HEH.be
Sciences
et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Rapport Technique

- Ce document est le rapport technique de SmartCity – Energie.

Bachelier en Informatique – ITR2 – GR10

Projet Interdisciplinaire
SMARTCITY (Énergie) – Décembre 2024

Tom Deneyer
Anthony Vergeylen
Quentin Hemeryck
Kelian Noullet

Rapport Technique SmartCity - Energie

Table des matières

Introduction

Ce rapport technique présente une vue d'ensemble complète des algorithmes, du code, et des configurations développées dans le cadre du projet interdisciplinaire.

Ce document a pour objectif de fournir une documentation claire et structurée, permettant de comprendre le fonctionnement des systèmes développés, les approches adoptées tout au long du projet.

Codes Python

Générer des données de capteurs aléatoires

Code disponible :
https://github.com/AnthonyCodeDev/SmartCity_CDCGR10/blob/GestionAutomatiqueDonnee/donneeCapteur.py

1. Connexion à la Base de Données

La première étape du processus consiste à établir une connexion à la base de données **MySQL**.
Nous utilisons **« mysql.connector »** pour gérer le connexion avec **MySQL**.

```python
def connect_to_db():
    """Connecte à la base de données et retourne la connexion."""
    try:
        conn = mysql.connector.connect(**db_config)
        return conn
    except mysql.connector.Error as err:
        print(f"Erreur de connexion: {err}")
        exit(1)
```

- **db_config** contient les informations de connexion nécessaires à l'accès à la base de données. Ce dictionnaire comprend l'hôte, l'utilisateur, le mot de passe et le nom de la base de données.
- La fonction **« connect_to_db »** tente de se connecter en utilisant ces informations. En cas d'échec (par exemple, si la base de données est inaccessible), une erreur est affichée et le programme se termine (exit(1)).

2. Récupération des Capteurs Actifs

Le code récupère les capteurs actifs dans la base de données. Les capteurs sont identifiés par leur état StateUp = 1, ce qui signifie qu'ils sont en état de fonctionnement.

```python
def fetch_sensors(conn):
    """Récupère les capteurs actifs (StateUp = 1) et leurs informations depuis la table sensor."""
    cursor = conn.cursor(dictionary=True)
    query = "SELECT IPv4, ID_SensorType FROM sensor WHERE StateUp = 1"
    try:
        cursor.execute(query)
        sensors = cursor.fetchall()
        return sensors
    except mysql.connector.Error as err:
        print(f"Erreur lors de la récupération des capteurs: {err}")
        return []
    finally:
        cursor.close()
```

- **« fetch_sensors »** exécute une requête SQL pour obtenir les capteurs actifs, c'est-à-dire ceux dont StateUp = 1.

- La méthode **« cursor.fetchall() »** permet de récupérer toutes les lignes renvoyées par la requête sous forme de liste de dictionnaires, ce qui facilite l'accès aux données (par exemple, l'adresse IP et le type de capteur).

3. Insertion des Données dans la Table capteurs_energie

Une fois les capteurs récupérés, des données aléatoires sont insérées dans la table **« capteurs_energie »**. Cela simule la collecte d'informations sur la production d'énergie des capteurs.

```python
def insert_into_capteurs_energie(conn, sensors):
    """Insère une entrée aléatoire dans la table capteurs_energie."""
    if not sensors:
        print("Aucun capteur disponible pour l'insertion.")
        return

    cursor = conn.cursor()
    sensor = random.choice(sensors)
    ip = sensor['IPv4']
    type_energie = 'solaire' if sensor['ID_SensorType'] == 1 else 'éolienne'
    valeur = round(random.uniform(1, 50), 2)
    date_mesure = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    query = (
        "INSERT INTO capteurs_energie (id_capteur, type_energie, valeur, date_mesure) "
        "VALUES (%s, %s, %s, %s)"
    )
    data = (ip, type_energie, valeur, date_mesure)

    try:
        cursor.execute(query, data)
        conn.commit()
        print(f"Donnée insérée dans capteurs_energie: {data}")
    except mysql.connector.Error as err:
        print(f"Erreur lors de l'insertion dans capteurs_energie: {err}")
    finally:
        cursor.close()
```

- **« insert_into_capteurs_energie »** choisit un capteur actif aléatoire à partir de la liste des capteurs.
- En fonction du type de capteur, il génère une valeur d'énergie (aléatoire entre 1 et 50) et l'insère dans la table **« capteurs_energie »** avec la date actuelle.
- La requête SQL utilise des paramètres (%s) pour éviter les injections SQL.

4. Insertion des Données dans la Table consommation_energie

Similaire à l'insertion dans **« capteurs_energie »**, une entrée aléatoire est insérée dans la table **« consommation_energie »**, simulant une consommation d'énergie dans des adresses spécifiques.

```python
def insert_into_consommation_energie(conn):
    """Insère une entrée aléatoire dans la table consommation_energie."""
    cursor = conn.cursor()
    id_adresse = random.randint(1, 100)  # Générer un id_adresse aléatoire
    consommation = round(random.uniform(1, 5), 2)
    date = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    query = (
        "INSERT INTO consommation_energie (id_adresse, consommation, date) "
        "VALUES (%s, %s, %s)"
    )
    data = (id_adresse, consommation, date)

    try:
        cursor.execute(query, data)
        conn.commit()
        print(f"Donnée insérée dans consommation_energie: {data}")
    except mysql.connector.Error as err:
        print(f"Erreur lors de l'insertion dans consommation_energie: {err}")
    finally:
        cursor.close()
```

- **« insert_into_consommation_energie »** génère une entrée de la db aléatoire pour l'adresse et la consommation énergétique. Les valeurs sont insérées dans la table correspondante avec la date du moment.

5. Tests Unitaires

Des tests unitaires ont été réalisés pour vérifier le bon fonctionnement des fonctions. Pour simuler la connexion à la base de données, l'exécution des requêtes SQL et les insertions de données.

Les tests couvrent les scénarios suivants :

- **Connexion réussie à la base de données.**
- **Échec de la connexion à la base de données.**
- **Récupération correcte des capteurs.**
- **Insertion correcte des données dans les tables « capteurs_energie » et « consommation_energie ».**

Les tests permettent de s'assurer que chaque partie du code fonctionne comme prévu (sans nécessiter de connexion réelle à la base de données).

Détecter des surcharges automatiquement et envoyer une alerte

Code disponible :
https://github.com/AnthonyCodeDev/SmartCity_CDCGR10/blob/GestionAutomatiqueDonnee/detect_sensor_overload.py

1. Récupération de la Production des 7 Derniers Jours

Le code récupère les données de production pour chaque capteur actif sur les 7 derniers jours.
Ces données sont regroupées par capteur et totalisées.

```python
def fetch_last_week_production(conn):
    """Récupère la production des capteurs pour les 7 derniers jours."""
    cursor = conn.cursor(dictionary=True)
    end_date = datetime.now()
    start_date = end_date - timedelta(days=7)

    query = (
        "SELECT id_capteur, SUM(valeur) AS total_production "
        "FROM capteurs_energie "
        "WHERE date_mesure BETWEEN %s AND %s "
        "GROUP BY id_capteur"
    )
    cursor.execute(query, (start_date, end_date))
    results = cursor.fetchall()
    cursor.close()
    return results
```

- La méthode **« fetch_last_week_production »** exécute une requête SQL pour calculer la somme de la production **(SUM(valeur))** de chaque capteur (id_capteur) sur une période de 7 jours.
  Les résultats sont retournés sous forme de liste de dictionnaires.

3. Calcul de la Moyenne de Production Anterieure

Cette étape calcule la moyenne de production pour un capteur spécifique sur les périodes précédant les 7 derniers jours.

```python
def fetch_previous_average_production(conn, id_capteur, start_date):
    """Récupère la production moyenne avant une période donnée."""
    cursor = conn.cursor(dictionary=True)
    query = (
        "SELECT AVG(valeur) AS avg_production "
        "FROM capteurs_energie "
        "WHERE id_capteur = %s AND date_mesure < %s"
    )
    cursor.execute(query, (id_capteur, start_date))
    result = cursor.fetchone()
    cursor.close()
    return result['avg_production'] if result['avg_production'] else 0
```

- La fonction **« fetch_previous_average_production »** exécute une requête SQL pour calculer la moyenne **(AVG(valeur))** des valeurs de production pour un capteur donné (id_capteur) **avant une date spécifique**.

4. Vérification des Alertes Récentes

Ce bout de code vérifie si une alerte a été récemment enregistré pour éviter les doublons dans la détection d'anomalies.

```
def has_recent_alert(conn, id_capteur):
    """Vérifie si une alerte a été envoyée pour un capteur dans les dernières 24 heures."""
    cursor = conn.cursor(dictionary=True)
    check_date = datetime.now() - timedelta(hours=24)

    query = (
        "SELECT COUNT(*) AS recent_alerts "
        "FROM alertes_surcharge "
        "WHERE id_capteur = %s AND date_signalement > %s"
    )
    cursor.execute(query, (id_capteur, check_date))
    result = cursor.fetchone()
    cursor.close()
    return result['recent_alerts'] > 0
```

- La fonction **« has_recent_alert »** vérifie dans la table **« alertes_surcharge »** si une alerte a été enregistrée pour le capteur dans les dernières 24 heures.
  Cela évite de générer plusieurs alertes pour la même anomalie.

5. Insertion d'une Nouvelle Alerte

Lorsqu'une anomalie est détectée et qu'il n'existe pas d'alerte récente, une nouvelle alerte est ajoutée à la table **« alertes_surcharge »**.

```
def insert_alert(conn, id_capteur, description, niveau):
    """Insère une alerte dans la table alertes_surcharge."""
    cursor = conn.cursor()
    query = (
        "INSERT INTO alertes_surcharge (id_capteur, description, niveau, date_signalement) "
        "VALUES (%s, %s, %s, %s)"
    )
    data = (id_capteur, description, niveau, datetime.now())
    try:
        cursor.execute(query, data)
        conn.commit()
        print(f"Alerte insérée: {data}")
    except mysql.connector.Error as err:
        print(f"Erreur lors de l'insertion de l'alerte: {err}")
    finally:
        cursor.close()
```

La fonction **« insert_alert »** insère une alerte avec un niveau (critique/moyen/…), une description et la date actuelle dans la table **« alertes_surcharge »**.

6. Vérification des Anomalies de Production

La fonction principale coordonne **toutes les étapes pour détecter les anomalies** et **insérer des alertes si nécessaire**.

```
def check_production_anomaly():
    """Vérifie les anomalies de production et insère des alertes si nécessaire."""
    conn = connect_to_db()
    try:
        last_week_data = fetch_last_week_production(conn)

        for entry in last_week_data:
            id_capteur = entry['id_capteur']
            total_production = entry['total_production']
            start_date = datetime.now() - timedelta(days=7)

            previous_avg = fetch_previous_average_production(conn, id_capteur, start_date)

            if previous_avg > 0 and total_production > 2 * previous_avg:
                if not has_recent_alert(conn, id_capteur):
                    niveau = 'critique' if total_production > 3 * previous_avg else 'moyen'
                    description = (
                        f"Surcharge détectée: production {total_production} kWh, "
                        f"soit +{round((total_production - previous_avg) / previous_avg * 100)}% par rapport à la
moyenne."
                    )
                    insert_alert(conn, id_capteur, description, niveau)
                else:
                    print(f"Alerte récente déjà présente pour le capteur {id_capteur}, aucune nouvelle alerte
envoyée.")
    finally:
        conn.close()
```

Cette fonction combine toutes les étapes précédentes :

- ❖ **Récupération des données de production des 7 derniers jours.**
- ❖ **Comparaison avec les moyennes passées.**
- ❖ **Vérification des alertes récentes.**
- ❖ **Envoi d'une nouvelle alerte en cas d'anomalie détectée.**

Tests Unitaires

Des tests unitaires ont été réalisés pour valider les fonctions principales, incluant :

- **La connexion à la base de données.**
- **La récupération des données de production.**
- **Le calcul des moyennes passées.**
- **La vérification des alertes récentes.**
- **L'insertion d'alertes dans la base.**

Ces tests simulent les interactions avec la base de données.

Codes PHP

Système d'Authentification via LDAP

Code disponible :
https://github.com/AnthonyCodeDev/SmartCity_CDCGR10/blob/SiteWeb/modele/AuthModele.php

1. Connexion au Serveur LDAP

Le code fait une connexion sécurisée **au serveur LDAP** spécifié via son URL et son port.

```php
$ldapConnection = ldap_connect($this->ldapHost, $this->ldapPort);

if (!$ldapConnection) {
    return false; // Échec de connexion au serveur LDAP
}

ldap_set_option($ldapConnection, LDAP_OPT_PROTOCOL_VERSION, 3);
ldap_set_option($ldapConnection, LDAP_OPT_REFERRALS, 0);
```

- La fonction **ldap_connect** établit une connexion avec l'hôte et le port LDAP définis.
- Les options **LDAP_OPT_PROTOCOL_VERSION** et **LDAP_OPT_REFERRALS** sont configurées pour s'assurer que ***la communication respecte le protocole LDAP version 3***.

2. Authentification Initiale

Une première authentification est effectuée avec le format **username@domain** pour vérifier les informations d'identification.

```php
$ldapBind = @ldap_bind($ldapConnection, $username . "@smartcity.lan", $motDePasse);

if (!$ldapBind) {
    $_SESSION['error'] = "Échec de l'authentification : mot de passe incorrect.";
    ldap_close($ldapConnection);
    return false;
}
```

- **ldap_bind** tente de lier l'utilisateur avec **son nom d'utilisateur** et **son mot de passe**.
- En cas d'échec, un message d'erreur est enregistré dans la session et **la connexion est fermée**.

3. Recherche de l'Utilisateur dans le Répertoire

Une requête **LDAP** est exécutée pour récupérer les informations de l'utilisateur à l'aide d'un filtre basé sur son **sAMAccountName**.

```php
$searchFilter = "(sAMAccountName=$username)";
$searchResult = @ldap_search($ldapConnection, $this->ldapBaseDn, $searchFilter);

if (!$searchResult) {
    $_SESSION['error'] = "Erreur LDAP lors de la recherche : " . ldap_error($ldapConnection);
    ldap_close($ldapConnection);
    return false;
}
```

- **ldap_search** effectue une recherche dans le répertoire **LDAP** à partir de la base **DN** spécifiée **($this->ldapBaseDn)**.

Si la recherche échoue, **une erreur descriptive est ajoutée à la session**.

4. Validation de l'Utilisateur

Les résultats de la recherche sont analysés pour vérifier si l'utilisateur existe. Si c'est le cas, une seconde liaison est effectuée avec le **DN (Distinguished Name)** complet de l'utilisateur pour **valider son mot de passe**.

```php
$entries = ldap_get_entries($ldapConnection, $searchResult);

if ($entries['count'] === 0) {
    $_SESSION['error'] = "Utilisateur non trouvé dans le répertoire LDAP.";
    ldap_close($ldapConnection);
    return false;
}

$userDn = $entries[0]['dn'];

if (@ldap_bind($ldapConnection, $userDn, $motDePasse)) {
    $memberOf = $entries[0]['memberof'] ?? [];
    $role = 'user'; // Rôle par défaut

    if (is_array($memberOf) && in_array('CN=GG_AdminEnergie,OU=GG,OU=Groups,DC=smartcity,DC=lan', $memberOf)) {
        $role = 'admin';
    }

    ldap_unbind($ldapConnection);

    return [
        'nom' => $entries[0]['sn'][0] ?? '',
        'prenom' => $entries[0]['givenname'][0] ?? '',
        'email' => $entries[0]['mail'][0] ?? '',
        'role' => $role
    ];
} else {
    $_SESSION['error'] = "Échec de l'authentification : mot de passe incorrect.";
}
```

- La méthode **ldap_get_entries** permet de récupérer les informations utilisateur sous forme de tableau.
- La vérification du champ **memberof** détermine si l'utilisateur appartient au groupe GG_**AdminEnergie**, ce qui lui **confère le rôle admin**.

5. Gestion des Rôles

Le module **attribue un rôle par défaut** (user) à chaque utilisateur, sauf s'il appartient au groupe spécifique des **administrateurs**.

```php
if (is_array($memberOf) && in_array('CN=GG_AdminEnergie,OU=GG,OU=Groups,DC=smartcity,DC=lan', $memberOf)) {
    $role = 'admin';
}
```

- La liste des groupes auxquels l'utilisateur appartient est parcourue pour vérifier la présence du groupe **GG_AdminEnergie**.
- ***En cas de correspondance***, le rôle est modifié en **admin**.

Tests Unitaires

Des tests unitaires ont été réalisés pour vérifier les cas suivants :

- **Connexion au serveur LDAP avec des identifiants valides.**
- **Échec de connexion avec des identifiants incorrects.**
- **Recherche d'un utilisateur inexistant.**
- **Gestion des rôles (user et admin).**
- **Échec de connexion au serveur LDAP simulant un serveur inaccessible.**

Ces tests garantissent le bon fonctionnement des principales fonctionnalités et la gestion appropriée des erreurs.

Gérer les capteurs de SmartCity Energie

Code disponible :
https://github.com/AnthonyCodeDev/SmartCity_CDCGR10/blob/SiteWeb/modele/CapteurModele.php

2. Calcul de la Production des Dernières Heures

Ce code calcule **la production totale d'un capteur donné** sur une période définie en heures.

```php
public function calculerProductionDerniieresHeures($ipCapteur, $heures = 6) {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Calculer la production des dernières heures

    Arguments: ipCapteur (string), heures (int)
    Return: int
    */
    $stmt = $this->pdo->prepare("
        SELECT SUM(valeur) as total
        FROM capteurs_energie
        WHERE id_capteur = :ipCapteur
        AND date_mesure >= NOW() - INTERVAL :heures HOUR
    ");
    $stmt->bindParam(':ipCapteur', $ipCapteur);
    $stmt->bindParam(':heures', $heures, PDO::PARAM_INT);
    $stmt->execute();
    $result = $stmt->fetch();
    return $result['total'] ?? 0; // Retourne 0 si aucun résultat
}
```

- La méthode exécute une requête SQL qui additionne **(SUM(valeur))** les valeurs enregistrées pour un capteur donné (id_capteur) sur une période récente **(NOW() - INTERVAL :heures HOUR)**.
- Si aucune donnée n'est trouvée, **la méthode retourne 0**.

3. Récupération des Capteurs par Type

Cette méthode **retourne une liste de capteurs filtrés par leur type**, incluant des informations comme leur **état** et le **nombre de données associées**.

```php
public function recupererCapteursParType($type) {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Récupérer les capteurs par type

    Arguments: type (int)
    Return: array
    */
    $stmt = $this->pdo->prepare("
        SELECT
            s.IPv4,
            s.Name,
            s.StateUp,
            s.DateAdded,
            COUNT(c.id) as nombre_donnees
        FROM sensor s
        LEFT JOIN capteurs_energie c ON s.IPv4 = c.id_capteur
        WHERE s.ID_SensorType = :type
        GROUP BY s.IPv4, s.Name, s.StateUp, s.DateAdded
        ORDER BY s.StateUp DESC, nombre_donnees DESC, s.DateAdded DESC
    ");
    $stmt->bindParam(':type', $type, PDO::PARAM_INT);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

- La méthode exécute une jointure entre les tables « **sensor »** et **« capteurs_energie »** pour associer les capteurs à leurs données.
- Elle regroupe les résultats par capteur **(GROUP BY)** et les trie selon plusieurs critères : ***état, nombre de données, et date d'ajout.µ***

4. Récupération de Tous les Capteurs

Cette méthode retourne une liste complète de **tous les capteurs enregistrés**, avec des informations de base comme leur état et leur date d'ajout.

```php
public function recupererCapteurs() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Récupérer tous les capteurs avec leur statut

    Arguments: aucun
    Return: array
    */
    $stmt = $this->pdo->query("
        SELECT IPv4, Name, StateUp, DateAdded
        FROM sensor
        ORDER BY DateAdded DESC
    ");
    return $stmt->fetchAll();
}
```

La méthode exécute une requête SQL pour récupérer toutes les informations des capteurs (sensor) et les trie par date d'ajout décroissante **(ORDER BY DateAdded DESC)**.

5. Comptage des Données d'un Capteur

Cette méthode **compte le nombre de données enregistrées** pour un capteur spécifique.

```php
public function compterDonneesCapteur($ipCapteur) {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Compter le nombre de données de consommation pour chaque capteur

    Arguments: ipCapteur (string)
    Return: int
    */
    $stmt = $this->pdo->prepare("
        SELECT COUNT(*) as total
        FROM capteurs_energie
        WHERE id_capteur = :ipCapteur
    ");
    $stmt->bindParam(':ipCapteur', $ipCapteur);
    $stmt->execute();
    $result = $stmt->fetch();
    return $result['total'];
}
```

La méthode utilise **COUNT(*)** pour compter toutes les entrées associées à un capteur spécifique dans la table capteurs_energie.

6. Mise à Jour de l'État d'un Capteur

Cette méthode met à jour l'état **(StateUp)** d'un capteur pour indiquer s'il est actif **(1)** ou non **(0)**.

```php
public function mettreAJourEtatCapteur($ipCapteur, $etat) {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Mettre à jour l'état d'un capteur (Start/Stop)

    Arguments: ipCapteur (string), etat (bool)
    Return: aucun
    */
    $stmt = $this->pdo->prepare("
        UPDATE Sensor
        SET StateUp = :etat
        WHERE IPv4 = :ipCapteur
    ");
    $stmt->bindParam(':etat', $etat, PDO::PARAM_BOOL);
    $stmt->bindParam(':ipCapteur', $ipCapteur);
    $stmt->execute();
}
```

La méthode exécute une requête **SQL UPDATE** pour modifier **l'état du capteur identifié par son adresse IP (IPv4)**.

Tests Unitaires

Des tests unitaires ont été réalisés pour valider les fonctionnalités suivantes :

- **Calcul de la production totale sur une période.**
- **Récupération des capteurs par type et vérification de leurs informations.**
- **Mise à jour de l'état d'un capteur.**
- **Comptage des données associées à un capteur.**
- **Récupération de la liste complète des capteurs.**

Ces tests simulent les interactions avec la base de données garantissant que ***chaque méthode produit les résultats attendus***.

Accueil du site web

Code disponible :
https://github.com/AnthonyCodeDev/SmartCity_CDCGR10/blob/SiteWeb/modele/HomeModele.php

1. Récupération du Message de Bienvenue

Cette méthode retourne un message de bienvenue **basé sur les informations utilisateur** disponibles dans la session.

```php
public function getWelcomeMessage() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne un message de bienvenue

    Arguments: aucun
    Return: string
    */
    $utilisateur = $_SESSION['utilisateur'] ?? null;

    if (!$utilisateur) {
        return "Tout va mal ! 😱";
    }

    $nomUtilisateur = ucwords("$utilisateur[nom] $utilisateur[prenom]");

    return "Tout va bien $nomUtilisateur ! 😊";
}
```

- La méthode vérifie si les **informations utilisateur** sont présentes dans **$_SESSION**.
- Si elles sont disponibles, elle retourne un **message personnalisé** contenant le nom et prénom formatés.

2. Récupération de la Production d'Énergie des Dernières 24 Heures

Cette méthode **calcule la production totale d'énergie sur les dernières 24 heures** pour un type d'énergie donné **(1 pour solaire, 2 pour éolien)**.

```php
public function recupererProductiondER($typeEnergie) {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne la production d'énergie des 24 dernières heures

    Arguments: typeEnergie (int)
    Return: int
    */
    $stmt = $this->pdo->prepare("
        SELECT SUM(valeur) as total
        FROM capteurs_energie
        WHERE type_energie = :typeEnergie
        AND date_mesure >= NOW() - INTERVAL 24 HOUR
    ");
    $stmt->bindParam(':typeEnergie', $typeEnergie, PDO::PARAM_INT);
    $stmt->execute();
    $result = $stmt->fetch();
    return $result['total'] ?? 0; // Retourne 0 si aucune donnée
}
```

- La méthode exécute une requête SQL pour additionner **(SUM(valeur))** les valeurs d'énergie mesurées **au cours des dernières 24 heures**.
- Le type d'énergie est passé en paramètre, permettant de **différencier solaire et éolien**.

3. Récupération des Informations Globales

Ce code **récupère et retourne les données globales** de consommation et de production pour après les afficher sur la vue de la page !

```php
public function recupererInformationsGlobales() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne les informations globales de consommation et production

    Arguments: aucun
    Return: array
    */
    $consommation = $this->recupererConsommationdER(); // Consommation sur 24h
    $productionSolaire = $this->recupererProductiondER(1); // Type 1 pour solaire
    $productionEolienne = $this->recupererProductiondER(2); // Type 2 pour éolien

    return [
        'consommation' => $consommation,
        'productionSolaire' => $productionSolaire,
        'productionEolienne' => $productionEolienne,
    ];
}
```

La méthode appelle d'autres fonctions pour récupérer :

- **La consommation énergétique** des 24 dernières heures.

- **La production d'énergie solaire et éolienne**.

4. Récupération de la Consommation des Derniers 30 Jours

Cette méthode retourne **une liste des consommations énergétiques** par jour **sur les 30 derniers jours**.

```php
public function recupererConsommation30Jours() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne la consommation d'énergie des 30 derniers jours

    Arguments: aucun
    Return: array
    */
    $stmt = $this->pdo->query("
        SELECT DATE(date) as jour, SUM(consommation) as total
        FROM consommation_energie
        WHERE date >= NOW() - INTERVAL 30 DAY
        GROUP BY DATE(date)
        ORDER BY DATE(date) ASC
    ");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

- La requête regroupe les données par date **(GROUP BY DATE(date))** et **calcule la somme** de la **consommation pour chaque jour**.

5. Récupération de la Production des Derniers 30 Jours

Cette méthode **retourne la production quotidienne d'énergie solaire et éolienne** pour les **30 derniers jours**.

```php
public function recupererProduction30Jours() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne la production d'énergie des 30 derniers jours

    Arguments: aucun
    Return: array
    */
    $stmt = $this->pdo->query("
        SELECT DATE(date_mesure) as jour,
               SUM(CASE WHEN type_energie = 1 THEN valeur ELSE 0 END) as solaire,
               SUM(CASE WHEN type_energie = 2 THEN valeur ELSE 0 END) as eolienne
        FROM capteurs_energie
        WHERE date_mesure >= NOW() - INTERVAL 30 DAY
        GROUP BY DATE(date_mesure)
        ORDER BY DATE(date_mesure) ASC
    ");
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

La méthode **utilise des clauses CASE** pour différencier les productions solaire et éolienne dans une seule requête.

6. Récupération des Alertes et Incidents

Cette méthode **récupère les alertes de surcharge** et **les incidents des 3 derniers jours**, limités aux quatre entrées les plus récentes.

```php
public function recupererAlertesProduction() {
    /*
    QUI: Vergeylen Anthony
    QUAND: 18-12-2024
    QUOI: Retourne les alertes de surcharge des 3 derniers jours et les incidents

    Arguments: aucun
    Return: array
    */
    $stmtAlertes = $this->pdo->query("
        SELECT *
        FROM alertes_surcharge
        WHERE date_signalement >= NOW() - INTERVAL 3 DAY
        ORDER BY date_signalement DESC
        LIMIT 4
    ");
    $alertes = $stmtAlertes->fetchAll(PDO::FETCH_ASSOC);

    $stmtIncidents = $this->pdo->query("
        SELECT *
        FROM incidents
        WHERE date_creation >= NOW() - INTERVAL 3 DAY
        ORDER BY date_creation DESC
        LIMIT 4
    ");
    $incidents = $stmtIncidents->fetchAll(PDO::FETCH_ASSOC);

    return [
        'alertes' => $alertes,
        'incidents' => $incidents
    ];
}
```

La méthode exécute **deux requêtes SQL** :

- Une pour récupérer les alertes.
- Une autre pour récupérer les incidents.

7. Tests Unitaires

Des tests unitaires ont été réalisés pour vérifier les points suivants :

- **Le message de bienvenue s'adapte correctement aux informations utilisateur.**
- **Les données de production et de consommation sont calculées et agrégées correctement.**
- **Les alertes et incidents sont récupérés dans les délais et limites spécifiés.**
- **Les données des 30 derniers jours sont regroupées par jour.**

Ces tests ont été réalisés pour **voir le résultat attendu**, **voir si c'est correcte** et **aider au débug** et à la **correction** et la **protections des méthodes de l'app**.

Branches Github

Voici un visuel de nos branches Github. https://github.com/AnthonyCodeDev/SmartCity_CDCGR10

1. **main** : Il s'agit de la branche principale du projet.
2. **GestionAutomatiqueDonnee** : Une branche dédiée à la gestion automatisée des données. Cela inclus des scripts pour automatiser les tâches liées aux données, comme **detect_sensor_overload.py** ou **donneeCapteur.py**.
3. **Ressources** : Cette branche contient des fichiers de ressources, des documents, comme des configurations windows, linux, switch, base de données, etc.
4. **SiteWeb** : Une branche dédiée au développement du site web de SmartCity Énergie.

Conclusion

Cette documentation technique offre *une vue d'ensemble* claire et détaillée des concepts, processus, et fonctionnalités clés développés dans le cadre du projet. Elle permet de *comprendre les mécanismes*, tout en assurant la complétude et la finalité du code !

En adoptant une approche *structurée et explicative*, ce document *vise à faciliter la compréhension, l'utilisation, et l'évolution du projet*.

Enfin, ce travail de documentation *souligne l'importance d'une méthodologie rigoureuse* et d'une attention particulière aux détails, contribuant ainsi à la robustesse et à la pérennité du projet !