**IDOCS** :

**I**nfrastructure de **D**onnées pour l’**O**ptimisation et la **C**entralisation des **S**ystèmes IoT

~

Mise en situation 1 (E1)

Laurence Berville

Formation « Développeur en Intelligence Artificielle »

RNCP 37827 ~ Promotion 2023-2025



Table des matières

[Introduction 2](#_Toc197329135)

[Présentation du projet 2](#_Toc197329136)

[Contexte du projet 2](#_Toc197329137)

[Spécifications techniques 2](#_Toc197329138)

[Extraction des données 3](#_Toc197329139)

[Données issues d’un service web (API) 5](#_Toc197329140)

[Données issues d’une page web 5](#_Toc197329141)

[Données issues d’un fichier 5](#_Toc197329142)

[Données issues d’une base de données relationnelles 5](#_Toc197329143)

[Données issues d’un système big data 6](#_Toc197329144)

[Requêtage des données 7](#_Toc197329145)

[API Open-Meteo 7](#_Toc197329146)

[API du ministère de l'Éducation nationale, Sports et Jeunesse 7](#_Toc197329147)

[Requêtes de type SQL 7](#_Toc197329148)

[Requêtes depuis un système big data 8](#_Toc197329149)

[Agrégation des données 8](#_Toc197329150)

[Préparation & Agrégation des données 8](#_Toc197329151)

[Jeu de données final 9](#_Toc197329152)

[Création de la base de données 10](#_Toc197329153)

[Modélisation des données 10](#_Toc197329154)

[Modèle physique des données (MPD) 11](#_Toc197329155)

[Choix du système de gestion de la base données (SGBD) 11](#_Toc197329156)

[Création de la base de données 11](#_Toc197329157)

[Import des données 11](#_Toc197329158)

[Conformité RGPD 11](#_Toc197329159)

[Développement de l’API 13](#_Toc197329160)

[Spécifications fonctionnelles et techniques 13](#_Toc197329161)

[Conception de l’architecture de l’API 13](#_Toc197329162)

[Documentation 13](#_Toc197329163)

[Perspectives et améliorations 13](#_Toc197329164)

[Conclusions 13](#_Toc197329165)

[Annexes 14](#_Toc197329166)

# Introduction

Ce rapport constitue un livrable (E1 : Collecte, stockage et mise à disposition des données d’un projet IA) indispensable à la validation de ma Licence professionnelle ([École IA Microsoft By Simplon.](https://simplon.co/formation/ecole-ia-microsoft/23)), mention Métiers du décisionnel et de la statistique, spécialisée en Développement en Intelligence Artificielle, promotion 2023-2025. Ce projet s'inscrit dans le cadre du [RNCP37827](https://www.francecompetences.fr/recherche/rncp/37827/). Ce projet est porté par moi-même, en tant que conceptrice et développeuse de la solution. L’approche vise à centraliser les données issues des différentes sources dans un système unique, facilitant l’analyse, la visualisation, le partage et à terme, le pilotage à distance ou la prédiction des consommations futures. Ce rapport présentera dans un premier temps le cadre général du projet, puis les choix techniques et fonctionnels réalisés pour répondre aux enjeux identifiés. Il exposera enfin les perspectives d’évolution, notamment vers une automatisation plus poussée et une intégration à des systèmes domotiques complets.

## Présentation du projet

Dans un contexte de transition énergétique et de recherche constante d’optimisation des ressources, l'Internet des Objets (IoT) s'impose comme une solution innovante pour la gestion intelligente des habitations. Le projet présenté dans ce rapport s’inscrit dans cette dynamique, en explorant l’intégration de capteurs connectés et d’API pour suivre, analyser et dans un futur proche, ajuster en temps réel la consommation domestique de chauffage, tout en surveillant la température ambiante via des thermomètres connectés. L’objectif est double : d’une part, permettre une gestion plus fine et personnalisée des consommations énergétiques à l’échelle d’un foyer ; d’autre part, proposer une architecture technique fiable, interopérable et évolutive, reposant sur les principes de l’IoT et de l’analyse de données en temps réel. Grâce à des objets connectés, il est possible de collecter des données précises, de détecter des anomalies, et d’automatiser certaines décisions (comme la régulation du chauffage en fonction de la température ambiante).

## Contexte du projet

Le projet a été déployé dans une maison personnelle située à Tours, construite au début du XXe siècle. Cette habitation, répartie sur trois étages et occupée par deux personnes, présente des caractéristiques architecturales anciennes, notamment une isolation thermique peu performante. Toutefois, elle est équipée d’une chaudière récente, installée il y a moins d’un an, offrant ainsi une base moderne pour l’optimisation du chauffage. L’environnement domestique constitue un terrain d’expérimentation concret mais également contraint par la structure du bâtiment et l’hétérogénéité des équipements existants. Les principales contraintes techniques rencontrées sont liées à la découverte et à l’apprentissage progressif des technologies IoT. Certaines notions, comme la communication entre objets connectés, l’intégration d’API ou encore la gestion centralisée des données via une plateforme domotique, ont nécessité un temps d’appropriation. Ce projet s’inscrit donc aussi comme une démarche formatrice personnelle, permettant de renforcer mes compétences en ingénierie des données et en technologies embarquées.

Le budget alloué au projet n’était pas initialement plafonné, l’objectif étant de privilégier des solutions durables, évolutives et utiles à long terme. Un détail des dépenses engagées est présenté en annexe pour plus de transparence. En ce qui concerne l’organisation du travail, la planification a été conduite de manière flexible, selon une approche « au fil de l’eau », en s’adaptant à la fois aux disponibilités, à l’évolution des connaissances techniques et aux résultats observés à chaque étape du déploiement.

## Spécifications techniques

La réalisation de ce projet s’est appuyée sur un ensemble cohérent de technologies et de services externes visant à assurer la collecte, le traitement, l’analyse et la visualisation des données issues des capteurs IoT déployés dans l’habitation. Deux types de thermomètres connectés de la marque Govee ont été utilisés : le premier permet un accès direct à la température / humidité via un écran (Bluetooth), ainsi qu'une consultation à distance via une application mobile ; le second, dépourvu d’affichage, ne permet la lecture des données que sur demande via l'application, mais offre une plus grande souplesse d’interrogation à distance. À cela s’ajoutent, de la marque Tapo, des prises connectées (2 types), des boutons multifonctions, une passerelle de communication centralisant les échanges entre certains objets, ainsi que l’intégration des données de consommation électrique issues du compteur Linky via l’interface proposée par Enedis. L’ensemble de ces dispositifs permettra une remontée des données environnementales et énergétiques, facilitant leur exploitation à des fins d’analyse fine et de visualisation dynamique.

Le langage principal utilisé est **Python**, en raison de sa richesse en bibliothèques spécialisées et de sa compatibilité avec la majorité des API et capteurs connectés. Un environnement dédié a été créé pour le développement en Python, afin de garantir la reproductibilité, l’isolation des dépendances et la stabilité du projet. Cet environnement virtuel regroupe l’ensemble des bibliothèques nécessaires. Cette structuration facilite le déploiement et la maintenance du code tout au long du projet. Pour la gestion de version du code, **GitHub** a été utilisé, permettant un suivi rigoureux de l’évolution du projet et la traçabilité des modifications. L’environnement Python s’est appuyé sur un large éventail de packages (liste non exhaustive), des besoins variés :

* **Communication et asynchronisme :** bleak (Wifi), asyncio, requests (API), uuid (Object),
* **Analyse et visualisation des données :** pandas, summarytools, sweetviz, plotly (Graphique), numpy
* **Traitement de données temporelles :** datetime, schedule, holidays
* **Traitement de données météorologique et géographiques :** geopy, openmeteo
* **Scraping et automatisation web :** selenium
* **Stockage NoSQL et fichiers volumineux :** pymongo, gridfs
* **Traitement de texte et de documents :** nltk, PyPDF2
* **Tests unitaires :** pytest

Afin de faciliter l’interaction avec les données et d’en garantir l’accessibilité, une interface utilisateur a été développée à l’aide de Streamlit. Conçue pour évoluer de manière itérative avec le développement technique du projet, elle permet un accès direct et intuitif aux principales fonctionnalités. L’utilisateur peut notamment y renseigner les caractéristiques du bâtiment (nombre de façades, étages, pièces, ouvertures, orientation, type de chauffage…), consulter les mesures collectées par les capteurs et accéder aux analyses générées. L’objectif a été de programmer des scripts robustes et pérennes permettant une extraction automatisée des données, tout en assurant leur traçabilité et leur stockage dans les formats appropriés. Pour la gestion des données structurées, **SQL** a été choisi, avec une base de données relationnelle consultable via l’outil **DBeaver**. Dans ce cadre, le **diagramme de flux de données** (DFD) modélise le chemin que suivent les données depuis les objets connectés jusqu’à leur visualisation ou analyse, Figure 1.

# Extraction des données

La réussite du projet repose en grande partie sur la capacité à collecter de manière fiable, continue et automatisée des données issues de multiples sources hétérogènes. Cinq sources principales ont été intégrées, chacune fournissant des informations complémentaires indispensables à l’analyse et à la gestion énergétique du bâtiment. Toutes ces sources alimentent un pipeline automatisé de collecte, orchestré par des scripts Python spécifiques. Les données brutes sont ensuite stockées selon leur nature dans des bases de données non relationnelles (NoSQL) ou relationnelles (SQL) ou avant traitement et analyse.

La première source est l’API, notamment celles fournissant des données météorologiques ou des informations de calendrier scolaire via le ministère de l’Éducation nationale. La deuxième source est constituée de fichiers plats (CSV, Excel) fournis par les opérateurs d’électricité, le gestionnaire de réseau Enedis, et le fournisseur de gaz ; ces fichiers contiennent un historique des consommations, avec des formats variables selon les fournisseurs et le type de données (horaire, journalier, mensuel, annuel). La troisième source repose sur du web scraping automatisé de pages web afin de récupérer des documents PDF, comme les grilles tarifaires ou autres données contextuelles. La quatrième source est une base de données interne contenant des séries temporelles de température intérieure issues de capteurs installés antérieurement dans le bâtiment. Enfin, une base de données MongoDB a été créée pour centraliser les données non structurées ou semi-structurées, facilitant ainsi leur accès et leur manipulation.

Figure  : Diagramme de flux de données représentant la circulation de l'information dans le système IDOCS.



[Traitement

T°C …]

[Traitement]

[Récupération par Wifi, Bluetooth]

[Récupération par API]

[Mise

en Base]

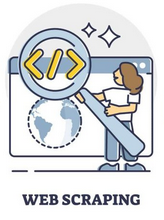
[Récupération par API et

fichiers plats]

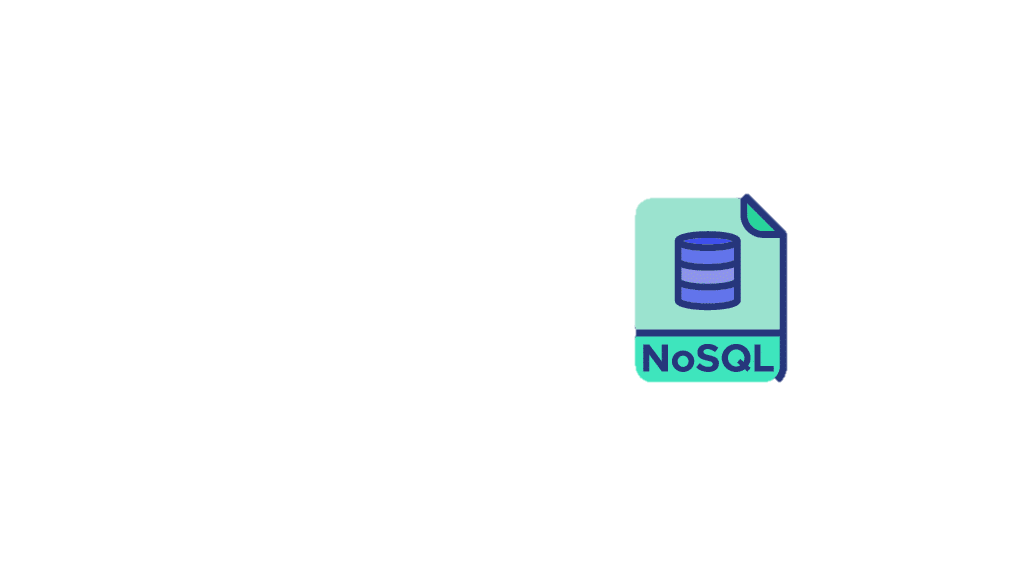


[Traitement

kWh…]



[Récupération par scraping]



[Mise

en

Base]

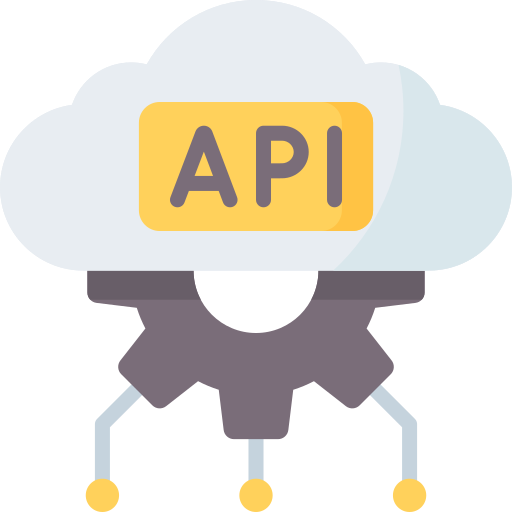
[Traitement]



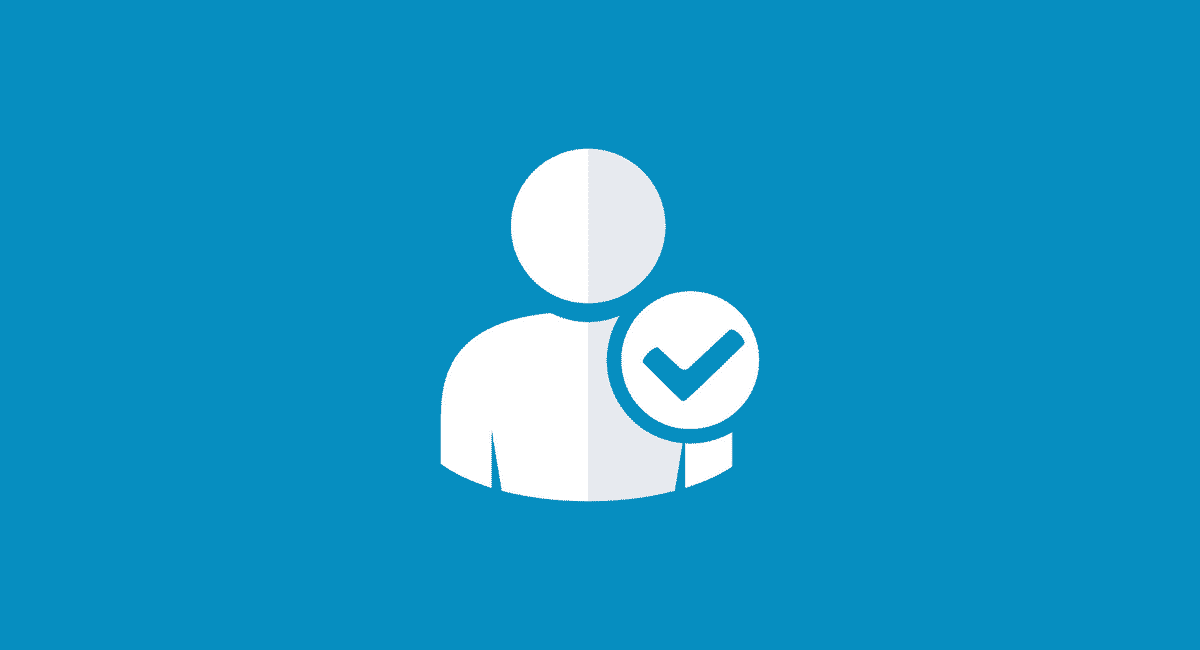
[Traitements]

[Récupération

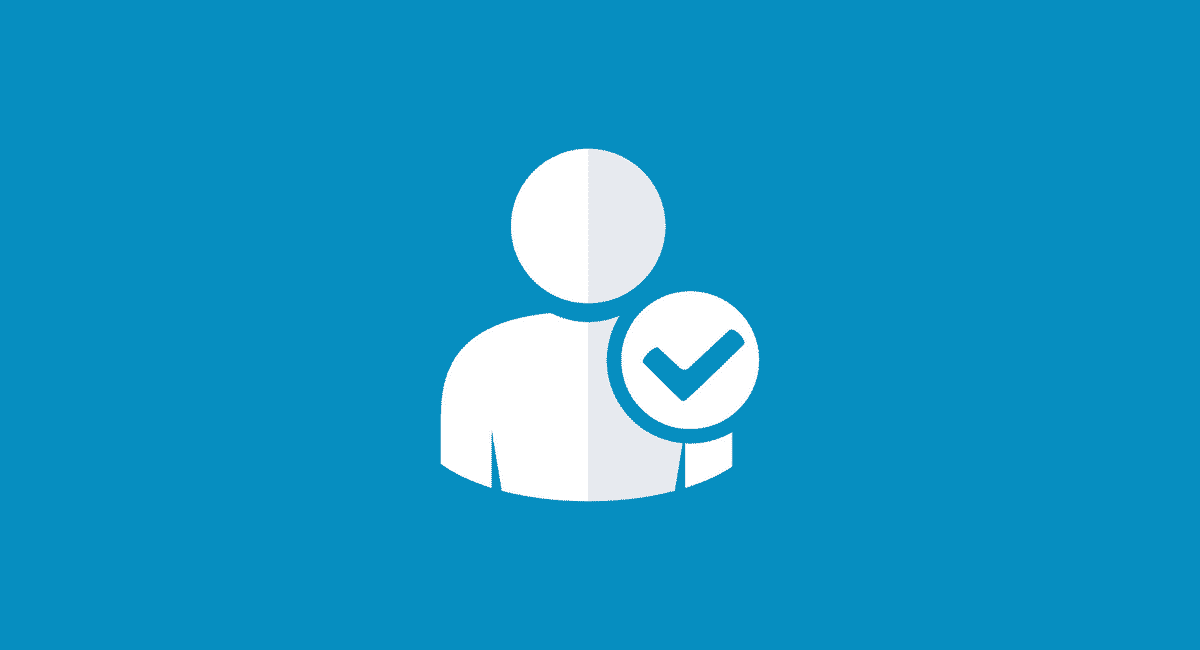
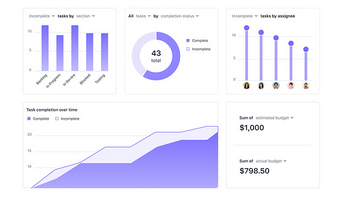
par API]



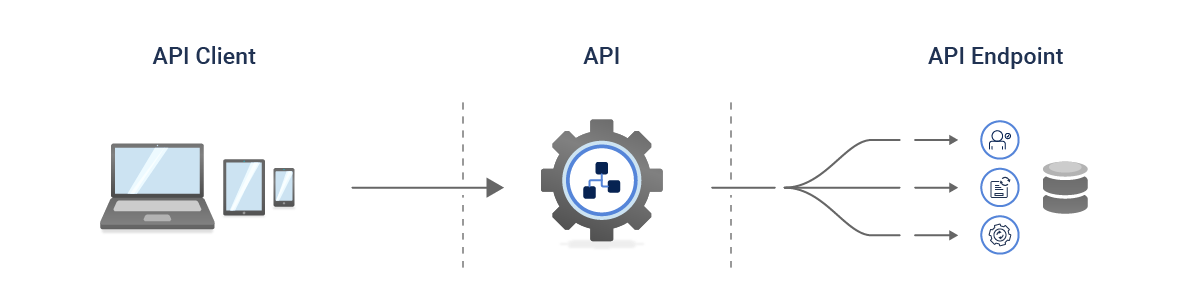
Ministères de l'Éducation nationale, Sports et Jeunesse



[Insertion dans formulaire]



[Récupération des données du bâtiment]



## Données issues d’un service web (API)

Dans le cadre de ce projet, plusieurs services web exposant des **API REST** ont été intégrés afin d’enrichir le système de données. Ces API ont été sélectionnées pour leur pertinence dans l’analyse des usages énergétiques et dans l’adaptation des comportements domestiques aux contextes extérieurs (météo, calendrier, activité). Chaque API a fait l’objet d’un script d’appel dédié, avec gestion du cache, des erreurs et de la fréquence d’appel, assurant la pérennité et la résilience de la collecte.

|  |  |  |
| --- | --- | --- |
| Service API | Type de données | Utilité dans le projet |
| Open-Meteo | Latitude, Longitude, ainsi que pour chaque heure : température (°C), humidité, pluie (0 ou 1), neige (0 ou 1), couverture nuageuse (%), évaporation, vitesse du vent, direction du vent et température du sol. | Utilisation des données météo pour calculer le delta entre la température interne et externe. Recherche des facteurs influençant la quantité de chauffage nécessaire. |
| Ministère de l’Éducation nationale | Zones, jours fériés, vacances | Facteurs influençant, possiblement la présence des utilisateurs du bâtiment. |
| Chaudière connectée (API fabricant) | Statut, consommation, réglages | Anticipation des besoins en chauffage, corrélations climatiques |
| Jeedom (domotique) | Activité des capteurs, données de consommation électriques des prises connectés aux chauffages. | remonter des mesures |

## Données issues d’une page web

Scraping site gaz (factures) Page web beautifulsoup4, selenium, webdriver-manager Hebdomadaire Historique et analyse des coûts et usages du gaz

## Données issues d’un fichier

En complément des données collectées via API, le projet a également intégré plusieurs jeux de données provenant de fichiers fournis par les opérateurs énergétiques. EDF, ne fournit pas d’API. Enedis et GRDF fournissent une API aux personnes morale seulement. Ces fichiers, aux formats hétérogènes (CSV, Excel), ont été récupérés via les espaces clients des fournisseurs tels que **EDF**, **Enedis** et **GDF**. Ils contiennent des informations historiques précieuses sur la consommation d’électricité, de gaz ou encore sur la facturation. Ces fichiers ont été exploités pour reconstituer l’évolution des consommations sur plusieurs années.

|  |  |  |  |
| --- | --- | --- | --- |
| Fournisseur | Type de fichier | Contenu | Format(s) |
| EDF | Historique client | Index, consommation journalière | .xlsx |
| Enedis | Télérelève Linky | Consommation / heure ou par jour (kW) | .xlsx |
| GRDF | Relevés de gaz | Volume consommé (m3), Coefficient de conversion, Energie consommée (kWh) – date variable | .xlsx |
| Govee | Relevés | Température, Humidité de chaque pièces – 15 minutes | .csv |

## Données issues d’une base de données relationnelles

Une base de données relationnelle a été spécifiquement conçue pour structurer et centraliser l’ensemble des données nécessaires au bon fonctionnement du projet. Hébergée localement et visualisée à l’aide de l’outil DBeaver, elle joue un rôle central dans l’agrégation des données issues de sources multiples (API, fichiers plats, capteurs locaux), leur historisation, ainsi que leur mise à disposition pour les traitements analytiques. En plus des données collectées via les trois premières sources, cette base intègre également les séries temporelles de température intérieure extraites d’anciennes mesures effectuées dans le bâtiment.

Pour assurer la cohérence de ces données entre les différentes pièces, des informations structurelles sur le bâtiment (noms et caractéristiques des pièces, nombre d’ouvertures, étages, orientation, etc.) ont été collectées et intégrées à la base. Celle-ci est organisée autour de plusieurs tables logiques, structurées selon trois grands axes : les caractéristiques du bâtiment, les données de capteurs, et les données contextuelles externes. La base relationnelle est conçue pour évoluer : son schéma est flexible et documenté pour intégrer de nouveaux capteurs, indicateurs ou bâtiments à terme. Un mécanisme d’identification unique (uuid) est utilisé pour tracer chaque entrée de mesure, assurant une parfaite traçabilité des sources et des dates de collecte. L’ensemble offre une base solide pour l’analyse temporelle, spatiale et comparative des données environnementales et énergétiques, tout en garantissant la cohérence, la qualité et la pérennité des données collectées.

### Caractéristiques du bâtiment

Cette table regroupe les informations statiques sur les pièces du logement (surface, nombre d’étages, type de chauffage…). Elle est conçue pour être enrichie dynamiquement via un formulaire disponible sur le front-end Streamlit, permettant à terme l’alimentation automatique de cette table à partir de données utilisateur.

### Mesures de température internes

Une table stocke les relevés de température intérieure issus des deux types de thermomètres connectés placés à différents endroits de la maison (RDC et étages). Les données historiques collectées jusqu’à janvier 2025 ont été nettoyées et insérées en base via scripts Python. Les données postérieures sont automatiquement ajoutées via API pour les relevés distants ou via Bluetooth lorsque l’utilisateur est présent physiquement dans le bâtiment, ce qui introduit une logique d’ingestion hybride (locale / distante).

## Données issues d’un système big data

Bien que le projet se déroule dans un environnement domestique, la **volumétrie des données générées** par les capteurs IoT, les appels réguliers aux API, et l’intégration de fichiers historiques nécessite une gestion structurée, inspirée des systèmes Big Data. Avec des relevés de température toutes les 10 minutes, des données de consommation horodatées, et des flux météorologiques ou de pilotage thermique fréquents, le volume total de données stockées s’accroît rapidement, imposant des solutions adaptées pour l’**ingestion, la conservation et l’analyse à moyen terme**.

Pour répondre à ces exigences, les choix suivants ont été mis en œuvre :

* **Stockage hybride** :
  + Les données structurées (relevés de consommation, mesures météorologiques, historiques) sont centralisées dans une **base relationnelle SQL**, permettant des requêtes optimisées et une visualisation via **DBeaver**.
  + Les données semi-structurées ou volumineuses (logs bruts des capteurs, documents PDF, fichiers JSON) sont stockées dans une base **MongoDB**, avec gestion des fichiers lourds via **GridFS**, pour assurer la scalabilité.

Même si l’infrastructure ne relève pas d’un véritable cluster distribué (type Hadoop ou Spark), l’architecture mise en place respecte les **principes fondamentaux des systèmes Big Data** : gestion de flux de données hétérogènes, automatisation des processus de collecte, stockage structuré et semi-structuré, traitement différé, scalabilité potentielle. Cela constitue une **base solide pour une évolution future** vers une architecture cloud ou distribuée, si le projet venait à s’étendre ou à être mutualisé.

# Requêtage des données

Dans le cadre de ce projet, l’extraction et l’exploitation des données s’appuient sur un système de gestion de base de données relationnelle **(SQLite)**, conçu pour stocker de façon structurée les informations météorologiques et géographiques associées au bâtiment étudié. Les requêtes sont réalisées à l’aide du module sqlite3, combiné avec **pandas** pour la manipulation tabulaire et geopy pour la géolocalisation.

## API Open-Meteo

### Objectifs des requêtes API Open-Meteo

Les requêtes ont plusieurs objectifs :

* **Récupérer les coordonnées géographiques** d’une ville spécifiée, via le service Nominatim.
* **Interroger l’API Open-Meteo** à l’aide de ces coordonnées et d’une plage de dates définie.
* **Extraire les variables pertinentes** : température, humidité, précipitations, couverture nuageuse, etc.

### Résultats et automatisation

Les scripts sont conçus pour :

* Insérer en masse les données météorologiques (insert\_weather\_data) dans un format structuré (date + variables horaires),
* Fermer proprement la connexion et s’assurer de la validité des insertions via des logs et des commits.
* L’usage de **requests\_cache** et **retry\_requests** assure la robustesse des appels API, même en cas de latence réseau ou d’erreurs temporaires.

**Exemple de cas d’usage**

Une requête lancée pour la ville de **Tours**, entre deux dates données, permet de récupérer plusieurs milliers d’entrées horaires que l’on peut ensuite exploiter dans les modèles de prédiction ou les visualisations (via Streamlit, Plotly, etc.). Cette démarche permet de **pérenniser la collecte** via des appels automatisés tout en maintenant la cohérence entre les sources de données et la base développée.

## API du ministère de l'Éducation nationale, Sports et Jeunesse

### Données géographiques et météorologiques

A faire

## Requêtes de type SQL

Un script en Python gère une base de données SQLite utilisée pour stocker, structurer et interroger des données énergétiques, thermiques et météorologiques. Il utilise sqlite3 pour la connexion, pandas pour le traitement des données, et dotenv pour la configuration via des variables d’environnement. Des fonctions créent des tables relationnelles si elles n'existent pas :

* **Batiment** : informations générales sur les bâtiments.
* **Piece** : détails des pièces associées à chaque bâtiment.
* **ConsoHeureElec** et **ConsoJourGaz** : consommation électrique horaire et consommation de gaz journalière.
* **TemperaturePiece** : mesures de température et humidité par pièce.
* **city\_info** : coordonnées géographiques des villes.
* **weather** : données météorologiques horaires, liées à city\_info.

Chaque fonction de création utilise CREATE TABLE IF NOT EXISTS et inclut des clés primaires/étrangères pour l’intégrité relationnelle. Des **fonctions d’insertion** permettent d’ajouter ou de mettre à jour des données (insert\_weather\_data, insert\_or\_update\_city\_info). Les **fonctions de lecture** comme recuperer\_conso\_data et recuperer\_conso\_dataGAZ extraient les données dans un DataFrame pandas, avec conversion des horodatages et gestion des valeurs invalides. Les dates existantes peuvent aussi être récupérées pour éviter les doublons (get\_existing\_dates).

Une seconde table centralise les coordonnées géographiques des lieux étudiés : la maison principale à Tours, mais aussi d’autres villes suivies dans les appels API météo. Cette structure permet de faire le lien entre les données météorologiques collectées et le lieu de vie, en vue d’analyses croisées. - Insérer ces données dans la base SQLite, dans deux tables : city\_info (localisation) et weather\_data (météo horodatée). - Créer automatiquement les tables si elles n’existent pas (creer\_table\_city\_info, creer\_table\_weather),

- Alimenter ou mettre à jour les informations liées à la ville (insert\_or\_update\_city\_info),

## Requêtes depuis un système big data

Objectif des requêtes et résultats obtenus (détails possible en annexe)

# Agrégation des données

Chaque source a nécessité l’écriture de scripts de nettoyage et de transformation personnalisés, en raison de la variété des formats, des structures internes des fichiers et des conventions de nommage propres à chaque fournisseur. Les bibliothèques pandas, datetime, numpy, ainsi que des utilitaires de type summarytools et sweetviz ont été utilisés pour faciliter l’exploration initiale, la validation des formats et la création de pipelines de transformation. Les données une fois standardisées ont été injectées dans la base SQL utilisée par le projet, afin d’en permettre l’interrogation dynamique et l’intégration aux tableaux de bord. Cette étape a permis de capitaliser sur des données historiques riches, qui sont venues compléter les mesures en temps réel pour une analyse plus globale.

## Préparation & Agrégation des données

Le traitement des fichiers d’entrée inclut plusieurs étapes essentielles de **nettoyage, normalisation** et de **gestion des données manquantes ou redondantes**, avant insertion dans la base de données.

### Extraction de métadonnées

Le nom du fichier est utilisé pour déduire le nom de la pièce via une fonction personnalisée. Une vérification s’assure que le format du nom respecte une structure attendue (présence d’un underscore \_), avec gestion des erreurs en cas de non-conformité.

### Nettoyage sémantique des labels de pièces

Une fonction homogénéise les noms de pièces issus des fichiers utilisateurs : suppression d’espaces, standardisation des libellés ambigus ou incohérents (ex. : "Emma" devient "ChambreE").

### Chargement et filtrage des données de consommation électrique

* + Le fichier CSV est lu en sautant les deux premières lignes (métadonnées inutiles).
  + Les colonnes utiles sont sélectionnées et renommées.
  + Les horodatages sont convertis au format datetime ISO standard, en UTC.
  + Un filtrage évite les doublons : seules les dates absentes de la base (get\_existing\_dates) sont conservées pour insertion.

### Prétraitement des données de consommation de gaz

* + Les fichiers gaz sont aussi nettoyés : conversion des virgules en points, transformation des colonnes numériques (m³ et coefficient), normalisation des dates.
  + Les lignes corrompues ou mal formatées sont écartées via errors="coerce".
  + Une détection des doublons temporels empêche les insertions redondantes.

### Gestion des erreurs et retour utilisateur

À chaque étape, des messages clairs (erreurs, succès, avertissements) sont affichés à l’utilisateur via Streamlit, assurant une interface robuste et informative.

Normalisation des différentes sources pour un jeu de données unique

## Jeu de données final



# Création de la base de données

Dans le respect du RGPD en élaborant les modèles conceptuels et physiques des données à partir des données préparées et en programmant leur import afin de stocker le jeu de données du projet.

## Modélisation des données

La méthode Merise est une approche structurée de conception et de gestion de systèmes d'information, largement utilisée en France. Elle repose sur une séparation rigoureuse entre les données et les traitements, et se décline en plusieurs niveaux d’abstraction : conceptuel, logique et physique. Le formalisme Merise s'appuie sur des outils comme le Modèle Conceptuel de Données (MCD), qui représente les entités et leurs relations sous forme d’un schéma entité-association, et le Modèle Organisationnel des Traitements (MOT) ou Modèle Conceptuel des Traitements (MCT) pour décrire les processus métiers indépendamment de toute implémentation technique. Dans un projet, Merise permet d’analyser les besoins, de concevoir une base de données cohérente (via le MCD, puis le Modèle Logique de Données ou MLD), et d’organiser les flux de données et traitements de manière modulaire. Pour l'appliquer efficacement, on commence par l’analyse fonctionnelle du système à informatiser (étude des processus), puis on modélise les données de manière conceptuelle avant de passer aux structures logiques et physiques adaptées au SGBD cible. Cette méthode est particulièrement pertinente dans les projets où la rigueur, la documentation et la clarté de la structure des données sont essentielles.

Dans le cadre d’un projet d’**IA appliquée à une maison connectée**, la méthode **Merise** permet de structurer efficacement les données générées par les capteurs IoT (température, luminosité, mouvement, etc.) et de formaliser les traitements intelligents associés. On commence par élaborer un **Modèle Conceptuel de Données (MCD)** : on identifie les **entités** comme *Capteur*, *Pièce*, *Mesure*, *Période*, *Utilisateur*, et leurs **relations** (par exemple, *Un capteur enregistre plusieurs mesures*, *Une mesure est prise dans une pièce*). Cela permet de construire une vision claire et indépendante du système technique. Ensuite, on transforme ce MCD en **Modèle Logique de Données (MLD)**, en tenant compte des contraintes du SGBD choisi (ex. : PostgreSQL, MongoDB).

En parallèle, le **Modèle Conceptuel des Traitements (MCT)** décrit les processus métier : *Enregistrement d'une mesure*, *Détection d’anomalie*, *Recommandation d’action (ex. : allumer le chauffage)*. Ces traitements peuvent ensuite être affinés dans un **Modèle Physique des Traitements** prenant en compte les technologies spécifiques (flux MQTT, stockage en base de données, appel à des modèles IA, etc.). Grâce à Merise, le système est documenté de manière rigoureuse, facilitant la mise en œuvre, la maintenance, et la communication entre les acteurs du projet.

Voici un exemple de schéma **MCD (Modèle Conceptuel de Données)** pour un projet IoT domestique utilisant la méthode Merise :

le schéma MCD IoT Domestique

Ce schéma montre :

* Un **Utilisateur** qui possède un ou plusieurs **Capteurs**
* Un **Capteur** qui enregistre plusieurs **Mesures**
* Chaque **Mesure** est associée à une **Pièce** de la maison
* Voici le schéma **MLD (Modèle Logique de Données)** correspondant à l’exemple IoT domestique, basé sur la méthode Merise :
* 📎 Télécharger le schéma MLD IoT Domestique
* Ce MLD détaille les **tables**, **clés primaires (PK)** et **clés étrangères (FK)**, en cohérence avec le MCD précédent. Tu peux t’en servir comme base pour créer la base de données relationnelle du projet.

## Modèle physique des données (MPD)

Voici le Modèle Physique des Données (MPD) pour ton projet IoT domestique, basé sur le MLD :

📎 Télécharger le schéma MPD IoT Domestique

Ce MPD précise :

Les types SQL pour chaque champ (INT, VARCHAR, FLOAT, TIMESTAMP)

Les clés primaires et clés étrangères

La structure finale prête pour l’implémentation dans un SGBD (comme PostgreSQL ou MySQL)

## Choix du système de gestion de la base données (SGBD)

Dans le cadre de ce projet IoT appliqué à une maison individuelle, le **choix du Système de Gestion de Base de Données (SGBD)** s’est porté sur **SQLite**, une solution **légère, embarquée et sans serveur**, particulièrement adaptée aux projets à petite échelle ou en phase de prototypage. Le système vise à collecter et structurer des données issues de capteurs connectés (température, mouvement, etc.), à les stocker localement, puis à les exploiter via des modules d’intelligence artificielle pour des actions automatisées (détection d’anomalie, recommandations).

SQLite présente plusieurs avantages clés dans ce contexte : il est **facile à déployer**, **ne nécessite aucune configuration réseau** (car les données sont stockées dans un fichier local), et consomme très peu de ressources, ce qui est idéal pour une intégration sur des dispositifs embarqués comme un **Raspberry Pi** ou tout autre microcontrôleur local. De plus, malgré sa simplicité, SQLite offre la **puissance d’un moteur SQL complet**, permettant de maintenir une structure relationnelle rigoureuse (intégrité référentielle, requêtes complexes) comme définie dans le modèle Merise.

Bien que SQLite ne soit pas conçu pour gérer de très gros volumes de données ni un haut niveau de concurrence, il est parfaitement **adéquat pour une maison connectée à échelle individuelle**, où la volumétrie reste modérée et où l'accès aux données est généralement séquentiel ou mono-utilisateur. Ce choix permet de **concilier légèreté, portabilité et compatibilité** avec des outils d’analyse et d’IA développés en Python.

## Création de la base de données

## Import des données

Alimentation de la base de données à partir de scripts d’import

## Conformité RGPD

Registre et procédure de conformité RGPD le cas échéant

# Développement de l’API

*Mettre à disposition le jeu de données en utilisant l’architecture REST pour permettre l’exploitation des données par les autres composants du projet.*

## Spécifications fonctionnelles et techniques

Analyses des spécifications fonctionnelles et techniques

Intégrer un schéma fonctionnel si possible.

## Conception de l’architecture de l’API

Conception de l’architecture de l’API, les points de terminaison, les règles d’accès, etc.

# Documentation

Rédiger une documentation technique portant sur :

* extraction des données depuis les bases de données : sélection, filtrage, jointure, optimisations, etc
* agrégation : dépendances, nettoyage, homogénéisation, etc
* création de la base : dépendances, commandes pour l’exécution des scripts, conformité RGPD, etc
* API : points de terminaison, règles d’authentification, etc

# Perspectives et améliorations

Déploiement et usage à venir de l’API développée

Les points possibles d’améliorations si il y en a

# Conclusions

# Annexes

Annexes possibles :

* Informations plus détaillées sur les sources de données
* Requêtes ~~SQL et~~ Big Data : exemples d’instructions et commentaires associés
* Règles d’agrégation : fournir ici les règles si complexes et nombreuses pour alléger le rapport
* Base de données : schéma de la BDD complet si étendue, schéma par domaine spécifique de la BDD, dictionnaires de données, modélisation (MCD, MLD, MPD), etc
* RGPD : registre de traitement des données personnelles, procédure de conformité, etc
* API : documentations

