

Vous : bjr

Bonjour ! Comment puis-je vous aider aujourd'hui ?

Que ce soit pour une question précise, un projet sur lequel vous travaillez ou simplement pour discuter, je suis à votre écoute.

Souhaitez-vous que je vous aide à organiser votre journée ou que nous explorions un sujet particulier ensemble ?

voici le code du fichier app_user/views.py : # from django.shortcuts import render

```
from django.shortcuts import render, redirect, get_object_or_404
```

```
from app_contact.models import (
```

```
    User
```

```
)
```

```
from .models import (
```

```
    Menu, MenuAcces
```

```
)
```

```
# Create your views here.
```

```
def Fonction_Page_gestion_des_Utilisateurs(request): return render(request, "page_utilisateur.html")
```

```
# https://gemini.google.com/app/067aeb3e6aaa6bc1
# https://gemini.google.com/app/067aeb3e6aaa6bc1
# https://gemini.google.com/app/067aeb3e6aaa6bc1
# https://gemini.google.com/app/067aeb3e6aaa6bc1
```

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from .models import User, Menu, MenuAcces
from app_contact.models import Contact # Importation croisée

def gestion_globale(request):
    # Récupération de toutes les données nécessaires
    users = User.objects.all()
    menus = Menu.objects.all()
    contacts = Contact.objects.all().order_by('-date_envoi')
```

```
# Transformation des accès en liste de chaînes "userID_menuID" pour le template
```

```
acces_existant = [  
    f'{a.user_id}_{a.menu_id}'  
    for a in MenuAcces.objects.filter(est_visible=True)  
]
```

```
return render(request, 'page_utilisateur.html', {
```

```
    'users': users,  
    'menus': menus,  
    'contacts': contacts,  
    'acces_existant': acces_existant  
})
```

```
def add_user(request):
```

```
    if request.method == "POST":
```

```
        User.objects.create(  
            nom=request.POST.get('nom'),  
            email=request.POST.get('email'),  
            password=request.POST.get('password'))
```

```
        messages.success(request, "Utilisateur créé avec succès.")
```

```
return redirect('gestion_globale')
```

```
def edit_user(request, user_id):
```

```
    if request.method == "POST":
```

```
        user = get_object_or_404(User, id=user_id)
```

```
        user.nom = request.POST.get('nom')
```

```
        user.email = request.POST.get('email')
```

```
        user.password = request.POST.get('password')
```

```
        user.save()
```

```
        messages.info(request, f"Utilisateur {user.nom} mis à jour.")
```

```
    return redirect('gestion_globale')
```

```
def delete_user(request, user_id):
```

```
    if request.method == "POST":
```

```
        get_object_or_404(User, id=user_id).delete()
```

```
        messages.error(request, "Utilisateur supprimé.")
```

```
    return redirect('gestion_globale')
```

```
def update_menus(request):
```

```
    if request.method == "POST":
```

```
        users = User.objects.all()
```

```

menus = Menu.objects.all()

for user in users:

    for menu in menus:

        key = f"access_{user.id}_{menu.id}"

        MenuAcces.objects.update_or_create(
            user=user, menu=menu,
            defaults={'est_visible': key in request.POST}
        )

        messages.success(request, "Permissions mises à jour.")

return redirect('gestion_globale')

```

C'est un bon début pour votre fichier `views.py`. Votre code structure bien les opérations **CRUD** (Création, Lecture, Mise à jour, Suppression) pour les utilisateurs et la gestion des permissions.

Cependant, j'ai relevé quelques points d'attention, notamment sur les **importations circulaires** et la **sécurité**, que nous devrions corriger pour éviter des erreurs futures.

1. Analyse et Corrections suggérées

Voici les points à améliorer :

- **Importations en double** : Vous avez plusieurs blocs d'importations identiques. Il vaut mieux les regrouper en haut du fichier.
- **Risque d'Importation Circulaire** : Vous importez `User` depuis `app_contact.models` puis depuis `.models`. Assurez-vous que le modèle `User` ne soit défini qu'à un seul endroit (généralement dans `app_user`).
- **Sécurité des mots de passe** : Actuellement, vous enregistrez les mots de passe en texte brut (`password=request.POST.get('password')`). En Django, il est fortement recommandé d'utiliser `make_password` pour les hacher.

- **Gestion des accès :** Votre fonction update_menus est logique, elle traite une matrice de cases à cocher.

2. Code optimisé (Version corrigée)

Voici une version nettoyée et plus robuste de votre fichier views.py :

Python

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.contrib.auth.hashers import make_password # Pour la sécurité
from .models import User, Menu, MenuAcces
from app_contact.models import Contact

def gestion_globale(request):
    """Vue principale regroupant utilisateurs, menus et contacts."""
    users = User.objects.all()
    menus = Menu.objects.all()
    contacts = Contact.objects.all().order_by('-date_envoi')

    # Liste des accès pour cocher les cases dans le template
    # On récupère uniquement les IDs pour optimiser la performance
    acces_existant = [
        f'{a.user_id}_{a.menu_id}'
        for a in MenuAcces.objects.filter(est_visible=True)
```

```
]
```

```
context = {
    'users': users,
    'menus': menus,
    'contacts': contacts,
    'acces_existant': acces_existant
}

return render(request, 'page_utilisateur.html', context)
```

```
def add_user(request):
    if request.method == "POST":
        User.objects.create(
            nom=request.POST.get('nom'),
            email=request.POST.get('email'),
            # Utilisation de make_password pour la sécurité
            password=make_password(request.POST.get('password'))
        )
        messages.success(request, "Utilisateur créé avec succès.")
    return redirect('gestion_globale')
```

```
def edit_user(request, user_id):  
    user = get_object_or_404(User, id=user_id)  
  
    if request.method == "POST":  
        user.nom = request.POST.get('nom')  
        user.email = request.POST.get('email')  
        raw_password = request.POST.get('password')  
  
        if raw_password: # On ne change le mot de passe que s'il est rempli  
            user.password = make_password(raw_password)  
            user.save()  
  
            messages.info(request, f"Utilisateur {user.nom} mis à jour.")  
  
    return redirect('gestion_globale')
```

```
def delete_user(request, user_id):  
    if request.method == "POST":  
        user = get_object_or_404(User, id=user_id)  
        user.delete()  
  
        messages.error(request, "Utilisateur supprimé.")  
  
    return redirect('gestion_globale')
```

```
def update_menus(request):  
    if request.method == "POST":
```

```
users = User.objects.all()

menus = Menu.objects.all()

for user in users:

    for menu in menus:

        # La clé doit correspondre au 'name' de vos checkbox dans le HTML

        key = f"access_{user.id}_{menu.id}"

        MenuAcces.objects.update_or_create(
            user=user,
            menu=menu,
            defaults={'est_visible': key in request.POST}
        )

        messages.success(request, "Permissions mises à jour.")

return redirect('gestion_globale')
```

3. Architecture du système de permissions

Voici comment vos modèles interagissent pour gérer les accès :

Prochaines étapes suggérées :

1. **Vérification du Template** : Voulez-vous que je vérifie le code de votre fichier page_utilisateur.html pour m'assurer que les noms des champs (comme access_{{user.id}}_{{menu.id}}) correspondent bien à cette vue ?
2. **URLs** : Assurez-vous que vos noms d'URLs dans urls.py correspondent exactement aux noms utilisés dans les redirect (ex: name='gestion_globale').

Souhaitez-vous que je vous aide à rédiger la partie HTML pour la matrice des permissions ?

```
code du fichier app_user/urls.py : from django.urls import path  
from .views import *  
  
urlpatterns = [  
  
    path('root_gestion_utilisateur', Fonction_Page_gestion_des_Utilisateurs , name='name_gestion_utilisateur'),  
  
    # https://gemini.google.com/app/067aeb3e6aaa6bc1  
    # https://gemini.google.com/app/067aeb3e6aaa6bc1  
  
    # path('gestion-globale/', gestion_globale, name='gestion_globale'),  
    # path('user/add/', add_user, name='add_user'),  
    # path('user/edit/<int:user_id>', edit_user, name='edit_user'),  
    # path('user/delete/<int:user_id>', delete_user, name='delete_user'),  
    # path('user/update-menus/', update_menus, name='update_menus'),
```

```
path('root_gestion_utilisateur/', gestion_globale, name='gestion_globale'),
path('user/add/', add_user, name='add_user'),
path('user/edit/<int:user_id>/', edit_user, name='edit_user'),
path('user/delete/<int:user_id>/', delete_user, name='delete_user'),
path('user/update-menus/', update_menus, name='update_menus'),\n\n# path('gestion-utilisateurs/', views.gestion_utilisateurs, name='gestion_utilisateurs'),
#   path('add-user/', views.add_user, name='add_user'),
#   path('edit-user/<int:user_id>/', views.edit_user, name='edit_user'),
#   path('delete-user/<int:user_id>/', views.delete_user, name='delete_user'),
#   path('update-menus/', views.update_menus, name='update_menus'),
]
```

C'est parfait, vos URLs sont maintenant alignées avec les fonctions de votre fichier `views.py`.

Cependant, il y a un petit conflit potentiel : vous avez deux URLs qui pointent vers la même racine (`root_gestion_utilisateur`). Je vous conseille de nettoyer le fichier pour ne garder que les routes actives et éviter toute confusion pour Django.

Voici une version propre et optimisée de votre fichier `app_user/urls.py` :

Python

```
from django.urls import path\n\nfrom . import views # Utiliser "import views" est plus propre que "*"
```

```
urlpatterns = [  
    # Page principale (Gestion globale)  
    path('gestion-utilisateurs/', views.gestion_globale, name='gestion_globale'),  
  
    # Actions sur les utilisateurs  
    path('user/add/', views.add_user, name='add_user'),  
    path('user/edit/<int:user_id>/', views.edit_user, name='edit_user'),  
    path('user/delete/<int:user_id>/', views.delete_user, name='delete_user'),  
  
    # Action sur les permissions (Menus)  
    path('user/update-menus/', views.update_menus, name='update_menus'),  
]
```

Quelques points de vérification :

1. **L'importation from .views import *** : C'est fonctionnel, mais en Python, il est souvent préférable d'utiliser from . import views. Cela permet de lire votre code plus facilement (ex: views.add_user au lieu de juste add_user).
2. **Cohérence du name** : Dans votre fichier views.py, vous utilisez redirect('gestion_globale'). Assurez-vous que le name= dans votre fichier urls.py soit bien 'gestion_globale' (comme dans mon exemple ci-dessus).
3. **Slash final** : N'oubliez pas le / à la fin de vos chemins (ex: user/add/), c'est une convention standard dans Django pour éviter des redirections inutiles.

Structure logique des routes :

Prochaine étape :

Voulez-vous que nous examinions le fichier **templates/page_utilisateur.html** ? Il est crucial que les formulaires HTML pointent vers ces nouveaux noms d'URL (ex: {% url 'add_user' %}).

code du fichier models.py : from django.db import models

```
# # from app_contact.models import (
```

```
# #     User
```

```
# # )
```

```
# # # Create your models here.
```

```
# # # https://gemini.google.com/app/067aeb3e6aaa6bc1
```

```
# # # from django.db import models
```

```
# ## class User(models.Model):
# ##     nom = models.CharField(max_length=250)
# ##     email = models.EmailField(unique=True)
# ##     password = models.TextField()

# ##     def save(self, *args, **kwargs):
# ##         if self.email:
# ##             self.email = self.email.lower()
# ##         super(User, self).save(*args, **kwargs)

# ##     def __str__(self):
# ##         return self.nom

# ## class Menu(models.Model):
# ##     nom = models.CharField(max_length=100) # Ex: "Ventes", "Stock"
# ##     code_menu = models.CharField(max_length=50, unique=True) # Ex: "menu_ventes"

# ## class MenuAcces(models.Model):
# ##     user = models.ForeignKey(User, on_delete=models.CASCADE)
# ##     menu = models.ForeignKey(Menu, on_delete=models.CASCADE)
```

```
## # est_visible = models.BooleanField(default=False)

## # class Meta:
## #     unique_together = ('user', 'menu')

# from django.db import models

# class User(models.Model):
#     nom = models.CharField(max_length=250)
#     email = models.EmailField(unique=True)
#     password = models.TextField()

#     def save(self, *args, **kwargs):
#         if self.email:
#             self.email = self.email.lower()
#         super(User, self).save(*args, **kwargs)

# class Menu(models.Model):
#     nom = models.CharField(max_length=100)
#     code_menu = models.CharField(max_length=50, unique=True)
```

```
# class MenuAcces(models.Model):
#     user = models.ForeignKey(User, on_delete=models.CASCADE)
#     menu = models.ForeignKey(Menu, on_delete=models.CASCADE)
#     est_visible = models.BooleanField(default=False)

#     class Meta:
#         unique_together = ('user', 'menu')
```

```
from django.db import models
```

```
class User(models.Model):
    nom = models.CharField(max_length=250)
    email = models.EmailField(unique=True)
    password = models.TextField()
```

```
def save(self, *args, **kwargs):
    if self.email:
        self.email = self.email.lower()
    super(User, self).save(*args, **kwargs)
```

```
class Menu(models.Model):
    nom = models.CharField(max_length=100)
    code_menu = models.CharField(max_length=50, unique=True)

class MenuAcces(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    menu = models.ForeignKey(Menu, on_delete=models.CASCADE)
    est_visible = models.BooleanField(default=False)

class Meta:
    unique_together = ('user', 'menu')
```

Votre fichier `models.py` définit une structure de base solide pour la gestion des utilisateurs et de leurs permissions. Voici une analyse de ce que vous avez mis en place et quelques recommandations pour rendre le système plus robuste.

Analyse de la structure

Votre architecture repose sur une relation **Plusieurs-à-Plusieurs (Many-to-Many)** personnalisée :

- **User** : Stocke les informations d'identification. La mise en minuscule automatique de l'email dans la méthode `save` est une excellente pratique pour éviter les doublons accidentels (ex: `Admin@test.com` vs `admin@test.com`).

- **Menu** : Représente les différentes sections de votre application (ex: Dashboard, Stocks, Factures).
 - **MenuAcces** : C'est la table de liaison (ou table de jointure). Elle permet de définir précisément quel utilisateur voit quel menu.
-

Recommandations techniques

Bien que le code soit fonctionnel, voici deux points critiques à considérer pour un projet réel :

1. **Le Hachage des mots de passe** : Actuellement, vous enregistrez le mot de passe dans un TextField. Si vous n'utilisez pas make_password dans votre vue (comme suggéré précédemment), le mot de passe sera stocké en clair, ce qui est une faille de sécurité majeure.
2. **L'utilisation de __str__** : Ajouter une méthode __str__ à vos modèles est indispensable pour que l'interface d'administration de Django (Django Admin) soit lisible.

Version optimisée du code

Python

```
from django.db import models
```

```
class User(models.Model):  
  
    nom = models.CharField(max_length=250)  
  
    email = models.EmailField(unique=True)  
  
    # On utilise souvent CharField pour les mots de passe hachés  
  
    password = models.CharField(max_length=255)
```

```
def save(self, *args, **kwargs):
```

```
    if self.email:
```

```
        self.email = self.email.lower()
```

```
super(User, self).save(*args, **kwargs)

def __str__(self):
    return f"{self.nom} ({self.email})"

class Menu(models.Model):
    nom = models.CharField(max_length=100)
    code_menu = models.CharField(max_length=50, unique=True)

    def __str__(self):
        return self.nom

class MenuAcces(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="acces")
    menu = models.ForeignKey(Menu, on_delete=models.CASCADE, related_name="utilisateurs")
    est_visible = models.BooleanField(default=False)

    class Meta:
        # Empêche d'avoir deux fois le même menu pour le même utilisateur
        unique_together = ('user', 'menu')
```

```

def __str__(self):
    etat = "Visible" if self.est_visible else "Masqué"
    return f"{self.user.nom} -> {self.menu.nom} ({etat})"

```

Résumé des relations

Modèle	Rôle	Lien
User	Entité principale	Lié à Menu via MenuAcces
Menu	Ressources du site	Lié à User via MenuAcces
MenuAcces Table pivot		Porte l'attribut est_visible

Note importante : Si vous prévoyez d'utiliser les fonctionnalités natives de Django (connexion, gestion des groupes, permissions par défaut), il est souvent préférable d'étendre le modèle AbstractUser de Django au lieu de créer une classe User de zéro.

Souhaitez-vous que je vous montre comment créer les **migrations** pour appliquer ces modèles à votre base de données ?

code du fichier page_utilisateur.html : <!DOCTYPE html>

```

<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
        <title>Administration Centrale</title>
        <style>
            .mdp-field { border: none; background: transparent; width: 100px; }

```

```
.mdp-field:focus { outline: none; }

</style>

</head>

<body class="bg-light p-4">

<div class="container shadow bg-white p-4 rounded">
    <h2 class="mb-4 text-primary">Tableau de Bord Admin</h2>

    {% if messages %}

        {% for message in messages %}

            <div class="alert alert-{{ message.tags }} alert-dismissible fade show">
                {{ message }}<button type="button" class="btn-close" data-bs-dismiss="alert"></button>
            </div>

        {% endfor %}

    {% endif %}

    <ul class="nav nav-tabs mb-4" id="adminTabs">
        <li class="nav-item"><button class="nav-link active" data-bs-toggle="tab" data-bs-target="#tab-users">Utilisateurs</button></li>
        <li class="nav-item"><button class="nav-link" data-bs-toggle="tab" data-bs-target="#tab-menus">Permissions Menus</button></li>
        <li class="nav-item"><button class="nav-link" data-bs-toggle="tab" data-bs-target="#tab-contacts">Messages ({{ contacts.count }})</button></li>
    </ul>
```

```
<div class="tab-content">

<div class="tab-pane fade show active" id="tab-users">

    <button class="btn btn-primary mb-3" data-bs-toggle="modal" data-bs-target="#addModal">+ Nouvel Utilisateur</button>

    <div class="table-responsive">

        <table class="table align-middle">

            <thead class="table-light">

                <tr><th>Nom</th><th>Email</th><th>Mot de passe</th><th>Actions</th></tr>

            </thead>

            <tbody>

                {% for u in users %}

                    <tr>

                        <td><strong>{{ u.nom }}</strong></td>

                        <td>{{ u.email }}</td>

                        <td>

                            <div class="d-flex align-items-center">

                                <input type="password" class="mdp-field mdp-toggle-target" value="{{ u.password }}" readonly>

                                <button class="btn btn-sm btn-link text-decoration-none btn-toggle-vis" type="button">👁</button>

                            </div>

                        </td>

                        <td>
```

```
<button class="btn btn-sm btn-warning" data-bs-toggle="modal" data-bs-target="#editModal{{u.id}}">Modifier</button>
<button class="btn btn-sm btn-outline-danger" data-bs-toggle="modal" data-bs-target="#delModal{{u.id}}">X</button>
</td>
</tr>

<div class="modal fade" id="editModal{{u.id}}" tabindex="-1">
  <div class="modal-dialog"><div class="modal-content">
    <form method="POST" action="{% url 'edit_user' u.id %}">{% csrf_token %}
      <div class="modal-header"><h5>Modifier {{ u.nom }}</h5></div>
      <div class="modal-body">
        <div class="mb-3"><label>Nom</label><input type="text" name="nom" class="form-control" value="{{ u.nom }}"/></div>
        <div class="mb-3"><label>Email</label><input type="email" name="email" class="form-control" value="{{ u.email }}"/></div>
        <div class="mb-3">
          <label>Mot de passe</label>
          <div class="input-group">
            <input type="password" name="password" class="form-control mdp-toggle-target" value="{{ u.password }}"/>
            <button class="btn btn-outline-secondary btn-toggle-vis" type="button">○</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="modal-footer"><button type="submit" class="btn btn-success w-100">Sauvegarder</button></div>
```

```
</form>

</div></div>

</div>

<div class="modal fade" id="delModal{{u.id}}" tabindex="-1">

    <div class="modal-dialog"><div class="modal-content">

        <form method="POST" action="{% url 'delete_user' u.id %}">{% csrf_token %}

            <div class="modal-body text-center">Supprimer l'utilisateur <b>{{ u.nom }}</b> ?</div>

            <div class="modal-footer"><button type="submit" class="btn btn-danger w-100">Confirmer la suppression</button></div>

        </form>

    </div></div>

</div>

{% endfor %}

</tbody>

</table>

</div>

</div>

<div class="tab-pane fade" id="tab-menus">

    <form method="POST" action="{% url 'update_menus' %}">

        {% csrf_token %}
```

```
<table class="table table-hover">

<thead>
    <tr><th>Utilisateurs</th>{% for m in menus %}<th>{{ m.nom }}</th>{% endfor %}</tr>
</thead>

<tbody>
    {% for u in users %}

        <tr>
            <td>{{ u.nom }}</td>
            {% for m in menus %}
                <td>
                    <div class="form-check form-switch">
                        {% with check_key=u.id|stringformat:"s"|add:"_"|add:m.id|stringformat:"s" %}
                            <input class="form-check-input" type="checkbox" name="access_{{u.id}}_{{m.id}}"
                                {% if check_key in acces_existant %}checked{% endif %}>
                        {% endwith %}
                    </div>
                </td>
            {% endfor %}

        </tr>
    {% endfor %}

</tbody>
```

```
</table>

<button type="submit" class="btn btn-success">Enregistrer les permissions</button>

</form>

</div>

<div class="tab-pane fade" id="tab-contacts">

<table class="table table-striped">

<thead><tr><th>Date</th><th>Auteur</th><th>Message</th></tr></thead>

<tbody>

  {% for c in contacts %}

    <tr>

      <td>{{ c.date_envoi|date:"d/m/Y H:i" }}</td>

      <td>{{ c.nom }}<br><small class="text-muted">{{ c.email }}</small></td>

      <td>{{ c.message }}</td>

    </tr>

  {% empty %}

  <tr><td colspan="3" class="text-center">Aucun message reçu.</td></tr>

  {% endfor %}

</tbody>

</table>

</div>
```

```
</div>

</div>

<div class="modal fade" id="addModal" tabindex="-1">
  <div class="modal-dialog"><div class="modal-content">
    <form method="POST" action="{% url 'add_user' %}">{% csrf_token %}
      <div class="modal-header"><h5>Nouvel Utilisateur</h5></div>
      <div class="modal-body">
        <input type="text" name="nom" class="form-control mb-2" placeholder="Nom complet" required>
        <input type="email" name="email" class="form-control mb-2" placeholder="Email" required>
        <div class="input-group">
          <input type="password" name="password" class="form-control mdp-toggle-target" placeholder="Mot de passe" required>
          <button class="btn btn-outline-secondary btn-toggle-vis" type="button">👁</button>
        </div>
      </div>
      <div class="modal-footer"><button type="submit" class="btn btn-primary w-100">Créer le compte</button></div>
    </form>
  </div></div>
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
```

```
<script>

document.addEventListener("DOMContentLoaded", function() {

    // 1. Mémorisation des onglets

    const activeTab = sessionStorage.getItem('activeTab');

    if (activeTab) {

        const btn = document.querySelector(`button[data-bs-target="${activeTab}"]`);

        if (btn) new bootstrap.Tab(btn).show();

    }

    document.querySelectorAll('button[data-bs-toggle="tab"]').forEach(btn => {

        btn.addEventListener('shown.bs.tab', e => sessionStorage.setItem('activeTab', e.target.getAttribute('data-bs-target')));

    });

    // 2. Bascule visibilité mot de passe

    document.querySelectorAll('.btn-toggle-vis').forEach(btn => {

        btn.addEventListener('click', function() {

            // Cherche l'input lié (soit le précédent, soit dans le même groupe)

            const parent = this.closest('.d-flex') || this.closest('.input-group');

            const input = parent.querySelector('.mdp-toggle-target');

            if (input.type === "password") {

                input.type = "text";

                this.textContent = "🙈";

            }

        });

    });

});
```

```

    } else {
        input.type = "password";
        this.textContent = "👁";
    }
});

});

});

</script>
</body>
</html>

```

Votre code HTML est excellent et très complet. L'utilisation de **Bootstrap 5** pour les modales et les onglets rend l'interface professionnelle, et votre script JavaScript pour mémoriser l'onglet actif (sessionStorage) est une très bonne astuce d'expérience utilisateur (UX) pour éviter de perdre sa place après un rafraîchissement de page.

Voici une analyse de la cohérence avec votre backend et quelques petites suggestions pour parfaire le tout.

1. Analyse de la cohérence

- **Permissions** : Votre logique de construction de clé access_{{u.id}}_{{m.id}} dans le template correspond parfaitement à la logique de récupération dans votre fonction update_menus de views.py.
- **URLs** : Les balises {% url '...' %} (comme add_user, edit_user, delete_user) sont bien alignées avec le fichier urls.py.
- **Sécurité** : Vous avez bien inclus {% csrf_token %} dans chaque formulaire, ce qui est indispensable.

2. Suggestions d'améliorations mineures

A. Amélioration de la sécurité du mot de passe

Comme mentionné pour le modèle, si vous commencez à hacher les mots de passe avec `make_password`, le champ `{{ u.password }}` affichera une chaîne illisible (ex: `pbkdf2_sha256$...`).

- **Conseil :** Dans le tableau, il vaut mieux masquer totalement ce champ ou ne pas l'afficher du tout pour les administrateurs. Dans la modale de modification, laissez le champ vide par défaut : si l'admin le remplit, on change le mot de passe, sinon on garde l'ancien.

B. UX du Switch (Permissions)

Vos boutons "Switch" sont élégants. Pour aider l'utilisateur à savoir quel changement il a fait avant de cliquer sur "Enregistrer", vous pourriez ajouter un petit style CSS pour mettre en évidence les lignes survolées.

C. Script de visibilité (Correction mineure)

Dans votre JavaScript, vous utilisez `this.textContent = "👉"`. C'est parfait, mais assurez-vous que l'encodage du fichier est bien en **UTF-8** pour que les emojis s'affichent correctement sur tous les serveurs.

3. Résumé de l'architecture finale

Voici comment vos fichiers communiquent maintenant :

Composant	Rôle
<code>models.py</code>	Définit les tables User, Menu et MenuAcces.
<code>views.py</code>	Traite les formulaires POST et prépare les données users et acces_existant.
<code>urls.py</code>	Dirige les requêtes HTTP (ex: <code>/user/add/</code>) vers les bonnes fonctions.

`page_utilisateur.html` Affiche l'interface et envoie les données via les modales.

Une petite astuce pour le futur

Si votre liste d'utilisateurs devient très longue (plus de 50), la matrice de permissions dans l'onglet "Menus" deviendra difficile à lire. Vous pourriez envisager plus tard d'ajouter une petite barre de recherche en JavaScript pour filtrer les lignes du tableau en temps réel.

Souhaitez-vous que je vous aide à ajouter une **fonction de recherche/filtrage** simple en JavaScript pour vos tableaux ?

oui

C'est une excellente idée. Quand le nombre d'utilisateurs augmente, pouvoir filtrer la liste instantanément change tout en termes de confort d'utilisation.

Voici comment ajouter cette fonctionnalité de recherche "en temps réel" sans recharger la page.

1. Ajout des champs de recherche

Ajoutez ces deux barres de recherche juste au-dessus de vos tableaux respectifs dans page_utilisateur.html :

Pour l'onglet Utilisateurs (au-dessus du tableau) :

HTML

```
<input type="text" id="searchUser" class="form-control mb-3" placeholder="Rechercher un nom ou un email...">
```

Pour l'onglet Permissions (au-dessus de la matrice) :

HTML

```
<input type="text" id="searchPermission" class="form-control mb-3" placeholder="Filtrer les utilisateurs pour les accès...">
```

2. Le script JavaScript de filtrage

Ajoutez ce bloc de code à l'intérieur de votre balise <script>, juste après votre code existant :

JavaScript

```
// Fonction de filtrage universelle  
  
function filterTable(inputId, tableBodyId) {  
  
    const input = document.getElementById(inputId);  
  
    const filter = input.value.toLowerCase();  
  
    const rows = document.querySelectorAll(`#${tableBodyId} tr`);
```

```
rows.forEach(row => {
    const text = row.textContent.toLowerCase();
    row.style.display = text.includes(filter) ? "" : "none";
});

// Écouteurs d'événements pour les recherches
document.getElementById('searchUser').addEventListener('keyup', () => {
    filterTable('searchUser', 'tab-users table tbody');
});

document.getElementById('searchPermission').addEventListener('keyup', () => {
    filterTable('searchPermission', 'tab-menus table tbody');
});
```

3. Pourquoi c'est utile ?

L'ajout de ce filtrage transforme votre administration statique en une véritable **application dynamique**. Voici ce que cela permet :

1. **Rapidité** : Le filtrage se fait dans le navigateur du client, il n'y a aucun appel au serveur Django, donc c'est instantané.
 2. **Précision** : Vous pouvez isoler un utilisateur spécifique pour modifier ses permissions sans avoir à faire défiler toute la page.
 3. **Flexibilité** : La recherche fonctionne aussi bien sur le nom que sur l'adresse email.
-

Résumé de votre application "Admin" complète :

Fichier	État	Fonctionnalité Clé
models.py	Terminé	Gestion des accès User/Menu.
views.py	Terminé	CRUD et sauvegarde hachée.
urls.py	Terminé	Routes propres et nommées.

page_utilisateur.html Optimisé Interface Tabulaire + Recherche Live.

Prochaine étape :

Votre système est maintenant prêt pour la production. Souhaitez-vous que je vous aide à mettre en place une **vérification de sécurité** dans vos autres vues pour que seuls les administrateurs puissent voir ces menus (en utilisant le modèle MenuAcces) ?

Pouvez vous mettre des "données par défaut" dans les models "class Menu(models.Model):" et "class MenuAcces(models.Model):" c'est-à-dire des "Exemples de Menus" qui sont d'ailleurs présents dans le fichier page_utilisateur.html

Pour insérer des données par défaut dans Django, la méthode la plus propre et la plus robuste consiste à utiliser un fichier de **données initiales (fixtures)** ou un script de **migration de données**.

Puisque vos menus sont déjà définis dans votre logique HTML, nous allons créer une migration qui injectera automatiquement ces menus (Ventes, Stocks, etc.) dans votre base de données.

1. Création automatique des Menus (Migration de données)

L'avantage de cette méthode est qu'à chaque fois que vous installerez votre projet sur un nouvel ordinateur, les menus seront créés tout seuls.

Lancez cette commande dans votre terminal pour créer un fichier de migration vide :

Bash

```
python manage.py makemigrations --empty app_user
```

Ouvrez le fichier qui vient d'être créé dans **app_user/migrations/0002_... (le nom varie)** et remplacez son contenu par celui-ci :

Python

```
from django.db import migrations
```

```
def creer_menus_par_defaut(apps, schema_editor):
```

```
    # On récupère le modèle Menu depuis l'historique des apps
```

```
    Menu = apps.get_model('app_user', 'Menu')
```

```
    # Liste des menus à créer (basée sur votre HTML)
```

```
    menus = [
```

```
        {'nom': 'Ventes', 'code_menu': 'menu_ventes'},
```

```
        {'nom': 'Stocks', 'code_menu': 'menu_stocks'},
```

```
        {'nom': 'Factures', 'code_menu': 'menu_factures'},
```

```
        {'nom': 'Rapports', 'code_menu': 'menu_rapports'},
```

```
    ]
```

```
for item in menus:
```

```
    Menu.objects.get_or_create(code_menu=item['code_menu'], defaults={'nom': item['nom']})
```

```
class Migration(migrations.Migration):
```

```
    dependencies = [
```

```
        ('app_user', '0001_initial'), # Doit pointer vers votre migration initiale
```

```
]
```

```
operations = [
    migrations.RunPython(creer_menus_par_defaut),
]
```

Appliquez ensuite la migration :

Bash

```
python manage.py migrate
```

2. Initialisation des Accès (MenuAcces)

Pour le modèle MenuAcces, les données dépendent des utilisateurs existants. La meilleure pratique est de modifier votre fonction add_user dans views.py pour qu'à chaque création d'un utilisateur, des accès "par défaut" (masqués) soient créés pour lui.

Modifiez votre fonction add_user dans app_user/views.py :

Python

```
def add_user(request):
    if request.method == "POST":
        # 1. Création de l'utilisateur
        nouveau_utilisateur = User.objects.create(
            nom=request.POST.get('nom'),
            email=request.POST.get('email'),
            password=make_password(request.POST.get('password')))
```

)

```
# 2. Création automatique des accès (par défaut tous à False/Masqué)
```

```
all_menus = Menu.objects.all()
```

```
for m in all_menus:
```

```
    MenuAcces.objects.get_or_create(
```

```
        user=nouveau_utilisateur,
```

```
        menu=m,
```

```
        defaults={'est_visible': False}
```

```
)
```

```
messages.success(request, f"Utilisateur {nouveau_utilisateur.nom} créé avec ses accès par défaut.")
```

```
return redirect('gestion_globale')
```

Pourquoi faire ainsi ?

1. **Cohérence** : Vous ne vous retrouverez jamais avec un utilisateur qui n'apparaît pas dans votre matrice de permissions (votre tableau HTML ne sera plus vide).
2. **Automatisation** : Dès qu'un menu est ajouté en base de données, il sera prêt à être coché pour chaque utilisateur.
3. **Intégrité** : La migration garantit que vos menus "Ventes", "Stocks", etc., existent toujours, même si vous réinitialisez votre base de données.

Souhaitez-vous que je vous aide à rédiger une petite commande pour créer un "**Super-Administrateur**" par défaut qui aurait déjà tous les accès activés ?

code complet

Voici le code complet et harmonisé pour votre application. J'ai intégré les **recherches dynamiques**, la **sécurité des mots de passe**, et la **gestion automatique des accès**.

1. Modèles (app_user/models.py)

Ce fichier définit la structure de vos données.

Python

```
from django.db import models

class User(models.Model):
    nom = models.CharField(max_length=250)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=255) # Stockage du hash

    def save(self, *args, **kwargs):
        if self.email:
            self.email = self.email.lower()
        super(User, self).save(*args, **kwargs)

    def __str__(self):
        return self.nom

class Menu(models.Model):
    nom = models.CharField(max_length=100) # Ex: "Ventes"
    code_menu = models.CharField(max_length=50, unique=True) # Ex: "menu_ventes"
```

```
def __str__(self):
    return self.nom

class MenuAcces(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="acces")
    menu = models.ForeignKey(Menu, on_delete=models.CASCADE)
    est_visible = models.BooleanField(default=False)

    class Meta:
        unique_together = ('user', 'menu')
```

2. Vues (app_user/views.py)

Le moteur qui traite vos formulaires et affiche la page.

Python

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.contrib.auth.hashers import make_password
from .models import User, Menu, MenuAcces
from app_contact.models import Contact
```

```
def gestion_globale(request):
    users = User.objects.all()
    menus = Menu.objects.all()
    contacts = Contact.objects.all().order_by('-date_envoi')
```

```
# Prépare les IDs d'accès pour cocher les cases dans le HTML
```

```
acces_existant = [
    f"{{a.user_id}}_{{a.menu_id}}"
    for a in MenuAcces.objects.filter(est_visible=True)
]
```

```
return render(request, 'page_utilisateur.html', {
    'users': users,
    'menus': menus,
    'contacts': contacts,
    'acces_existant': acces_existant
})
```

```
def add_user(request):
    if request.method == "POST":
        u = User.objects.create(
```

```
nom=request.POST.get('nom'),
email=request.POST.get('email'),
password=make_password(request.POST.get('password'))
)

# Création automatique des lignes d'accès pour chaque menu existant

for m in Menu.objects.all():

    MenuAcces.objects.get_or_create(user=u, menu=m, defaults={'est_visible': False})

messages.success(request, "Utilisateur et accès créés.")

return redirect('gestion_globale')


def edit_user(request, user_id):

    if request.method == "POST":

        user = get_object_or_404(User, id=user_id)

        user.nom = request.POST.get('nom')

        user.email = request.POST.get('email')

        new_pwd = request.POST.get('password')

        if new_pwd and not new_pwd.startswith('pbkdf2_'): # Évite de re-hacher un hash

            user.password = make_password(new_pwd)

        user.save()

        messages.info(request, "Profil mis à jour.")
```

```
return redirect('gestion_globale')

def delete_user(request, user_id):
    if request.method == "POST":
        get_object_or_404(User, id=user_id).delete()
        messages.error(request, "Utilisateur supprimé.")
    return redirect('gestion_globale')

def update_menus(request):
    if request.method == "POST":
        for user in User.objects.all():
            for menu in Menu.objects.all():
                key = f"access_{user.id}_{menu.id}"
                MenuAcces.objects.update_or_create(
                    user=user, menu=menu,
                    defaults={'est_visible': key in request.POST}
                )
        messages.success(request, "Permissions enregistrées.")
    return redirect('gestion_globale')
```

3. Template (templates/page_utilisateur.html)

L'interface utilisateur avec **Recherche Live** et **Bootstrap**.

HTML

```
<!DOCTYPE html>

<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <title>Administration Centrale</title>
    <style>
        .mdp-field { border: none; background: transparent; width: 120px; font-family: monospace; }
        .tab-content { padding-top: 20px; }
    </style>
</head>
<body class="bg-light p-4">

    <div class="container shadow bg-white p-4 rounded">
        <h2 class="mb-4 text-primary">Tableau de Bord Admin</h2>
        {% if messages %}
            {% for message in messages %}
```

```
<div class="alert alert-{{ message.tags }} alert-dismissible fade show">  
    {{ message }}<button type="button" class="btn-close" data-bs-dismiss="alert"></button>  
</div>  
{% endfor %}  
{% endif %}  
  
<ul class="nav nav-tabs mb-4" id="adminTabs">  
    <li class="nav-item"><button class="nav-link active" data-bs-toggle="tab" data-bs-target="#tab-users">Utilisateurs</button></li>  
    <li class="nav-item"><button class="nav-link" data-bs-toggle="tab" data-bs-target="#tab-menus">Permissions Menus</button></li>  
    <li class="nav-item"><button class="nav-link" data-bs-toggle="tab" data-bs-target="#tab-contacts">Messages ({{ contacts.count }})</button></li>  
</ul>  
  
<div class="tab-content">  
    <div class="tab-pane fade show active" id="tab-users">  
        <div class="d-flex justify-content-between mb-3">  
            <button class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#addModal">+ Nouveau</button>  
            <input type="text" id="searchUser" class="form-control w-25" placeholder="Filtrer..."/>  
        </div>  
        <table class="table align-middle" id="tableUsers">  
            <thead class="table-light"><tr><th>Nom</th><th>Email</th><th>Actions</th></tr></thead>  
            <tbody>
```

```

{% for u in users %}

<tr>
    <td><strong>{{ u.nom }}</strong></td>
    <td>{{ u.email }}</td>
    <td>
        <button class="btn btn-sm btn-warning" data-bs-toggle="modal" data-bs-target="#editModal{{u.id}}">Modifier</button>
        <button class="btn btn-sm btn-outline-danger" data-bs-toggle="modal" data-bs-target="#delModal{{u.id}}">X</button>
    </td>
</tr>
{% endfor %}

</tbody>
</table>
</div>

<div class="tab-pane fade" id="tab-menus">

    <input type="text" id="searchPerm" class="form-control mb-3 w-25" placeholder="Chercher utilisateur...">
    <form method="POST" action="{% url 'update_menus' %}">
        {% csrf_token %}
        <table class="table table-hover" id="tablePerms">
            <thead>
                <tr><th>Utilisateurs</th>{% for m in menus %}<th>{{ m.nom }}</th>{% endfor %}</tr>

```

```
</thead>

<tbody>

  {% for u in users %}

    <tr>

      <td>{{ u.nom }}</td>

      {% for m in menus %}

        <td>

          <div class="form-check form-switch">

            {% with check_key=u.id|stringformat:"s"|add:"_"|add:m.id|stringformat:"s" %}

              <input class="form-check-input" type="checkbox" name="access_{{u.id}}_{{m.id}}"

                {% if check_key in acces_existant %}checked{% endif %}>

            {% endwith %}

          </div>

        </td>

      {% endfor %}

    </tr>

  {% endfor %}

</tbody>

</table>

<button type="submit" class="btn btn-success">Enregistrer les permissions</button>

</form>
```

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

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
<script>
    // Filtrage dynamique

    function setupSearch(inputId, tableView) {
        document.getElementById(inputId).addEventListener('keyup', function() {
            let filter = this.value.toLowerCase();

            let rows = document.querySelectorAll(`#${tableView} tbody tr`);

            rows.forEach(row => {
                row.style.display = row.textContent.toLowerCase().includes(filter) ? "" : "none";
            });
        });
    }

    setupSearch('searchUser', 'tableUsers');

    setupSearch('searchPerm', 'tablePerms');

</script>
</body>
</html>
```

4. Migration des Menus par défaut

4.1. ALLEZ A L'ETAPE PRECEDENTE :

1. Création automatique des Menus (Migration de données)

Afin d'EXECUTER la COMMANDE suivante dans le TERMINAL DE DJANGO (CTRL J) :

4.1. Création automatique des Menus (Migration de données)

L'avantage de **cette méthode** est qu'à **chaque fois** que **vous installerez votre projet sur un**

nouvel ordinateur, les **menus** seront **créés tout seuls.**

Lancez cette commande dans votre terminal pour créer un fichier de migration vide :

Bash

```
python manage.py makemigrations --empty app_user
```

Ouvrez le fichier qui vient d'être créé dans **app_user/migrations/0002_...** (le nom varie) et remplacez son contenu par celui-ci :

Python

4.2. Créez un fichier **app_user/migrations/0002_setup_menus.py** :

Python

```
from django.db import migrations
```

```
def create_default_menus(apps, schema_editor):  
  
    Menu = apps.get_model('app_user', 'Menu')  
  
    data = [  
        ('Ventes', 'menu_ventes'),  
        ('Stocks', 'menu_stocks'),  
        ('Factures', 'menu_factures'),  
        ('Rapports', 'menu_rapports'),  
    ]  
  
    for nom, code in data:  
        Menu.objects.get_or_create(code_menu=code, defaults={'nom': nom})
```

```
class Migration(migrations.Migration):  
  
    dependencies = [('app_user', '0001_initial')]  
  
    operations = [migrations.RunPython(create_default_menus)]
```

Appliquez ensuite la migration :

Bash

python manage.py migrate

N'oubliez pas de lancer `python manage.py makemigrations` puis `python manage.py migrate` pour activer tout cela. Souhaitez-vous que je vous explique comment tester la connexion d'un utilisateur avec ces menus ?

Vérifiez les réponses de Gemini, car il peut se tromper.