**Vue d'ensemble**

Le code Python crée une **application web interactive avec Streamlit** conçue pour la gestion des informations des défunts. Elle permet aux utilisateurs d'effectuer deux actions principales : **ajouter de nouveaux enregistrements de défunts** et **rechercher des enregistrements existants**. L'application intègre également une **carte Folium** pour visualiser un plan de cimetière.

**Bibliothèques utilisées**

* pandas (sous l'alias pd) : Utilisée pour la manipulation des données, notamment pour créer et travailler avec des DataFrames (structures de données tabulaires).
* streamlit (sous l'alias st) : La bibliothèque principale pour construire des applications web interactives en Python.
* folium (sous l'alias flm) : Utilisée pour créer des cartes interactives.
* geopandas (sous l'alias gpd) : (Importée mais non explicitement utilisée dans l'extrait de code fourni pour l'affichage de la carte ; souvent utilisée pour la manipulation de données géospatiales).
* json : Utilisée pour travailler avec des données JSON, spécifiquement pour charger le fichier GeoJSON de la carte.
* streamlit\_folium : Un composant Streamlit qui permet d'intégrer des cartes Folium directement dans une application Streamlit.

**Structure et logique de l'application**

**1. Titre de la page et bienvenue**

Python

st.title("Page de gestion des défunts")

st.title("Bienvenue")

Ces lignes définissent les titres principaux de l'application Streamlit, indiquant clairement à l'utilisateur l'objet de la page.

**2. Initialisation de la base de données (st.session\_state.BD)**

Python

if 'BD' not in st.session\_state:

st.session\_state.BD = {

'Nom': [], 'Prenom': [], 'Date\_de\_naissance': [],

# ... (autres clés) ...

'Numero\_de\_la\_tombe': [],

}

C'est une partie cruciale de la gestion des données dans une application Streamlit. st.session\_state est un objet similaire à un dictionnaire qui **persiste les données** entre les exécutions du script (par exemple, lorsqu'un utilisateur interagit avec un widget).

* Il vérifie si une clé nommée 'BD' (pour "Base de Données") existe dans l'état de session actuel.
* Si ce n'est pas le cas, il initialise 'BD' comme un dictionnaire Python.
* Ce dictionnaire stockera toutes les informations sur les défunts. Chaque clé (par exemple, 'Nom', 'Prenom', 'Date\_de\_naissance') est associée à une liste vide. Lorsqu'un nouvel enregistrement est ajouté, les données pour chaque champ sont ajoutées à leur liste respective. Cette structure est adaptée pour une conversion ultérieure en DataFrame Pandas.

**3. Gestion des choix de l'utilisateur (st.session\_state.choix\_utilisateur)**

Python

if 'choix\_utilisateur' not in st.session\_state:

st.session\_state.choix\_utilisateur = []

choix = st.multiselect("Que voulez-vous faire ?", ['Ajouter defunt', 'Rechercher defunt'])

if st.button("Valider"):

st.session\_state.choix\_utilisateur = choix

Cette section gère les actions que l'utilisateur souhaite effectuer :

* Elle initialise st.session\_state.choix\_utilisateur comme une liste vide si elle n'existe pas. Cette liste stockera les options sélectionnées par l'utilisateur.
* st.multiselect affiche une liste de sélection multiple où l'utilisateur peut choisir entre "Ajouter defunt" et "Rechercher defunt".
* Lorsque le bouton "Valider" est cliqué, les options sélectionnées dans le multiselect sont sauvegardées dans st.session\_state.choix\_utilisateur. Cela garantit que l'application "se souvient" des choix de l'utilisateur même après une nouvelle exécution.

**4. Ajout d'un enregistrement de défunt**

Python

if 'Ajouter defunt' in st.session\_state.choix\_utilisateur:

st.header("📝 Ajout d'un défunt")

st.subheader("Informations du défunt")

# ... (champs de saisie pour les informations du défunt) ...

st.subheader("Personne à contacter")

# ... (champs de saisie pour la personne à contacter) ...

st.subheader("Inhumation")

# ... (champs de saisie pour les informations d'inhumation) ...

if st.button("Enregistrer"):

BD = st.session\_state.BD

# ... (ajout des données aux listes de BD) ...

st.success("✅ Défunt enregistré avec succès.")

st.dataframe(pd.DataFrame(BD))

Ce bloc de code est exécuté uniquement si "Ajouter defunt" a été sélectionné par l'utilisateur et validé :

* **En-têtes et sous-en-têtes :** st.header et st.subheader sont utilisés pour structurer clairement le formulaire de saisie.
* **Champs de saisie :** Divers widgets d'entrée Streamlit sont utilisés pour collecter les données :
  + st.text\_input : Pour les informations textuelles (Nom, Lieu de naissance, etc.).
  + st.date\_input : Pour la sélection de dates (Date de naissance, Date de décès).
  + st.time\_input : Pour la sélection de l'heure du décès.
  + st.number\_input : Pour les entrées numériques (numéros de CNI, numéros de téléphone, détails de la tombe).
  + **Argument key :** Chaque widget d'entrée possède une key unique. C'est important dans Streamlit pour garantir que les widgets sont correctement gérés, en particulier lorsque leur état peut changer entre les exécutions.
* **Bouton "Enregistrer" :** Lorsque ce bouton est cliqué :
  + Le dictionnaire st.session\_state.BD est récupéré.
  + Les valeurs saisies dans les champs d'entrée sont **ajoutées** à leurs listes respectives dans le dictionnaire BD.
  + Un message de succès (st.success) est affiché.
  + Enfin, pd.DataFrame(BD) convertit le dictionnaire de listes en un DataFrame Pandas, et st.dataframe affiche ce DataFrame, montrant tous les défunts actuellement enregistrés.

**5. Recherche d'un enregistrement de défunt**

Python

if 'choix\_utilisateur' not in st.session\_state:

st.session\_state.choix\_utilisateur = ['Rechercher defunt'] # Choix par défaut si aucun n'est sélectionné

if 'Rechercher defunt' in st.session\_state.choix\_utilisateur:

st.header("🔍 Rechercher un défunt")

with st.form(key='form\_recherche\_defunt'):

st.subheader("Critères de recherche")

nom\_recherche = st.text\_input("Nom du défunt", key="search\_nom\_input")

prenom\_recherche = st.text\_input("Prénom du défunt", key="search\_prenom\_input")

permis\_recherche = st.text\_input("Numéro du permis d'inhumer", key="search\_permis\_input")

submitted = st.form\_submit\_button("Rechercher le défunt")

if submitted:

df = pd.DataFrame(st.session\_state.BD)

if df.empty:

st.warning("⚠️ La base de données est vide. Aucun défunt à rechercher.")

else:

# Conversion de type pour une comparaison précise

df['Nom'] = df['Nom'].astype(str)

df['Prenom'] = df['Prenom'].astype(str)

df['Delivrance\_du\_Permis\_d\_inhumer'] = df['Delivrance\_du\_Permis\_d\_inhumer'].astype(str)

# Filtrage du DataFrame

resultats = df[

(df['Nom'].str.strip().str.lower() == nom\_recherche.strip().lower()) &

(df['Prenom'].str.strip().str.lower() == prenom\_recherche.strip().lower()) &

(df['Delivrance\_du\_Permis\_d\_inhumer'].str.strip() == permis\_recherche.strip())

]

if not resultats.empty:

st.success(f"✅ {len(resultats)} résultat(s) trouvé(s) :")

st.dataframe(resultats)

else:

st.error("❌ Aucun défunt trouvé avec ces critères. Veuillez vérifier les informations saisies.")

Cette section gère la fonctionnalité de recherche :

* **Choix par défaut :** La ligne st.session\_state.choix\_utilisateur = ['Rechercher defunt'] semble être un comportement par défaut, ou un moyen de s'assurer que l'option de recherche est active si aucun autre choix n'a été explicitement fait. Si un utilisateur clique initialement sur "Valider" sans rien sélectionner, ou si l'état de la session n'est pas parfaitement géré, cela garantit que "Rechercher defunt" est actif.
* **Formulaire de recherche (with st.form) :**
  + st.form de Streamlit crée un formulaire isolé. Tous les widgets à l'intérieur du bloc with st.form(...) sont traités comme faisant partie de ce formulaire. Lorsque le st.form\_submit\_button à l'intérieur du formulaire est cliqué, seul ce formulaire est soumis, déclenchant une ré-exécution de l'application spécifiquement liée à la soumission de ce formulaire. C'est utile pour contrôler les interactions et éviter les ré-exécutions complètes inutiles de l'application.
  + Trois champs st.text\_input sont fournis pour les critères de recherche : Nom, Prénom et Numéro de permis.
  + Le st.form\_submit\_button lance la recherche.
* **Logique de recherche (if submitted:) :**
  + La st.session\_state.BD (notre base de données en mémoire) est convertie en un DataFrame Pandas df. Cela facilite et rend plus efficace la recherche.
  + **Vérification de la base de données vide :** Le code vérifie d'abord si le DataFrame est vide et affiche un avertissement si c'est le cas.
  + **Conversion de type :** Les colonnes Nom, Prenom et Delivrance\_du\_Permis\_d\_inhumer sont explicitement converties en type chaîne de caractères (astype(str)). C'est crucial car les champs de saisie peuvent fournir des nombres, et des comparaisons de chaînes sont nécessaires pour strip() et lower().
  + **Filtrage :** Le cœur de la recherche est effectué à l'aide de l'indexation booléenne sur le DataFrame :
    - df['Nom'].str.strip().str.lower() == nom\_recherche.strip().lower() : Cela compare la colonne Nom (après avoir supprimé les espaces de début/fin et converti en minuscules) avec l'entrée nom\_recherche (également traitée de manière similaire). Cela rend la recherche **insensible à la casse** et **tolérante aux espaces**.
    - La même logique s'applique à Prenom et Delivrance\_du\_Permis\_d\_inhumer.
    - L'opérateur & combine ces conditions, ce qui signifie que *toutes* les conditions doivent être vraies pour qu'une ligne soit considérée comme une correspondance.
  + **Affichage des résultats :**
    - Si resultats (le DataFrame filtré) n'est pas vide, un message de succès et les enregistrements correspondants sont affichés à l'aide de st.dataframe.
    - Si aucun résultat n'est trouvé, un message d'erreur (st.error) est affiché.

**6. Visualisation de la carte du cimetière**

Python

## Localisation sur le plan du cimetière

st.subheader("Localisation sur le plan du cimetière")

geojson\_path = "C:\\Users\\dell\\Desktop\\plateforme\\tombes.geojson"

try:

with open(geojson\_path, 'r', encoding='utf-8') as f:

geojson\_data = json.load(f)

carte = flm.Map(location=[14.755922, -17.399996], zoom\_start=15, tiles="OpenStreetMap")

flm.GeoJson(geojson\_data, name="cimetiere\_tombes").add\_to(carte)

flm.LayerControl().add\_to(carte)

st\_folium(carte, width=700, height=500)

except FileNotFoundError:

st.error(f"❌ Erreur : Le fichier GeoJSON '{geojson\_path}' est introuvable. Vérifiez le chemin d'accès.")

except json.JSONDecodeError:

st.error(f"❌ Erreur : Le fichier '{geojson\_path}' n'est pas un fichier JSON valide.")

except Exception as e:

st.error(f"❌ Une erreur inattendue est survenue lors du chargement ou de l'affichage de la carte : {e}")

Cette partie affiche une carte interactive du cimetière, que la recherche ait été effectuée ou non :

* **En-tête :** st.subheader introduit cette section.
* **Chemin GeoJSON :** geojson\_path spécifie le chemin absolu vers un fichier GeoJSON nommé tombes.geojson. Ce fichier est censé contenir des données géographiques (par exemple, des polygones représentant des parcelles de tombes ou des sections du cimetière).
  + **Remarque importante :** Le chemin C:\\Users\\dell\\Desktop\\plateforme\\tombes.geojson est un chemin absolu Windows. Si cette application est déployée sur un système d'exploitation différent (comme Linux pour un déploiement cloud) ou déplacée, ce chemin devra être ajusté à un chemin relatif ou à un chemin absolu approprié pour cet environnement.
* **Gestion des erreurs (try...except) :**
  + Le code utilise un bloc try...except pour gérer les problèmes potentiels de manière élégante :
    - FileNotFoundError : Attrape si le fichier tombes.geojson n'existe pas au chemin spécifié.
    - json.JSONDecodeError : Attrape si le fichier existe mais n'est pas un JSON valide.
    - Exception as e : Attrape toute autre erreur inattendue pendant le processus.
    - Des messages d'erreur appropriés sont affichés à l'utilisateur à l'aide de st.error.
* **Chargement du GeoJSON :** json.load(f) lit le contenu du fichier GeoJSON.
* **Création de la carte Folium :**
  + flm.Map(location=[14.755922, -17.399996], zoom\_start=15, tiles="OpenStreetMap") crée une carte Folium.
    - location=[14.755922, -17.399996] définit le centre de la carte. Ces coordonnées correspondent approximativement au **cimetière de Yoff à Dakar, Sénégal**.
    - zoom\_start=15 définit le niveau de zoom initial de la carte.
    - tiles="OpenStreetMap" utilise la couche de fond OpenStreetMap standard.
  + flm.GeoJson(geojson\_data, name="cimetiere\_tombes").add\_to(carte) ajoute les données géographiques du fichier GeoJSON à la carte.
  + flm.LayerControl().add\_to(carte) ajoute un contrôle de couche à la carte, permettant aux utilisateurs d'activer/désactiver différentes couches (utile s'il y en avait plusieurs).
* **Affichage de la carte dans Streamlit :** st\_folium(carte, width=700, height=500) rend la carte Folium interactive dans l'application Streamlit avec une largeur et une hauteur spécifiées.

En résumé, cette application Streamlit est un outil simple mais efficace pour la gestion de base des registres de défunts, combinant la saisie de données, la recherche et une visualisation géographique.

Avez-vous d'autres questions sur des parties spécifiques du code ou souhaitez-vous explorer comment l'améliorer ?