SIMPLON
OUTRE-MER

Société Publique Locale
afpar

BRIEF 1C 12/2022

ANALYSIS

BUSINESS INTELLIGENCE

DOCUMENTATION TECHNIQUE

Ugo FOK-YIN (03)

Lien GitHub du projet: [UgoDIA/1C: Brief 1C (github.com)]

Sommaire

1. Contexte du projet

Cette ESN spécialisée dans la réalisation d'applications de type BI et intelligence artificielle travaille sur un futur outil d'aide à la décision e-Commerce pour un de ses clients.

Vous avez déjà accès à un fichier de données brutes, matérialisant un export depuis les bases de données opérationnelles de ce client

Avec le PoC du futur dashboard que vous réaliserez, le client doit pouvoir lancer un ETL dédié pour collecter des données de qualité. Celles-ci pourront être illustrées sous forme de graphiques.

2. Ressources et outils utilisés

Framework Django:

Django est un framework de développement d'application web open source qui s'inspire du principe MVT.

Documentation de Django | Documentation de Django

PostgreSQL:

PostgreSQL est un SGBD orienté objet puissant et open/ source qui est capable de prendre en charge en toute sécurité les charges de travail de données les plus complexes.

Documentation PostgreSQL 15.0

HTML/CSS et Bootstrap:

Bootstrap.css est un framework CSS qui organise et gère la mise en page d'un site web. Alors que le HTML gère le contenu et la structure d'une page web, le CSS s'occupe de la mise en page du site. Pour cette raison, les deux structures doivent coexister pour effectuer une action particulière.

Get started with Bootstrap · Bootstrap v5.2

JavaScript:

JavaScript (souvent abrégé en « JS ») est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web. Il permet d'interagir avec la page.

jQuery:

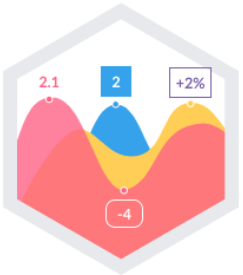
jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts coté client dans le code HTML des pages web

Chart.js:

Chart.js est une bibliothèque JavaScript open source gratuite pour la visualisation de données.

Plugin datalabels:

Permet d'afficher des étiquettes sur les graphiques



GitHub:

La plateforme GitHub permet aux programmeurs informatiques de collaborer librement sur des projets de code et nous a permis de planifier le projet avec un scrumboard.

Pandas:

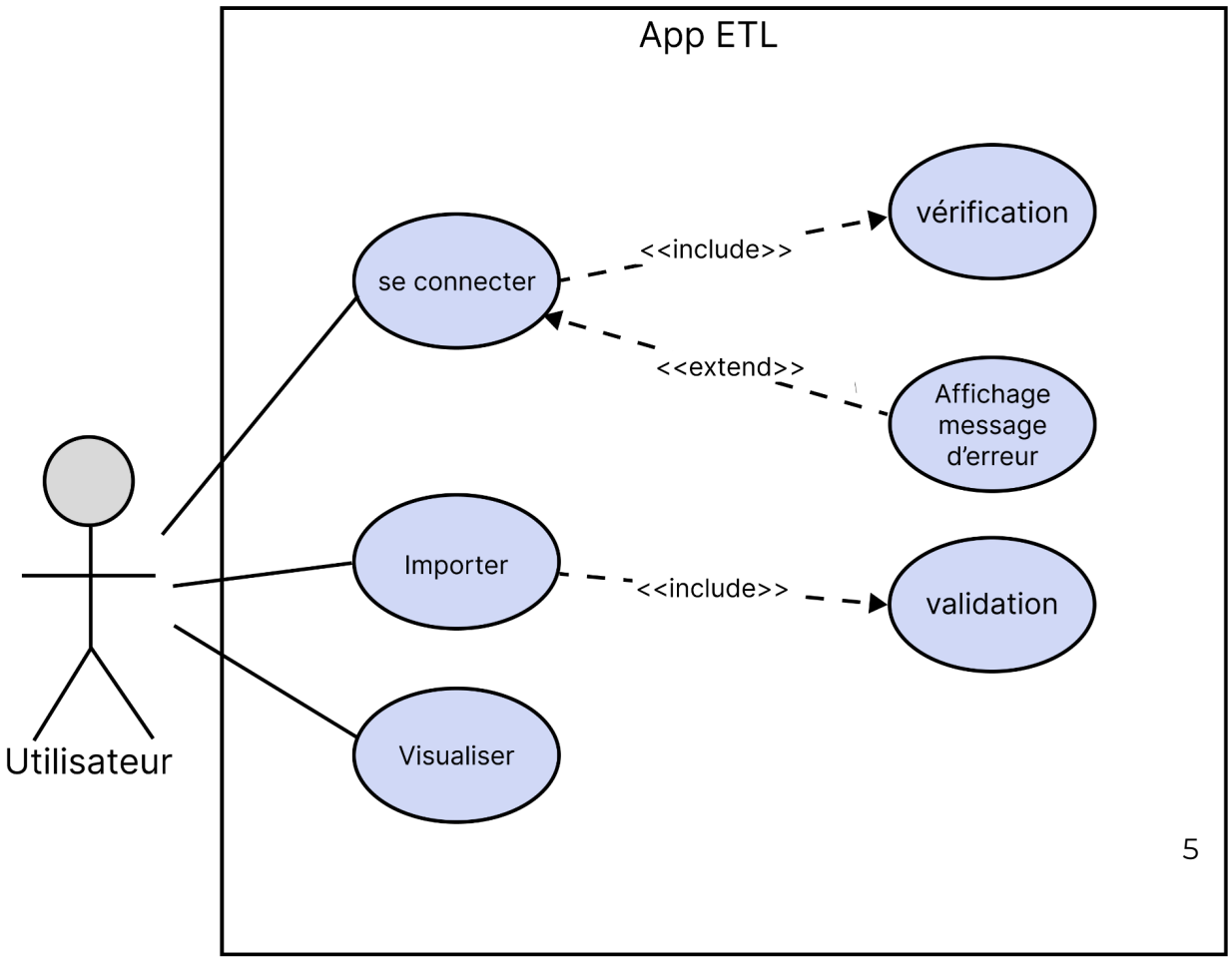
La bibliothèque logicielle open-source Pandas est spécifiquement conçue pour la manipulation et l'analyse de données en langage Python. Elle est à la fois performante, flexible et simple d'utilisation.

3. Maquette, gdD, UC et architecture

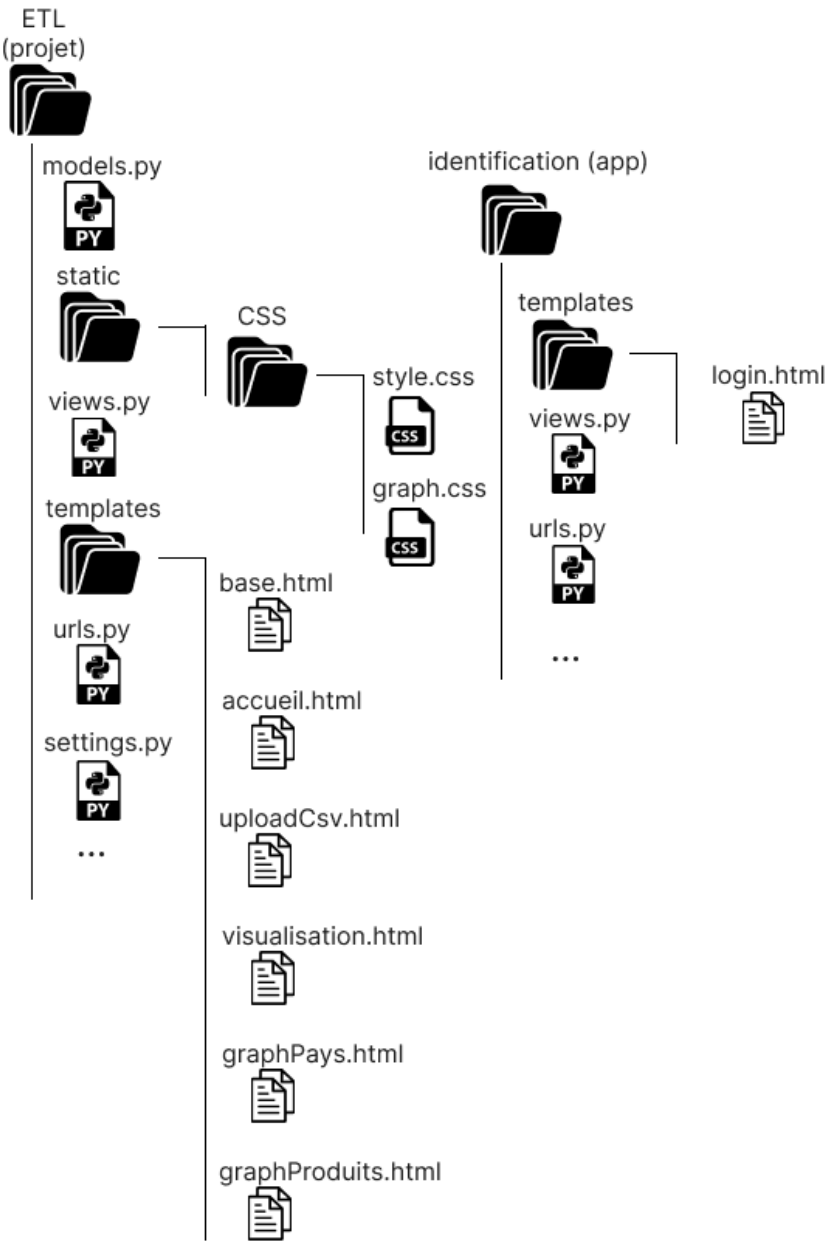
Maquette/graphe de dialogue:

Figma

Diagramme cas d'utilisation:



Arborescence Django:

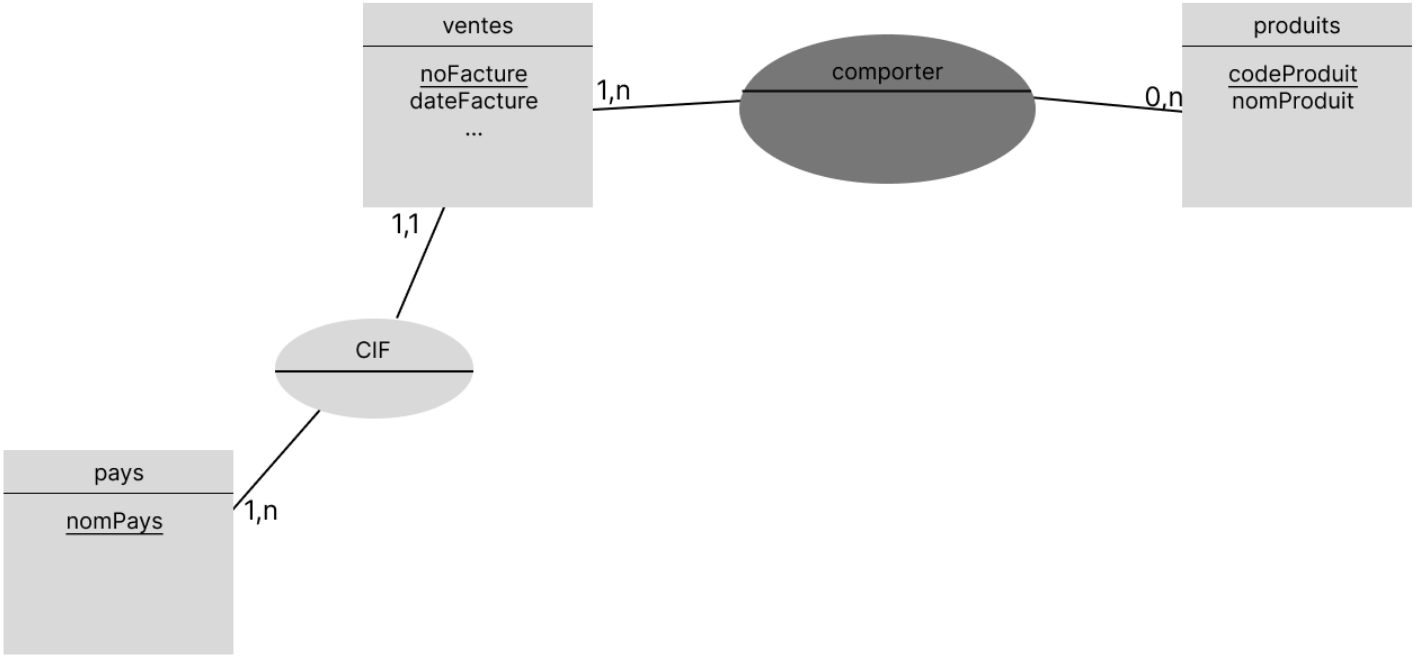


4. Base de données

Dictionnaire de données:

Attributs	Types de données	Valeur Null	Définitinon	Entité
codeProduit	varchar(6)		numéro identifiant un produit	produits
noFacture	varchar(6)		numéro identifiant une facture	ventes
dateFacture	date		date de la facture	ventes
pays	varchar(20)		pays d'origine du client	pays
nomProduit	varchar(50)		nom du produit	produits

MCD:



MPD dégradé:

5.Détails des fonctionnalités

Traitement du CSV avec pandas:

On renomme les colonnes que l'on va utiliser

```
df.rename(columns={'InvoiceNo':'noFacture','StockCode':'codeProduit','D
escription':'nomProduit','InvoiceDate':'dateFacture','Country':'pays'},
inplace=True)
```

On supprime les colonnes qui sont inutiles

```
df.drop(['CustomerID','UnitPrice'], inplace=True, axis=1)
```

On convertit tous les caractères minuscules en majuscules dans la colonne codeProduit

```
df['codeProduit']=df['codeProduit'].str.upper()
```

Exemple dans le CSV :

```
InvoiceNo,StockCode,Description,Country
536365,85123A,WHITE HANGING HEART T-LIGHT HOLDER,United Kingdom
536982,85123a,WHITE HANGING HEART T-LIGHT HOLDER,United Kingdom
539651,M,Manual,United Kingdom
539736,m,Manual,United Kingdom
```

On supprime les lignes où les noFacture et codeProduit sont dupliquées

```
df=df.drop_duplicates(['noFacture','codeProduit'])
```

On vérifie si tout les noFacture n'ont pas de lettre à l'intérieur et si il y en a, ils seront changé en NaN puis on les supprime par la suite

```
df['noFacture']=pd.to_numeric(df['noFacture'],errors='coerce')
df.dropna(subset=['noFacture'], inplace = True)
```

On vérifie si toutes les dates respecte le format m/d/y h:m et si il y a erreur, il seront changé en NaT puis supprimées

```
df['dateFacture']=pd.to_datetime(df['dateFacture'], format='%m/%d/%Y
%H:%M',errors='coerce')
df.dropna(subset=['dateFacture'], inplace = True)
```

On convertit les codeProduit plus grand que 8 en NaN puis on les supprime

```
df.loc[df['codeProduit'].str.len()>8 ]=np.nan
df.dropna(inplace = True)
```

exemple:

```
558066,gift_0001_50,Dotcomgiftshop Gift Voucher 50.00,1,6/24/2011
15:45,41.67,,United Kingdom
```

On garde que les quantités négatives puis on supprime la colonne quantité

```
df=df[(df['Quantity']>0)]
df.drop(['Quantity'], inplace=True, axis=1)
```

On supprime les lignes avec comme pays “unspecified” et les produits avec comme nom “mailout”

```
df.drop(df[df['pays']== 'Unspecified'].index,inplace=True)
df.drop(df[df['nomProduit']== 'mailout'].index,inplace=True)
```

On fait une copie du dataframe df et on supprime les lignes dupliquées sur la colonne pays pour avoir une liste des pays unique puis un supprime les colonnes inutiles

```
listePays=df.drop_duplicates('pays').copy()
listePays.drop(['noFacture','dateFacture','codeProduit','nomProduit',
'dateFacture'], inplace=True, axis=1)
```

On fait une copie du dataframe df et on supprime les lignes dupliquées sur la colonne noFacture pour avoir une liste des facture unique puis on supprime les colonnes inutiles

```
listeFacture=df.drop_duplicates(subset=['noFacture']).copy()
listeFacture.drop(['codeProduit','nomProduit'], inplace=True, axis=1)
```

On fait une copie du dataframe pour avoir une liste des produits puis on enlève les espaces à la fin des noms de produits

```
listeProduits=df.drop_duplicates(subset=['codeProduit']).copy()
listeProduits['nomProduit']=listeProduits['nomProduit'].str.rstrip()
```

exemple:

```
539631,22862,LOVE HEART NAPKIN BOX ,United Kingdom
539634,22198,LARGE POPCORN HOLDER ,United Kingdom
```

On revient sur le dataframe df et on supprime les colonnes dont on a plus besoin afin d’avoir un dataframe qui nous convient

```
df.drop(['dateFacture','pays','nomProduit'], inplace=True, axis=1)
```

Insertion dans la base de données après validation:

Après avoir affiché le feedback, on a la possibilité d'enregistrer les données à partir d'un bouton formulaire qui emmène vers l'url save qui va lancer la fonction save dans la views et dans cette fonction on va utiliser les différents dataframe qu'on a obtenu pour les sauvegarder dans la base de données.

```
<form action="{% url 'save' %}" method="POST" enctype="multipart/form-data">
{% csrf_token %}
<button type ="submit" id='btUp2' style= " font-size: 17px" class="btn
btn-warning btn-block btn-outlined"> Enregistrer dans la base de
données</button>
</form>
```

La table pays:

pour la table pays on fait une simple boucle sur la liste obtenu

```
P=listePays['pays'].to_list()
for i in range(len(P)):
     newPays=Pays(pays=P[i])
     newPays.save()
```

La table produits:

pour la table produits on utilise la méthode iterrows qui permet d'itérer sur chaque lignes d'un dataframe

```
for index, row in listeProduits.iterrows():
      newProduits=Produits(codeproduit=row['codeProduit'],nomproduit=row[
      'nomProduit'])
      newProduits.save()
```

Avec la méthode to_sql:

Pour cette méthode on a besoin d'installer le module sqlalchemy et importer create_engine afin de pouvoir créer une connexion avec la base de données qui est nécessaire pour la méthode to_sql

```
from sqlalchemy import create_engine

conn= 'postgresql://postgres:0000@localhost:5432/ETLDB'
engine=create_engine(conn)
```

La table ventes:

La méthode prends comme paramètre le nom de la table, la connexion à une db, et le comportement dans le cas où la table existerait déjà puis si il compte l'index du dataframe comme colonne à insérer.
Le dataframe à insérer doit avoir le même nombre de colonnes ainsi que les mêmes noms des colonnes.

```
listeFacture.to_sql('ventes',engine, if_exists='append',index= False)
```

La table detailsVentes:

```
df.to_sql('detailsVentes',engine, if_exists='append',index= False)
```

Visualisation avec chart.js:

Répartition des ventes par Pays:

La table filtre:

nFiltre [PK] integer	filtrePays character varying (10)	filtreDate character varying (10)	filtreProduits character varying (50)	filtreTopPays character varying (30)
1	top	2010	85123A	United Kingdom

```
filtre=Filtre.objects.get(nfiltre=1)
filtrep=filtre.filtrepays
filtred=filtre.filtredate
...
```

Exemple de requête SQL en fonction des filtres pour le graphique:

top 10 des pays sur toutes les années:

```
cursor.execute('''SELECT pays, COUNT(pays) FROM ventes INNER JOIN
"detailsVentes" on ventes."noFacture" = "detailsVentes"."noFacture" GROUP BY
pays ORDER BY 2 DESC LIMIT 10''')
```

```
SELECT pays, COUNT(pays)
    FROM ventes INNER JOIN "detailsVentes"
    on ventes."noFacture" = "detailsVentes"."noFacture"
    GROUP BY pays
    ORDER BY 2 DESC
    LIMIT 10
```

Top 10 des pays pour une année choisie

```
cursor.execute('''SELECT pays, COUNT(pays) FROM ventes INNER JOIN
"detailsVentes" on ventes."noFacture" = "detailsVentes"."noFacture" WHERE
EXTRACT(YEAR FROM ventes."dateFacture")=%(filtred)s GROUP BY pays ORDER BY 2
DESC LIMIT 10''',{"filtred":filtred})
```

```
SELECT pays, COUNT(pays)
    FROM ventes INNER JOIN "detailsVentes"
    on ventes."noFacture" = "detailsVentes"."noFacture"
    WHERE EXTRACT(YEAR FROM ventes."dateFacture")= %(filtred)s
    GROUP BY pays
    ORDER BY 2 DESC
    LIMIT 10
```

Répartition des ventes par Pays et par produits:

SQL pour top 5 des produits pour un pays et une date sélectionnés

```
cursor.execute('''SELECT "nomProduit",count(*) FROM produits INNER JOIN
"detailsVentes" ON produits."codeProduit" = "detailsVentes"."codeProduit"
INNER JOIN ventes ON "detailsVentes"."noFacture" = ventes."noFacture" Where
pays = %(filtretoppays)s AND EXTRACT(YEAR FROM "dateFacture")=%(filtred)s
GROUP BY "nomProduit" ORDER BY 2 DESC LIMIT 5
''',{"filtretoppays":filtretoppays,"filtred":filtred})
```

```
SELECT "nomProduit",count(*)
    FROM produits INNER JOIN "detailsVentes"
    ON produits."codeProduit" = "detailsVentes"."codeProduit"
    INNER JOIN ventes ON "detailsVentes"."noFacture" = ventes."noFacture"
    Where pays = %(filtretoppays)s AND EXTRACT(YEAR FROM "dateFacture")=%(filtred)s
    GROUP BY "nomProduit"
    ORDER BY 2 DESC
    LIMIT 5
```

Clic sur les barres: Doc

Lors d'un clic sur une barre de l'histogramme, on est redirigé vers l'url filtrePays qui va nous permettre de récupérer une variable via l'url pour pouvoir l'utiliser après dans la view.

```
 function clickHandler(click){
      const points = myChart.getElementsAtEventForMode(click, 'nearest' ,
{intersect:true}, true);
      if (points.length) {
        const firstPoint = points[0];
        value= myChart.data.datasets[firstPoint.datasetIndex].label[0]
        window.location.replace("filtrePays/"+value);
      }
    }
    ctx.onclick = clickHandler;
```

urls.py:
L'url filtrePays va donc appeler la fonction filtrePays dans la view

```
path('ETL/visualisation/filtrePays/<str:pays>',views.filtrePays,
name='filtrePays'),
```

views.py
On récupère la variable pays depuis l'url puis on l'utilise pour mettre à jour la table filtre puis on recharge la page

```
def filtrePays(request,pays):
    Filtre.objects.filter(nfiltre=1).update(filtretoppays=pays)
    return HttpResponseRedirect(reverse('graphPays'))
```

Tout le processus de visualisation pour les ventes par pays est quasiment le même pour les ventes par produits.

6. To do list et conclusion

-Plus d'interactivité au niveau de l'import de plusieurs fichiers
-Plus d'options de visualisations
-Corriger les bugs et nettoyer le code