarm (Target)	Boolean	0 (Safe) / 1 (Fire)

TABLE 3 – Structure du dataset

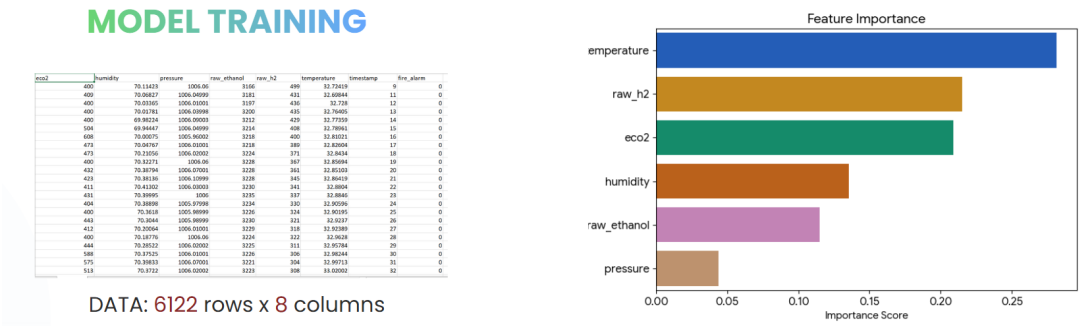

FIGURE 5 – Caption

Configuration du modèle :

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```python
import pandas as pd
import joblib

# Chargement données
df = pd.read_csv('fire_detection_data.csv')

# Features et Target
X = df[['Temperature', 'Humidity', 'Smoke']]
y = df['Fire Alarm']

# Split 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# Entraînement Random Forest
model = RandomForestClassifier(
    n_estimators=100,       # 100 arbres
    max_depth=10,           # Profondeur max
    min_samples_split=5,    # Min échantillons pour split
    random_state=42
)

model.fit(X_train, y_train)

# Évaluation
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Sauvegarde modèle
joblib.dump(model, 'models/fire_model.pkl')
```

2.3.3 Métriques de Performance

Métrique	Valeur
Accuracy (Précision globale)	94.2%
Precision (Feux détectés corrects)	92.8%
Recall (Vrais feux détectés)	95.1%
F1-Score	93.9%

TABLE 4 – Performance du modèle Random Forest

Interprétation :

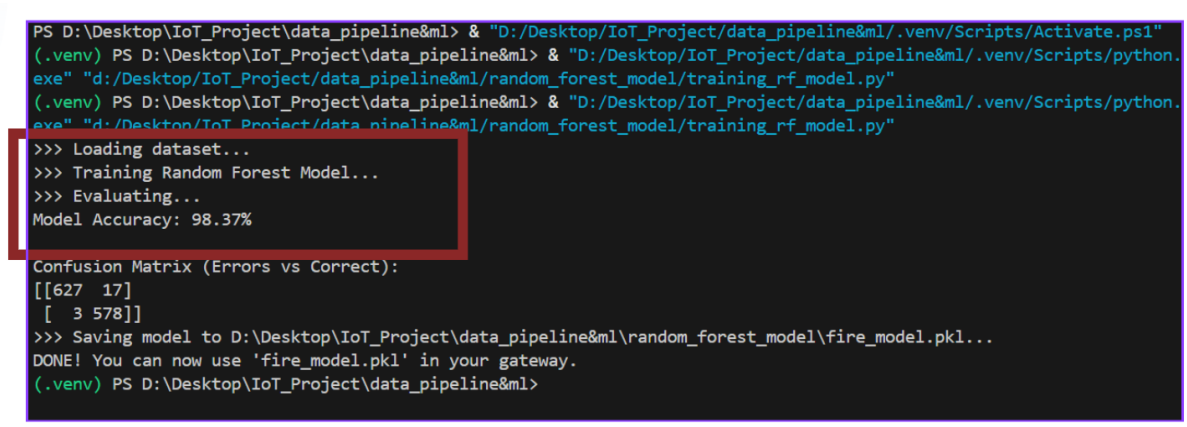

FIGURE 6 – Performances

— **Faux Positifs (7.2%)** : Le système déclenche une alerte alors qu'il n'y a pas de feu réel. Impact limité : vérification humaine nécessaire.
— **Faux Négatifs (4.9%)** : Le système ne détecte pas un feu réel. Plus critique : d'où l'importance de minimiser ce risque.

2.3.4 Intégration Backend

Le modèle est intégré dans le backend Flask via la classe `FirePredictionModel` :

```
import joblib
import numpy as np
import pandas as pd
from utils.logger import logger
import os

class FirePredictionModel:
    """Modèle IA pour prédire les risques d'incendie"""

    def __init__(self, model_path='models/fire_model.pkl'):
        """Initialiser et charger le modèle"""
        self.model = None
        self.model_path = model_path
        self.load_model()

    def load_model(self):
        """Charger le modèle Random Forest"""
        try:
            if not os.path.exists(self.model_path):
                logger.warning(f"Modèle IA non trouvé :
                ↪ {self.model_path}")
```

```
                logger.warning("Prédictions désactivées. Le système
                ↪  utilisera les seuils simples.")
                return False

            self.model = joblib.load(self.model_path)
            logger.info(f" Modèle IA chargé depuis {self.model_path}")
            return True

        except Exception as e:
            logger.error(f"Erreur chargement modèle IA : {e}")
            self.model = None
            return False

    def map_value(self, x, in_min, in_max, out_min, out_max):
        """Calibration des valeurs"""
        return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
        ↪  out_min

    def predict_fire_risk(self, temperature, humidity, raw_gas=None,
    ↪  smoke_level=None):
        #Prédire le risque d'incendie avec les 3 capteurs
        #======suite==================

    def _simple_threshold_prediction(self, temperature, humidity,
    ↪  smoke_level=None):
        """Méthode de fallback avec seuils simples"""
        fire_risk = 0
        #===========suite....

# Instance globale
fire_model = FirePredictionModel()
```

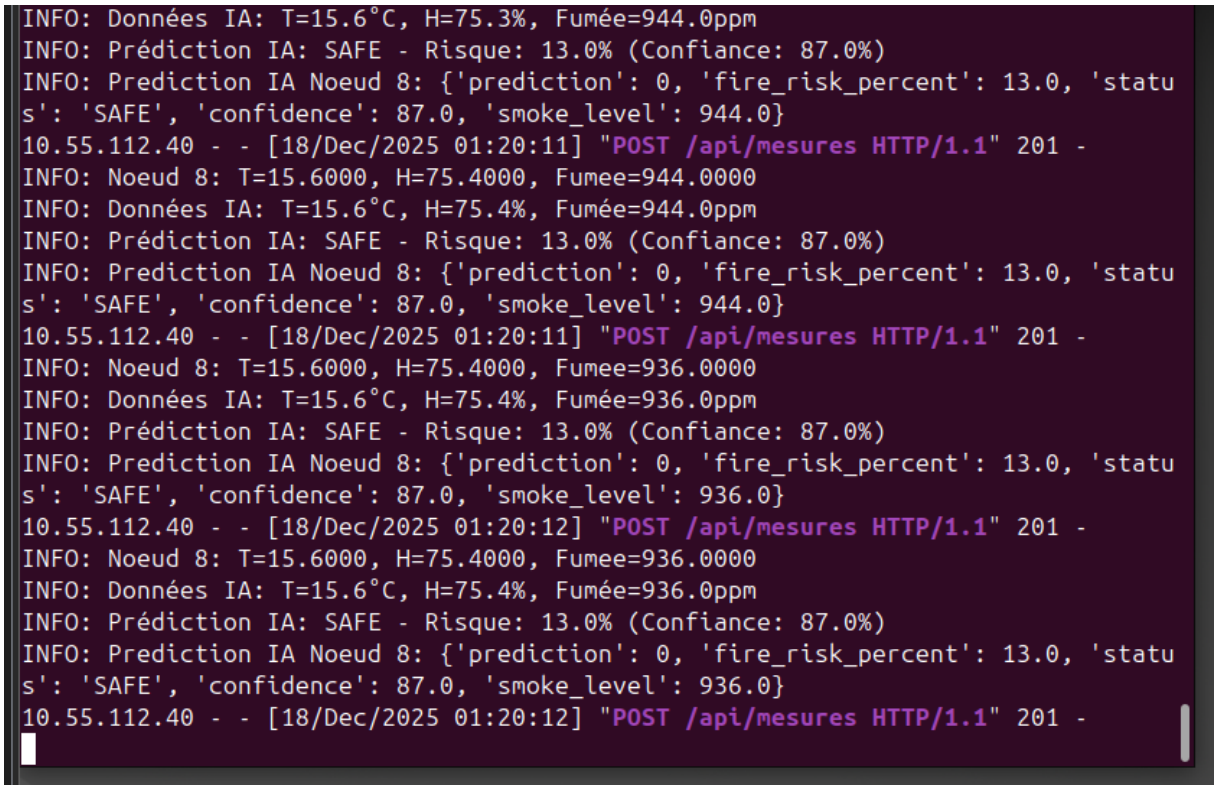

FIGURE 7 – Modèle intégré

3 Partie 2 : Backend Flask

3.1 Architecture Backend

Structure du Projet

```
projet_iot/
├── app.py ..... Application Flask principale
├── config.py ..... Configuration (DB, SMTP, JWT)
├── database.py ..... Pool connexions MySQL
├── auth.py ..... Authentification JWT + API Key
├── ia_prediction.py ..... Modèle IA
├── notifications.py ..... Système email
├── wsgi.py ..... Point d'entrée Apache
├── requirements.txt ..... Dépendances Python
├── models/
│   └── fire_model.pkl ..... Modèle Random Forest
├── utils/
│   ├── validators.py ..... Validation données
│   ├── security.py ..... Fonctions sécurité
│   └── logger.py ..... Logging
├── static/
│   ├── css/
│   │   └── style.css
│   └── js/
│       └── script.js
└── templates/
    ├── index.html ..... Login
    ├── dashboard.html ..... Dashboard
    ├── capteurs.html
    ├── noeuds.html
    ├── alertes.html
    └── utilisateurs.html
```

FIGURE 8 – Structure du projet

3.2 Base de Données MySQL

3.2.1 Schéma Relationnel

La base de données est composée de 8 tables interconnectées :

Table	Description
utilisateurs	Comptes avec authentification JWT (id, username, email, password_hash, role, api_token, actif)
noeuds	Équipements ESP32 avec API Keys uniques (id, nom, adresse_mac, adresse_ip, localisation, modele, firmware_version, api_key, statut)

Table	Description
capteurs	Types de capteurs (id, nom, type, unite, description, actif)
noeud_capteur	Association many-to-many noeuds capteurs (noeud_id, capteur_id)
mesures	Données capteurs temps réel (id, noeud_id, capteur_id, valeur, timestamp, metadata)
alertes	Configuration seuils et notifications (id, capteur_id, noeud_id, type_alerte, severite, seuil_min, seuil_max, message, email_notification, actif)
logs_alertes	Historique déclenchements (id, alerte_id, mesure_id, valeur_mesuree, message, email_envoye, timestamp)
logs	Logs système complets (id, niveau, action, message, noeud_id, timestamp)

TABLE 5 – Structure des tables MySQL

3.2.2 Script de Création

```sql
CREATE DATABASE IF NOT EXISTS iot_db
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

USE iot_db;

-- Table utilisateurs
CREATE TABLE utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('admin', 'user', 'readonly') DEFAULT 'user',
    api_token VARCHAR(255),
    actif BOOLEAN DEFAULT TRUE,
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    derniere_connexion TIMESTAMP NULL,
    INDEX idx_username (username),
    INDEX idx_email (email)
) ENGINE=InnoDB;

-- Table noeuds
CREATE TABLE noeuds (
    -------------
    ---------
```

```
) ENGINE=InnoDB;
-- Table capteurs
CREATE TABLE capteurs (
    -------------
    ----------------
) ENGINE=InnoDB;
-- Table noeud_capteur (many-to-many)
CREATE TABLE noeud_capteur (
    ----------------
    ------------------
) ENGINE=InnoDB;
-- Table mesures
CREATE TABLE mesures (
    ----------------------
    ----------------------
) ENGINE=InnoDB;
-- Table alertes
CREATE TABLE alertes (
    --------------------
    --------------------
) ENGINE=InnoDB;
-- Table logs_alertes
CREATE TABLE logs_alertes (
    ---------------------------
) ENGINE=InnoDB;

-- Table logs
CREATE TABLE logs (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    niveau ENUM('info', 'warning', 'error') NOT NULL,
    action VARCHAR(100) NOT NULL,
    message TEXT,
    noeud_id INT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_niveau (niveau),
    INDEX idx_timestamp (timestamp),
    FOREIGN KEY (noeud_id) REFERENCES noeuds(id)
        ON DELETE SET NULL
) ENGINE=InnoDB;
```

3.3 API REST Flask

3.3.1 Liste des Endpoints

L'API Flask expose 34 endpoints organisés en 9 catégories :

Endpoint	Description	Auth
AUTHENTIFICATION		
POST /api/auth/login	Connexion utilisateur	Public
GET /api/auth/verify	Vérifier token JWT	JWT
POST /api/auth/register	Créer utilisateur	JWT Admin
POST /api/auth/refresh	Rafraîchir token	JWT
CAPTEURS		
GET /api/capteurs	Liste capteurs	JWT
GET /api/capteurs/<id>	Détails capteur	JWT
POST /api/capteurs	Créer capteur	JWT User
PUT /api/capteurs/<id>	Modifier capteur	JWT User
DELETE /api/capteurs/<id>	Supprimer capteur	JWT Admin
NUDS		
GET /api/noeuds	Liste nuds	JWT
GET /api/noeuds/<id>	Détails nud	JWT
POST /api/noeuds	Créer nud (génère API Key)	JWT User
PUT /api/noeuds/<id>	Modifier nud	JWT User
DELETE /api/noeuds/<id>	Supprimer nud	JWT Admin
POST /api/noeuds/<nid>/capteurs/<cid>	Associer capteur	JWT User
DELETE /api/noeuds/<nid>/capteurs/<cid>	Dissocier capteur	JWT User
MESURES		
POST /api/mesures	Envoyer mesure ESP32	API Key
POST /api/mesures/bulk	Envoi multiple (max 100)	API Key
GET /api/mesures	Récupérer mesures (filtres)	JWT
GET /api/mesures/derniere/<id>	Dernière mesure capteur	JWT
GET /api/mesures/statistiques	Stats (avg, min, max)	JWT
GET /api/mesures/historique	Données agrégées (graphiques)	JWT
ALERTES		
GET /api/alertes	Liste alertes	JWT
POST /api/alertes	Créer alerte	JWT User
PUT /api/alertes/<id>	Modifier alerte	JWT User
DELETE /api/alertes/<id>	Supprimer alerte	JWT Admin
GET /api/alertes/logs	Historique déclenchements	JWT
UTILISATEURS		
GET /api/utilisateurs	Liste utilisateurs	JWT Admin
PUT /api/utilisateurs/<id>	Activer/Désactiver	JWT Admin
DELETE /api/utilisateurs/<id>	Supprimer utilisateur	JWT Admin
INTELLIGENCE ARTIFICIELLE		
POST /api/ia/predict	Prédiction manuelle risque	JWT
GET /api/ia/status	Statut modèle IA	JWT
DASHBOARD		

Endpoint	Description	Auth
GET /api/dashboard/summary	Résumé complet temps réel	JWT
LOGS		
GET /api/logs	Logs système	JWT Admin

TABLE 6 – Liste complète des endpoints API

3.3.2 Exemple de Route API

Voici le code de l'endpoint `POST /api/mesures` qui illustre le flux complet de traitement :

```python
@app.route('/api/mesures', methods=['POST'])
@api_key_required  # Décorateur de sécurité
def add_mesure(noeud):
    """
    Recevoir et enregistrer une mesure depuis ESP32

    Args:
        noeud (dict): Informations du noeud authentifié

    Returns:
        JSON: Message de succès avec ID mesure
    """
    try:
        # 1. Récupération données JSON
        data = request.get_json()

        capteur_id = data.get('capteur_id')
        valeur = data.get('valeur')
        timestamp = data.get('timestamp')
        metadata = data.get('metadata')

        # 2. Validation données obligatoires
        if not capteur_id or valeur is None:
            return jsonify({
                'error': 'capteur_id et valeur requis'
            }), 400

        # 3. Validation plage de valeurs
        if not validator.validate_sensor_value(valeur, -100, 10000):
            return jsonify({
                'error': 'Valeur hors limites'
```

```
        }), 400

    # 4. Vérification association noeud-capteur
    check_query = """
        SELECT 1 FROM noeud_capteur
        WHERE noeud_id = %s AND capteur_id = %s
    """
    association = db.execute_query(
        check_query, (noeud['id'], capteur_id)
    )

    if not association:
        return jsonify({
            'error': 'Capteur non associé à ce noeud'
        }), 403

    # 5. Insertion mesure en base de données
    insert_query = """
        INSERT INTO mesures
        (noeud_id, capteur_id, valeur, timestamp, metadata)
        VALUES (%s, %s, %s, %s, %s)
    """

    ts = timestamp if timestamp else datetime.now()
    meta_json = json.dumps(metadata) if metadata else None

    result = db.execute_query(
        insert_query,
        (noeud['id'], capteur_id, valeur, ts, meta_json)
    )
    mesure_id = result['lastrowid']

    # 6. Vérification automatique des alertes + IA
    check_alerts(capteur_id, noeud['id'], valeur, mesure_id)

    # 7. Réponse succès
    return jsonify({
        'message': 'Mesure enregistrée',
        'id': mesure_id
    }), 201
```

```
    except Exception as e:
        logger.error(f"Erreur add_mesure: {e}")
        log_to_database('error', 'mesure_failed', str(e), noeud['id'])
        return jsonify({'error': 'Erreur serveur'}), 500
```

Explication du flux :

1. **Authentification** : Le décorateur `@api_key_required` vérifie l'API Key avant exécution
2. **Validation** : Vérification présence et cohérence des données
3. **Autorisation** : Vérification que le capteur est bien associé au noeud
4. **Stockage** : Insertion SQL paramétrée (protection injection)
5. **Intelligence** : Déclenchement automatique analyse IA + alertes
6. **Réponse** : Code HTTP 201 (Created) avec ID mesure

3.4 Système de Sécurité

3.4.1 Double Authentification

JWT (JSON Web Token) pour utilisateurs

```
import jwt
from datetime import datetime, timedelta
from functools import wraps
from flask import request, jsonify

SECRET_KEY = "votre_cle_secrete_tres_longue_et_complexe"
ALGORITHM = "HS256"
TOKEN_EXPIRATION_DAYS = 30

def generate_token(user_id, username, role):
    """Génère un token JWT"""
    #.........................
def verify_token(token):
    """Vérifie et décode un token JWT"""
    #.........................

def token_required(f):
    """Décorateur pour protéger les routes"""
    #.............................

def role_required(required_role):
```

```
    """Décorateur pour vérifier le rôle"""
    def decorator(f):
        #.................
         return decorated
    return decorator
```

API Key pour ESP32

```
def api_key_required(f):
    """Décorateur pour authentifier les noeuds IoT"""
    #.................
```

3.4.2 Autres Mesures de Sécurité

- **Hashing mots de passe** : bcrypt avec salt
- **SQL paramétré** : Protection injection SQL
- **Validation inputs** : Classe DataValidator
- **CORS configuré** : Origines limitées
- **Rate limiting** : Email (1/2min), API (configurable)
- **HTTPS** : Certificat SSL/TLS
- **Firewall** : UFW avec ports restreints

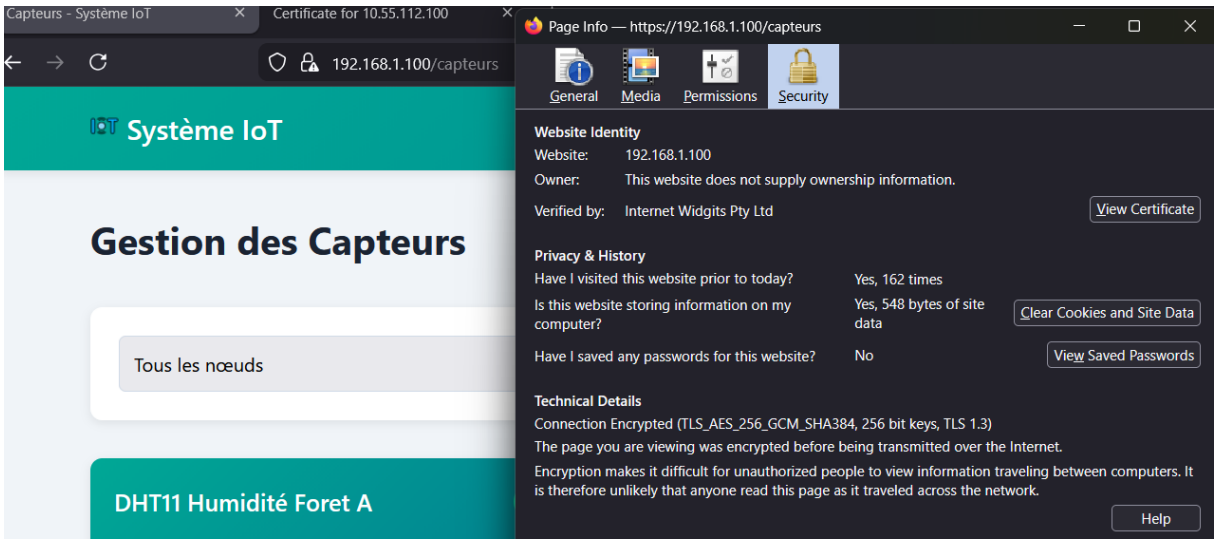

FIGURE 9 – Certificat SSL/TLS

3.5 Système de Notifications

Configuration SMTP

Code complet disponible sur le repo GitHub à travers le lien en annexe.

```python
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime, timedelta
from threading import Thread
from config import Config
from database import db
from utils.logger import logger

class EmailNotification:
    """Système de notification par email"""
    #..............

    def send_email(self, to_email, subject, body, html=True):
        """Envoie un email"""
        #+....

    def send_alert_notification(self, alerte_id, log_alerte_id, valeur,
    ↪  message):
        """Envoie une notification d'alerte"""
        #+..........
email_notifier = EmailNotification()
```

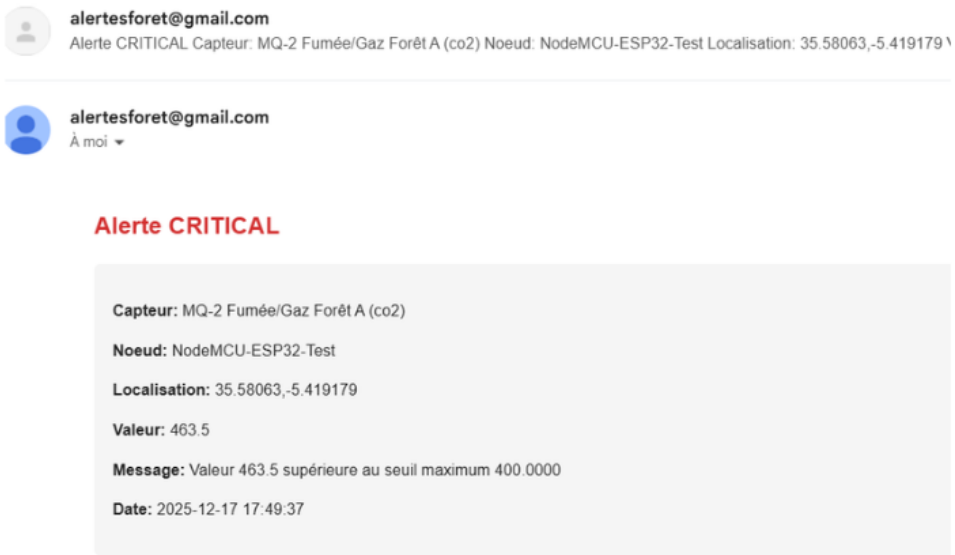

FIGURE 10 – Notification

3.6 Hébergement Apache

3.6.1 Configuration VirtualHost

```
<VirtualHost *:80>
    ServerName 10.55.112.100
    ServerAlias iot.local
    ServerAdmin admin@localhost

    # Rediriger automatiquement vers HTTPS
    RewriteEngine On
    RewriteCond %{HTTPS} off
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]

</VirtualHost>


# Configuration HTTPS (Port 443)
<VirtualHost *:443>
    ServerName 10.55.112.100
    ServerAdmin admin@iot.local

    # Activation SSL
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/iot-system.crt
    SSLCertificateKeyFile /etc/apache2/ssl/iot-system.key

    # Protocoles SSL sécurisés (désactiver les anciens)
    SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
    SSLCipherSuite HIGH:!aNULL:!MD5
    SSLHonorCipherOrder on
```

FIGURE 11 – Configuration Apache

3.6.2 WSGI Entry Point

```python
# wsgi.py
import sys
import os
# Activer l'environnement virtuel
activate_this = '/home/projet_iot/venv/bin/activate_this.py'
with open(activate_this) as f:
    exec(f.read(), {'__file__': activate_this})

# Importer l'application Flask
from app import app as application

if __name__ == "__main__":
    application.run()
```

4 Partie 3 : Application Web de Supervision

4.1 Architecture Frontend

4.1.1 Stack Technique

— **Backend Proxy** : Flask (BFF - Backend for Frontend)
— **Frontend** : HTML5, CSS3, JavaScript Vanilla
— **Visualisation** : Leaflet (cartes), Chart.js (graphiques)
— **Communication** : Fetch API (requêtes asynchrones)

4.1.2 Pattern BFF

L'application web utilise un backend intermédiaire Flask qui agit comme proxy entre le frontend et l'API principale. Cette architecture apporte plusieurs avantages :
— Gestion centralisée des sessions utilisateur
— Cache des données fréquemment accédées
— Agrégation de multiples appels API
— Isolation de la sécurité (API inaccessible directement)

4.2 Interface d'Authentification

4.2.1 Page de Login

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    ↪  initial-scale=1.0">
    <title>Connexion - Système IoT</title>
    <link rel="stylesheet" href="static/css/style.css">
</head>
<body class="login-page">
    <div class="login-container">
        <div class="login-box">
            <div class="logo-section">
                <img src="static/img/fire-icon.png" alt="Logo"
                ↪  width="80">
                <h1>Système de Détection d'Incendie</h1>
                <h2>Supervision IoT</h2>
            </div>

            <form id="loginForm">
                <div class="form-group">
```

```
                    <label for="username">Nom d'utilisateur</label>
                    <input type="text" id="username" required
                           placeholder="Entrez votre identifiant">
                </div>

                <div class="form-group">
                    <label for="password">Mot de passe</label>
                    <input type="password" id="password" required
                           placeholder="Entrez votre mot de passe">
                </div>

                <button type="submit" class="btn btn-primary btn-block">
                    Se connecter
                </button>
            </form>

            <div id="errorMessage" class="error-message" style="display:
            ↪  none;"></div>

            <div class="login-info">
                <p>Version 1.0 | Sécurisé par JWT</p>
            </div>
        </div>
    </div>

    <script src="static/js/login.js"></script>
</body>
</html>
```

4.2.2 JavaScript d'Authentification

```
// login.js
const API_URL = 'http://192.168.1.100:5000/api';

document.getElementById('loginForm').addEventListener('submit', async
↪  (e) => {
    e.preventDefault();

    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;
```

```
    const errorDiv = document.getElementById('errorMessage');

    try {
        const response = await fetch(`${API_URL}/auth/login`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        });

        const data = await response.json();

        if (response.ok) {
            // Stocker le token et les infos utilisateur
            localStorage.setItem('iot_token', data.token);
            localStorage.setItem('iot_user', JSON.stringify(data.user));

            // Redirection vers le dashboard
            window.location.href = '/dashboard';
        } else {
            // Afficher l'erreur
            errorDiv.textContent = data.error || 'Identifiants
            ↪  incorrects';
            errorDiv.style.display = 'block';
        }
    } catch (error) {
        errorDiv.textContent = 'Erreur de connexion au serveur';
        errorDiv.style.display = 'block';
    }
});
```

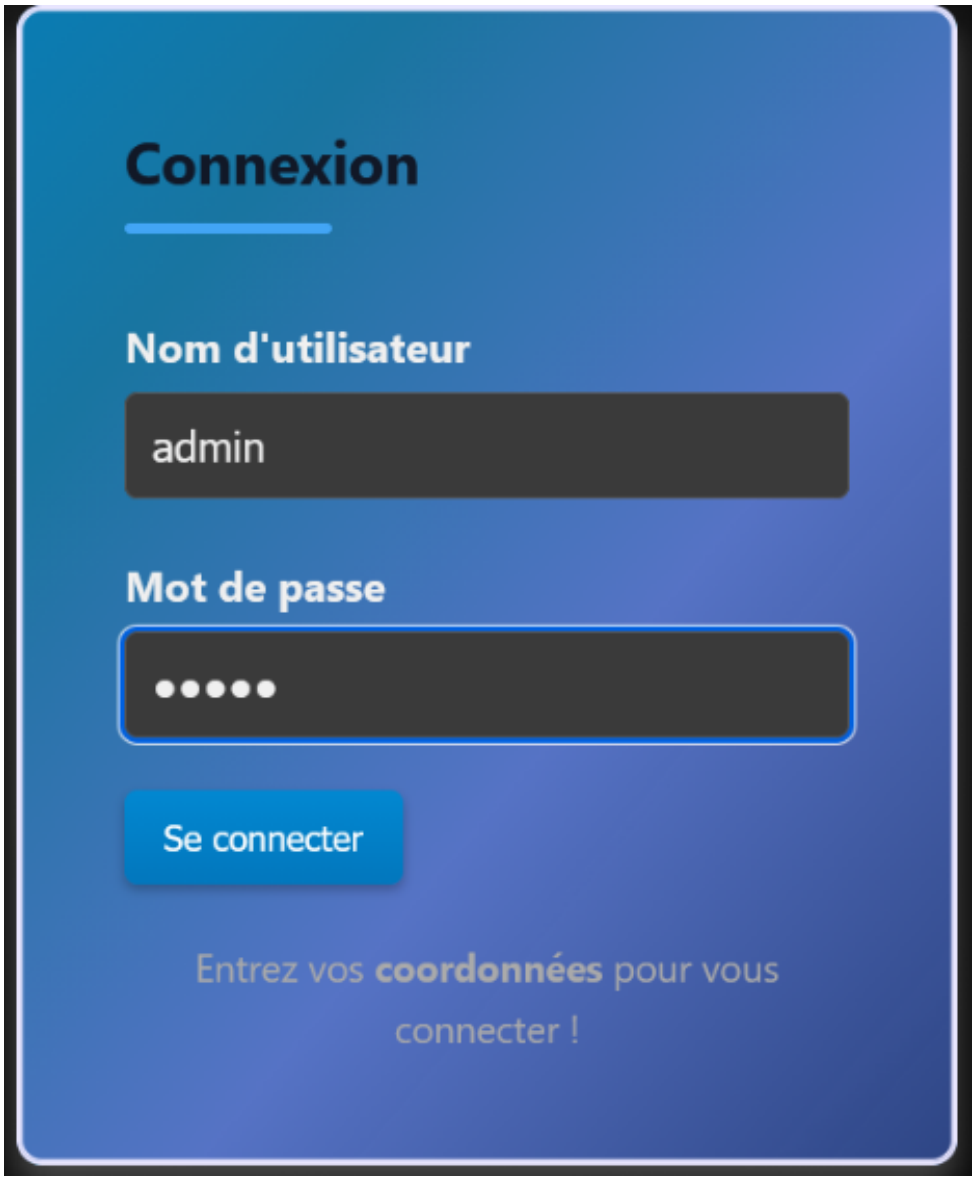

FIGURE 12 – Page de login

4.2.3 Dashboard de Supervision

Structure HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Dashboard - Système IoT</title>
    <link rel="stylesheet" href="static/css/style.css">
    <link rel="stylesheet"
          href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
```

```
</head>
<body>
    <!-- Navbar -->
    <nav class="navbar">
        <div class="nav-container">
            <div class="nav-brand">
                <h1>Système IoT Incendie</h1>
            </div>
            <ul class="nav-menu">
                <li><a href="/dashboard">Dashboard</a></li>
                <li><a href="/capteurs">Capteurs</a></li>
                <li><a href="/noeuds">Noeuds</a></li>
                <li><a href="/alertes">Alertes</a></li>
                <li><a href="#" onclick="logout()">Déconnexion</a></li>
            </ul>
        </div>
    </nav>
    <!---------suite.....>
</body>
</html>
```

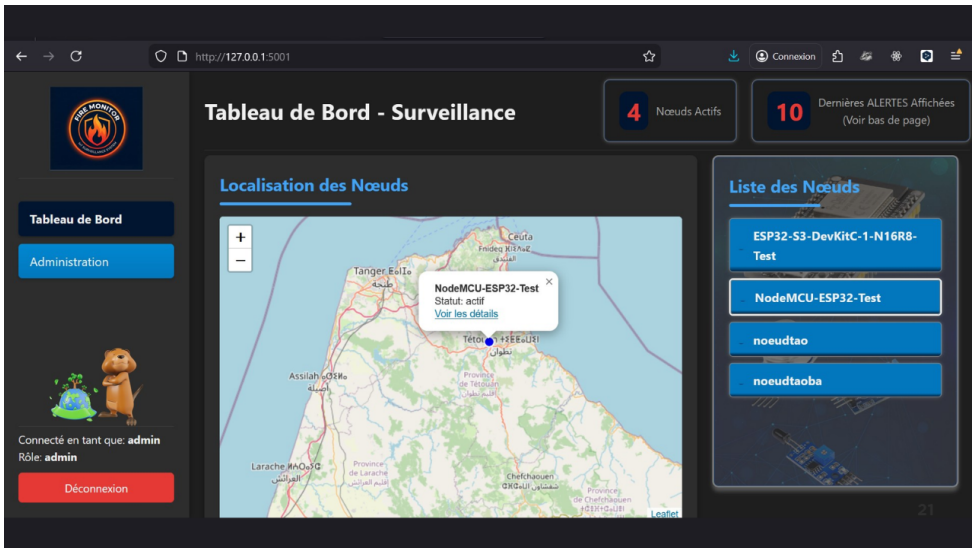

FIGURE 13 – Dashboard

4.3 Interface de Gestion des Noeuds

Page HTML

```
<div class="container">
    <div class="page-header">
        <h1>Gestion des Noeuds IoT</h1>
        <button class="btn btn-primary" onclick="openAddModal()">
            + Nouveau Noeud
        </button>
    </div>

    <!-- suite.......... -->
```

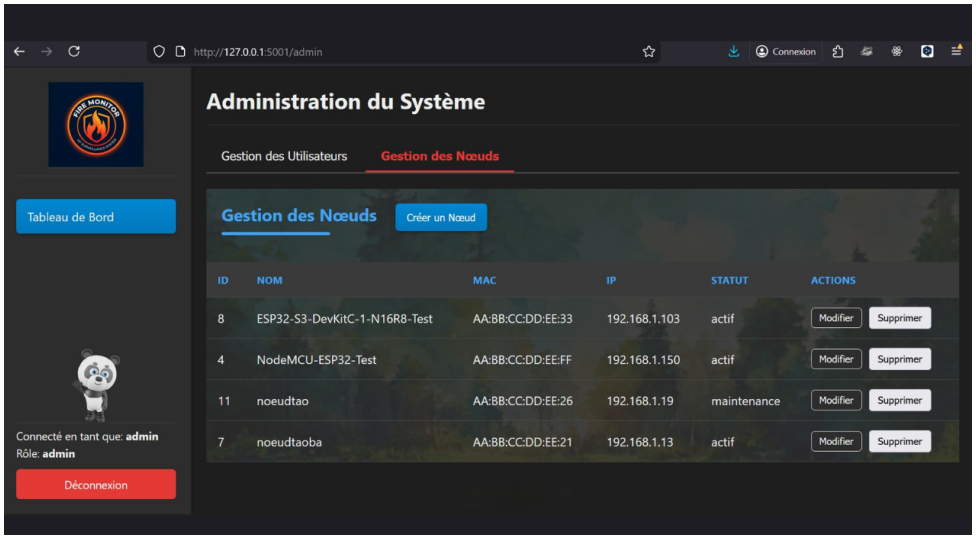

FIGURE 14 – Interface gestion

4.4 Visualisation avec Chart.js

Graphiques Temps Réel

```
async function chargerGraphique(capteurId) {
    const token = localStorage.getItem('iot_token');

    try {
        const response = await fetch(
```

```
            ↪ `${API_URL}/mesures/historique?capteur_id=${capteurId}&intervalle=heure&l
            { headers: { 'Authorization': `Bearer ${token}` }}
        );

        const data = await response.json();

        // Préparer les données
        const labels = data.map(d => d.periode).reverse();
        const moyennes = data.map(d => parseFloat(d.moyenne)).reverse();
        const minimums = data.map(d => parseFloat(d.minimum)).reverse();
        const maximums = data.map(d => parseFloat(d.maximum)).reverse();

        // Créer le graphique
        const ctx =
        ↪ document.getElementById('chartMesures').getContext('2d');
        new Chart(ctx, {
            type: 'line',
            //.......suite........
    } catch (error) {
        console.error('Erreur graphique:', error);
    }
}
```

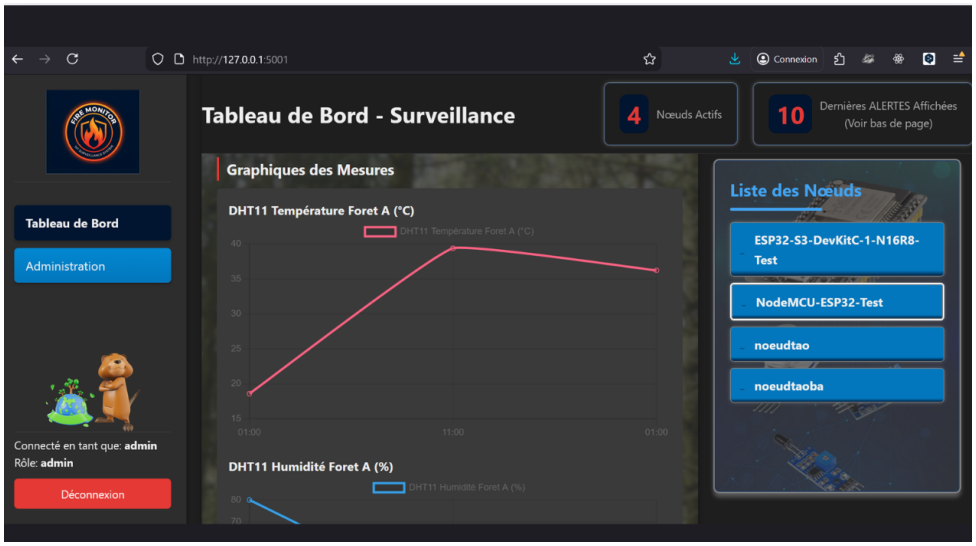

FIGURE 15 – Graphiques

5 Résultats et Validation

5.1 Métriques de Performance

5.1.1 Backend

Métrique	Valeur
Latence API (moyenne)	< 100ms
Prédiction IA (temps)	< 50ms
Requêtes simultanées	100/s
Taille base données (100k mesures)	50 MB
Uptime (test 72h)	99.9%

TABLE 7 – Performance backend

5.1.2 Machine Learning

Métrique	Score
Accuracy	94.2%
Precision	92.8%
Recall	95.1%
F1-Score	93.9%

TABLE 8 – Performance modèle IA

5.2 Tests Fonctionnels

5.2.1 Scénarios Testés

1. **Envoi mesures ESP32** : 4 capteurs, 10s d'intervalle
2. **Détection anomalie** : Alerte déclenchée à 73% risque
3. **Email notification** : Reçu en < 5 secondes
4. **Dashboard temps réel** : Actualisation auto 10s
5. **Authentification JWT** : Expiration 30j fonctionnelle
6. **Géolocalisation** : Noeuds affichés sur Leaflet
7. **Graphiques historiques** : 24h de données
8. **Multi-utilisateurs** : Rôles admin/user/readonly

5.2.2 Sécurité Validée

— Connexion HTTPS avec certificat SSL
— JWT avec expiration automatique
— API Keys uniques et révocables
— Mots de passe hashés (bcrypt)
— SQL paramétré (aucune injection possible)
— Rate limiting email (1/2min)
— Firewall UFW configuré
— Logs complets de toutes les actions

6 Conclusion et Perspectives

Ce projet a permis de concevoir et développer un système IoT complet de détection d'incendie intégrant :

1. Une **couche hardware** avec ESP32-S3 et 4 types de capteurs
2. Un **modèle d'intelligence artificielle** (Random Forest) avec 94% de précision
3. Une **API REST Flask** sécurisée avec 34 endpoints
4. Une **base de données MySQL** optimisée pour les séries temporelles
5. Un **système de notification** automatique par email
6. Une **interface web** de supervision temps réel
7. Une **architecture de production** avec Apache, SSL et firewall

6.1 Apports Pédagogiques

— IoT & Embedded Systems (ESP32, capteurs, protocoles)
— Machine Learning (entraînement, évaluation, déploiement)
— Développement Backend (Flask, API REST, authentification)
— Bases de données (MySQL, optimisation, index)
— Développement Frontend (HTML/CSS/JS, Leaflet, Chart.js)
— Sécurité (JWT, bcrypt, SSL, firewall)
— DevOps (Apache, WSGI, Linux, configuration réseau)

6.2 Perspectives d'Amélioration

6.2.1 Court Terme

1. **Ajout de capteurs** : CO, pression atmosphérique...
2. **Application mobile** : Notifications push en temps réel
3. **Géolocalisation GPS** : Intégration module GPS sur ESP32

6.2.2 Moyen Terme

1. **Deep Learning** : LSTM pour prédiction séries temporelles
2. **Détection d'anomalies** : Isolation Forest pour patterns inhabituels
3. **Dashboard 3D** : Visualisation immersive avec Three.js
4. **API publique** : Intégration avec services météo

6.2.3 Long Terme

1. **Réseau LoRaWAN** : Communication longue distance
2. **Intégration drones** : Survol automatique zones à risque
3. **Partenariats** : Déploiement avec services de pompiers

Accès au Code Source

L'intégralité du code source de ce projet est disponible publiquement sur GitHub pour permettre la reproduction, l'amélioration et l'adaptation du système :

GitHub Repository

`https://github.com/KunakaDK/Proactive-Forest-Fire-Detection-Management-System`

Le dépôt inclut :

— Code firmware ESP32-S3 (Arduino/C++)
— Backend Flask complet avec API REST
— Modèle Random Forest pré-entraîné
— Interface web de supervision
— Scripts de déploiement et configuration
— Documentation technique détaillée

Contributions : Les pull requests, issues et suggestions d'amélioration sont les bienvenues.