Certification
Développeuse en Intelligence Artificielle

E1 : Gestion des données

Projet “Problem Management”

Octobre 2024

- Rapport rédigé par : Manon Platteau

Sommaire

I. Automatiser l'extraction des données depuis un service web

A. Contexte du projet

Le projet "Problem Management" est un projet réel rencontré lors de mon alternance au sein de l'entreprise Roquette, spécialisée dans l'industrie agroalimentaire. Ce projet est né d'une demande d'une partie de l'équipe Infrastructure & Opérations qui se charge de la résolution des incidents informatiques dans l'entreprise. Les incidents sont signalés de manière automatique ou par les utilisateurs et stockés sur une plateforme "Easy Vista". Pour ce projet, nous nous concentrons sur les incidents remontés automatiquement, ce sont les incidents "Zabbix". Les opérateurs de l'équipe regroupent les incidents en groupes d'incidents semblables appelés "problèmes". Cela peut permettre d'identifier des causes communes et ainsi de faciliter et accélérer le traitement des incidents. Jusqu'alors, la tâche de regroupement des incidents en problèmes est réalisée manuellement par les opérateurs, ce qui est une opération très coûteuse en termes de temps. La demande du service I&Ops est donc d'automatiser ce regroupement d'incidents semblables pour obtenir un gain de temps et de productivité considérable. L'augmentation de "problèmes" (incidents semblables) permettra à long terme de réduire le nombre d'incidents et de réduire les coûts liés à leur résolution.

Le budget alloué à ce projet dans le cadre professionnel est de 107.5K€ et comprend les labor costs (100K€), l'intervention de prestataires (1.5K€) et les coûts de fonctionnement de l'application en run (6K€).

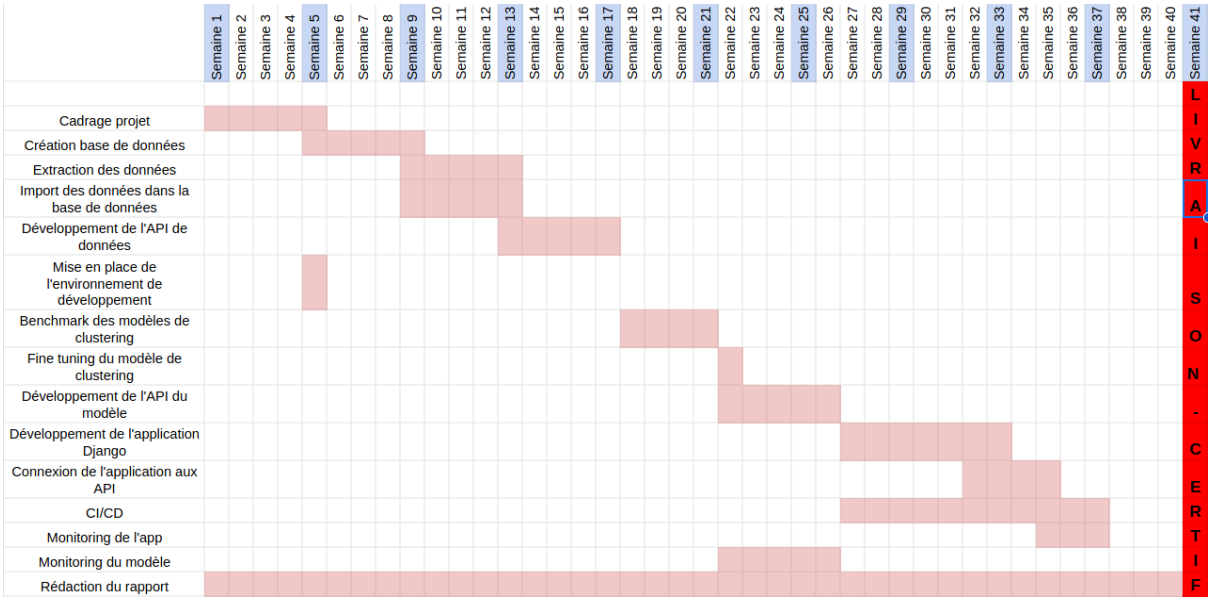
Les membres de l'équipe projet et la répartition des tâches sont :

- Equipe support IT (Bharat, Riddav) : interlocuteurs principaux côté business, rédigent les user stories, formalisent les besoins et réalisent les feedbacks ;
- Data Scientists : Elisa RIOU et Manon PLATTEAU, responsables de l'entraînement du modèle, du développement de l'application et de la restitution des résultats ;
- ML Engineer : Mathieu KLIMCZAK, mise en production du module de data science ;
- Data Engineer : Alexandre VEREPT, récupération des données sur Easy Vista et mise à disposition dans le datawarehouse.

Le contexte du projet dans le cadre pédagogique diffère légèrement pour respecter les exigences de la certification et les contraintes de sécurité de l'entreprise. Les points détaillés à partir de maintenant seront exclusivement propres au cadre pédagogique.

D’un point de vue technique, l’objectif est de fournir une application web permettant à l’équipe de fournir des incidents à répartir dans des “problèmes” (groupe d’incidents semblables). L’application renverrait les incidents associés à un problème. Il est prévu d’utiliser un modèle de clustering Kmeans, un modèle de sentence-transformers pour former des embeddings (format de données accepté par le modèle de clustering), ainsi qu’un modèle GPT-4 OpenAI pour le nommage des problèmes identifiés lors de l’entraînement du modèle. L’environnement technique est composé de machines virtuelles hébergées sur Azure, équipée d’un IDE Visual Studio Code. Nous utilisons le service Cloud Azure.

L’organisation du travail suit une méthode agile (*le détail de cette organisation est présenté dans le rapport E4*) et la planification des tâches pour le projet a donné lieu à un rétro-planning, débutant en janvier 2024 et ayant pour objectif de livrer l’application en octobre 2024 :
(*voir annexe 1 en taille réelle*)



B. Spécifications techniques

Pour mener à bien ce projet, le langage python est utilisé. Nous utilisons le service cloud Azure pour l’hébergement de nos bases de données, le déploiement de nos API et de notre application web, l’utilisation d’une ressource OpenAI ainsi que pour le monitoring de notre application. Le code est rédigé dans l’IDE Visual Studio Code et il est versionné sur un dépôt Github. Il est disponible à l’adresse : https://github.com/Manonp59/prbmg. Les pipelines de CI/CD sont exécutés dans Github Actions.

C. Extraction des données

Nous avons besoin de données relatives aux incidents afin d'entraîner notre modèle de clustering. Ces données sont extraites à partir de deux sources de données :

- un fichier csv qui stocke les localisations des équipements informatiques sujets aux incidents ;
- une base de données Azure SQL Server qui stocke les incidents et leurs attributs. En entreprise, l'outil de gestion de données est le datawarehouse Snowflake. Cependant, dans un souci de sécurité et de confidentialité, il n'est pas possible de l'utiliser dans le cadre pédagogique de ce projet.

L'information de la localisation des équipements informatiques semble importante pour le clustering, c'est pourquoi nous devrons agréger les deux sources de données et ainsi collecter toutes les informations nécessaires sur les incidents.

Depuis le dépôt git, le script d'extraction des données est disponible dans le fichier python : ***data_extraction_cleaning.py*** et la documentation explicite les étapes réalisées : Documentation : extraction des données.

Dans ce script, 3 étapes se succèdent :

- l'extraction des données ;
- le nettoyage et homogénéisation des données ;
- l'agrégation des données en un seul jeu de données et sa sauvegarde.

Les dépendances et connexions externes sont initialisées au début du script, notamment l'accès à la base de données SQL Server contenant les données sur les incidents.

```python
import pandas as pd
import re
import os
from dotenv import load_dotenv
from sqlalchemy import create_engine
import pyodbc

load_dotenv()

# Database configuration
driver = os.getenv("DRIVER")
server = os.getenv("AZURE_SERVER_NAME")
database = os.getenv("AZURE_DATABASE_NAME")
database_raw = os.getenv("AZURE_DATABASE_NAME_RAW")
username = os.getenv("AZURE_DATABASE_USERNAME")
password = os.getenv("AZURE_DATABASE_PASSWORD")

# Connection to Azure databases
azure_connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"
engine = create_engine(azure_connection_string)

azure_connection_string_raw = f"mssql+pyodbc://{username}:{password}@{server}/{database_raw}?driver=ODBC+Driver+17+for+SQL+Server"
engine_raw = create_engine(azure_connection_string_raw)
```

L’extraction des données est réalisée grâce à Pandas qui lit dans la base de données et dans le fichier csv.

```
### EXTRACTION ###

# Read table from the raw incidents database
table_name = 'incidents_raw'
df = pd.read_sql_table(table_name, con=engine_raw)

# Read ci locations from a csv file
ci_name = pd.read_csv('/home/utilisateur/DevIA/prbmg/api_database/data/brutes/CMDB.CSV',delimiter='\t')
```

II. Développer des requêtes de type SQL d’extraction des données

Le script d’extraction des données réalise ensuite des transformations pour filtrer le dataset pour sélectionner les incidents zabbix (remontés automatiquement par le système), sélectionner et nettoyer les colonnes, supprimer les doublons et traiter les valeurs manquantes.

Exemple de fonction de nettoyage :

```
def clean_column_names(df:pd.DataFrame)  -> pd.DataFrame:
    """
    Clean column names in the DataFrame by standardizing their format.

    This function:
    - Strips spaces from the start and end of column names.
    - Replaces spaces with underscores.
    - Removes non-alphanumeric characters except underscores.

    Args:
        df (pd.DataFrame): The DataFrame with columns to be cleaned.

    Returns:
        pd.DataFrame: DataFrame with cleaned column names.
    """
    clean_names = {}
    for col in df.columns:
        # Remplacer les espaces par des _
        new_name = col.strip().lower().replace(' ', '_')
        # Retirer tous les caractères spéciaux
        new_name = re.sub('[^a-zA-Z0-9_]', '', new_name)
        clean_names[col] = new_name

    # Renommer les colonnes avec les nouveaux noms nettoyés
    df = df.rename(columns=clean_names)

    return df
```

III. Développer des règles d'agrégation de données issues de différentes sources

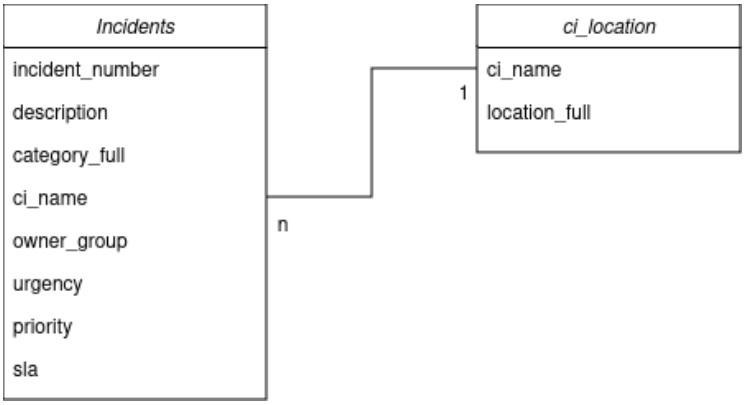
Enfin, après les étapes de nettoyage et d'homogénéisation nécessaires, le script d'extraction enregistre les deux jeux de données dans la base de données du projet (les détails de la création de cette base de données sont présentés dans la partie *D. Créer une base de données*) dans les tables incidents et ci_location. Enfin, il réalise la jointure entre les deux tables pour agréger les données en un seul jeu de données brut final, sauvegardé dans la table incidents de la base de données.

```
### IMPORT AND AGREGATION ###

# Save clean incidents in the new database
df.to_sql('incidents',con=engine, if_exists='append',index=False)

# Save clean ci location on the database
ci_name.to_sql('ci_location',con=engine, if_exists='append',index=False)


conn = pyodbc.connect('DRIVER='+driver+';SERVER=tcp:'+server+';PORT=1433;DATABASE='+database+';UID='+username+';PWD='+ password)


# Update incidents table to add location full, with a join from ci_location on ci_name
with conn.cursor() as cursor:
    add_column_query = """
    IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME = 'incidents' AND COLUMN_NAME = 'location_full')
    BEGIN
        ALTER TABLE incidents ADD location_full VARCHAR(300);
    END
    """
    cursor.execute(add_column_query)
    conn.commit()

with conn.cursor() as cursor:
    update_query = """
    UPDATE incidents
    SET location_full = l.location_full
    FROM ci_location l
    WHERE incidents.ci_name = l.ci_name;
    """
    cursor.execute(update_query)
    conn.commit()
```

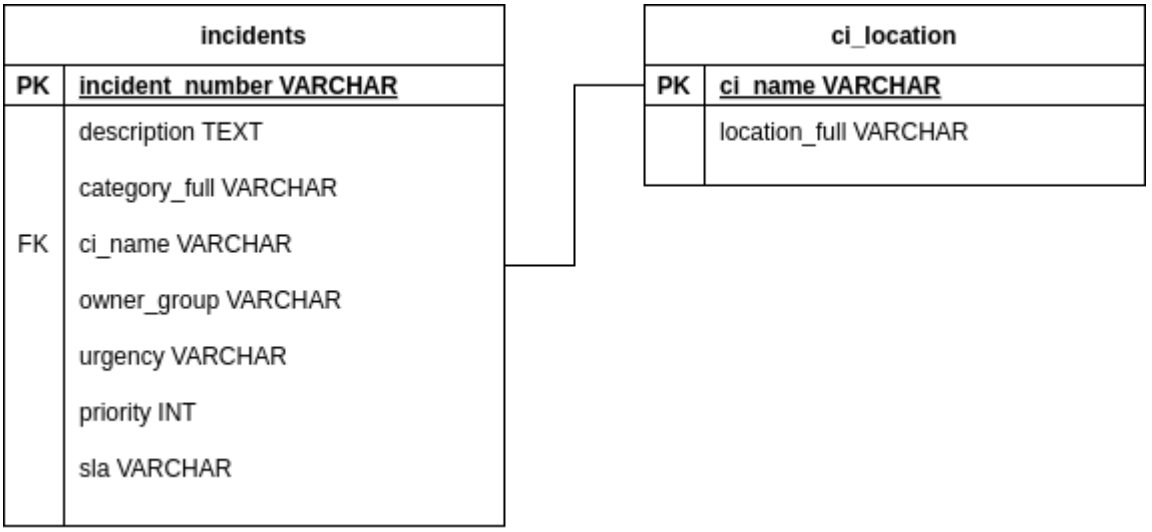
IV. Créer une base de données

La documentation concernant la création de la base de données est versionnée à la racine du projet et est accessible ici : Documentation : création de la base de données

Modèle conceptuel des données



Modèle physique des données

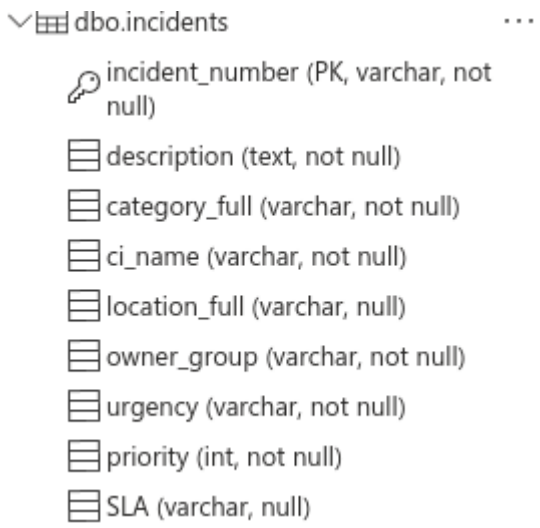


Depuis le dépôt git, le script de création de la base de données et de création des tables sont disponibles dans les fichiers :

- ***creation_database.sh***
- ***creation_table.py***

La base de données créée est une base de données SQL Server car nous avons besoin d'une base de données relationnelle, c'est un choix proposé par notre service cloud Azure, qui permet une bonne sécurisation des accès, avec un temps de requête optimisé, et qui est adaptée pour recevoir le type de données que nous souhaitons stocker.

Sur Azure, nous constatons que la base de données souhaitée a bien été créée ainsi que la table qui contient les données brutes sur les incidents :



Le script de création des tables prévoit également la création de la table prédictions qui intervient plus tard dans le projet, lors du stockage des prédictions du modèle. Au démarrage du projet, elle est vide. Les schémas de base de données initiaux ne font apparaître que les tables qui contiennent les données brutes du début de projet.

Comme évoqué précédemment, le script d'extraction et d'agrégation des données contient également l'insertion des données dans la base de données SQL server.

```
### IMPORT AND AGREGATION ###

# Save clean incidents in the new database
df.to_sql('incidents',con=engine, if_exists='append',index=False)

# Save clean ci location on the database
ci_name.to_sql('ci_location',con=engine, if_exists='append',index=False)


conn = pyodbc.connect('DRIVER='+driver+';SERVER=tcp:'+server+';PORT=1433;DATABASE='+database+';UID='+username+';PWD='+ password)


# Update incidents table to add location full, with a join from ci_location on ci_name
with conn.cursor() as cursor:
    add_column_query = """
    IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME = 'incidents' AND COLUMN_NAME = 'location_full')
    BEGIN
        ALTER TABLE incidents ADD location_full VARCHAR(300);
    END
    """
    cursor.execute(add_column_query)
    conn.commit()

with conn.cursor() as cursor:
    update_query = """
    UPDATE incidents
    SET location_full = l.location_full
    FROM ci_location l
    WHERE incidents.ci_name = l.ci_name;
    """
    cursor.execute(update_query)
    conn.commit()
```

Une requête de la base de données nous permet de constater que les données extraites, nettoyées et agrégées sont bien sauvegardées dans la base de données créée :

```
1   select * from [dbo].[incidents]
```

Résultats Messages

Recherche pour filtrer des éléments...

incident_number	description	category_full	ci_name	location_full
I230331_000233	Trigger: Host has been restarte...	Incidents/Infrastructure/System...	S273A12	INDIA/INDIA/MUMBAI
I230331_000234	Trigger: Interface Ethernet1/16(...	Incidents/Infrastructure/ERP/S...	SWU200H-B	EUROPE/FRANCE/LESTREM/DT...
I230331_000235	Trigger: Interface Ethernet1/20(...	Incidents/Infrastructure/Netwo...	SWI235B	EUROPE/FRANCE/LESTREM/AL...
I230331_000236	Trigger: SECP98A/pacemaker: F...	Incidents/Infrastructure/Netwo...	SECP98A	EUROPE/FRANCE/LESTREM/AL...
I230331_000237	Trigger: Job JOBS.SARE.FEX_00...	Incidents/Infrastructure/ERP/Sc...	S101A779	EUROPE/FRANCE/LESTREM/SA...
I230331_000238	Trigger: Host has been restarte...	Incidents/Infrastructure/System...	S201A174	EUROPE/ITALY/CASSANO SPIN...
I230331_000239	Trigger: High memory utilizatio...	Incidents/Infrastructure/Netwo...	SMTP99A	EUROPE/FRANCE/LA MADELEI...
I230331_000240	Trigger: High memory utilizatio...	Incidents/Infrastructure/Netwo...	SWTP98A	EUROPE/FRANCE/VIC SUR AIS...

Les données stockées dans la base de données ne contiennent pas d'information personnelle. Il n'est donc pas nécessaire de mettre à jour un registre des traitement de données personnelles comme la réglementation RGPD l'impose.

Dans le cadre professionnel, nous traitons les incidents Non-Zabbix dont les descriptions sont rédigées par les utilisateurs dans un champ de texte libre. Il est donc fréquent d'y retrouver tous types d'informations personnelles. De ce fait, nous travaillons en collaboration avec la Data Protection Officer de l'entreprise pour mettre en œuvre les mesures nécessaires au respect de la réglementation RGPD, comme notamment la mise à jour d'un registre des traitements des données personnelles.

V. Développer une API mettant à disposition le jeu de données

Afin de pouvoir communiquer facilement avec la base de données, nous avons créé une API. L'API REST conçue pour mettre à disposition les données du projet est développée avec le framework FastAPI. Elle est protégée par une clé API nécessaire pour toute requête.

La documentation de l'API couvre tous les points de terminaison et les règles d'authentification : Documentation API database.

L'API dispose de plusieurs endpoints utiles au reste du projet.

En premier lieu, elle permet de récupérer l'ensemble des données sur les incidents permettant l'entraînement du modèle de clustering :

```python
database_api_key = os.getenv('API_DATABASE_SECRET_KEY')

def get_incidents():
    """
    Fetches incident data from a remote API and returns it as a DataFrame.

    This function sends a GET request to a specified API endpoint to retrieve incident data. The data is returned
    in JSON format and is then converted into a pandas DataFrame for further processing.

    Returns:
        pd.DataFrame: A DataFrame containing the incident data fetched from the API.

    Raises:
        HTTPError: If the HTTP request returned an unsuccessful status code.
    """
    url = "http://prbmg-api-database.francecentral.azurecontainer.io:8000/incidents/"
    headers = {"X-API-Key": database_api_key}
    response = requests.get(url, headers=headers)
    response.raise_for_status()  # Assurez-vous que la requête est réussie
    incidents = response.json()
    incidents = pd.DataFrame(incidents)

    return incidents
```

Elle permet aussi d'accéder à la table ci_location afin de récupérer la localisation des équipements informatiques concernés par les nouveaux incidents qui doivent être clusterisés.

```python
@app.get("/ci_location", response_model=List[CILocation])
def get_ci_location(request: Request,  db: Session = Depends(get_db_azure)) -> List[CILocation]:
    """Retrieve all incident locations.

    Args:
        request (Request): The incoming request.
        db (Session, optional): SQLAlchemy session to interact with the database. Defaults to Depends(get_db_azure).

    Returns:
        List[Incident]: List of retrieved incident locations.

    Raises:
        HTTPException: If no incident locations are found.
    """
    try:
        db_ci_location = read_db_ci_location(db)
    except NotFoundError as e:
        raise HTTPException(status_code=404) from e
    return [CILocation(**ci.__dict__) for ci in db_ci_location]
```

Enfin, un endpoint permet de récupérer toutes les prédictions stockées dans la base de données :

```python
@app.get("/predictions", response_model=List[Prediction])
def get_predictions(request: Request,  db: Session = Depends(get_db_azure)) -> List[Prediction]:
    """Retrieve all predictions.

    Args:
        request (Request): The incoming request.
        db (Session, optional): SQLAlchemy session to interact with the database. Defaults to Depends(get_db_azure).

    Returns:
        List[Prediction]: List of retrieved incidents.

    Raises:
        HTTPException: If no predictions are found.
    """
    try:
        db_predictions = read_db_predictions(db)
    except NotFoundError as e:
        raise HTTPException(status_code=404) from e
    return [Prediction(**prediction.__dict__) for prediction in db_predictions]
```

Même s’ils ne sont pas directement utiles au reste du projet, d’autres endpoints typiques d’une API de données ont été créés. Ce sont ceux qui permettent les opérations du CRUD (create - read - update - delete) :

- créer un incident
- mettre à jour un incident
- supprimer un incident
- lire un incident d’après son identifiant.

Annexes

Annexe 1 : Rétro-planning