TP Python Django

TP6 Python Django – Sécurité et site d'administration

TOYI Francois

October 2024

1 Authentification

1. Voici la partie du fichier settings.py qui contient les donnée pour l'authentification, on a :

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

2.Voici se que j'avais lister, on a :

- administrateur : qui à accès à tout les crud total et qui peut manipuler les données .
- Les enseignants .
- Les élèves .
- Les visiteurs de notre site aussi qui ont un accès très limité sur le site .

Ceux qui peuvent utiliser la vue de formulaire d'ajout d'une note sont :

- administrateur
- les enseignants

3. Ces 2 classes sont : Utilisateurs et groupes .

Il y a l'utilisateurs qu'on a créer qui s'affiche là bas . Oui c'est normal et cela viens du fait qu'on à créer un superuser pour notre site admin.

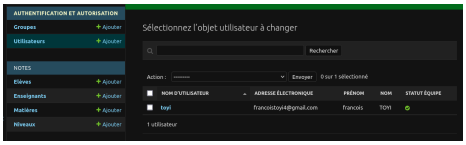

Figure 1: Listes des Utilisateurs

4. Créons les 3 groupes, on a :
Groupes Élèves :

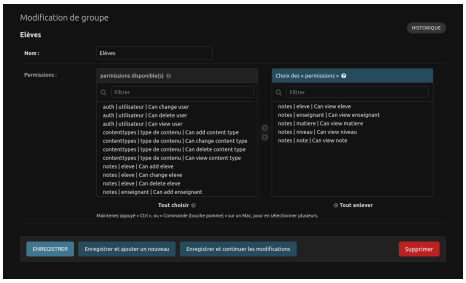

Figure 2: Groupes élèves et ces droits

Groupes Enseignants :

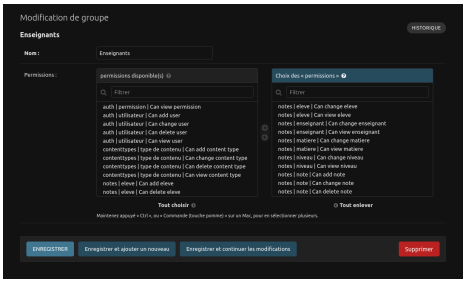

Figure 3: Groupes enseignants et ces droits

Groupes Admin :

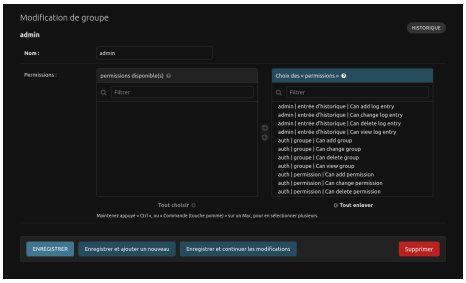

Figure 4: Groupes admin et ces droits

5. Les 2 manières d'étendre le modèle **User** par défaut sans que cela modifie notre propre modèle c'est de créer le modèle User personnalisé en utilisant **AbstractUser** où en créant un profil utilisateur avec une relation **OneToOneField**.

6. J'ai ajouté un attribut dans la models personne, voici la ligne que j'ai ajouter à ma classe, on a :

```
user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
```

7.Voici une image des instance d'utilisateur, on a :

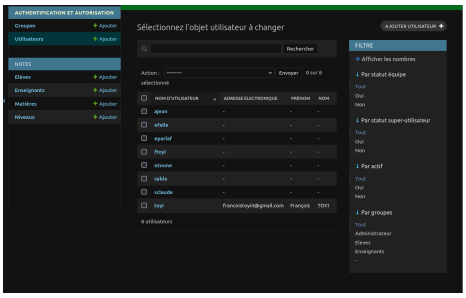

Figure 5: Users

8. Voici se que j'ai fait et voici la copie, on a :

```
toyi@toyi-Latitude-E6420:~/Documents/L3/L3/sem_5/django/ifnti_l3$ py manage.py

makemigrations
It is impossible to change a nullable field 'user' on eleve to non-nullable without

providing a default. This is because the database needs something to populate existing
rows.
Please select a fix:
 1) Provide a one-off default now (will be set on all existing rows with a null value for
 this column)
 2) Ignore for now. Existing rows that contain NULL values will have to be handled
 manually,for example with a RunPython or RunSQL operation.
 3) Quit and manually define a default value in models.py.
Select an option: 2
It is impossible to change a nullable field 'user' on enseignant to non-nullable without
providing a default. This is because the database needs something to populate existing rows.
Please select a fix:
 1) Provide a one-off default now (will be set on all existing rows with a null value for th
 2) Ignore for now. Existing rows that contain NULL values will have to be handled manually,
 3) Quit and manually define a default value in models.py.
Select an option: 2
Migrations for 'notes':
  notes/migrations/0005_alter_eleve_user_alter_enseignant_user.py
    ~ Alter field user on eleve
    ~ Alter field user on enseignant
toyi@toyi-Latitude-E6420:~/Documents/L3/L3/sem_5/django/ifnti_l3$ py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, notes, sessions
Running migrations:
  Applying notes.0005_alter_eleve_user_alter_enseignant_user... OK
toyi@toyi-Latitude-E6420:~/Documents/L3/L3/sem_5/django/ifnti_l3$
```

9. J'ai associé les élèves aux groupes des **Eleves**, les enseignants aux groupes des **Enseignants** .

10. Pour qu'on exige l'authentification et qu'on exige les permissions j'ai utilisé les méthodes suivantes :

```
@login_require                    d
```

11. Pour tester correctement les restrictions d'accès aux vues, il est important de se déconnecter et simuler un utilisateur non authentifié.

J'ai respectivement les erreur suivants :

- **Erreur 403 :** Permission denied" si l'utilisateur n'a pas les permissions .
- **Erreur 302 :** Redirection vers la page de connexion pour les utilisateurs non authentifiés

12.Ici pour éviter cet erreur on vas ajouter le mot clé **@permission_required** avec une valeur appropriée.

Code à écrire, on a :

```
from django.shortcuts import render, get_object_or_404, redirect
from django.http import HttpResponse, HttpResponseBadRequest
from notes.models import Eleve, Matiere, Note
from notes.forms.NoteForm import NoteForm
from django.contrib.auth.decorators import login_required, permission_required




@login_required
@permission_required('notes.add_note')
def add_note(request, eleve_id, matiere_id):
    eleve = get_object_or_404(Eleve, id=eleve_id)
    matiere = get_object_or_404(Matiere, id=matiere_id)

    # Vérification que l'élève suit bien la matière
    if matiere not in eleve.matieres.all():
        raise Exception(f"L'élève {eleve.nom} ne suit pas la matière {matiere.nom}.")

    if request.method == 'POST':
        form = NoteForm(request.POST)
        if form.is_valid():
            # Création de la note avec les relations élève et matière
            note = form.save(commit=False)
            note.eleve = eleve
            note.matiere = matiere
            note.save()
            return HttpResponse(f"Note {note.valeur} ajoutée pour {eleve.nom} en {matiere.no
        else:
            return HttpResponseBadRequest("Formulaire invalide. Veuillez vérifier les valeu
    else:
        # Affichage du formulaire si la requête est en GET
        form = NoteForm()

    return render(request, 'notes/add_note.html', {'form': form, 'eleve': eleve, 'matiere':
```

Déconnexion

Dans Django, la déconnexion d'un utilisateur peut être effectuée en utilisant la vue intégrée `LogoutView`. Cette vue gère la déconnexion en détruisant la session de l'utilisateur, ce qui le déconnecte de l'application.

- **URL de déconnexion:** Par défaut, Django fournit une URL pour la déconnexion : `/accounts/logout/`.
- **Vue de déconnexion:** Utilisez `django.contrib.auth.views.LogoutView` pour gérer la déconnexion.

Exemple de configuration de l'URL :

```
from django.urls import path
from django.contrib.auth.views import LogoutView

urlpatterns = [
    path('logout/', LogoutView.as_view(), name='logout'),
]
```

Changement de mot de passe

Django offre une vue intégrée pour changer le mot de passe de l'utilisateur. Cette vue permet à un utilisateur authentifié de changer son mot de passe actuel pour un nouveau mot de passe.

- **URL pour changer le mot de passe :** Par défaut, Django propose l'URL `/accounts/password/change/`.
- **Vue pour changer le mot de passe :** Utilisez `django.contrib.auth.views.PasswordChangeView` pour gérer cette fonctionnalité.

Exemple de configuration de l'URL :

```
from django.urls import path
from django.contrib.auth.views import PasswordChangeView

urlpatterns = [
    path('password/change/', PasswordChangeView.as_view(), name='password_change'),
]
```

Réinitialisation du mot de passe

La réinitialisation du mot de passe est une fonctionnalité qui permet à un utilisateur ayant oublié son mot de passe de le réinitialiser via un e-mail de réinitialisation.

- **Vues de réinitialisation :**
  - `PasswordResetView` : Affiche un formulaire pour saisir l'email.
  - `PasswordResetDoneView` : Affiche un message confirmant l'envoi de l'e-mail.
  - `PasswordResetConfirmView` : Permet à l'utilisateur de saisir un nouveau mot de passe.
  - `PasswordResetCompleteView` : Confirme que le mot de passe a été réinitialisé avec succès.
- **URL pour réinitialiser le mot de passe :** Par défaut, les URL suivantes sont utilisées :
  - `/accounts/password/reset/`
  - `/accounts/password/reset/done/`
  - `/accounts/password/reset/<uidb64>/<token>/`
  - `/accounts/password/reset/complete/`

Exemple de configuration de l'URL :

```
from django.urls import path
from django.contrib.auth.views import PasswordResetView, PasswordResetDoneView, PasswordRese

urlpatterns = [
    path('password/reset/', PasswordResetView.as_view(), name='password_reset'),
    path('password/reset/done/', PasswordResetDoneView.as_view(), name='password_reset_done
    path('password/reset/<uidb64>/<token>/', PasswordResetConfirmView.as_view(), name='passw
    path('password/reset/complete/', PasswordResetCompleteView.as_view(), name='password_res
]
```

Les problèmes de sécurité

1. Qu'est-ce que le Cross Site Scripting (XSS) ? Comment est-ce que Django s'occupe de ce problème ? Quelles sont ses limites ? Que devez-vous faire pour éviter les attaques XSS ?

Le Cross Site Scripting (**XSS**) est une vulnérabilité de sécurité dans les applications web qui permet à un attaquant d'injecter des scripts malveillants dans des pages web qui seront ensuite exécutées par le navigateur de l'utilisateur. Ces scripts peuvent voler des informations sensibles, comme les cookies, ou exécuter des actions non autorisées au nom de l'utilisateur.

Django protège contre les attaques XSS en échappant par défaut toutes les données envoyées dans les templates. Cela signifie que Django transforme les

caractères spéciaux (comme "<", ">", et "&") en leurs entités HTML correspondantes, ce qui empêche le navigateur d'exécuter du JavaScript malveillant.

Limites : Django ne protège pas contre les attaques XSS dans tous les cas. Par exemple, si un utilisateur désactive l'échappement des caractères dans un template (par exemple en utilisant `% raw %` ou en exécutant du code JavaScript dans des contextes spécifiques), l'application reste vulnérable.

Prévention : Pour éviter les attaques XSS, il est recommandé de :

- Toujours utiliser les fonctionnalités d'échappement de Django.
- Éviter d'utiliser `% raw %` ou d'autres mécanismes qui désactivent l'échappement.
- Utiliser des bibliothèques de nettoyage comme `bleach` si du contenu HTML est autorisé.

2. Qu'est-ce que CSRF ? En quoi cela consiste-t-il ? Vous ne vous en souvenez certainement pas, mais vous avez déjà fait quelque chose permettant d'éviter les attaques CSRF. Qu'avez-vous donc déjà fait ? En quoi consiste cette protection ? Dans quel cas devez-vous vous protéger ? Quelles sont les limites de la protection de Django contre les attaques CSRF ?

Le Cross Site Request Forgery (CSRF) est une attaque où un attaquant force un utilisateur authentifié à effectuer des actions non autorisées sur une application web.

Django protège contre cette attaque en utilisant des tokens CSRF.

Voici se que j'ai eut à faire, on a :

```
    <!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ajouter Élève</title>
</head>
<body>
    <h1>Ajouter un Élève</h1>
    <form method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary">Enregistrer</button>
    </form>
```

```
    <a href="{% url 'notes:eleves' %}">Retour à la liste des élèves</a>
</body>
</html>
```

Cette protection consiste : à inclure une balise `% csrf_token %` dans les formulaires, et toutes les vues de type `POST` nécessitent un token CSRF valide.

Cas de protection : Vous devez vous protéger contre les attaques CSRF dans toutes les vues qui modifient l'état du serveur, comme les mises à jour de base de données.

Limites : La protection CSRF de Django est efficace pour les requêtes de type `POST`, mais elle ne protège pas contre les attaques `GET`, et elle ne fonctionne pas si les cookies sont désactivés ou si des requêtes externes sont autorisées sans restrictions.

3. Qu'est-ce qu'une injection SQL ? Comment Django se protège de ce type d'attaque ? Avec quelles limites ?

L'injection SQL est une vulnérabilité où un attaquant insère des commandes SQL malveillantes dans une requête pour accéder ou manipuler des données.

Django protège contre les injections SQL en utilisant des **requêtes préparées** et en paramétrant automatiquement toutes les requêtes de base de données. Au lieu de construire des chaînes SQL manuellement, Django utilise un ORM (Object-Relational Mapping) qui génère des requêtes SQL sécurisées.

Limites : Django ne protège pas contre les injections SQL si vous utilisez des requêtes SQL brutes sans paramètres sécurisés (par exemple, avec `connection.cursor()`).

Prévention : Pour éviter les injections SQL, utilisez toujours l'ORM de Django ou des requêtes préparées avec des paramètres sécurisés.

4. Qu'est-ce que le détournement de clic ? En quoi cela consiste-t-il ? Donnez un exemple d'attaque par détournement de clic.

Le détournement de clic (Clickjacking) est une attaque où un utilisateur est incité à cliquer sur un élément invisible ou déguisé d'une page web, ce qui déclenche des actions non intentionnelles, telles que l'envoi de données personnelles ou la modification de paramètres.

Un exemple d'attaque de détournement de clic pourrait être une page web contenant un iframe caché qui pointe vers une page de paramètres d'un utilisateur. Si l'utilisateur clique sur un bouton prétendant effectuer une autre action, il peut en réalité changer ses paramètres sans le savoir.

Prévention : Utilisez des en-têtes HTTP tels que `X-Frame-Options` pour empêcher que votre site soit intégré dans un iframe.

5. Qu'est-ce que SSL ? Qu'est-ce que HTTPS ? Qu'est-ce que HSTS ? Expliquez aussi pourquoi ces sigles sont importants.

SSL (Secure Sockets Layer) est un protocole de sécurité qui permet de chiffrer les communications entre un client et un serveur.

HTTPS (HyperText Transfer Protocol Secure) est la version sécurisée de HTTP. Il utilise SSL/TLS pour chiffrer les données échangées entre le client et le serveur, garantissant ainsi la confidentialité et l'intégrité des informations.

HSTS (HTTP Strict Transport Security) est un mécanisme de sécurité qui oblige les navigateurs à utiliser uniquement des connexions HTTPS pour accéder à un site web, empêchant les attaques de type `man-in-the-middle`.

6. Donnez une explication brève sur les Sessions en Django, sur la Validation de l'en-tête Host et sur le Contenu envoyé par les utilisateurs.

Sessions en Django : Les sessions permettent de stocker des informations sur un utilisateur entre les requêtes. Django utilise un système de stockage de sessions côté serveur, et chaque utilisateur reçoit un identifiant de session unique.

Validation de l'en-tête Host : Django valide l'en-tête `Host` de chaque requête pour s'assurer que la demande provient d'un domaine autorisé, ce qui empêche certaines attaques comme l'usurpation d'identité de domaine.

Contenu envoyé par les utilisateurs : Django fournit des mécanismes pour valider et nettoyer les données envoyées par les utilisateurs afin de prévenir les attaques telles que XSS ou les injections SQL. Il est important de valider tous les champs de formulaire et d'échapper les données avant de les afficher.

Customiser le site d'administration de Django

1. Pourquoi est-il intéressant de customiser le site d'administration ?

Customiser le site d'administration de Django présente plusieurs intérêts majeurs :

- **Amélioration de l'expérience utilisateur :** Le site d'administration par défaut de Django est fonctionnel mais générique. En le personnalisant, il devient plus intuitif et adapté aux besoins spécifiques des utilisateurs finaux.
- **Gain d'efficacité :** En organisant les informations et les actions de manière claire (par exemple en regroupant ou en filtrant des champs, ou

en ajoutant des raccourcis), les administrateurs peuvent travailler plus rapidement et éviter les erreurs.

- **Mise en conformité avec l'identité visuelle :** La personnalisation permet d'intégrer des éléments graphiques ou des couleurs propres à l'organisation, renforçant ainsi la cohérence de l'image de marque.
- **Adaptation aux besoins spécifiques :** Certaines organisations nécessitent des fonctionnalités ou des données spécifiques qui ne sont pas disponibles dans l'administration par défaut. Les customisations permettent d'ajouter des champs, des actions, ou des vues adaptées.

2. Pour qui sera-t-il amélioré ?

La customisation du site d'administration bénéficie principalement aux groupes suivants :

- **Les administrateurs système :** Ceux qui gèrent les utilisateurs et les droits d'accès apprécieront une interface simplifiée pour leurs tâches fréquentes, comme l'ajout d'utilisateurs ou la modification de permissions.
- **Les responsables métier :** Dans le cas d'une application liée à une organisation (par exemple une école ou une entreprise), les responsables qui utilisent l'administration pour gérer des données (élèves, enseignants, commandes, etc.) bénéficieront d'une interface qui reflète leur vocabulaire et leurs besoins spécifiques.
- **Les développeurs et concepteurs :** Une administration bien customisée est plus facile à maintenir et peut être adaptée rapidement pour répondre aux évolutions des besoins de l'organisation.
- **L'organisation dans son ensemble :** Une gestion administrative plus fluide et efficace contribue indirectement à une meilleure gestion globale de l'organisation.

2 Comment pourrions-nous rendre nos modèles accessibles depuis le site d'administration d'une autre manière que celle que nous avons utilisée ?

On peut le faire en les enregistrant dans le fichier `admin.py` avec la fonction `admin.site.register()`.

```
from django.contrib import admin
from .models import MonModele

admin.site.register(MonModele)
```

3. Expliquons les termes suivantes, on a :

- **date_hierarchy** : Permet d'ajouter une barre de navigation temporelle dans l'interface d'administration pour filtrer les objets en fonction d'une date ou d'un champ de type `DateField/DateTimeField`.
- **empty_value_display** : Définit une valeur par défaut affichée pour les champs ayant une valeur `null` ou vide dans la liste d'objets.
- **list_filter** : Ajoute des filtres dans la barre latérale pour permettre un filtrage rapide des objets par des champs spécifiques.
- **list_per_page** : Spécifie le nombre d'objets affichés par page dans la liste des objets.
- **list_display, list_display_links et list_editable** :
  - **list_display** : Définit les colonnes affichées dans la liste des objets.
  - **list_display_links** : Spécifie les colonnes qui contiennent des liens vers l'édition de l'objet.
  - **list_editable** : Permet d'éditer directement certaines colonnes sans ouvrir la page de détail.
- **fields et exclude** :
  - **fields** : Liste des champs à afficher dans le formulaire d'édition.
  - **exclude** : Liste des champs à exclure du formulaire.
- **save_as et save_as_continue** :
  - **save_as** : Affiche un bouton pour enregistrer un objet en tant que nouveau.
  - **save_as_continue** : Après avoir enregistré comme nouveau, continue sur la page de modification.
- **raw_id_fields** : Affiche un champ de recherche pour les relations au lieu d'un menu déroulant.
- **readonly_fields** : Rend certains champs non modifiables dans l'interface d'administration.
- **show_full_result_count** : Permet d'afficher ou non le nombre total d'objets dans la liste.
- **save_on_top** : Ajoute les boutons `Enregistrer` en haut du formulaire d'édition.
- **sortable_by** : Détermine quels champs peuvent être triés dans la liste des objets.

- **search_fields** : Ajoute une barre de recherche pour rechercher des objets en fonction de champs spécifiques.
- **filter_horizontal et filter_vertical** : Pour les relations `ManyToManyField`, permet de choisir entre une interface horizontale ou verticale pour sélectionner les objets associés.
- **view_on_site** : Affiche un lien `Voir sur le site` pour chaque objet, permettant de le visualiser dans l'application publique.

4. Voici la classe EleveAdmin et une capture d'écran associé.

```
from django.contrib import admin
from .models import Eleve, Niveau, Matiere
from .forms.EleveForm import EleveForm  # Si ce formulaire existe

class EleveAdmin(admin.ModelAdmin):
    form = EleveForm  # Si ce formulaire personnalisé existe
    list_display = ('id', 'prenom', 'nom', 'niveau')  # Assurez-vous que 'prenom' et 'nom' v
    list_filter = ('niveau',)  # Filtrable par niveau
    search_fields = ('id', 'prenom', 'nom')  # Recherche par ID, prénom, ou nom
    filter_horizontal = ('matieres',)  # Permet une interface plus conviviale pour les ManyT
    readonly_fields = ()  # Supprimez 'date_inscription' ou ajoutez un champ existant
    fields = ('id', 'prenom', 'nom', 'niveau', 'matieres')  # Champs affichés dans le formul
    list_per_page = 10  # Pagination des résultats

# Enregistrement du modèle avec la configuration admin personnalisée
admin.site.register(Eleve, EleveAdmin)
admin.site.register(Niveau)
admin.site.register(Matiere)
```

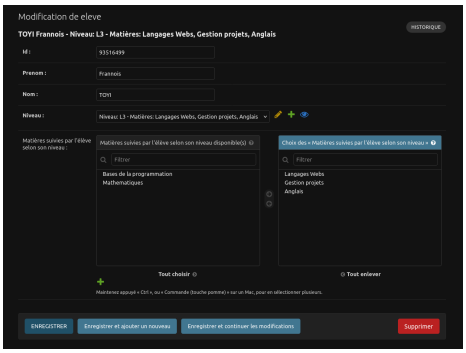

Figure 6: Page admin d'élève

5. Oui on peut modifier les templates d'administration. Pour définir les templates par défaut on doit le faire aux niveaux de setting en incluant dans

le paramètre `DIRS` de la configuration `TEMPLATES`; Oui, il est également possible de créer de nouveaux templates pour des fonctionnalités ou des sections personnalisées.

6. Oui il est tout à fait possible de modifier ou d'ajouter du JavaScript et du CSS au site d'administration de Django. Pour insérer notre propre code javascript où css on doit créer une copie du template d'administration qu'on souhaite personnaliser, par exemple `base_site.html`. En suite on vas utiliser les balises suivantes des balises `<link>` ou `<script>`.

Exemple :

```
{% block extrahead %}
    {{ block.super }}
    <link rel="stylesheet" type="text/css" href="{% static 'css/custom.css' %}">
    <script src="{% static 'js/custom.js' %}"></script>
{% endblock %}
```

En fin on créer notre dossier static qui vas contenir tout nos fichier static .js où .css .