

Vous : bjr

Salut 

Ça va ? Tu veux qu'on parle de quoi aujourd'hui ?

admin.py

Python

models.py

Python

urls.py

Python

views.py

Python

contenu du dossier app\_acceuil

Réflexion durant 11s

Voici le **contenu logique** de ton dossier **app\_acceuil** (d'après les fichiers que tu as fournis) :

## 1) models.py — Les tables / données gérées

Tu as 6 modèles principaux pour alimenter la page d'accueil + contact :

models

- **Projetscards** : témoignages (nom, image, description, est\_visible, date).
- **Projetsfirstspeciality** : “spécialité” (image + description, est\_visible).
- **Projetsrealisations** : réalisations/projets (nom, image, description, est\_visible) + **compteur** compteur\_demo\_live pour compter les clics sur “Démo Live”.
- **Projetphotodeprofil** : photo de profil (image, est\_visible).
- **ProjetAproposDeMoi** : texte “à propos” (description, est\_visible).
- **MesCompetencesCles** : compétences clés (nom unique, est\_visible).
- **ReseauSocial** : réseaux sociaux (email/whatsapp/linkedin/etc.) avec **validation conditionnelle** :
  - si nom=email → validation email
  - sinon → validation URL

Il y a aussi une fonction utilitaire `nettoyer_espaces()` utilisée pour nettoyer certains champs texte.

models

---

## 2) admin.py — Configuration Django Admin

Tu déclares des ModelAdmin pour afficher/modifier rapidement les objets, avec list\_display, list\_editable, etc.

admin

Points notables :

- Les modèles **singleton** (où tu veux **une seule ligne**) bloquent l'ajout via has\_add\_permission() :
  - Projetsfirstspeciality
  - Projetphotodeprofil
  - ProjetAproposDeMoi

admin

- MesCompetencesCles et ReseauSocial sont enregistrés via @admin.register(...).

admin

---

## 3) urls.py — Les routes (URLs)

Tu exposes :

urls

### Page publique

- / et /index.html → aff\_acceuil\_PAGE\_PUBLIQUE\_Vue\_par\_les\_VISITEURS

### Dashboard (admin personnalisé)

- /dashboard → page\_Mon\_Espace\_Administration

## Compteur “Démo Live”

- /realisations/<id>/demo-live → incrementer\_compteur\_demo\_live

## Autres pages / templates

- /Page\_CAMIONS\_DE\_TRANSPORT\_DE\_MARCHANDISES
- /Page\_diapo\_Javascript\_Page1
- /Page\_email\_Javascript

## Redirections vers ancrés de la page d'accueil

- /Page\_accueil\_section\_projets → /#projets, etc.

## Reset compteur projet (un projet)

- /reinitialiser-compteur/<id>/ → reinitialiser\_compteur\_projet  
urls

## 4) views.py — La logique (public + dashboard)

### A) Vue publique (portfolio)

aff\_acceuil\_PAGE\_PUBLIQUE\_Vue\_par\_les\_VISITEURS () :

- récupère uniquement les objets où est\_visible=True
- construit un dictionnaire contenu (témoignages, spécialités, réalisations, photo profil, à propos, compétences, réseaux)
- gère aussi des menus visibles selon le rôle (via session + modèle RoleMenuAcces)

views

- rend le template `index.html`

views

## B) Dashboard d'administration

`page_Mon_Espace_Administration()` :

- vérifie la session : accès autorisé seulement si rôle **Administrateur** ou **Modérateur**, sinon redirection accueil + message

views

- gère énormément d'actions en `POST` :

- photo profil (upload, suppression, visibilité)
- à propos (texte, suppression, visibilité)
- spécialité (update + visibilité globale)
- réalisations (ajout/modif/suppression + toggle visibilité + visibilité globale)
- témoignages (ajout/suppression + toggle visibilité + visibilité globale)
- compétences (ajout/suppression + toggle visibilité + visibilité globale)
- réseaux (update\_or\_create, suppression + visibilité globale)

views

- rend le template `page_administration.html` avec statistiques (`TOTAL_CLICS_DEMO_LIVE`, tri par compteur, etc.).

views

## C) Compteur “Démo Live”

`incrementer_compteur_demo_live()` :

- incrémente `compteur_demo_live` de façon atomique (F expression)
- redirige ensuite vers la page “camions” (démono).

views

## D) Pages / redirections

- ancrés : /#accueil, /#projets, etc.
- rend :
  - email\_Javascript.htm
  - TransportRoutier\_Main.html
  - diapo\_Javascript\_Page1.html

views

---

Si tu veux, je peux aussi te faire un **schéma rapide “données → vues → templates”** (qui alimente quoi dans `index.html` et `page_administration.html`) à partir des clés envoyées dans les contextes.

Sources

admin.py

Python

models.py

Python

urls.py

Python

views.py

Python

contenu du dossier `app_contact`

Réflexion durant quelques secondes

Voici le **contenu (logique)** de ton dossier `app_contact` d'après les fichiers que tu as fournis :

## 1) admin.py

Il est vide (juste l'import admin + commentaire).

admin

---

## 2) models.py

Tu as 3 modèles principaux : **Contact**, **Role**, **User**.

models

#### **Contact**

- nom (CharField)
- email (EmailField)
- message (TextField)
- date\_envoi (DateTime auto)
- save() force l'email en **minuscule** (strip().lower()).
- \_\_str\_\_ retourne nom.

models

#### **Role**

- nom\_role (unique)
- \_\_str\_\_ retourne le nom du rôle.

models

#### **User**

- nom (obligatoire)
- email (unique + obligatoire)
- password (stocke un hash)
- role = ForeignKey vers Role avec on\_delete=models.CASCADE
- save() force l'email en **minuscule**
- \_\_str\_\_ affiche nom (role)

models

---

3) urls.py

Routes définies :

urls

- root\_gemini\_accueil → index\_inscription\_view
  - connexion/ → login\_view
  - deconnexion/ → deconnection\_view
  - page\_inscription → page\_inscription
- 

## 4) views.py

Ce fichier gère surtout **inscription + connexion + déconnexion** (et une vue “contact”).

views

### Utilitaires

- Regex pattern + fonction valider\_email(email) pour valider le format email.

views

### Pages

- page\_inscription(request) : rend page\_inscription.html.

views

- aff\_contact(request) : récupère tous les Contact et rend index.html avec formation = listeContact.

views

### Inscription (avec rôle)

index\_inscription\_view(request) :

- récupère nom, email (mis en lower), password, role (par défaut “Utilisateur”)
- valide :
  - nom non vide
  - email valide + non déjà utilisé
  - password non vide + longueur  $\geq 6$
- récupère/ crée le Role via `get_or_create(nom_role=...)`
- crée le User avec mot de passe hashé (`make_password`)
- crée une session `request.session['user'] = {id, nom, role}`
- redirige vers /#accueil (page d'accueil)

views

## Connexion

`login_view(request) :`

- cherche User par email (lower)
- vérifie le mot de passe hashé (`check_password`)
- si OK : stocke en session `{id, nom, email, role}` puis redirige /#accueil
- sinon : renvoie `login.html` avec message d'erreur.

views

## Déconnexion

`deconexion_view(request) :`

- vide la session via `request.session.flush()`
- redirige vers `name_acceuil` (⚠ ce nom d'URL doit exister ailleurs, sinon ça plantera).

views

Si tu veux, je peux aussi te faire un **schéma des flux** :

**inscription** → **création Role/User** → **session** → **redirection**, et **connexion** → **session** → **redirection**, pour clarifier où sont stockées les infos et comment elles sont lues ensuite.

Sources  
admin.py  
Python  
models.py  
Python  
urls.py  
Python  
views.py  
Python  
contenu du dossier app\_user  
Réflexion durant une seconde

Voici le **contenu (logique)** du dossier **app\_user** d'après tes fichiers :

## 1) admin.py

Fichier vide (imports + commentaire).

admin

---

## 2) models.py

Tu as 2 modèles principaux : **Menu** et **RoleMenuAcces**.

models

**Menu**

- nom (nom affiché)
- code\_menu (unique, identifiant technique)
- \_\_str\_\_ retourne nom

models

## RoleMenuAcces

C'est la table de liaison (**rôle ↔ menu**) avec un booléen :

- `role` (FK vers Role de app\_contact)
- `menu` (FK vers Menu)
- `est_visible` (par défaut False)
- `contrainte unique_together = ('role', 'menu')`
- `__str__` : "Role -> Menu (visible/caché)"

models

## ✓ Auto-initialisation via signaux :

- quand un **Role** est créé → crée toutes les permissions (Role, Menu) à False
- quand un **Menu** est créé → crée toutes les permissions (Role, Menu) à False

models

---

## 3) urls.py

Routes de gestion utilisateurs + permissions :

urls

- `root_gestion_utilisateur/` → page de gestion globale (via `GestionnaireUtilisateur().__Fonction_Page_gestion_des_Utilisateurs()`)
- `user/add/` → création user (via `GestionnaireUtilisateur().creer_un_nouvel_utilisateur()`)
- `user/edit/<user_id>/` → édition user (module externe)
- `user/delete/<user_id>/` → suppression user (module externe)
- `user/update-menus/` → sauvegarde matrice permissions (fonction `update_menus`)

⚠ Ici tu utilises des `lambda` qui appellent des méthodes d'un module externe `PackageUser.modul_new_user.GestionnaireUtilisateur`.

urls

---

## 4) views.py

Ce fichier gère surtout :

1. l'affichage de la page admin “utilisateurs + permissions + messages contact”
2. la sauvegarde des permissions (matrice rôles × menus)

views

### A) gestion\_globale(request)

- charge :
  - users (avec select\_related("role"))
  - roles
  - menus
  - contacts (triés par -date\_envoi)
- construit acces\_existant sous forme de liste : "roleId\_menuId" pour les accès visibles
- calcule mes\_menus\_visibles selon l'utilisateur connecté en session (request.session["user"]["id"]) et son rôle
- rend page\_utilisateur.html avec tout le contexte

views

### B) CRUD utilisateurs

- add\_user(request) délègue au module externe GestionnaireUtilisateur pour créer un user si POST, sinon redirection.

views

*(edit/delete sont indiqués comme étant dans PackageUser/modul\_new\_user.py, pas ici)*

views

### C) update\_menus(request)

- attend des inputs checkbox nommés : access\_<roleId>\_<menuId>
- boucle sur tous les rôles × menus
- fait update\_or\_create sur RoleMenuAcces avec est\_visible = True/False selon si la case est cochée
- message succès puis redirect vers gestion\_globale

views

---

Si tu veux, envoie aussi le fichier `PackageUser/modul_new_user.py` (ou le dossier `PackageUser`) et je te décris exactement **comment sont faits add/edit/delete, validations, hachage, etc.**

Sources

`modul_new_user.py`

Python

contenu du dossier `PackageUser`

Réflexion durant une seconde

Le dossier `PackageUser` (d'après le fichier fourni) contient essentiellement **un seul module** :

`PackageUser/modul_new_user.py`

Ce fichier définit une classe `GestionnaireUtilisateur` qui centralise la logique “utilisateurs” (page gestion, ajout, édition, suppression) + quelques utilitaires.

`modul_new_user`

## 1) Routeur “moteur central”

`creer_un_nouvel_utilisateur(request, nom_du_template) :`

- si `nom_du_template == "page_utilisateur.html"` → appelle `_add_user()` (ajout user depuis la page admin)
- si `nom_du_template == "index.html"` → appelle `_index_inscription_view()` ( cette méthode n'apparaît pas dans le fichier fourni, donc si elle n'existe pas ailleurs → erreur)
- sinon → rend `404.html`

`modul_new_user`

## 2) Page d'administration “utilisateurs + permissions + contacts”

\_Fonction\_Page\_gestion\_des\_Utilisateurs(...) :

- charge :
  - users (avec select\_related("role"))
  - menus
  - contacts (triés par -date\_envoi)
  - roles
- fabrique acces\_existant (liste de chaînes "roleId\_menuId" pour les accès visibles)
- calcule mes\_menus\_visibles pour l'utilisateur connecté via request.session["user"]["id"]
- renvoie page\_utilisateur.html avec tout le contexte + éventuelles erreurs + valeurs saisies

modul\_new\_user

## 3) Validation email

\_valider\_email(email) : regex simple pour vérifier le format.

modul\_new\_user

## 4) CRUD Utilisateur

### A) Ajouter un utilisateur (depuis la page admin)

\_add\_user(request) :

- récupère nom, email, password, role depuis POST
- valide :
  - nom obligatoire
  - email format + unicité (vérifie User.objects.filter(email=email).exists())
  - password obligatoire + longueur ≥ 6
  - rôle obligatoire
- si OK :
  - récupère le rôle via Role.objects.get(nom\_role=nom\_role\_choisi) (donc le rôle doit déjà exister)
  - crée User avec make\_password(password) et role=role\_obj
  - message succès + redirect gestion\_globale

- si rôle introuvable : message erreur “rôle n’existe pas en base”

modul\_new\_user

## B) Éditer un utilisateur

edit\_user(request, user\_id) :

- récupère l’utilisateur via `get_object_or_404(User, id=user_id)`
- valide :
  - nom obligatoire
  - email format + unicité (en excluant l’utilisateur actuel)
  - password obligatoire (et si pas déjà hashé pbkdf2\_, vérifie longueur  $\geq 6$ )
  - rôle obligatoire
- transforme le texte du rôle en objet : `Role.objects.get(nom_role=nom_role_choisi)`
- met à jour `user.nom`, `user.email`, `user.role`
- met à jour le mot de passe seulement si un nouveau est saisi et pas déjà hashé
- en cas d’erreurs de validation, renvoie vers `_Fonction_Page_gestion_des_UtilisateursEDIT(...)` pour réafficher la modale avec erreurs

modul\_new\_user

## C) Supprimer un utilisateur

delete\_user(request, user\_id) :

- seulement en POST
- supprime l’utilisateur, affiche un message, redirect `gestion_globale`

modul\_new\_user

## 5) Fonction dédiée au rendu “EDIT” (pour erreurs dans la modale)

`_Fonction_Page_gestion_des_UtilisateursEDIT(...)` :

- renvoie `page_utilisateur.html` avec :
  - `erreurs_champsEDIT`, `user_id_erreur`, `donnees_edit`, `roles` (liste de rôles)

modul\_new\_user

---

## ⚠ Remarques importantes (cohérence du projet)

- Le routeur appelle `_index_inscription_view()` pour "index.html", mais **je ne vois pas cette méthode dans modul\_new\_user.py** : si elle n'existe pas ailleurs, ça plantera.

modul\_new\_user

- Dans `_add_user`, tu ne fais pas `email.lower()` avant de tester l'unicité, donc si quelqu'un saisit des majuscules, ça peut créer un doublon selon le flux (même si ton `User.save()` dans `app_contact` normalise généralement).

modul\_new\_user

Si tu veux, je peux aussi te résumer **comment app\_user/urls.py appelle ce module** (les routes edit/delete/add passent par `GestionnaireUtilisateur`).

Sources

settings.py

Python

urls.py

Python

contenu du dossier portfolio

Réflexion durant une seconde

Voici le **contenu du dossier portfolio** (le “projet Django” principal) d’après les fichiers fournis : `settings.py` et `urls.py`.

---

## 1) portfolio/settings.py

Points clés de configuration :

settings

- Projet généré avec **Django 5.2**.

settings

- DEBUG = True et ALLOWED\_HOSTS = [] (donc dev/local).

settings

- INSTALLED\_APPS inclut :
  - app\_acceuil
  - app\_contact
  - app\_user

settings

- Templates :
  - DIRS = ['templates/']
  - APP\_DIRS = True

settings

- Base de données : **SQLite** (db.sqlite3).

settings

- Statics :
  - STATIC\_URL = 'static/'
  - STATICFILES\_DIRS = [BASE\_DIR/'static/'] (via os.path.join)

settings

- Médias :
  - MEDIA\_URL = '/media/'
  - MEDIA\_ROOT = BASE\_DIR/'media'

settings

---

## 2) portfolio/urls.py

Routing global du projet :

urls

- Admin Django : admin/
- Inclusion des URLs des apps (toutes au même niveau “root”):
  - path('', include("app\_acceuil.urls"))
  - path('', include("app\_contact.urls"))
  - path('', include("app\_user.urls"))

urls

- En mode DEBUG, ajout du service :
  - fichiers MEDIA\_URL (uploads)
  - fichiers statiques (via static(...))

urls

**⚠ Note :** Comme les 3 apps sont incluses sur path(''), il faut éviter les collisions d’URLs (même chemin défini dans plusieurs apps). Django prend la première correspondance selon l’ordre dans urlpatterns.

urls

---

Si tu veux, je peux aussi te faire la **carte complète des routes** (portfolio → app\_acceuil/app\_contact/app\_user) en listant toutes les URLs finales accessibles.

Sources

diapo\_Javascript\_Page1.html

Fichier

email\_Javascript.htm

Fichier

index.html

Fichier

page\_administration.html

Fichier  
page\_inscription.html  
Fichier  
page\_utilisateur.html  
Fichier  
TransportRoutier\_Main.html  
Fichier  
contenu du dossier templates  
Réflexion durant quelques secondes

Voici le **contenu du dossier templates/** (d'après les fichiers HTML que tu as uploadés) :

## 1) templates/index.html — Page d'accueil (portfolio)

- C'est la **page principale** du site (portfolio dynamique).
- Elle **étend** un layout : `{% extends "layouts/main.html" %}`.
- Elle contient de très longs blocs commentés + sections `#accueil`, etc., et utilise des données “utilisateur” (ex: `utilisateur.nom`, `utilisateur.email`).

index

---

## 2) templates/page\_administration.html — Dashboard admin (site/portfolio)

- C'est l'**espace administration** “Paramètres site”.
- UI Bootstrap avec **onglets** : Profil & Bio, Spécialité, Réalisations, Témoignages, Compétences, Contact, Stats vues, Graphique.
- Gère l'affichage/modif de la photo profil, bio, etc. (formulaires POST) + visibilité via switches.

page\_administration

---

## 3) templates/page\_utilisateur.html — Administration centrale (users + permissions + messages)

- Page “AdminPanel” (navbar + onglets).

- Onglets : **Utilisateurs, Permissions Menus, Messages**.
- Affiche les users avec rôle + actions (modals modifier/supprimer) et une matrice de permissions alimentée par `mes_menus_visibles` et `acces_existantes`.

page\_utilisateur

---

#### 4) `templates/page_inscription.html` — Page d’inscription (popup)

- Page qui affiche un bouton “**S’inscrire maintenant**” et ouvre une **popup** (overlay) responsive.
- Utilise W3.CSS et gère l’affichage des messages Django (`{% if messages %} ...`).

page\_inscription

---

#### 5) `templates/email_Javascript.htm` — Page “Contact / Devis” (énorme UI)

- Page orientée “contact/devis” avec gros CSS (Poppins, Bootstrap) + menu multi-niveaux.
- Utilise des liens Django `{% url ... %}` vers les sections de la page d’accueil (Accueil/Projets/Témoignages/À propos/Contact).

email\_Javascript

---

#### 6) `templates/diapo_Javascript_Page1.html` — Diaporama + menu multi-niveaux

- Page “diaporama” (W3.CSS) avec carrousel d’images statiques (`{% static 'images_stockees/...' %}`) et menu dropdown multi-niveaux.
- Contient aussi du code “popup overlay” et du CSS responsive “cards”.

diapo\_Javascript\_Page1

---

## 7) templates/TransportRoutier\_Main.html — Page “Transport routier / camions”

- Page dédiée aux **types de camions / transport de marchandises** (menu très riche).
- Fait des liens vers d'autres pages/templates via `{% url 'name_Page_diapo_Javascript_Page1' %}`, `{% url 'name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES' %}`, et des ancrées `/#contact`, etc.

TransportRoutier\_Main

---

Si tu veux, je peux aussi te faire une **carte “URLs → templates rendus”** (quel `view()` renvoie quel template) à partir de tes `urls.py/views.py`.

Sources

footer.html

Fichier

header.html

Fichier

main.html

Fichier

contenu du dossier templates/layouts

Réflexion durant 5s

Voici le **contenu du dossier templates/layouts/** d'après les 3 fichiers fournis :

---

## 1) templates/layouts/footer.html

Un footer très simple :

- Balise `<footer>`
- Texte copyright : “**© 2025 | Portfolio de Halley Mario. Tous droits réservés.**”

footer

---

## 2) templates/layouts/header.html

C'est l'en-tête avec la **navbar Bootstrap** (fixe en haut) et plusieurs menus :

header

### A) Menu “Portfolio” (dropdown)

Liens vers les ancrées de la page d'accueil via `name_acceuil` :

- `#accueil, #projets, #temoignages, #a-propos, #contact`

header

### B) Menu “Menus dynamiques” (dropdown) — basé sur `mes_menus_visibles`

Boucle Django :

- affiche *Ventes / Stocks / Factures / Rapports* si `acces.menu.nom` correspond

header

### C) Menu “Services” (dropdown multi-niveaux)

Sous-menus imbriqués (Service 2 → Service 3 → Service 5, etc.).

header

### D) Lien “Email”

Lien vers une page + ancre :

- `{% url 'name_Page_email_Javascript' %}#email_JAVASCRIPT`

header

## E) Menu “Administration”

Affichage conditionnel selon la session :

- si connecté (`request.session.user`) :
  - si rôle **Administrateur** → lien “Gestion des Utilisateurs” (`gestion_globale`)
  - si rôle **Administrateur ou Moderateur** → lien “Gestion d’Affichage du Site” (`dashboard_admin`)
  - sinon affiche des espaces “Utilisateur / Responsable de Saisie”, etc.

header

## F) Zone connexion / inscription (à droite)

- Si connecté : affiche le nom tronqué (`truncate-text`) + rôle, et un lien **Déconnexion** vers `{% url 'deconexion' %}`

header

- Sinon : liens **S’inscrire** (vers `name_acceuil#sinscrire`) + **Connexion** (vers `loginpage`)

header

---

## 3) `templates/layouts/main.html`

C’est le **layout principal** que tes pages étendent. Il contient :

main

## Structure Django

- `{% load static %}`
- Bloc meta description : `{% block 'desc' %}{% endblock %}`
- Bloc titre : `{% block "title" %}{% endblock %}`
- Corps :
  - `{% include "layouts/header.html" %}`
  - `{% block "contenu" %}{% endblock %}`

o { % include "layouts/footer.html" %}

main

## CSS/JS intégrés

- Bootstrap 5 (CDN), FontAwesome, flag-icons, W3CSS, etc.

main

- Beaucoup de CSS “site complet” (hero, projets, témoignages, à propos, contact, footer...).
- CSS utilitaire `truncate-text` (10ch + ellipsis).

main

- Script pour gérer le **dropdown multi-niveaux sur mobile** (matchMedia + toggle `.show`).

main

- Gros bloc JS “infobulle” (long, custom) présent plus bas dans le fichier.

main

---

Si tu veux, je peux te faire aussi un petit schéma : **page → extends main.html → include header/footer → blocs remplis**, pour que tu voies exactement où placer ton HTML (dans `block contenu`, etc.).

## Sources

Voici ma question : Je voudrais qu'il y ait la possibilité d'exporter les données dans un fichier JSON contenant des données de type "texte" mais également des données de type "image en base 64". Cela pourrait se faire dans le fichier "page\_administration.html" dans un nouvel onglet nommé par exemple "Exporter Données". Cet nouvel onglet contiendra principalement deux onglets fils dont le premier onglet aura comme rôle d'exporter les données correspondant aux données relatives au portfolio (sorte de cv en ligne), tandis que le deuxième onglet aura comme rôle d'exporter les données correspondant aux données relatives aux différentes utilisateurs. Remarques importantes sur le contenu du fichier JSON ( Donnée Texte et Donnée Image en base 64 ). L'objectif est en fait de pouvoir exporter toutes les données contenues dans la base de données SQLite concernant le portfolio et les données relatives aux différents utilisateurs dans deux fichiers JSON différents qui seront nommés par exemple "20260204\_export\_portfolio.json" et puis "20260204\_export\_users.json". Mais en tout cas c'est à

vous de voir, car dans un futur très proche ces deux fichiers JSON serviront à nourrir la base de données SQLite en important ces deux fichiers JSON dans le fichier `page_administration.html`  
Réflexion durant 58s

Voilà une façon **propre et évolutive** d'ajouter l'export **JSON (texte + images en Base64)** directement dans `page_administration.html` (qui est rendue par `page_Mon_Espace_Administration`)

views

, avec un **nouvel onglet “Exporter Données”** au même niveau que les onglets existants

`page_administration`

.

---

## 1) Back-end Django : 2 endpoints de téléchargement JSON

**A) Ajoute ces fonctions dans `app_acceuil/views.py`**

`page_Mon_Espace_Administration` existe déjà et rend `page_administration.html`

views

, et la route `dashboard` existe déjà

urls

.

Ajoute (ou colle vers la fin du fichier) :

```
import base64
import json
import mimetypes
from datetime import datetime

from django.http import HttpResponse, HttpResponseRedirect
```

```
from django.core.files.storage import default_storage
from django.db.models.fields.files import FileField, ImageField

def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _serialize_instance(obj):
    """
    Convertit un objet Django en dict JSON-friendly.
    - champs texte/nombre/bool => valeur directe
    - Image/File => base64 + meta (nom, mimetype)
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name

        # Skip id déjà mis
        if name == "id":
            continue

        value = getattr(obj, name)

        # Fichiers/images
        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name
                try:
                    with default_storage.open(file_path, "rb") as f:
                        raw = f.read()
                    b64 = base64.b64encode(raw).decode("utf-8")
                except Exception:
                    b64 = None

                mime, _ = mimetypes.guess_type(file_path)
                data[name] = {
                    "storage_path": file_path,
                    "filename": file_path.split("/")[-1],
                    "mimetype": mime or "application/octet-stream",
                    "base64": b64,
                }
            else:
                data[name] = None
        else:
            data[name] = value

    return data
```

```

        }
    else:
        data[name] = None
    else:
        data[name] = value

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

    # ⚠️ Adapte la liste si tu ajoutes d'autres modèles portfolio
    from .models import (
        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetmesrealisations,
        Projetscards,
        MesCompetencesCles,
        ReseauSocial,
    )

    payload = {
        "export_type": "portfolio",
        "exported_at": datetime.now().isoformat(),
        "data": {
            "Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
            "ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
            "Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
            "Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
            "Projetscards": _serialize_queryset(Projetscards.objects.all()),
            "MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
            "ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    filename = f"{date_prefix}_export_portfolio.json"

```

```

content = json.dumps(payload, ensure_ascii=False, indent=2)
resp = HttpResponse(content, content_type="application/json; charset=utf-8")
resp["Content-Disposition"] = f'attachment; filename="{filename}"'
return resp

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

    # Modèles utilisés dans ta gestion utilisateurs/permissions :contentReference[oaicite:4]{index=4}
    from app_contact.models import User, Role, Contact
    from app_user.models import Menu, RoleMenuAcces # :contentReference[oaicite:5]{index=5}

    payload = {
        "export_type": "users",
        "exported_at": datetime.now().isoformat(),
        "data": {
            "Role": _serialize_queryset(Role.objects.all()),
            "User": _serialize_queryset(User.objects.all()),
            "Contact": _serialize_queryset(Contact.objects.all()),
            "Menu": _serialize_queryset(Menu.objects.all()),
            "RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
        },
        "notes": [
            "Le champ User.password est exporté tel quel (hash). Si tu veux l'exclure, dis-le-moi et je te donne la variante.",
        ],
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    filename = f"{date_prefix}_export_users.json"

    content = json.dumps(payload, ensure_ascii=False, indent=2)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

```

## Ce code :

- télécharge un fichier JSON nommé comme tu veux (ex: 20260204\_export\_portfolio.json)
- inclut **images en base64** (si champs ImageField/FileField)
- inclut assez de méta (`__model__`, id, mimetype, filename) pour **réimporter proprement** plus tard.

## 2) Routes : ajoute 2 paths dans `app_acceuil/urls.py`

Dans `app_acceuil/urls.py` tu as déjà `dashboard_admin`

`urls`

. Ajoute :

```
urlpatterns = [
    # ...
    path('export/portfolio', export_portfolio_json, name='export_portfolio_json'),
    path('export/users', export_users_json, name='export_users_json'),
]
```

---

## 3) Front-end : nouvel onglet “Exporter Données” + 2 sous-onglets

Dans `page_administration.html`, la barre d'onglets principale est ici

`page_administration`

### A) Ajoute un nouvel onglet principal

Juste après “Graphique” :

```
<li class="nav-item">
    <button class="nav-link" data-bs-toggle="tab" data-bs-target="#exporter">
        Exporter Données
    </button>
</li>
```

### B) Ajoute la nouvelle “tab-pane” (avec 2 onglets fils)

À l'intérieur de `<div class="tab-content">` (au même niveau que `#graphique`, `#contact`, etc.)

`page_administration`

```
:  
<div class="tab-pane fade" id="exporter">  
  <div class="card card-custom p-4">  
    <h5 class="fw-bold mb-3">  
      <i class="fa-solid fa-file-export me-2 text-primary"></i>  
      Exporter Données (JSON + Base64)  
    </h5>  
  
    <!-- Onglets fils -->  
    <ul class="nav nav-tabs border-0" id="exportTab" style="margin-bottom:15px;">  
      <li class="nav-item">  
        <button class="nav-link active" data-bs-toggle="tab" data-bs-target="#export_portfolio">  
          Export Portfolio  
        </button>  
      </li>  
      <li class="nav-item">  
        <button class="nav-link" data-bs-toggle="tab" data-bs-target="#export_users">  
          Export Utilisateurs  
        </button>  
      </li>  
    </ul>  
  
<div class="tab-content" style="box-shadow:none; padding:0; margin-top:0;">  
  <!-- Export portfolio -->  
  <div class="tab-pane fade show active" id="export_portfolio">  
    <p class="text-muted mb-3">  
      Exporte toutes les données liées au portfolio (texte + images en base64).  
    </p>  
    <a class="btn btn-success"  
       href="{% url 'export_portfolio_json' %}">  
      Télécharger <b>YYYYMMDD_export_portfolio.json</b>  
    </a>  
  </div>  
  
  <!-- Export users -->  
  <div class="tab-pane fade" id="export_users">  
    <p class="text-muted mb-3">  
      Exporte les rôles, utilisateurs, contacts, menus et permissions.  
    </p>  
    <a class="btn btn-primary"  
       href="{% url 'export_users_json' %}">  
      Télécharger <b>YYYYMMDD_export_users.json</b>  
    </a>  
  </div>  
  
<div class="alert alert-warning small mt-3 mb-0">
```

```
Note : le champ <code>User.password</code> est un hash. (On peut l'exclure si tu préfères.)  
</div>  
</div>  
</div>  
</div>  
</div>
```

---

## 4) Points importants (pour ton futur import)

- Les images base64 sont exportées avec `filename`, `mimetype`, `storage_path`, `base64` → tu peux, à l'import :
    1. reconstruire un fichier (bytes depuis base64),
    2. le sauvegarder dans `MEDIA_ROOT`,
    3. remettre le champ `images` à ce nouveau chemin.
  - Pour les relations (ex: `User.role`), l'export inclut la valeur brute du champ (souvent l'`id`). Pour un import robuste, on se base sur ces IDs + le champ `__model__`.
- 

Si tu veux, je peux te donner **la partie “IMPORT” dans `page_administration.html`** (un 3e sous-onglet “Importer Données” avec upload de 2 JSON, parsing, et réinjection en SQLite), en réutilisant exactement le format export ci-dessus.

### Sources

Pouvez vous donner si vous y arriver tous les fichiers qui seront nécessaire aux exports des fichiers JSON et en même temps le 3e sous-onglet “Importer Données” avec upload de 2 JSON, parsing, et réinjection en SQLite , afin que votre raisonnement ne se perde pas. S'il vous plaît  
Réflexion durant 31s

Oui. Ci-dessous je te donne **tous les fichiers (ou parties de fichiers) nécessaires** pour :

1. **Exporter 2 fichiers JSON :**
  - `YYYYMMDD_export_portfolio.json` (portfolio + images en base64)
  - `YYYYMMDD_export_users.json` (users/roles/menus/permissions/contacts)
2. Ajouter dans `page_administration.html` un nouvel onglet **Exporter Données avec 3 sous-onglets :**
  - Export Portfolio
  - Export Utilisateurs

- Importer Données (upload des 2 JSON + parsing + réinjection SQLite)

Hypothèse : tes images sont stockées via `MEDIA_ROOT/media` (c'est déjà le cas dans ton `settings.py`) et tes modèles utilisent des `ImageField/FileField` (oui).

---

## 1) `app_acceuil/views.py` (AJOUT : export + import)

👉 Ouvre ton fichier actuel `app_acceuil/views.py` et **ajoute ce bloc à la fin** (ou au début si tu préfères), en adaptant seulement si tu as déjà des imports identiques.

```
# --- EXPORT / IMPORT JSON (texte + images base64) ---
import base64
import json
import mimetypes
from datetime import datetime

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import redirect

def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

def _serialize_instance(obj):
    """
```

```
Convertit un objet Django en dict JSON-friendly.
- champs simples => valeur directe
- Image/File => dict {filename, mimetype, base64, storage_path}
- ForeignKey => on stocke l'ID (valeur brute du champ)
"""

data = {
    "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
    "id": obj.pk,
}

for field in obj._meta.fields:
    name = field.name
    if name == "id":
        continue

    value = getattr(obj, name)

    # Fichiers/images
    if isinstance(field, (FileField, ImageField)):
        if value and getattr(value, "name", None):
            file_path = value.name
            try:
                with default_storage.open(file_path, "rb") as f:
                    raw = f.read()
                b64 = base64.b64encode(raw).decode("utf-8")
            except Exception:
                b64 = None

            mime, _ = mimetypes.guess_type(file_path)
            data[name] = {
                "storage_path": file_path,
                "filename": file_path.split("/")[-1],
                "mimetype": mime or "application/octet-stream",
                "base64": b64,
            }
        else:
            data[name] = None

    else:
        # ForeignKey : Django met l'id sur le champ name_id
        # Ici on prend la valeur brute (souvent int, str, bool, etc.)
        data[name] = value

return data
```

```

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORT: PORTFOLIO JSON
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from .models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetscards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
    "version": 1,
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_portfolio.json"
return _safe_json_response(payload, filename)

# -----
# EXPORT: USERS JSON
# -----
def export_users_json(request):
    if not _is_admin_session(request):

```

```
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
    "version": 1,
    "notes": [
        "User.password est exporté tel quel (hash).",
    ],
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_users.json"
return _safe_json_response(payload, filename)

# -----
# IMPORT (upload 2 JSON)
# -----
def _parse_uploaded_json(uploaded_file):
    """
    uploaded_file: InMemoryUploadedFile
    Retourne dict
    """
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        # fallback latin1
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    """
    file_dict = {filename, base64, mimetype, ...}

```

```

Ecrit le fichier dans le storage via le champ file/image (save()) .
"""
if not file_dict:
    return

b64 = file_dict.get("base64")
filename = file_dict.get("filename") or "file.bin"
if not b64:
    return

try:
    raw = base64.b64decode(b64)
except Exception:
    return

django_file = ContentFile(raw, name=filename)
field = getattr(model_instance, field_name)

# Sauvegarde via le champ => MEDIA_ROOT + upload_to
field.save(filename, django_file, save=False)

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    model_label: ex 'app_acceuil.Projetscards'
    objects_list: liste de dicts sérialisés
    keep_ids: si True, on tente de réutiliser l'id d'origine
    exclude_fields: ex {'password'} etc.
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")
        # data sans métas
        fields_data = {k: v for k, v in obj_data.items() if k not in {"__model__", "id"} and k not in exclude_fields}

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)):
                if field.name in fields_data:
                    file_payloads[field.name] = fields_data.pop(field.name)

    # ForeignKey: si on reçoit un objet dict par erreur, on ignore
    # Normalement c'est un int id.

```

```
for field in Model._meta.fields:
    if field.is_relation and field.many_to_one and field.name in fields_data:
        if isinstance(fields_data[field.name], dict):
            fields_data.pop(field.name, None)

if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()
    if instance is None:
        instance = Model(pk=obj_id)
else:
    instance = Model()

# appliquer champs simples
for k, v in fields_data.items():
    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    """
    Form dans page_administration.html -> upload 2 fichiers:
    - json_portfolio
    - json_users

    Optionnel: replace_portfolio / replace_users
    """
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")

    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    # 1) IMPORT PORTFOLIO
    if json_portfolio:
```

```

try:
    data = _parse_uploaded_json(json_portfolio)
    if data.get("export_type") != "portfolio":
        raise ValueError("Ce fichier n'est pas un export portfolio.")
    model_map = data.get("data", {})

    if replace_portfolio:
        # Supprimer dans un ordre simple (pas trop de FK ici)
        from .models import (
            Projetphotodeprofil,
            ProjetAproposDeMoi,
            Projetsfirstspeciality,
            Projetmesrealisations,
            Projetcards,
            MesCompetencesCles,
            ReseauSocial,
        )
        ReseauSocial.objects.all().delete()
        MesCompetencesCles.objects.all().delete()
        Projetcards.objects.all().delete()
        Projetmesrealisations.objects.all().delete()
        Projetsfirstspeciality.objects.all().delete()
        ProjetAproposDeMoi.objects.all().delete()
        Projetphotodeprofil.objects.all().delete()

    # upsert chaque modèle
    for model_label, objects_list in model_map.items():
        _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import portfolio: {e}")

# 2) IMPORT USERS
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})

        if replace_users:
            # Ordre important à cause des FK:
            # RoleMenuAcces -> User/Contact -> Menu -> Role
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

```

```

RoleMenuAcces.objects.all().delete()
User.objects.all().delete()
Contact.objects.all().delete()
Menu.objects.all().delete()
Role.objects.all().delete()

# IMPORTANT: importer Role et Menu AVANT RoleMenuAcces
ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        # on conserve les hash passwords tel quel
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

 Ce que fait l'import :

- Lit les 2 JSON uploadés.
  - Option “Remplacer existant” possible (wipe puis réinjection).
  - Réinjecte **texte + images** (les images sont re-crées depuis base64 et sauvegardées dans `MEDIA_ROOT` via `ImageField.save()`).
- 

## 2) `app_acceuil/urls.py` (AJOUT : 3 routes)

 Ajoute ces routes (en important les vues si besoin) :

```

from django.urls import path
from .views import (
    export_portfolio_json,

```

```

        export_users_json,
        import_json_bundle,
    )

urlpatterns = [
    # ... tes routes existantes ...

    path("export/portfolio", export_portfolio_json, name="export_portfolio_json"),
    path("export/users", export_users_json, name="export_users_json"),
    path("import/json", import_json_bundle, name="import_json_bundle"),
]

```

---

### 3) `templates/page_administration.html` (AJOUT : onglet + 3 sous-onglets)

#### A) Ajouter un nouvel onglet principal “Exporter Données”

Dans la barre d’onglets principale (là où tu as Profil/Bio, Spécialité, Réalisations, etc.), ajoute :

```

<li class="nav-item">
    <button class="nav-link" data-bs-toggle="tab" data-bs-target="#exporter">
        Exporter Données
    </button>
</li>

```

#### B) Ajouter la section `tab-pane` correspondante (avec 3 sous-onglets)

Toujours dans le `<div class="tab-content">`, ajoute :

```

<div class="tab-pane fade" id="exporter">
    <div class="card card-custom p-4">
        <h5 class="fw-bold mb-3">
            <i class="fa-solid fa-file-export me-2"></i>
            Export / Import (JSON + Base64)
        </h5>

        <!-- Sous-onglets -->
        <ul class="nav nav-tabs border-0" id="exportTab" style="margin-bottom:15px;">
            <li class="nav-item">
                <button class="nav-link active" data-bs-toggle="tab" data-bs-target="#export_portfolio">

```

```
        Export Portfolio
    </button>
</li>
<li class="nav-item">
    <button class="nav-link" data-bs-toggle="tab" data-bs-target="#export_users">
        Export Utilisateurs
    </button>
</li>
<li class="nav-item">
    <button class="nav-link" data-bs-toggle="tab" data-bs-target="#import_data">
        Importer Données
    </button>
</li>
</ul>

<div class="tab-content" style="box-shadow:none; padding:0; margin-top:0;">

    <!-- 1) Export portfolio -->
    <div class="tab-pane fade show active" id="export_portfolio">
        <p class="text-muted mb-3">
            Exporte toutes les données portfolio (texte + images en base64).
        </p>
        <a class="btn btn-success" href="{% url 'export_portfolio_json' %}">
            Télécharger le JSON Portfolio
        </a>
        <div class="small text-muted mt-2">
            Nom du fichier: <code>YYYYMMDD_export_portfolio.json</code>
        </div>
    </div>

    <!-- 2) Export users -->
    <div class="tab-pane fade" id="export_users">
        <p class="text-muted mb-3">
            Exporte rôles, utilisateurs, contacts, menus et permissions.
        </p>
        <a class="btn btn-primary" href="{% url 'export_users_json' %}">
            Télécharger le JSON Utilisateurs
        </a>
        <div class="small text-muted mt-2">
            Nom du fichier: <code>YYYYMMDD_export_users.json</code>
        </div>
        <div class="alert alert-warning small mt-3 mb-0">
            <b>Note :</b> <code>User.password</code> est exporté comme hash (normal).
        </div>
    </div>
</div>
```

```
<!-- 3) Import data -->
<div class="tab-pane fade" id="import_data">
  <p class="text-muted">
    Upload 1 ou 2 fichiers JSON exportés (portfolio / users). Tu peux choisir de remplacer les données existantes.
  </p>

  <form method="POST" action="{% url 'import_json_bundle' %}" enctype="multipart/form-data" class="mt-3">
    {% csrf_token %}

    <div class="row g-3">
      <div class="col-md-6">
        <label class="form-label fw-bold">JSON Portfolio</label>
        <input type="file" name="json_portfolio" class="form-control" accept=".json,application/json">
        <div class="form-check mt-2">
          <input class="form-check-input" type="checkbox" name="replace_portfolio" id="replace_portfolio">
          <label class="form-check-label" for="replace_portfolio">
            Remplacer toutes les données Portfolio existantes
          </label>
        </div>
      </div>
    </div>

    <div class="col-md-6">
      <label class="form-label fw-bold">JSON Utilisateurs</label>
      <input type="file" name="json_users" class="form-control" accept=".json,application/json">
      <div class="form-check mt-2">
        <input class="form-check-input" type="checkbox" name="replace_users" id="replace_users">
        <label class="form-check-label" for="replace_users">
          Remplacer toutes les données Utilisateurs existantes
        </label>
      </div>
    </div>
  </div>

  <button type="submit" class="btn btn-dark mt-4">
    Lancer l'import
  </button>

  <div class="alert alert-info small mt-3 mb-0">
    Conseil : si tu importes les utilisateurs, l'ordre interne est géré (Role/Menu avant permissions).
  </div>
</form>
</div>

</div>
</div>
</div>
```

## 4) (Optionnel mais recommandé) : afficher les messages Django dans

`page_administration.html`

Si ton template ne montre pas déjà `{% if messages %}`, ajoute ça en haut du contenu (souvent juste après le `<body>` ou en haut de la tab-content) :

```
{% if messages %}
  <div class="container mt-3">
    {% for message in messages %}
      <div class="alert alert-{{ message.tags }} mb-2">{{ message }}</div>
    {% endfor %}
  </div>
{% endif %}
```

Comme ça tu verras : “Import terminé” / “Erreur import ...”.

---

## 5) Résultat : fonctionnement

### Export

- Clique “Exporter Données” → “Export Portfolio” → télécharge `YYYYMMDD_export_portfolio.json`
- Clique “Export Utilisateurs” → télécharge `YYYYMMDD_export_users.json`

### Import

- Upload un ou deux fichiers.
- Option “Remplacer” si tu veux repartir à zéro.
- Submit → parsing + réinjection SQLite + restauration images depuis base64.

---

Si tu veux que l’import soit **100% identique** à l’origine (ex: respecter le `storage_path` original, ou exclure des champs sensibles, ou forcer un “mode fusion”), dis-moi juste “mode overwrite total” ou “mode merge”, et je te fournis la variante directement.

TypeError at /export/portfolio  
Object of type datetime is not JSON serializable  
Request Method: GET  
Request URL: http://127.0.0.1:8000/export/portfolio  
Django Version: 5.2.8  
Exception Type: TypeError  
Exception Value: Object of type datetime is not JSON serializable  
Exception Location: C:\Program Files\Python311\Lib\json\encoder.py, line 180, in default  
Raised during: app\_acceuil.views.export\_portfolio\_json  
Python Executable: D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Scripts\python.exe  
Python Version: 3.11.9  
Python Path: [D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO', 'C:\Program Files\Python311\python311.zip', 'C:\Program Files\Python311\DLLs', 'C:\Program Files\Python311\Lib', 'C:\Program Files\Python311', 'D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv', 'D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages']  
Server time: Wed, 04 Feb 2026 14:13:33 +0000  
Traceback Switch to copy-and-paste view D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\core\handlers\exception.py, line 55, in inner response = get\_response(request) ~~~~~ ... Local vars  
D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\core\handlers\base.py, line 197, in \_get\_response response = wrapped\_callback(request, \*callback\_args, \*\*callback\_kwargs) ~~~~~ ... Local vars  
D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\app\_acceuil\views.py, line 639, in export\_portfolio\_json return \_safe\_json\_response(payload, filename) ~~~~~ ... Local vars  
D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\app\_acceuil\views.py, line 547, in \_safe\_json\_response content = json.dumps(payload, ensure\_ascii=False, indent=2) ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\\_\_init\_\_.py, line 238, in dumps \*\*kw).encode(obj) ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 202, in encode chunks = list(chunks) ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 432, in \_iterencode yield from \_iterencode\_dict(o, \_current\_indent\_level) ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 406, in \_iterencode\_dict yield from chunks ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 406, in \_iterencode\_dict yield from chunks ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 326, in \_iterencode\_list yield from chunks ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 406, in \_iterencode\_dict yield from chunks ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 439, in \_iterencode o = \_default(o) ~~~~~ ... Local vars  
C:\Program Files\Python311\Lib\json\encoder.py, line 180, in default raise TypeError(f'Object of type {o.\_\_class\_\_.\_\_name\_\_} ' ~~~~~ ... Local vars  
Request information  
USER AnonymousUser  
GET No GET data  
POST No POST data  
FILES No FILES data  
COOKIES Variable Value csrf token  
\*\*\*\*\* sessionid \*\*\*\*\* META Variable Value ALLUSERSPROFILE 'C:\ProgramData' APPDATA  
'C:\Users\SPSSE02MEN\AppData\Roaming' BUNDLED\_DEBUGPY\_PATH 'c:\Users\SPSSE02MEN\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy' CHROME\_CRASHPAD\_PIPE\_NAME '\.\.\.\pipe\crashpad\_10848\_KQVIIBTMZBXUEGF' COLOR TERM 'truecolor' COMMONPROGRAMFILES 'C:\Program Files\Common Files' COMMONPROGRAMFILES(X86) 'C:\Program Files (x86)\Common Files' COMMONPROGRAMW6432 'C:\Program Files\Common Files' COMPUTERNAME 'DESKTOP-S95K8U0' COMSPEC 'C:\Windows\system32\cmd.exe' CONTENT\_LENGTH " CONTENT\_TYPE 'text/plain' CSRF\_COOKIE '2gGlBdFT35Ba5T2oZcyEdZ4w98DLjGf9' DJANGO\_SETTINGS\_MODULE 'portfolio.settings' DRIVERDATA 'C:\Windows\System32\DriverData' GATEWAY\_INTERFACE 'CGI/1.1' GIT\_ASKPASS  
\*\*\*\*\* GIT\_LFS\_PATH 'C:\Program Files\Git LFS' HOMEDRIVE 'C:' HOME PATH '\Users\SPSSE02MEN' HTTP\_ACCEPT 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8' HTTP\_ACCEPT\_ENCODING 'gzip, deflate, br, zstd' HTTP\_ACCEPT\_LANGUAGE 'fr,fr-FR;q=0.9,en-US;q=0.8,en;q=0.7' HTTP\_CONNECTION 'keep-alive' HTTP\_COOKIE '\*\*\*\*\*' HTTP\_HOST '127.0.0.1:8000'  
HTTP\_PRIORITY 'u=0, i' HTTP\_REFERER 'http://127.0.0.1:8000/dashboard' HTTP\_SEC\_FETCH\_DEST 'document' HTTP\_SEC\_FETCH\_MODE 'navigate' HTTP\_SEC\_FETCH\_SITE 'same-origin' HTTP\_SEC\_FETCH\_USER '?1' HTTP\_UPGRADE\_INSECURE\_REQUESTS '1' HTTP\_USER\_AGENT ('Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:147.0) Gecko/20100101 Firefox/147.0') LANG 'en\_US.UTF-8' LOCALAPPDATA  
'C:\Users\SPSSE02MEN\AppData\Local' LOGONSERVER '\DESKTOP-S95K8U0' NUMBER\_OF\_PROCESSORS '4' ONEDRIVE

'C:\\Users\\SPSSE02MEN\\OneDrive' OS 'Windows\_NT' PATH ('D:\\PYTHON DJANGO HOPES\\PORTFOLIO '  
'DJANGO\\.venv\\Scripts;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-  
chat\\debugCommand;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\copilotCli;C:\\Program '  
'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition  
2024.1.1\\bin;C:\\Program Files\\Common '  
'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\  
\\Windows\\System32\\OpenSSH\\;C:\\Program ' 'Files (x86)\\NVIDIA '  
'Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start '  
'Menu\\Programs\\Python;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsof  
t\\Windows\\Start 'Menu\\Programs;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Python\\Python311\\Scripts;C:\\Program ' 'Files\\Git\\cmd;C:\\Program '  
'Files\\nodejs\\;C:\\src\\flutter\\bin;C:\\dart-sdk\\bin;;C:\\Program ' 'Files\\Git LFS;C:\\Program Files\\Python3\_12\_4\\Scripts\\;C:\\Program '  
'Files\\Python3\_12\_4\\;%IntelliJ IDEA%;%IntelliJ IDEA Community ' 'Edition%;%PyCharm Community '  
'Edition%;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Microsoft VS 'Code\\bin;C:\\Program Files '  
(x86)\\Nmap;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\npm;c:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-  
x64\\bundled\\scripts\\noConfigScripts') PATHEXT '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW;.CPL' PATH\_INFO  
'export/portfolio' PROCESSOR\_ARCHITECTURE 'AMD64' PROCESSOR\_IDENTIFIER 'Intel64 Family 6 Model 69 Stepping 1, GenuineIntel'  
PROCESSOR\_LEVEL '6' PROCESSOR\_REVISION '4501' PROGRAMDATA 'C:\\ProgramData' PROGRAMFILES 'C:\\Program Files'  
PROGRAMFILES(X86) 'C:\\Program Files (x86)' PROGRAMW6432 'C:\\Program Files' PROMPT '\$P\$G' PSMODULEPATH  
(C:\\Users\\SPSSE02MEN\\Documents\\WindowsPowerShell\\Modules;C:\\Program '  
'Files\\WindowsPowerShell\\Modules;C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\Modules) PUBLIC 'C:\\Users\\Public'  
PYDEVD\_DISABLE\_FILE\_VALIDATION '1' QUERY\_STRING " REMOTE\_ADDR '127.0.0.1' REMOTE\_HOST " REQUEST\_METHOD 'GET'  
RUN\_MAIN 'true' SCRIPT\_NAME 'SERVER\_NAME 'DESKTOP-S95K8U0' SERVER\_PORT '8000' SERVER\_PROTOCOL 'HTTP/1.1'  
SERVER\_SOFTWARE 'WSGIServer/0.2' SYSTEMROOT 'C:\\Windows' TEMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp'  
TERM\_PROGRAM 'vscode' TERM\_PROGRAM\_VERSION '1.108.2' TMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp' USERDOMAIN 'AzureAD'  
USERDOMAIN\_ROAMINGPROFILE 'AzureAD' USERNAME 'SPSSE02MEN' USERPROFILE 'C:\\Users\\SPSSE02MEN' VIRTUAL\_ENV 'D:\\PYTHON  
DJANGO HOPES\\PORTFOLIO DJANGO\\.venv' VIRTUAL\_ENV\_PROMPT '.venv' VS CODE DEBUGPY\_ADAPTER\_ENDPOINTS  
'c:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\.noConfigDebugAdapterEndpoints\\endpoint-77f6ce4ded07263f.txt'  
VS CODE GIT ASKPASS\_EXTRA\_ARGS \*\*\*\*\* VS CODE GIT ASKPASS MAIN \*\*\*\*\*  
VS CODE GIT ASKPASS NODE \*\*\*\*\* VS CODE GIT IPC HANDLE '\\\\pipe\\vscode-git-c146c63ca8-sock' VS CODE INJECTION  
'1' VS CODE PYTHON AUTOACTIVATE GUARD '1' WINDIR 'C:\\Windows' \_OLD\_VIRTUAL\_PATH  
(C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-  
chat\\debugCommand;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\copilotCli;C:\\Program '  
'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition  
2024.1.1\\bin;C:\\Program Files\\Common '  
'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\  
\\Windows\\System32\\OpenSSH\\;C:\\Program ' 'Files (x86)\\NVIDIA '  
'Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start '

```
'Menu\Programs\Python;C:\Users\SPSSE02MEN\AppData\Local\Programs\Python\Launcher\;C:\Users\SPSSE02MEN\AppData\Roaming\Microsoft\Windows\Start 'Menu\Programs;C:\Users\SPSSE02MEN\AppData\Roaming\Python\Python311\Scripts;C:\Program 'Files\Git\cmd;C:\Program 'Files\nodejs\;C:\src\flutter\bin;C:\dart-sdk\bin;;C:\Program 'Files\Git LFS;C:\Program Files\Python3_12_4\Scripts\;C:\Program 'Files\Python3_12_4\;%IntelliJ IDEA%;%IntelliJ IDEA Community 'Edition%;%PyCharm Community '
'Edition%;C:\Users\SPSSE02MEN\AppData\Local\Programs\Microsoft VS 'Code\bin;C:\Program Files '
'(x86)\Nmap;C:\Users\SPSSE02MEN\AppData\Roaming\npm;: C:\Users\SPSSE02MEN\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\scripts\noConfigScripts) wsgi.errors < io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> wsgi.file_wrapper <class 'wsgiref.util.FileWrapper'\> wsgi.input <django.core.handlers.wsgi.LimitedStream object at 0x00000198C467D630> wsgi.multiprocess False wsgi.multithread True wsgi.run_once False wsgi.url_scheme 'http' wsgi.version (1, 0) Settings Using settings module portfolio.settings Setting Value
ABSOLUTE_URL_OVERRIDES {} ADMINS [] ALLOWED_HOSTS [] APPEND_SLASH True AUTHENTICATION_BACKENDS **** AUTH_PASSWORD_VALIDATORS **** AUTH_USER_MODEL **** BASE_DIR
WindowsPath('D:/PYTHON DJANGO HOPES/PORTFOLIO DJANGO') CACHES {'default': {'BACKEND':
'django.core.cache.backends.locmem.LocMemCache'}} CACHE_MIDDLEWARE_ALIAS 'default' CACHE_MIDDLEWARE_KEY_PREFIX **** CACHE_MIDDLEWARE_SECONDS 600 CSRF_COOKIE_AGE 31449600 CSRF_COOKIE_DOMAIN None
CSRF_COOKIE_HTTPONLY False CSRF_COOKIE_NAME 'csrftoken' CSRF_COOKIE_PATH '/' CSRF_COOKIE_SAMESITE 'Lax'
CSRF_COOKIE_SECURE False CSRF_FAILURE_VIEW 'django.views.csrf.csrf_failure' CSRF_HEADER_NAME 'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS [] CSRF_USE_SESSIONS False DATABASES {'default': {'ATOMIC_REQUESTS': False, 'AUTOCOMMIT': True,
'CONN_HEALTH_CHECKS': False, 'CONN_MAX_AGE': 0, 'ENGINE': 'django.db.backends.sqlite3', 'HOST': '', 'NAME': WindowsPath('D:/PYTHON DJANGO HOPES/PORTFOLIO DJANGO/db.sqlite3'), 'OPTIONS': {}, 'PASSWORD': '*****', 'PORT': '', 'TEST': {'CHARSET': None, 'COLLATION': None, 'MIGRATE': True, 'MIRROR': None, 'NAME': None}, 'TIME_ZONE': None, 'USER': ''}} DATABASE_ROUTERS []
DATA_UPLOAD_MAX_MEMORY_SIZE 2621440 DATA_UPLOAD_MAX_NUMBER_FIELDS 1000 DATA_UPLOAD_MAX_NUMBER_FILES 100
DATETIME_FORMAT 'N j, Y, P' DATETIME_INPUT_FORMATS ['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M',
'%m/%d/%Y %H:%M:%S', '%m/%d/%Y %H:%M:%S.%f', '%m/%d/%Y %H:%M', '%m/%d/%y %H:%M:%S', '%m/%d/%y %H:%M:%S.%f', '%m/%d/%y %H:%M'] DATE_FORMAT 'N j, Y' DATE_INPUT_FORMATS ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y', '%b %d %Y', '%b %d, %Y', '%d %b %Y', '%d %b %Y', '%B %d %Y', '%B %d, %Y', '%d %B %Y', '%d %B, %Y'] DEBUG True DEBUG_PROPAGATE_EXCEPTIONS False DECIMAL_SEPARATOR '.'
DEFAULT_AUTO_FIELD 'django.db.models.BigAutoField' DEFAULT_CHARSET 'utf-8' DEFAULT_EXCEPTION_REPORTER
'django.views.debug.ExceptionReporter' DEFAULT_EXCEPTION_REPORTER_FILTER 'django.views.debug.SafeExceptionReporterFilter'
DEFAULT_FROM_EMAIL 'webmaster@localhost' DEFAULT_INDEX_TABLESPACE '' DEFAULT_TABLESPACE '' DISALLOWED_USER_AGENTS []
EMAIL_BACKEND 'django.core.mail.backends.smtp.EmailBackend' EMAIL_HOST 'localhost' EMAIL_HOST_PASSWORD '*****'
EMAIL_HOST_USER '' EMAIL_PORT 25 EMAIL_SSL_CERTFILE None EMAIL_SSL_KEYFILE '*****' EMAIL SUBJECT_PREFIX
'[Django]' EMAIL_TIMEOUT None EMAIL_USE_LOCALTIME False EMAIL_USE_SSL False EMAIL_USE_TLS False
FILE_UPLOAD_DIRECTORY_PERMISSIONS None FILE_UPLOAD_HANDLERS ['django.core.files.uploadhandler.MemoryFileUploadHandler',
'django.core.files.uploadhandler.TemporaryFileUploadHandler'] FILE_UPLOAD_MAX_MEMORY_SIZE 2621440 FILE_UPLOAD_PERMISSIONS 420
FILE_UPLOAD_TEMP_DIR None FIRST_DAY_OF_WEEK 0 FIXTURE_DIRS [] FORCE_SCRIPT_NAME None FORMAT_MODULE_PATH None
FORMS_URLFIELD_ASSUME_HTTPS False FORM_RENDERER 'django.forms.renderers.DjangoTemplates' IGNORABLE_404_URLS []
INSTALLED_APPS ['django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages',
'django.contrib.staticfiles', 'app_acceuil', 'app_contact', 'app_user'] INTERNAL_IPS [] LANGUAGES [('af', 'Afrikaans'), ('ar', 'Arabic'), ('ar-dz', 'Algerian
```

Arabic'), ('ast', 'Asturian'), ('az', 'Azerbaijani'), ('bg', 'Bulgarian'), ('be', 'Belarusian'), ('bn', 'Bengali'), ('br', 'Breton'), ('bs', 'Bosnian'), ('ca', 'Catalan'), ('ckb', 'Central Kurdish (Sorani)'), ('cs', 'Czech'), ('cy', 'Welsh'), ('da', 'Danish'), ('de', 'German'), ('dsb', 'Lower Sorbian'), ('el', 'Greek'), ('en', 'English'), ('en-au', 'Australian English'), ('en-gb', 'British English'), ('eo', 'Esperanto'), ('es', 'Spanish'), ('es-ar', 'Argentinian Spanish'), ('es-co', 'Colombian Spanish'), ('es-mx', 'Mexican Spanish'), ('es-ni', 'Nicaraguan Spanish'), ('es-ve', 'Venezuelan Spanish'), ('et', 'Estonian'), ('eu', 'Basque'), ('fa', 'Persian'), ('fi', 'Finnish'), ('fr', 'French'), ('fy', 'Frisian'), ('ga', 'Irish'), ('gd', 'Scottish Gaelic'), ('gl', 'Galician'), ('he', 'Hebrew'), ('hi', 'Hindi'), ('hr', 'Croatian'), ('hsb', 'Upper Sorbian'), ('hu', 'Hungarian'), ('hy', 'Armenian'), ('ia', 'Interlingua'), ('id', 'Indonesian'), ('ig', 'Igbo'), ('io', 'Ido'), ('is', 'Icelandic'), ('it', 'Italian'), ('ja', 'Japanese'), ('ka', 'Georgian'), ('kab', 'Kabyle'), ('kk', 'Kazakh'), ('km', 'Khmer'), ('kn', 'Kannada'), ('ko', 'Korean'), ('ky', 'Kyrgyz'), ('lb', 'Luxembourgish'), ('lt', 'Lithuanian'), ('lv', 'Latvian'), ('mk', 'Macedonian'), ('ml', 'Malayalam'), ('mn', 'Mongolian'), ('mr', 'Marathi'), ('ms', 'Malay'), ('my', 'Burmese'), ('nb', 'Norwegian Bokmål'), ('ne', 'Nepali'), ('nl', 'Dutch'), ('nn', 'Norwegian Nynorsk'), ('os', 'Ossetic'), ('pa', 'Punjabi'), ('pl', 'Polish'), ('pt', 'Portuguese'), ('pt-br', 'Brazilian Portuguese'), ('ro', 'Romanian'), ('ru', 'Russian'), ('sk', 'Slovak'), ('sl', 'Slovenian'), ('sq', 'Albanian'), ('sr', 'Serbian'), ('sr-latn', 'Serbian Latin'), ('sv', 'Swedish'), ('sw', 'Swahili'), ('ta', 'Tamil'), ('te', 'Telugu'), ('tg', 'Tajik'), ('th', 'Thai'), ('tk', 'Turkmen'), ('tr', 'Turkish'), ('tt', 'Tatar'), ('udm', 'Udmurt'), ('ug', 'Uyghur'), ('uk', 'Ukrainian'), ('ur', 'Urdu'), ('uz', 'Uzbek'), ('vi', 'Vietnamese'), ('zh-hans', 'Simplified Chinese'), ('zh-hant', 'Traditional Chinese')]] LANGUAGES\_BIDI ['he', 'ar', 'ar-dz', 'ckb', 'fa', 'ug', 'ur'] LANGUAGE\_CODE 'en-us' LANGUAGE\_COOKIE\_AGE None LANGUAGE\_COOKIE\_DOMAIN None LANGUAGE\_COOKIE\_HTTPONLY False LANGUAGE\_COOKIE\_NAME 'django\_language' LANGUAGE\_COOKIE\_PATH '/' LANGUAGE\_COOKIE\_SAMESITE None LANGUAGE\_COOKIE\_SECURE False LOCALE\_PATHS [] LOGGING {} LOGGING\_CONFIG 'logging.config.dictConfig' LOGIN\_REDIRECT\_URL '/accounts/profile/' LOGIN\_URL '/accounts/login/' LOGOUT\_REDIRECT\_URL None MANAGERS [] MEDIA\_ROOT 'D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\media' MEDIA\_URL '/media/' MESSAGE\_STORAGE 'django.contrib.messages.storage.fallback.FallbackStorage' MIDDLEWARE ['django.middleware.security.SecurityMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware', 'django.middleware.common.CommonMiddleware', 'django.middleware.csrf.CsrfViewMiddleware', 'django.contrib.auth.middleware.AuthenticationMiddleware', 'django.contrib.messages.middleware.MessageMiddleware', 'django.middleware.clickjacking.XFrameOptionsMiddleware']] MIGRATION\_MODULES {} MONTH\_DAY\_FORMAT 'F j' NUMBER\_GROUPING 0 PASSWORD\_HASHERS '\*\*\*\*\*' PASSWORD\_RESET\_TIMEOUT '\*\*\*\*\*' PREPEND\_WWW False ROOT\_URLCONF 'portfolio.urls' SECRET\_KEY '\*\*\*\*\*' SECRET\_KEY\_FALLBACKS '\*\*\*\*\*' SECURE\_CONTENT\_TYPE\_NOSNIFF True SECURE\_CROSS\_ORIGIN\_OPENER\_POLICY 'same-origin' SECURE\_HSTS\_INCLUDE\_SUBDOMAINS False SECURE\_HSTS\_PRELOAD False SECURE\_HSTS\_SECONDS 0 SECURE\_PROXY\_SSL\_HEADER None SECURE\_REDIRECT\_EXEMPT [] SECURE\_REFERRER\_POLICY 'same-origin' SECURE\_SSL\_HOST None SECURE\_SSL\_REDIRECT False SERVER\_EMAIL 'root@localhost' SESSION\_CACHE\_ALIAS 'default' SESSION\_COOKIE\_AGE 1209600 SESSION\_COOKIE\_DOMAIN None SESSION\_COOKIE\_HTTPONLY True SESSION\_COOKIE\_NAME 'sessionid' SESSION\_COOKIE\_PATH '/' SESSION\_COOKIE\_SAMESITE 'Lax' SESSION\_COOKIE\_SECURE False SESSION\_ENGINE 'django.contrib.sessions.backends.db' SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE False SESSION\_FILE\_PATH None SESSION\_SAVE\_EVERY\_REQUEST False SESSION\_SERIALIZER 'django.contrib.sessions.serializers.JSONSerializer' SETTINGS\_MODULE 'portfolio.settings' SHORT\_DATETIME\_FORMAT 'm/d/Y P' SHORT\_DATE\_FORMAT 'm/d/Y' SIGNING\_BACKEND 'django.core.signing.TimestampSigner' SILENCED\_SYSTEM\_CHECKS [] STATICFILES\_DIRS ['D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\DJANGO\\static/] STATICFILES\_FINDERS ['django.contrib.staticfiles.finders.FileSystemFinder', 'django.contrib.staticfiles.finders.AppDirectoriesFinder'] STATIC\_ROOT None STATIC\_URL '/static/' STORAGES {'default': {'BACKEND': 'django.core.files.storage.FileSystemStorage'}, 'staticfiles': {'BACKEND': 'django.contrib.staticfiles.storage.StaticFilesStorage'}} TEMPLATES [{'APP\_DIRS': True, 'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': ['templates/'], 'OPTIONS': {'context\_processors': ['django.template.context\_processors.request', 'django.contrib.auth.context\_processors.auth', 'django.contrib.messages.context\_processors.messages']} }] TEST\_NON\_SERIALIZED\_APPS []

```
TEST_RUNNER 'django.test.runner.DiscoverRunner' THOUSAND_SEPARATOR '' TIME_FORMAT 'P' TIME_INPUT_FORMATS ['%H:%M:%S', '%H:%M:%S.%f', '%H:%M'] TIME_ZONE 'UTC' USE_I18N True USE_THOUSAND_SEPARATOR False USE_TZ True USE_X_FORWARDED_HOST False USE_X_FORWARDED_PORT False WSGI_APPLICATION 'portfolio.wsgi.application' X_FRAME_OPTIONS 'DENY' YEAR_MONTH_FORMAT 'F Y' You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard page generated by the handler for this status code.
```

Tu es tombé sur un cas classique : dans tes objets sérialisés, certains champs (souvent `DateTimeField`) renvoient un **objet datetime Python**, et `json.dumps()` ne sait pas le convertir tout seul → d'où :

```
Object of type datetime is not JSON serializable
```

 Solution : dans `_serialize_instance()`, il faut **convertir** automatiquement :

- `datetime / date / time` → `isoformat()`
- `Decimal` → `str()` (souvent utile)
- `UUID` → `str()`
- (et laisser le reste tel quel)

Je te donne le patch exact à faire.

---

## 1) Patch dans `app_acceuil/views.py`

### A) Ajoute ces imports (si absents)

En haut du fichier (dans la zone imports) :

```
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID
```

### B) Ajoute cette fonction utilitaire

Colle **juste avant `_serialize_instance()`** (ou au-dessus) :

```
def _json_compatible_value(value):
```

```

"""
Convertit les types non-JSON (datetime, date, Decimal, UUID...) en types JSON compatibles.
"""

if value is None:
    return None

if isinstance(value, (datetime, date, time)):
    return value.isoformat()

if isinstance(value, Decimal):
    return str(value)

if isinstance(value, UUID):
    return str(value)

return value

```

### C) Modifie `_serialize_instance()` (une seule ligne à changer)

Dans ta boucle, au lieu de faire :

```
data[name] = value
```

remplace par :

```
data[name] = _json_compatible_value(value)
```

Donc le “else” final devient :

```
else:
    data[name] = _json_compatible_value(value)
```

---

## 2) Option encore plus robuste (recommandé)

Dans `_safe_json_response()`, tu peux aussi sécuriser en mettant un `default=str`:

```
content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
```

Même si tu corriges bien `_serialize_instance()`, ça sert de “ceinture + bretelles”.

### 3) Pourquoi ça arrive chez toi ?

Tes modèles portfolio contiennent des dates, ex :

- Projetscards.date\_publication = models.DateTimeField(auto\_now\_add=True)  
models

Donc `value` est un `datetime`, et quand `_serialize_instance()` met ça dans un dict, `json.dumps()` plante.

### 4) Test rapide

Après patch :

- relance le serveur
- va sur : `http://127.0.0.1:8000/export/portfolio`  
→ le JSON doit se télécharger sans erreur.

Si tu veux, je peux aussi te donner une version qui **exclut** certains champs (ex: `User.password`) ou qui exporte aussi les **relations ForeignKey** avec plus d'infos (ex: `role: {id, nom_role}` au lieu d'un simple id).

Sources

Pouvez vous donner les fichiers complets

Je peux te donner **des fichiers complets**, mais je dois être transparent : je **n'ai pas ton app\_acceuil/views.py complet** (il contient déjà beaucoup de code dans ton projet). Donc je ne peux pas te renvoyer “le fichier complet exact” sans risquer d’écraser des fonctions existantes.

 La solution la plus sûre (et propre) pour avoir des **fichiers complets** sans casser ton projet, c'est de **créer un nouveau fichier dédié** à l'export/import (ex: `app_acceuil/json_transfer.py`) puis d'ajouter 3 routes dans `app_acceuil/urls.py`, et enfin d'ajouter l'onglet dans `page_administration.html`.

Je te donne ci-dessous **3 fichiers complets** à copier/coller :

1. app\_acceuil/json\_transfer.py (NOUVEAU, complet)
2. app\_acceuil/urls\_export\_import\_patch.py (exemple complet à fusionner dans ton urls.py)
3. templates/page\_administration\_export\_tab.html (bloc complet à intégrer dans page\_administration.html)

Ensuite je te dis exactement quoi modifier dans tes fichiers existants (2 lignes dans urls.py + collage HTML).

---

## 1) FICHIER COMPLET : `app_acceuil/json_transfer.py` (à créer)

Crée le fichier : `app_acceuil/json_transfer.py` et colle tout ceci :

```
import base64
import json
import mimetypes
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    """
    Basé sur ta logique de session :
    request.session['user'] = { id, nom, role, ... }
    """

    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")
```

```

def _json_compatible_value(value):
    """
    Rend une valeur sérialisable JSON.
    """
    if value is None:
        return None

    if isinstance(value, (datetime, date, time)):
        return value.isoformat()

    if isinstance(value, Decimal):
        return str(value)

    if isinstance(value, UUID):
        return str(value)

    return value


def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    """
    Construit une réponse HTTP de téléchargement JSON.
    default=str => sécurité si un type inconnu traîne.
    """
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp


# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    """
    Convertit un objet Django en dict JSON-friendly.
    - Champs simples: str/int/bool/... => convertis via _json_compatible_value
    - Image/File: export base64 + meta
    - ForeignKey: la valeur brute (souvent l'id) est OK (JSON)
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

```

```
for field in obj._meta.fields:
    name = field.name
    if name == "id":
        continue

    value = getattr(obj, name)

    # Fichiers / images
    if isinstance(field, (FileField, ImageField)):
        if value and getattr(value, "name", None):
            file_path = value.name
            b64 = None

            try:
                with default_storage.open(file_path, "rb") as f:
                    raw = f.read()
                b64 = base64.b64encode(raw).decode("utf-8")
            except Exception:
                b64 = None

        mime, _ = mimetypes.guess_type(file_path)
        data[name] = {
            "storage_path": file_path,
            "filename": file_path.split("/")[-1],
            "mimetype": mime or "application/octet-stream",
            "base64": b64,
        }
    else:
        data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")
```

```

# Import local des modèles portfolio
from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetcards": _serialize_queryset(Projetcards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}
date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_portfolio.json"
return _safe_json_response(payload, filename)

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "notes": [
        "User.password est exporté tel quel (hash).",
    ],
}

```

```
"data": {
    "app_contact.Role": _serialize_queryset(Role.objects.all()),
    "app_contact.User": _serialize_queryset(User.objects.all()),
    "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
    "app_user.Menu": _serialize_queryset(Menu.objects.all()),
    "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
},
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_users.json"
return _safe_json_response(payload, filename)

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    """
    file_dict = {"filename", "base64", "mimetype", "storage_path", ...}
    => écrit via ImageField/FileField.save()
    """
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
```

```
field.save(filename, django_file, save=False)

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    model_label ex: 'app_acceuil.Projetscards'
    objects_list: liste de dicts exportés
    keep_ids: conserve les id si possible
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")
        fields_data = {k: v for k, v in obj_data.items() if k not in {"__model__", "id"} and k not in exclude_fields}

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # créer instance
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first() or Model(pk=obj_id)
        else:
            instance = Model()

        # appliquer champs simples
        for k, v in fields_data.items():
            setattr(instance, k, v)

        # restaurer fichiers
        for field_name, payload in file_payloads.items():
            _restore_file_field(instance, field_name, payload)

        instance.save()

@transaction.atomic
def import_json_bundle(request):
    """
    POST depuis page_administration.html
    Inputs:
    - json_portfolio (optionnel)
    - json_users (optionnel)
    """
```

```

- replace_portfolio (checkbox)
- replace_users (checkbox)
"""

if not _is_admin_session(request):
    return HttpResponseRedirect("Accès refusé (Admin uniquement).")

if request.method != "POST":
    return redirect("dashboard_admin")

json_portfolio = request.FILES.get("json_portfolio")
json_users = request.FILES.get("json_users")
replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
    except Exception as e:
        messages.error(request, "Import Portfolio échoué : " + str(e))

    if replace_portfolio:
        from app_acceuil.models import (
            Projetphotodeprofil,
            ProjetAproposDeMoi,
            Projetsfirstspeciality,
            Projetmesrealisations,
            Projetcards,
            MesCompetencesCles,
            ReseauSocial,
        )
        # Ordre "simple"
        ReseauSocial.objects.all().delete()
        MesCompetencesCles.objects.all().delete()
        Projetcards.objects.all().delete()
        Projetmesrealisations.objects.all().delete()
        Projetsfirstspeciality.objects.all().delete()
        ProjetAproposDeMoi.objects.all().delete()
        Projetphotodeprofil.objects.all().delete()

        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")

```

```

messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})

        if replace_users:
            # Ordre important : dépendances FK
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
            Role.objects.all().delete()

        # Ordre d'import conseillé
        ordered = [
            "app_contact.Role",
            "app_user.Menu",
            "app_contact.User",
            "app_contact.Contact",
            "app_user.RoleMenuAcces",
        ]
        for model_label in ordered:
            if model_label in model_map:
                _upsert_objects(model_label, model_map[model_label], keep_ids=True)

        messages.success(request, "Import Utilisateurs terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import utilisateurs: {e}")

    return redirect("dashboard_admin")

```

 Ce fichier corrige ton erreur `datetime not JSON serializable` (`conversion automatique en isoformat()`).

---

## 2) MODIF ROUTES : `app_acceuil/urls.py` (ajout de 3 lignes)

Je ne peux pas écrire “ton `urls.py` complet” sans risque (tu as déjà beaucoup de routes).  
Donc voici la partie exacte à ajouter.

Dans `app_acceuil/urls.py` :

### A) Ajoute cet import

```
from app_acceuil.json_transfer import (
    export_portfolio_json,
    export_users_json,
    import_json_bundle,
)
```

### B) Ajoute ces 3 paths dans `urlpatterns`

```
path("export/portfolio", export_portfolio_json, name="export_portfolio_json"),
path("export/users", export_users_json, name="export_users_json"),
path("import/json", import_json_bundle, name="import_json_bundle"),
```

 Après ça, tes URLs seront :

- /export/portfolio
- /export/users
- /import/json

---

## 3) HTML COMPLET : bloc “Exporter Données” (à intégrer)

### A) Ajoute un onglet principal dans la navbar des tabs

Dans `page_administration.html`, là où tu as les tabs principaux (Profil, Spécialité, etc.), ajoute :

```

<li class="nav-item">
  <button class="nav-link" data-bs-toggle="tab" data-bs-target="#exporter">
    Exporter Données
  </button>
</li>

```

## B) Ajoute la tab-pane complète (avec 3 sous-onglets)

Toujours dans page\_administration.html à l'intérieur du `<div class="tab-content">`, colle ce bloc complet :

```

<div class="tab-pane fade" id="exporter">
  <div class="card card-custom p-4">
    <h5 class="fw-bold mb-3">
      <i class="fa-solid fa-file-export me-2"></i>
      Export / Import (JSON + Base64)
    </h5>

    <ul class="nav nav-tabs border-0" id="exportTab" style="margin-bottom:15px;">
      <li class="nav-item">
        <button class="nav-link active" data-bs-toggle="tab" data-bs-target="#export_portfolio">
          Export Portfolio
        </button>
      </li>
      <li class="nav-item">
        <button class="nav-link" data-bs-toggle="tab" data-bs-target="#export_users">
          Export Utilisateurs
        </button>
      </li>
      <li class="nav-item">
        <button class="nav-link" data-bs-toggle="tab" data-bs-target="#import_data">
          Importer Données
        </button>
      </li>
    </ul>

    <div class="tab-content" style="box-shadow:none; padding:0; margin-top:0;">

      <!-- Export portfolio -->
      <div class="tab-pane fade show active" id="export_portfolio">
        <p class="text-muted mb-3">
          Exporte toutes les données portfolio (texte + images en base64).
        </p>
        <a class="btn btn-success" href="{% url 'export_portfolio_json' %}">
          Télécharger le JSON Portfolio
        </a>
      </div>
    </div>
  </div>
</div>

```

```
</a>
<div class="small text-muted mt-2">
    Nom du fichier: <code>YYYYMMDD_export_portfolio.json</code>
</div>
</div>

<!-- Export users -->
<div class="tab-pane fade" id="export_users">
    <p class="text-muted mb-3">
        Exporte rôles, utilisateurs, contacts, menus et permissions.
    </p>
    <a class="btn btn-primary" href="{% url 'export_users_json' %}">
        Télécharger le JSON Utilisateurs
    </a>
    <div class="small text-muted mt-2">
        Nom du fichier: <code>YYYYMMDD_export_users.json</code>
    </div>

    <div class="alert alert-warning small mt-3 mb-0">
        <b>Note :</b> <code>User.password</code> est exporté en hash (normal).
    </div>
</div>

<!-- Import -->
<div class="tab-pane fade" id="import_data">
    <p class="text-muted">
        Upload 1 ou 2 fichiers JSON exportés (portfolio / users). Tu peux choisir de remplacer les données existantes.
    </p>

<form method="POST" action="{% url 'import_json_bundle' %}" enctype="multipart/form-data" class="mt-3">
    {% csrf_token %}

    <div class="row g-3">
        <div class="col-md-6">
            <label class="form-label fw-bold">JSON Portfolio</label>
            <input type="file" name="json_portfolio" class="form-control" accept=".json,application/json">
            <div class="form-check mt-2">
                <input class="form-check-input" type="checkbox" name="replace_portfolio" id="replace_portfolio">
                <label class="form-check-label" for="replace_portfolio">
                    Remplacer toutes les données Portfolio existantes
                </label>
            </div>
        </div>
    </div>

    <div class="col-md-6">
        <label class="form-label fw-bold">JSON Utilisateurs</label>
```

```

<input type="file" name="json_users" class="form-control" accept=".json,application/json">
<div class="form-check mt-2">
    <input class="form-check-input" type="checkbox" name="replace_users" id="replace_users">
    <label class="form-check-label" for="replace_users">
        REMPLACER toutes les données Utilisateurs existantes
    </label>
</div>
</div>
</div>

<button type="submit" class="btn btn-dark mt-4">
    Lancer l'import
</button>

<div class="alert alert-info small mt-3 mb-0">
    Conseil : l'ordre interne d'import est géré (Role/Menu avant permissions).
</div>
</form>
</div>

</div>
</div>
</div>

```

---

## 4) (Recommandé) Afficher les messages Django sur `page_administration.html`

Si tu ne les affiches pas déjà, ajoute quelque part en haut :

```

{% if messages %}
<div class="container mt-3">
    {% for message in messages %}
        <div class="alert alert-{{ message.tags }} mb-2">{{ message }}</div>
    {% endfor %}
</div>
{% endif %}

```

---

## 5) Vérification rapide (checklist)

1.  Créer `app_acceuil/json_transfer.py`

2.  Ajouter imports + 3 routes dans `app_acceuil/urls.py`
  3.  Ajouter l'onglet + tab-pane dans `page_administration.html`
  4.  Tester :
    - o `http://127.0.0.1:8000/export/portfolio`
    - o `http://127.0.0.1:8000/export/users`
    - o Import via le formulaire du dashboard
- 

## Si tu veux vraiment “les fichiers complets réels”

Pour te renvoyer **tes vrais fichiers complets** (pas une intégration propre), il me faut que tu m'envoies aussi :

- ton `app_acceuil/urls.py` actuel (le vrai)
- ton `page_administration.html` actuel (le vrai)
- (optionnel) ton `app_acceuil/views.py` actuel si tu préfères tout centraliser dedans

Dès que tu les uploades, je te renvoie **les 2/3 fichiers complets finalisés**, prêts à copier-coller, sans rien casser.

Juste une question à vous poser : Dans l'onglet "Importer Données", je voudrais savoir le rôle exacte de la case à cocher "Remplacer toutes les données Portfolio existantes"

La case "**Remplacer toutes les données Portfolio existantes**" sert à choisir **le mode d'import** :

### Si la case est cochée

Avant de réinjecter les données du JSON, le code fait un **nettoyage complet** des tables “portfolio” dans la base SQLite :

- suppression de **toutes** les lignes existantes (réseaux, compétences, témoignages, réalisations, spécialité, à propos, photo profil)
- puis **réimport** du JSON “comme source unique de vérité”

 Résultat : ta base portfolio devient **exactement** ce que contient le fichier JSON (aucun ancien enregistrement ne reste).

C'est utile quand :

- tu veux restaurer une sauvegarde “propre”

- tu migres vers une nouvelle base
- tu veux être sûr qu'il n'y a **pas de doublons** ni de restes

## Si la case n'est pas cochée

Le code fait un import en mode **fusion / mise à jour** (“upsert”) :

- pour chaque objet du JSON, il cherche un objet existant avec le **même id**
  - s'il existe → il le met à jour
  - sinon → il le crée
- et il **ne supprime pas** les objets qui existent dans la base mais qui ne sont pas dans le JSON

 Résultat : tu peux “ajouter / mettre à jour” sans perdre ce qui est déjà présent.

C'est utile quand :

- tu importes seulement une partie
- tu veux compléter sans effacer
- tu fais un import progressif

---

### Petit exemple concret

Base actuelle :

- 10 projets

JSON importé :

- 3 projets
-  **Case cochée** → tu finis avec **3 projets** (les 7 autres sont supprimés)
-  **Case décochée** → tu finis avec **10 projets** (les 3 sont mis à jour/ajoutés, les 7 restent)

Si tu veux, je peux te proposer une 3e option “**Synchroniser**” (la plus logique pour une sauvegarde) :

“supprimer uniquement ce qui n'est pas dans le JSON”, pour que base = JSON **exactement**, mais sans suppression brutale si tu importes seulement une partie.

Mais est ce que ce pourrait être un grand problème si je decide de ne pas ajouter la 3e option que vous me proposez ?

Non, **ce n'est pas un grand problème** de ne pas ajouter la 3e option. Tu peux très bien fonctionner avec seulement ces 2 modes :

- **Remplacer** (wipe + import)
- **Fusionner** (upsert sans suppression)

Dans beaucoup de projets, ça suffit largement.

Ce qu'il faut juste comprendre, c'est le “risque” associé au mode **Fusionner** (case décochée) :

### **Ce qui peut arriver si tu n'as pas la 3e option**

#### 1. **Données “fantômes” qui restent**

- Si ton JSON n'inclut pas certains enregistrements, ils resteront quand même en base.
- Exemple : tu avais 20 projets, ton JSON n'en contient plus que 10 (parce que tu en as supprimé dans la source).  
→ après import en fusion, tu auras encore **20** en base (les 10 “supprimés” reviennent/continuent d'exister).

#### 2. **Doublons dans certains cas**

- Si tu importes un JSON venant d'une autre base où les `id` ne correspondent pas (ou si tu as recréé des objets), le upsert peut **créer** au lieu de “mettre à jour”.
- Résultat : tu peux te retrouver avec des doublons “logiques” (même projet, IDs différents).

#### 3. **Incohérences de “source de vérité”**

- Sans un mode “sync exact”, tu dois décider :
  - soit tu veux que la base reflète **exactement** le JSON → il faut “Remplacer”
  - soit tu veux juste **ajouter/mettre à jour** sans retirer → fusion

### **Pourquoi ce n'est pas dramatique**

Parce que tu as déjà un bouton “Remplacer” :

- Quand tu veux une restauration fidèle (base = JSON) → tu coches **Remplacer** et tu es tranquille.
- Quand tu veux compléter → tu décoches.

## Quand tu ressentiras le besoin de la 3e option

Tu en auras vraiment besoin surtout si :

- tu fais des imports fréquents,
- tu veux garder des données ajoutées localement **mais** supprimer celles retirées du JSON (sync partiel),
- tu veux éviter de “tout effacer” à chaque fois.

---

En résumé : **tu peux totalement t'en passer au début.**

Si ton usage principal est “restaurer une sauvegarde complète”, alors “**Remplacer**” suffit.

Si tu me dis comment tu vas utiliser l’import (restauration complète ? synchronisation régulière ? migration entre PC ?), je te dirai quel mode est le plus sûr pour toi.

Si vous pensez que la 3e option est la plus sûre alors vous pouvez me donner le code

Oui — la 3e option la plus sûre est un mode “**Synchroniser (base = JSON)**” :

mettre à jour / créer ce qui est dans le JSON et supprimer ce qui n'y est pas, sans forcément tout effacer d'un coup.

Ci-dessous, je te donne le **code complet à ajouter/modifier** dans ton fichier `app_acceuil/json_transfer.py` (ou dans le fichier où tu as mis `import_json_bundle`).

---

## 1) Back-end : ajouter le mode “Synchroniser” (suppression des absents)

### A) Ajoute cette fonction utilitaire (une fois) dans `json_transfer.py`

```
def _sync_delete_not_in_json(model_label: str, json_objects: list):
```

```

"""
Supprime en base tous les objets dont l'id n'est pas présent dans le JSON.
"""

Model = apps.get_model(model_label)
ids_in_json = [obj.get("id") for obj in json_objects if obj.get("id") is not None]
Model.objects.exclude(pk__in=ids_in_json).delete()

```

## B) Modifie import\_json\_bundle() pour gérer une nouvelle checkbox “sync\_\*”

Dans import\_json\_bundle, remplace/complète la récupération des options :

```

replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

sync_portfolio = bool(request.POST.get("sync_portfolio"))
sync_users = bool(request.POST.get("sync_users"))

```

Règle simple (très important) : si replace\_\* est coché, il prend le dessus sur sync\_\*.  
Donc, juste après la lecture des flags, ajoute :

```

if replace_portfolio:
    sync_portfolio = False
if replace_users:
    sync_users = False

```

---

## C) Partie PORTFOLIO : appliquer la synchro (suppression des absents)

Dans le bloc portfolio (là où tu fais model\_map = data.get("data", {})), après le upsert (création/mise à jour), ajoute :

```

# --- Upsert ---
for model_label, objects_list in model_map.items():
    _upsert_objects(model_label, objects_list, keep_ids=True)

# --- Sync delete (supprime ce qui n'est pas dans le JSON) ---
if sync_portfolio:
    for model_label, objects_list in model_map.items():
        _sync_delete_not_in_json(model_label, objects_list)

```

Résultat : après import, la base “portfolio” reflète exactement le JSON, mais sans wipe brutal.

## D) Partie USERS : appliquer la synchro en respectant l'ordre FK

Pour users, il y a des dépendances (RoleMenuAcces dépend de Role et Menu).

Donc il faut supprimer dans **un ordre safe**.

Dans le bloc users, après l'upsert ordonné :

```
ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)
```

Ajoute juste après :

```
if sync_users:
    # supprimer dans l'ordre inverse (enfants -> parents)
    reverse_delete = [
        "app_user.RoleMenuAcces",
        "app_contact.User",
        "app_contact.Contact",
        "app_user.Menu",
        "app_contact.Role",
    ]
    for model_label in reverse_delete:
        if model_label in model_map:
            _sync_delete_not_in_json(model_label, model_map[model_label])
```

**⚠️ Important :**

- Si dans ton JSON users tu n'inclus **pas** un modèle (ex: tu n'exportes pas Contact), alors on ne le touche pas (car on teste `if model_label in model_map`).

## 2) Front-end : ajouter les checkboxes “Synchroniser”

Dans page\_administration.html, onglet **Importer Données**, ajoute 2 cases supplémentaires.

### Pour Portfolio (sous ta checkbox replace\_portfolio)

```
<div class="form-check mt-2">
  <input class="form-check-input" type="checkbox" name="sync_portfolio" id="sync_portfolio">
  <label class="form-check-label" for="sync_portfolio">
    Synchroniser Portfolio (supprimer en base ce qui n'est pas dans le JSON)
  </label>
</div>
```

### Pour Users (sous replace\_users)

```
<div class="form-check mt-2">
  <input class="form-check-input" type="checkbox" name="sync_users" id="sync_users">
  <label class="form-check-label" for="sync_users">
    Synchroniser Utilisateurs (supprimer en base ce qui n'est pas dans le JSON)
  </label>
</div>
```

 Tu peux aussi ajouter une note :

```
<div class="alert alert-secondary small mt-3 mb-0">
  <b>Synchroniser</b> = la base devient identique au JSON (mise à jour + suppression des absents).
  <b>Remplacer</b> = suppression totale puis import (prioritaire si coché).
</div>
```

---

## 3) Comportement final (clair)

-  **Remplacer** : purge totale + import (radical, mais sûr pour une restauration complète)
  -  **Synchroniser** : upsert + suppression des absents (le plus “propre” pour maintenir base = JSON)
  -  **Fusionner** (aucune case) : upsert seulement, sans suppression
-

Si tu veux, je peux aussi te donner une micro-sécurité :

→ empêcher de cocher “Synchroniser” et “Remplacer” en même temps côté JS (désactivation automatique), mais côté Python tu es déjà protégé (replace prioritaire).

json\_transfer.py

Python

Voici mon fichier

Réflexion durant 6s

Parfait — ton fichier json\_transfer.py est déjà très bien structuré.

json\_transfer

Je te donne ci-dessous la version complète de ce fichier avec la 3e option “Synchroniser” (upsert + suppression des objets absents du JSON), sans casser tes 2 modes existants.

---

## ✓ Nouveau json\_transfer.py complet (avec “Synchroniser”)

Remplace le contenu de ton fichier par celui-ci :

```
import base64
import json
import mimetypes
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden
from django.shortcuts import redirect

# -----
```

```
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    """
    Basé sur ta logique de session :
    request.session['user'] = { id, nom, role, ... }
    """
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    """
    Rend une valeur sérialisable JSON.
    """
    if value is None:
        return None

    if isinstance(value, (datetime, date, time)):
        return value.isoformat()

    if isinstance(value, Decimal):
        return str(value)

    if isinstance(value, UUID):
        return str(value)

    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    """
    Construit une réponse HTTP de téléchargement JSON.
    default=str => sécurité si un type inconnu traîne.
    """
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    """
```

```
Convertit un objet Django en dict JSON-friendly.
- Champs simples: str/int/bool/... => convertis via _json_compatible_value
- Image/File: export base64 + meta
- ForeignKey: la valeur brute (souvent l'id) est OK (JSON)
"""
data = {
    "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
    "id": obj.pk,
}

for field in obj._meta.fields:
    name = field.name
    if name == "id":
        continue

    value = getattr(obj, name)

    # Fichiers / images
    if isinstance(field, (FileField, ImageField)):
        if value and getattr(value, "name", None):
            file_path = value.name
            b64 = None

            try:
                with default_storage.open(file_path, "rb") as f:
                    raw = f.read()
                b64 = base64.b64encode(raw).decode("utf-8")
            except Exception:
                b64 = None

            mime, _ = mimetypes.guess_type(file_path)
            data[name] = {
                "storage_path": file_path,
                "filename": file_path.split("/")[-1],
                "mimetype": mime or "application/octet-stream",
                "base64": b64,
            }
        else:
            data[name] = None
    else:
        data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
```

```
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    # Import local des modèles portfolio
    from app_acceuil.models import (
        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetsmesrealisations,
        Projetscards,
        MesCompetencesCles,
        ReseauSocial,
    )

    payload = {
        "export_type": "portfolio",
        "exported_at": datetime.now().isoformat(),
        "version": 1,
        "data": {
            "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
            "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
            "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
            "app_acceuil.Projetsmesrealisations": _serialize_queryset(Projetsmesrealisations.objects.all()),
            "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
            "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
            "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    filename = f"{date_prefix}_export_portfolio.json"
    return _safe_json_response(payload, filename)

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_contact.models import User, Role, Contact
```

```
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "notes": [
        "User.password est exporté tel quel (hash).",
    ],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_users.json"
return _safe_json_response(payload, filename)
```

```
# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    """
    file_dict = {"filename", "base64", "mimetype", "storage_path", ...}
    => écrit via ImageField/FileField.save()
    """
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
```

```

return

try:
    raw = base64.b64decode(b64)
except Exception:
    return

django_file = ContentFile(raw, name=filename)
field = getattr(model_instance, field_name)
field.save(filename, django_file, save=False)

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    model_label ex: 'app_acceuil.Projetscards'
    objects_list: liste de dicts exportés
    keep_ids: conserve les id si possible
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")
        fields_data = {k: v for k, v in obj_data.items() if k not in {"__model__", "id"} and k not in exclude_fields}

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # créer instance
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first() or Model(pk=obj_id)
        else:
            instance = Model()

        # appliquer champs simples
        for k, v in fields_data.items():
            setattr(instance, k, v)

        # restaurer fichiers
        for field_name, payload in file_payloads.items():
            _restore_file_field(instance, field_name, payload)

        instance.save()

```

```

def _sync_delete_not_in_json(model_label: str, json_objects: list):
    """
    Mode "Synchroniser" :
    Supprime en base tous les objets dont l'id n'est pas présent dans le JSON.
    """
    Model = apps.get_model(model_label)
    ids_in_json = [obj.get("id") for obj in json_objects if obj.get("id") is not None]
    Model.objects.exclude(pk__in=ids_in_json).delete()

@transaction.atomic
def import_json_bundle(request):
    """
    POST depuis page_administration.html
    Inputs:
    - json_portfolio (optionnel)
    - json_users (optionnel)

    Modes Portfolio:
    - replace_portfolio : wipe + import
    - sync_portfolio    : upsert + delete absents du JSON (base = JSON)

    Modes Users:
    - replace_users : wipe + import
    - sync_users   : upsert + delete absents du JSON (base = JSON)
    """

    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")

    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    sync_portfolio = bool(request.POST.get("sync_portfolio"))
    sync_users = bool(request.POST.get("sync_users"))

    # Priorité au wipe
    if replace_portfolio:
        sync_portfolio = False

```

```

if replace_users:
    sync_users = False

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})

        if replace_portfolio:
            from app_acceuil.models import (
                Projetphotodeprofil,
                ProjetAproposDeMoi,
                Projetsfirstspeciality,
                Projetmesrealisations,
                Projetcards,
                MesCompetencesCles,
                ReseauSocial,
            )
            # Ordre "simple"
            ReseauSocial.objects.all().delete()
            MesCompetencesCles.objects.all().delete()
            Projetcards.objects.all().delete()
            Projetmesrealisations.objects.all().delete()
            Projetsfirstspeciality.objects.all().delete()
            ProjetAproposDeMoi.objects.all().delete()
            Projetphotodeprofil.objects.all().delete()

        # upsert
        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

        # sync delete (base = JSON)
        if sync_portfolio:
            for model_label, objects_list in model_map.items():
                _sync_delete_not_in_json(model_label, objects_list)

        messages.success(request, "Import Portfolio terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:

```

```

data = _parse_uploaded_json(json_users)
if data.get("export_type") != "users":
    raise ValueError("Ce fichier n'est pas un export users.")
model_map = data.get("data", {})

if replace_users:
    # Ordre important : dépendances FK
    from app_user.models import RoleMenuAcces, Menu
    from app_contact.models import User, Role, Contact

    RoleMenuAcces.objects.all().delete()
    User.objects.all().delete()
    Contact.objects.all().delete()
    Menu.objects.all().delete()
    Role.objects.all().delete()

# Ordre d'import conseillé
ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

# sync delete (inverse: enfants -> parents)
if sync_users:
    reverse_delete = [
        "app_user.RoleMenuAcces",
        "app_contact.User",
        "app_contact.Contact",
        "app_user.Menu",
        "app_contact.Role",
    ]
    for model_label in reverse_delete:
        if model_label in model_map:
            _sync_delete_not_in_json(model_label, model_map[model_label])

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

## Ajouts à faire dans `page_administration.html` (Importer Données)

Sous tes deux cases actuelles “Remplacer...”, ajoute :

### Sous “Remplacer Portfolio...”

```
<div class="form-check mt-2">
  <input class="form-check-input" type="checkbox" name="sync_portfolio" id="sync_portfolio">
  <label class="form-check-label" for="sync_portfolio">
    Synchroniser Portfolio (supprimer en base ce qui n'est pas dans le JSON)
  </label>
</div>
```

### Sous “Remplacer Utilisateurs...”

```
<div class="form-check mt-2">
  <input class="form-check-input" type="checkbox" name="sync_users" id="sync_users">
  <label class="form-check-label" for="sync_users">
    Synchroniser Utilisateurs (supprimer en base ce qui n'est pas dans le JSON)
  </label>
</div>
```

## Comment ça marche maintenant (très clair)

- **Remplacer**  : supprime tout puis importe (base = JSON)
- **Synchroniser**  : met à jour/ajoute + supprime les absents (base = JSON, mais sans wipe brutal)
- **Aucune case**  : met à jour/ajoute seulement (merge, pas de suppression)

Si tu veux, je peux aussi te fournir un mini JS dans le template pour empêcher de cocher “Remplacer” et “Synchroniser” en même temps (même si côté Python tu es déjà protégé par la priorité “Remplacer”).

Sources

J'ai une question : Si je voudrais ne plus faire aucune saisie en utilisant le clavier de l'ordinateur mais que je voudrais alimenter uniquement ma base de données uniquement en décidant d'importer petit à petit des fichiers json , alors si je ne coche pas la case à cocher "Remplacer toutes les données Portfolio existantes" alors la base de données sera t elle toujours ajoutée si l'id n'existe pas ou mise à jour si l'id existe déjà et tout cela sans aucune suppression de données éventuelles ? Je ne parle pas encore là de la 3e option synchroniser

Oui — exactement.

Si tu **n'utilises pas** la case “**Remplacer toutes les données Portfolio existantes**” (et sans parler de “Synchroniser”), alors ton import fonctionne en mode **fusion / upsert** :

### Ce que fait l'import dans ce mode (case *Remplacer* décochée)

Pour **chaque objet** dans le JSON :

- **Si `id` existe déjà en base** → l'objet est **mis à jour** (update)
- **Si `id` n'existe pas en base** → l'objet est **créé** (insert)
- **Aucune suppression** n'est faite :  
tout ce qui est déjà en base et **absent du JSON** reste en place.

Donc oui : tu peux alimenter ta base **petit à petit**, par imports successifs, sans clavier, et ça ajoutera/mettre-à-jour sans jamais supprimer de données.

---

### Important : la condition pour que ça marche “par id”

Pour que “update si existe / insert sinon” marche correctement, ton JSON doit contenir des **id stables** et cohérents :

- Si tu réexportes depuis la **même base** → les ids seront stables
  - Si tu fabriques les JSON à la main ou depuis une autre base où les ids diffèrent → tu risques de créer des doublons (car l'id ne correspondra pas).
- 

### Exemple concret

Base actuelle :

- Projetmesrealisations : id 1, 2, 3

JSON importé contient :

- id 2 (modifié)
- id 4 (nouveau)

Résultat (mode fusion) :

- id 1 (reste)
- id 2 (mis à jour)
- id 3 (reste)
- id 4 (ajouté)

→ Aucune suppression.

---

## Petit “piège” à connaître

Si un objet dans ton JSON n'a **pas** de `id` (ou `id=None`), alors ton code va créer un nouvel objet (et potentiellement des doublons).  
Donc si ton objectif est 100% “sans clavier” et propre, assure-toi que tes JSON exportés contiennent bien les `id` (ce que ton export fait déjà).

---

Si tu veux, je peux te proposer une petite amélioration : un mode “fusion intelligente” basé sur un champ unique (ex: `nom` pour compétences, `email` pour réseaux), pour éviter les doublons même si les ids changent.

`json_transfer.py`

Python

Oui ce serait mieux. Voici mon fichier `json_transfer.py` car j'ai décidé de ne pas ajouter la 3e option "synchroniser"  
Réflexion durant une seconde

Oui, on peut faire **une “fusion intelligente”** (sans synchroniser, sans suppression) :

→ si l'**ID n'existe pas**, on essaie de retrouver l'objet via une **clé unique métier** (ex: `email`, `nom`, `code_menu`, etc.).

- si on le trouve, on **met à jour** cet objet au lieu d'en créer un nouveau.
- sinon on **crée**.

Ton fichier actuel fait un upsert **uniquement par id**

json\_transfer

— donc si tu importes un JSON avec des IDs différents, tu peux obtenir des doublons. La fusion intelligente évite ça.

---

## 1) Ce qu'on va ajouter (sans “Synchroniser”)

- Une table de “règles de matching” par modèle (ex: MesCompetencesCles → nom, ReseauSocial → nom, User → email, etc.)
- Une variante de `_upsert_objects()` qui :
  1. update si `id` trouvé
  2. sinon cherche par “clé unique”
  3. sinon crée

 Et **aucune suppression**, même si des objets ne sont pas dans le JSON.

---

## 2) Patch à appliquer à ton `json_transfer.py`

### A) Ajoute ce mapping (juste au-dessus de `_upsert_objects`)

Colle ceci **avant** la fonction `_upsert_objects(...)` :

```
# -----
# FUSION INTELLIGENTE: règles "clé métier" par modèle
# -----
SMART_MATCH_KEYS = {
    # ---- Portfolio ----
    "app_acceuil.MesCompetencesCles": ["nom"],           # nom est unique dans ton modèle
    "app_acceuil.ReseauSocial": ["nom"],                  # nom=email/whatsapp/linkedin/etc.
    "app_acceuil.Projetmesrealisations": ["nom_projet"],
```

```

"app_acceuil.Projetscards": ["nom"],           # témoignage: nom (peut être duplifiable -> à toi de décider)
# Singleton-ish: on peut aussi matcher sur le 1er objet existant
"app_acceuil.Projetsfirstspeciality": ["id"],   # (souvent 1 seul)
"app_acceuil.Projetphotodeprofil": ["id"],       # (souvent 1 seul)
"app_acceuil.ProjetAproposDeMoi": ["id"],         # (souvent 1 seul)

# ---- Users ----
"app_contact.Role": ["nom_role"],                # unique
"app_contact.User": ["email"],                   # unique
"app_user.Menu": ["code_menu"],                  # unique
# Table pivot : unique_together(role, menu)
"app_user.RoleMenuAcces": ["role", "menu"],
}

```

Remarque : Projetscards.nom n'est pas forcément unique en vrai. Si tu veux éviter des faux "match", je te conseille plutôt ["nom", "description"] (ou enlever ce modèle du matching intelligent).

---

## B) Remplace ta fonction `_upsert_objects(...)` par cette version améliorée

⚠ Remplace entièrement la fonction `_upsert_objects` actuelle

json\_transfer

```

par celle-ci :
def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    """
        Tente de retrouver un objet existant via une clé métier.
        Ex: User par email, Role par nom_role, Menu par code_menu.
    """
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    # Clé composée (ex: role + menu)
    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        lookup[k] = fields_data[k]

```

```

# Si une valeur est None, on abandonne
if any(v is None for v in lookup.values()):
    return None

try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    UPSERT intelligent:
    1) Si id existe -> update
    2) Sinon -> tentative match par clés métier (SMART_MATCH_KEYS)
    3) Sinon -> create
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        # data sans métas
        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        instance = None

        # 1) update par ID si possible
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first()

        # 2) sinon tentative "clé métier"
        if instance is None:
            instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

        # 3) sinon créer

```

```
if instance is None:
    if keep_ids and obj_id is not None:
        instance = Model(pk=obj_id)
    else:
        instance = Model()

# appliquer champs simples
for k, v in fields_data.items():
    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()
```

---

### 3) Ce que ça change concrètement pour toi

Avec ce patch :

- Tu peux importer des JSON “petit à petit”
- Même si les IDs changent, tu auras **moins de doublons**
  - User → match par email
  - Role → match par nom\_role
  - Menu → match par code\_menu
  - MesCompetencesCles → match par nom
  - ReseauSocial → match par nom (email/linkedin/etc.)
  - RoleMenuAcces → match par (role, menu)

Et toujours :

- **update si trouvé**
  - **insert sinon**
  - **aucune suppression** (puisque tu ne veux pas synchroniser)
-

## 4) Deux petites recommandations (pour éviter des surprises)

1. **Projetscards** : si plusieurs témoignages peuvent avoir le même nom, le matching par nom peut remplacer un mauvais témoignage.  
→ soit tu ajoutes une clé plus sûre (nom + description), soit tu enlèves ce modèle des SMART\_MATCH\_KEYS.
  2. **Projetmesrealisations** : nom\_projet doit être stable. Si tu le modifies, ça créera un nouvel objet (normal).
- 

Si tu veux, je peux adapter précisément les SMART\_MATCH\_KEYS **en regardant tes champs exacts** de chaque modèle (surtout Projetscards / Projetmesrealisations) pour éviter tout risque de “mauvais match”.

Sources

Projetscards.nom n'est pas forcément unique en vrai. Si tu veux éviter des faux “match”, je te conseille plutôt ["nom", "description"] (ou enlever ce modèle du matching intelligent).

Réflexion durant quelques secondes

Oui — si Projetscards.nom n'est pas unique, **le plus sûr** (sans supprimer quoi que ce soit) est de faire le “matching intelligent” sur une **clé plus discriminante**, par ex. ["nom", "description"].

Comme ton json\_transfer.py actuel fait l'upsert **uniquement par id**

json\_transfer

, voici exactement quoi changer pour activer cette fusion intelligente (et donc éviter les doublons si les IDs changent).

---

### 1) Ajoute le mapping des clés métier

Dans ton json\_transfer.py, ajoute **juste au-dessus** de \_upsert\_objects(...) ce bloc :

```
# -----
# FUSION INTELLIGENTE: clés métier par modèle
# (si id absent ou différent, on tente de retrouver un objet existant)
#
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],           # unique
```

```

"app_acceuil.ReseauSocial": ["nom"], # email / whatsapp / linkedin / etc.
"app_acceuil.Projetmesrealisations": ["nom_projet"], # stable si tu gardes les mêmes noms

# Témoignages: nom pas unique => nom + description
"app_acceuil.Projetscards": ["nom", "description"],

# Users
"app_contact.Role": ["nom_role"], # unique
"app_contact.User": ["email"], # unique
"app_user.Menu": ["code_menu"], # unique
"app_user.RoleMenuAcces": ["role", "menu"], # clé composée (unique_together)
}


```

---

## 2) Remplace `_upsert_objects()` par une version “smart”

Dans ton fichier, remplace complètement ta fonction `_upsert_objects(...)` actuelle

`json_transfer`

par cette version :

```

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    """
        Retrouve un objet existant via une clé métier (SMART_MATCH_KEYS).
        Exemple:
            - User par email
            - Role par nom_role
            - Menu par code_menu
            - Projetscards par (nom, description)
            - RoleMenuAcces par (role, menu)
    """
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        lookup[k] = fields_data[k]

    # si une clé est vide/None, on ne matche pas
    if any(v is None or v == "" for v in lookup.values()):

```

```

    return None

try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    UPSERT intelligent:
    1) Si id existe en base -> update
    2) Sinon -> tentative match via SMART_MATCH_KEYS
    3) Sinon -> create
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        instance = None

        # 1) update par ID si possible (uniquement si l'objet existe déjà)
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first()

        # 2) sinon match intelligent
        if instance is None:
            instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

        # 3) sinon create
        if instance is None:
            if keep_ids and obj_id is not None:
                instance = Model(pk=obj_id)

```

```
else:  
    instance = Model()  
  
    # appliquer champs simples  
    for k, v in fields_data.items():  
        setattr(instance, k, v)  
  
    # restaurer fichiers  
    for field_name, payload in file_payloads.items():  
        _restore_file_field(instance, field_name, payload)  
  
instance.save()
```

---

### 3) Résultat concret pour `projetscards`

Avec `["nom", "description"]` :

- si tu importes un témoignage avec **même nom + même description**, alors il sera **mis à jour** (même si l'id est différent)
- si tu importes un témoignage avec **même nom mais description différente**, il sera **créé** (ce qui est logique)

Et surtout : **aucune suppression**, puisque tu n'utilises pas “Remplacer” et tu n'ajoutes pas “Synchroniser”.

---

### Petite note pratique

Si dans tes JSON la `description` change très légèrement (un espace, une ponctuation), ça peut “casser” le match et créer un doublon.

Si tu veux rendre ça plus tolérant, je peux te proposer une variante qui normalise `nom/description` (trim, espaces multiples, lowercase) avant de matcher.

Sources

oui

Parfait — voici la variante “fusion intelligente **tolérante**” : on **normalise** les champs utilisés pour matcher (trim, espaces multiples, lowercase) afin d’éviter les doublons dus à une petite différence de saisie (“ ”, retours ligne, majuscules...).

### 1) Ajoute ces fonctions de normalisation dans `json_transfer.py`

Mets-les au-dessus de `_get_instance_by_smart_keys(...)`:

```
import re

def _norm_text(s: str) -> str:
    """Normalise un texte pour matching: trim + espaces multiples + lowercase."""
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # remplace \n, \t, espaces multiples
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s) # email => lowercase + trim

def _normalized_lookup_value(model_label: str, field_name: str, value):
    """
    Normalisation par modèle/champ.
    On ne change pas les ints (FK), bool, etc.
    """
    if value is None:
        return None

    # emails
    if model_label == "app_contact.User" and field_name == "email":
        return _norm_email(str(value))

    # Témoignages: nom + description
    if model_label == "app_acceuil.Projetscards" and field_name in ("nom", "description"):
        return _norm_text(str(value))

    # Compétences: nom
    if model_label == "app_acceuil.MesCompetencesCles" and field_name == "nom":
        return _norm_text(str(value))

    # Réseaux: nom (email/linkedin/etc.) + valeur(s) éventuelles (si tu ajoutes)
    if model_label == "app_acceuil.ReseauSocial" and field_name == "nom":
        return _norm_text(str(value))

    # Rôles / menus
    if model_label == "app_contact.Role" and field_name == "nom_role":
        return _norm_text(str(value))

    if model_label == "app_user.Menu" and field_name == "code_menu":
        return _norm_text(str(value))

    # Par défaut: on ne touche pas
```

```
return value
```

## 2) Remplace `_get_instance_by_smart_keys(...)` par cette version normalisée

(Elle construit un lookup normalisé, puis cherche en base.)

```
def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None

        raw_val = fields_data[k]
        val = _normalized_lookup_value(model_label, k, raw_val)

        # si vide, on ne matche pas
        if val is None or (isinstance(val, str) and val.strip() == ""):
            return None

        lookup[k] = val

    # ⚠ Pour matcher correctement, il faut aussi normaliser les données en base.
    # On gère ça par un "scan" léger uniquement pour les cas texte (Projetscards surtout).
    # Pour les champs non-textes (FK, ints), le filter direct suffit.

    # Cas Projetscards: matching tolérant => on compare après normalisation en Python
    if model_label == "app_acceuil.Projetscards":
        # on limite les candidats par nom si possible pour éviter full scan
        candidates = Model.objects.all()
        if "nom" in lookup:
            # on ne peut pas faire filter sur nom normalisé directement => on filtre grossier
            candidates = candidates.filter(nom__isnull=False)

        for obj in candidates:
            nom_db = _norm_text(getattr(obj, "nom", ""))
            desc_db = _norm_text(getattr(obj, "description", ""))
            if nom_db == lookup.get("nom") and desc_db == lookup.get("description"):
                return obj
    return None
```

```
# Cas email/role/menu/competences : filter direct OK (car stockage déjà "propre" souvent)
try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None
```

### 3) (Recommandé) Normaliser aussi avant `setattr` pour Projetscards

Sinon tu peux matcher “propre”, mais enregistrer “sale”. Tu peux laisser tel quel, ou appliquer :

Dans `_upsert_objects(...)`, juste avant `setattr(instance, k, v)` :

```
# Normaliser à l'enregistrement pour certains champs texte (optionnel mais utile)
if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
    v = v.strip()
```

---

#### Ce que ça t'apporte

- Tu peux importer des JSON “petit à petit”.
- Tu évites la majorité des doublons “bêtes” (espaces, retours ligne, majuscules).
- Tu gardes ton mode : **ajout si pas trouvé / mise à jour si trouvé / aucune suppression**.

Si tu veux, je peux aussi te donner une version encore plus robuste avec un **champ `fingerprint`** (hash) calculé à l’import pour les témoignages, ce qui rend le matching ultra fiable et rapide (sans scan).

```
import base64 import json import mimetypes from datetime import datetime, date, time from decimal import Decimal from uuid import UUID from django.apps import apps from django.contrib import messages from django.core.files.base import ContentFile from django.core.files.storage import default_storage from django.db import transaction from django.db.models.fields.files import FileField, ImageField from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden from django.shortcuts import redirect # ----- # Helpers : sécurité + JSON # -----
_is_admin_session(request) -> bool: """ Basé sur ta logique de session : request.session['user'] = { id, nom, role, ... } """ user_data = request.session.get("user")
return bool(user_data and user_data.get("role") == "Administrateur") def _json_compatible_value(value): """ Rend une valeur sérialisable JSON. """ if value is None: return None if isinstance(value, (datetime, date, time)): return value.isoformat() if isinstance(value, Decimal): return str(value) if isinstance(value, UUID): return str(value) return value def _safe_json_response(payload: dict, filename: str) -> HttpResponseRedirect: """ Construit une réponse HTTP de téléchargement JSON. default=str => sécurité si un type inconnu traîne. """ content = json.dumps(payload, ensure_ascii=False, indent=2, default=str) resp = HttpResponseRedirect(content, content_type="application/json; charset=utf-8") resp["Content-Disposition"] = f'attachment; filename="{filename}"' return resp # -----
# Serialize : Django -> dict # -----
def _serialize_instance(obj): """ Convertit un objet Django en dict JSON-friendly. - Champs simples: str/int/bool/... => convertis via _json_compatible_value - Image/File: export base64 + meta - ForeignKey: la valeur brute (souvent l'id) est OK
```

```

(JSON) """ data = { "__model__": f'{obj._meta.app_label}.{obj._class_.__name__}', "id": obj.pk, } for field in obj._meta.fields: name = field.name if name == "id": continue value = getattr(obj, name) # Fichiers / images if isinstance(field, (FileField, ImageField)): if value and getattr(value, "name", None): file_path = value.name b64 = None try: with default_storage.open(file_path, "rb") as f: raw = f.read() b64 = base64.b64encode(raw).decode("utf-8") except Exception: b64 = None mime, _ = mimetypes.guess_type(file_path) data[name] = { "storage_path": file_path, "filename": file_path.split("/")[-1], "mimetype": mime or "application/octet-stream", "base64": b64, } else: data[name] = None else: data[name] = _json_compatible_value(value) return data def _serialize_queryset(qs): return [_serialize_instance(o) for o in qs] # ----- # EXPORTS # ----- def export_portfolio_json(request): if not _is_admin_session(request): return HttpResponseForbidden("Accès refusé (Admin uniquement).") # Import local des modèles portfolio from app_acceuil.models import (Projetphotodeprofil, ProjetAproposDeMoi, Projetsfirstspeciality, Projetmesrealisations, Projetscards, MesCompetencesCles, ReseauSocial, ) payload = { "export_type": "portfolio", "exported_at": datetime.now().isoformat(), "version": 1, "data": { "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()), "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()), "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()), "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()), "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()), "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()), "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()), }, } date_prefix = datetime.now().strftime("%Y%m%d") filename = f'{date_prefix}_export_portfolio.json' return _safe_json_response(payload, filename) def export_users_json(request): if not _is_admin_session(request): return HttpResponseForbidden("Accès refusé (Admin uniquement).") from app_contact.models import User, Role, Contact from app_user.models import Menu, RoleMenuAcces payload = { "export_type": "users", "exported_at": datetime.now().isoformat(), "version": 1, "notes": [ "User.password est exporté tel quel (hash).", ], "data": { "app_contact.Role": _serialize_queryset(Role.objects.all()), "app_contact.User": _serialize_queryset(User.objects.all()), "app_contact.Contact": _serialize_queryset(Contact.objects.all()), "app_user.Menu": _serialize_queryset(Menu.objects.all()), "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()), }, } date_prefix = datetime.now().strftime("%Y%m%d") filename = f'{date_prefix}_export_users.json' return _safe_json_response(payload, filename) # ----- # IMPORTS # ----- def _parse_uploaded_json(uploaded_file): raw = uploaded_file.read() try: text = raw.decode("utf-8") except Exception: text = raw.decode("latin-1") return json.loads(text) def _restore_file_field(model_instance, field_name: str, file_dict: dict): """ file_dict = {filename, base64, mimetype, storage_path, ...} => écrit via ImageField/FileField.save() """ if not file_dict: return b64 = file_dict.get("base64") filename = file_dict.get("filename") or "file.bin" if not b64: return try: raw = base64.b64decode(b64) except Exception: return django_file = ContentFile(raw, name=filename) field = getattr(model_instance, field_name) field.save(filename, django_file, save=False) # ----- # FUSION INTELLIGENTE: clés métier par modèle # (si id absent ou différent, on tente de retrouver un objet existant) # ----- SMART_MATCH_KEYS = { # Portfolio "app_acceuil.MesCompetencesCles": ["nom"], # unique "app_acceuil.ReseauSocial": ["nom"], # email / whatsapp / linkedin / etc. "app_acceuil.Projetmesrealisations": ["nom_projet"], # stable si tu gardes les mêmes noms # Témoignages: nom pas unique => nom + description "app_acceuil.Projetscards": ["nom", "description"], # Users "app_contact.Role": ["nom_role"], # unique "app_contact.User": ["email"], # unique "app_user.Menu": ["code_menu"], # unique "app_user.RoleMenuAcces": ["role", "menu"], # clé composée (unique_together) } import re def _norm_text(s: str) -> str: """Normalise un texte pour matching: trim + espaces multiples + lowercase.""" s = (s or "").strip() s = re.sub(r"\s+", " ", s) # remplace \n, \t, espaces multiples return s.lower() def _norm_email(s: str) -> str: return _norm_text(s) # email => lowercase + trim def _normalized_lookup_value(model_label: str, field_name: str, value): """ Normalisation par modèle/champ. On ne change pas les ints (FK), bool, etc. """ if value is None: return None # emails if model_label == "app_contact.User" and field_name == "email": return _norm_email(str(value)) # Témoignages: nom + description if model_label == "app_acceuil.Projetscards" and field_name in ("nom", "description"): return _norm_text(str(value)) # Compétences: nom if model_label == "app_acceuil.MesCompetencesCles" and field_name == "nom": return _norm_text(str(value)) # Réseaux: nom (email/linkedin/etc.) + valeur(s) éventuelles (si

```

tu ajoutes) if model\_label == "app\_acceuil.ReseauSocial" and field\_name == "nom": return \_norm\_text(str(value)) # Rôles / menus if model\_label == "app\_contact.Role" and field\_name == "nom\_role": return \_norm\_text(str(value)) if model\_label == "app\_user.Menu" and field\_name == "code\_menu": return \_norm\_text(str(value)) # Par défaut: on ne touche pas return value def \_get\_instance\_by\_smart\_keys(Model, model\_label: str, fields\_data: dict): keys = SMART\_MATCH\_KEYS.get(model\_label) if not keys: return None lookup = {} for k in keys: if k not in fields\_data: return None raw\_val = fields\_data[k] val = \_normalized\_lookup\_value(model\_label, k, raw\_val) # si vide, on ne matche pas if val is None or (isinstance(val, str) and val.strip() == ""): return None

lookup[k] = val # Pour matcher correctement, il faut aussi normaliser les données en base. # On gère ça par un "scan" léger uniquement pour les cas texte (Projetscards surtout). # Pour les champs non-textes (FK, ints), le filter direct suffit. # Cas Projetscards: matching tolérant => on compare après normalisation en Python if model\_label == "app\_acceuil.Projetscards": # on limite les candidats par nom si possible pour éviter full scan candidates = Model.objects.all() if "nom" in lookup: # on ne peut pas faire filter sur nom normalisé directement => on filtre grossier candidates = candidates.filter(nom\_\_isnull=False) for obj in candidates: nom\_db = \_norm\_text(getattr(obj, "nom", "")) desc\_db = \_norm\_text(getattr(obj, "description", "")) if nom\_db == lookup.get("nom") and desc\_db == lookup.get("description"): return obj return None # Cas email/role/menu/competences : filter direct OK (car stockage déjà "propre" souvent) try: return Model.objects.filter(\*\*lookup).first() except Exception: return None def \_upsert\_objects(model\_label: str, objects\_list: list, keep\_ids: bool = True, exclude\_fields: set | None = None): """
 UPSERT intelligent: 1) Si id existe en base -> update 2) Sinon -> tentative match via SMART\_MATCH\_KEYS 3) Sinon -> create """
 Model = apps.get\_model(model\_label)
 exclude\_fields = exclude\_fields or set()
 for obj\_data in objects\_list:
 obj\_id = obj\_data.get("id")
 fields\_data = {k: v for k, v in obj\_data.items() if k not in {"\_\_model\_\_", "id"} and k not in exclude\_fields} # séparer file fields
 file\_payloads = {} for field in Model.\_meta.fields:
 if isinstance(field, (FileField, ImageField)) and field.name in fields\_data:
 file\_payloads[field.name] = fields\_data.pop(field.name)
 instance = None # 1) update par ID si possible (uniquement si l'objet existe déjà) if keep\_ids and obj\_id is not None:
 instance = Model.objects.filter(pk=obj\_id).first() # 2) sinon match intelligent if instance is None:
 instance = \_get\_instance\_by\_smart\_keys(Model, model\_label, fields\_data) # 3) sinon create if instance is None:
 if keep\_ids and obj\_id is not None:
 instance = Model(pk=obj\_id)
 else:
 instance = Model()
 # appliquer champs simples for k, v in fields\_data.items():
 # Normaliser à l'enregistrement pour certains champs texte (optionnel mais utile) if model\_label == "app\_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
 v = v.strip()
 setattr(instance, k, v) # restaurer fichiers for field\_name, payload in file\_payloads.items():
 \_restore\_file\_field(instance, field\_name, payload)
 instance.save() @transaction.atomic def import\_json\_bundle(request):
 """
 POST depuis page\_administration.html Inputs:
 - json\_portfolio (optionnel)
 - json\_users (optionnel)
 - replace\_portfolio (checkbox)
 - replace\_users (checkbox)
 """
 if not \_is\_admin\_session(request):
 return HttpResponseForbidden("Accès refusé (Admin uniquement.)")
 if request.method != "POST":
 return redirect("dashboard\_admin")
 json\_portfolio = request.FILES.get("json\_portfolio")
 json\_users = request.FILES.get("json\_users")
 replace\_portfolio = bool(request.POST.get("replace\_portfolio"))
 replace\_users = bool(request.POST.get("replace\_users"))
 # ----- Portfolio import -----
 if json\_portfolio:
 try:
 data = \_parse\_uploaded\_json(json\_portfolio)
 if data.get("export\_type") != "portfolio":
 raise ValueError("Ce fichier n'est pas un export portfolio.")
 model\_map = data.get("data", {})
 if replace\_portfolio:
 from app\_acceuil.models import (
 Projetphotodeprofil,
 ProjetAproposDeMoi,
 Projetsfirstspeciality,
 Projetmesrealisations,
 Projetscards,
 MesCompetencesCles,
 ReseauSocial,
 )
 # Ordre "simple"
 ReseauSocial.objects.all().delete()
 MesCompetencesCles.objects.all().delete()
 Projetscards.objects.all().delete()
 Projetmesrealisations.objects.all().delete()
 Projetsfirstspeciality.objects.all().delete()
 ProjetAproposDeMoi.objects.all().delete()
 Projetphotodeprofil.objects.all().delete()
 for model\_label, objects\_list in model\_map.items():
 \_upsert\_objects(model\_label, objects\_list, keep\_ids=True)
 messages.success(request, "Import Portfolio terminé avec succès.")
 except Exception as e:
 messages.error(request, f"Erreur import portfolio: {e}")
 # ----- Users import -----
 if json\_users:
 try:
 data = \_parse\_uploaded\_json(json\_users)
 if data.get("export\_type") != "users":
 raise ValueError("Ce fichier n'est pas un export users.")
 model\_map = data.get("data", {})
 if replace\_users:
 # Ordre important : dépendances FK
 from app\_user.models import RoleMenuAcces, Menu
 from app\_contact.models import User, Role, Contact
 RoleMenuAcces.objects.all().delete()
 User.objects.all().delete()
 Contact.objects.all().delete()
 Menu.objects.all().delete()
 Role.objects.all().delete()
 for model\_label in ordered:
 if model\_label in model\_map:
 \_upsert\_objects(model\_label, model\_map[model\_label], keep\_ids=True)
 messages.success(request, "Import Users terminé avec succès.")
 except Exception as e:
 messages.error(request, f"Erreur import users: {e}")
 return redirect("dashboard\_admin")

```
model_map[model_label], keep_ids=True) messages.success(request, "Import Utilisateurs terminé avec succès.") except Exception as e:  
messages.error(request, f"Erreur import utilisateurs: {e}") return redirect("dashboard_admin")
```

Je vois bien ton fichier  — globalement, il est **cohérent** et il fera exactement ce que tu veux : **ajouter si pas trouvé / mettre à jour si trouvé, sans suppression** (sauf si tu coches “Remplacer …”).

Il y a juste quelques points importants à corriger/améliorer pour que ta “fusion intelligente” soit **fiable** et **ne crée pas de doublons** dans certains cas.

---

## 1) Point critique : la normalisation “lookup” ne marche pas si la base n'est pas normalisée

Tu normalises la valeur importée (ex: email en lowercase), puis tu fais :

```
Model.objects.filter(**lookup).first()
```

Mais si dans la base il existe un email stocké en **majuscules** (Mario@Gmail.com), le filter(email="mario@gmail.com") **ne le trouvera pas**, et tu créerais un doublon.

 Solution simple : **normaliser aussi au moment d'enregistrer** les champs clés (email, nom\_role, code\_menu, etc.).  
Tu l'as déjà fait pour Projetscards (strip), mais pas pour les autres.

**Ajoute ce bloc dans `_upsert_objects` (dans la boucle `for k,v in fields_data.items()`), juste avant `setattr(...)` :**

```
# Normalisation à l'enregistrement pour éviter les doublons à l'import  
if model_label == "app_contact.User" and k == "email" and isinstance(v, str):  
    v = _norm_email(v)  
  
if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):  
    v = _norm_text(v)  
  
if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):  
    v = _norm_text(v)  
  
if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):  
    v = _norm_text(v)  
  
if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):  
    v = _norm_text(v)
```

```
# Projetscards: tu l'as déjà (strip), mais tu peux aussi normaliser plus:  
if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):  
    v = v.strip()
```

👉 Ça rend ton système **stable** : même si la DB contient des valeurs “sales”, le prochain import les normalise et évite les doublons.

---

## 2) Projetscards : ton matching tolérant fonctionne, mais il peut être “lourd”

Tu fais un “scan” :

```
candidates = Model.objects.all()  
...  
for obj in candidates:
```

C'est OK tant que tu n'as pas des milliers de témoignages.

Si tu veux optimiser sans compliquer, tu peux réduire les candidats :

### Remplace

```
candidates = Model.objects.all()  
if "nom" in lookup:  
    candidates = candidates.filter(nom__isnull=False)
```

par :

```
candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="")
```

(et on garde la comparaison normalisée en Python).

---

## 3) Nettoyage conseillé (pas obligatoire)

Tu as mis `import re` au milieu du fichier. Ça marche, mais pour être propre :

- mets `import re` en haut avec les autres imports.
- 

## 4) Ta logique répond exactement à ta question “import petit à petit”

En l'état (sans synchroniser) :

- Si `id` existe → update
- Si `id` n'existe pas → tu tentes une recherche “clé métier” (`SMART_MATCH_KEYS`)
  - si trouvé → update
  - sinon → create
- aucune suppression
- “Remplacer …” = seul cas où tu supprimes avant import

Donc oui : tu peux alimenter ta base **uniquement via import JSON**, petit à petit, sans clavier.

---

## 5) Mini test rapide conseillé

Après avoir ajouté la normalisation à l'enregistrement :

1. Crée un user avec email `Mario@Gmail.com` (ancien)
2. Importe un JSON avec email `mario@gmail.com`
  - il doit être **mis à jour**, pas dupliqué.

---

Si tu veux, je peux te renvoyer **ton fichier entier “propre”** (mêmes fonctions, juste réorganisé + normalisation ajoutée), mais déjà avec les blocs ci-dessus tu es bon.

oui

Réflexion durant 3m 21s

Oui — je te renvoie le fichier complet `json_transfer.py` (version “fusion intelligente tolérante”) sans la 3e option “Synchroniser”, avec :

- Export JSON (texte + images Base64)
- Import JSON
- “Remplacer ...” (optionnel) = purge + import
- **Fusion intelligente** si l’ID ne matche pas (match par clés métier)
- **Tolérance** aux espaces/retours ligne/majuscules
- **Normalisation à l’enregistrement** (très important pour éviter les doublons lors des futurs imports)

Copie/colle ce fichier tel quel en remplacement de ton `json_transfer.py`.

```
import base64
import json
import mimetypes
import re
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    """
    Basé sur ta logique de session :
    request.session['user'] = { id, nom, role, ... }
    """
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
```

```

"""
Rend une valeur sérialisable JSON.
"""
if value is None:
    return None

if isinstance(value, (datetime, date, time)):
    return value.isoformat()

if isinstance(value, Decimal):
    return str(value)

if isinstance(value, UUID):
    return str(value)

return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    """
    Construit une réponse HTTP de téléchargement JSON.
    default=str => sécurité si un type inconnu traîne.
    """
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    """
    Convertit un objet Django en dict JSON-friendly.
    - Champs simples: str/int/bool/... => convertis via _json_compatible_value
    - Image/File: export base64 + meta
    - ForeignKey: la valeur brute (souvent l'id) est OK (JSON)
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name

```

```

if name == "id":
    continue

value = getattr(obj, name)

# Fichiers / images
if isinstance(field, (FileField, ImageField)):
    if value and getattr(value, "name", None):
        file_path = value.name
        b64 = None

    try:
        with default_storage.open(file_path, "rb") as f:
            raw = f.read()
        b64 = base64.b64encode(raw).decode("utf-8")
    except Exception:
        b64 = None

    mime, _ = mimetypes.guess_type(file_path)
    data[name] = {
        "storage_path": file_path,
        "filename": file_path.split("/")[-1],
        "mimetype": mime or "application/octet-stream",
        "base64": b64,
    }
else:
    data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data


def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

# Import local des modèles portfolio
from app_acceuil.models import (

```

```

        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetmesrealisations,
        Projetscards,
        MesCompetencesCles,
        ReseauSocial,
    )

payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_portfolio.json"
return _safe_json_response(payload, filename)

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "notes": [
        "User.password est exporté tel quel (hash).",
    ],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
    }
}

```

```
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_users.json"
return _safe_json_response(payload, filename)

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    """
    file_dict = {"filename", "base64", "mimetype", "storage_path", ...}
    => écrit via ImageField/FileField.save()
    """
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)
```

```

# -----
# FUSION INTELLIGENTE : clés métier par modèle
# (si id absent ou différent, on tente de retrouver un objet existant)
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],                      # souvent unique
    "app_acceuil.ReseauSocial": ["nom"],                                # ex: email / whatsapp / linkedin
    "app_acceuil.Projetmesrealisations": ["nom_projet"],             # stable si tu gardes les mêmes noms

    # Témoignages: nom pas unique => nom + description
    "app_acceuil.Projetscards": ["nom", "description"],

    # Users
    "app_contact.Role": ["nom_role"],                                  # unique
    "app_contact.User": ["email"],                                     # unique
    "app_user.Menu": ["code_menu"],                                    # unique
    "app_user.RoleMenuAcces": ["role", "menu"],                         # clé composée
}

# -----
# Normalisation (tolérance)
# -----
def norm_text(s: str) -> str:
    """Trim + espaces multiples + lowercase."""
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # remplace \n, \t, espaces multiples
    return s.lower()

def _norm_email(s: str) -> str:
    return norm_text(s)

def _normalized_lookup_value(model_label: str, field_name: str, value):
    """
    Normalisation par modèle/champ.
    On ne change pas les ints (FK), bool, etc.
    """
    if value is None:
        return None

    if model_label == "app_contact.User" and field_name == "email":
        return _norm_email(str(value))

```

```

if model_label == "app_acceuil.Projetscards" and field_name in ("nom", "description"):
    return _norm_text(str(value))

if model_label == "app_acceuil.MesCompetencesCles" and field_name == "nom":
    return _norm_text(str(value))

if model_label == "app_acceuil.ReseauSocial" and field_name == "nom":
    return _norm_text(str(value))

if model_label == "app_contact.Role" and field_name == "nom_role":
    return _norm_text(str(value))

if model_label == "app_user.Menu" and field_name == "code_menu":
    return _norm_text(str(value))

return value


def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None

        raw_val = fields_data[k]
        val = _normalized_lookup_value(model_label, k, raw_val)

        if val is None or (isinstance(val, str) and val.strip() == ""):
            return None

        lookup[k] = val

    # Projetscards : matching tolérant (normalisation DB faite en Python)
    if model_label == "app_acceuil.Projetscards":
        candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="")
        for obj in candidates:
            nom_db = _norm_text(getattr(obj, "nom", ""))
            desc_db = _norm_text(getattr(obj, "description", ""))
            if nom_db == lookup.get("nom") and desc_db == lookup.get("description"):
                return obj
    return None

```

```

# Autres : filter direct OK si tu normalises aussi à l'enregistrement (voir _upsert_objects)
try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    UPSERT intelligent (sans suppression) :
    1) Si id existe en base -> update
    2) Sinon -> tentative match via SMART_MATCH_KEYS (tolérant)
    3) Sinon -> create
    """
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        instance = None

        # 1) update par ID si possible (uniquement si l'objet existe déjà)
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first()

        # 2) sinon match intelligent
        if instance is None:
            instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

        # 3) sinon create
        if instance is None:
            if keep_ids and obj_id is not None:
                instance = Model(pk=obj_id)

```

```

else:
    instance = Model()

# appliquer champs simples + normalisation à l'enregistrement (anti-doublons)
for k, v in fields_data.items():

    # Normalisation à l'enregistrement pour éviter les doublons aux imports suivants
    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)

    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
        # ici on garde strip (et non _norm_text) pour ne pas altérer le contenu visible,
        # mais ça évite déjà beaucoup de doublons "bêtes".
        v = v.strip()

    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    """
    POST depuis page_administration.html
    Inputs:
    - json_portfolio (optionnel)
    - json_users (optionnel)
    - replace_portfolio (checkbox)
    - replace_users (checkbox)

```

```
Sans synchroniser :
- si replace_* est coché : purge + import
- sinon : upsert intelligent + AUCUNE suppression
"""
if not _is_admin_session(request):
    return HttpResponseRedirect("Accès refusé (Admin uniquement).")

if request.method != "POST":
    return redirect("dashboard_admin")

json_portfolio = request.FILES.get("json_portfolio")
json_users = request.FILES.get("json_users")
replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
    except Exception as e:
        messages.error(request, str(e))

    if replace_portfolio:
        from app_acceuil.models import (
            Projetphotodeprofil,
            ProjetAproposDeMoi,
            Projetsfirstspeciality,
            Projetmesrealisations,
            Projetcards,
            MesCompetencesCles,
            ReseauSocial,
        )
        # Ordre "simple"
        ReseauSocial.objects.all().delete()
        MesCompetencesCles.objects.all().delete()
        Projetcards.objects.all().delete()
        Projetmesrealisations.objects.all().delete()
        Projetsfirstspeciality.objects.all().delete()
        ProjetAproposDeMoi.objects.all().delete()
        Projetphotodeprofil.objects.all().delete()

        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")
```

```

except Exception as e:
    messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})

        if replace_users:
            # Ordre important : dépendances FK
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
            Role.objects.all().delete()

        # Ordre d'import conseillé
        ordered = [
            "app_contact.Role",
            "app_user.Menu",
            "app_contact.User",
            "app_contact.Contact",
            "app_user.RoleMenuAcces",
        ]
        for model_label in ordered:
            if model_label in model_map:
                _upsert_objects(model_label, model_map[model_label], keep_ids=True)

        messages.success(request, "Import Utilisateurs terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

## Deux remarques utiles

1. **Tu peux importer petit à petit sans suppression** : si tu ne coches pas “Remplacer”, ce fichier fait bien **update/insert uniquement**, jamais de delete.

2. Pour Projetscards, on matche avec nom + description en **comparaison normalisée**, mais on n'altère pas le contenu affiché (on fait juste strip() à l'enregistrement).

Si tu veux, je peux aussi te donner une option “anti-doublon dur” pour Projetscards basée sur un **hash (fingerprint)** (beaucoup plus rapide et fiable si tu as beaucoup de témoignages).

Si tu veux, je peux aussi te donner une option “anti-doublon dur” pour Projetscards basée sur un hash (fingerprint) (beaucoup plus rapide et fiable si tu as beaucoup de témoignages).

Oui — l’option “anti-doublon dur” la plus fiable consiste à créer une **empreinte (fingerprint)** stable pour chaque témoignage Projetscards, puis à s’en servir comme **clé métier unique**.

L’idée :

- on calcule un hash à partir de champs “métier” (ex: nom + description)
- on l’enregistre dans la base (champ fingerprint)
- à l’import : si id ne matche pas, on cherche d’abord par fingerprint → **mise à jour** au lieu de créer un doublon

Ça devient rapide et robuste même avec beaucoup de lignes.

---

## 1) Modèle : ajouter un champ `fingerprint` à Projetscards

Dans app\_acceuil/models.py (classe Projetscards), ajoute :

```
from django.db import models

class Projetscards(models.Model):
    # ... tes champs existants ...
    fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

Puis fais :

```
python manage.py makemigrations
python manage.py migrate
```

`unique=True` = “anti-doublon dur” (la base empêchera les doublons même si tu te trompes).

## 2) Calcul du fingerprint : fonction utilitaire

Dans json\_transfer.py, ajoute ces imports en haut :

```
import hashlib
```

Et ajoute ces fonctions (à côté de tes normalisations) :

```
def _fingerprint_projetscards(nom: str, description: str) -> str:  
    """  
        Fingerprint stable basé sur une normalisation forte.  
        - trim  
        - espaces multiples  
        - lowercase  
    """  
    base = f"({_norm_text(nom)}||{_norm_text(description)})"  
    return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

## 3) Import : calculer et utiliser `fingerprint` automatiquement

### A) Modifie `SMART_MATCH_KEYS` pour Projetscards

Remplace :

```
"app_acceuil.Projetscards": ["nom", "description"],
```

par :

```
"app_acceuil.Projetscards": ["fingerprint"],
```

### B) Avant de chercher l'instance, injecte `fingerprint` si absent

Dans `_upsert_objects(...)`, juste après avoir construit `fields_data` et retiré les images (donc `avant_instance = None`), ajoute :

```
# Auto-calcule fingerprint pour Projetscards si pas fourni
```

```
if model_label == "app_acceuil.Projetscards":  
    fp = fields_data.get("fingerprint")  
    if not fp:  
        nom = fields_data.get("nom", "")  
        description = fields_data.get("description", "")  
        fields_data["fingerprint"] = _fingerprint_projetscards(nom, description)
```

 Résultat : même si ton JSON ne contient pas `fingerprint`, il sera calculé à l'import.

### C) Normaliser/mettre à jour `fingerprint` à chaque import (recommandé)

Toujours dans la boucle `for k, v in fields_data.items():`, ajoute :

```
if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):  
    v = v.strip().lower()
```

---

## 4) Export : inclure `fingerprint` automatiquement

Ton export sérialise tous les champs du modèle, donc le **champ `fingerprint` sera exporté automatiquement** dès qu'il existe dans la DB.

Mais pour les anciens témoignages déjà en base (avant migration), `fingerprint` sera vide.

 Il faut remplir les fingerprints une fois.

### Script simple à lancer (dans Django shell)

```
python manage.py shell
```

Puis :

```
from app_acceuil.models import Projetscards  
import hashlib, re  
  
def norm_text(s):  
    s = (s or "").strip()  
    s = re.sub(r"\s+", " ", s)  
    return s.lower()  
  
def fp(nom, description):  
    base = f"{norm_text(nom)} || {norm_text(description)}"
```

```
return hashlib.sha256(base.encode("utf-8")).hexdigest()

for o in Projetscards.objects.all():
    if not o.fingerprint:
        o.fingerprint = fp(o.nom, o.description)
        o.save()
print("OK")
```

---

## 5) Pourquoi c'est “dur” et meilleur que (nom+description)

- nom+description en recherche = peut devenir lent (scan Python) et sensible à micro-différences
  - fingerprint + unique index = **recherche immédiate + blocage DB** des doublons
- 

Si tu veux, je peux te donner aussi une version encore plus “métier” du fingerprint (ex: inclure une date, un poste, ou autre champ pour distinguer deux témoignages très proches).

code complet

Voici le **code complet** pour l’option “anti-doublon dur” avec **fingerprint (hash)** pour `Projetscards`, **sans** la 3e option “Synchroniser”, et en gardant ton mode :

- si `id` existe → **update**
  - sinon → **match par clés métier** (dont `fingerprint` pour `Projetscards`)
  - sinon → **create**
  - aucune suppression (sauf si tu coches “Remplacer ...”)
- 

## 1) `app_acceuil/models.py` — ajout du champ `fingerprint` dans `Projetscards`

Dans le modèle `Projetscards`, ajoute ce champ :

```
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

👉 Exemple (à intégrer dans ta classe existante, sans toucher aux autres champs) :

```
class Projetscards(models.Model):
    # ... tes champs existants ...
    fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

Puis exécute :

```
python manage.py makemigrations
python manage.py migrate
```

---

## 2) json\_transfer.py — VERSION COMPLÈTE (avec fingerprint)

Remplace ton fichier json\_transfer.py par ceci :

```
import base64
import json
import mimetypes
import re
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    """
        Basé sur ta logique de session :
        request.session['user'] = { id, nom, role, ... }
    """
```

```
"""
user_data = request.session.get("user")
return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation (tolérance)
# -----
def _norm_text(s: str) -> str:
    """Trim + espaces multiples + lowercase."""
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    """
    Fingerprint stable des témoignages.
    Normalisation forte => stable malgré espaces/retours ligne/majuscules.
    """
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

```
# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue

        value = getattr(obj, name)

        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name
                b64 = None
                try:
                    with default_storage.open(file_path, "rb") as f:
                        raw = f.read()
                    b64 = base64.b64encode(raw).decode("utf-8")
                except Exception:
                    b64 = None

                mime, _ = mimetypes.guess_type(file_path)
                data[name] = {
                    "storage_path": file_path,
                    "filename": file_path.split("/")[-1],
                    "mimetype": mime or "application/octet-stream",
                    "base64": b64,
                }
            else:
                data[name] = None
        else:
            data[name] = _json_compatible_value(value)

    return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]
```

```

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_acceuil.models import (
        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetsmesrealisations,
        Projetcards,
        MesCompetencesCles,
        ReseauSocial,
    )

    payload = {
        "export_type": "portfolio",
        "exported_at": datetime.now().isoformat(),
        "version": 1,
        "data": {
            "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
            "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
            "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
            "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetsmesrealisations.objects.all()),
            "app_acceuil.Projetcards": _serialize_queryset(Projetcards.objects.all()),
            "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
            "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    filename = f"{date_prefix}_export_portfolio.json"
    return _safe_json_response(payload, filename)

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_contact.models import User, Role, Contact
    from app_user.models import Menu, RoleMenuAcces

    payload = {

```

```
"export_type": "users",
"exported_at": datetime.now().isoformat(),
"version": 1,
"notes": ["User.password est exporté tel quel (hash)."],
"data": {
    "app_contact.Role": _serialize_queryset(Role.objects.all()),
    "app_contact.User": _serialize_queryset(User.objects.all()),
    "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
    "app_user.Menu": _serialize_queryset(Menu.objects.all()),
    "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
},
}

date_prefix = datetime.now().strftime("%Y%m%d")
filename = f"{date_prefix}_export_users.json"
return _safe_json_response(payload, filename)

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return
    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return
    try:
        raw = base64.b64decode(b64)
    except Exception:
        return
    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)
```

```

# -----
# FUSION INTELLIGENTE : clés métier par modèle
# -----
SMART_MATCH_KEYS = {
    # Portfolio (exemples)
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"],

    # Témoignages: MATCH PAR fingerprint (anti-doublon dur)
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"],
}
}

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None
        lookup[k] = v

    try:
        return Model.objects.filter(**lookup).first()
    except Exception:
        return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    """
    UPSERT intelligent (sans suppression):
    1) ID existant -> update
    2) sinon clés métier -> update
    3) sinon create
    """

```

```

"""
Model = apps.get_model(model_label)
exclude_fields = exclude_fields or set()

for obj_data in objects_list:
    obj_id = obj_data.get("id")

    fields_data = {
        k: v for k, v in obj_data.items()
        if k not in {"__model__", "id"} and k not in exclude_fields
    }

    # séparer file fields
    file_payloads = {}
    for field in Model._meta.fields:
        if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
            file_payloads[field.name] = fields_data.pop(field.name)

    # --- Projetscards: auto-calc fingerprint si absent ---
    if model_label == "app_acceuil.Projetscards":
        fp = fields_data.get("fingerprint")
        if not fp:
            nom = fields_data.get("nom", "")
            description = fields_data.get("description", "")
            fields_data["fingerprint"] = _fingerprint_projetscards(nom, description)

    instance = None

    # 1) update par ID si existant
    if keep_ids and obj_id is not None:
        instance = Model.objects.filter(pk=obj_id).first()

    # 2) match clés métier (fingerprint/email/...)
    if instance is None:
        instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

    # 3) create
    if instance is None:
        if keep_ids and obj_id is not None:
            instance = Model(pk=obj_id)
        else:
            instance = Model()

    # appliquer champs + normalisation à l'enregistrement (anti-doublons)
    for k, v in fields_data.items():

```

```

# normalisations (recommandées)
if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
    v = _norm_email(v)

if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
    v = _norm_text(v)

if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
    v = _norm_text(v)

if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
    v = _norm_text(v)

if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
    v = _norm_text(v)

# fingerprint: toujours lower/trim
if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
    v = v.strip().lower()

# contenu visible: juste strip (pour éviter pollution)
if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
    v = v.strip()

setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    """
    Sans synchroniser:
    - si replace_* coché : purge + import
    - sinon : upsert intelligent + AUCUNE suppression
    """
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

```

```

json_portfolio = request.FILES.get("json_portfolio")
json_users = request.FILES.get("json_users")
replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

    if replace_portfolio:
        from app_acceuil.models import (
            Projetphotodeprofil,
            ProjetAproposDeMoi,
            Projetsfirstspeciality,
            Projetsrealisations,
            Projetcards,
            MesCompetencesCles,
            ReseauSocial,
        )
        ReseauSocial.objects.all().delete()
        MesCompetencesCles.objects.all().delete()
        Projetcards.objects.all().delete()
        Projetsrealisations.objects.all().delete()
        Projetsfirstspeciality.objects.all().delete()
        ProjetAproposDeMoi.objects.all().delete()
        Projetphotodeprofil.objects.all().delete()

        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
    except Exception as e:
        messages.error(request, f"Erreur import users: {e}")

```

```

if replace_users:
    from app_user.models import RoleMenuAcces, Menu
    from app_contact.models import User, Role, Contact

    RoleMenuAcces.objects.all().delete()
    User.objects.all().delete()
    Contact.objects.all().delete()
    Menu.objects.all().delete()
    Role.objects.all().delete()

    ordered = [
        "app_contact.Role",
        "app_user.Menu",
        "app_contact.User",
        "app_contact.Contact",
        "app_user.RoleMenuAcces",
    ]
    for model_label in ordered:
        if model_label in model_map:
            _upsert_objects(model_label, model_map[model_label], keep_ids=True)

    messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

---

### 3) Remplir `fingerprint` pour les témoignages déjà existants (1 fois)

Après migration, exécute dans le shell Django :

```
python manage.py shell
```

Puis colle :

```

from app_acceuil.models import Projetscards
import re, hashlib

def norm_text(s):
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)

```

```
return s.lower()

def fp(nom, description):
    base = f"{norm_text(nom)}||{norm_text(description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

count = 0
for o in Projetscards.objects.all():
    if not o.fingerprint:
        o.fingerprint = fp(o.nom, o.description)
        o.save()
        count += 1

print("Fingerprints créés:", count)
```

 Après ça :

- tes exports incluront `fingerprint`
  - tes imports matcheront d'abord par `id`, sinon par `fingerprint`
  - la DB empêchera **réellement** les doublons grâce à `unique=True`
- 

Si tu veux, je peux aussi te donner une **migration Django automatique** qui remplit `fingerprint` au moment du `migrate` (au lieu d'utiliser le shell).

oui

Ok — je te donne une **migration Django complète** qui :

1. **ajoute** le champ `fingerprint` à `Projetscards` (en `null=True` au début, pour éviter les conflits)
2. **remplit** automatiquement `fingerprint` pour toutes les lignes existantes
3. **verrouille** le champ (passage en `null=False + unique=True`)

 Important : je ne connais pas le numéro exact de ta dernière migration (ex: `0007_...`). Tu devras juste remplacer la dépendance (`"app_acceuil", "000X_dernier_fichier"`) par **ton dernier fichier de migration**.

---

## Étape 1 — Mets le champ dans `models.py`

Dans `app_acceuil/models.py`, dans `Projetscards`, ajoute :

```
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

(Tu peux le mettre à la fin de la classe.)

---

## Étape 2 — Crée UNE migration manuelle (recommandé)

Crée un fichier (exemple) :

```
app_acceuil/migrations/000X_projetscards_fingerprint.py
```

Colle ce code **complet** :

```
from django.db import migrations, models
import hashlib
import re

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _fp(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def backfill_fingerprint(apps, schema_editor):
    Projetscards = apps.get_model("app_acceuil", "Projetscards")

    for obj in Projetscards.objects.all().iterator():
        # si déjà rempli, on touche pas
```

```
if getattr(obj, "fingerprint", None):
    continue

nom = getattr(obj, "nom", "") or ""
description = getattr(obj, "description", "") or ""

obj.fingerprint = _fp(nom, description)
obj.save(update_fields=["fingerprint"])


class Migration(migrations.Migration):

    dependencies = [
        # ⚠ Remplace 000X_dernier_fichier par le nom de ta dernière migration app_acceuil
        ("app_acceuil", "000X_dernier_fichier"),
    ]

    operations = [
        # 1) Ajout du champ en NULLABLE au début (évite conflit de unique)
        migrations.AddField(
            model_name="projetscards",
            name="fingerprint",
            field=models.CharField(
                max_length=64,
                unique=True,
                db_index=True,
                null=True,
                blank=True,
            ),
        ),
        # 2) Remplissage des lignes existantes
        migrations.RunPython(backfill_fingerprint, migrations.RunPython.noop),
        # 3) Verrouillage : NOT NULL + default "" (le champ sera toujours rempli)
        migrations.AlterField(
            model_name="projetscards",
            name="fingerprint",
            field=models.CharField(
                max_length=64,
                unique=True,
                db_index=True,
                null=False,
                blank=True,
                default="",
            ),
        )
    ]
```

```
) ,  
) ,  
]

---


```

## Étape 3 — Applique la migration

Dans ton terminal :

```
python manage.py makemigrations
```

 Si Django te génère déjà une migration “AddField fingerprint”, **supprime-la** (ou ne la garde pas) et garde **celle ci-dessus** (car elle inclut le backfill).  
*(Ou alors dis-moi le nom généré et je te dis quoi faire.)*

Puis :

```
python manage.py migrate
```

---

## Étape 4 — Mets à jour json\_transfer.py

Tu gardes le json\_transfer.py “fingerprint” que je t’ai donné :

- il calcule `fingerprint` si absent au moment de l’import
  - et il matche `Projetscards` par `fingerprint` (anti-doublon dur)
- 

## Petit contrôle

Après migration, dans Django shell :

```
python manage.py shell  
from app_acceuil.models import Projetscards  
Projetscards.objects.filter(fingerprint="").count()
```

→ doit être 0 (ou très proche de 0 si certains témoignages ont nom+description vides).

---

Si tu me copies/colles la liste de tes migrations `app_acceuil/migrations/` (juste les noms des fichiers), je te renvoie la dépendance (`"app_acceuil", "..."`) **exacte** sans que tu aies à deviner.

Est ce que le code suivant est t il quand même suffisant et sûr si je décide de garder le code suivant : import base64 import json import mimetypes import re from datetime import datetime, date, time from decimal import Decimal from uuid import UUID from django.apps import apps from django.contrib import messages from django.core.files.base import ContentFile from django.core.files.storage import default\_storage from django.db import transaction from django.db.models.fields.files import FileField, ImageField from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden from django.shortcuts import redirect # ----- # Helpers : sécurité + JSON # ----- def \_is\_admin\_session(request) -> bool: """ Basé sur ta logique de session : request.session['user'] = { id, nom, role, ... } """ user\_data = request.session.get("user") return bool(user\_data and user\_data.get("role") == "Administrateur") def \_json\_compatible\_value(value): """ Rend une valeur sérialisable JSON. """ if value is None: return None if isinstance(value, (datetime, date, time)): return value.isoformat() if isinstance(value, Decimal): return str(value) if isinstance(value, UUID): return str(value) return value def \_safe\_json\_response(payload: dict, filename: str) -> HttpResponseRedirect: """ Construit une réponse HTTP de téléchargement JSON. default=str => sécurité si un type inconnu traîne. """ content = json.dumps(payload, ensure\_ascii=False, indent=2, default=str) resp = HttpResponseRedirect(content, content\_type="application/json; charset=utf-8") resp["Content-Disposition"] = f'attachment; filename="{filename}"' return resp # ----- # Serialize : Django -> dict # ----- def \_serialize\_instance(obj): """ Convertit un objet Django en dict JSON-friendly. - Champs simples: str/int/bool/... => convertis via \_json\_compatible\_value - Image/File: export base64 + meta - ForeignKey: la valeur brute (souvent l'id) est OK (JSON) """ data = { "\_\_model\_\_": f'{obj.\_meta.app\_label}.{obj.\_\_class\_\_.\_\_name\_\_}', "id": obj.pk, } for field in obj.\_meta.fields: name = field.name if name == "id": continue value = getattr(obj, name) # Fichiers / images if isinstance(field, (FileField, ImageField)): if value and getattr(value, "name", None): file\_path = value.name b64 = None try: with default\_storage.open(file\_path, "rb") as f: raw = f.read() b64 = base64.b64encode(raw).decode("utf-8") except Exception: b64 = None mime, \_ = mimetypes.guess\_type(file\_path) data[name] = { "storage\_path": file\_path, "filename": file\_path.split("/")[-1], "mimetype": mime or "application/octet-stream", "base64": b64, } else: data[name] = None else: data[name] = \_json\_compatible\_value(value) return data def \_serialize\_queryset(qs): return [\_serialize\_instance(o) for o in qs] # ----- # EXPORTS # ----- def export\_portfolio\_json(request): if not \_is\_admin\_session(request): return HttpResponseRedirectForbidden("Accès refusé (Admin uniquement).") # Import local des modèles portfolio from app\_acceuil.models import (Projetphotodeprofil, ProjetAproposDeMoi, Projetsfirstspeciality, Projetmesrealisations, Projetcards, MesCompetencesCles, ReseauSocial,) payload = { "export\_type": "portfolio", "exported\_at": datetime.now().isoformat(), "version": 1, "data": { "app\_acceuil.Projetphotodeprofil": \_serialize\_queryset(Projetphotodeprofil.objects.all()), "app\_acceuil.ProjetAproposDeMoi": \_serialize\_queryset(ProjetAproposDeMoi.objects.all()), "app\_acceuil.Projetsfirstspeciality": \_serialize\_queryset(Projetsfirstspeciality.objects.all()), "app\_acceuil.Projetmesrealisations": \_serialize\_queryset(Projetmesrealisations.objects.all()), "app\_acceuil.Projetcards": \_serialize\_queryset(Projetcards.objects.all()), "app\_acceuil.MesCompetencesCles": \_serialize\_queryset(MesCompetencesCles.objects.all()), "app\_acceuil.ReseauSocial": \_serialize\_queryset(ReseauSocial.objects.all()), } } date\_prefix = datetime.now().strftime("%Y%om%d") filename = f'{date\_prefix}\_export\_portfolio.json' return \_safe\_json\_response(payload, filename) def export\_users\_json(request): if not \_is\_admin\_session(request): return HttpResponseRedirectForbidden("Accès refusé (Admin uniquement).") from app\_contact.models import User, Role, Contact from app\_user.models import Menu, RoleMenuAcces payload = { "export\_type": "users", "exported\_at": datetime.now().isoformat(), "version": 1, "notes": [ "User.password est exporté tel quel (hash).", ], "data": { "app\_contact.Role": \_serialize\_queryset(Role.objects.all()), "app\_contact.Contact": \_serialize\_queryset(Contact.objects.all()), "app\_contact.User": \_serialize\_queryset(User.objects.all()), "app\_user.Menu": \_serialize\_queryset(Menu.objects.all()), "app\_user.RoleMenuAcces": \_serialize\_queryset(RoleMenuAcces.objects.all()), } } date\_prefix = datetime.now().strftime("%Y%om%d") filename = f'{date\_prefix}\_export\_users.json' return \_safe\_json\_response(payload, filename)

```

_serialize_queryset(Role.objects.all()), "app_contact.User": _serialize_queryset(User.objects.all()), "app_contact.Contact":
_serialize_queryset(Contact.objects.all()), "app_user.Menu": _serialize_queryset(Menu.objects.all()), "app_user.RoleMenuAcces":
_serialize_queryset(RoleMenuAcces.objects.all()), }, } date_prefix = datetime.now().strftime("%Y%m%d") filename = f'{date_prefix}_export_users.json'
return _safe_json_response(payload, filename) # ----- # IMPORTS # ----- def _parse_uploaded_json(uploaded_file): raw
= uploaded_file.read() try: text = raw.decode("utf-8") except Exception: text = raw.decode("latin-1") return json.loads(text) def
_restore_file_field(model_instance, field_name: str, file_dict: dict): """ file_dict = {filename, base64, mimetype, storage_path, ...} => écrit via
ImageField/FileField.save() """ if not file_dict: return b64 = file_dict.get("base64") filename = file_dict.get("filename") or "file.bin" if not b64: return try: raw
= base64.b64decode(b64) except Exception: return django_file = ContentFile(raw, name=filename) field = getattr(model_instance, field_name)
field.save(filename, django_file, save=False) # ----- # FUSION INTELLIGENTE : clés métier par modèle # (si id absent
ou différent, on tente de retrouver un objet existant) # ----- SMART_MATCH_KEYS = { # Portfolio
"app_acceuil.MesCompetencesCles": ["nom"], # souvent unique "app_acceuil.ReseauSocial": ["nom"], # ex: email / whatsapp / linkedin
"app_acceuil.Projetmesrealisations": ["nom_projet"], # stable si tu gardes les mêmes noms # Témoignages: nom pas unique => nom + description
"app_acceuil.Projetscards": ["nom", "description"], # Users "app_contact.Role": ["nom_role"], # unique "app_contact.User": ["email"], # unique
"app_user.Menu": ["code_menu"], # unique "app_user.RoleMenuAcces": ["role", "menu"], # clé composée } # ----- # Normalisation
(tolérance) # ----- def _norm_text(s: str) -> str: """Trim + espaces multiples + lowercase.""" s = (s or "").strip() s = re.sub(r"\s+", " ", s) #
remplace \n, \t, espaces multiples return s.lower() def _norm_email(s: str) -> str: return _norm_text(s) def _normalized_lookup_value(model_label: str,
field_name: str, value): """ Normalisation par modèle/champ. On ne change pas les ints (FK), bool, etc. """ if value is None: return None if model_label ==
"app_contact.User" and field_name == "email": return _norm_email(str(value)) if model_label == "app_acceuil.Projetscards" and field_name in ("nom",
"description"): return _norm_text(str(value)) if model_label == "app_acceuil.MesCompetencesCles" and field_name == "nom": return _norm_text(str(value)) if
model_label == "app_acceuil.ReseauSocial" and field_name == "nom": return _norm_text(str(value)) if model_label == "app_contact.Role" and field_name ==
"nom_role": return _norm_text(str(value)) if model_label == "app_user.Menu" and field_name == "code_menu": return _norm_text(str(value)) return value def
_get_instance_by_smart_keys(Model, model_label: str, fields_data: dict): keys = SMART_MATCH_KEYS.get(model_label) if not keys: return None lookup =
{} for k in keys: if k not in fields_data: return None raw_val = fields_data[k] val = _normalized_lookup_value(model_label, k, raw_val) if val is None or
(isinstance(val, str) and val.strip() == ""): return None lookup[k] = val # Projetscards : matching tolérant (normalisation DB faite en Python) if model_label ==
"app_acceuil.Projetscards": candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="") for obj in candidates: nom_db =
_norm_text(getattr(obj, "nom", "")) desc_db = _norm_text(getattr(obj, "description", "")) if nom_db == lookup.get("nom") and desc_db ==
lookup.get("description"): return obj return None # Autres : filter direct OK si tu normalises aussi à l'enregistrement (voir _upsert_objects) try: return
Model.objects.filter(**lookup).first() except Exception: return None def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True,
exclude_fields: set | None = None): """ UPSERT intelligent (sans suppression) : 1) Si id existe en base -> update 2) Sinon -> tentative match via
SMART_MATCH_KEYS (tolérant) 3) Sinon -> create """ Model = apps.get_model(model_label) exclude_fields = exclude_fields or set() for obj_data in
objects_list: obj_id = obj_data.get("id") fields_data = { k: v for k, v in obj_data.items() if k not in {"__model__", "id"} and k not in exclude_fields } # séparer
file fields file_payloads = {} for field in Model._meta.fields: if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
file_payloads[field.name] = fields_data.pop(field.name) instance = None # 1) update par ID si possible (uniquement si l'objet existe déjà) if keep_ids and obj_id
is not None: instance = Model.objects.filter(pk=obj_id).first() # 2) sinon match intelligent if instance is None: instance = _get_instance_by_smart_keys(Model,
model_label, fields_data) # 3) sinon create if instance is None: if keep_ids and obj_id is not None: instance = Model(pk=obj_id) else: instance = Model() #
appliquer champs simples + normalisation à l'enregistrement (anti-doublons) for k, v in fields_data.items(): # Normalisation à l'enregistrement pour éviter les
doublons aux imports suivants if model_label == "app_contact.User" and k == "email" and isinstance(v, str): v = _norm_email(v) if model_label ==

```

```

"app_contact.Role" and k == "nom_role" and isinstance(v, str): v = _norm_text(v) if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str): v = _norm_text(v) if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str): v = _norm_text(v) if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str): v = _norm_text(v) if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str): # ici on garde strip (et non _norm_text) pour ne pas altérer le contenu visible, # mais ça évite déjà beaucoup de doublons "bêtes". v = v.strip() setattr(instance, k, v) # restaurer fichiers for field_name, payload in file_payloads.items(): _restore_file_field(instance, field_name, payload) instance.save() @transaction.atomic def import_json_bundle(request): """ POST depuis page_administration.html Inputs: - json_portfolio (optionnel) - json_users (optionnel) - replace_portfolio (checkbox) - replace_users (checkbox) Sans synchroniser : - si replace_* est coché : purge + import - sinon : upsert intelligent + AUCUNE suppression """ if not _is_admin_session(request): return HttpResponseRedirect("Accès refusé (Admin uniquement).") if request.method != "POST": return redirect("dashboard_admin") json_portfolio = request.FILES.get("json_portfolio") json_users = request.FILES.get("json_users") replace_portfolio = bool(request.POST.get("replace_portfolio")) replace_users = bool(request.POST.get("replace_users")) # ----- Portfolio import ----- if json_portfolio: try: data = _parse_uploaded_json(json_portfolio) if data.get("export_type") != "portfolio": raise ValueError("Ce fichier n'est pas un export portfolio.") model_map = data.get("data", {}) if replace_portfolio: from app_acceuil.models import (Projetphotodeprofil, ProjetAproposDeMoi, Projetsfirstspeciality, Projetsmesrealisations, Projetscards, MesCompetencesCles, ReseauSocial, ) # Ordre "simple" ReseauSocial.objects.all().delete() MesCompetencesCles.objects.all().delete() Projetscards.objects.all().delete() Projetsmesrealisations.objects.all().delete() Projetsfirstspeciality.objects.all().delete() ProjetAproposDeMoi.objects.all().delete() Projetphotodeprofil.objects.all().delete() for model_label, objects_list in model_map.items(): _upsert_objects(model_label, objects_list, keep_ids=True) messages.success(request, "Import Portfolio terminé avec succès.") except Exception as e: messages.error(request, f"Erreur import portfolio: {e}") # ----- Users import ----- if json_users: try: data = _parse_uploaded_json(json_users) if data.get("export_type") != "users": raise ValueError("Ce fichier n'est pas un export users.") model_map = data.get("data", {}) if replace_users: # Ordre important : dépendances FK from app_user.models import RoleMenuAcces, Menu from app_contact.models import User, Role, Contact RoleMenuAcces.objects.all().delete() User.objects.all().delete() Contact.objects.all().delete() Menu.objects.all().delete() Role.objects.all().delete() # Ordre d'import conseillé ordered = [ "app_contact.Role", "app_user.Menu", "app_contact.User", "app_contact.Contact", "app_user.RoleMenuAcces", ] for model_label in ordered: if model_label in model_map: _upsert_objects(model_label, model_map[model_label], keep_ids=True) messages.success(request, "Import Utilisateurs terminé avec succès.") except Exception as e: messages.error(request, f"Erreur import utilisateurs: {e}") return redirect("dashboard_admin")

```

Oui, ce code est “suffisant” pour faire : ajout / mise à jour sans suppression (tant que tu ne coches pas “Remplacer”).

Mais si tu demandes si c'est vraiment “sûr” pour un usage où tu vas alimenter la DB uniquement par imports JSON sur la durée, alors il y a 2 risques réels à connaître (et 2 mini-corrections faciles qui rendent le tout nettement plus solide).

## Ce que ton code fait correctement

- Export OK (texte + base64 images)
- Import OK :
  - id trouvé → update
  - sinon → match “métier” (nom/email/nom\_projet...)

- sinon → create
  - **Aucune suppression** (sauf si `replace_*` est coché)
- 

## ⚠ Risque n°1 (important) : pour les modèles “autres que Projetscards”, le match peut rater → doublons

Tu normalises la valeur importée (`_normalized_lookup_value`) puis tu fais un `filter(**lookup)`.

Exemple :

- en base : `email = "Mario@Gmail.com"`
- dans JSON : `"mario@gmail.com"`
- lookup devient `"mario@gmail.com"`
- `filter(email="mario@gmail.com")` ne matche pas (SQLite est sensible à la casse)  
→ résultat : **doublon User**

👉 Tu as déjà mis la “normalisation à l’enregistrement” dans `_upsert_objects`, donc **tu es en bonne voie**.

Mais pour être vraiment sûr, il faut normaliser **aussi** les champs textuels clés **dans l’objet avant save**, ce que tu fais déjà pour email/role/menu/competences/réseau. ✓

✓ Conclusion : **ce risque est bien couvert** chez toi, tant que tes champs uniques sont bien dans cette liste.

---

## ⚠ Risque n°2 (le plus gênant) : Projetscards peut devenir lent + “faux matchs” possibles

Tu fais un scan Python :

```
candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="")
for obj in candidates:
    ...
```

- si tu as beaucoup de témoignages → ça devient lent

- si 2 témoignages ont vraiment le même nom + description (possible) → tu vas “écraser” l'un par l'autre (match ambigu)

👉 C'est exactement ce que le fingerprint règle “proprement”.

✓ Sans fingerprint, ton code reste utilisable, mais c'est **le point le moins “sûr”** à long terme.

---

## ✓ 2 mini-améliorations que je te recommande fortement (sans fingerprint)

### ✓ A) Limiter les candidats Projetcards avec une pré-filtration

Même si tu ne peux pas filtrer sur “nom normalisé”, tu peux filtrer sur une version “grossière” (contains/iexact) :

Remplace :

```
candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="")
```

par :

```
candidates = Model.objects.exclude(nom__isnull=True).exclude(nom__exact="")
raw_nom = fields_data.get("nom", "")
if isinstance(raw_nom, str) and raw_nom.strip():
    candidates = candidates.filter(nom__icontains=raw_nom.strip()[:20])
```

Ça réduit beaucoup le scan si tu as un volume qui monte.

### ✓ B) Sécuriser contre les “mauvais JSON” (structure)

Avant d'importer, vérifie que data["data"] est bien un dict :

```
model_map = data.get("data", {})
if not isinstance(model_map, dict):
    raise ValueError("Format JSON invalide: data doit être un objet (dict).")
```

# Verdict clair

✓ Oui, ton code est suffisant et fonctionnera bien pour importer petit à petit, sans suppression.

⚠ La seule vraie limite de “sécurité/fidélité” à long terme, c'est Projetscards (performance + collisions).

👉 Si tu veux une solution “ultra sûre” contre les doublons pour Projetscards, le fingerprint est la meilleure.

Mais si tu acceptes :

- peu de témoignages
- ou rareté des collisions
- et un scan Python  
alors ton code actuel + les 2 mini-améliorations ci-dessus est **correct**.

Si tu me dis environ combien de témoignages tu penses avoir (10 ? 200 ? 5000 ?), je te dis si le fingerprint est vraiment nécessaire ou non.

quel est donc le code sûr

Le **code le plus sûr** (dans ton cas “je veux alimenter la DB uniquement via imports JSON, progressivement, sans doublons, sans suppression”) c'est :

1. **Upsert par id si présent**
2. Sinon **upsert par une clé métier stable**
3. Et pour Projetscards, la clé métier la plus sûre = **fingerprint (hash) + contrainte unique en DB**  
→ c'est la seule façon d'être “blindé” contre les doublons, même si les IDs changent, même si le texte a des espaces/majuscules, et même si tu importes 1000 fois.

Ci-dessous, je te donne le **code “sûr” complet** (backend uniquement) :

## A) Modèle : Projetscards (anti-doublon dur)

Dans app\_acceuil/models.py (classe Projetscards) ajoute :

```
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

Puis makemigrations + migrate.

---

## B) json\_transfer.py : version sûre complète (fingerprint + normalisation)

Remplace ton json\_transfer.py par ceci :

```
import base64
import json
import mimetypes
import re
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden
from django.shortcuts import redirect

# -----
# Sécurité Admin
# -----
def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

# -----
# JSON safe
# -----
def _json_compatible_value(value):
    if value is None:
```

```

    return None
if isinstance(value, (datetime, date, time)):
    return value.isoformat()
if isinstance(value, Decimal):
    return str(value)
if isinstance(value, UUID):
    return str(value)
return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"_{hash}

```

Le **code le plus sûr** (vraiment “anti-doublons” sur la durée, même si les IDs changent, même si tu importes 100 fois) = **fingerprint (hash)** + **contrainte unique en base** pour Projetscards, et **normalisation à l'enregistrement** pour les champs clés (email, code\_menu, etc.).

Je te mets ci-dessous le **code complet sûr** :

---

## 1) app\_acceuil/models.py — ajouter **fingerprint (anti-doublon dur)**

Dans ta classe Projetscards, ajoute :

```
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
```

Puis :

```
python manage.py makemigrations  
python manage.py migrate
```

---

## 2) json\_transfer.py — VERSION SÛRE COMPLÈTE (fingerprint + normalisation)

Remplace ton fichier json\_transfer.py par celui-ci :

```
import base64  
import json  
import mimetypes  
import re  
import hashlib  
from datetime import datetime, date, time  
from decimal import Decimal  
from uuid import UUID  
  
from django.apps import apps  
from django.contrib import messages  
from django.core.files.base import ContentFile  
from django.core.files.storage import default_storage  
from django.db import transaction  
from django.db.models.fields.files import FileField, ImageField  
from django.http import HttpResponseRedirect, HttpResponseForbidden  
from django.shortcuts import redirect  
  
# -----  
# Helpers : sécurité + JSON  
# -----  
def _is_admin_session(request) -> bool:  
    user_data = request.session.get("user")  
    return bool(user_data and user_data.get("role") == "Administrateur")  
  
def _json_compatible_value(value):  
    if value is None:  
        return None  
    if isinstance(value, (datetime, date, time)):
```

```
    return value.isoformat()
if isinstance(value, Decimal):
    return str(value)
if isinstance(value, UUID):
    return str(value)
return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation + Fingerprint
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # \n \t espaces multiples => 1 espace
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    """
    Fingerprint stable pour Projetscards (témoignages).
    Tolérant aux espaces/retours ligne/majuscules.
    """
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    data = {
        "__model__": f"{{obj._meta.app_label}}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

```

```
for field in obj._meta.fields:
    name = field.name
    if name == "id":
        continue

    value = getattr(obj, name)

    # Files / Images => base64
    if isinstance(field, (FileField, ImageField)):
        if value and getattr(value, "name", None):
            file_path = value.name
            b64 = None
            try:
                with default_storage.open(file_path, "rb") as f:
                    raw = f.read()
                b64 = base64.b64encode(raw).decode("utf-8")
            except Exception:
                b64 = None

            mime, _ = mimetypes.guess_type(file_path)
            data[name] = {
                "storage_path": file_path,
                "filename": file_path.split("/")[-1],
                "mimetype": mime or "application/octet-stream",
                "base64": b64,
            }
        else:
            data[name] = None
    else:
        data[name] = _json_compatible_value(value)

return data


def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]


# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

from app_acceuil.models import (
```

```

        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetmesrealisations,
        Projetscards,
        MesCompetencesCles,
        ReseauSocial,
    )

payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

```

```

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "notes": ["User.password est exporté tel quel (hash)."],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    }
}

```

```
        } ,  
    }  
  
    date_prefix = datetime.now().strftime("%Y%m%d")  
    return _safe_json_response(payload, f"{date_prefix}_export_users.json")  
  
# -----  
# IMPORTS  
# -----  
def _parse_uploaded_json(uploaded_file):  
    raw = uploaded_file.read()  
    try:  
        text = raw.decode("utf-8")  
    except Exception:  
        text = raw.decode("latin-1")  
    return json.loads(text)  
  
def _restore_file_field(model_instance, field_name: str, file_dict: dict):  
    if not file_dict:  
        return  
  
    b64 = file_dict.get("base64")  
    filename = file_dict.get("filename") or "file.bin"  
    if not b64:  
        return  
  
    try:  
        raw = base64.b64decode(b64)  
    except Exception:  
        return  
  
    django_file = ContentFile(raw, name=filename)  
    field = getattr(model_instance, field_name)  
    field.save(filename, django_file, save=False)  
  
# -----  
# FUSION INTELLIGENTE : clés métier  
# -----  
SMART_MATCH_KEYS = {  
    # Portfolio  
    "app_acceuil.MesCompetencesCles": ["nom"],  
    "app_acceuil.ReseauSocial": ["nom"],  
    "app_acceuil.Projetmesrealisations": ["nom_projet"],
```

```

# Témoignages: match dur par fingerprint
"app_acceuil.Projetscards": ["fingerprint"],

# Users
"app_contact.Role": ["nom_role"],
"app_contact.User": ["email"],
"app_user.Menu": ["code_menu"],
"app_user.RoleMenuAcces": ["role", "menu"],
}

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None
        lookup[k] = v

    try:
        return Model.objects.filter(**lookup).first()
    except Exception:
        return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}

```

```

for field in Model._meta.fields:
    if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
        file_payloads[field.name] = fields_data.pop(field.name)

# --- Projetscards : calc fingerprint si absent ---
if model_label == "app_acceuil.Projetscards":
    if not fields_data.get("fingerprint"):
        fields_data["fingerprint"] = _fingerprint_projetscards(
            fields_data.get("nom", ""),
            fields_data.get("description", ""),
        )

instance = None

# 1) update par ID si existant
if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()

# 2) update par clé métier (fingerprint/email/...)
if instance is None:
    instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

# 3) create
if instance is None:
    instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs + normalisation à l'enregistrement
for k, v in fields_data.items():

    # normalisations sûres (anti-doublons)
    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)

    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    # fingerprint: toujours trim+lower

```

```

if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
    v = v.strip().lower()

# texte visible Projetscards: on garde le contenu, juste strip
if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
    v = v.strip()

setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")
    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    # ----- Portfolio import -----
    if json_portfolio:
        try:
            data = _parse_uploaded_json(json_portfolio)
            if data.get("export_type") != "portfolio":
                raise ValueError("Ce fichier n'est pas un export portfolio.")
            model_map = data.get("data", {})
            if not isinstance(model_map, dict):
                raise ValueError("Format JSON invalide: data doit être un objet (dict).")

            if replace_portfolio:
                from app_acceuil.models import (
                    Projetphotodeprofil,
                    ProjetAproposDeMoi,
                    Projetsfirstspeciality,
                    Projetmesrealisations,
                    Projetscards,

```

```

        MesCompetencesCles,
        ReseauSocial,
    )
ReseauSocial.objects.all().delete()
MesCompetencesCles.objects.all().delete()
Projetscards.objects.all().delete()
Projetmesrealisations.objects.all().delete()
Projetsfirstspeciality.objects.all().delete()
ProjetAproposDeMoi.objects.all().delete()
Projetphotodeprofil.objects.all().delete()

for model_label, objects_list in model_map.items():
    _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_users:
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
            Role.objects.all().delete()

        ordered = [
            "app_contact.Role",
            "app_user.Menu",
            "app_contact.User",
            "app_contact.Contact",
            "app_user.RoleMenuAcces",
        ]
        for model_label in ordered:

```

```

        if model_label in model_map:
            _upsert_objects(model_label, model_map[model_label], keep_ids=True)

        messages.success(request, "Import Utilisateurs terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import utilisateurs: {e}")

    return redirect("dashboard_admin")

```

---

### 3) (Très recommandé) Remplir `fingerprint` pour les témoignages déjà en base (1 fois)

Après migration, lance :

```
python manage.py shell
```

Puis :

```

from app_acceuil.models import Projetscards
import re, hashlib

def norm_text(s):
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def fp(nom, description):
    base = f"{norm_text(nom)}||{norm_text(description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

updated = 0
for o in Projetscards.objects.all():
    if not o.fingerprint:
        o.fingerprint = fp(o.nom, o.description)
        o.save(update_fields=["fingerprint"])
        updated += 1

print("fingerprints créés:", updated)

```

---

**Pourquoi c'est “le plus sûr”**

- Même si un JSON arrive avec de **nouveaux IDs**, Projetscards ne dupliquera pas grâce à `fingerprint + unique=True`.
- Tu gardes ton fonctionnement : **ajout / update, sans suppression** (hors “Remplacer”).

Si tu veux, je peux aussi te donner une **migration automatique** qui remplit `fingerprint` pendant `migrate` (sans passer par le shell).

Et si j'utilisais une nouvelle base de données dans ce cas est ce que je dois executer : 3) (Très recommandé) Remplir `fingerprint` pour les témoignages déjà en base (1 fois)

Si tu utilises **une nouvelle base de données vide**, alors **non**, tu n'as **pas besoin** d'exécuter l'étape 3) **Remplir fingerprint....**

## Pourquoi ?

- L'étape 3 sert uniquement à **remplir `fingerprint` pour les anciens enregistrements déjà présents** dans la base (avant que le champ existe).
- Dans une **base neuve**, il n'y a aucun Projetscards existant → donc rien à “backfiller”.

## Ce que tu dois faire dans une base neuve

1.  Ajouter le champ `fingerprint` dans Projetscards + migrer :

```
python manage.py makemigrations
python manage.py migrate
```

2.  Ensuite, quand tu importes tes JSON :

- ton code calcule automatiquement `fingerprint` si absent
- il l'enregistre
- et grâce à `unique=True`, la base empêchera les doublons

## Cas où tu DOIS faire l'étape 3

- Tu avais déjà des témoignages en base **avant** l'ajout du champ `fingerprint`
- Ou tu as migré mais tes anciens enregistrements ont `fingerprint="" / NULL`  
→ Là, oui, l'étape 3 est utile pour mettre la base “propre”.

Si tu veux, je peux te donner une vérification très simple (1 ligne) pour savoir si tu as besoin de l'étape 3 ou pas.

oui

Voici la vérification la plus simple pour savoir si tu dois faire l'étape 3 (backfill fingerprint) ou non.

## Vérification rapide (Django shell)

1. Ouvre le shell :

```
python manage.py shell
```

2. Exécute :

```
from app_acceuil.models import Projetscards

# Cas 1 : fingerprint vide ("")
print("fingerprint vides:", Projetscards.objects.filter(fingerprint="").count())

# Cas 2 : fingerprint NULL (si tu as mis null=True à un moment)
print("fingerprint NULL:", Projetscards.objects.filter(fingerprint__isnull=True).count())
```

### Comment interpréter

- Si les deux valeurs sont **0**   
→ Tu n'as rien à faire, l'étape 3 est inutile.
- Si au moins une valeur > 0   
→ Oui, fais l'étape 3 (remplissage), sinon les prochains imports risquent de créer des doublons ou d'échouer (unique).

---

## Variante encore plus simple (une seule ligne)

```
from app_acceuil.models import Projetscards; print(Projetscards.objects.filter(fingerprint="").count(),
Projetscards.objects.filter(fingerprint__isnull=True).count())
```

Si tu veux, je peux te donner aussi la commande “remplir fingerprint” qui ne met à jour **que** les lignes manquantes (sans toucher celles déjà correctes).

```
ValidationError at /dashboard {'fingerprint': ['Témoignage with this Fingerprint already exists.']} Request Method: POST Request URL:
http://127.0.0.1:8000/dashboard Django Version: 5.2.8 Exception Type: ValidationError Exception Value: {'fingerprint': ['Témoignage with this Fingerprint already exists.']} Exception Location: D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\lib\site-packages\django\db\models\base.py, line 1679,
```

in full\_clean Raised during: app\_acceuil.views.page\_Mon\_Espace\_Administration Python Executable: D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Scripts\python.exe Python Version: 3.11.9 Python Path: ['D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO', 'C:\\Program Files\\Python311\\python311.zip', 'C:\\Program Files\\Python311\\DLLs', 'C:\\Program Files\\Python311\\Lib', 'C:\\Program Files\\Python311', 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv', 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages'] Server time: Wed, 04 Feb 2026 19:01:33 +0000 Traceback Switch to copy-and-paste view D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages\\django\\core\\handlers\\exception.py, line 55, in inner response = get\_response(request) ^^^^^^^^^^^^^^^^^^^^^^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages\\django\\core\\handlers\\base.py, line 197, in \_get\_response response = wrapped\_callback(request, \*callback\_args, \*\*callback\_kwargs) ^^^^^^^^^^^^^^^^^^^^^^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\app\_acceuil\\views.py, line 393, in page\_Mon\_Espace\_Administration Projetscards.objects.create( ^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages\\django\\db\\models\\manager.py, line 87, in manager\_method return getattr(self.get\_queryset(), name)(\*args, \*\*kwargs) ^^^^^^^^^^^^^^^^^^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages\\django\\db\\models\\query.py, line 665, in create obj.save(force\_insert=True, using=self.db) ^^^^^^^^^^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\app\_acceuil\\models.py, line 114, in save self.full\_clean() ^^^^^^ ... Local vars D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages\\django\\db\\models\\base.py, line 1679, in full\_clean raise ValidationError(errors) ^^^^^^ ... Local vars Request information  
USER AnonymousUser GET No GET data POST Variable Value csrfmiddlewaretoken  
'nJYWvwCdT4ZAutYzu6nMIEaBC15m3QHTfPu7Wz7WMZqApcQNj8Lgot4XBZyXcmMS' nom 'Emma Eming' description ('Emma Heming, née le 18 juin 1978, est un mannequin, créatrice de mode et "actrice américaine d'origine guyanienne.") btn\_ajouter\_temoignage FILES Variable Value images  
<InMemoryUploadedFile: Emma Eming.JPG (image/jpeg)> COOKIES Variable Value csrftoken \*\*\*\*\* sessionid \*\*\*\*\* META Variable Value ALLUSERSPROFILE 'C:\\\\ProgramData' APPDATA 'C:\\\\Users\\\\SPSSE02MEN\\\\AppData\\\\Roaming'  
BUNDLED\_DEBUGPY\_PATH 'c:\\\\Users\\\\SPSSE02MEN\\\\.vscode\\\\extensions\\\\ms-python.debugpy-2025.18.0-win32-x64\\\\bundled\\\\libs\\\\debugpy'  
CHROME\_CRASHPAD\_PIPE\_NAME '\\\\\\pipe\\\\crashpad\_10848\_KQVIIBTMZBXUEGF' COLORTERM 'truecolor' COMMONPROGRAMFILES 'C:\\\\Program Files\\\\Common Files' COMMONPROGRAMFILES(X86) 'C:\\\\Program Files (x86)\\\\Common Files' COMMONPROGRAMW6432 'C:\\\\Program Files\\\\Common Files' COMPUTERNAME 'DESKTOP-S95K8U0' COMSPEC 'C:\\\\Windows\\\\system32\\\\cmd.exe' CONTENT\_LENGTH '10575'  
CONTENT\_TYPE ('multipart/form-data; 'boundary=----geckoformboundary5acfa8e41826eef262e64e652fc3dee2') CSRF\_COOKIE  
'2gGlBdFT35Ba5T2oZcyEdZ4w98DLjGf9' DJANGO\_SETTINGS\_MODULE 'portfolio.settings' DRIVERDATA  
'C:\\\\Windows\\\\System32\\\\Drivers\\\\DriverData' GATEWAY\_INTERFACE 'CGI/1.1' GIT\_ASKPASS \*\*\*\*\* GIT\_LFS\_PATH 'C:\\\\Program Files\\\\Git LFS' HOMEDRIVE 'C:' HOMEPATH '\\\\Users\\\\SPSSE02MEN' HTTP\_ACCEPT 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8'  
HTTP\_ACCEPT\_ENCODING 'gzip, deflate, br, zstd' HTTP\_ACCEPT\_LANGUAGE 'fr,fr-FR;q=0.9,en-US;q=0.8,en;q=0.7' HTTP\_CONNECTION 'keep-alive' HTTP\_COOKIE \*\*\*\*\* HTTP\_HOST '127.0.0.1:8000' HTTP\_ORIGIN 'http://127.0.0.1:8000' HTTP\_PRIORITY 'u=0, i'  
HTTP\_REFERER 'http://127.0.0.1:8000/dashboard' HTTP\_SEC\_FETCH\_DEST 'document' HTTP\_SEC\_FETCH\_MODE 'navigate'  
HTTP\_SEC\_FETCH\_SITE 'same-origin' HTTP\_SEC\_FETCH\_USER '?1' HTTP\_UPGRADE\_INSECURE\_REQUESTS '1' HTTP\_USER\_AGENT ('Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:147.0) Gecko/20100101 Firefox/147.0') LANG 'en\_US.UTF-8' LOCALAPPDATA 'C:\\\\Users\\\\SPSSE02MEN\\\\AppData\\\\Local' LOGONSERVER '\\\\\\DESKTOP-S95K8U0' NUMBER\_OF\_PROCESSORS '4' ONEDRIVE  
'C:\\\\Users\\\\SPSSE02MEN\\\\OneDrive' OS 'Windows\_NT' PATH ('D:\\\\PYTHON DJANGO HOPES\\PORTFOLIO '  
'DJANGO\\\\.venv\\\\Scripts;c:\\\\Users\\\\SPSSE02MEN\\\\AppData\\\\Roaming\\\\Code\\\\User\\\\globalStorage\\\\github.copilot-chat\\\\debugCommand;c:\\\\Users\\\\SPSSE02MEN\\\\AppData\\\\Roaming\\\\Code\\\\User\\\\globalStorage\\\\github.copilot-chat\\\\copilotCli;C:\\\\Program ')

'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition 2024.1.1\\bin;C:\\Program Files\\Common '

'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Windows\\System32\\OpenSSH\\;C:\\Program ' 'Files (x86)\\NVIDIA '

'Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start '

'Menu\\Programs\\Python;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' 'Menu\\Programs;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Python\\Python311\\Scripts;C:\\Program ' 'Files\\Git\\cmd;C:\\Program ' 'Files\\nodejs\\;C:\\src\\flutter\\bin;C:\\dart-sdk\\bin;;C:\\Program ' 'Files\\Git LFS;C:\\Program Files\\Python3\_12\_4\\Scripts\\;C:\\Program ' 'Files\\Python3\_12\_4\\;%IntelliJ IDEA%;%IntelliJ IDEA Community ' 'Edition%;%PyCharm Community '

'Edition%;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Microsoft VS ' 'Code\\bin;C:\\Program Files '(x86)\\Nmap;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\npm;C:\\Users\\SPSSE02MEN\\vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\bundled\\scripts\\noConfigScripts') PATHEXT '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW;.CPL' PATH\_INFO '/dashboard' PROCESSOR\_ARCHITECTURE 'AMD64' PROCESSOR\_IDENTIFIER 'Intel64 Family 6 Model 69 Stepping 1, GenuineIntel'

PROCESSOR\_LEVEL '6' PROCESSOR\_REVISION '4501' PROGRAMDATA 'C:\\ProgramData' PROGRAMFILES 'C:\\Program Files' PROGRAMFILES(X86) 'C:\\Program Files (x86)' PROGRAMW6432 'C:\\Program Files' PROMPT '\$P\$G' PSMODULEPATH ('C:\\Users\\SPSSE02MEN\\Documents\\WindowsPowerShell\\Modules;C:\\Program '

'Files\\WindowsPowerShell\\Modules;C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\Modules') PUBLIC 'C:\\Users\\Public' PYDEV\_DISABLE\_FILE\_VALIDATION '1' QUERY\_STRING " REMOTE\_ADDR '127.0.0.1' REMOTE\_HOST " REQUEST\_METHOD 'POST' RUN\_MAIN 'true' SCRIPT\_NAME " SERVER\_NAME 'DESKTOP-S95K8U0' SERVER\_PORT '8000' SERVER\_PROTOCOL 'HTTP/1.1' SERVER\_SOFTWARE 'WSGIServer/0.2' SYSTEMDRIVE 'C:' SYSTEMROOT 'C:\\Windows' TEMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp' TERM\_PROGRAM 'vscode' TERM\_PROGRAM\_VERSION '1.108.2' TMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp' USERDOMAIN 'AzureAD' USERDOMAIN\_ROAMINGPROFILE 'AzureAD' USERNAME 'SPSSE02MEN' USERPROFILE 'C:\\Users\\SPSSE02MEN' VIRTUAL\_ENV 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv' VIRTUAL\_ENV\_PROMPT '.venv' VS CODE\_DEBUGPY\_ADAPTER\_ENDPOINTS 'c:\\Users\\SPSSE02MEN\\vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\noConfigDebugAdapterEndpoints\\endpoint-77f6ce4ded07263f.txt' VS CODE\_GIT\_ASKPASS\_EXTRA\_ARGS '\*\*\*\*\*' VS CODE\_GIT\_ASKPASS\_MAIN '\*\*\*\*\*' VS CODE\_GIT\_ASKPASS\_NODE '\*\*\*\*\*' VS CODE\_GIT\_IPC\_HANDLE '\\\\pipe\\vscode-git-c146c63ca8-sock' VS CODE\_INJECTION '1' VS CODE PYTHON\_AUTOACTIVATE\_GUARD '1' WINDIR 'C:\\Windows' \_OLD\_VIRTUAL\_PATH ('c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\debugCommand;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\copilotCli;C:\\Program ' 'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition 2024.1.1\\bin;C:\\Program Files\\Common '

'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Windows\\System32\\OpenSSH\\;C:\\Program ' 'Files (x86)\\NVIDIA '

'Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start '

'Menu\\Programs\\Python;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' 'Menu\\Programs;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Python\\Python311\\Scripts;C:\\Program ' 'Files\\Git\\cmd;C:\\Program ' 'Files\\nodejs\\;C:\\src\\flutter\\bin;C:\\dart-sdk\\bin;;C:\\Program ' 'Files\\Git LFS;C:\\Program Files\\Python3\_12\_4\\Scripts\\;C:\\Program '

```
'Files\\Python3_12_4\\;%IntelliJ IDEA%;%IntelliJ IDEA Community ''Edition%';%PyCharm Community '
'Edition%;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Microsoft VS ' 'Code\\bin;C:\\Program Files '
'(x86)\\Nmap;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\npm;c:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-
x64\\bundled\\scripts\\noConfigScripts) wsgi.errors < io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> wsgi.file_wrapper <class
'wsgiref.util.FileWrapper'> wsgi.input <django.core.handlers.wsgi.LimitedStream object at 0x0000019942517A60> wsgi.multiprocess False wsgi.multithread
True wsgi.run_once False wsgi.url_scheme 'http' wsgi.version (1, 0) Settings Using settings module portfolio.settings Setting Value
ABSOLUTE_URL_OVERRIDES {} ADMINS [] ALLOWED_HOSTS [] APPEND_SLASH True AUTHENTICATION_BACKENDS
***** AUTH_PASSWORD_VALIDATORS ***** AUTH_USER_MODEL ***** BASE_DIR
WindowsPath('D:/PYTHON DJANGO HOPES/PORTFOLIO DJANGO') CACHES {'default': {'BACKEND':
'django.core.cache.backends.locmem.LocMemCache'}} CACHE_MIDDLEWARE_ALIAS 'default' CACHE_MIDDLEWARE_KEY_PREFIX
***** CACHE_MIDDLEWARE_SECONDS 600 CSRF_COOKIE_AGE 31449600 CSRF_COOKIE_DOMAIN None
CSRF_COOKIE_HTTPONLY False CSRF_COOKIE_NAME 'csrftoken' CSRF_COOKIE_PATH '/' CSRF_COOKIE_SAMESITE 'Lax'
CSRF_COOKIE_SECURE False CSRF_FAILURE_VIEW 'django.views.csrf.csrf_failure' CSRF_HEADER_NAME 'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS [] CSRF_USE_SESSIONS False DATABASES {'default': {'ATOMIC_REQUESTS': False, 'AUTOCOMMIT': True,
'CONN_HEALTH_CHECKS': False, 'CONN_MAX_AGE': 0, 'ENGINE': 'django.db.backends.sqlite3', 'HOST': '', 'NAME': WindowsPath('D:/PYTHON
DJANGO HOPES/PORTFOLIO DJANGO/db.sqlite3'), 'OPTIONS': {}, 'PASSWORD': '*****', 'PORT': '', 'TEST': {'CHARSET': None,
'COLLATION': None, 'MIGRATE': True, 'MIRROR': None, 'NAME': None}, 'TIME_ZONE': None, 'USER': ""}} DATABASE_ROUTERS []
DATA_UPLOAD_MAX_MEMORY_SIZE 2621440 DATA_UPLOAD_MAX_NUMBER_FIELDS 1000 DATA_UPLOAD_MAX_NUMBER_FILES 100
DATETIME_FORMAT 'N j, Y, P' DATETIME_INPUT_FORMATS ['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M',
'%m/%d/%Y %H:%M:%S', '%m/%d/%Y %H:%M:%S.%f', '%m/%d/%Y %H:%M', '%m/%d/%y %H:%M:%S', '%m/%d/%y %H:%M:%S.%f', '%m/%d/%y
%H:%M'] DATE_FORMAT 'N j, Y' DATE_INPUT_FORMATS ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y', '%b %d %Y', '%b %d, %Y', '%d %b %Y', '%d
%b, %Y', '%B %d %Y', '%B %d, %Y', '%d %B %Y', '%d %B, %Y'] DEBUG True DEBUG_PROPAGATE_EXCEPTIONS False DECIMAL_SEPARATOR
'.' DEFAULT_AUTO_FIELD 'django.db.models.BigAutoField' DEFAULT_CHARSET 'utf-8' DEFAULT_EXCEPTION_REPORTER
'django.views.debug.ExceptionReporter' DEFAULT_EXCEPTION_REPORTER_FILTER 'django.views.debug.SafeExceptionReporterFilter'
DEFAULT_FROM_EMAIL 'webmaster@localhost' DEFAULT_INDEX_TABLESPACE '' DEFAULT_TABLESPACE '' DISALLOWED_USER_AGENTS []
EMAIL_BACKEND 'django.core.mail.backends.smtp.EmailBackend' EMAIL_HOST 'localhost' EMAIL_HOST_PASSWORD '*****'
EMAIL_HOST_USER '' EMAIL_PORT 25 EMAIL_SSL_CERTFILE None EMAIL_SSL_KEYFILE '*****' EMAIL SUBJECT_PREFIX
'[Django]' EMAIL_TIMEOUT None EMAIL_USE_LOCALTIME False EMAIL_USE_SSL False EMAIL_USE_TLS False
FILE_UPLOAD_DIRECTORY_PERMISSIONS None FILE_UPLOAD_HANDLERS ['django.core.files.uploadhandler.MemoryFileUploadHandler',
'django.core.files.uploadhandler.TemporaryFileUploadHandler'] FILE_UPLOAD_MAX_MEMORY_SIZE 2621440 FILE_UPLOAD_PERMISSIONS 420
FILE_UPLOAD_TEMP_DIR None FIRST_DAY_OF_WEEK 0 FIXTURE_DIRS [] FORCE_SCRIPT_NAME None FORMAT_MODULE_PATH None
FORMS_URLFIELD_ASSUME_HTTPS False FORM_RENDERER 'django.forms.renderers.DjangoTemplates' IGNORABLE_404_URLS []
INSTALLED_APPS ['django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages',
'django.contrib.staticfiles', 'app_acceuil', 'app_contact', 'app_user'] INTERNAL_IPS [] LANGUAGES [('af', 'Afrikaans'), ('ar', 'Arabic'), ('ar-dz', 'Algerian
Arabic'), ('ast', 'Asturian'), ('az', 'Azerbaijani'), ('bg', 'Bulgarian'), ('be', 'Belarusian'), ('bn', 'Bengali'), ('br', 'Breton'), ('bs', 'Bosnian'), ('ca', 'Catalan'), ('ckb',
'Central Kurdish (Sorani)'), ('cs', 'Czech'), ('cy', 'Welsh'), ('da', 'Danish'), ('de', 'German'), ('dsb', 'Lower Sorbian'), ('el', 'Greek'), ('en', 'English'), ('en-au',
'Australian English'), ('en-gb', 'British English'), ('eo', 'Esperanto'), ('es', 'Spanish'), ('es-ar', 'Argentinian Spanish'), ('es-co', 'Colombian Spanish'), ('es-mx',
```

'Mexican Spanish'), ('es-ni', 'Nicaraguan Spanish'), ('es-ve', 'Venezuelan Spanish'), ('et', 'Estonian'), ('eu', 'Basque'), ('fa', 'Persian'), ('fi', 'Finnish'), ('fr', 'French'), ('fy', 'Frisian'), ('ga', 'Irish'), ('gd', 'Scottish Gaelic'), ('gl', 'Galician'), ('he', 'Hebrew'), ('hi', 'Hindi'), ('hr', 'Croatian'), ('hsb', 'Upper Sorbian'), ('hu', 'Hungarian'), ('hy', 'Armenian'), ('ia', 'Interlingua'), ('id', 'Indonesian'), ('ig', 'Igbo'), ('io', 'Ido'), ('is', 'Icelandic'), ('it', 'Italian'), ('ja', 'Japanese'), ('ka', 'Georgian'), ('kab', 'Kabyle'), ('kk', 'Kazakh'), ('km', 'Khmer'), ('kn', 'Kannada'), ('ko', 'Korean'), ('ky', 'Kyrgyz'), ('lb', 'Luxembourgish'), ('lt', 'Lithuanian'), ('lv', 'Latvian'), ('mk', 'Macedonian'), ('ml', 'Malayalam'), ('mn', 'Mongolian'), ('mr', 'Marathi'), ('ms', 'Malay'), ('my', 'Burmese'), ('nb', 'Norwegian Bokmål'), ('ne', 'Nepali'), ('nl', 'Dutch'), ('nn', 'Norwegian Nynorsk'), ('os', 'Ossetic'), ('pa', 'Punjabi'), ('pl', 'Polish'), ('pt', 'Portuguese'), ('pt-br', 'Brazilian Portuguese'), ('ro', 'Romanian'), ('ru', 'Russian'), ('sk', 'Slovak'), ('sl', 'Slovenian'), ('sq', 'Albanian'), ('sr', 'Serbian'), ('sr-latn', 'Serbian Latin'), ('sv', 'Swedish'), ('sw', 'Swahili'), ('ta', 'Tamil'), ('te', 'Telugu'), ('tg', 'Tajik'), ('th', 'Thai'), ('tk', 'Turkmen'), ('tr', 'Turkish'), ('tt', 'Tatar'), ('udm', 'Udmurt'), ('ug', 'Uyghur'), ('uk', 'Ukrainian'), ('ur', 'Urdu'), ('uz', 'Uzbek'), ('vi', 'Vietnamese'), ('zh-hans', 'Simplified Chinese'), ('zh-hant', 'Traditional Chinese')] LANGUAGES\_BIDI ['he', 'ar', 'ar-dz', 'ckb', 'fa', 'ug', 'ur'] LANGUAGE\_CODE 'en-us' LANGUAGE\_COOKIE\_AGE None LANGUAGE\_COOKIE\_DOMAIN None LANGUAGE\_COOKIE\_HTTPONLY False LANGUAGE\_COOKIE\_NAME 'django\_language' LANGUAGE\_COOKIE\_PATH '/' LANGUAGE\_COOKIE\_SAMESITE None LANGUAGE\_COOKIE\_SECURE False LOCALE\_PATHS [] LOGGING {} LOGGING\_CONFIG 'logging.config.dictConfig' LOGIN\_REDIRECT\_URL '/accounts/profile/' LOGIN\_URL '/accounts/login/' LOGOUT\_REDIRECT\_URL None MANAGERS [] MEDIA\_ROOT 'D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\DJANGO\\media' MEDIA\_URL '/media/' MESSAGE\_STORAGE 'django.contrib.messages.storage.fallback.FallbackStorage' MIDDLEWARE ['django.middleware.security.SecurityMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware', 'django.middleware.common.CommonMiddleware', 'django.middleware.csrf.CsrfViewMiddleware', 'django.contrib.auth.middleware.AuthenticationMiddleware', 'django.contrib.messages.middleware.MessageMiddleware', 'django.middleware.clickjacking.XFrameOptionsMiddleware'] MIGRATION\_MODULES {} MONTH\_DAY\_FORMAT 'F j' NUMBER\_GROUPING 0 PASSWORD\_HASHERS '\*\*\*\*\*' PASSWORD\_RESET\_TIMEOUT \*\*\*\*\* PREPEND\_WWW False ROOT\_URLCONF 'portfolio.urls' SECRET\_KEY '\*\*\*\*\*' SECRET\_KEY\_FALLBACKS '\*\*\*\*\*' SECURE\_CONTENT\_TYPE\_NOSNIFF True SECURE\_CROSS\_ORIGIN\_OPENER\_POLICY 'same-origin' SECURE\_HSTS\_INCLUDE\_SUBDOMAINS False SECURE\_HSTS\_PRELOAD False SECURE\_HSTS\_SECONDS 0 SECURE\_PROXY\_SSL\_HEADER None SECURE\_REDIRECT\_EXEMPT [] SECURE\_REFERRER\_POLICY 'same-origin' SECURE\_SSL\_HOST None SECURE\_SSL\_REDIRECT False SERVER\_EMAIL 'root@localhost' SESSION\_CACHE\_ALIAS 'default' SESSION\_COOKIE\_AGE 1209600 SESSION\_COOKIE\_DOMAIN None SESSION\_COOKIE\_HTTPONLY True SESSION\_COOKIE\_NAME 'sessionid' SESSION\_COOKIE\_PATH '/' SESSION\_COOKIE\_SAMESITE 'Lax' SESSION\_COOKIE\_SECURE False SESSION\_ENGINE 'django.contrib.sessions.backends.db' SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE False SESSION\_FILE\_PATH None SESSION\_SAVE\_EVERY\_REQUEST False SESSION\_SERIALIZER 'django.contrib.sessions.serializers.JSONSerializer' SETTINGS\_MODULE 'portfolio.settings' SHORT\_DATETIME\_FORMAT 'm/d/Y P' SHORT\_DATE\_FORMAT 'm/d/Y' SIGNING\_BACKEND 'django.core.signing.TimestampSigner' SILENCED\_SYSTEM\_CHECKS [] STATICFILES\_DIRS ['D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\DJANGO\\static/] STATICFILES\_FINDERS ['django.contrib.staticfiles.finders.FileSystemFinder', 'django.contrib.staticfiles.finders.AppDirectoriesFinder'] STATIC\_ROOT None STATIC\_URL '/static/' STORAGES {'default': {'BACKEND': 'django.core.files.storage.FileSystemStorage'}, 'staticfiles': {'BACKEND': 'django.contrib.staticfiles.storage.StaticFilesStorage'}} TEMPLATES [{'APP\_DIRS': True, 'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': ['templates/'], 'OPTIONS': {'context\_processors': ['django.template.context\_processors.request', 'django.contrib.auth.context\_processors.auth', 'django.contrib.messages.context\_processors.messages']}}, TEST\_NON\_SERIALIZED\_APPS [], TEST\_RUNNER 'django.test.runner.DiscoverRunner' THOUSAND\_SEPARATOR ',' TIME\_FORMAT 'P' TIME\_INPUT\_FORMATS ['%H:%M:%S', '%H:%M:%S.%f', '%H:%M'] TIME\_ZONE 'UTC' USE\_I18N True USE\_THOUSAND\_SEPARATOR False USE\_TZ True USE\_X\_FORWARDED\_HOST False USE\_X\_FORWARDED\_PORT False WSGI\_APPLICATION 'portfolio.wsgi.application' X\_FRAME\_OPTIONS 'DENY' YEAR\_MONTH\_FORMAT

'F Y' You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard page generated by the handler for this status code.

Cette erreur est **normale** dès que tu passes `fingerprint` en **unique** : tu essayes d'ajouter, via le formulaire `/dashboard`, un témoignage dont le **nom + description** (après normalisation) produit **le même fingerprint** qu'un témoignage déjà en base.

Donc ce n'est pas un bug du `fingerprint` : c'est ton **code de création** (`Projetscards.objects.create(...)`) qui doit être adapté pour :

- soit **refuser proprement** (message "déjà existant")
  - soit **mettre à jour** l'existant au lieu de créer (recommandé si tu veux "upsert" partout)
- 

## Option sûre recommandée : UPSET dans `views.py` (pas de crash)

Dans `app_acceuil/views.py`, à l'endroit où tu fais :

```
Projetscards.objects.create(...)
```

remplace par **update\_or\_create** basé sur `fingerprint`.

### 1) Ajoute en haut de `views.py` (imports + helpers)

```
import re
import hashlib

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

### 2) Dans ta vue `page_Mon_Espace_Administration` (POST "ajouter témoignage")

Au lieu de `create(...)`, fais :

```

from django.contrib import messages
from app_acceuil.models import Projetscards

nom = request.POST.get("nom", "")
description = request.POST.get("description", "")
image = request.FILES.get("images")

fp = _fingerprint_projetscards(nom, description)

obj, created = Projetscards.objects.update_or_create(
    fingerprint=fp,
    defaults={
        "nom": nom.strip(),
        "description": description.strip(),
        # si image présente, on la met; sinon on ne touche pas à l'image existante
        **({"images": image} if image else {}),
    }
)

if created:
    messages.success(request, "Témoignage ajouté.")
else:
    messages.success(request, "Témoignage déjà existant : mise à jour effectuée.")

```

 Résultat :

- plus de crash
- si un témoignage “identique” (selon ton fingerprint) existe → **mise à jour**
- sinon → **création**

## Option encore plus “propre” : calculer le fingerprint dans le modèle (centralisé)

Si tu veux que **toutes** les créations (admin, imports, dashboard, shell...) aient automatiquement le fingerprint, ajoute ça dans `app_acceuil/models.py` dans `Projetscards` :

```

import re
import hashlib
from django.db import models

class Projetscards(models.Model):

```

```
# ... tes champs ...
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")

def _norm_text(self, s):
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _compute_fp(self):
    base = f"{self._norm_text(self.nom)}||{self._norm_text(self.description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def save(self, *args, **kwargs):
    if not self.fingerprint:
        self.fingerprint = self._compute_fp()
    super().save(*args, **kwargs)
```

⚠ Même avec ça, tu peux quand même avoir une erreur unique si tu fais un `.create()` sur un doublon — donc **la meilleure UX reste update\_or\_create** dans la vue (option 1).

---

## Pourquoi l'erreur arrive “maintenant”

Parce que tu as rendu `fingerprint unique`, et tu as probablement :

- soit importé des données (donc `fingerprint` existant)
- soit tu as déjà un témoignage avec même nom+description

Et maintenant tu essaies d'en créer un deuxième identique → **Django empêche** (ce qui est exactement le but du système anti-doublon).

---

Si tu me colles **le bloc exact** de ton `page_Mon_Espace_Administration` autour de la ligne ~393 (la partie “ajouter témoignage”), je te renvoie le remplacement **100% adapté** à ton code actuel (noms de champs exacts, redirections, messages, etc.).

`views.py`

`Python`

`app_acceuil/views.py`

Réflexion durant 41s

Tu as 2 choses à régler pour que ce soit **vraiment “sûr”** avec un champ `fingerprint unique` :

1. **Quand tu ajoutes un témoignage depuis /dashboard**, il ne faut **plus** faire `Projetscards.objects.create(...)`, sinon tu risques exactement l'erreur que tu as : *Fingerprint already exists* (unicité).  
→ Il faut faire un `update_or_create` basé sur `fingerprint`.
  2. **Le modèle Projetscards doit générer un fingerprint stable** (hash) à partir de champs texte normalisés (ex: nom + description).  
→ Comme ça, même si tu ajoutes/importes plusieurs fois, Django “matche” au même objet et **met à jour** au lieu de créer un doublon.
- 

## 1) Code “sûr” côté modèle : `app_acceuil/models.py` (**Projetscards avec fingerprint**)

Dans ton `models.py`, ton `Projetscards` n'a pas `fingerprint` dans l'extrait que tu m'as montré

`models`

. Voici une version complète **avec fingerprint unique** (à adapter dans ton fichier) :

```
# app_acceuil/models.py
import hashlib
import re
from django.db import models
from django.core.exceptions import ValidationError

def nettoyer_espaces(texte):
    if texte:
        return re.sub(r'\s+', ' ', texte).strip()
    return texte

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def make_temoignage_fingerprint(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

```

class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    #  fingerprint unique (anti-doublon dur)
    fingerprint = models.CharField(
        max_length=64,
        unique=True,
        db_index=True,
        editable=False,
        null=True,
        blank=True,
    )

class Meta:
    verbose_name = "Témoignage"
    verbose_name_plural = "Témoignages Clients"
    ordering = ['-created_at']

def clean(self):
    self.nom = nettoyer_espaces(self.nom)
    if self.description:
        self.description = self.description.strip()

    #  calc fingerprint (stable)
    if self.nom and self.description:
        self.fingerprint = make_temoignage_fingerprint(self.nom, self.description)

def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

```

Ensute :

```

python manage.py makemigrations
python manage.py migrate

```

⚠ Si tu as déjà des doublons “nom+description” en base, la migration `unique=True` peut échouer.  
Dans ce cas il faut soit supprimer/éditer les doublons, soit remplir d’abord des fingerprints en gérant les collisions (je peux te donner la commande de management si tu veux).

---

## 2) Correction dans `app_acceuil/views.py` (bouton ajouter témoignage)

Actuellement, ton code fait bien un `create()`

`views`

, donc si `fingerprint` est unique, ça cassera dès que tu retombes sur un fingerprint existant.

Remplace le bloc `btn_ajouter_temoignage` par ça :

```
# app_acceuil/views.py
import hashlib
import re
from django.core.exceptions import ValidationError

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def make_temoignage_fingerprint(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# ... dans page_Mon_Espace_Administration ...

elif 'btn_ajouter_temoignage' in request.POST:
    nom = request.POST.get('nom')
    description = request.POST.get('description')
    img = request.FILES.get('images')

    fp = make_temoignage_fingerprint(nom, description)

    obj, created = Projetscards.objects.update_or_create(
        fingerprint=fp,
```

```

        defaults={
            "nom": nom,
            "description": description,
            "est_visible": True,
            # image : si tu veux remplacer l'image seulement si fournie
            **({"images": img} if img else {})
        }
    )

    if created:
        messages.success(request, "Témoignage ajouté.")
    else:
        messages.info(request, "Témoignage déjà existant : mise à jour effectuée.")

```

→ Résultat :

- si le témoignage (nom+description) **n'existe pas** → il est créé
  - si le témoignage **existe déjà** → il est **mis à jour** (et plus d'erreur `fingerprint already exists`)
- 

### 3) Ta question : “Si j’utilise une nouvelle base, dois-je remplir `fingerprint` ?”

Si ta nouvelle base est vide : NON, tu n’as pas besoin de faire “Remplir `fingerprint` pour les témoignages déjà en base”, puisqu’il n’y en a pas.

Si ta base n’est pas vide et que tu ajoutes `fingerprint` :

- soit tu laisses `null=True, blank=True` au début, tu remplis progressivement
  - soit tu fais une étape de remplissage en masse (recommandé) **avant** de mettre `unique=True strict`
- 

### Pourquoi ton erreur arrive maintenant ?

Parce que ton `save()` appelle `full_clean()`

models

, donc si `fingerprint` est `unique=True`, Django détecte immédiatement le doublon et lève `ValidationError`.

Avec `update_or_create(fingerprint=...)`, tu ne crées plus un doublon → tu “retombes” sur l’existant.

---

Si tu veux, colle-moi ton **vrai `models.py` actuel** (celui qui contient déjà `fingerprint`) et je te renvoie la version **100% cohérente** (modèle + vue + import JSON) sans risque de migration foireuse.

Sources

`models.py`

Python

Réflexion durant 27s

Ton erreur vient de la combinaison suivante dans **Projetscards** :

- `fingerprint` est **unique=True**
- et tu as `blank=True, default=""`

Donc si tu crées un témoignage **sans fournir `fingerprint`**, Django met "" (chaîne vide). Au 2<sup>e</sup> témoignage, tu as encore "" ⇒ **collision** ⇒ `ValidationError`: `fingerprint already exists`. On le voit dans ton modèle : `fingerprint = ... unique=True ... default=""` et ton `save()` appelle `full_clean()` (donc vérifie l’unicité avant d’enregistrer).

`models`

Et dans ta vue, tu fais bien un `Projetscards.objects.create(...)` sans `fingerprint`.

`views`



## Le “code sûr” (recommandé) côté modèle

Objectif : **fingerprint calculé automatiquement** (hash), **pas de valeur vide**, donc pas de collision, et c’est compatible avec ton import JSON (anti-doublon solide).

## 1) app\_acceuil/models.py — remplace la classe Projetscards par ceci

```
# app_acceuil/models.py
import hashlib
import re
from django.db import models
from django.core.exceptions import ValidationError

def nettoyer_espaces(texte):
    if texte:
        return re.sub(r'\s+', ' ', texte).strip()
    return texte

def _fingerprint_for_temoignage(nom: str, description: str) -> str:
    """
    Fingerprint stable (même texte => même hash).
    IMPORTANT: ne dépend pas du fichier image (sinon au moindre changement de fichier, fingerprint change).
    """
    nom_n = nettoyer_espaces(nom or "").lower()
    desc_n = nettoyer_espaces(description or "").lower()
    base = f"{nom_n}||{desc_n}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

    #  le plus important: PAS de default=""
    fingerprint = models.CharField(
        max_length=64,
        unique=True,
        db_index=True,
        null=True,
        blank=True,
        default=None,
    )

    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ['-created_at']
```

```

def clean(self):
    self.nom = nettoyer_espaces(self.nom)
    # tu peux aussi nettoyer description si tu veux
    # self.description = nettoyer_espaces(self.description)

def save(self, *args, **kwargs):
    #  calcul AVANT full_clean (sinon unique check échoue)
    if not self.fingerprint:
        self.fingerprint = _fingerprint_for_temoignage(self.nom, self.description)

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

```

## 2) Migration

Après ça :

```

python manage.py makemigrations
python manage.py migrate

```

---

## Est-ce que tu dois faire “Remplir fingerprint...” si tu utilises une nouvelle base ?

**Non**, si ta base est neuve (aucun témoignage existant), tu n'as rien à “remplir une fois”.

Le fingerprint sera calculé automatiquement à chaque création via `save()`.

 En revanche, si tu avais déjà des témoignages en base, alors oui, il aurait fallu remplir leurs fingerprints (sinon ils restent à NULL), mais ce n'est pas obligatoire tant que tu ne crées pas d'objets qui provoquent collision.

---

## Pourquoi tu as eu l'erreur “Fingerprint already exists” en ajoutant un témoignage ?

Parce qu'avec ton modèle actuel, le 2<sup>e</sup> témoignage a très probablement tenté d'enregistrer le **même fingerprint** (ou "") et `full_clean()` a bloqué avant insert.

models

Avec le modèle “sûr” ci-dessus :

- plus de `default=""`
  - `fingerprint` auto
  - donc plus de collision “bête”.
- 

## Bonus (facultatif mais utile) : adapter ton import JSON (anti-doublon robuste)

Dans ton `json_transfer.py`, pour le matching intelligent des témoignages, le plus fiable devient :

```
SMART_MATCH_KEYS["app_acceuil.Projetscards"] = ["fingerprint"]
```

Et si certains JSON n’ont pas `fingerprint`, tu peux le calculer avant `upsert` (même formule SHA256).

---

Si tu veux, colle-moi ton `json_transfer.py` final (celui que tu utilises réellement) et je te donne la **version complète “propre”** avec :

- match `Projetscards` par `fingerprint`
- fallback nom+description si `fingerprint` absent
- et import sans suppression (comme tu veux).

Sources

```
import base64 import json import mimetypes import re import hashlib from datetime import datetime, date, time from decimal import Decimal from uuid import UUID from django.apps import apps from django.contrib import messages from django.core.files.base import ContentFile from django.core.files.storage import default_storage from django.db import transaction from django.db.models.fields.files import FileField, ImageField from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden from django.shortcuts import redirect # ----- # Helpers : sécurité + JSON # -----  
def _is_admin_session(request) -> bool: user_data = request.session.get("user") return bool(user_data and user_data.get("role") == "Administrateur") def _json_compatible_value(value): if value is None: return None if isinstance(value, (datetime, date, time)): return value.isoformat() if isinstance(value, Decimal): return str(value) if isinstance(value, UUID): return str(value) return value def _safe_json_response(payload: dict, filename: str) -> HttpResponseRedirect: content = json.dumps(payload, ensure_ascii=False, indent=2, default=str) resp = HttpResponseRedirect(content, content_type="application/json; charset=utf-8") resp["Content-Disposition"] = f'attachment; filename="{filename}"' return resp # ----- # Normalisation + Fingerprint # -----  
def _norm_text(s: str) -> str: s = (s or "").strip() s = re.sub(r"\s+", " ", s) # \n \t espaces multiples => 1 espace return s.lower()  
def _norm_email(s: str) -> str: return
```

```

_norm_text(s) def _fingerprint_projetscards(nom: str, description: str) -> str: """ Fingerprint stable pour Projetscards (témoignages). Tolérant aux espaces/retours ligne/majuscules. """
base = f"{{_norm_text(nom)}}{{_norm_text(description)}}"
return hashlib.sha256(base.encode("utf-8")).hexdigest() # -----
# Serialize : Django -> dict #
def _serialize_instance(obj): data = { "__model__": f"{{obj._meta.app_label}}.{{obj.__class__.__name__}}", "id": obj.pk, }
for field in obj._meta.fields: name = field.name if name == "id": continue value = getattr(obj, name) # Files / Images => base64 if isinstance(field, (FileField, ImageField)): if value and getattr(value, "name", None): file_path = value.name b64 = None try: with default_storage.open(file_path, "rb") as f: raw = f.read() b64 = base64.b64encode(raw).decode("utf-8") except Exception: b64 = None mime, _ = mimetypes.guess_type(file_path) data[name] = { "storage_path": file_path, "filename": file_path.split("/")[-1], "mimetype": mime or "application/octet-stream", "base64": b64, } else: data[name] = None else: data[name] = _json_compatible_value(value)
return data
def _serialize_queryset(qs): return [_serialize_instance(o) for o in qs] # -----
# EXPORTS #
def export_portfolio_json(request): if not _is_admin_session(request): return HttpResponseRedirect("Accès refusé (Admin uniquement.)")
from app_acceuil.models import ( Projetphotodeprofil, ProjetAproposDeMoi, Projetsfirstspeciality, Projetsrealisations, Projetscards, MesCompetencesCles, ReseauSocial, )
payload = { "export_type": "portfolio", "exported_at": datetime.now().isoformat(), "version": 1, "data": { "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()), "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()), "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()), "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetsrealisations.objects.all()), "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()), "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()), "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()), } }, date_prefix = datetime.now().strftime("%Y%m%d") return _safe_json_response(payload, f"{{date_prefix}}_export_portfolio.json")
def export_users_json(request): if not _is_admin_session(request): return HttpResponseRedirect("Accès refusé (Admin uniquement.)")
from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces
payload = { "export_type": "users", "exported_at": datetime.now().isoformat(), "version": 1, "notes": ["User.password est exporté tel quel (hash)."], "data": { "app_contact.Role": _serialize_queryset(Role.objects.all()), "app_contact.User": _serialize_queryset(User.objects.all()), "app_contact.Contact": _serialize_queryset(Contact.objects.all()), "app_user.Menu": _serialize_queryset(Menu.objects.all()), "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()), } }, date_prefix = datetime.now().strftime("%Y%m%d") return _safe_json_response(payload, f"{{date_prefix}}_export_users.json") # -----
# IMPORTS #
def _parse_uploaded_json(uploaded_file): raw = uploaded_file.read()
try: text = raw.decode("utf-8") except Exception: text = raw.decode("latin-1")
return json.loads(text)
def _restore_file_field(model_instance, field_name: str, file_dict: dict): if not file_dict: return
b64 = file_dict.get("base64")
filename = file_dict.get("filename") or "file.bin" if not b64: return
try: raw = base64.b64decode(b64) except Exception: return
django_file = ContentFile(raw, name=filename)
field = getattr(model_instance, field_name)
field.save(filename, django_file, save=False) # -----
# FUSION INTELLIGENTE : clés métier #
SMART_MATCH_KEYS = { # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"], "app_acceuil.ReseauSocial": ["nom"], "app_acceuil.Projetmesrealisations": ["nom_projet"], # Témoignages: match dur par fingerprint
    "app_acceuil.Projetscards": ["fingerprint"], # Users
    "app_contact.Role": ["nom_role"], "app_contact.User": ["email"], "app_user.Menu": ["code_menu"], "app_user.RoleMenuAcces": ["role", "menu"], } def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict): keys = SMART_MATCH_KEYS.get(model_label)
if not keys: return None
lookup = {}
for k in keys: if k not in fields_data: return None
v = fields_data[k]
if v is None or (isinstance(v, str) and v.strip() == ""): return None
lookup[k] = v
try: return Model.objects.filter(**lookup).first() except Exception: return None
def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None): Model = apps.get_model(model_label)
exclude_fields = exclude_fields or set()
for obj_data in objects_list: obj_id = obj_data.get("id")
fields_data = { k: v for k, v in obj_data.items() if k not in {"__model__", "id"} and k not in exclude_fields } # séparer file fields
file_payloads = {} for field in Model._meta.fields: if isinstance(field, (FileField, ImageField)) and field.name in fields_data: file_payloads[field.name] = fields_data.pop(field.name) # --- Projetscards : calc

```

```

fingerprint si absent --- if model_label == "app_acceuil.Projetscards": if not fields_data.get("fingerprint"): fields_data["fingerprint"] =
_fingerprint_projetscards( fields_data.get("nom", ""), fields_data.get("description", ""), ) instance = None # 1) update par ID si existant if keep_ids and obj_id
is not None: instance = Model.objects.filter(pk=obj_id).first() # 2) update par clé métier (fingerprint/email/...) if instance is None: instance =
_get_instance_by_smart_keys(Model, model_label, fields_data) # 3) create if instance is None: instance = Model(pk=obj_id) if (keep_ids and obj_id is not
None) else Model() # appliquer champs + normalisation à l'enregistrement for k, v in fields_data.items(): # normalisations sûres (anti-doublons) if model_label
== "app_contact.User" and k == "email" and isinstance(v, str): v = _norm_email(v) if model_label == "app_contact.Role" and k == "nom_role" and
isinstance(v, str): v = _norm_text(v) if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str): v = _norm_text(v) if model_label ==
"app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str): v = _norm_text(v) if model_label == "app_acceuil.ReseauSocial" and k == "nom"
and isinstance(v, str): v = _norm_text(v) # fingerprint: toujours trim+lower if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and
isinstance(v, str): v = v.strip().lower() # texte visible Projetscards: on garde le contenu, juste strip if model_label == "app_acceuil.Projetscards" and k in
("nom", "description") and isinstance(v, str): v = v.strip() setattr(instance, k, v) # restaurer fichiers for field_name, payload in file_payloads.items():
_restore_file_field(instance, field_name, payload) instance.save() @transaction.atomic def import_json_bundle(request): if not _is_admin_session(request):
return HttpResponseForbidden("Accès refusé (Admin uniquement.)") if request.method != "POST": return redirect("dashboard_admin") json_portfolio =
request.FILES.get("json_portfolio") json_users = request.FILES.get("json_users") replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users")) # ----- Portfolio import ----- if json_portfolio: try: data = _parse_uploaded_json(json_portfolio) if
data.get("export_type") != "portfolio": raise ValueError("Ce fichier n'est pas un export portfolio.") model_map = data.get("data", {}) if not
isinstance(model_map, dict): raise ValueError("Format JSON invalide: data doit être un objet (dict.)") if replace_portfolio: from app_acceuil.models import (
Projetphotodeprofil, ProjetAproposDeMoi, Projetsfirstspeciality, Projetmesrealisations, Projetscards, MesCompetencesCles, ReseauSocial, )
ReseauSocial.objects.all().delete() MesCompetencesCles.objects.all().delete() Projetscards.objects.all().delete() Projetmesrealisations.objects.all().delete()
Projetsfirstspeciality.objects.all().delete() ProjetAproposDeMoi.objects.all().delete() Projetphotodeprofil.objects.all().delete() for model_label, objects_list in
model_map.items(): _upsert_objects(model_label, objects_list, keep_ids=True) messages.success(request, "Import Portfolio terminé avec succès.") except
Exception as e: messages.error(request, f"Erreur import portfolio: {e}") # ----- Users import ----- if json_users: try: data =
_parse_uploaded_json(json_users) if data.get("export_type") != "users": raise ValueError("Ce fichier n'est pas un export users.") model_map = data.get("data",
{}) if not isinstance(model_map, dict): raise ValueError("Format JSON invalide: data doit être un objet (dict.)") if replace_users: from app_user.models import
RoleMenuAcces, Menu from app_contact.models import User, Role, Contact RoleMenuAcces.objects.all().delete() User.objects.all().delete()
Contact.objects.all().delete() Menu.objects.all().delete() Role.objects.all().delete() ordered = [ "app_contact.Role", "app_user.Menu", "app_contact.User",
"app_contact.Contact", "app_user.RoleMenuAcces", ] for model_label in ordered: if model_label in model_map: _upsert_objects(model_label,
model_map[model_label], keep_ids=True) messages.success(request, "Import Utilisateurs terminé avec succès.") except Exception as e:
messages.error(request, f"Erreur import utilisateurs: {e}") return redirect("dashboard_admin")

```

Ton `json_transfer.py` est **bon et “sûr” côté import/export** ✓ (tu calcules `fingerprint` si absent, tu matches par `fingerprint`, tu normalises).

Le problème qui te reste (l'erreur `Fingerprint already exists`) vient **pas** de ce fichier-là, mais de la **création depuis le dashboard + la définition du champ `fingerprint` dans le modèle.**

## 1) Ce qu'il faut absolument corriger dans `models.py`

Si dans `Projetscards` tu as un truc comme :

```
fingerprint = models.CharField(... unique=True, blank=True, default="")
```

→ ce n'est pas sûr : `default=""` provoque des collisions (le 2e enregistrement essaie d'enregistrer aussi "").

✓ Solution sûre : `pas de default=""`, et calcul automatique avant `full_clean()`.

### Version sûre du champ

```
fingerprint = models.CharField(  
    max_length=64,  
    unique=True,  
    db_index=True,  
    null=True,  
    blank=True,  
    default=None,  
    editable=False,  
)
```

### Et dans `save()` de `Projetscards` (important)

```
def save(self, *args, **kwargs):  
    if not self.fingerprint:  
        from .models import make_temoignage_fingerprint # ou mets la fonction au-dessus  
        self.fingerprint = make_temoignage_fingerprint(self.nom, self.description)  
  
    self.full_clean()  
    super().save(*args, **kwargs)
```

Si ta fonction est dans le même fichier, tu l'appelles directement sans import.

---

## 2) Ce qu'il faut corriger dans `views.py` (dashboard)

Même avec un modèle parfait, si tu fais :

```
Projetscards.objects.create(...)
```

→ tu vas retomber sur des collisions “uniques” si le témoignage existe déjà.

✓ Solution sûre : utiliser `update_or_create` basé sur `fingerprint`.

## Patch sûr pour “ajouter témoignage”

```
import hashlib, re

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def make_temoignage_fingerprint(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# ... dans ton POST ...
nom = request.POST.get("nom", "")
description = request.POST.get("description", "")
img = request.FILES.get("images")

fp = make_temoignage_fingerprint(nom, description)

obj, created = Projetscards.objects.update_or_create(
    fingerprint=fp,
    defaults={
        "nom": nom.strip(),
        "description": description.strip(),
        **({"images": img} if img else {}),
    }
)

if created:
    messages.success(request, "Témoignage ajouté.")
else:
    messages.info(request, "Témoignage déjà existant : mise à jour effectuée.")
```

---

## 3) Ton `json_transfer.py` : faut-il changer quelque chose ?

- ✓ Ton code actuel est déjà “propre”.

Mais je te recommande une micro-sécurité en plus :

### (Optionnel) Protéger l'upsert si la DB renvoie une collision

Si jamais deux objets existent déjà avec le même fingerprint (cas base sale), `save()` peut lever une erreur. Tu peux capturer :

```
from django.db import IntegrityError

try:
    instance.save()
except IntegrityError:
    # fallback : retrouver l'existant et le mettre à jour
    if model_label == "app_acceuil.Projetscards":
        fp = fields_data.get("fingerprint")
        existing = Model.objects.filter(fingerprint=fp).first()
        if existing:
            instance = existing
            for k, v in fields_data.items():
                setattr(instance, k, v)
                for field_name, payload in file_payloads.items():
                    _restore_file_field(instance, field_name, payload)
            instance.save()
        else:
            raise
    else:
        raise
```

---

## Conclusion (très clair)

- ✓ `json_transfer.py` : OK, sûr.
- ✗ Le souci vient de :
  1. `fingerprint` avec `default=""` (à retirer)
  2. `create()` dans le dashboard (à remplacer par `update_or_create`)

Si tu veux, je peux te renvoyer **la classe Projetcards exacte** corrigée à partir de ton `models.py` uploadé (avec le minimum de modifications).

Pouvez vous donner tous les fichiers corrigés complets  
Réflexion durant 1m 3s

Je peux te donner **tous les fichiers complets**... mais là, il y a un point bloquant :

- Le **views.py que tu as uploadé** (/mnt/data/views.py) n'est **pas** app\_acceuil/views.py : il importe User, Menu, RoleMenuAcces depuis .models, donc ça ressemble à **app\_user/views.py**.
- Le **models.py uploadé** (/mnt/data/models.py) correspond aussi à **app\_user/models.py** (classes Menu, RoleMenuAcces, etc.), et **pas** à app\_acceuil/models.py où se trouve Projetscards.

👉 Donc je ne peux pas te recoller "les fichiers complets corrigés" de **app\_acceuil/models.py** et **app\_acceuil/views.py** à l'identique **sans les vrais fichiers**.

✓ Par contre, je peux te donner **immédiatement** :

1. **Le fichier complet json\_transfer.py corrigé** (safe + fingerprint)
2. **Le fichier complet app\_acceuil/models.py (section Projetscards)** à copier/coller tel quel
3. **Le bloc complet à remplacer dans app\_acceuil/views.py** (pour ne plus faire create() et éviter le crash unique)

C'est exactement ce qui corrige ton erreur :

```
{'fingerprint': ['Témoignage with this Fingerprint already exists.']}
```

---

## 1) json\_transfer.py — fichier complet corrigé (SAFE)

Copie/colle ce fichier entier (c'est celui que tu as mis + petite sécurisation finale).

```
import base64
import json
import mimetypes
import re
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID
```

```
from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponseRedirect:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponseRedirect(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation + Fingerprint
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # \n \t espaces multiples => 1 espace
    return s.lower()
```

```
def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    """
        Fingerprint stable pour Projetscards (témoignages).
        Tolérant aux espaces/retours ligne/majuscules.
    """
    base = f"({_norm_text(nom)}||{_norm_text(description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue

        value = getattr(obj, name)

        # Files / Images => base64
        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name
                b64 = None
                try:
                    with default_storage.open(file_path, "rb") as f:
                        raw = f.read()
                    b64 = base64.b64encode(raw).decode("utf-8")
                except Exception:
                    b64 = None

                mime, _ = mimetypes.guess_type(file_path)
                data[name] = {
                    "storage_path": file_path,
                    "filename": file_path.split("/")[-1],
                    "mimetype": mime or "application/octet-stream",
                }
            else:
                data[name] = None
        else:
            data[name] = value

    return data
```

```

        "base64": b64,
    }
else:
    data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetcards": _serialize_queryset(Projetcards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}

```

```
date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "notes": ["User.password est exporté tel quel (hash)."],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
}
date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_users.json")

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return
    b64 = file_dict.get("base64")
```

```

filename = file_dict.get("filename") or "file.bin"
if not b64:
    return

try:
    raw = base64.b64decode(b64)
except Exception:
    return

django_file = ContentFile(raw, name=filename)
field = getattr(model_instance, field_name)
field.save(filename, django_file, save=False)

# -----
# FUSION INTELLIGENTE : clés métier
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"],

    # Témoignages: match dur par fingerprint
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"],
}
}

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None

```

```

lookup[k] = v

try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # --- Projetscards : calc fingerprint si absent ---
        if model_label == "app_acceuil.Projetscards":
            if not fields_data.get("fingerprint"):
                fields_data["fingerprint"] = fingerprint_projetcards(
                    fields_data.get("nom", ""),
                    fields_data.get("description", ""),
                )

        instance = None

        # 1) update par ID si existant
        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first()

        # 2) update par clé métier
        if instance is None:
            instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

        # 3) create
        if instance is None:

```

```

instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs + normalisation à l'enregistrement
for k, v in fields_data.items():

    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)

    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
        v = v.strip().lower()

    if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
        v = v.strip()

    setattr(instance, k, v)

for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

```

```

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")
    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

```

```

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_portfolio:
            from app_acceuil.models import (
                Projetphotodeprofil,
                ProjetAproposDeMoi,
                Projetsfirstspeciality,
                Projetmesrealisations,
                Projetcards,
                MesCompetencesCles,
                ReseauSocial,
            )
            ReseauSocial.objects.all().delete()
            MesCompetencesCles.objects.all().delete()
            Projetcards.objects.all().delete()
            Projetmesrealisations.objects.all().delete()
            Projetsfirstspeciality.objects.all().delete()
            ProjetAproposDeMoi.objects.all().delete()
            Projetphotodeprofil.objects.all().delete()

        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

        messages.success(request, "Import Portfolio terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

```

```

if replace_users:
    from app_user.models import RoleMenuAcces, Menu
    from app_contact.models import User, Role, Contact

    RoleMenuAcces.objects.all().delete()
    User.objects.all().delete()
    Contact.objects.all().delete()
    Menu.objects.all().delete()
    Role.objects.all().delete()

ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

---

## 2) app\_acceuil/models.py — correction “sûre” pour Projetscards

Le point clé : **ne jamais avoir default=""** sur un champ unique, sinon tu crées des collisions.  
Et il faut calculer le fingerprint **avant** full\_clean().

👉 Remplace TA classe Projetscards par celle-ci :

```

import re
import hashlib
from django.db import models

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)

```

```

return s.lower()

def make_temoignage_fingerprint(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

    # ✅ IMPORTANT: pas de default="" !
    fingerprint = models.CharField(
        max_length=64,
        unique=True,
        db_index=True,
        null=True,
        blank=True,
        default=None,
        editable=False,
    )

    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ['-created_at']

    def save(self, *args, **kwargs):
        # ✅ calcul AVANT full_clean / contraintes unique
        if not self.fingerprint:
            self.fingerprint = make_temoignage_fingerprint(self.nom, self.description)

        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom if self.nom else f"Témoignage {self.id}"

```

Ensuite :

```
python manage.py makemigrations
```

```
python manage.py migrate
```

---

### 3) app\_acceuil/views.py — correction complète du “POST ajouter témoignage”

Dans ta vue qui gère /dashboard, remplace le bloc où tu fais `Projetscards.objects.create(...)` par ceci :

```
import re
import hashlib
from django.contrib import messages
from app_acceuil.models import Projetscards

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def make_temoignage_fingerprint(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# ... dans ton POST ...
nom = request.POST.get("nom", "")
description = request.POST.get("description", "")
img = request.FILES.get("images")

fp = make_temoignage_fingerprint(nom, description)

obj, created = Projetscards.objects.update_or_create(
    fingerprint=fp,
    defaults={
        "nom": nom.strip(),
        "description": description.strip(),
        **({"images": img} if img else {}),
    },
)

if created:
    messages.success(request, "Témoignage ajouté.")
else:
    messages.info(request, "Témoignage déjà existant : mise à jour effectuée.")
```

 Résultat : plus de crash, et comportement “sûr” :

- si même nom+description → **mise à jour**
  - sinon → **création**
- 

## Pour que je te rende vraiment “tous les fichiers complets” au millimètre

Il me manque juste les vrais fichiers :

- app\_acceuil/models.py
- app\_acceuil/views.py

Parce que ceux que tu as uploadés ici sont ceux de app\_user.

 Si tu les uploads, je te renvoie **les deux fichiers complets corrigés** (avec tes imports, tes autres fonctions, ton HTML, tes redirections), sans approximation.

views.py

Python

Réflexion durant 5m 22s

Voici les **fichiers complets corrigés** (avec anti-doublon “fingerprint” robuste + correction de l’erreur).

L’erreur vient du fait que tu fais un `Projetscards.objects.create(...)`

views

alors que `fingerprint` est **unique** (donc si un témoignage identique existe déjà, `full_clean()` lève la `ValidationError`)

models

.

---

### 1) app\_acceuil/models.py (corrigé complet)

Génère automatiquement le `fingerprint` avant `full_clean()`, et le normalise.

```
from django.db import models
import re
import hashlib

def nettoyer_espaces(chaine):
    """Supprime les espaces multiples et trims."""
    if chaine:
        return ' '.join(chaine.split())
    return chaine

# 1. TEMOIGNAGES
class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

    # Anti-doublon (stable, tolérant aux espaces/majuscules)
    fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")

    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ['-created_at']

    @staticmethod
    def _norm_text(s: str) -> str:
        s = (s or "").strip()
        s = re.sub(r"\s+", " ", s)
        return s.lower()

    @classmethod
    def make_fingerprint(cls, nom: str, description: str) -> str:
        base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
        return hashlib.sha256(base.encode("utf-8")).hexdigest()

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)
```

```

def save(self, *args, **kwargs):
    # Génère/normalise le fingerprint avant validation
    if not self.fingerprint:
        self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    else:
        self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

# 2. SPECIALITES
class Projetsfirstspeciality(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description_speciality = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Ma Photo de Spécialisation"
        verbose_name_plural = "Mes Photos de Spécialisation"
        ordering = ['-created_at']

    def __str__(self):
        return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"


# 3. REALISATIONS
class Projetmesrealisations(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    compteur_demo_live = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Projet"
        verbose_name_plural = "Mes Projets"
        ordering = ['-created_at']

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

```

```
def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Projet {self.id}"

# 4. PHOTO DE PROFIL
class Projetphotodeprofil(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "Photo de profil"
        verbose_name_plural = "Photo de profil"

    def __str__(self):
        return f"Photo {self.id}"


# 5. A PROPOS DE MOI
class ProjetAproposDeMoi(models.Model):
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "À propos de moi"
        verbose_name_plural = "À propos de moi"

    def __str__(self):
        return self.description[:50] if self.description else f"A propos {self.id}"


# 6. COMPETENCES
class MesCompetencesCles(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Compétence"
        verbose_name_plural = "Compétences"
        ordering = ['nom']
```

```

def clean(self):
    self.nom = nettoyer_espaces(self.nom)

def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Compétence {self.id}"

# 7. RESEAUX SOCIAUX
class ReseauSocial(models.Model):
    nom = models.CharField(max_length=100, unique=True)
    url = models.CharField(max_length=255)
    est_visible = models.BooleanField(default=True)

    class Meta:
        verbose_name = "Réseau social"
        verbose_name_plural = "Réseaux sociaux"
        ordering = ['nom']

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom

```

---

## 2) app\_acceuil/views.py (corrigé complet)

Remplace le `create()` (qui casse sur unicité) par `update_or_create()` basé sur `fingerprint`.  
C'est exactement l'endroit qui provoquait l'erreur

views

```
.
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib import messages
```

```
from django.db.models import Sum
import re
import hashlib

from app_contact.models import User as ContactUser
from app_user.models import RoleMenuAcces

from .models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
# -----
# Helpers : normalisation + fingerprint (témoignages)
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _fingerprint_projetcards(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def _get_role_id_from_session(request):
    """
    Retourne role_id de l'utilisateur connecté via request.session['user']['id'].
    Sinon None.
    """
    user_session = request.session.get("user")
    if not user_session:
        return None

    user_id = user_session.get("id")
    if not user_id:
        return None

    u = ContactUser.objects.filter(id=user_id).select_related("role").first()
    if not u:
        return None
```

```

    return u.role_id

# =====
# 1. VUE PUBLIQUE (Portfolio)
# =====
def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    """Filtre et affiche uniquement les éléments marqués comme visibles."""
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetsrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemoni = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    mes_menus_visibles = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_visibles = RoleMenuAcces.objects.filter(
            role_id=role_id,
            est_visible=True
        ).select_related("menu")

    contenu = {
        "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
        "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
        "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
        "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
        "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
        "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
        "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
        "A_PROPOS_DE_MOI": aproposdemoni,
        "Mes_Competences_Cles": mescompetencescles,
        "Mes_Reseaux": reseaux,
        "mes_menus_visibles": mes_menus_visibles,
    }
    return render(request, "index.html", contenu)

# =====
# 2. DASHBOARD ADMIN (page_administration.html)
# =====
def page_Mon_Espace_Administration(request):

```

```
user_data = request.session.get("user")
if not user_data or user_data.get("role") != "Administrateur":
    return redirect("index_login")

photo_profil = Projetphotodeprofil.objects.first()
a_propos = ProjetAproposDeMoi.objects.first()
specialite = Projetsfirstspeciality.objects.first()

if request.method == "POST":

    # --- PHOTO DE PROFIL (SWITCH AUTO) ---
    if "btn_visibilite_photo_auto" in request.POST:
        vis = "est_visible" in request.POST
        if photo_profil:
            photo_profil.est_visible = vis
            photo_profil.save()
        messages.info(request, "Visibilité de la photo mise à jour.")

    # --- PHOTO DE PROFIL (UPLOAD/UPDATE) ---
    elif "btn_photo" in request.POST:
        img = request.FILES.get("images")
        if photo_profil:
            if img:
                photo_profil.images = img
                photo_profil.save()
            else:
                Projetphotodeprofil.objects.create(images=img, est_visible=True)
        messages.success(request, "Photo enregistrée.")

    elif "btn_supprimer_photo" in request.POST:
        if photo_profil:
            photo_profil.delete()
        messages.warning(request, "Photo supprimée.")

    # --- À PROPOS (BIO - SWITCH AUTO) ---
    elif "btn_visibilite_bio_auto" in request.POST:
        vis = "est_visible_bio" in request.POST
        if a_propos:
            a_propos.est_visible = vis
            a_propos.save()
        messages.info(request, "Visibilité de la bio mise à jour.")

    # --- À PROPOS (BIO - TEXTE) ---
    elif "btn_apropos" in request.POST:
        description = request.POST.get("description")
        if a_propos:
```

```

        a_propos.description = description
        a_propos.save()
    else:
        ProjetAproposDeMoi.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")

elif "btn_supprimer_apropos" in request.POST:
    if a_propos:
        a_propos.delete()
    messages.error(request, "Texte biographique supprimé.")

# --- SPÉCIALITÉ ---
elif "btn_visibilite_section_spec" in request.POST:
    vis = "est_visible" in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")

elif "btn_specialite" in request.POST:
    img = request.FILES.get("images")
    desc = request.POST.get("description_speciality")
    if specialite:
        specialite.description_speciality = desc
        if img:
            specialite.images = img
        specialite.save()
    else:
        Projetsfirstspeciality.objects.create(description_speciality=desc, images=img, est_visible=True)
    messages.success(request, "Détails de la spécialité mis à jour.")

# --- RÉALISATIONS (PROJETS) ---
elif "btn_visibilite_section_real" in request.POST:
    vis = "est_visible" in request.POST
    Projetmesrealisations.objects.all().update(est_visible=vis)
    messages.info(request, "Section Projets mise à jour.")

elif "btn_bascule_vis_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_real" in request.POST:
    Projetmesrealisations.objects.create(
        nom=request.POST.get("nom"),
        description=request.POST.get("description"),
        images=request.FILES.get("images"),
        est_visible=True,

```

```

)
messages.success(request, "Projet ajouté.")

elif "btn_modifier_real" in request.POST:
    obj = get_object_or_404(Projetsrealisations, id=request.POST.get("id_realisation"))
    obj.nom = request.POST.get("nom")
    obj.description = request.POST.get("description")
    if request.FILES.get("images"):
        obj.images = request.FILES.get("images")
    obj.save()
    messages.info(request, "Projet modifié.")

elif "btn_supprimer_real" in request.POST:
    get_object_or_404(Projetsrealisations, id=request.POST.get("id_realisation")).delete()
    messages.warning(request, "Projet supprimé")

# --- TÉMOIGNAGES ---
elif "btn_visibilite_section_tem" in request.POST:
    vis = "est_visible" in request.POST
    Projetscards.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_tem" in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get("id_temoignage"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_temoignage" in request.POST:
    nom = (request.POST.get("nom") or "").strip()
    description = (request.POST.get("description") or "").strip()
    image = request.FILES.get("images")

    fp = _fingerprint_Projetscards(nom, description)

    obj, created = Projetscards.objects.update_or_create(
        fingerprint=fp,
        defaults={
            "nom": nom,
            "description": description,
            "est_visible": True,
        },
    )

    # si on envoie une image, on écrase; sinon on garde l'ancienne
    if image:
        obj.images = image
        obj.save()

```

```

if created:
    messages.success(request, "Témoignage ajouté.")
else:
    messages.info(request, "Témoignage déjà existant : mise à jour effectuée (anti-doublon).")

elif "btn_supprimer_temoignage" in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get("id_temoignage")).delete()

# --- COMPÉTENCES ---
elif "btn_visibilite_section_comp" in request.POST:
    vis = "est_visible" in request.POST
    MesCompetencesCles.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_comp" in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_comp" in request.POST:
    MesCompetencesCles.objects.create(nom=request.POST.get("nom_competence"), est_visible=True)
    messages.success(request, "Compétence ajoutée.")

elif "btn_supprimer_comp" in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence")).delete()

# --- RÉSEAUX SOCIAUX ---
elif "btn_visibilite_section_res" in request.POST:
    vis = "est_visible" in request.POST
    ReseauSocial.objects.all().update(est_visible=vis)

elif "btn_reseau" in request.POST:
    nom = (request.POST.get("nom_reseau") or "").strip()
    url = (request.POST.get("url_reseau") or "").strip()

    if nom.lower() == "email" and url.lower().startswith("mailto:"):
        url = url.split(":", 1)[1].strip()
    if nom.lower() == "whatsapp" and url and url.startswith("wa.me/"):
        url = "https://" + url

    ReseauSocial.objects.update_or_create(
        nom=nom,
        defaults={"url": url, "est_visible": True},
    )
    messages.success(request, "Réseau mis à jour.")

```

```

        elif "btn_supprimer_reseau" in request.POST:
            get_object_or_404(ReseauSocial, id=request.POST.get("id_reseau")).delete()

        return redirect("dashboard_admin")

context = {
    "AFFICHER_la_PHOTO_DE_PROFIL": photo_profil,
    "A_PROPOS_DE_MOI": a_propos,
    "SPECIALITE": specialite,
    "AFFICHER_les_REALISATIONS_PERSONNELLES": Projetsrealisations.objects.all().order_by("-id"),
    "REALISATIONS_STATS": Projetsrealisations.objects.all().order_by("-compteur_demo_live", "-id"),
    "TOTAL_CLICS_DEMO_LIVE": (Projetsrealisations.objects.aggregate(total=Sum("compteur_demo_live"))["total"] or 0),
    "TEMOIGNAGES_des_PERSONNES_cards": Projetscards.objects.all().order_by("-id"),
    "Mes_Competences_Cles": MesCompetencesCles.objects.all().order_by("nom"),
    "Mes_Reseaux": ReseauSocial.objects.all(),
}
return render(request, "page_administration.html", context)

```

---

### 3) app\_acceuil/json\_transfer.py (complet)

- Export/Import JSON + fusion intelligente + gestion fingerprint (identique à ta dernière version, mais “propre”/cohérente).

```

# (Copie-colle ici EXACTEMENT le contenu complet que je t'ai donné dans ce fichier)
# ➡ Pour éviter de doubler 300 lignes dans ce message, dis-moi si tu veux
#     que je te le remette ici en entier aussi, ou si tu l'as déjà dans ton projet.

```

*(Si tu veux, je te le recolle en entier au prochain message — il est long et ton chat est déjà très chargé.)*

---

### 4) app\_acceuil/urls.py (corrigé complet)

- Ajoute les routes export/import (si tu utilises json\_transfer.py).

```

from django.urls import path
from .views import *
from .json_transfer import export_portfolio_json, export_users_json, import_json_bundle

urlpatterns = [

```

```
# --- Export / Import JSON (Admin) ---
path('export/portfolio', export_portfolio_json, name='export_portfolio_json'),
path('export/users', export_users_json, name='export_users_json'),
path('import/json-bundle', import_json_bundle, name='import_json_bundle'),

# Pages publiques / admin existantes
path("", aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="accueil"),
path("dashboard", page_Mon_Espace_Administration, name="dashboard_admin"),
]
```

---

## Commandes à exécuter (important)

1. Si tu viens d'ajouter `fingerprint` dans le modèle :

```
python manage.py makemigrations
python manage.py migrate
```

2. Si ta base **contient déjà** des témoignages *sans fingerprint*, alors oui : il faut faire un “remplissage 1 fois”.

→ Mais si tu pars sur une nouvelle base vide, tu n'as **pas besoin** de remplir quoi que ce soit (les futurs `save()` généreront le `fingerprint` automatiquement).

---

## Pourquoi ton erreur disparaît

- Ton modèle a une unicité `fingerprint`

models

- Ton view faisait un `create()`

views

✓ Maintenant on fait `update_or_create(fingerprint=...)` → plus de crash, et tu gardes l'anti-doublon.

---

Si tu veux, envoie-moi **ton json\_transfer.py actuel** (fichier réel) et je te le renverrai **100% complet** corrigé comme les autres (sans raccourci).

## Sources

```
from django.shortcuts import render, redirect, get_object_or_404 from django.contrib import messages from django.db.models import F, Sum from .models import ( Projetscards, Projetsfirstspeciality, Projetmesrealisations, Projetphotodeprofil, ProjetAproposDeMoi, MesCompetencesCles, ReseauSocial ) #
Importations des modèles from app_contact.models import User as ContactUser from app_user.models import RoleMenuAcces # // window.location.href = "/";
// Garde l'historique ? Oui // Navigation standard entre les pages. # // On ""RECHARGE COMPLETEMENT"" la ""PAGE D'ACCUEIL"" nommé
"""/index.html"""" afin de ""FAIRE APPARAITRE"""" les ""IMAGES"""" contenues dans le FICHIER """/index.html"""" ::::::::::::::::::::: # /////
window.location.href = "/" // Garde l'historique ? Oui // Navigation standard entre les pages. # https://gemini.google.com/app/3bd97af244610ffd?hl=fr #
https://gemini.google.com/app/3bd97af244610ffd?hl=fr # https://gemini.google.com/app/3bd97af244610ffd?hl=fr #
https://gemini.google.com/app/3bd97af244610ffd?hl=fr # https://gemini.google.com/app/3bd97af244610ffd?hl=fr #
https://gemini.google.com/app/bbba657c1b185397 # https://gemini.google.com/app/bbba657c1b185397 # https://gemini.google.com/app/bbba657c1b185397 #
https://gemini.google.com/app/bbba657c1b185397 # https://gemini.google.com/app/bbba657c1b185397 # https://gemini.google.com/app/1332cca0d76a0b1e #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e def _get_role_id_from_session(request): """
    Retourne
    role_id de l'utilisateur connecté via request.session['user']['id']. Sinon None.
    user_session = request.session.get("user") if not user_session: return None
    user_id = user_session.get("id") if not user_id: return None u = ContactUser.objects.filter(id=user_id).select_related("role").first() if not u: return None return
    u.role_id # ===== # 1. VUE PUBLIQUE (Portfolio) #
===== def
aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request): """
    Filtre et affiche uniquement les éléments marqués comme visibles.
    temoignages =
    Projetscards.objects.filter(est_visible=True) specialites = Projetsfirstspeciality.objects.filter(est_visible=True) realisations =
    Projetmesrealisations.objects.filter(est_visible=True) mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True) reseaux =
    ReseauSocial.objects.filter(est_visible=True) photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first() aproposdemoi =
    ProjetAproposDeMoi.objects.filter(est_visible=True).first() # Menus visibles (par rôle) mes_menus_visibles = [] role_id = _get_role_id_from_session(request)
if role_id: mes_menus_visibles = RoleMenuAcces.objects.filter( role_id=role_id, est_visible=True ).select_related("menu") contenu = {
    "TEMOIGNAGES_des_PERSONNES_cards": temoignages, "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
    "AFFICHER_la_PREMIERE_SPECIAISATION": specialites, "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
    "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations, "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
    "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil, "A_PROPOS_DE_MOI": aproposdemoi, "Mes_Competences_Cles": mescompetencescles,
    "Mes_Reseaux": reseaux, "mes_menus_visibles": mes_menus_visibles, } return render(request, "index.html", contenu) #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e #
https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e #
===== # 2. DASHBOARD D'ADMINISTRATION #
```

```
===== def page_Mon_Espace_Administration(request): #
Récupération des données existantes (pour affichage dans les formulaires) photo_profil = Projetphotodeprofil.objects.first() a_propos =
ProjetAproposDeMoi.objects.first() specialite = Projetsfirstspeciality.objects.first() print(f"page_Mon_Espace_Administration") # DEBUT ======>
⚠ Accès refusé : vous devez être Administrateur. ::::::::::::::::::::: # DEBUT ======> ⚠ Accès refusé : vous devez être Administrateur.
::::::::::::::::::: # DEBUT ======> ⚠ Accès refusé : vous devez être Administrateur. ::::::::::::::::::::: # DEBUT ======> ⚠ Accès
refusé : vous devez être Administrateur. ::::::::::::::::::::: # DEBUT ======> ⚠ Accès refusé : vous devez être Administrateur. ::::::::::::::::::::: #
DEBUT ======> ⚠ Accès refusé : vous devez être Administrateur. ::::::::::::::::::::: # DEBUT ======> ⚠ Accès refusé : vous devez
être Administrateur. ::::::::::::::::::::: # 1. On récupère les données de l'utilisateur (on utilise .get pour éviter une erreur si 'user' n'existe pas) user_data =
request.session.get('user') # https://gemini.google.com/app/5a9a859ff2a2f94e # https://gemini.google.com/app/5a9a859ff2a2f94e #
https://gemini.google.com/app/5a9a859ff2a2f94e # Pourquoi utiliser or au lieu de and ici ? # En programmation, quand on veut "bloquer" l'accès, on utilise
souvent la logique suivante : "Si tu n'as pas de badge OU si ton badge n'est pas le bon, tu sors". # En programmation, quand on veut "bloquer" l'accès, on utilise
souvent la logique suivante : "Si tu n'as pas de badge OU si ton badge n'est pas le bon, tu sors". # En programmation, quand on veut "bloquer" l'accès, on utilise
souvent la logique suivante : "Si tu n'as pas de badge OU si ton badge n'est pas le bon, tu sors". # En programmation, quand on veut "bloquer" l'accès, on utilise
souvent la logique suivante : "Si tu n'as pas de badge OU si ton badge n'est pas le bon, tu sors". # En programmation, quand on veut "bloquer" l'accès, on utilise
souvent la logique suivante : "Si tu n'as pas de badge OU si ton badge n'est pas le bon, tu sors". # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas
'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas 'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas
'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas 'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas
'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas 'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas
'Administrateur' # 2. On vérifie : si la session n'existe pas OU si le rôle n'est pas 'Administrateur' # # Redirection vers l'accueil si l'une des deux conditions est
vraie ## TABLE "*****app_contact_role*****" , COLONNE "*****nom_role*****" , VALEUR = { Administrateur , Moderateur , Utilisateur ,
Responsable de Saisie } ## TABLE "*****app_contact_role*****" , COLONNE "*****nom_role*****" , VALEUR = { Administrateur , Moderateur ,
Utilisateur , Responsable de Saisie } ## TABLE "*****app_contact_role*****" , COLONNE "*****nom_role*****" , VALEUR = { Administrateur ,
Moderateur , Utilisateur , Responsable de Saisie } ## TABLE "*****app_contact_role*****" , COLONNE "*****nom_role*****" , VALEUR = {
Administrateur , Moderateur , Utilisateur , Responsable de Saisie } ## TABLE "*****app_contact_role*****" , COLONNE "*****nom_role*****" ,
VALEUR = { Administrateur , Moderateur , Utilisateur , Responsable de Saisie } ## TABLE "*****app_contact_role*****" , COLONNE
"*****nom_role*****" , VALEUR = { Administrateur , Moderateur , Utilisateur , Responsable de Saisie } #
https://gemini.google.com/app/57bb0c3daec258cd # https://gemini.google.com/app/57bb0c3daec258cd # https://gemini.google.com/app/57bb0c3daec258cd #
https://gemini.google.com/app/57bb0c3daec258cd # https://gemini.google.com/app/57bb0c3daec258cd if not user_data or (user_data.get('role') !=
'Administrateur' and user_data.get('role') != 'Moderateur'): messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.") return
redirect('name_acceuil') # 3. Si on arrive ici, l'utilisateur est connecté ET c'est un admin # Le reste de ton code s'exécute normalement... # FIN
=====> ⚠ Accès refusé : vous devez être Administrateur. !!!!!!!!!!!!!!! # FIN ======> ⚠ Accès refusé : vous devez être
Administrateur. !!!!!!!!!!!!!!! # FIN ======> ⚠ Accès refusé : vous devez être Administrateur. !!!!!!!!!!!!!!! # FIN
=====> ⚠ Accès refusé : vous devez être Administrateur. !!!!!!!!!!!!!!! # FIN ======> ⚠ Accès refusé : vous devez être
Administrateur. !!!!!!!!!!!!!!! # FIN ======> ⚠ Accès refusé : vous devez être Administrateur. !!!!!!!!!!!!!!! # FIN
```

===== > **⚠ Accès refusé : vous devez être Administrateur. !!!!!!!!!!!!!!! else: # L'UTILISATEUR est "CONNECTE" avec son EMAIL" et son "MOT DE PASSE" # On vérifie si l'utilisateur est présent ::::::::::::: # On récupère le dictionnaire user\_data = request.session['user'] # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # On accède aux valeurs spécifiques user\_id = user\_data['id'] # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # https://gemini.google.com/app/1332cca0d76a0b1e # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # nom\_utilisateur = user\_data['nom'] # LECTURE des "VARIABLES SESSIONS" # VARIABLES SESSIONS =====> "request.session['user'] = {'id': nouvel\_utilisateur.id, 'nom': nouvel\_utilisateur.nom}" # =====> Ces "VARIABLES SESSIONS" sont "PRESENTES" dans la FONCTION "def index\_inscription\_view(request):" dans le FICHIER "app\_contact/views.py" # print(f"user\_id de l'UTILISATEUR CONNECTE = {user\_id}") print(f"nom\_utilisateur de l'UTILISATEUR CONNECTE = {nom\_utilisateur}") if request.method == 'POST': # --- PHOTO DE PROFIL (SWITCH AUTO) --- if 'btn\_visibilite\_photo\_auto' in request.POST: vis = 'est visible' in request.POST if photo\_profil: photo\_profil.est\_visible = vis photo\_profil.save()**

```
messages.info(request, "Visibilité de la photo mise à jour.") # --- PHOTO DE PROFIL (UPLOAD/UPDATE) --- elif 'btn_photo' in request.POST: img = request.FILES.get('images') if photo_profil: if img: photo_profil.images = img photo_profil.save() else: Projetphotodeprofil.objects.create(images=img, est_visible=True) messages.success(request, "Photo enregistrée.") elif 'btn_supprimer_photo' in request.POST: if photo_profil: photo_profil.delete() messages.warning(request, "Photo supprimée.") # --- À PROPOS (BIO - SWITCH AUTO) --- elif 'btn_visibilite_bio_auto' in request.POST: vis = 'est_visible_bio' in request.POST if a_propos: a_propos.est_visible = vis a_propos.save() print(f"vis = {vis}") # vis = False # vis = True messages.info(request, "Visibilité de la bio mise à jour.") # --- À PROPOS (BIO - TEXTE) --- elif 'btn_apropos' in request.POST: description = request.POST.get('description') if a_propos: a_propos.description = description a_propos.save() else: ProjetaProposDeMoi.objects.create(description=description, est_visible=True) messages.success(request, "Texte de la bio mis à jour.") elif 'btn_supprimer_apropos' in request.POST: if a_propos: a_propos.delete() messages.error(request, "Texte biographique supprimé.") # --- SPÉCIALITÉ --- elif 'btn_visibilite_section_spec' in request.POST: vis = 'est_visible' in request.POST Projetsfirstspeciality.objects.all().update(est_visible=vis) messages.info(request, "Visibilité section Spécialité modifiée.") elif 'btn_specialite' in request.POST: img = request.FILES.get('images') desc = request.POST.get('description_speciality') if specialite: specialite.description_speciality = desc if img: specialite.images = img specialite.save() else: Projetsfirstspeciality.objects.create(description_speciality=desc, images=img, est_visible=True) messages.success(request, "Détails de la spécialité mis à jour.") # --- RÉALISATIONS (PROJETS) --- elif 'btn_visibilite_section_real' in request.POST: vis = 'est_visible' in request.POST Projetsrealisations.objects.all().update(est_visible=vis) messages.info(request, "Section Projets mise à jour.") elif 'btn_bascule_vis_real' in request.POST: obj = get_object_or_404(Projetsrealisations, id=request.POST.get('id_realisation')) obj.est_visible = not obj.est_visible obj.save() elif 'btn_ajouter_real' in request.POST: Projetsrealisations.objects.create( nom=request.POST.get('nom'), description=request.POST.get('description'), images=request.FILES.get('images'), est_visible=True ) messages.success(request, "Projet ajouté.") elif 'btn_modifier_real' in request.POST: obj = get_object_or_404(Projetsrealisations, id=request.POST.get('id_realisation')) obj.nom = request.POST.get('nom') obj.description = request.POST.get('description') if request.FILES.get('images'): obj.images = request.FILES.get('images') obj.save() messages.info(request, "Projet modifié.") elif 'btn_supprimer_real' in request.POST: get_object_or_404(Projetsrealisations, id=request.POST.get('id_realisation')).delete() messages.warning(request, "Projet supprimé.") # --- TÉMOIGNAGES --- elif 'btn_visibilite_section_tem' in request.POST: vis = 'est_visible' in request.POST Projetcards.objects.all().update(est_visible=vis) elif 'btn_bascule_vis_tem' in request.POST: obj = get_object_or_404(Projetcards, id=request.POST.get('id_temoignage')) obj.est_visible = not obj.est_visible obj.save() elif 'btn_ajouter_temoignage' in request.POST: Projetcards.objects.create( nom=request.POST.get('nom'), description=request.POST.get('description'), images=request.FILES.get('images'), est_visible=True ) messages.success(request, "Témoignage ajouté.") elif 'btn_supprimer_temoignage' in request.POST: get_object_or_404(Projetcards, id=request.POST.get('id_temoignage')).delete() # --- COMPÉTENCES --- elif 'btn_visibilite_section_comp' in request.POST: vis = 'est_visible' in request.POST MesCompetencesCles.objects.all().update(est_visible=vis) elif 'btn_bascule_vis_comp' in request.POST: obj = get_object_or_404(MesCompetencesCles, id=request.POST.get('id_competence')) obj.est_visible = not obj.est_visible obj.save() elif 'btn_ajouter_comp' in request.POST: MesCompetencesCles.objects.create(nom=request.POST.get('nom_competence'), est_visible=True) messages.success(request, "Compétence ajoutée.") elif 'btn_supprimer_comp' in request.POST: get_object_or_404(MesCompetencesCles, id=request.POST.get('id_competence')).delete() # --- RÉSEAUX SOCIAUX --- elif 'btn_visibilite_section_res' in request.POST: vis = 'est_visible' in request.POST ReseauSocial.objects.all().update(est_visible=vis) elif 'btn_reseau' in request.POST: nom = (request.POST.get('nom_reseau') or "").strip() url = (request.POST.get('url_reseau') or "").strip() # Petites normalisations pour éviter les doublons "mailto:" / liens WhatsApp incomplets if nom.lower() == 'email' and url.lower().startswith('mailto:'): url = url.split(':', 1)[1].strip() if nom.lower() == 'whatsapp' and url and url.startswith('wa.me/'): url = 'https://' + url ReseauSocial.objects.update_or_create( nom=nom, defaults={'url': url, 'est_visible': True} ) messages.success(request, "Réseau mis à jour.") elif 'btn_supprimer_reseau' in request.POST: get_object_or_404(ReseauSocial, id=request.POST.get('id_reseau')).delete() # Redirection finale return redirect('dashboard_admin') # Contexte context = { 'AFFICHER_la_PHOTO_DE_PROFIL': photo_profil, 'A_PROPOS_DE_MOI': a_propos, 'SPECIALITE':
```

```

specialite, 'AFFICHER_les_REALISATIONS_PERSONNELLES': Projetsrealisations.objects.all().order_by('-id'), 'REALISATIONS_STATS':
Projetsrealisations.objects.all().order_by('-compteur_demo_live', '-id'), 'TOTAL_CLICS_DEMO_LIVE':
(Projetsrealisations.objects.aggregate(total=Sum('compteur_demo_live'))['total'] or 0), 'TEMOIGNAGES_des_PERSONNES_cards':
Projetscards.objects.all().order_by('-id'), 'Mes_Competences_Cles': MesCompetencesCles.objects.all().order_by('nom'), 'Mes_Reseaux':
ReseauSocial.objects.all(), } return render(request, 'page_administration.html', context) #
===== # 2bis. COMPTEUR DE CLICS "DÉMO LIVE" PAR
PROJET # ===== def
incrementer_compteur_demo_live(request, realisation_id): """Incrémente le compteur quand un visiteur clique sur 'Démo Live' puis redirige vers la démo.""" #
Incrément atomique (évite les soucis si plusieurs clics en même temps) Projetsrealisations.objects.filter(id=realisation_id).update(
compteur_demo_live=F('compteur_demo_live') + 1 ) # On redirige vers la page de démo existante (actuellement la page Camions) return
redirect('name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES') #
===== # 3. REDIRECTIONS #
===== def Fonction_Page_accueil_section_Accueil(request):
return redirect('/#accueil') def Fonction_Page_accueil_section_projets(request): return redirect('/#projets') def
Fonction_Page_accueil_section_temoignages(request): return redirect('/#temoignages') def Fonction_Page_accueil_section_a_propos(request): return
redirect('/#a-propos') def Fonction_Page_accueil_section_contact(request): return redirect('/#contact') # def Fonction_Page_email_Javascript(request): return
redirect('/#email_JAVASCRIPT') def Fonction_Page_email_Javascript(request): return render(request, 'email_Javascript.htm') def
Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request): return render(request, 'TransportRoutier_Main.html') # nouveau : def
Fonction_Page_diapo_Javascript_Page1(request): return render(request, 'diapo_Javascript_Page1.html') # def reinitialiser_compteur_projet(request): #
Projetsrealisations.objects.all().update(compteur_demo_live=0) # messages.success(request, "Compteurs réinitialisés.") # return
redirect("dashboard_admin") # ===== # RÉINITIALISATION
DU COMPTEUR POUR UN PROJET SPÉCIFIQUE #
===== def reinitialiser_compteur_projet(request, realisation_id): """Remet à zéro le compteur de clics pour un projet donné.""" #
On récupère le projet ou on renvoie une erreur 404 projet = get_object_or_404(Projetsrealisations, id=realisation_id) # Remise à zéro projet.compteur_demo_live = 0 projet.save()
messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.") return redirect('dashboard_admin')
Réflexion durant 11s

```

Parfait — ton `views.py` est **quasiment bon**, mais il te manque la modification clé dans `btn_ajouter_temoignage` : tu fais encore un `Projetscards.objects.create(...)`, donc dès qu'un témoignage “identique” (même fingerprint) existe déjà, Django bloque avec :

```
ValidationError: {'fingerprint': [...already exists.]}
```

Voici ton fichier `app_accueil/views.py` complet corrigé, en gardant ton style + tes routes + ton contrôle d'accès Admin/Modérateur, et en ajoutant uniquement ce qu'il faut (`fingerprint + update_or_create`).

---



## app\_acceuil/views.py (COMPLET corrigé)

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.db.models import F, Sum
import re
import hashlib

from .models import (
    Projetscards, Projetsfirstspeciality, Projetmesrealisations,
    Projetphotodeprofil, ProjetAproposDeMoi, MesCompetencesCles, ReseauSocial
)

# Importations des modèles
from app_contact.models import User as ContactUser
from app_user.models import RoleMenuAcces

# -----
# Helpers : fingerprint Projetscards
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # \n \t espaces multiples => 1 espace
    return s.lower()

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def _get_role_id_from_session(request):
    """
    Retourne role_id de l'utilisateur connecté via request.session['user']['id'].
    Sinon None.
    """
    user_session = request.session.get("user")
    if not user_session:
        return None

    user_id = user_session.get("id")
    if not user_id:
        return None
```

```

u = ContactUser.objects.filter(id=user_id).select_related("role").first()
if not u:
    return None

return u.role_id

# =====
# 1. VUE PUBLIQUE (Portfolio)
# =====
def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    """Filtre et affiche uniquement les éléments marqués comme visibles."""
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetsrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemonoi = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    # Menus visibles (par rôle)
    mes_menus_visibles = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_visibles = RoleMenuAcces.objects.filter(
            role_id=role_id,
            est_visible=True
        ).select_related("menu")

    contenu = {
        "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
        "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
        "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
        "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
        "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
        "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
        "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
        "A_PROPOS_DE_MOI": aproposdemonoi,
        "Mes_Competences_Cles": mescompetencescles,
        "Mes_Reseaux": reseaux,
        "mes_menus_visibles": mes_menus_visibles,
    }
    return render(request, "index.html", contenu)

```

```
# =====
# 2. DASHBOARD D'ADMINISTRATION
# =====
def page_Mon_Espace_Administration(request):
    # Récupération des données existantes (pour affichage dans les formulaires)
    photo_profil = Projetphotodeprofil.objects.first()
    a_propos = ProjetAproposDeMoi.objects.first()
    specialite = Projetsfirstspeciality.objects.first()

    print(f"page_Mon_Espace_Administration")

    # --- Contrôle d'accès (Admin OU Modérateur) ---
    user_data = request.session.get('user')
    if not user_data or (user_data.get('role') != 'Administrateur' and user_data.get('role') != 'Moderateur'):
        messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.")
        return redirect('name_acceuil')

    # Ici l'utilisateur est autorisé
    user_id = user_data.get('id')
    nom_utilisateur = user_data.get('nom')
    print(f"user_id de l'UTILISATEUR CONNECTE = {user_id}")
    print(f"nom_utilisateur de l'UTILISATEUR CONNECTE = {nom_utilisateur}")

    if request.method == 'POST':

        # --- PHOTO DE PROFIL (SWITCH AUTO) ---
        if 'btn_visibilite_photo_auto' in request.POST:
            vis = 'est_visible' in request.POST
            if photo_profil:
                photo_profil.est_visible = vis
                photo_profil.save()
            messages.info(request, "Visibilité de la photo mise à jour.")

        # --- PHOTO DE PROFIL (UPLOAD/UPDATE) ---
        elif 'btn_photo' in request.POST:
            img = request.FILES.get('images')
            if photo_profil:
                if img:
                    photo_profil.images = img
                    photo_profil.save()
            else:
                Projetphotodeprofil.objects.create(images=img, est_visible=True)
            messages.success(request, "Photo enregistrée.")

        elif 'btn_supprimer_photo' in request.POST:
            if photo_profil:
```

```
    photo_profil.delete()
messages.warning(request, "Photo supprimée.")

# --- À PROPOS (BIO - SWITCH AUTO) ---
elif 'btn_visibilite_bio_auto' in request.POST:
    vis = 'est_visible_bio' in request.POST
    if a_propos:
        a_propos.est_visible = vis
        a_propos.save()

    print(f"vis = {vis}") # vis = False / True
    messages.info(request, "Visibilité de la bio mise à jour.")

# --- À PROPOS (BIO - TEXTE) ---
elif 'btn_apropos' in request.POST:
    description = request.POST.get('description')
    if a_propos:
        a_propos.description = description
        a_propos.save()
    else:
        ProjetAproposDeMoi.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")

elif 'btn_supprimer_apropos' in request.POST:
    if a_propos:
        a_propos.delete()
    messages.error(request, "Texte biographique supprimé.")

# --- SPÉCIALITÉ ---
elif 'btn_visibilite_section_spec' in request.POST:
    vis = 'est_visible' in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")

elif 'btn_specialite' in request.POST:
    img = request.FILES.get('images')
    desc = request.POST.get('description_speciality')
    if specialite:
        specialite.description_speciality = desc
        if img:
            specialite.images = img
        specialite.save()
    else:
        Projetsfirstspeciality.objects.create(description_speciality=desc, images=img, est_visible=True)
    messages.success(request, "Détails de la spécialité mis à jour.")
```

```

# --- RÉALISATIONS (PROJETS) ---
elif 'btn_visibilite_section_real' in request.POST:
    vis = 'est_visible' in request.POST
    Projetmesrealisations.objects.all().update(est_visible=vis)
    messages.info(request, "Section Projets mise à jour.")

elif 'btn_bascule_vis_real' in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get('id_realisation'))
    obj.est_visible = not obj.est_visible
    obj.save()

elif 'btn_ajouter_real' in request.POST:
    Projetmesrealisations.objects.create(
        nom=request.POST.get('nom'),
        description=request.POST.get('description'),
        images=request.FILES.get('images'),
        est_visible=True
    )
    messages.success(request, "Projet ajouté.")

elif 'btn_modifier_real' in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get('id_realisation'))
    obj.nom = request.POST.get('nom')
    obj.description = request.POST.get('description')
    if request.FILES.get('images'):
        obj.images = request.FILES.get('images')
    obj.save()
    messages.info(request, "Projet modifié.")

elif 'btn_supprimer_real' in request.POST:
    get_object_or_404(Projetmesrealisations, id=request.POST.get('id_realisation')).delete()
    messages.warning(request, "Projet supprimé.")

# --- TÉMOIGNAGES ---
elif 'btn_visibilite_section_tem' in request.POST:
    vis = 'est_visible' in request.POST
    Projetscards.objects.all().update(est_visible=vis)

elif 'btn_bascule_vis_tem' in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get('id_temoignage'))
    obj.est_visible = not obj.est_visible
    obj.save()

elif 'btn_ajouter_temoignage' in request.POST:
    # ✅ CORRECTION : anti-doublon fingerprint + update_or_create

```

```

nom = (request.POST.get('nom') or '').strip()
description = (request.POST.get('description') or '').strip()
image = request.FILES.get('images')

fp = _fingerprint_projetscards(nom, description)

obj, created = Projetscards.objects.update_or_create(
    fingerprint=fp,
    defaults={
        "nom": nom,
        "description": description,
        "est_visible": True,
    }
)

# si une image est envoyée, on la met/écrase ; sinon on garde l'ancienne
if image:
    obj.images = image
    obj.save()

if created:
    messages.success(request, "Témoignage ajouté.")
else:
    messages.info(request, "Témoignage déjà existant : mise à jour effectuée (anti-doublon.)")

elif 'btn_supprimer_temoignage' in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get('id_temoignage')).delete()

# --- COMPÉTENCES ---
elif 'btn_visibilite_section_comp' in request.POST:
    vis = 'est_visible' in request.POST
    MesCompetencesCles.objects.all().update(est_visible=vis)

elif 'btn_bascule_vis_comp' in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get('id_competence'))
    obj.est_visible = not obj.est_visible
    obj.save()

elif 'btn_ajouter_comp' in request.POST:
    MesCompetencesCles.objects.create(nom=request.POST.get('nom_competence'), est_visible=True)
    messages.success(request, "Compétence ajoutée.")

elif 'btn_supprimer_comp' in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get('id_competence')).delete()

# --- RÉSEAUX SOCIAUX ---

```

```

    elif 'btn_visibilite_section_res' in request.POST:
        vis = 'est_visible' in request.POST
        ReseauSocial.objects.all().update(est_visible=vis)

    elif 'btn_reseau' in request.POST:
        nom = (request.POST.get('nom_reseau') or '').strip()
        url = (request.POST.get('url_reseau') or '').strip()

        # Petites normalisations
        if nom.lower() == 'email' and url.lower().startswith('mailto:'):
            url = url.split(':', 1)[1].strip()
        if nom.lower() == 'whatsapp' and url and url.startswith('wa.me/'):
            url = 'https://' + url

        ReseauSocial.objects.update_or_create(
            nom=nom,
            defaults={'url': url, 'est_visible': True}
        )
        messages.success(request, "Réseau mis à jour.")

    elif 'btn_supprimer_reseau' in request.POST:
        get_object_or_404(ReseauSocial, id=request.POST.get('id_reseau')).delete()

    # Redirection finale
    return redirect('dashboard_admin')

# Contexte
context = {
    'AFFICHER_la_PHOTO_DE_PROFIL': photo_profil,
    'A_PROPOS_DE_MOI': a_propos,
    'SPECIALITE': specialite,
    'AFFICHER_les_REALISATIONS_PERSONNELLES': Projetsrealisations.objects.all().order_by('-id'),
    'REALISATIONS_STATS': Projetsrealisations.objects.all().order_by('-compteur_demo_live', '-id'),
    'TOTAL_CLICS_DEMO_LIVE': (Projetsrealisations.objects.aggregate(total=Sum('compteur_demo_live'))['total'] or 0),
    'TEMOIGNAGES_des_PERSONNES_cards': Projetscards.objects.all().order_by('-id'),
    'Mes_Competences_Cles': MesCompetencesCles.objects.all().order_by('nom'),
    'Mes_Reseaux': ReseauSocial.objects.all(),
}
return render(request, 'page_administration.html', context)

# =====
# 2bis. COMPTEUR DE CLICS "DÉMO LIVE" PAR PROJET
# =====
def incrementer_compteur_demo_live(request, realisation_id):
    """Incrémente le compteur quand un visiteur clique sur 'Démo Live' puis redirige vers la démo."""

```

```

Projetmesrealisations.objects.filter(id=realisation_id).update(
    compteur_demo_live=F('compteur_demo_live') + 1
)
return redirect('name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES')

# =====
# 3. REDIRECTIONS
# =====
def Fonction_Page_accueil_section_Accueil(request): return redirect('/#accueil')
def Fonction_Page_accueil_section_projets(request): return redirect('/#projets')
def Fonction_Page_accueil_section_temoignages(request): return redirect('/#temoignages')
def Fonction_Page_accueil_section_a_propos(request): return redirect('/#a-propos')
def Fonction_Page_accueil_section_contact(request): return redirect('/#contact')

def Fonction_Page_email_Javascript(request): return render(request, 'email_Javascript.htm')
def Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request): return render(request, 'TransportRoutier_Main.html')
def Fonction_Page_diapo_Javascript_Page1(request): return render(request, 'diapo_Javascript_Page1.html')

# =====
# RÉINITIALISATION DU COMPTEUR POUR UN PROJET SPÉCIFIQUE
# =====
def reinitialiser_compteur_projet(request, realisation_id):
    """Remet à zéro le compteur de clics pour un projet donné."""
    projet = get_object_or_404(Projetmesrealisations, id=realisation_id)
    projet.compteur_demo_live = 0
    projet.save()
    messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.")
    return redirect('dashboard_admin')

```

---

## Ce que tu dois faire ensuite (indispensable)

1. Vérifie que ton modèle Projetcards a bien :

- fingerprint = models.CharField(max\_length=64, unique=True, db\_index=True, ...)

2. Puis :

```

python manage.py makemigrations
python manage.py migrate

```

# Important (pour éviter d'autres erreurs)

 Avec ce `views.py` corrigé :

- si le témoignage n'existe pas → il est **ajouté**
- si le témoignage existe déjà (même nom+description après normalisation) → il est **mis à jour** (et éventuellement image remplacée si tu en envoies une)

Donc tu n'auras **plus** la `ValidationError`.

---

Si tu veux, colle-moi maintenant ton `app_acceuil/models.py` complet actuel (ou juste la classe `Projetscards`) et je te renverrai aussi **le modèle complet final** parfaitement cohérent avec ce `views.py`.

```
from django.db import models
import re
import hashlib
def nettoyer_espaces(chaine):
    """Supprime les espaces multiples et trims."""
    if chaine:
        return ' '.join(chaine.split())
    return chaine

# 1. TEMOIGNAGES class Projetscards(models.Model):
#     nom = models.CharField(max_length=100, null=True, blank=False)
#     images = models.ImageField(upload_to="static/images/", null=True, blank=False)
#     description = models.TextField(null=True, blank=False)
#     # Anti-doublon (stable, tolérant aux espaces/majuscules)
#     fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
#     est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
#     created_at = models.DateTimeField(auto_now_add=True, null=True)

#     class Meta:
#         verbose_name = "Témoignage"
#         verbose_name_plural = "Témoignages Clients"
#         ordering = ['-created_at']

#     @staticmethod
#     def _norm_text(s: str) -> str:
#         s = (s or "").strip()
#         s = re.sub(r"\s+", " ", s)
#         return s.lower()

#     @classmethod
#     def make_fingerprint(cls, nom: str, description: str) -> str:
#         base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
#         return hashlib.sha256(base.encode("utf-8")).hexdigest()

#     def clean(self):
#         self.nom = nettoyer_espaces(self.nom)
#         self.description = nettoyer_espaces(self.description)

#     def save(self, *args, **kwargs):
#         # Génère/normalise le fingerprint avant validation
#         if not self.fingerprint:
#             self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
#         else:
#             self.fingerprint = (self.fingerprint or "").strip().lower()
#         self.full_clean()
#         super().save(*args, **kwargs)

#     def __str__(self):
#         return self.nom if self.nom else f"Témoignage {self.id}"

# 2. SPECIALITES class Projetsfirstspeciality(models.Model):
#     images = models.ImageField(upload_to="static/images/", null=True, blank=False)
#     description_speciality = models.TextField(null=True, blank=False)
#     est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
#     created_at = models.DateTimeField(auto_now_add=True, null=True)

#     class Meta:
#         verbose_name = "Ma Photo de Spécialisation"
#         verbose_name_plural = "Mes Photos de Spécialisation"
#         ordering = ['-created_at']

#     def __str__(self):
#         return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"

# 3. REALISATIONS class Projetsmesrealisations(models.Model):
#     nom = models.CharField(max_length=100, null=True, blank=False)
#     images = models.ImageField(upload_to="static/images/", null=True, blank=False)
#     description = models.TextField(null=True, blank=False)
#     est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
#     compteur_demo_live = models.IntegerField(default=0)
#     created_at = models.DateTimeField(auto_now_add=True, null=True)

#     class Meta:
#         verbose_name = "Projet"
#         verbose_name_plural = "Mes Projets"
#         ordering = ['-created_at']

#     def clean(self):
#         self.nom = nettoyer_espaces(self.nom)
#         self.description = nettoyer_espaces(self.description)

#     def save(self, *args, **kwargs):
#         self.full_clean()
#         super().save(*args, **kwargs)

#     def __str__(self):
#         return self.nom if self.nom else f"Projet {self.id}"

# 4. PHOTO DE PROFIL class
```

```
Projetphotodeprofil(models.Model): images = models.ImageField(upload_to="static/images/", null=True, blank=False) est_visible =  
models.BooleanField(default=True, verbose_name="Afficher sur le site") class Meta: verbose_name = "Photo de profil" verbose_name_plural = "Photo de  
profil" def __str__(self): return f"Photo {self.id}" # 5. A PROPOS DE MOI class ProjetAproposDeMoi(models.Model): description =  
models.TextField(null=True, blank=False) est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site") class Meta: verbose_name =  
"À propos de moi" verbose_name_plural = "À propos de moi" def __str__(self): return self.description[:50] if self.description else f"A propos {self.id}" # 6.  
COMPETENCES class MesCompetencesCles(models.Model): nom = models.CharField(max_length=100, null=True, blank=False) est_visible =  
models.BooleanField(default=True, verbose_name="Afficher sur le site") created_at = models.DateTimeField(auto_now_add=True, null=True) class Meta:  
verbose_name = "Compétence" verbose_name_plural = "Compétences" ordering = ['nom'] def clean(self): self.nom = nettoyer_espaces(self.nom) def save(self,  
*args, **kwargs): self.full_clean() super().save(*args, **kwargs) def __str__(self): return self.nom if self.nom else f"Compétence {self.id}" # 7. RESEAUX  
SOCIAUX class ReseauSocial(models.Model): nom = models.CharField(max_length=100, unique=True) url = models.CharField(max_length=255) est_visible =  
models.BooleanField(default=True) class Meta: verbose_name = "Réseau social" verbose_name_plural = "Réseaux sociaux" ordering = ['nom'] def  
clean(self): self.nom = nettoyer_espaces(self.nom) def save(self, *args, **kwargs): self.full_clean() super().save(*args, **kwargs) def __str__(self): return  
self.nom
```

Ton `models.py` est **bon** sur le principe (`fingerprint` unique + calcul dans `save()`), mais tu as un point qui explique **exactement** ton erreur actuelle :

### Pourquoi tu as `Fingerprint already exists`

Quand tu ajoutes un témoignage via le dashboard, ton code faisait un `Projetscards.objects.create(...)` sans gérer l'existant.  
Du coup, au moment du `save()` (où tu fais `self.full_clean()`), Django voit que le `fingerprint` calculé existe déjà → **ValidationError**.

La solution sûre, c'est :

- garder ton modèle tel quel (ou avec une mini amélioration ci-dessous)
- et côté `views.py`, faire `update_or_create(fingerprint=..., defaults=...)` (comme je t'ai donné).

---

## Le modèle est-il “sûr” ?

Oui à **condition** que tu respectes ces 2 règles :

1. **Ne jamais faire `create()` pour `Projetscards`** si tu veux éviter les doublons → toujours `update_or_create()` (ou `get_or_create()`).
2. Ton `fingerprint` ne doit **pas rester vide** pour plusieurs lignes.  
Actuellement tu as `blank=True, default=""`.  
Ça marche si ton `save()` le remplit toujours, mais c'est plus sûr de **ne pas autoriser le vide**.

## Modèle Projetscards (version plus sûre)

Voici la **version recommandée** (petits changements seulement) :

- blank=False
- default supprimé (ou default=None si tu veux)
- calcul fingerprint seulement si nom+description présents
- clean() normalise aussi description (optionnel mais utile)

```
class Projetscards(models.Model):  
    nom = models.CharField(max_length=100, null=True, blank=False)  
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)  
    description = models.TextField(null=True, blank=False)  
  
    fingerprint = models.CharField(  
        max_length=64,  
        unique=True,  
        db_index=True,  
        blank=False,  
        null=False,  
    )  
  
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")  
    created_at = models.DateTimeField(auto_now_add=True, null=True)  
  
    class Meta:  
        verbose_name = "Témoignage"  
        verbose_name_plural = "Témoignages Clients"  
        ordering = ['-created_at']  
  
    @staticmethod  
    def _norm_text(s: str) -> str:  
        s = (s or "").strip()  
        s = re.sub(r"\s+", " ", s)  
        return s.lower()  
  
    @classmethod  
    def make_fingerprint(cls, nom: str, description: str) -> str:  
        base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"  
        return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

```

def clean(self):
    self.nom = nettoyer_espaces(self.nom)
    self.description = nettoyer_espaces(self.description)

def save(self, *args, **kwargs):
    # Génère/normalise le fingerprint avant validation
    if not self.fingerprint:
        self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    else:
        self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

```

Si tu as déjà des données en base avec `fingerprint=""`, fais une migration + un petit script pour remplir (je peux te donner le script).

---

## Ce que tu dois faire dans ton `views.py` (obligatoire)

Dans `btn_ajouter_temoignage` il faut **impérativement** remplacer `create()` par ça :

```

nom = (request.POST.get('nom') or '').strip()
description = (request.POST.get('description') or '').strip()
image = request.FILES.get('images')

fp = Projetscards.make_fingerprint(nom, description)

obj, created = Projetscards.objects.update_or_create(
    fingerprint=fp,
    defaults={"nom": nom, "description": description, "est_visible": True}
)

if image:
    obj.images = image
    obj.save()

```

 Résultat :

- s'il n'existe pas → créé
  - s'il existe → update (et plus jamais de ValidationError)
- 

## Et si tu utilises une nouvelle base de données ?

Si tu pars de zéro : **NON**, tu n'as pas besoin du “remplir fingerprint pour l'existant” (puisque il n'y a rien d'existant).

Tu en auras besoin **uniquement** si :

- tu ajoutes `fingerprint` après coup
  - ou tu importes des anciens témoignages sans fingerprint
- 

Si tu veux, je te donne maintenant **le fichier `app_acceuil/models.py` complet final** (toutes les classes) avec uniquement des corrections “sûres” + compatible avec ton import/export JSON.

oui

Parfait. Voici ton fichier **`app_acceuil/models.py` complet**, en version **sûre et cohérente** avec :

- `fingerprint` anti-doublon pour Projetscards
- `normalisation nom + description`
- pas de `fingerprint` vide (évite les comportements bizarres)
- garde tes champs et ton style (SQLite OK)
- compatible avec l'import JSON (le `fingerprint` sera recalculé si absent)

Après avoir remplacé ton fichier par celui-ci :

```
python manage.py makemigrations puis python manage.py migrate
```

---



**`app_acceuil/models.py` (COMPLET corrigé)**

```
from django.db import models
import re
import hashlib

def nettoyer_espaces(chaine):
    """Supprime les espaces multiples et trims."""
    if chaine is None:
        return chaine
    return ' '.join(str(chaine).split())

# 1. TEMOIGNAGES
class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

    # Anti-doublon stable (tolérant aux espaces/majuscules)
    # IMPORTANT : pas de default="" pour éviter "plusieurs lignes vides"
    fingerprint = models.CharField(
        max_length=64,
        unique=True,
        db_index=True,
        null=False,
        blank=False,
    )

    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ['-created_at']

    @staticmethod
    def _norm_text(s: str) -> str:
        s = (s or "").strip()
        s = re.sub(r"\s+", " ", s)
        return s.lower()

    @classmethod
    def make_fingerprint(cls, nom: str, description: str) -> str:
        base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
        return hashlib.sha256(base.encode("utf-8")).hexdigest()
```

```

def clean(self):
    # normalisation légère pour réduire les faux doublons
    self.nom = nettoyer_espaces(self.nom)
    self.description = nettoyer_espaces(self.description)

def save(self, *args, **kwargs):
    # Génère/normalise le fingerprint avant validation
    if not self.fingerprint:
        self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    else:
        self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

# 2. SPECIALITES
class Projetsfirstspeciality(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description_speciality = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Ma Photo de Spécialisation"
        verbose_name_plural = "Mes Photos de Spécialisation"
        ordering = ['-created_at']

    def __str__(self):
        return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"

# 3. REALISATIONS
class Projetmesrealisations(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    compteur_demo_live = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:

```

```
verbose_name = "Projet"
verbose_name_plural = "Mes Projets"
ordering = ['-created_at']

def clean(self):
    self.nom = nettoyer_espaces(self.nom)
    self.description = nettoyer_espaces(self.description)

def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Projet {self.id}"

# 4. PHOTO DE PROFIL
class Projetphotodeprofil(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "Photo de profil"
        verbose_name_plural = "Photos de profil"

    def __str__(self):
        return f"Photo {self.id}"

# 5. A PROPOS DE MOI
class ProjetAproposDeMoi(models.Model):
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "À propos de moi"
        verbose_name_plural = "À propos de moi"

    def clean(self):
        self.description = nettoyer_espaces(self.description)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
```

```
        return self.description[:50] if self.description else f"A propos {self.id}"\n\n# 6. COMPETENCES\nclass MesCompetencesCles(models.Model):\n    nom = models.CharField(max_length=100, null=True, blank=False)\n    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")\n    created_at = models.DateTimeField(auto_now_add=True, null=True)\n\n    class Meta:\n        verbose_name = "Compétence"\n        verbose_name_plural = "Compétences"\n        ordering = ['nom']\n\n    def clean(self):\n        self.nom = nettoyer_espaces(self.nom)\n\n    def save(self, *args, **kwargs):\n        self.full_clean()\n        super().save(*args, **kwargs)\n\n    def __str__(self):\n        return self.nom if self.nom else f"Compétence {self.id}"\n\n# 7. RESEAUX SOCIAUX\nclass ReseauSocial(models.Model):\n    nom = models.CharField(max_length=100, unique=True)\n    url = models.CharField(max_length=255)\n    est_visible = models.BooleanField(default=True)\n\n    class Meta:\n        verbose_name = "Réseau social"\n        verbose_name_plural = "Réseaux sociaux"\n        ordering = ['nom']\n\n    def clean(self):\n        self.nom = nettoyer_espaces(self.nom)\n        self.url = (self.url or "").strip()\n\n    def save(self, *args, **kwargs):\n        self.full_clean()\n        super().save(*args, **kwargs)\n\n    def __str__(self):\n        return self.nom
```

## ⚠️ Important : une migration peut échouer si tu as déjà des lignes sans fingerprint

Si tu as déjà des témoignages dans ta base actuelle, la migration qui rend `fingerprint` non vide peut poser problème **si certains enregistrements ont `fingerprint=""` ou `NULL`.**

Si tu me dis juste :

“j’ai déjà des témoignages dans SQLite” **oui/non**,  
je te donne **le script de remplissage automatique** à exécuter une fois avant de migrer (ou après selon ta config).

Le code complet de `views.py`

Réflexion durant 11s

# app\_acceuil/views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.db.models import F, Sum

# Models Portfolio
from .models import (
    Projetscards,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    MesCompetencesCles,
    ReseauSocial,
)

# Autres apps (menus + users)
from app_contact.models import User as ContactUser
from app_user.models import RoleMenuAcces

# =====
# Helpers (menus par rôle)
# =====
def _get_role_id_from_session(request):
    """
    Retourne role_id de l'utilisateur connecté via request.session['user']['id'].
    Sinon None.
    """
```

```

"""
user_session = request.session.get("user")
if not user_session:
    return None

user_id = user_session.get("id")
if not user_id:
    return None

u = ContactUser.objects.filter(id=user_id).select_related("role").first()
if not u:
    return None

return u.role_id


# =====
# 1) VUE PUBLIQUE (Portfolio)
# =====

def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    """Filtre et affiche uniquement les éléments marqués comme visibles."""
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetsrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemonoi = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    # Menus visibles (par rôle)
    mes_menus_visibles = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_visibles = RoleMenuAcces.objects.filter(
            role_id=role_id,
            est_visible=True
        ).select_related("menu")

    contenu = {
        "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
        "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
        "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
        "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
        "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
        "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
    }

```

```

    "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
    "A_PROPOS_DE_MOI": aproposdemoi,
    "Mes_Competences_Cles": mescompetencescles,
    "Mes_Reseaux": reseaux,
    "mes_menus_visibles": mes_menus_visibles,
}
return render(request, "index.html", contenu)

# =====
# 2) DASHBOARD D'ADMINISTRATION
# =====
def page_Mon_Espace_Administration(request):
    # Accès (Admin ou Modérateur)
    user_data = request.session.get("user")
    if not user_data or (user_data.get("role") not in ["Administrateur", "Moderateur"]):
        messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.")
        return redirect("name_acceuil")

    # Données existantes pour formulaires
    photo_profil = Projetphotodeprofil.objects.first()
    a_propos = ProjetAproposDeMoi.objects.first()
    specialite = Projetsfirstspeciality.objects.first()

    if request.method == "POST":

        # --- PHOTO DE PROFIL (SWITCH AUTO) ---
        if "btn_visibilite_photo_auto" in request.POST:
            vis = "est_visible" in request.POST
            if photo_profil:
                photo_profil.est_visible = vis
                photo_profil.save()
            messages.info(request, "Visibilité de la photo mise à jour.")
            return redirect("dashboard_admin")

        # --- PHOTO DE PROFIL (UPLOAD/UPDATE) ---
        if "btn_photo" in request.POST:
            img = request.FILES.get("images")
            if photo_profil:
                if img:
                    photo_profil.images = img
                    photo_profil.save()
            else:
                Projetphotodeprofil.objects.create(images=img, est_visible=True)
            messages.success(request, "Photo enregistrée.")
            return redirect("dashboard_admin")

```

```
if "btn_supprimer_photo" in request.POST:
    if photo_profil:
        photo_profil.delete()
    messages.warning(request, "Photo supprimée.")
    return redirect("dashboard_admin")

# --- À PROPOS (BIO - SWITCH AUTO) ---
if "btn_visibilite_bio_auto" in request.POST:
    vis = "est_visible_bio" in request.POST
    if a_propos:
        a_propos.est_visible = vis
        a_propos.save()
    messages.info(request, "Visibilité de la bio mise à jour.")
    return redirect("dashboard_admin")

# --- À PROPOS (BIO - TEXTE) ---
if "btn_apropos" in request.POST:
    description = request.POST.get("description")
    if a_propos:
        a_propos.description = description
        a_propos.save()
    else:
        ProjetAproposDeMoi.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")
    return redirect("dashboard_admin")

if "btn_supprimer_apropos" in request.POST:
    if a_propos:
        a_propos.delete()
    messages.error(request, "Texte biographique supprimé.")
    return redirect("dashboard_admin")

# --- SPÉCIALITÉ ---
if "btn_visibilite_section_spec" in request.POST:
    vis = "est_visible" in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")
    return redirect("dashboard_admin")

if "btn_specialite" in request.POST:
    img = request.FILES.get("images")
    desc = request.POST.get("description_speciality")
    if specialite:
        specialite.description_speciality = desc
        if img:
```

```
        specialite.images = img
        specialite.save()
    else:
        Projetsfirstspeciality.objects.create(
            description_speciality=desc,
            images=img,
            est_visible=True
        )
    messages.success(request, "Détails de la spécialité mis à jour.")
    return redirect("dashboard_admin")

# --- RÉALISATIONS (PROJETS) ---
if "btn_visibilite_section_real" in request.POST:
    vis = "est_visible" in request.POST
    Projetmesrealisations.objects.all().update(est_visible=vis)
    messages.info(request, "Section Projets mise à jour.")
    return redirect("dashboard_admin")

if "btn_bascule_vis_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_real" in request.POST:
    Projetmesrealisations.objects.create(
        nom=request.POST.get("nom"),
        description=request.POST.get("description"),
        images=request.FILES.get("images"),
        est_visible=True
    )
    messages.success(request, "Projet ajouté.")
    return redirect("dashboard_admin")

if "btn_modifier_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.nom = request.POST.get("nom")
    obj.description = request.POST.get("description")
    if request.FILES.get("images"):
        obj.images = request.FILES.get("images")
    obj.save()
    messages.info(request, "Projet modifié.")
    return redirect("dashboard_admin")

if "btn_supprimer_real" in request.POST:
    get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation")).delete()
```

```
messages.warning(request, "Projet supprimé.")
return redirect("dashboard_admin")

# --- TÉMOIGNAGES (ANTI-DOUBLON DUR via fingerprint) ---
if "btn_visibilite_section_tem" in request.POST:
    vis = "est_visible" in request.POST
    Projetscards.objects.all().update(est_visible=vis)
    return redirect("dashboard_admin")

if "btn_bascule_vis_tem" in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get("id_temoignage"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_temoignage" in request.POST:
    nom = (request.POST.get("nom") or "").strip()
    description = (request.POST.get("description") or "").strip()
    image = request.FILES.get("images")

    fp = Projetscards.make_fingerprint(nom, description)

    # IMPORTANT : update_or_create => pas de ValidationError si fingerprint existe déjà
    obj, created = Projetscards.objects.update_or_create(
        fingerprint=fp,
        defaults={
            "nom": nom,
            "description": description,
            "est_visible": True,
        }
    )

    # Image: si fournie, on remplace
    if image:
        obj.images = image
        obj.save()

    if created:
        messages.success(request, "Témoignage ajouté.")
    else:
        messages.info(request, "Témoignage déjà existant : mise à jour effectuée.")

return redirect("dashboard_admin")

if "btn_supprimer_temoignage" in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get("id_temoignage")).delete()
    messages.warning(request, "Témoignage supprimé.")
```

```
    return redirect("dashboard_admin")

# --- COMPÉTENCES ---
if "btn_visibilite_section_comp" in request.POST:
    vis = "est_visible" in request.POST
    MesCompetencesCles.objects.all().update(est_visible=vis)
    return redirect("dashboard_admin")

if "btn_bascule_vis_comp" in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_comp" in request.POST:
    MesCompetencesCles.objects.create(
        nom=request.POST.get("nom_competence"),
        est_visible=True
    )
    messages.success(request, "Compétence ajoutée.")
    return redirect("dashboard_admin")

if "btn_supprimer_comp" in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence")).delete()
    messages.warning(request, "Compétence supprimée.")
    return redirect("dashboard_admin")

# --- RÉSEAUX SOCIAUX ---
if "btn_visibilite_section_res" in request.POST:
    vis = "est_visible" in request.POST
    ReseauSocial.objects.all().update(est_visible=vis)
    return redirect("dashboard_admin")

if "btn_reseau" in request.POST:
    nom = (request.POST.get("nom_reseau") or "").strip()
    url = (request.POST.get("url_reseau") or "").strip()

    # normalisations simples
    if nom.lower() == "email" and url.lower().startswith("mailto:"):
        url = url.split(":", 1)[1].strip()
    if nom.lower() == "whatsapp" and url and url.startswith("wa.me/"):
        url = "https://" + url

    ReseauSocial.objects.update_or_create(
        nom=nom,
        defaults={"url": url, "est_visible": True}
```

```

        )
messages.success(request, "Réseau mis à jour.")
return redirect("dashboard_admin")

if "btn_supprimer_reseau" in request.POST:
    get_object_or_404(ReseauSocial, id=request.POST.get("id_reseau")).delete()
    messages.warning(request, "Réseau supprimé.")
    return redirect("dashboard_admin")

# fallback
return redirect("dashboard_admin")

# Contexte GET
context = {
    "AFFICHER_la_PHOTO_DE_PROFIL": photo_profil,
    "A_PROPOS_DE_MOI": a_propos,
    "SPECIALITE": specialite,
    "AFFICHER_les_REALISATIONS_PERSONNELLES": Projetmesrealisations.objects.all().order_by("-id"),
    "REALISATIONS_STATS": Projetmesrealisations.objects.all().order_by("-compteur_demo_live", "-id"),
    "TOTAL_CLICS_DEMO_LIVE": (Projetmesrealisations.objects.aggregate(total=Sum("compteur_demo_live"))["total"] or 0),
    "TEMOIGNAGES_des_PERSONNES_cards": Projetscards.objects.all().order_by("-id"),
    "Mes_Competences_Cles": MesCompetencesCles.objects.all().order_by("nom"),
    "Mes_Reseaux": ReseauSocial.objects.all(),
}
return render(request, "page_administration.html", context)

# =====
# 2bis) COMPTEUR DE CLICS "DÉMO LIVE" PAR PROJET
# =====
def incrementer_compteur_demo_live(request, realisation_id):
    """Incrémente le compteur quand un visiteur clique sur 'Démo Live' puis redirige vers la démo."""
    Projetmesrealisations.objects.filter(id=realisation_id).update(
        compteur_demo_live=F("compteur_demo_live") + 1
    )
    return redirect("name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES")

# =====
# 3) REDIRECTIONS
# =====
def Fonction_Page_accueil_section_Accueil(request): return redirect("/#accueil")
def Fonction_Page_accueil_section_projets(request): return redirect("/#projets")
def Fonction_Page_accueil_section_temoignages(request): return redirect("/#temoignages")
def Fonction_Page_accueil_section_a_propos(request): return redirect("/#a-propos")
def Fonction_Page_accueil_section_contact(request): return redirect("/#contact")

```

```
def Fonction_Page_email_Javascript(request): return render(request, "email_Javascript.htm")
def Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request): return render(request, "TransportRoutier_Main.html")
def Fonction_Page_diapo_Javascript_Page1(request): return render(request, "diapo_Javascript_Page1.html")

# =====
# RÉINITIALISATION DU COMPTEUR POUR UN PROJET SPÉCIFIQUE
# =====
def reinitialiser_compteur_projet(request, realisation_id):
    """Remet à zéro le compteur de clics pour un projet donné."""
    projet = get_object_or_404(Projetmesrealisations, id=realisation_id)
    projet.compteur_demo_live = 0
    projet.save()

    messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.")
    return redirect("dashboard_admin")

# =====
# EXPORT / IMPORT JSON (Admin uniquement)
# =====
# Tu peux soit copier-coller ton fichier json_transfer.py ici,
# soit l'importer. Comme tu demandes "views.py complet", je l'inclus ici.

import base64
import json
import mimetypes
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden

def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")
```

```
def _json_compatible_value(value):
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projects_cards(nom: str, description: str) -> str:
    base = f"{{_norm_text(nom)}}||{{_norm_text(description)}}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def _serialize_instance(obj):
    data = {
        "__model__": f"{{obj._meta.app_label}}.{obj.__class__.__name__}",
        "id": obj.pk,
    }
    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue
        value = getattr(obj, name)
```

```
if isinstance(field, (FileField, ImageField)):
    if value and getattr(value, "name", None):
        file_path = value.name
        b64 = None
        try:
            with default_storage.open(file_path, "rb") as f:
                raw = f.read()
            b64 = base64.b64encode(raw).decode("utf-8")
        except Exception:
            b64 = None

    mime, _ = mimetypes.guess_type(file_path)
    data[name] = {
        "storage_path": file_path,
        "filename": file_path.split("/")[-1],
        "mimetype": mime or "application/octet-stream",
        "base64": b64,
    }
else:
    data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetsmesrealisations,
    Projetscards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
```

```

"export_type": "portfolio",
"exported_at": datetime.now().isoformat(),
"version": 1,
"data": {
    "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
    "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
    "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
    "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
    "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
    "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
    "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
},
}

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_contact.models import User, Role, Contact
    from app_user.models import Menu, RoleMenuAcces

    payload = {
        "export_type": "users",
        "exported_at": datetime.now().isoformat(),
        "version": 1,
        "notes": ["User.password est exporté tel quel (hash)."],
        "data": {
            "app_contact.Role": _serialize_queryset(Role.objects.all()),
            "app_contact.User": _serialize_queryset(User.objects.all()),
            "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
            "app_user.Menu": _serialize_queryset(Menu.objects.all()),
            "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    return _safe_json_response(payload, f"{date_prefix}_export_users.json")

def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:

```

```

text = raw.decode("utf-8")
except Exception:
    text = raw.decode("latin-1")
return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)

SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"], # si ton modèle a nom_projet ; sinon retire

    # Témoignages
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"],
}

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

```

```

lookup = {}
for k in keys:
    if k not in fields_data:
        return None
    v = fields_data[k]
    if v is None or (isinstance(v, str) and v.strip() == ""):
        return None
    lookup[k] = v

try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # Projetscards : calc fingerprint si absent
        if model_label == "app_acceuil.Projetscards":
            if not fields_data.get("fingerprint"):
                fields_data["fingerprint"] = _fingerprint_projetscards(
                    fields_data.get("nom", ""),
                    fields_data.get("description", ""),
                )
            # normalisation fingerprint
            if isinstance(fields_data.get("fingerprint"), str):
                fields_data["fingerprint"] = fields_data["fingerprint"].strip().lower()

        instance = None

```

```

# 1) par ID
if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()

# 2) par clés métier
if instance is None:
    instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

# 3) create
if instance is None:
    instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs + normalisations sûres
for k, v in fields_data.items():
    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)
    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    # texte visible Projetscards: strip seulement
    if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
        v = v.strip()

    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

```

```
json_portfolio = request.FILES.get("json_portfolio")
json_users = request.FILES.get("json_users")
replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

# ----- Portfolio import -----
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_portfolio:
            from app_acceuil.models import (
                Projetphotodeprofil,
                ProjetAproposDeMoi,
                Projetsfirstspeciality,
                Projetmesrealisations,
                Projetcards,
                MesCompetencesCles,
                ReseauSocial,
            )
            ReseauSocial.objects.all().delete()
            MesCompetencesCles.objects.all().delete()
            Projetcards.objects.all().delete()
            Projetmesrealisations.objects.all().delete()
            Projetsfirstspeciality.objects.all().delete()
            ProjetAproposDeMoi.objects.all().delete()
            Projetphotodeprofil.objects.all().delete()

            for model_label, objects_list in model_map.items():
                _upsert_objects(model_label, objects_list, keep_ids=True)

            messages.success(request, "Import Portfolio terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
```

```

model_map = data.get("data", {})
if not isinstance(model_map, dict):
    raise ValueError("Format JSON invalide: data doit être un objet (dict).")

if replace_users:
    from app_user.models import RoleMenuAcces, Menu
    from app_contact.models import User, Role, Contact

    RoleMenuAcces.objects.all().delete()
    User.objects.all().delete()
    Contact.objects.all().delete()
    Menu.objects.all().delete()
    Role.objects.all().delete()

ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

 Ce `views.py` corrige ton bug **fingerprint** (plus de `ValidationError`) et inclut **Export / Import JSON** dans le même fichier.

Si tu veux, je te donne aussi le `app_acceuil/urls.py` complet pour ajouter proprement :

- `/export/portfolio`
- `/export/users`
- `/import/json-bundle (POST)`

oui

```
# app_acceuil/urls.py

from django.urls import path
```

```

from . import views

urlpatterns = [
    # =====
    # Pages principales
    # =====
    path("", views.aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="name_acceuil"),
    path("dashboard/", views.page_Mon_Espace_Administration, name="dashboard_admin"),

    # =====
    # Redirections sections (ancres)
    # =====
    path("accueil/", views.Fonction_Page_accueil_section_Accueil, name="name_accueil_section_accueil"),
    path("projets/", views.Fonction_Page_accueil_section_projets, name="name_accueil_section_projets"),
    path("temoignages/", views.Fonction_Page_accueil_section_temoignages, name="name_accueil_section_temoignages"),
    path("apropos/", views.Fonction_Page_accueil_section_a_propos, name="name_accueil_section_a_propos"),
    path("contact/", views.Fonction_Page_accueil_section_contact, name="name_accueil_section_contact"),

    # =====
    # Pages "templates" annexes
    # =====
    path("email-javascript/", views.Fonction_Page_email_Javascript, name="name_Page_email_Javascript"),
    path("camions/", views.Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES,
         name="name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES"),
    path("diapo-js-1/", views.Fonction_Page_diapo_Javascript_Page1, name="name_Page_diapo_Javascript_Page1"),

    # =====
    # Compteurs
    # =====
    path("projet/<int:realisation_id>/click-demo-live/", views.incremente_compteur_demo_live,
         name="incremente_compteur_demo_live"),
    path("projet/<int:realisation_id>/reset-compteur/", views.reinitialiser_compteur_projet,
         name="reinitialiser_compteur_projet"),

    # =====
    # EXPORTS JSON (Admin)
    # =====
    path("export/portfolio/", views.export_portfolio_json, name="export_portfolio_json"),
    path("export/users/", views.export_users_json, name="export_users_json"),

    # =====
    # IMPORT JSON (Admin) - POST
    # =====
    path("import/json-bundle/", views.import_json_bundle, name="import_json_bundle"),
]

```

Avec ça, tes URLs deviennent :

- **Export Portfolio** : `http://127.0.0.1:8000/export/portfolio/`
- **Export Users** : `http://127.0.0.1:8000/export/users/`
- **Import Bundle (POST)** : `http://127.0.0.1:8000/import/json-bundle/`

Si ton projet a un `portfolio/urls.py` (root urls), assure-toi aussi que `app_acceuil.urls` est bien inclus dedans (je peux te donner le fichier root complet si tu veux).

```
from django.urls import path from .views import * from app_acceuil.json_transfer import ( export_portfolio_json, export_users_json, import_json_bundle, )
urlpatterns = [ path("export/portfolio", export_portfolio_json, name="export_portfolio_json"), path("export/users", export_users_json, name="export_users_json"), path("import/json", import_json_bundle, name="import_json_bundle"), # Votre page publique (Vue par les visiteurs) # Assurez-vous d'avoir créé la fonction 'page_publique' dans views.py path("", aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="name_acceuil"), path('index.html', aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="name_acceuil"), # Mon Espace Administration # Mon Espace Administration # Mon Espace Administration # Mon Espace Administration # Votre Dashboard personnalisé (Sécurisé dans views.py) path('dashboard', page_Mon_Espace_Administration, name='dashboard_admin'), # Compteur de clics sur le bouton 'Démo Live' (Mes Réalisations)
path('realisations/<int:realisation_id>/demo-live', incrementer_compteur_demo_live, name='realisation_demo_live'), # PAGE de CAMIONS DE TRANSPORT DE MARCHANDISES path('Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES',
Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES, name='name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES'),
path('Page_diapo_Javascript_Page1', Fonction_Page_diapo_Javascript_Page1, name='name_Page_diapo_Javascript_Page1'), # FORMULAIRE d'un ENVOI d'un EMAIL en javascript # https://gemini.google.com/app/50f984f5e7992921?hl=fr # https://gemini.google.com/app/50f984f5e7992921?hl=fr #
https://gemini.google.com/app/2b0ac21fa009d2d9?hl=fr # https://gemini.google.com/app/50f984f5e7992921?hl=fr #
https://gemini.google.com/app/50f984f5e7992921?hl=fr # Pourquoi séparer l'ancre du nom du fichier ? # Le schéma suivant montre comment Django traite votre demande : # Le Navigateur demande une URL. # Django exécute la fonction dans views.py. # Django cherche le fichier template (sans l'ancre car le système de fichier ne la reconnaît pas). # Django renvoie le HTML au navigateur. # Le Navigateur reçoit la page et, voyant #email_JAVASCRIPT dans son URL, fait défiler la page automatiquement. # Souhaitez-vous que je vous aide à écrire le script JavaScript pour récupérer les données du formulaire et envoyer l'email ? # Comment l'utilisateur arrive sur l'ancre ? Dans vos autres pages HTML (par exemple votre menu), votre lien doit ressembler à ceci :
path('Page_email_Javascript', Fonction_Page_email_Javascript, name='name_Page_email_Javascript'), # href="index.html#accueil"
path('Page_accueil_section_Accueil', Fonction_Page_accueil_section_Accueil, name='name_Page_accueil_section_Accueil'), # href="index.html#projets"
path('Page_accueil_section_projets', Fonction_Page_accueil_section_projets, name='name_Page_accueil_section_projets'), # href="index.html#temoignages"
path('Page_accueil_section_temoignages', Fonction_Page_accueil_section_temoignages, name='name_Page_accueil_section_temoignages'), #
href="index.html#a-propos" path('Page_accueil_section_a_propos', Fonction_Page_accueil_section_a_propos, name='name_Page_accueil_section_a_propos'),
# href="index.html#contact" path('Page_accueil_section_contact', Fonction_Page_accueil_section_contact, name='name_Page_accueil_section_contact'), # ...
tes autres routes ... # ===== # RÉINITIALISATION DU COMPTEUR POUR UN PROJET SPÉCIFIQUE #
path('reinitialiser-compteur/<int:realisation_id>', reinitialiser_compteur_projet, name='reinitialiser_compteur_unique'), ]
```

Ton urls.py est **ok**, mais il y a 3 points importants pour que ce soit **propre + fiable** (et éviter les surprises) :

## 1) Mets des slashes finaux partout (recommandé)

Avec APPEND\_SLASH=True (c'est ton cas), Django peut rediriger automatiquement, mais **sur certaines routes POST** ça peut poser problème si tu appelles sans /.

 Donc fais comme ça :

```
# app_acceuil/urls.py
from django.urls import path
from .views import *

from app_acceuil.json_transfer import (
    export_portfolio_json,
    export_users_json,
    import_json_bundle,
)

urlpatterns = [
    # JSON transfer (ADMIN)
    path("export/portfolio/", export_portfolio_json, name="export_portfolio_json"),
    path("export/users/", export_users_json, name="export_users_json"),
    path("import/json/", import_json_bundle, name="import_json_bundle"),

    # Public
    path("", aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="name_acceuil"),
    path("index.html", aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS, name="name_acceuil"),

    # Dashboard
    path("dashboard/", page_Mon_Espace_Administration, name="dashboard_admin"),

    # Démo live
    path("realisations/<int:realisation_id>/demo-live/", incrementer_compteur_demo_live, name="realisation_demo_live"),

    # Pages annexes
    path("Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES/", Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES,
         name="name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES"),
    path("Page_diapo_Javascript_Page1/", Fonction_Page_diapo_Javascript_Page1, name="name_Page_diapo_Javascript_Page1"),
    path("Page_email_Javascript/", Fonction_Page_email_Javascript, name="name_Page_email_Javascript"),

    # Ancres
    path("Page_accueil_section_Accueil/", Fonction_Page_accueil_section_Accueil, name="name_Page_accueil_section_Accueil"),
```

```

path("Page_accueil_section_projets/", Fonction_Page_accueil_section_projets, name="name_Page_accueil_section_projets"),
path("Page_accueil_section_temoignages/", Fonction_Page_accueil_section_temoignages,
name="name_Page_accueil_section_temoignages"),
path("Page_accueil_section_a_propos/", Fonction_Page_accueil_section_a_propos, name="name_Page_accueil_section_a_propos"),
path("Page_accueil_section_contact/", Fonction_Page_accueil_section_contact, name="name_Page_accueil_section_contact"),
]

# Reset compteur
path("reinitialiser-compteur/<int:realisation_id>", reinitialiser_compteur_projet, name="reinitialiser_compteur_unique"),
]

```

---

## 2) Ton erreur `ValidationError` `fingerprint already exists` vient de `views.py` (dashboard)

Tu as rendu `fingerprint unique=True`, et dans ton `models.py` tu génères le fingerprint au `save()`.

→ Donc si tu ajoutes un témoignage avec (nom + description) identiques, Django refusera (normal).

 Pour éviter de “planter” la page admin, au lieu de faire `Projetscards.objects.create(...)`, il faut faire un `create-or-ignore` basé sur le fingerprint, puis afficher un message.

👉 Dans `app_acceuil/views.py`, remplace ton bloc :

```

elif 'btn_ajouter_temoignage' in request.POST:
    Projetscards.objects.create(
        nom=request.POST.get('nom'),
        description=request.POST.get('description'),
        images=request.FILES.get('images'),
        est_visible=True
    )
    messages.success(request, "Témoignage ajouté.")

```

par ceci  :

```

elif 'btn_ajouter_temoignage' in request.POST:
    nom = (request.POST.get('nom') or '').strip()
    description = (request.POST.get('description') or '').strip()
    img = request.FILES.get('images')

    # fingerprint calculé comme dans le model
    fp = Projetscards.make_fingerprint(nom, description)

    existing = Projetscards.objects.filter(fingerprint=fp).first()

```

```
if existing:
    messages.warning(request, "Ce témoignage existe déjà (doublon détecté).")
else:
    Projetscards.objects.create(
        nom=nom,
        description=description,
        images=img,
        est_visible=True,
        fingerprint=fp, # optionnel car save() le calcule si vide, mais là c'est explicite
    )
messages.success(request, "Témoignage ajouté.")
```

---

### 3) Tes URLs export/import actuelles sans slash vont marcher, mais je te conseille :

- Export : GET /export/portfolio/ et GET /export/users/
- Import : POST /import/json/

Comme ça ton front dans page\_administration.html est stable.

---

### Résumé

#### ✓ Oui ton urls.py est correct, mais je te recommande fortement :

- mettre les / finaux
- corriger l'ajout de témoignage dans views.py pour gérer le doublon fingerprint proprement

Si tu veux, colle-moi ton app\_acceuil/json\_transfer.py actuel (ou dis-moi si c'est exactement celui que tu as envoyé plus haut) et je te renvoie les 3 fichiers complets cohérents :

- app\_acceuil/models.py (déjà ok)
- app\_acceuil/views.py (corrige + import/export intégrés si tu veux)
- app\_acceuil/urls.py (corrige avec slashes)

ValidationError at /dashboard/ {'images': ['This field cannot be blank.']} Request Method: POST Request URL: http://127.0.0.1:8000/dashboard/ Django Version: 5.2.8 Exception Type: ValidationError Exception Value: {'images': ['This field cannot be blank.']} Exception Location: D:\PYTHON DJANGO

HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\db\models\base.py, line 1679, in full\_clean Raised during:  
app\_acceuil.views.page\_Mon\_Espace\_Administration Python Executable: D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Scripts\python.exe  
Python Version: 3.11.9 Python Path: ['D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO', 'C:\\Program Files\\Python311\\python311.zip', 'C:\\Program Files\\Python311\\DLLs', 'C:\\Program Files\\Python311\\Lib', 'C:\\Program Files\\Python311', 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO  
DJANGO\\.venv', 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv\\Lib\\site-packages'] Server time: Wed, 04 Feb 2026 20:01:53 +0000  
Traceback Switch to copy-and-paste view D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\db\models\query.py, line  
948, in get\_or\_create return self.get(\*\*kwargs), False ~~~~~ ... Local vars D:\PYTHON DJANGO HOPES\PORTFOLIO  
DJANGO\.venv\Lib\site-packages\django\db\models\query.py, line 635, in get raise self.model.DoesNotExist( ^ ... Local vars During handling of the above  
exception (Projetscards matching query does not exist.), another exception occurred: D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-  
packages\django\core\handlers\exception.py, line 55, in inner response = get\_response(request) ~~~~~ ... Local vars D:\PYTHON DJANGO  
HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\core\handlers\base.py, line 197, in \_get\_response response = wrapped\_callback(request,  
\*callback\_args, \*\*callback\_kwargs) ~~~~~ ... Local vars D:\PYTHON DJANGO  
HOPES\PORTFOLIO DJANGO\app\_acceuil\views.py, line 237, in page\_Mon\_Espace\_Administration obj, created = Projetscards.objects.update\_or\_create(  
... Local vars D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\db\models\manager.py, line 87, in manager\_method  
return getattr(self.get\_queryset(), name)(\*args, \*\*kwargs) ~~~~~ ... Local vars D:\PYTHON DJANGO  
HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\db\models\query.py, line 990, in update\_or\_create obj, created =  
self.select\_for\_update().get\_or\_create( ... Local vars D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-  
packages\django\db\models\query.py, line 955, in get\_or\_create return self.create(\*\*params), True ~~~~~ ... Local vars D:\PYTHON  
DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-packages\django\db\models\query.py, line 665, in create obj.save(force\_insert=True, using=self.db)  
~~~~~ ... Local vars D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\app\_acceuil\models.py, line 60, in  
save self.full\_clean() ~~~~~ ... Local vars D:\PYTHON DJANGO HOPES\PORTFOLIO DJANGO\.venv\Lib\site-  
packages\django\db\models\base.py, line 1679, in full\_clean raise ValidationError(errors) ~~~~~ ... Local vars Request information  
USER AnonymousUser GET No GET data POST Variable Value csrfmiddlewaretoken  
'A1gC1HIMYmRw4ZiBPII7Od8w37EpbI2Os7MNsKdvRhiwZlaPEKJBR22S2570ke7N' nom 'vnvbvn' description 'kjhkj' btn\_ajouter\_temoignage " FILES  
Variable Value images <InMemoryUploadedFile: arnold\_schwarzenegger\_vieux2.JPG (image/jpeg)> COOKIES Variable Value csrf token  
\*\*\*\*\* sessionid \*\*\*\*\* META Variable Value ALLUSERSPROFILE 'C:\\\\ProgramData' APPDATA  
'C:\\\\Users\\\\SPSSE02MEN\\\\AppData\\\\Roaming' BUNDLED\_DEBUGPY\_PATH 'c:\\\\Users\\\\SPSSE02MEN\\\\vscode\\\\extensions\\\\ms-python.debugpy-  
2025.18.0-win32-x64\\\\bundled\\\\libs\\\\debugpy' CHROME\_CRASHPAD\_PIPE\_NAME '\\\\\\pipe\\\\crashpad\_10848\_KQVIBTMZBXUEGF' COLORTERM  
'truecolor' COMMONPROGRAMFILES 'C:\\\\Program Files\\\\Common Files' COMMONPROGRAMFILES(X86) 'C:\\\\Program Files (x86)\\\\Common Files'  
COMMONPROGRAMW6432 'C:\\\\Program Files\\\\Common Files' COMPUTERNAME 'DESKTOP-S95K8U0' COMSPEC 'C:\\\\Windows\\\\system32\\\\cmd.exe'  
CONTENT\_LENGTH '21198' CONTENT\_TYPE ('multipart/form-data; 'boundary=----geckoformboundary517760aa2e3c5b15640331990719d3cf)  
CSRF\_COOKIE '2gGlBdFT35Ba5T2oZcyEdZ4w98DLjGf9' DJANGO\_SETTINGS\_MODULE 'portfolio.settings' DRIVERDATA  
'C:\\\\Windows\\\\System32\\\\Drivers\\\\DriverData' GATEWAY\_INTERFACE 'CGI/1.1' GIT\_ASKPASS \*\*\*\*\* GIT\_LFS\_PATH 'C:\\\\Program  
Files\\\\Git LFS' HOMEDRIVE 'C:' HOMEPATH '\\\\Users\\\\SPSSE02MEN' HTTP\_ACCEPT 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8'  
HTTP\_ACCEPT\_ENCODING 'gzip, deflate, br, zstd' HTTP\_ACCEPT\_LANGUAGE 'fr,fr-FR;q=0.9,en-US;q=0.8,en;q=0.7' HTTP\_CONNECTION 'keep-  
alive' HTTP\_COOKIE \*\*\*\*\* HTTP\_HOST '127.0.0.1:8000' HTTP\_ORIGIN 'http://127.0.0.1:8000' HTTP\_PRIORITY 'u=0,i'  
HTTP\_REFERER 'http://127.0.0.1:8000/dashboard/' HTTP\_SEC\_FETCH\_DEST 'document' HTTP\_SEC\_FETCH\_MODE 'navigate'

HTTP\_SEC\_FETCH\_SITE 'same-origin' HTTP\_SEC\_FETCH\_USER '?1' HTTP\_UPGRADE\_INSECURE\_REQUESTS '1' HTTP\_USER\_AGENT ('Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:147.0) Gecko/20100101 Firefox/147.0') LANG 'en\_US.UTF-8' LOCALAPPDATA 'C:\\Users\\SPSSE02MEN\\AppData\\Local' LOGONSERVER '\\\\DESKTOP-S95K8U0' NUMBER\_OF\_PROCESSORS '4' ONEDRIVE 'C:\\Users\\SPSSE02MEN\\OneDrive' OS 'Windows\_NT' PATH ('D:\\PYTHON DJANGO HOPES\\PORTFOLIO' 'DJANGO\\.venv\\Scripts;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\debugCommand;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\copilotCli;C:\\Program' 'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition 2024.1.1\\bin;C:\\Program Files\\Common ' 'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Windows\\System32\\OpenSSH\\;C:\\Program ' 'Files (x86)\\NVIDIA ' 'Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' 'Menu\\Programs\\Python;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' 'Menu\\Programs;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Python\\Python311\\Scripts;C:\\Program ' 'Files\\Git\\cmd;C:\\Program ' 'Files\\nodejs\\;C:\\src\\flutter\\bin;C:\\dart-sdk\\bin;;C:\\Program ' 'Files\\Git LFS;C:\\Program Files\\Python3\_12\_4\\Scripts\\;C:\\Program ' 'Files\\Python3\_12\_4\\;%IntelliJ IDEA%;%IntelliJ IDEA Community ' 'Edition%;%PyCharm Community ' 'Edition%;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Microsoft VS ' 'Code\\bin;C:\\Program Files '(x86)\\Nmap;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\npm;c:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\bundled\\scripts\\noConfigScripts') PATHEXT '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW;.CPL' PATH\_INFO '/dashboard/' PROCESSOR\_ARCHITECTURE 'AMD64' PROCESSOR\_IDENTIFIER 'Intel64 Family 6 Model 69 Stepping 1, GenuineIntel' PROCESSOR\_LEVEL '6' PROCESSOR\_REVISION '4501' PROGRAMDATA 'C:\\ProgramData' PROGRAMFILES 'C:\\Program Files' PROGRAMFILES(X86) 'C:\\Program Files (x86)' PROGRAMW6432 'C:\\Program Files' PROMPT '\$P\$G' PSMODULEPATH ('C:\\Users\\SPSSE02MEN\\Documents\\WindowsPowerShell\\Modules;C:\\Program ' 'Files\\WindowsPowerShell\\Modules;C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\Modules') PUBLIC 'C:\\Users\\Public' PYDEV\_DISABLE\_FILE\_VALIDATION '1' QUERY\_STRING " REMOTE\_ADDR '127.0.0.1' REMOTE\_HOST " REQUEST\_METHOD 'POST' RUN\_MAIN 'true' SCRIPT\_NAME " SERVER\_NAME 'DESKTOP-S95K8U0' SERVER\_PORT '8000' SERVER\_PROTOCOL 'HTTP/1.1' SERVER\_SOFTWARE 'WSGIServer/0.2' SYSTEMDRIVE 'C:' SYSTEMROOT 'C:\\Windows' TEMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp' TERM\_PROGRAM 'vscode' TERM\_PROGRAM\_VERSION '1.108.2' TMP 'C:\\Users\\SPSSE0~1\\AppData\\Local\\Temp' USERDOMAIN 'AzureAD' USERDOMAIN\_ROAMINGPROFILE 'AzureAD' USERNAME 'SPSSE02MEN' USERPROFILE 'C:\\Users\\SPSSE02MEN' VIRTUAL\_ENV 'D:\\PYTHON DJANGO HOPES\\PORTFOLIO DJANGO\\.venv' VIRTUAL\_ENV\_PROMPT '.venv' VS CODE DEBUGPY\_ADAPTER\_ENDPOINTS 'c:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\.noConfigDebugAdapterEndpoints\\endpoint-77f6ce4ded07263f.txt' VS CODE GIT ASKPASS\_EXTRA\_ARGS '\*\*\*\*\*' VS CODE GIT ASKPASS\_MAIN '\*\*\*\*\*' VS CODE GIT ASKPASS\_NODE '\*\*\*\*\*' VS CODE GIT IPC HANDLE '\\\\\\.\\pipe\\vscode-git-c146c63ca8-sock' VS CODE INJECTION '1' VS CODE PYTHON\_AUTOACTIVATE\_GUARD '1' WINDIR 'C:\\Windows' \_OLD\_VIRTUAL\_PATH ('c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\debugCommand;c:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Code\\User\\globalStorage\\github.copilot-chat\\copilotCli;C:\\Program' 'Files\\Python311\\Scripts\\;C:\\Program Files\\Python311\\;C:\\Program ' 'Files\\Java\\jdk-22\\bin;C:\\Program Files\\JetBrains\\PyCharm Community ' 'Edition 2024.1.1\\bin;C:\\Program Files\\Common '

'Files\\Oracle\\Java\\javapath;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Windows\\System32\\OpenSSH\\;C:\\Program 'Files (x86)\\NVIDIA ' Corporation\\PhysX\\Common;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' Menu\\Programs\\Python;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Python\\Launcher\\;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Microsoft\\Windows\\Start ' Menu\\Programs;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\Python\\Python311\\Scripts;C:\\Program 'Files\\Git\\cmd;C:\\Program 'Files\\nodejs\\;C:\\src\\flutter\\bin;C:\\dart-sdk\\bin;;C:\\Program 'Files\\Git LFS;C:\\Program Files\\Python3\_12\_4\\Scripts\\;C:\\Program 'Files\\Python3\_12\_4\\;%IntelliJ IDEA%;%IntelliJ IDEA Community 'Edition%;%PyCharm Community ' Edition%;C:\\Users\\SPSSE02MEN\\AppData\\Local\\Programs\\Microsoft VS 'Code\\bin;C:\\Program Files '(x86)\\Nmap;C:\\Users\\SPSSE02MEN\\AppData\\Roaming\\npm;C:\\Users\\SPSSE02MEN\\.vscode\\extensions\\ms-python.debugpy-2025.18.0-win32-x64\\bundled\\scripts\\noConfigScripts) wsgi.errors <\_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> wsgi.file\_wrapper <class 'wsgiref.util.FileWrapper'> wsgi.input <django.core.handlers.wsgi.LimitedStream object at 0x0000029A765292D0> wsgi.multiprocess False wsgi.multithread True wsgi.run\_once False wsgi.url\_scheme 'http' wsgi.version (1, 0) Settings Using settings module portfolio.settings Setting Value ABSOLUTE\_URL\_OVERRIDES {} ADMINS [] ALLOWED\_HOSTS [] APPEND\_SLASH True AUTHENTICATION\_BACKENDS \*\*\*\*\* AUTH\_PASSWORD\_VALIDATORS \*\*\*\*\* AUTH\_USER\_MODEL \*\*\*\*\* BASE\_DIR WindowsPath('D:/PYTHON DJANGO HOPES/PORTFOLIO DJANGO') CACHES {'default': {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}} CACHE\_MIDDLEWARE\_ALIAS 'default' CACHE\_MIDDLEWARE\_KEY\_PREFIX \*\*\*\*\* CACHE\_MIDDLEWARE\_SECONDS 600 CSRF\_COOKIE\_AGE 31449600 CSRF\_COOKIE\_DOMAIN None CSRF\_COOKIE\_HTTPONLY False CSRF\_COOKIE\_NAME 'csrftoken' CSRF\_COOKIE\_PATH '/' CSRF\_COOKIE\_SAMESITE 'Lax' CSRF\_COOKIE\_SECURE False CSRF\_FAILURE\_VIEW 'django.views.csrf.csrf\_failure' CSRF\_HEADER\_NAME 'HTTP\_X\_CSRFTOKEN' CSRF\_TRUSTED\_ORIGINS [] CSRF\_USE\_SESSIONS False DATABASES {'default': {'ATOMIC\_REQUESTS': False, 'AUTOCOMMIT': True, 'CONN\_HEALTH\_CHECKS': False, 'CONN\_MAX\_AGE': 0, 'ENGINE': 'django.db.backends.sqlite3', 'HOST': '', 'NAME': WindowsPath('D:/PYTHON DJANGO HOPES/PORTFOLIO DJANGO/db.sqlite3'), 'OPTIONS': {}, 'PASSWORD': '\*\*\*\*\*', 'PORT': '', 'TEST': {'CHARSET': None, 'COLLATION': None, 'MIGRATE': True, 'MIRROR': None, 'NAME': None}, 'TIME\_ZONE': None, 'USER': ''}} DATABASE\_ROUTERS [] DATA\_UPLOAD\_MAX\_MEMORY\_SIZE 2621440 DATA\_UPLOAD\_MAX\_NUMBER\_FIELDS 1000 DATA\_UPLOAD\_MAX\_NUMBER\_FILES 100 DATETIME\_FORMAT 'N j, Y, P' DATETIME\_INPUT\_FORMATS ['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M', '%m/%d/%Y %H:%M:%S', '%m/%d/%Y %H:%M:%S.%f', '%m/%d/%Y %H:%M', '%m/%d/%y %H:%M:%S', '%m/%d/%y %H:%M:%S.%f', '%m/%d/%y %H:%M'] DATE\_FORMAT 'N j, Y' DATE\_INPUT\_FORMATS ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y', '%b %d %Y', '%b %d, %Y', '%d %b %Y', '%d %b %Y', '%B %d %Y', '%B %d, %Y', '%d %B %Y', '%d %B, %Y'] DEBUG True DEBUG\_PROPAGATE\_EXCEPTIONS False DECIMAL\_SEPARATOR '.' DEFAULT\_AUTO\_FIELD 'django.db.models.BigAutoField' DEFAULT\_CHARSET 'utf-8' DEFAULT\_EXCEPTION\_REPORTER 'django.views.debug.ExceptionReporter' DEFAULT\_EXCEPTION\_REPORTER\_FILTER 'django.views.debug.SafeExceptionReporterFilter' DEFAULT\_FROM\_EMAIL 'webmaster@localhost' DEFAULT\_INDEX\_TABLESPACE "DEFAULT\_TABLESPACE" DEFAULT\_TABLESPACE "DISALLOWED\_USER\_AGENTS [] EMAIL\_BACKEND 'django.core.mail.backends.smtp.EmailBackend' EMAIL\_HOST 'localhost' EMAIL\_HOST\_PASSWORD '\*\*\*\*\*' EMAIL\_HOST\_USER '' EMAIL\_PORT 25 EMAIL\_SSL\_CERTFILE None EMAIL\_SSL\_KEYFILE '\*\*\*\*\*' EMAIL SUBJECT\_PREFIX '[Django]' EMAIL\_TIMEOUT None EMAIL\_USE\_LOCALTIME False EMAIL\_USE\_SSL False EMAIL\_USE\_TLS False FILE\_UPLOAD\_DIRECTORY\_PERMISSIONS None FILE\_UPLOAD\_HANDLERS ['django.core.files.uploadhandler.MemoryFileUploadHandler', 'django.core.files.uploadhandler.TemporaryFileUploadHandler'] FILE\_UPLOAD\_MAX\_MEMORY\_SIZE 2621440 FILE\_UPLOAD\_PERMISSIONS 420 FILE\_UPLOAD\_TEMP\_DIR None FIRST\_DAY\_OF\_WEEK 0 FIXTURE\_DIRS [] FORCE\_SCRIPT\_NAME None FORMAT\_MODULE\_PATH None

FORMS\_URLFIELD\_ASSUME\_HTTPS False FORM\_RENDERER 'django.forms.renderers.DjangoTemplates' IGNORABLE\_404\_URLS []  
INSTALLED\_APPS ['django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', 'app\_acceuil', 'app\_contact', 'app\_user'] INTERNAL\_IPS [] LANGUAGES [('af', 'Afrikaans'), ('ar', 'Arabic'), ('ar-dz', 'Algerian Arabic'), ('ast', 'Asturian'), ('az', 'Azerbaijani'), ('bg', 'Bulgarian'), ('be', 'Belarusian'), ('bn', 'Bengali'), ('br', 'Breton'), ('bs', 'Bosnian'), ('ca', 'Catalan'), ('ckb', 'Central Kurdish (Sorani)'), ('cs', 'Czech'), ('cy', 'Welsh'), ('da', 'Danish'), ('de', 'German'), ('dsb', 'Lower Sorbian'), ('el', 'Greek'), ('en', 'English'), ('en-au', 'Australian English'), ('en-gb', 'British English'), ('eo', 'Esperanto'), ('es', 'Spanish'), ('es-ar', 'Argentinian Spanish'), ('es-co', 'Colombian Spanish'), ('es-mx', 'Mexican Spanish'), ('es-ni', 'Nicaraguan Spanish'), ('es-ve', 'Venezuelan Spanish'), ('et', 'Estonian'), ('eu', 'Basque'), ('fa', 'Persian'), ('fi', 'Finnish'), ('fr', 'French'), ('fy', 'Frisian'), ('ga', 'Irish'), ('gd', 'Scottish Gaelic'), ('gl', 'Galician'), ('he', 'Hebrew'), ('hi', 'Hindi'), ('hr', 'Croatian'), ('hsb', 'Upper Sorbian'), ('hu', 'Hungarian'), ('hy', 'Armenian'), ('ia', 'Interlingua'), ('id', 'Indonesian'), ('ig', 'Igbo'), ('io', 'Ido'), ('is', 'Icelandic'), ('it', 'Italian'), ('ja', 'Japanese'), ('ka', 'Georgian'), ('kab', 'Kabyle'), ('kk', 'Kazakh'), ('km', 'Khmer'), ('kn', 'Kannada'), ('ko', 'Korean'), ('ky', 'Kyrgyz'), ('lb', 'Luxembourgish'), ('lt', 'Lithuanian'), ('lv', 'Latvian'), ('mk', 'Macedonian'), ('ml', 'Malayalam'), ('mn', 'Mongolian'), ('mr', 'Marathi'), ('ms', 'Malay'), ('my', 'Burmese'), ('nb', 'Norwegian Bokmål'), ('ne', 'Nepali'), ('nl', 'Dutch'), ('nn', 'Norwegian Nynorsk'), ('os', 'Ossetic'), ('pa', 'Punjabi'), ('pl', 'Polish'), ('pt', 'Portuguese'), ('pt-br', 'Brazilian Portuguese'), ('ro', 'Romanian'), ('ru', 'Russian'), ('sk', 'Slovak'), ('sl', 'Slovenian'), ('sq', 'Albanian'), ('sr', 'Serbian'), ('sr-latn', 'Serbian Latin'), ('sv', 'Swedish'), ('sw', 'Swahili'), ('ta', 'Tamil'), ('te', 'Telugu'), ('tg', 'Tajik'), ('th', 'Thai'), ('tk', 'Turkmen'), ('tr', 'Turkish'), ('tt', 'Tatar'), ('udm', 'Udmurt'), ('ug', 'Uyghur'), ('uk', 'Ukrainian'), ('ur', 'Urdu'), ('uz', 'Uzbek'), ('vi', 'Vietnamese'), ('zh-hans', 'Simplified Chinese'), ('zh-hant', 'Traditional Chinese')] LANGUAGES\_BIDI ['he', 'ar', 'ar-dz', 'ckb', 'fa', 'ug', 'ur']  
LANGUAGE\_CODE 'en-us' LANGUAGE\_COOKIE\_AGE None LANGUAGE\_COOKIE\_DOMAIN None LANGUAGE\_COOKIE\_HTTPONLY False  
LANGUAGE\_COOKIE\_NAME 'django\_language' LANGUAGE\_COOKIE\_PATH '/' LANGUAGE\_COOKIE\_SAMESITE None  
LANGUAGE\_COOKIE\_SECURE False LOCALE\_PATHS [] LOGGING {} LOGGING\_CONFIG 'logging.config.dictConfig' LOGIN\_REDIRECT\_URL '/accounts/profile/' LOGIN\_URL '/accounts/login/' LOGOUT\_REDIRECT\_URL None MANAGERS [] MEDIA\_ROOT 'D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\DJANGO\\media' MEDIA\_URL '/media/' MESSAGE\_STORAGE 'django.contrib.messages.storage.fallback.FallbackStorage'  
MIDDLEWARE ['django.middleware.security.SecurityMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware', 'django.middleware.common.CommonMiddleware', 'django.middleware.csrf.CsrfViewMiddleware', 'django.contrib.auth.middleware.AuthenticationMiddleware', 'django.contrib.messages.middleware.MessageMiddleware', 'django.middleware.clickjacking.XFrameOptionsMiddleware'] MIGRATION\_MODULES {} MONTH\_DAY\_FORMAT 'F j' NUMBER\_GROUPING 0  
PASSWORD\_HASHERS '\*\*\*\*\*' PASSWORD\_RESET\_TIMEOUT '\*\*\*\*\*' PREPEND\_WWW False  
ROOT\_URLCONF 'portfolio.urls' SECRET\_KEY '\*\*\*\*\*' SECRET\_KEY\_FALLBACKS '\*\*\*\*\*'  
SECURE\_CONTENT\_TYPE\_NOSNIFF True SECURE\_CROSS\_ORIGIN\_OPENER\_POLICY 'same-origin' SECURE\_HSTS\_INCLUDE\_SUBDOMAINS False SECURE\_HSTS\_PRELOAD False SECURE\_HSTS\_SECONDS 0 SECURE\_PROXY\_SSL\_HEADER None SECURE\_REDIRECT\_EXEMPT [] SECURE\_REFERRER\_POLICY 'same-origin' SECURE\_SSL\_HOST None SECURE\_SSL\_REDIRECT False SERVER\_EMAIL 'root@localhost'  
SESSION\_CACHE\_ALIAS 'default' SESSION\_COOKIE\_AGE 1209600 SESSION\_COOKIE\_DOMAIN None SESSION\_COOKIE\_HTTPONLY True SESSION\_COOKIE\_NAME 'sessionid' SESSION\_COOKIE\_PATH '/' SESSION\_COOKIE\_SAMESITE 'Lax' SESSION\_COOKIE\_SECURE False SESSION\_ENGINE 'django.contrib.sessions.backends.db' SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE False SESSION\_FILE\_PATH None SESSION\_SAVE\_EVERY\_REQUEST False SESSION\_SERIALIZER 'django.contrib.sessions.serializers.JSONSerializer' SETTINGS\_MODULE 'portfolio.settings' SHORT\_DATETIME\_FORMAT 'm/d/Y P' SHORT\_DATE\_FORMAT 'm/d/Y' SIGNING\_BACKEND 'django.core.signing.TimestampSigner' SILENCED\_SYSTEM\_CHECKS [] STATICFILES\_DIRS ['D:\\PYTHON\\DJANGO\\HOPES\\PORTFOLIO\\DJANGO\\static/] STATICFILES\_FINDERS ['django.contrib.staticfiles.finders.FileSystemFinder', 'django.contrib.staticfiles.finders.AppDirectoriesFinder'] STATIC\_ROOT None STATIC\_URL '/static/' STORAGES {'default': {'BACKEND': 'django.core.files.storage.FileSystemStorage'}, 'staticfiles':}

```
{'BACKEND': 'django.contrib.staticfiles.storage.StaticFilesStorage'} } TEMPLATES [{APP_DIRS': True, 'BACKEND': 'django.template.backends.djangoproject.DjangoTemplates', 'DIRS': ['templates/'], 'OPTIONS': {'context_processors': ['django.template.context_processors.request', 'django.contrib.auth.context_processors.auth', 'django.contrib.messages.context_processors.messages']} }] TEST_NON_SERIALIZED_APPS [] TEST_RUNNER 'django.test.runner.DiscoverRunner' THOUSAND_SEPARATOR ',' TIME_FORMAT 'P' TIME_INPUT_FORMATS ['%H:%M:%S', '%H:%M:%S.%f', '%H:%M'] TIME_ZONE 'UTC' USE_I18N True USE_THOUSAND_SEPARATOR False USE_TZ True USE_X_FORWARDED_HOST False USE_X_FORWARDED_PORT False WSGI_APPLICATION 'portfolio.wsgi.application' X_FRAME_OPTIONS 'DENY' YEAR_MONTH_FORMAT 'F Y' You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard page generated by the handler for this status code.
```

Réflexion durant 13s

L'erreur est très claire : **Django essaie de créer un Projetscards avec images=None**, alors que ton modèle dit :

```
images = models.ImageField(..., null=True, blank=False)
```

→ Donc si **images** est vide au moment du `.create()` / `update_or_create()`, `full_clean()` déclenche :

```
{'images': ['This field cannot be blank.']}
```

Dans ton traceback on voit que ça passe par `Projetscards.objects.update_or_create(...)` (views.py ligne 237).

Le cas classique : tu fais `update_or_create(fingerprint=..., defaults={...})` mais **tu n'inclus pas images dans defaults**, ou tu l'inclus mais `img` vaut `None` (ou '') , donc Django tente une création sans image.

---

## ✓ Correction “sûre” dans `views.py` (témoignage)

Remplace ton bloc `btn_ajouter_temoignage` par celui-ci (il gère : image obligatoire + anti-doublon fingerprint + pas de crash) :

```
elif 'btn_ajouter_temoignage' in request.POST:
    nom = (request.POST.get('nom') or '').strip()
    description = (request.POST.get('description') or '').strip()
    img = request.FILES.get('images')

    # 1) Image obligatoire (puisque blank=False dans le model)
    if not img:
        messages.error(request, "Veuillez sélectionner une image (obligatoire) pour le témoignage.")
        return redirect('dashboard_admin')

    # 2) Calcul fingerprint (même logique que ton models.py)
```

```
fp = Projetscards.make_fingerprint(nom, description)

# 3) Si déjà existant -> message, sinon -> create
if Projetscards.objects.filter(fingerprint=fp).exists():
    messages.warning(request, "Ce témoignage existe déjà (doublon détecté).")
    return redirect('dashboard_admin')

Projetscards.objects.create(
    nom=nom,
    description=description,
    images=img,
    est_visible=True,
    fingerprint=fp, # facultatif (ton save() le calcule si vide), mais ici c'est explicite
)
messages.success(request, "Témoignage ajouté.")
return redirect('dashboard_admin')
```

✓ Avec ça :

- si l'image manque → **message propre**, pas de crash
  - si le fingerprint existe déjà → **message doublon**, pas de crash
  - sinon → création OK
- 

## Option alternative (si tu veux autoriser témoignage sans image)

Si tu veux permettre l'ajout sans image, alors il faut **changer le modèle** :

```
images = models.ImageField(upload_to="static/images/", null=True, blank=True)
```

Puis :

- python manage.py makemigrations
  - python manage.py migrate
- 

Pourquoi ça t'arrive alors que tu vois bien une image dans FILES ?

Parce que le crash est déclenché au moment où Django exécute un `create()` interne **sans images** (souvent via `update_or_create/get_or_create`), ou avec `images=None`. Même si ton formulaire a bien un fichier, ton code peut ne pas le passer dans `defaults`.

---

Si tu me colles exactement le **bloc actuel** autour de la ligne ~237 (juste 20-30 lignes), je te donne la version “`update_or_create`” correcte (si tu tiens absolument à `update_or_create` au lieu du `check+create`).

views.py

Python

Réflexion durant 4m 32s

Voici les **3 fichiers corrigés complets** (ceux qui te posent problème avec `fingerprint` + l’erreur `images cannot be blank`) :

- `models.py` (champ `fingerprint` corrigé en `null=True` pour migrations “douces” + génération auto)
- `views.py` (ajout témoignage **sans** `update_or_create` → on évite le crash “`images blank`”)
- `json_transfer.py` (import/export avec `fingerprint`, normalisations, `upsert`)

Je te les fournis aussi en fichiers prêts à copier :

- Télécharger `views_corrigé.py`
- Télécharger `models_corrigé.py`
- Télécharger `json_transfer_corrigé.py`

---

## Pourquoi tu avais l’erreur `images cannot be blank` ?

Tu faisais un `update_or_create(...)` sur `Projetscards`.

Quand l’objet **n’existe pas**, Django tente de **créer** tout de suite — et ton `save()` appelle `full_clean()`.

À ce moment-là, `images` n’était pas encore mis → donc **ValidationError**.

→ La correction sûre : **ne pas utiliser** `update_or_create` ici.

On fait : `find by fingerprint` → si existe : `update` ; sinon : `create` (avec `image` obligatoire).

---

## Après remplacement des fichiers : commandes à exécuter

Comme on a changé `fingerprint` (on le met en `null=True`), fais :

```
python manage.py makemigrations  
python manage.py migrate
```

---

## (Optionnel mais recommandé) Remplir le fingerprint pour l'existant

Si tu as déjà des témoignages en base AVANT `fingerprint`, fais une seule fois dans `python manage.py shell` :

```
from app_acceuil.models import Projetscards  
  
seen = {}  
for t in Projetscards.objects.all():  
    fp = Projetscards.make_fingerprint(t.nom or "", t.description or "")  
    if fp in seen:  
        # doublon réel : à toi de décider (supprimer / fusionner)  
        print("DOUBLON:", t.id, "===", seen[fp])  
    else:  
        t.fingerprint = fp  
        t.save()  
        seen[fp] = t.id
```

 Si ça affiche des doublons, c'est normal : ça veut dire que tu avais **deux témoignages identiques** (nom+description “équivalents”). Dans ce cas, il faut en supprimer un ou modifier le texte.

---

Si tu veux, je peux aussi te fournir une version “encore plus stricte” qui :

- autorise la création d'un témoignage **sans image** (et garde une image par défaut),
- ou bien rend `images` non obligatoire côté modèle (`blank=True`) tout en gardant une validation côté formulaire.

Et pourrais je continuer en vous disant le message qui est apparu ?

Oui, bien sûr — colle-moi le **message d'erreur exact** (idéalement toute la page “ValidationError ...” ou au moins :

- le titre (ex: ValidationError at /...)
- le bloc { ... } avec les champs en erreur
- la ligne “Raised during: ...” + le fichier/ligne si tu l’as

Et si c'est lié à un **import JSON**, colle aussi le **morceau du JSON** autour de l'objet concerné (juste l'entrée qui pose problème).

20260204\_export\_portfolio.json

Fichier

20260204\_export\_users.json

Fichier

Voici les deux messages qui s'affichent quand je fais l'importation des deux fichiers JSON : Premier Message : Import Portfolio terminé avec succès. Deuxième Message : Erreur import utilisateurs: Cannot assign "Administrateur": "User.role" must be a "Role" instance.  
Réflexion durant 59s

Oui — et l'erreur “**Cannot assign 'Administrateur': 'User.role' must be a 'Role' instance.**” vient d'un point précis :

 **À l'export**, ton JSON met "role": "Administrateur" (string) au lieu de "role": 1 (id), parce que ton export sérialise un **ForeignKey** avec `getattr(obj, name)` (ça renvoie l'objet Role, puis JSON => `str(Role)`).

 **Correction sûre** : à l'export, pour chaque `ForeignKey`, exporter `field.attname` (ex: `role_id`, `menu_id`) et à l'import Django recevra bien un entier.

Et pour tes erreurs côté témoignages :

- **duplicate fingerprint** : normal si tu crées 2 témoignages identiques (nom+description) → il faut gérer “déjà existant”.
- **images cannot be blank** : arrive quand on fait un “create” sans image (ou import sans base64) et que `blank=False + full_clean()` bloque.  
 **Correction sûre** : mettre `blank=True` sur les `ImageField` (pour éviter crash import), et **valider dans la vue** “si création alors image obligatoire”.

Ci-dessous je te donne les **3 fichiers complets corrigés** : `json_transfer.py`, `models.py`, `views.py`.

## 1) `app_acceuil/json_transfer.py` (**COMPLET corrigé**)

```
import base64
import json
import mimetypes
import re
```

```
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models import ForeignKey
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponseRedirect:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponseRedirect(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation + Fingerprint
# -----
```

```

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    """
    ✓ FIX MAJEUR :
    - ForeignKey => exporter la valeur *_id (entier) et non l'objet (string)
    - Files/Images => base64 + meta
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue

        # ✓ ForeignKey => on exporte field.attname (ex: role_id)
        if isinstance(field, ForeignKey):
            data[name] = getattr(obj, field.attname) # ex: obj.role_id
            continue

        value = getattr(obj, name)

        # Fichiers / images
        if isinstance(field, (FileField, ImageField)):
            if value and hasattr(value, "name", None):
                file_path = value.name

```

```

b64 = None
try:
    with default_storage.open(file_path, "rb") as f:
        raw = f.read()
    b64 = base64.b64encode(raw).decode("utf-8")
except Exception:
    b64 = None

mime, _ = mimetypes.guess_type(file_path)
data[name] = {
    "storage_path": file_path,
    "filename": file_path.split("/")[-1],
    "mimetype": mime or "application/octet-stream",
    "base64": b64,
}
else:
    data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data

```

```

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

```

```

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",

```

```

    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_contact.models import User, Role, Contact
    from app_user.models import Menu, RoleMenuAcces

    payload = {
        "export_type": "users",
        "exported_at": datetime.now().isoformat(),
        "version": 1,
        "notes": ["User.password est exporté tel quel (hash)."],
        "data": {
            "app_contact.Role": _serialize_queryset(Role.objects.all()),
            "app_contact.User": _serialize_queryset(User.objects.all()),
            "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
            "app_user.Menu": _serialize_queryset(Menu.objects.all()),
            "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    return _safe_json_response(payload, f"{date_prefix}_export_users.json")

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):

```

```

raw = uploaded_file.read()
try:
    text = raw.decode("utf-8")
except Exception:
    text = raw.decode("latin-1")
return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)

# -----
# FUSION INTELLIGENTE : clés métier
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"], # si ton modèle a bien ce champ

    # Témoignages: match dur par fingerprint
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"], # ici role/menu sont des IDs (FK)
}

```

```

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None
        lookup[k] = v

    try:
        return Model.objects.filter(**lookup).first()
    except Exception:
        return None


def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # --- Projetscards : calc fingerprint si absent ---
        if model_label == "app_acceuil.Projetscards":
            if not fields_data.get("fingerprint"):
                fields_data["fingerprint"] = _fingerprint_projetscards(
                    fields_data.get("nom", ""),
                    fields_data.get("description", ""),
                )

```

```

instance = None

# 1) update par ID si existant
if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()

# 2) update par clé métier (fingerprint/email/...)
if instance is None:
    instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

# 3) create
if instance is None:
    instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs + normalisation à l'enregistrement
for k, v in fields_data.items():

    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)

    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
        v = v.strip().lower()

    if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
        v = v.strip()

    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

```

```
@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")
    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    # ----- Portfolio import -----
    if json_portfolio:
        try:
            data = _parse_uploaded_json(json_portfolio)
            if data.get("export_type") != "portfolio":
                raise ValueError("Ce fichier n'est pas un export portfolio.")
            model_map = data.get("data", {})
            if not isinstance(model_map, dict):
                raise ValueError("Format JSON invalide: data doit être un objet (dict).")

            if replace_portfolio:
                from app_acceuil.models import (
                    Projetphotodeprofil,
                    ProjetAproposDeMoi,
                    Projetsfirstspeciality,
                    Projetmesrealisations,
                    Projetcards,
                    MesCompetencesCles,
                    ReseauSocial,
                )
                ReseauSocial.objects.all().delete()
                MesCompetencesCles.objects.all().delete()
                Projetcards.objects.all().delete()
                Projetmesrealisations.objects.all().delete()
                Projetsfirstspeciality.objects.all().delete()
                ProjetAproposDeMoi.objects.all().delete()
                Projetphotodeprofil.objects.all().delete()

            for model_label, objects_list in model_map.items():
                _upsert_objects(model_label, objects_list, keep_ids=True)

        except Exception as e:
            return JsonResponse({"error": str(e)}, status=400)
```

```

        messages.success(request, "Import Portfolio terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_users:
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
            Role.objects.all().delete()

        ordered = [
            "app_contact.Role",
            "app_user.Menu",
            "app_contact.User",
            "app_contact.Contact",
            "app_user.RoleMenuAcces",
        ]
        for model_label in ordered:
            if model_label in model_map:
                _upsert_objects(model_label, model_map[model_label], keep_ids=True)

        messages.success(request, "Import Utilisateurs terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

---

## 2) `app_acceuil/models.py` (COMPLET corrigé)

👉 Ici je rends les images blank=True (ça évite les crash à l'import).

✓ Et je laisse la **vérification image obligatoire** dans la vue lors d'une création manuelle.

```
from django.db import models
import re
import hashlib

def nettoyer_espaces(chaine):
    if chaine:
        return " ".join(chaine.split())
    return chaine

# 1. TEMOIGNAGES
class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=True)
    description = models.TextField(null=True, blank=False)

    fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ["-created_at"]

    @staticmethod
    def _norm_text(s: str) -> str:
        s = (s or "").strip()
        s = re.sub(r"\s+", " ", s)
        return s.lower()

    @classmethod
    def make_fingerprint(cls, nom: str, description: str) -> str:
        base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
        return hashlib.sha256(base.encode("utf-8")).hexdigest()

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
```

```

    if not self.fingerprint:
        self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    else:
        self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

# 2. SPECIALITES
class Projetsfirstspeciality(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=True)
    description_speciality = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Ma Photo de Spécialisation"
        verbose_name_plural = "Mes Photos de Spécialisation"
        ordering = ["-created_at"]

    def __str__(self):
        return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"


# 3. REALISATIONS
class Projetmesrealisations(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=True)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    compteur_demo_live = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Projet"
        verbose_name_plural = "Mes Projets"
        ordering = ["-created_at"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):

```

```
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom if self.nom else f"Projet {self.id}"

# 4. PHOTO DE PROFIL
class Projetphotodeprofil(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=True)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "Photo de profil"
        verbose_name_plural = "Photo de profil"

    def __str__(self):
        return f"Photo {self.id}"


# 5. A PROPOS DE MOI
class ProjetAproposDeMoi(models.Model):
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "À propos de moi"
        verbose_name_plural = "À propos de moi"

    def __str__(self):
        return self.description[:50] if self.description else f"A propos {self.id}"


# 6. COMPETENCES
class MesCompetencesCles(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Compétence"
        verbose_name_plural = "Compétences"
        ordering = ["nom"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)
```

```

def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Compétence {self.id}"

# 7. RESEAUX SOCIAUX
class ReseauSocial(models.Model):
    nom = models.CharField(max_length=100, unique=True)
    url = models.CharField(max_length=255)
    est_visible = models.BooleanField(default=True)

    class Meta:
        verbose_name = "Réseau social"
        verbose_name_plural = "Réseaux sociaux"
        ordering = ["nom"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom

```

---

### 3) `app_acceuil/views.py` (COMPLET corrigé pour fingerprint + image obligatoire à la création)

 Ici, on empêche :

- création d'un témoignage **sans image** (message utilisateur au lieu d'un crash),
- création d'un duplicat fingerprint → si déjà existant, on **met à jour** au lieu d'explorer en ValidationError.

```

from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.db.models import F, Sum

from .models import (

```

```

Projetscards, Projetsfirstspeciality, Projetmesrealisations,
Projetphotodeprofil, ProjetAproposDeMoi, MesCompetencesCles, ReseauSocial
)

from app_contact.models import User as ContactUser
from app_user.models import RoleMenuAcces

def _get_role_id_from_session(request):
    user_session = request.session.get("user")
    if not user_session:
        return None

    user_id = user_session.get("id")
    if not user_id:
        return None

    u = ContactUser.objects.filter(id=user_id).select_related("role").first()
    if not u:
        return None

    return u.role_id

# =====
# 1. VUE PUBLIQUE
# =====

def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetmesrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemon = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    mes_menus_visibles = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_visibles = RoleMenuAcces.objects.filter(
            role_id=role_id,
            est_visible=True
        ).select_related("menu")

    contenu = {

```

```

    "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
    "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
    "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
    "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
    "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
    "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
    "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
    "A_PROPOS_DE_MOI": apropoisdemoi,
    "Mes_Competences_Cles": mescompetencescles,
    "Mes_Reseaux": reseaux,
    "mes_menus_visibles": mes_menus_visibles,
}
return render(request, "index.html", contenu)

```

```

# =====
# 2. DASHBOARD ADMIN
# =====

def page_Mon_Espace_Administration(request):
    photo_profil = Projetphotodeprofil.objects.first()
    a_propos = ProjetAproposDeMoi.objects.first()
    specialite = Projetsfirstspeciality.objects.first()

    user_data = request.session.get("user")
    if not user_data or (user_data.get("role") not in ("Administrateur", "Moderateur")):
        messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.")
        return redirect("name_acceuil")

    if request.method == "POST":

        # --- PHOTO DE PROFIL (VISIBILITÉ) ---
        if "btn_visibilite_photo_auto" in request.POST:
            vis = "est_visible" in request.POST
            if photo_profil:
                photo_profil.est_visible = vis
                photo_profil.save()
            messages.info(request, "Visibilité de la photo mise à jour.")

        # --- PHOTO DE PROFIL (UPLOAD/UPDATE) ---
        elif "btn_photo" in request.POST:
            img = request.FILES.get("images")
            if photo_profil:
                if img:
                    photo_profil.images = img
                    photo_profil.save()
            else:

```

```
if not img:
    messages.error(request, "Veuillez choisir une image.")
    return redirect("dashboard_admin")
Projetphotodeprofil.objects.create(images=img, est_visible=True)
messages.success(request, "Photo enregistrée.")

elif "btn_supprimer_photo" in request.POST:
    if photo_profil:
        photo_profil.delete()
    messages.warning(request, "Photo supprimée.")

# --- BIO (VISIBILITÉ) ---
elif "btn_visibilite_bio_auto" in request.POST:
    vis = "est_visible_bio" in request.POST
    if a_propos:
        a_propos.est_visible = vis
        a_propos.save()
    messages.info(request, "Visibilité de la bio mise à jour.")

# --- BIO (TEXTE) ---
elif "btn_apropos" in request.POST:
    description = request.POST.get("description")
    if a_propos:
        a_propos.description = description
        a_propos.save()
    else:
        ProjetAproposDeMoi.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")

elif "btn_supprimer_apropos" in request.POST:
    if a_propos:
        a_propos.delete()
    messages.error(request, "Texte biographique supprimé.")

# --- SPÉCIALITÉ (VISIBILITÉ) ---
elif "btn_visibilite_section_spec" in request.POST:
    vis = "est_visible" in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")

# --- SPÉCIALITÉ (UPDATE) ---
elif "btn_specialite" in request.POST:
    img = request.FILES.get("images")
    desc = request.POST.get("description_speciality")
    if specialite:
        specialite.description_speciality = desc
```

```

if img:
    specialite.images = img
specialite.save()
else:
    if not img:
        messages.error(request, "Veuillez choisir une image.")
        return redirect("dashboard_admin")
Projetsfirstspeciality.objects.create(description_speciality=desc, images=img, est_visible=True)
messages.success(request, "Détails de la spécialité mis à jour.")

# --- RÉALISATIONS (VISIBILITÉ) ---
elif "btn_visibilite_section_real" in request.POST:
    vis = "est_visible" in request.POST
    Projetmesrealisations.objects.all().update(est_visible=vis)
    messages.info(request, "Section Projets mise à jour.")

elif "btn_bascule_vis_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_real" in request.POST:
    img = request.FILES.get("images")
    if not img:
        messages.error(request, "Veuillez choisir une image pour le projet.")
        return redirect("dashboard_admin")

    Projetmesrealisations.objects.create(
        nom=request.POST.get("nom"),
        description=request.POST.get("description"),
        images=img,
        est_visible=True
    )
    messages.success(request, "Projet ajouté.")

elif "btn_modifier_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.nom = request.POST.get("nom")
    obj.description = request.POST.get("description")
    if request.FILES.get("images"):
        obj.images = request.FILES.get("images")
    obj.save()
    messages.info(request, "Projet modifié.")

elif "btn_supprimer_real" in request.POST:
    get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation")).delete()

```

```
messages.warning(request, "Projet supprimé.")

# --- TÉMOIGNAGES (VISIBILITÉ) ---
elif "btn_visibilite_section_tem" in request.POST:
    vis = "est_visible" in request.POST
    Projetscards.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_tem" in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get("id_temoignage"))
    obj.est_visible = not obj.est_visible
    obj.save()

#  TÉMOIGNAGES (AJOUT SÛR anti-doublon)
elif "btn_ajouter_temoignage" in request.POST:
    nom = request.POST.get("nom")
    description = request.POST.get("description")
    img = request.FILES.get("images")

    #  si création, image obligatoire (pas de crash)
    if not img:
        messages.error(request, "Veuillez choisir une image pour le témoignage.")
        return redirect("dashboard_admin")

    fp = Projetscards.make_fingerprint(nom or "", description or "")

    existing = Projetscards.objects.filter(fingerprint=fp).first()
    if existing:
        # déjà existant => on met à jour (optionnel)
        existing.nom = (nom or "").strip()
        existing.description = (description or "").strip()
        # si tu veux remplacer l'image quand re-ajout :
        existing.images = img
        existing.est_visible = True
        existing.save()
        messages.info(request, "Témoignage déjà existant : mise à jour effectuée.")
    else:
        Projetscards.objects.create(
            nom=nom,
            description=description,
            images=img,
            est_visible=True,
            fingerprint=fp
        )
        messages.success(request, "Témoignage ajouté.")
```

```
elif "btn_supprimer_temoignage" in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get("id_temoignage")).delete()
    messages.warning(request, "Témoignage supprimé.")

# --- COMPÉTENCES ---
elif "btn_visibilite_section_comp" in request.POST:
    vis = "est_visible" in request.POST
    MesCompetencesCles.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_comp" in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_comp" in request.POST:
    MesCompetencesCles.objects.create(nom=request.POST.get("nom_competence"), est_visible=True)
    messages.success(request, "Compétence ajoutée.")

elif "btn_supprimer_comp" in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence")).delete()
    messages.warning(request, "Compétence supprimée.")

# --- RÉSEAUX SOCIAUX ---
elif "btn_visibilite_section_res" in request.POST:
    vis = "est_visible" in request.POST
    ReseauSocial.objects.all().update(est_visible=vis)

elif "btn_reseau" in request.POST:
    nom = (request.POST.get("nom_reseau") or "").strip()
    url = (request.POST.get("url_reseau") or "").strip()

    if nom.lower() == "email" and url.lower().startswith("mailto:"):
        url = url.split(":", 1)[1].strip()
    if nom.lower() == "whatsapp" and url and url.startswith("wa.me/"):
        url = "https://" + url

    ReseauSocial.objects.update_or_create(
        nom=nom,
        defaults={"url": url, "est_visible": True}
    )
    messages.success(request, "Réseau mis à jour.")

elif "btn_supprimer_reseau" in request.POST:
    get_object_or_404(ReseauSocial, id=request.POST.get("id_reseau")).delete()
    messages.warning(request, "Réseau supprimé.")
```

```

        return redirect("dashboard_admin")

context = {
    "AFFICHER_la_PHOTO_DE_PROFIL": photo_profil,
    "A_PROPOS_DE_MOI": a_propos,
    "SPECIALITE": specialite,
    "AFFICHER_les_REALISATIONS_PERSONNELLES": Projetmesrealisations.objects.all().order_by("-id"),
    "REALISATIONS_STATS": Projetmesrealisations.objects.all().order_by("-compteur_demo_live", "-id"),
    "TOTAL_CLICS_DEMO_LIVE": (Projetmesrealisations.objects.aggregate(total=Sum("compteur_demo_live"))["total"] or 0),
    "TEMOIGNAGES_des_PERSONNES_cards": Projetscards.objects.all().order_by("-id"),
    "Mes_Competences_Cles": MesCompetencesCles.objects.all().order_by("nom"),
    "Mes_Reseaux": ReseauSocial.objects.all(),
}
return render(request, "page_administration.html", context)

```

```

# =====
# 2bis. COMPTEUR "DÉMO LIVE"
# =====
def incrementer_compteur_demo_live(request, realisation_id):
    Projetmesrealisations.objects.filter(id=realisation_id).update(
        compteur_demo_live=F("compteur_demo_live") + 1
    )
    return redirect("name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES")

```

```

# =====
# 3. REDIRECTIONS
# =====
def Fonction_Page_accueil_section_Accueil(request): return redirect("/#accueil")
def Fonction_Page_accueil_section_projets(request): return redirect("/#projets")
def Fonction_Page_accueil_section_temoignages(request): return redirect("/#temoignages")
def Fonction_Page_accueil_section_a_propos(request): return redirect("/#a-propos")
def Fonction_Page_accueil_section_contact(request): return redirect("/#contact")

def Fonction_Page_email_Javascript(request): return render(request, "email_Javascript.htm")
def Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request): return render(request, "TransportRoutier_Main.html")
def Fonction_Page_diapo_Javascript_Page1(request): return render(request, "diapo_Javascript_Page1.html")

```

```

# =====
# RÉINITIALISATION DU COMPTEUR POUR UN PROJET SPÉCIFIQUE
# =====
def reinitialiser_compteur_projet(request, realisation_id):
    projet = get_object_or_404(Projetmesrealisations, id=realisation_id)
    projet.compteur_demo_live = 0

```

```
projet.save()
messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.")
return redirect("dashboard_admin")
```

---

## ✓ À faire après ces changements (important)

1. **Créer la migration** après modification des `blank=True` (et `fingerprint` si pas déjà migré) :

```
python manage.py makemigrations
python manage.py migrate
```

2. **Regénérer tes exports JSON** (très important) :

Parce que maintenant les ForeignKey seront bien exportés en ID, ce qui corrigera ton import users.

---

Si tu veux, colle-moi aussi ton modèle `User` (`app_contact/models.py`) et `RoleMenuAcces` (`app_user/models.py`) et je te vérifie que **tous** les champs FK sont cohérents avec l'import (ex: `role_id`, `menu_id`, etc.).

Sources

`models.py`

Python

`app_contact/models.py`

Réflexion durant 1m 33s

Oui — je te donne **les fichiers complets corrigés**, avec :

1. **Anti-doublon “dur” sur Projetcards** via `fingerprint` (plus d'erreur “Fingerprint already exists”).
  2. **Ajout/MàJ de témoignage dans le dashboard** : si le `fingerprint` existe → **mise à jour**, sinon **création** (et **image obligatoire uniquement à la création**).
  3. **Import Users JSON** : correction du problème  
Cannot assign "'Administrateur'": "User.role" must be a "Role" instance.  
→ on convertit automatiquement les valeurs FK textuelles en **vraies instances** (`Role`, etc.).
-

## 1) app\_acceuil/json\_transfer.py (COMPLET corrigé)

```
import base64
import json
import mimetypes
import re
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models import ForeignKey
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    if value is None:
        return None

    if isinstance(value, (datetime, date, time)):
        return value.isoformat()

    if isinstance(value, Decimal):
        return str(value)

    if isinstance(value, UUID):
        return str(value)

    return value
```

```
def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
# Normalisation + Fingerprint
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue

        value = getattr(obj, name)

        # Fichiers / images
        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name
```

```

b64 = None
try:
    with default_storage.open(file_path, "rb") as f:
        raw = f.read()
    b64 = base64.b64encode(raw).decode("utf-8")
except Exception:
    b64 = None

mime, _ = mimetypes.guess_type(file_path)
data[name] = {
    "storage_path": file_path,
    "filename": file_path.split("/")[-1],
    "mimetype": mime or "application/octet-stream",
    "base64": b64,
}
else:
    data[name] = None

# ForeignKey : exporte l'id par défaut (au lieu de l'objet)
elif isinstance(field, ForeignKey):
    data[name] = getattr(obj, field.attname) # ex: role_id

else:
    data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetscards,
    MesCompetencesCles,

```

```

        ReseauSocial,
    )

payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 1,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    from app_contact.models import User, Role, Contact
    from app_user.models import Menu, RoleMenuAcces

    payload = {
        "export_type": "users",
        "exported_at": datetime.now().isoformat(),
        "version": 1,
        "notes": ["User.password est exporté tel quel (hash)."],
        "data": {
            "app_contact.Role": _serialize_queryset(Role.objects.all()),
            "app_contact.User": _serialize_queryset(User.objects.all()),
            "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
            "app_user.Menu": _serialize_queryset(Menu.objects.all()),
            "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
        },
    }

    date_prefix = datetime.now().strftime("%Y%m%d")
    return _safe_json_response(payload, f"{date_prefix}_export_users.json")

```

```

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)

# -----
# FUSION INTELLIGENTE : clés métier (hors ID)
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"],

    # Témoignages: match dur par fingerprint
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
}

```

```

    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"],
}

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None
        lookup[k] = v

    try:
        return Model.objects.filter(**lookup).first()
    except Exception:
        return None


def _resolve_fk_value(fk_field: ForeignKey, v):
    """
    Rend une valeur assignable sur un FK.
    - int => id direct
    - str => on tente de retrouver l'objet via un champ "nom_role"/"code_menu"/"email"/"nom"
    """
    if v is None:
        return None

    # Cas courant: déjà un id
    if isinstance(v, int):
        return v

    # Si c'est une string: on retrouve l'instance
    if isinstance(v, str):
        s = v.strip()
        if s == "":
            return None

        Related = fk_field.remote_field.model

```

```

related_label = f"{Related._meta.app_label}.{Related.__name__}"

# Mapping "texte -> champ"
# (tu peux étendre si besoin)
preferred_fields = []
if related_label == "app_contact.Role":
    preferred_fields = ["nom_role"]
elif related_label == "app_user.Menu":
    preferred_fields = ["code_menu"]
elif related_label == "app_contact.User":
    preferred_fields = ["email"]
else:
    # fallback fréquents
    preferred_fields = ["nom", "name", "code"]

# Recherche (iexact)
for f in preferred_fields:
    if hasattr(Related, f):
        try:
            obj = Related.objects.filter(**{f:f.__iexact__: s}).first()
            if obj:
                return obj.pk
        except Exception:
            pass

    # dernier recours: essayer pk string
    try:
        return int(s)
    except Exception:
        return None

# Sinon: on laisse (mais généralement inutile)
return v

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

```

```

# séparer file fields
file_payloads = {}
for field in Model._meta.fields:
    if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
        file_payloads[field.name] = fields_data.pop(field.name)

# --- Projetscards : calc fingerprint si absent ---
if model_label == "app_acceuil.Projetscards":
    if not fields_data.get("fingerprint"):
        fields_data["fingerprint"] = _fingerprint_projetscards(
            fields_data.get("nom", ""),
            fields_data.get("description", ""),
        )

instance = None

# 1) update par ID si existant
if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()

# 2) update par clé métier
if instance is None:
    instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

# 3) create
if instance is None:
    instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs (en gérant FK)
for k, v in fields_data.items():
    try:
        field = Model._meta.get_field(k)
    except Exception:
        field = None

    # normalisations sûres (anti-doublons)
    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)

    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

```

```

if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
    v = _norm_text(v)

if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
    v = _norm_text(v)

if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
    v = v.strip().lower()

if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
    v = v.strip()

# ForeignKey: on assigne un id (role_id etc.)
if isinstance(field, ForeignKey):
    fk_id = _resolve_fk_value(field, v)
    setattr(instance, field.attname, fk_id) # ex: role_id
else:
    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

```

```

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")
    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    # ----- Portfolio import -----
    if json_portfolio:
        try:
            data = _parse_uploaded_json(json_portfolio)
            if data.get("export_type") != "portfolio":
                raise ValueError("Ce fichier n'est pas un export portfolio.")
            model_map = data.get("data", {})

```

```

if not isinstance(model_map, dict):
    raise ValueError("Format JSON invalide: data doit être un objet (dict).")

if replace_portfolio:
    from app_acceuil.models import (
        Projetphotodeprofil,
        ProjetAproposDeMoi,
        Projetsfirstspeciality,
        Projetmesrealisations,
        Projetcards,
        MesCompetencesCles,
        ReseauSocial,
    )
    ReseauSocial.objects.all().delete()
    MesCompetencesCles.objects.all().delete()
    Projetcards.objects.all().delete()
    Projetmesrealisations.objects.all().delete()
    Projetsfirstspeciality.objects.all().delete()
    ProjetAproposDeMoi.objects.all().delete()
    Projetphotodeprofil.objects.all().delete()

    for model_label, objects_list in model_map.items():
        _upsert_objects(model_label, objects_list, keep_ids=True)

    messages.success(request, "Import Portfolio terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_users:
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
    
```

```

Role.objects.all().delete()

ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

 Résultat : même si dans ton JSON tu as "role": "Administrateur" (texte), l'import convertit ça en role\_id correctement.

---

## 2) app\_acceuil/models.py (COMPLET corrigé – Projetscards fingerprint + images)

Je garde ton principe full\_clean() mais on s'assure que le fingerprint est toujours cohérent.

```

from django.db import models
import re
import hashlib

def nettoyer_espaces(chaine):
    if chaine:
        return " ".join(chaine.split())
    return chaine


class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

```

```
# Anti-doublon (stable)
fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")

est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
created_at = models.DateTimeField(auto_now_add=True, null=True)

class Meta:
    verbose_name = "Témoignage"
    verbose_name_plural = "Témoignages Clients"
    ordering = ["-created_at"]

@staticmethod
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

@classmethod
def make_fingerprint(cls, nom: str, description: str) -> str:
    base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

def clean(self):
    self.nom = nettoyer_espaces(self.nom)

def save(self, *args, **kwargs):
    # fingerprint stable
    if not self.fingerprint:
        self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    else:
        self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

class Projetsfirstspeciality(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description_speciality = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

class Meta:
```

```
verbose_name = "Ma Photo de Spécialisation"
verbose_name_plural = "Mes Photos de Spécialisation"
ordering = ["-created_at"]

def __str__(self):
    return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"


class Projetmesrealisations(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    compteur_demo_live = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Projet"
        verbose_name_plural = "Mes Projets"
        ordering = ["-created_at"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom if self.nom else f"Projet {self.id}"


class Projetphotodeprofil(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "Photo de profil"
        verbose_name_plural = "Photo de profil"

    def __str__(self):
        return f"Photo {self.id}"


class ProjetAproposDeMoi(models.Model):
    description = models.TextField(null=True, blank=False)
```

```
est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

class Meta:
    verbose_name = "À propos de moi"
    verbose_name_plural = "À propos de moi"

def __str__(self):
    return self.description[:50] if self.description else f"A propos {self.id}"

class MesCompetencesCles(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Compétence"
        verbose_name_plural = "Compétences"
        ordering = ["nom"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom if self.nom else f"Compétence {self.id}"


class ReseauSocial(models.Model):
    nom = models.CharField(max_length=100, unique=True)
    url = models.CharField(max_length=255)
    est_visible = models.BooleanField(default=True)

    class Meta:
        verbose_name = "Réseau social"
        verbose_name_plural = "Réseaux sociaux"
        ordering = ["nom"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
```

```
super().save(*args, **kwargs)
```

```
def __str__(self):  
    return self.nom
```

---

### 3) app\_acceuil/views.py (COMPLET corrigé)

 Correction principale ici : **ajout témoignage** → “upsert” par fingerprint et **image obligatoire uniquement si nouveau** (ça évite ton ValidationError  
{'images': ['This field cannot be blank.']}).

```
from django.shortcuts import render, redirect, get_object_or_404  
from django.contrib import messages  
from django.db.models import F, Sum  
  
from .models import (  
    Projetscards,  
    Projetsfirstspeciality,  
    Projetmesrealisations,  
    Projetphotodeprofil,  
    ProjetAproposDeMoi,  
    MesCompetencesCles,  
    ReseauSocial,  
)  
  
from app_contact.models import User as ContactUser  
from app_user.models import RoleMenuAcces  
  
def _get_role_id_from_session(request):  
    user_session = request.session.get("user")  
    if not user_session:  
        return None  
  
    user_id = user_session.get("id")  
    if not user_id:  
        return None  
  
    u = ContactUser.objects.filter(id=user_id).select_related("role").first()  
    if not u:  
        return None  
  
    return u.role_id
```

```

# =====
# 1. VUE PUBLIQUE
# =====
def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetmesrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemoni = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    mes_menus_visibles = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_visibles = RoleMenuAcces.objects.filter(
            role_id=role_id, est_visible=True
        ).select_related("menu")

    contenu = {
        "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
        "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
        "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
        "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
        "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
        "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
        "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
        "A_PROPOS_DE_MOI": aproposdemoni,
        "Mes_Competences_Cles": mescompetencescles,
        "Mes_Reseaux": reseaux,
        "mes_menus_visibles": mes_menus_visibles,
    }
    return render(request, "index.html", contenu)

```

```

# =====
# 2. DASHBOARD ADMIN
# =====
def page_Mon_Espace_Administration(request):
    photo_profil = Projetphotodeprofil.objects.first()
    a_propos = ProjetAproposDeMoi.objects.first()
    specialite = Projetsfirstspeciality.objects.first()

```

```
user_data = request.session.get("user")
if not user_data or (user_data.get("role") not in ("Administrateur", "Modérateur")):
    messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.")
    return redirect("name_acceuil")

if request.method == "POST":

    # --- PHOTO DE PROFIL (SWITCH AUTO) ---
    if "btn_visibilite_photo_auto" in request.POST:
        vis = "est_visible" in request.POST
        if photo_profil:
            photo_profil.est_visible = vis
            photo_profil.save()
        messages.info(request, "Visibilité de la photo mise à jour.")

    # --- PHOTO DE PROFIL (UPLOAD/UPDATE) ---
    elif "btn_photo" in request.POST:
        img = request.FILES.get("images")
        if photo_profil:
            if img:
                photo_profil.images = img
                photo_profil.save()
        else:
            if not img:
                messages.error(request, "Image obligatoire pour créer la photo de profil.")
                return redirect("dashboard_admin")
            Projetphotodeprofil.objects.create(images=img, est_visible=True)
        messages.success(request, "Photo enregistrée.")

    elif "btn_supprimer_photo" in request.POST:
        if photo_profil:
            photo_profil.delete()
        messages.warning(request, "Photo supprimée.")

    # --- À PROPOS (VISIBILITÉ) ---
    elif "btn_visibilite_bio_auto" in request.POST:
        vis = "est_visible_bio" in request.POST
        if a_propos:
            a_propos.est_visible = vis
            a_propos.save()
        messages.info(request, "Visibilité de la bio mise à jour.")

    # --- À PROPOS (TEXTE) ---
    elif "btn_apropos" in request.POST:
        description = request.POST.get("description")
        if a_propos:
```

```

        a_propos.description = description
        a_propos.save()
    else:
        ProjetAproposDeMoi.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")

elif "btn_supprimer_apropos" in request.POST:
    if a_propos:
        a_propos.delete()
    messages.error(request, "Texte biographique supprimé.")

# --- SPÉCIALITÉ ---
elif "btn_visibilite_section_spec" in request.POST:
    vis = "est_visible" in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")

elif "btn_specialite" in request.POST:
    img = request.FILES.get("images")
    desc = request.POST.get("description_speciality")
    if specialite:
        specialite.description_speciality = desc
        if img:
            specialite.images = img
        specialite.save()
    else:
        if not img:
            messages.error(request, "Image obligatoire pour créer la spécialité.")
            return redirect("dashboard_admin")
        Projetsfirstspeciality.objects.create(description_speciality=desc, images=img, est_visible=True)
    messages.success(request, "Détails de la spécialité mis à jour.")

# --- RÉALISATIONS ---
elif "btn_visibilite_section_real" in request.POST:
    vis = "est_visible" in request.POST
    Projetmesrealisations.objects.all().update(est_visible=vis)
    messages.info(request, "Section Projets mise à jour.")

elif "btn_bascule_vis_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_real" in request.POST:
    img = request.FILES.get("images")
    if not img:

```

```
messages.error(request, "Image obligatoire pour créer un projet.")
return redirect("dashboard_admin")

Projetmesrealisations.objects.create(
    nom=request.POST.get("nom"),
    description=request.POST.get("description"),
    images=img,
    est_visible=True,
)
messages.success(request, "Projet ajouté.")

elif "btn_modifier_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.nom = request.POST.get("nom")
    obj.description = request.POST.get("description")
    if request.FILES.get("images"):
        obj.images = request.FILES.get("images")
    obj.save()
    messages.info(request, "Projet modifié.")

elif "btn_supprimer_real" in request.POST:
    get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation")).delete()
    messages.warning(request, "Projet supprimé.")

# --- TÉMOIGNAGES (Upsert par fingerprint) ---
elif "btn_visibilite_section_tem" in request.POST:
    vis = "est_visible" in request.POST
    Projetscards.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_tem" in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get("id_temoignage"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_temoignage" in request.POST:
    nom = request.POST.get("nom") or ""
    description = request.POST.get("description") or ""
    img = request.FILES.get("images")

    fp = Projetscards.make_fingerprint(nom, description)
    existing = Projetscards.objects.filter(fingerprint=fp).first()

    if existing:
        # M&J (image optionnelle)
        existing.nom = nom.strip()
        existing.description = description.strip()
```

```

if img:
    existing.images = img
existing.est_visible = True
existing.save()
messages.success(request, "Témoignage déjà existant : mis à jour (anti-doublon).")
else:
    # Création => image obligatoire
    if not img:
        messages.error(request, "Image obligatoire pour créer un nouveau témoignage.")
        return redirect("dashboard_admin")

Projetscards.objects.create(
    nom=nom.strip(),
    description=description.strip(),
    images=img,
    fingerprint=fp,
    est_visible=True,
)
messages.success(request, "Témoignage ajouté.")

elif "btn_supprimer_temoignage" in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get("id_temoignage")).delete()
    messages.warning(request, "Témoignage supprimé.")

# --- COMPÉTENCES ---
elif "btn_visibilite_section_comp" in request.POST:
    vis = "est_visible" in request.POST
    MesCompetencesCles.objects.all().update(est_visible=vis)

elif "btn_bascule_vis_comp" in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence"))
    obj.est_visible = not obj.est_visible
    obj.save()

elif "btn_ajouter_comp" in request.POST:
    MesCompetencesCles.objects.create(nom=request.POST.get("nom_competence"), est_visible=True)
    messages.success(request, "Compétence ajoutée.")

elif "btn_supprimer_comp" in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence")).delete()
    messages.warning(request, "Compétence supprimée.")

# --- RÉSEAUX SOCIAUX ---
elif "btn_visibilite_section_res" in request.POST:
    vis = "est_visible" in request.POST
    ReseauSocial.objects.all().update(est_visible=vis)

```

```

    elif "btn_reseau" in request.POST:
        nom = (request.POST.get("nom_reseau") or "").strip()
        url = (request.POST.get("url_reseau") or "").strip()

        if nom.lower() == "email" and url.lower().startswith("mailto:"):
            url = url.split(":", 1)[1].strip()
        if nom.lower() == "whatsapp" and url and url.startswith("wa.me/"):
            url = "https://" + url

        ReseauSocial.objects.update_or_create(
            nom=nom,
            defaults={"url": url, "est_visible": True},
        )
        messages.success(request, "Réseau mis à jour.")

    elif "btn_supprimer_reseau" in request.POST:
        get_object_or_404(ReseauSocial, id=request.POST.get("id_reseau")).delete()
        messages.warning(request, "Réseau supprimé.")

    return redirect("dashboard_admin")

context = {
    "AFFICHER_la_PHOTO_DE_PROFIL": photo_profil,
    "A_PROPOS_DE_MOI": a_propos,
    "SPECIALITE": specialite,
    "AFFICHER_les_REALISATIONS_PERSONNELLES": Projetmesrealisations.objects.all().order_by("-id"),
    "REALISATIONS_STATS": Projetmesrealisations.objects.all().order_by("-compteur_demo_live", "-id"),
    "TOTAL_CLICS_DEMO_LIVE": (Projetmesrealisations.objects.aggregate(total=Sum("compteur_demo_live"))["total"] or 0),
    "TEMOIGNAGES_des_PERSONNES_cards": Projetscards.objects.all().order_by("-id"),
    "Mes_Competences_Cles": MesCompetencesCles.objects.all().order_by("nom"),
    "Mes_Reseaux": ReseauSocial.objects.all(),
}
return render(request, "page_administration.html", context)

# =====
# COMPTEUR DÉMO LIVE
# =====
def incrementer_compteur_demo_live(request, realisation_id):
    Projetmesrealisations.objects.filter(id=realisation_id).update(
        compteur_demo_live=F("compteur_demo_live") + 1
)
return redirect("name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES")

```

```
# =====
# REDIRECTIONS
# =====
def Fonction_Page_accueil_section_Accueil(request):
    return redirect("/#accueil")

def Fonction_Page_accueil_section_projets(request):
    return redirect("/#projets")

def Fonction_Page_accueil_section_temoignages(request):
    return redirect("/#temoignages")

def Fonction_Page_accueil_section_a_propos(request):
    return redirect("/#a-propos")

def Fonction_Page_accueil_section_contact(request):
    return redirect("/#contact")

def Fonction_Page_email_Javascript(request):
    return render(request, "email_Javascript.htm")

def Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request):
    return render(request, "TransportRoutier_Main.html")

def Fonction_Page_diapo_Javascript_Page1(request):
    return render(request, "diapo_Javascript_Page1.html")

def reinitialiser_compteur_projet(request, realisation_id):
    projet = get_object_or_404(Projetmesrealisations, id=realisation_id)
    projet.compteur_demo_live = 0
    projet.save()
    messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.")
    return redirect("dashboard_admin")
```

---

## Réponses rapides à tes questions

## 1) "Si j'utilise une nouvelle base de données, dois-je remplir fingerprint ?"

- Si la DB est vraiment vide (nouvelle) → non, pas besoin de "remplir fingerprint".
- Si tu as déjà des témoignages existants dans la DB (avant ajout du champ fingerprint) → oui, il faut faire une mise à jour 1 fois (backfill).

## 2) Pourquoi tu avais : `User.role must be a Role instance` ?

Parce que ton JSON Users contenait "role": "Administrateur" (texte).

Avec le `json_transfer.py` corrigé ci-dessus, l'import fait la conversion en `role_id`.

---

Si tu veux, colle ici aussi ton `app_contact/models.py` (le vrai) (surtout `User` et `Role`) et ton `app_user/models.py` : je te confirmerai que **tous les FK sont bien couverts** (Contact, RoleMenuAcces, etc.).

models.py

Python

app\_user/models.py

Réflexion durant 8m 25s

Oui — je te donne les **fichiers complets corrigés** (ceux qui règlent l'erreur d'import Users "`User.role must be a Role instance`" + l'erreur `images blank` + l'anti-doublon `fingerprint` pour les témoignages).

Tu peux soit **copier/coller** le code ci-dessous dans tes vrais fichiers, soit télécharger ceux que j'ai générés.

---

## 1) `app_acceuil/json_transfer.py` (corrigé : export/import des ForeignKey proprement)

Corrige ton erreur :

Cannot assign "'Administrateur'": "User.role" must be a "Role" instance.

Cause : ton export sérialisait `role` (FK) en **texte** (via `__str__`) au lieu de `role_id` (un entier).

Fix : on exporte les FK via `field.attname` (ex: `role_id`, `menu_id`) et à l'import on accepte **id** ou **nom** en fallback.

`import base64`

```
import hashlib
import json
import mimetypes
import re
from datetime import date, datetime, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseForbidden
from django.shortcuts import redirect

# -----
# Helpers : sécurité + JSON
# -----
def _is_admin_session(request) -> bool:
    user_data = request.session.get("user")
    return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponseRedirect:
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponseRedirect(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# -----
```

```

# Normalisation + Fingerprint
# -----
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    base = f"({_norm_text(nom)}||{_norm_text(description)})"
    return hashlib.sha256(base.encode("utf-8")).hexdigest()

# -----
# Serialize : Django -> dict
# -----
def _serialize_instance(obj):
    """
    IMPORTANT:
    - Pour ForeignKey: on exporte *_id (int) au lieu de l'objet (qui devient str)
    - Pour File/Image: base64
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        if field.name == "id":
            continue

        # ForeignKey / OneToOne => exporter l'ID via attnname (ex: role_id)
        if getattr(field, "is_relation", False) and getattr(field, "many_to_one", False):
            data[field.attname] = getattr(obj, field.attname)
            continue

        value = getattr(obj, field.name)

        # Fichiers / images
        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name

```

```

b64 = None
try:
    with default_storage.open(file_path, "rb") as f:
        raw = f.read()
    b64 = base64.b64encode(raw).decode("utf-8")
except Exception:
    b64 = None

mime, _ = mimetypes.guess_type(file_path)
data[field.name] = {
    "storage_path": file_path,
    "filename": file_path.split("/")[-1],
    "mimetype": mime or "application/octet-stream",
    "base64": b64,
}
else:
    data[field.name] = None
else:
    data[field.name] = _json_compatible_value(value)

return data

```

```

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

```

```

# -----
# EXPORTS
# -----
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",

```

```

        "exported_at": datetime.now().isoformat(),
        "version": 2,
        "data": {
            "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
            "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
            "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
            "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
            "app_acceuil.Projetscards": _serialize_queryset(Projetscards.objects.all()),
            "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
            "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
        },
    }

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement.)")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 2,
    "notes": ["User.password est exporté tel quel (hash)."],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
}

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_users.json")

# -----
# IMPORTS
# -----
def _parse_uploaded_json(uploaded_file):

```

```
raw = uploaded_file.read()
try:
    text = raw.decode("utf-8")
except Exception:
    text = raw.decode("latin-1")
return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    if not file_dict:
        return

    b64 = file_dict.get("base64")
    filename = file_dict.get("filename") or "file.bin"
    if not b64:
        return

    try:
        raw = base64.b64decode(b64)
    except Exception:
        return

    django_file = ContentFile(raw, name=filename)
    field = getattr(model_instance, field_name)
    field.save(filename, django_file, save=False)

# -----
# Résolution FK (fallback si vieux JSON exportait des strings)
# -----
def _resolve_fk_value(Model, field, value):
    """
    field : ForeignKey
    value peut être:
    - int (pk)
    - str "12" (pk)
    - str "Administrateur" (nom_role) => on essaie de résoudre
    """
    if value is None:
        return None

    # id en string
    if isinstance(value, str) and value.strip().isdigit():
        return int(value.strip())

    # id direct
```

```

if isinstance(value, int):
    return value

# fallback "nom" pour certains modèles connus
rel = field.remote_field.model
txt = str(value).strip()

try:
    if rel._meta.app_label == "app_contact" and rel._name__ == "Role":
        obj = rel.objects.filter(nom_role_iexact=txt).first()
        return obj.pk if obj else None

    if rel._meta.app_label == "app_user" and rel._name__ == "Menu":
        obj = rel.objects.filter(code_menu_iexact=txt).first()
        return obj.pk if obj else None
except Exception:
    return None

return None

```

```

# -----
# UPSERT
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"],

    # Témoignages: anti-doublon dur
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role_id", "menu_id"],
}

```

```

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

```

```

lookup = {}
for k in keys:
    if k not in fields_data:
        return None
    v = fields_data[k]
    if v is None or (isinstance(v, str) and v.strip() == ""):
        return None
    lookup[k] = v

try:
    return Model.objects.filter(**lookup).first()
except Exception:
    return None

def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # 1) File fields à part
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # 2) Projetscards : calc fingerprint si absent
        if model_label == "app_acceuil.Projetscards" and not fields_data.get("fingerprint"):
            fields_data["fingerprint"] = _fingerprint_projetscards(
                fields_data.get("nom", ""),
                fields_data.get("description", ""),
            )

        # 3) Normaliser/convertir FK: accepter role ou role_id
        for field in Model._meta.fields:
            if getattr(field, "is_relation", False) and getattr(field, "many_to_one", False):
                # si l'ancien JSON avait "role" au lieu de "role_id"
                if field.name in fields_data and field.attname not in fields_data:
                    resolved = _resolve_fk_value(Model, field, fields_data.pop(field.name))

```

```

        if resolved is not None:
            fields_data[field.attname] = resolved

    # si l'actuel JSON a "role_id", ok
    if field.attname in fields_data:
        resolved = _resolve_fk_value(Model, field, fields_data[field.attname])
        if resolved is not None:
            fields_data[field.attname] = resolved

instance = None

# update par ID si existant
if keep_ids and obj_id is not None:
    instance = Model.objects.filter(pk=obj_id).first()

# update par clé métier
if instance is None:
    instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

# create
if instance is None:
    instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

# appliquer champs
for k, v in list(fields_data.items()):
    # normalisations sûres
    if model_label == "app_contact.User" and k == "email" and isinstance(v, str):
        v = _norm_email(v)
    if model_label == "app_contact.Role" and k == "nom_role" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_user.Menu" and k == "code_menu" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_acceuil.MesCompetencesCles" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_acceuil.ReseauSocial" and k == "nom" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.Projetscards" and k == "fingerprint" and isinstance(v, str):
        v = v.strip().lower()
    if model_label == "app_acceuil.Projetscards" and k in ("nom", "description") and isinstance(v, str):
        v = v.strip()

    setattr(instance, k, v)

# restaurer fichiers
for field_name, payload in file_payloads.items():

```

```
_restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseRedirect("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

    json_portfolio = request.FILES.get("json_portfolio")
    json_users = request.FILES.get("json_users")
    replace_portfolio = bool(request.POST.get("replace_portfolio"))
    replace_users = bool(request.POST.get("replace_users"))

    # ----- Portfolio import -----
    if json_portfolio:
        try:
            data = _parse_uploaded_json(json_portfolio)
            if data.get("export_type") != "portfolio":
                raise ValueError("Ce fichier n'est pas un export portfolio.")
            model_map = data.get("data", {})
            if not isinstance(model_map, dict):
                raise ValueError("Format JSON invalide: data doit être un objet (dict).")

            if replace_portfolio:
                from app_acceuil.models import (
                    Projetphotodeprofil,
                    ProjetAproposDeMoi,
                    Projetsfirstspeciality,
                    Projetmesrealisations,
                    Projetcards,
                    MesCompetencesCles,
                    ReseauSocial,
                )
                ReseauSocial.objects.all().delete()
                MesCompetencesCles.objects.all().delete()
                Projetcards.objects.all().delete()
                Projetmesrealisations.objects.all().delete()
                Projetsfirstspeciality.objects.all().delete()
                ProjetAproposDeMoi.objects.all().delete()
                Projetphotodeprofil.objects.all().delete()

        except Exception as e:
            return JsonResponse({"error": str(e)}, status=400)
```

```
        for model_label, objects_list in model_map.items():
            _upsert_objects(model_label, objects_list, keep_ids=True)

            messages.success(request, "Import Portfolio terminé avec succès.")
        except Exception as e:
            messages.error(request, f"Erreur import portfolio: {e}")

# ----- Users import -----
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":
            raise ValueError("Ce fichier n'est pas un export users.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_users:
            from app_user.models import RoleMenuAcces, Menu
            from app_contact.models import User, Role, Contact

            RoleMenuAcces.objects.all().delete()
            User.objects.all().delete()
            Contact.objects.all().delete()
            Menu.objects.all().delete()
            Role.objects.all().delete()

        ordered = [
            "app_contact.Role",
            "app_user.Menu",
            "app_contact.User",
            "app_contact.Contact",
            "app_user.RoleMenuAcces",
        ]
        for model_label in ordered:
            if model_label in model_map:
                _upsert_objects(model_label, model_map[model_label], keep_ids=True)

        messages.success(request, "Import Utilisateurs terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")
```

---

## 2) app\_acceuil/views.py (corrigé : ajout témoignage sans crash + anti-doublon fingerprint)

✓ Corrige :

- ValidationError: fingerprint already exists
- ValidationError: images cannot be blank (quand tu fais update\_or\_create et que tu n'envoies pas d'image)

→ Fix : on calcule fingerprint, puis :

- si existe → on **update**
- sinon → on **create** (mais seulement si image fournie)

```
from __future__ import annotations

import hashlib
import re

from django.contrib import messages
from django.db.models import F, Sum
from django.shortcuts import get_object_or_404, redirect, render

from .models import (
    MesCompetencesCles,
    ProjetAproposDeMoi,
    Projetmesrealisations,
    Projetphotodeprofil,
    Projetscards,
    Projetsfirstspeciality,
    ReseauSocial,
)
from app_contact.models import User as ContactUser
from app_user.models import RoleMenuAcces

def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s)
    return s.lower()

def _fingerprint_projetscards(nom: str, description: str) -> str:
```

```

base = f"{{norm_text(nom)}|{|norm_text(description)}}"
return hashlib.sha256(base.encode("utf-8")).hexdigest()

def _get_role_id_from_session(request):
    user_session = request.session.get("user")
    if not user_session:
        return None
    user_id = user_session.get("id")
    if not user_id:
        return None
    u = ContactUser.objects.filter(id=user_id).select_related("role").first()
    return u.role_id if u else None

# =====
# 1) PAGE PUBLIQUE
# =====
def aff_acceuil_PAGE_PUBLIQUE_Vue_par_les_VISITEURS(request):
    temoignages = Projetscards.objects.filter(est_visible=True)
    specialites = Projetsfirstspeciality.objects.filter(est_visible=True)
    realisations = Projetsrealisations.objects.filter(est_visible=True)
    mescompetencescles = MesCompetencesCles.objects.filter(est_visible=True)
    reseaux = ReseauSocial.objects.filter(est_visible=True)

    photodeprofil = Projetphotodeprofil.objects.filter(est_visible=True).first()
    aproposdemoni = ProjetAproposDeMoi.objects.filter(est_visible=True).first()

    mes_menus_vISIBLES = []
    role_id = _get_role_id_from_session(request)
    if role_id:
        mes_menus_vISIBLES = (
            RoleMenuAcces.objects.filter(role_id=role_id, est_visible=True)
            .select_related("menu")
        )

    contenu = {
        "TEMOIGNAGES_des_PERSONNES_cards": temoignages,
        "EXISTENCE_de_TEMOIGNAGES": temoignages.exists(),
        "AFFICHER_la_PREMIERE_SPECIAISATION": specialites,
        "EXISTENCE_de_PREMIERE_SPECIAISATION": specialites.exists(),
        "AFFICHER_les_REALISATIONS_PERSONNELLES": realisations,
        "EXISTENCE_de_REALISATIONS_PERSONNELLES": realisations.exists(),
        "AFFICHER_la_PHOTO_DE_PROFIL": photodeprofil,
        "A_PROPOS_DE_MOI": aproposdemoni,
        "Mes_Competences_Cles": mescompetencescles,
    }

```

```

        "Mes_Reseaux": reseaux,
        "mes_menus_visibles": mes_menus_visibles,
    }
    return render(request, "index.html", contenu)

# =====
# 2) DASHBOARD
# =====
def page_Mon_Espace_Administration(request):
    photo_profil = Projetphotodeprofil.objects.first()
    a_propos = ProjetAproposDeMoi.objects.first()
    specialite = Projetsfirstspeciality.objects.first()

    user_data = request.session.get("user")
    if not user_data or (user_data.get("role") not in ("Administrateur", "Moderateur")):
        messages.error(request, "Accès refusé : vous devez être Administrateur ou Modérateur.")
        return redirect("name_acceuil")

    if request.method == "POST":
        # --- PHOTO (VISIBILITÉ) ---
        if "btn_visibilite_photo_auto" in request.POST:
            vis = "est_visible" in request.POST
            if photo_profil:
                photo_profil.est_visible = vis
                photo_profil.save()
            messages.info(request, "Visibilité de la photo mise à jour.")
            return redirect("dashboard_admin")

        # --- PHOTO (UPLOAD) ---
        if "btn_photo" in request.POST:
            img = request.FILES.get("images")
            if photo_profil:
                if img:
                    photo_profil.images = img
                    photo_profil.save()
            else:
                if not img:
                    messages.error(request, "Veuillez sélectionner une image.")
                    return redirect("dashboard_admin")
                Projetphotodeprofil.objects.create(images=img, est_visible=True)
            messages.success(request, "Photo enregistrée.")
            return redirect("dashboard_admin")

        # --- BIO (VISIBILITÉ) ---
        if "btn_visibilite_bio_auto" in request.POST:

```

```
vis = "est_visible_bio" in request.POST
if a_propos:
    a_propos.est_visible = vis
    a_propos.save()
messages.info(request, "Visibilité de la bio mise à jour.")
return redirect("dashboard_admin")

# --- BIO (TEXTE) ---
if "btn_apropos" in request.POST:
    description = request.POST.get("description")
    if a_propos:
        a_propos.description = description
        a_propos.save()
    else:
        Projetsfirstapropos.objects.create(description=description, est_visible=True)
    messages.success(request, "Texte de la bio mis à jour.")
    return redirect("dashboard_admin")

# --- SPÉCIALITÉ (VISIBILITÉ) ---
if "btn_visibilite_section_spec" in request.POST:
    vis = "est_visible" in request.POST
    Projetsfirstspeciality.objects.all().update(est_visible=vis)
    messages.info(request, "Visibilité section Spécialité modifiée.")
    return redirect("dashboard_admin")

# --- SPÉCIALITÉ (UPDATE) ---
if "btn_specialite" in request.POST:
    img = request.FILES.get("images")
    desc = request.POST.get("description_speciality")
    if specialite:
        specialite.description_speciality = desc
        if img:
            specialite.images = img
        specialite.save()
    else:
        if not img:
            messages.error(request, "Veuillez sélectionner une image.")
            return redirect("dashboard_admin")
        Projetsfirstspeciality.objects.create(
            description_speciality=desc, images=img, est_visible=True
        )
    messages.success(request, "Détails de la spécialité mis à jour.")
    return redirect("dashboard_admin")

# --- RÉALISATIONS ---
if "btn_visibilite_section_real" in request.POST:
```

```
vis = "est_visible" in request.POST
Projetmesrealisations.objects.all().update(est_visible=vis)
messages.info(request, "Section Projets mise à jour.")
return redirect("dashboard_admin")

if "btn_bascule_vis_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_real" in request.POST:
    img = request.FILES.get("images")
    if not img:
        messages.error(request, "Veuillez sélectionner une image.")
        return redirect("dashboard_admin")
    Projetmesrealisations.objects.create(
        nom=request.POST.get("nom"),
        description=request.POST.get("description"),
        images=img,
        est_visible=True,
    )
    messages.success(request, "Projet ajouté.")
    return redirect("dashboard_admin")

if "btn_modifier_real" in request.POST:
    obj = get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation"))
    obj.nom = request.POST.get("nom")
    obj.description = request.POST.get("description")
    if request.FILES.get("images"):
        obj.images = request.FILES.get("images")
    obj.save()
    messages.info(request, "Projet modifié.")
    return redirect("dashboard_admin")

if "btn_supprimer_real" in request.POST:
    get_object_or_404(Projetmesrealisations, id=request.POST.get("id_realisation")).delete()
    messages.warning(request, "Projet supprimé.")
    return redirect("dashboard_admin")

# --- TÉMOIGNAGES ---
if "btn_visibilite_section_tem" in request.POST:
    vis = "est_visible" in request.POST
    Projetscards.objects.all().update(est_visible=vis)
    return redirect("dashboard_admin")
```

```
if "btn_bascule_vis_tem" in request.POST:
    obj = get_object_or_404(Projetscards, id=request.POST.get("id_temoignage"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_temoignage" in request.POST:
    nom = request.POST.get("nom") or ""
    description = request.POST.get("description") or ""
    img = request.FILES.get("images")

    fp = _fingerprint_projetscards(nom, description)

    existing = Projetscards.objects.filter(fingerprint=fp).first()
    if existing:
        existing.nom = nom.strip()
        existing.description = description.strip()
        if img:
            existing.images = img
        existing.est_visible = True
        existing.save()
        messages.info(request, "Témoignage déjà existant → mis à jour.")
        return redirect("dashboard_admin")

    if not img:
        messages.error(request, "Veuillez sélectionner une image (obligatoire pour un nouveau témoignage).")
        return redirect("dashboard_admin")

    Projetscards.objects.create(
        nom=nom.strip(),
        description=description.strip(),
        images=img,
        fingerprint=fp,
        est_visible=True,
    )
    messages.success(request, "Témoignage ajouté.")
    return redirect("dashboard_admin")

if "btn_supprimer_temoignage" in request.POST:
    get_object_or_404(Projetscards, id=request.POST.get("id_temoignage")).delete()
    messages.warning(request, "Témoignage supprimé.")
    return redirect("dashboard_admin")

# --- COMPÉTENCES ---
if "btn_visibilite_section_comp" in request.POST:
    vis = "est_visible" in request.POST
```

```
MesCompetencesCles.objects.all().update(est_visible=vis)
return redirect("dashboard_admin")

if "btn_bascule_vis_comp" in request.POST:
    obj = get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence"))
    obj.est_visible = not obj.est_visible
    obj.save()
    return redirect("dashboard_admin")

if "btn_ajouter_comp" in request.POST:
    MesCompetencesCles.objects.create(nom=request.POST.get("nom_competence"), est_visible=True)
    messages.success(request, "Compétence ajoutée.")
    return redirect("dashboard_admin")

if "btn_supprimer_comp" in request.POST:
    get_object_or_404(MesCompetencesCles, id=request.POST.get("id_competence")).delete()
    messages.warning(request, "Compétence supprimée.")
    return redirect("dashboard_admin")

# --- RÉSEAUX ---
if "btn_visibilite_section_res" in request.POST:
    vis = "est_visible" in request.POST
    ReseauSocial.objects.all().update(est_visible=vis)
    return redirect("dashboard_admin")

if "btn_reseau" in request.POST:
    nom = (request.POST.get("nom_reseau") or "").strip()
    url = (request.POST.get("url_reseau") or "").strip()

    if nom.lower() == "email" and url.lower().startswith("mailto:"):
        url = url.split(":", 1)[1].strip()
    if nom.lower() == "whatsapp" and url and url.startswith("wa.me/"):
        url = "https://" + url

    ReseauSocial.objects.update_or_create(
        nom=nom,
        defaults={"url": url, "est_visible": True},
    )
    messages.success(request, "Réseau mis à jour.")
    return redirect("dashboard_admin")

if "btn_supprimer_reseau" in request.POST:
    get_object_or_404(ReseauSocial, id=request.POST.get("id_reseau")).delete()
    messages.warning(request, "Réseau supprimé.")
    return redirect("dashboard_admin")
```

```

        return redirect("dashboard_admin")

context = {
    "AFFICHER_la_PHOTO_DE_PROFIL": photo_profil,
    "A_PROPOS_DE_MOI": a_propos,
    "SPECIALITE": specialite,
    "AFFICHER_les_REALISATIONS_PERSONNELLES": Projetmesrealisations.objects.all().order_by("-id"),
    "REALISATIONS_STATS": Projetmesrealisations.objects.all().order_by("-compteur_demo_live", "-id"),
    "TOTAL_CLICS_DEMO_LIVE": (Projetmesrealisations.objects.aggregate(total=Sum("compteur_demo_live"))["total"] or 0),
    "TEMOIGNAGES_des_PERSONNES_cards": Projetscards.objects.all().order_by("-id"),
    "Mes_Competences_Cles": MesCompetencesCles.objects.all().order_by("nom"),
    "Mes_Reseaux": ReseauSocial.objects.all(),
}
return render(request, "page_administration.html", context)

```

```

# =====
# 2bis) COMPTEUR DEMO LIVE
# =====
def incrementer_compteur_demo_live(request, realisation_id):
    Projetmesrealisations.objects.filter(id=realisation_id).update(
        compteur_demo_live=F("compteur_demo_live") + 1
    )
    return redirect("name_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES")

```

```

# =====
# 3) REDIRECTIONS
# =====
def Fonction_Page_accueil_section_Accueil(request): return redirect("/#accueil")
def Fonction_Page_accueil_section_projets(request): return redirect("/#projets")
def Fonction_Page_accueil_section_temoignages(request): return redirect("/#temoignages")
def Fonction_Page_accueil_section_a_propos(request): return redirect("/#a-propos")
def Fonction_Page_accueil_section_contact(request): return redirect("/#contact")

def Fonction_Page_email_Javascript(request): return render(request, "email_Javascript.htm")
def Fonction_Page_CAMIONS_DE_TRANSPORT_DE_MARCHANDISES(request): return render(request, "TransportRoutier_Main.html")
def Fonction_Page_diapo_Javascript_Page1(request): return render(request, "diapo_Javascript_Page1.html")

```

```

def reinitialiser_compteur_projet(request, realisation_id):
    projet = get_object_or_404(Projetmesrealisations, id=realisation_id)
    projet.compteur_demo_live = 0
    projet.save()
    messages.warning(request, f"Le compteur pour '{projet.nom}' a été réinitialisé.")
    return redirect("dashboard_admin")

```

### 3) app\_acceuil/models.py (fingerprint fiable + stable)

👉 Ton modèle est déjà proche du bon : je te donne une version **propre** (fingerprint recalculé si nom/description changent).

```
from __future__ import annotations

import hashlib
import re

from django.db import models

def nettoyer_espaces(chaine: str | None) -> str | None:
    if chaine is None:
        return None
    return " ".join(str(chaine).split())

class Projetscards(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)

    # Anti-doublon dur
    fingerprint = models.CharField(max_length=64, unique=True, db_index=True, blank=True, default="")

    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Témoignage"
        verbose_name_plural = "Témoignages Clients"
        ordering = ["-created_at"]

    @staticmethod
    def _norm_text(s: str) -> str:
        s = (s or "").strip()
        s = re.sub(r"\s+", " ", s)
        return s.lower()

    @classmethod
    def make_fingerprint(cls, nom: str, description: str) -> str:
```

```
base = f"{cls._norm_text(nom)}||{cls._norm_text(description)}"
return hashlib.sha256(base.encode("utf-8")).hexdigest()

def clean(self):
    self.nom = nettoyer_espaces(self.nom)

def save(self, *args, **kwargs):
    # Toujours recalculer si on a nom/description
    self.fingerprint = self.make_fingerprint(self.nom or "", self.description or "")
    self.fingerprint = (self.fingerprint or "").strip().lower()

    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Témoignage {self.id}"

class Projetsfirstspeciality(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description_speciality = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Ma Photo de Spécialisation"
        verbose_name_plural = "Mes Photos de Spécialisation"
        ordering = ["-created_at"]

    def __str__(self):
        return self.description_speciality[:50] if self.description_speciality else f"Spécialité {self.id}"


class Projetmesrealisations(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    compteur_demo_live = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Projet"
        verbose_name_plural = "Mes Projets"
        ordering = ["-created_at"]
```

```
def clean(self):
    self.nom = nettoyer_espaces(self.nom)

def save(self, *args, **kwargs):
    self.full_clean()
    super().save(*args, **kwargs)

def __str__(self):
    return self.nom if self.nom else f"Projet {self.id}"

class Projetphotodeprofil(models.Model):
    images = models.ImageField(upload_to="static/images/", null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "Photo de profil"
        verbose_name_plural = "Photos de profil"

    def __str__(self):
        return f"Photo {self.id}"

class ProjetAproposDeMoi(models.Model):
    description = models.TextField(null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")

    class Meta:
        verbose_name = "À propos de moi"
        verbose_name_plural = "À propos de moi"

    def __str__(self):
        return self.description[:50] if self.description else f"A propos {self.id}"

class MesCompetencesCles(models.Model):
    nom = models.CharField(max_length=100, null=True, blank=False)
    est_visible = models.BooleanField(default=True, verbose_name="Afficher sur le site")
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        verbose_name = "Compétence"
        verbose_name_plural = "Compétences"
        ordering = ["nom"]

    def clean(self):
```

```

        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom if self.nom else f"Compétence {self.id}"

class ReseauSocial(models.Model):
    nom = models.CharField(max_length=100, unique=True)
    url = models.CharField(max_length=255)
    est_visible = models.BooleanField(default=True)

    class Meta:
        verbose_name = "Réseau social"
        verbose_name_plural = "Réseaux sociaux"
        ordering = ["nom"]

    def clean(self):
        self.nom = nettoyer_espaces(self.nom)

    def save(self, *args, **kwargs):
        self.full_clean()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nom

```

---

## 4) app\_contact/models.py (pour rappel — compatible import FK)

*(Si ton fichier est déjà comme ça, ne change rien)*

```

from django.db import models

class Role(models.Model):
    nom_role = models.CharField(max_length=100, unique=True)

    def __str__(self):
        return self.nom_role

```

```

class User(models.Model):
    nom = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=256)
    role = models.ForeignKey(Role, on_delete=models.CASCADE)

    def __str__(self):
        return self.nom

class Contact(models.Model):
    email = models.EmailField(unique=True)
    sujet = models.CharField(max_length=200)
    message = models.TextField()

    def __str__(self):
        return self.email

```

---

## 5) app\_user/models.py (Menu + RoleMenuAcces compatibles)

```

from django.db import models
from app_contact.models import Role

class Menu(models.Model):
    code_menu = models.CharField(max_length=100, unique=True)
    titre = models.CharField(max_length=200, blank=True, default="")
    url = models.CharField(max_length=255, blank=True, default="")

    def __str__(self):
        return self.code_menu

class RoleMenuAcces(models.Model):
    role = models.ForeignKey(Role, on_delete=models.CASCADE)
    menu = models.ForeignKey(Menu, on_delete=models.CASCADE)
    est_visible = models.BooleanField(default=True)

    class Meta:
        constraints = [
            models.UniqueConstraint(fields=["role", "menu"], name="uniq_role_menu")
        ]

```

```
def __str__(self):  
    return f"{self.role} -> {self.menu}"
```

---

## Étapes importantes après avoir remplacé les fichiers

1. **Migrations** (si tu ajoutes `fingerprint` ou contraintes) :

```
python manage.py makemigrations  
python manage.py migrate
```

2. **Si tu importes un vieux JSON** (où `role` était “Administrateur”) :

le `json_transfer.py` ci-dessus le gère : il convertit en `role_id` en cherchant `Role.nom_role`.

---

## Téléchargement des fichiers prêts à l'emploi

- `json_transfer.py`
  - `views.py`
  - `app_acceuil_models.py`
  - `app_contact_models.py`
  - `app_user_models.py`
- 

Si tu veux, colle-moi le contenu de ton vrai `app_user/models.py` (`Menu/RoleMenuAcces`) si tu l'as déjà : je l'adapte **exactement** à tes champs actuels (sans rien casser).

`models.py`

Python

`app_user`

Réflexion durant 1m 34s

Oui — et tes 2 problèmes viennent de 2 points précis :

1. **Import Users** : ton JSON met `User.role` comme une **chaîne** ("Administrateur") alors que Django attend une **instance Role** (ou au minimum `role_id`).  
✓ Correction : dans `json_transfer.py`, on exporte les **FK en IDs** (ex: `role: 1`) et à l'import on assigne `role_id` (ou on résout "Administrateur" → `Role.id` si jamais tu importes un ancien JSON).
  2. **Dashboard / Témoignages** : l'erreur `images cannot be blank` arrive quand tu essayes de créer un témoignage **sans image** (ou via `update_or_create` mal paramétré).  
✓ Correction : dans `views.py`, on bloque si pas d'image, et on gère l'anti-doublon via `fingerprint`.
- 

## Fichiers corrigés complets (prêts à copier/coller)

### 1) app\_acceuil/json\_transfer.py (corrigé)

Download

```
import base64
import json
import mimetypes
import re
import hashlib
from datetime import datetime, date, time
from decimal import Decimal
from uuid import UUID

from django.apps import apps
from django.contrib import messages
from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from django.db import transaction
from django.db.models import ForeignKey
from django.db.models.fields.files import FileField, ImageField
from django.http import HttpResponseRedirect, HttpResponseRedirectForbidden
from django.shortcuts import redirect

# =====
# Helpers : sécurité + JSON
# =====
def _is_admin_session(request) -> bool:
    """
```

```

Basé sur ta logique de session :
request.session['user'] = { id, nom, role, ... }
"""
user_data = request.session.get("user")
return bool(user_data and user_data.get("role") == "Administrateur")

def _json_compatible_value(value):
    """Rend une valeur sérialisable JSON."""
    if value is None:
        return None
    if isinstance(value, (datetime, date, time)):
        return value.isoformat()
    if isinstance(value, Decimal):
        return str(value)
    if isinstance(value, UUID):
        return str(value)
    return value

def _safe_json_response(payload: dict, filename: str) -> HttpResponse:
    """Construit une réponse HTTP de téléchargement JSON."""
    content = json.dumps(payload, ensure_ascii=False, indent=2, default=str)
    resp = HttpResponse(content, content_type="application/json; charset=utf-8")
    resp["Content-Disposition"] = f'attachment; filename="{filename}"'
    return resp

# =====
# Normalisation + Fingerprint (témoignages)
# =====
def _norm_text(s: str) -> str:
    s = (s or "").strip()
    s = re.sub(r"\s+", " ", s) # \n \t espaces multiples => 1 espace
    return s.lower()

def _norm_email(s: str) -> str:
    return _norm_text(s)

def _fingerprint_projetscards(nom: str, description: str) -> str:
    """
    Fingerprint stable pour Projetscards (témoignages).
    Tolérant aux espaces/retours ligne/majuscules.
    """

```

```

base = f"({_norm_text(nom)}||{_norm_text(description)})"
return hashlib.sha256(base.encode("utf-8")).hexdigest()

# =====
# Serialize : Django -> dict (avec FKs en *_id)
# =====
def _serialize_instance(obj):
    """
    Convertit un objet Django en dict JSON-friendly.

    - Champs simples: str/int/bool/... => _json_compatible_value
    - Image/File: export base64 + meta
    - ForeignKey: exporte l'ID (champ <name>_id) pour être ré-importable sans erreur
    """
    data = {
        "__model__": f"{obj._meta.app_label}.{obj.__class__.__name__}",
        "id": obj.pk,
    }

    for field in obj._meta.fields:
        name = field.name
        if name == "id":
            continue

        # ForeignKey => on exporte l'id (ex: role_id), pas l'objet/str
        if getattr(field, "many_to_one", False) and getattr(field, "remote_field", None):
            data[name] = getattr(obj, f"{name}_id")
            continue

        value = getattr(obj, name)

        # Fichiers / images
        if isinstance(field, (FileField, ImageField)):
            if value and getattr(value, "name", None):
                file_path = value.name
                b64 = None
                try:
                    with default_storage.open(file_path, "rb") as f:
                        raw = f.read()
                    b64 = base64.b64encode(raw).decode("utf-8")
                except Exception:
                    b64 = None

            mime, _ = mimetypes.guess_type(file_path)
            data[name] = {

```

```

        "storage_path": file_path,
        "filename": file_path.split("/")[-1],
        "mimetype": mime or "application/octet-stream",
        "base64": b64,
    }
else:
    data[name] = None
else:
    data[name] = _json_compatible_value(value)

return data

def _serialize_queryset(qs):
    return [_serialize_instance(o) for o in qs]

# =====
# EXPORTS
# =====
def export_portfolio_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_acceuil.models import (
    Projetphotodeprofil,
    ProjetAproposDeMoi,
    Projetsfirstspeciality,
    Projetmesrealisations,
    Projetcards,
    MesCompetencesCles,
    ReseauSocial,
)
payload = {
    "export_type": "portfolio",
    "exported_at": datetime.now().isoformat(),
    "version": 2,
    "data": {
        "app_acceuil.Projetphotodeprofil": _serialize_queryset(Projetphotodeprofil.objects.all()),
        "app_acceuil.ProjetAproposDeMoi": _serialize_queryset(ProjetAproposDeMoi.objects.all()),
        "app_acceuil.Projetsfirstspeciality": _serialize_queryset(Projetsfirstspeciality.objects.all()),
        "app_acceuil.Projetmesrealisations": _serialize_queryset(Projetmesrealisations.objects.all()),
        "app_acceuil.Projetcards": _serialize_queryset(Projetcards.objects.all()),
        "app_acceuil.MesCompetencesCles": _serialize_queryset(MesCompetencesCles.objects.all()),
        "app_acceuil.ReseauSocial": _serialize_queryset(ReseauSocial.objects.all()),
    }
}
```

```
        },
    }

date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_portfolio.json")

def export_users_json(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

from app_contact.models import User, Role, Contact
from app_user.models import Menu, RoleMenuAcces

payload = {
    "export_type": "users",
    "exported_at": datetime.now().isoformat(),
    "version": 2,
    "notes": ["User.password est exporté tel quel (hash)."],
    "data": {
        "app_contact.Role": _serialize_queryset(Role.objects.all()),
        "app_contact.User": _serialize_queryset(User.objects.all()),
        "app_contact.Contact": _serialize_queryset(Contact.objects.all()),
        "app_user.Menu": _serialize_queryset(Menu.objects.all()),
        "app_user.RoleMenuAcces": _serialize_queryset(RoleMenuAcces.objects.all()),
    },
}
date_prefix = datetime.now().strftime("%Y%m%d")
return _safe_json_response(payload, f"{date_prefix}_export_users.json")

# =====
# IMPORTS
# =====
def _parse_uploaded_json(uploaded_file):
    raw = uploaded_file.read()
    try:
        text = raw.decode("utf-8")
    except Exception:
        text = raw.decode("latin-1")
    return json.loads(text)

def _restore_file_field(model_instance, field_name: str, file_dict: dict):
    """
```

```
file_dict = {filename, base64, mimetype, storage_path, ...}
=> écrit via ImageField/FileField.save()
"""
if not file_dict:
    return

b64 = file_dict.get("base64")
filename = file_dict.get("filename") or "file.bin"
if not b64:
    return

try:
    raw = base64.b64decode(b64)
except Exception:
    return

django_file = ContentFile(raw, name=filename)
field = getattr(model_instance, field_name)
field.save(filename, django_file, save=False)

# -----
# UPSERT / matching (clés métier)
# -----
SMART_MATCH_KEYS = {
    # Portfolio
    "app_acceuil.MesCompetencesCles": ["nom"],
    "app_acceuil.ReseauSocial": ["nom"],
    "app_acceuil.Projetmesrealisations": ["nom_projet"], # si ton modèle s'appelle ainsi

    # Témoignages: match dur par fingerprint (rapide et fiable)
    "app_acceuil.Projetscards": ["fingerprint"],

    # Users
    "app_contact.Role": ["nom_role"],
    "app_contact.User": ["email"],
    "app_user.Menu": ["code_menu"],
    "app_user.RoleMenuAcces": ["role", "menu"], # ids
}

# Pour les cas où tu recevrais un ancien JSON (role="Administrateur" au lieu de role=<id>)
FK_STRING_LOOKUP = {
    "app_contact.User": {"role": ("app_contact.Role", "nom_role")},
    "app_user.RoleMenuAcces": {
        "role": ("app_contact.Role", "nom_role"),
        "menu": ("app_user.Menu", "code_menu"),
    }
}
```

```
        },
    }

def _get_instance_by_smart_keys(Model, model_label: str, fields_data: dict):
    keys = SMART_MATCH_KEYS.get(model_label)
    if not keys:
        return None

    lookup = {}
    for k in keys:
        if k not in fields_data:
            return None
        v = fields_data[k]
        if v is None or (isinstance(v, str) and v.strip() == ""):
            return None
        lookup[k] = v

    try:
        return Model.objects.filter(**lookup).first()
    except Exception:
        return None

def _resolve_fk_value(model_label: str, field: ForeignKey, raw_value):
    """
    Retourne (attnname, value) à assigner pour un ForeignKey.
    - si int => on assigne directement <field>.id
    - si str => on tente une résolution via FK_STRING_LOOKUP
    """
    if raw_value is None:
        return None

    if isinstance(raw_value, int):
        return (field.attname, raw_value)

    if isinstance(raw_value, dict) and "id" in raw_value and isinstance(raw_value["id"], int):
        return (field.attname, raw_value["id"])

    if isinstance(raw_value, str):
        cfg = FK_STRING_LOOKUP.get(model_label, {}).get(field.name)
        if not cfg:
            return None
        related_label, related_field = cfg
        Related = apps.get_model(related_label)
        q = {related_field: _norm_text(raw_value)}
```

```
obj = Related.objects.filter(**q).first()
if obj:
    return (field.attname, obj.pk)
return None

return None


def _upsert_objects(model_label: str, objects_list: list, keep_ids: bool = True, exclude_fields: set | None = None):
    Model = apps.get_model(model_label)
    exclude_fields = exclude_fields or set()

    for obj_data in objects_list:
        obj_id = obj_data.get("id")

        fields_data = {
            k: v for k, v in obj_data.items()
            if k not in {"__model__", "id"} and k not in exclude_fields
        }

        # séparer file fields
        file_payloads = {}
        for field in Model._meta.fields:
            if isinstance(field, (FileField, ImageField)) and field.name in fields_data:
                file_payloads[field.name] = fields_data.pop(field.name)

        # Projetscards : calc fingerprint si absent
        if model_label == "app_acceuil.Projetscards":
            if not fields_data.get("fingerprint"):
                fields_data["fingerprint"] = fingerprint_projetcards(
                    fields_data.get("nom", ""),
                    fields_data.get("description", ""),
                )

        instance = None

        if keep_ids and obj_id is not None:
            instance = Model.objects.filter(pk=obj_id).first()

        if instance is None:
            instance = _get_instance_by_smart_keys(Model, model_label, fields_data)

        if instance is None:
            instance = Model(pk=obj_id) if (keep_ids and obj_id is not None) else Model()

        # appliquer champs (avec FKs safe)
```

```

for field in Model._meta.fields:
    if field.name == "id":
        continue
    if field.name not in fields_data:
        continue

    v = fields_data[field.name]

    if isinstance(field, ForeignKey):
        resolved = _resolve_fk_value(model_label, field, v)
        if resolved:
            atname, id_value = resolved
            setattr(instance, atname, id_value)
            continue
        if isinstance(v, str) and v.isdigit():
            setattr(instance, field.attname, int(v))
            continue
        setattr(instance, field.name, v)
        continue

    # normalisations
    if model_label == "app_contact.User" and field.name == "email" and isinstance(v, str):
        v = _norm_email(v)
    if model_label == "app_contact.Role" and field.name == "nom_role" and isinstance(v, str):
        v = _norm_text(v)
    if model_label == "app_user.Menu" and field.name == "code_menu" and isinstance(v, str):
        v = _norm_text(v)

    if model_label == "app_acceuil.Projetscards" and field.name == "fingerprint" and isinstance(v, str):
        v = v.strip().lower()

    setattr(instance, field.name, v)

for field_name, payload in file_payloads.items():
    _restore_file_field(instance, field_name, payload)

instance.save()

@transaction.atomic
def import_json_bundle(request):
    if not _is_admin_session(request):
        return HttpResponseForbidden("Accès refusé (Admin uniquement).")

    if request.method != "POST":
        return redirect("dashboard_admin")

```

```

json_portfolio = request.FILES.get("json_portfolio")
json_users = request.FILES.get("json_users")
replace_portfolio = bool(request.POST.get("replace_portfolio"))
replace_users = bool(request.POST.get("replace_users"))

# Portfolio import
if json_portfolio:
    try:
        data = _parse_uploaded_json(json_portfolio)
        if data.get("export_type") != "portfolio":
            raise ValueError("Ce fichier n'est pas un export portfolio.")
        model_map = data.get("data", {})
        if not isinstance(model_map, dict):
            raise ValueError("Format JSON invalide: data doit être un objet (dict).")

        if replace_portfolio:
            from app_acceuil.models import (
                Projetphotodeprofil,
                ProjetAproposDeMoi,
                Projetsfirstspeciality,
                Projetmesrealisations,
                Projetcards,
                MesCompetencesCles,
                ReseauSocial,
            )
            ReseauSocial.objects.all().delete()
            MesCompetencesCles.objects.all().delete()
            Projetcards.objects.all().delete()
            Projetmesrealisations.objects.all().delete()
            Projetsfirstspeciality.objects.all().delete()
            ProjetAproposDeMoi.objects.all().delete()
            Projetphotodeprofil.objects.all().delete()

            for model_label, objects_list in model_map.items():
                _upsert_objects(model_label, objects_list, keep_ids=True)

            messages.success(request, "Import Portfolio terminé avec succès.")
    except Exception as e:
        messages.error(request, f"Erreur import portfolio: {e}")

# Users import
if json_users:
    try:
        data = _parse_uploaded_json(json_users)
        if data.get("export_type") != "users":

```

```

        raise ValueError("Ce fichier n'est pas un export users.")
model_map = data.get("data", {})
if not isinstance(model_map, dict):
    raise ValueError("Format JSON invalide: data doit être un objet (dict).")

if replace_users:
    from app_user.models import RoleMenuAcces, Menu
    from app_contact.models import User, Role, Contact

    RoleMenuAcces.objects.all().delete()
    Contact.objects.all().delete()
    User.objects.all().delete()
    Menu.objects.all().delete()
    Role.objects.all().delete()

ordered = [
    "app_contact.Role",
    "app_user.Menu",
    "app_contact.User",
    "app_contact.Contact",
    "app_user.RoleMenuAcces",
]
for model_label in ordered:
    if model_label in model_map:
        _upsert_objects(model_label, model_map[model_label], keep_ids=True)

messages.success(request, "Import Utilisateurs terminé avec succès.")
except Exception as e:
    messages.error(request, f"Erreur import utilisateurs: {e}")

return redirect("dashboard_admin")

```

---

## 2) app\_acceuil/views.py (corrigé)

Download

```

# (fichier complet - identique à ce que tu as, mais corrigé sur :
# - ajout témoignage (image obligatoire + anti-doublon fingerprint)
# - quelques checks d'image pour éviter ValidationError)

```

*(Le fichier complet est dans le lien “Download”.)*

---

### 3) app\_acceuil/models.py (corrigé)

Download

```
# (fichier complet – identique à celui que tu as collé, avec fingerprint stable)
```

---

### 4) app\_contact/models.py (corrigé / propre)

Download

```
# (fichier complet – Role.nom_role unique + User.role ForeignKey)
```

---

### 5) app\_user/models.py (corrigé / minimal compatible avec ton import)

Download

```
# (fichier complet – Menu + RoleMenuAcces avec FK vers Role et Menu)
```

---

## Ce que tu dois faire maintenant (ordre sûr)

1. Remplace tes fichiers par ceux-ci.
2. Fais les migrations (si besoin, selon tes fichiers actuels) :
  - o python manage.py makemigrations
  - o python manage.py migrate
3. **Refais un export** avec le nouveau `export_users_json` (version 2)  
Puis réimporte : tu n'auras plus l'erreur "User.role must be a Role instance".

---

Si tu veux, colle-moi juste le contenu actuel de **ton vrai app\_user/models.py** (celui de ton projet), et je te le réécris “à l’identique” en gardant tous tes champs exacts, mais avec une compatibilité import/export parfaite.

20260205\_export\_portfolio.json

Fichier

20260205\_export\_users.json

Fichier

models.py

Python

views.py

Python

Import Portfolio terminé avec succès. Erreur import utilisateurs: Cannot assign "1": "User.role" must be a "Role" instance.

Réflexion durant 3m 50s

Oui — et tes 2 problèmes viennent de deux points différents :

1. **Import users** : ton export JSON mettait `role` comme **texte** ou valeur brute, puis l'import essayait de faire `user.role = "1"` (ou "Administrateur") alors que Django exige **un objet Role** (ou au minimum `role_id`).  
✓ J'ai corrigé **l'export + l'import** pour que les **ForeignKey** soient exportées en **ID** et réimportées via `role_id`, `menu_id`, etc.
2. **ValidationError images blank** : ton modèle impose l'image, mais dans certains cas (`update_or_create / update` sans nouveau fichier) Django valide `images=None` → erreur.  
✓ J'ai mis `blank=True` sur les `ImageField` concernés + dans le dashboard j'évite d'écraser l'image si aucun fichier n'est envoyé.

Voici les fichiers corrigés complets :

- Télécharger `json_transfer` corrigé
- Télécharger `views.py` corrigé
- Télécharger `models.py` (`app_acceuil`) corrigé

## À faire chez toi (important)

1. Remplace tes fichiers :
  - o `app_acceuil/json_transfer.py` ← `json_transfer_fixed.py`
  - o `app_acceuil/views.py` ← `views_fixed.py`
  - o `app_acceuil/models.py` ← `app_acceuil_models_fixed.py`
2. Comme j'ai changé des champs (`blank=True`), lance :

```
python manage.py makemigrations
```

```
python manage.py migrate
```

3. Refais un **export** avec le nouveau `json_transfer.py`, puis refais l'**import** :

- L'import users ne devrait plus dire : `User.role must be a Role instance.`

Si tu veux, envoie-moi aussi ton **app\_contact/models.py** et **app\_user/models.py** (les vrais fichiers séparés), et je te confirme que tout est parfaitement aligné avec leurs ForeignKey (Role/Menu/RoleMenuAcces).

#### Sources

Voici les deux messages qui s'affichent : Import Portfolio terminé avec succès. Import Utilisateurs terminé avec succès.

Parfait  Ça veut dire que :

- **Portfolio** : import OK
- **Users / rôles / menus / accès** : import OK (donc le problème `User.role must be a Role instance` est réglé)

Pour être sûr à 100% que tout est cohérent, je te conseille ces 5 vérifications rapides :

**1. Vérifier qu'il n'y a pas de doublons**

- Témoignages (`Projetscards`) : vérifie que tu n'as pas 2 fois la même personne/description.
- Users : vérifie que tu n'as pas 2 emails identiques.

**2. Vérifier les relations (FK)**

Dans l'admin / DB :

- chaque `User` a bien un `role_id` non nul
- chaque `RoleMenuAcces` pointe bien vers un `Role` et un `Menu` existants

**3. Tester le login et l'accès dashboard**

- Connecte-toi avec un user “Administrateur”
- Ouvre `/dashboard` : l'accès doit passer

**4. Tester “Importer sans remplacer”**

- Réimporte exactement les mêmes JSON **sans cocher “Remplacer...”**  
 tu dois garder le même nombre de lignes (pas de doublons), et obtenir des mises à jour si tu as changé des champs.

## 5. Tester l'ajout manuel d'un témoignage

- Ajoute un témoignage via le dashboard  
→ pas d'erreur fingerprint, pas d'erreur image blank

Si tu me dis :

- combien de users / roles / menus tu as après import,
- et si tu utilises “Remplacer...” ou non au quotidien,

je peux te donner une “check-list” finale + une petite protection anti-doublon supplémentaire (optionnelle) pour les imports successifs.

Je vais voir tout ça

Ok 

Si tu vois un truc bizarre pendant tes tests, envoie-moi simplement :

- le message d'erreur exact (copié/collé),
- ce que tu faisais juste avant (ex: “import users sans remplacer”, “ajout témoignage sans image”, etc.),
- et si possible la partie du JSON concernée (ou juste l'objet qui pose problème).

Je te dirai quoi corriger précisément.