

Tp 06 Sécurité et site administration

AMONA Birewa Audrey

4 décembre 2023

Table des matières

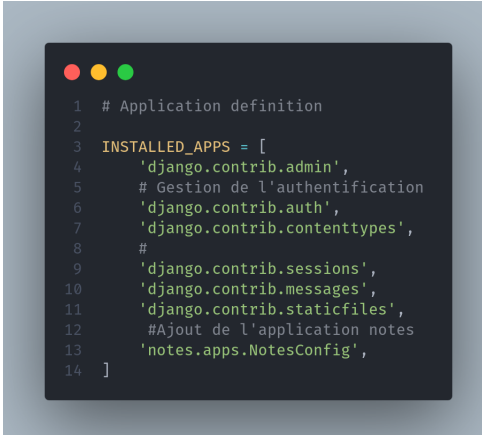
1	Authentification	1
2	Les problèmes de sécurité	7
3	Customiser le site d'administration	8

Résumé

Ce TP contient 3 parties indépendantes. D'abord nous nous pencherons sur l'authentification. Ensuite nous explorerons les problèmes de sécurité : CSRF, HTTPS, clé de chiffrement et détournement de clic. Enfin nous verrons comment customiser le site d'administration.

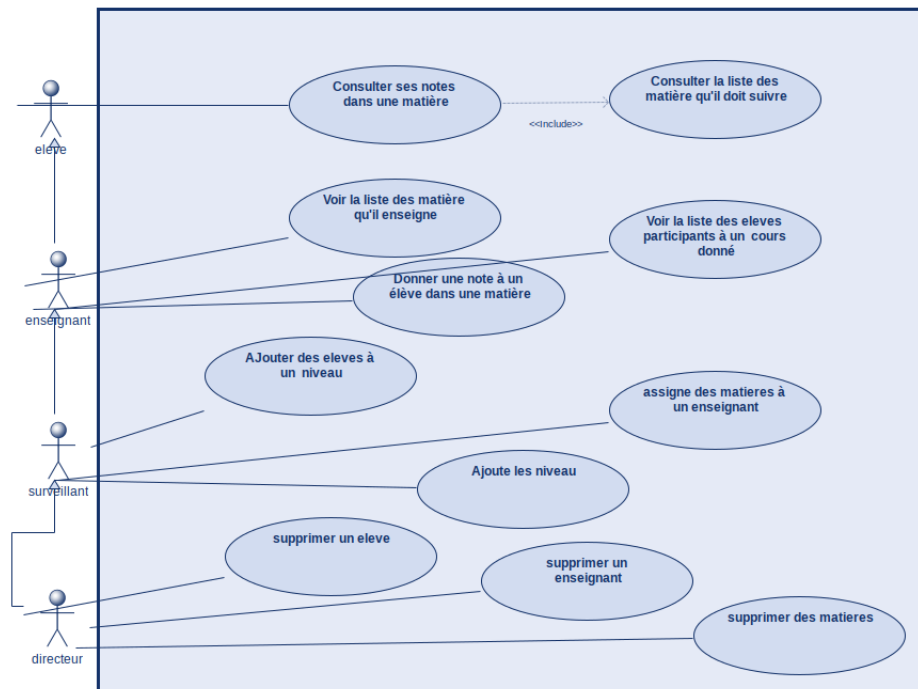
1 Authentification

1. Dans le fichier **settings.py** voici les lignes qui gère l'authentification :



```
1 # Application definition
2
3 INSTALLED_APPS = [
4     'django.contrib.admin',
5     # Gestion de l'authentification
6     'django.contrib.auth',
7     'django.contrib.contenttypes',
8     #
9     'django.contrib.sessions',
10    'django.contrib.messages',
11    'django.contrib.staticfiles',
12    #Ajout de l'application notes
13    'notes.apps.NotesConfig',
14 ]
```

2. Dans le Tp précédent, nous avons spécifié les différentes fonctionnalités en fonction des utilisateurs du système :



Il était convenu que seul les enseignant, les surveillants ou encore le directeur peuvent avoir accès au formulaire d'ajout de note.

3. Dans le site administration, nous remarquons qu'il y'a deux modèles de plus **Users** et **Groups**, que nous n'avons pas ajouté nous même. Le modèle **Users** possède les attributs suivant : username, lastname, firstname, email adress, staff, status. Il possède un enregistrement qui a pour username le nom de notre superutilisateur créé.
4. Nous allons créer des groupes avec les différents roles du diagramme de cas d'utilisation précédent.

eleve

HISTORY

Name:

eleve

Permissions:

Available permissions

Filter

auth | user | Can add user
auth | user | Can change user
auth | user | Can delete user
auth | user | Can view user
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type
contenttypes | content type | Can delete content type
contenttypes | content type | Can view content type
notes | Eleve | Can add Eleve
notes | Eleve | Can change Eleve
notes | Eleve | Can delete Eleve
notes | Enseignant | Can add Enseignant

Chosen permissions

Filter

notes | Eleve | Can view Eleve
notes | Matiere | Can view Matiere
notes | niveau | Can view niveau
notes | Note | Can view Note

enseignant

HISTORY

Name:

enseignant

Permissions:

Available permissions ?

Q

Filter

auth | permission | Can delete permission
auth | permission | Can view permission
auth | user | Can add user
auth | user | Can change user
auth | user | Can delete user
auth | user | Can view user
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type
contenttypes | content type | Can delete content type
contenttypes | content type | Can view content type
notes | Eleve | Can add Eleve
notes | Eleve | Can delete Eleve

Chosen permissions ?

Q

Filter

notes | Eleve | Can change Eleve
notes | Eleve | Can view Eleve
notes | Enseignant | Can view Enseignant
notes | Matiere | Can view Matiere
notes | niveau | Can view niveau
notes | Note | Can add Note
notes | Note | Can change Note
notes | Note | Can delete Note
notes | Note | Can view Note

surveillant

HISTORY

Name:

surveillant

Permissions:

Available permissions ?

Q

Filter

auth | permission | Can change permission
auth | permission | Can delete permission
auth | permission | Can view permission
auth | user | Can add user
auth | user | Can change user
auth | user | Can delete user
auth | user | Can view user
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type
contenttypes | content type | Can delete content type
contenttypes | content type | Can view content type
sessions | session | Can add session

Chosen permissions ?

Q

Filter

notes | Eleve | Can add Eleve
notes | Eleve | Can change Eleve
notes | Eleve | Can delete Eleve
notes | Eleve | Can view Eleve
notes | Enseignant | Can add Enseignant
notes | Enseignant | Can change Enseignant
notes | Enseignant | Can delete Enseignant
notes | Enseignant | Can view Enseignant
notes | Matiere | Can add Matiere
notes | Matiere | Can change Matiere
notes | Matiere | Can delete Matiere
notes | Matiere | Can view Matiere

Choose all ?

Remove all

- Django fournit une classe `User` qui contient l'ensemble des informations nécessaires à l'authentification et aux autorisations. Nous pouvons étendre ce modèle sans modifier nos modèles utilisateurs par défaut soit :
 - En créant une relation d'héritage entre notre modèle et le modèle par défaut `Users`
 - En créant une relation `OneToOne` entre notre modèle et le modèle par défaut `Users`
- Nous allons procéder à la création `OneToOne` entre les deux modèles, lorsqu'on lance `py manage.py makemigrations`, on a l'erreur suivante :

```

○ (app) audrey@LEFREREDUCOUZIN:~/Documents/django/TP/tp6/my_first_django_project$ py manage.py makemigrations
It is impossible to add a non-nullable field 'user' to eleve without specifying 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) Quit and manually define a default value in models.py.
Select an option: █

```

Ensuite on corrige le modèle en rendant l'attribut **user** nullable :

```
1 class Personne(models.Model):
2     nom = models.CharField(max_length=50)
3     prenom = models.CharField(max_length=50)
4     sexe = models.CharField(choices=(("M", "Masculin"), ("F", "Feminin")), max_length=20)
5     date_naissance = models.DateField()
6     user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
7
8     class Meta:
9         abstract = True
```

7. Nous allons maintenant créer une instance de **User** pour chaque personne que nous avons créé :

Select user to change

Search

Action: Go 0 of 9 selected

<input type="checkbox"/>	USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
<input type="checkbox"/>	aa				✗
<input checked="" type="checkbox"/>	audrey				✓
<input type="checkbox"/>	az				✗
<input type="checkbox"/>	bt				✗
<input type="checkbox"/>	ep				✗
<input type="checkbox"/>	ka				✗
<input type="checkbox"/>	sa				✗
<input type="checkbox"/>	sc				✗
<input type="checkbox"/>	td				✗

8. Nous allons maintenant enlever la propriété nullable sur l'attribut précédemment ajouter. Lorsqu'on essaie de relancer les migration, on a le message suivant au terminal :

```
o (app) audrey@LEFREREDUCOUZIN:~/Documents/django/TP/tp6/my_first_django_project$ py manage.py makemigrations
It is impossible to add a non-nullable field 'user' to eleve without specifying 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) Quit and manually define a default value in models.py.
Select an option: 1
```

Nous allons ignorer en choisissant l'option 3, puis nous allons aller dans le site administrateur et assigner les user aux différentes instances de personnes créées. L'étape suivante va consister à supprimer tout les fichiers présents dans le répertoire `/notes/migrations` sauf le fichier `__init__.py`. On relance la commande `py manage.py makemigrations` et puis `py manage.py migrate`.

9. Pour l'instant dans mon application, il n'y a que la vue `add_note` qui est restreinte à quelques utilisateurs. Nous allons donc faire en sorte qu'un utilisateur non authentifié et qui n'a pas les permissions requises ne puisse pas y accéder.

```

1 @login_required
2 @permission_required('notes.add_note', raise_exception=True)
3 def add_note(request, eleve_id, matiere_id):
4     eleve = get_object_or_404(Eleve, pk=eleve_id)
5     matiere = get_object_or_404(Matiere, pk=matiere_id)
6     form = NoteForm()
7
8     if request.method == "POST":
9         form = NoteForm(request.POST)
10        if form.is_valid():
11            valeur = form.cleaned_data.get("valeur")
12            if matiere not in eleve.niveau.matiere_set.all():
13                error_message = f"L'élève {eleve.nom} {eleve.prenom} ne suit pas le cours de matricule {matiere_id}"
14                return HttpResponse(error_message)
15
16            Note.objects.create(valeur=valeur, eleve=eleve, matiere=matiere)
17            return render(request, "eleves/show.html", {"eleve": eleve})
18
19 context = {"matiere": matiere, "eleve": eleve, "form": form}
20 return render(request, "notes/add_note.html", context)

```

Avant de tester cette fonctionnalité, nous allons mettre en place l'authentification des utilisateurs.

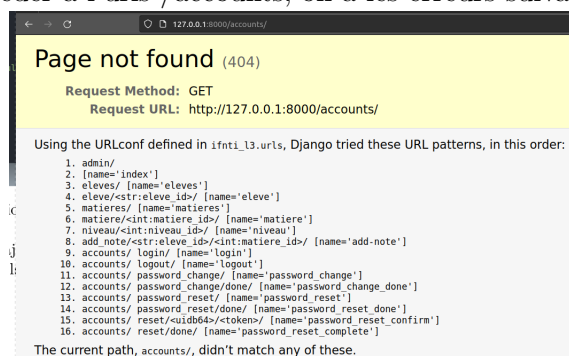
- **Ajout des urls d'auth** Dans le fichier `/ifnti_l3/urls.py`, nous allons ajouter les urls suivant `path('accounts/', include('django.contrib.auth.urls'))`, à l'urlpatterns :

```

1 urlpatterns = [
2     path("admin/", admin.site.urls),
3     path("", include("notes.urls")),
4     path('accounts/', include('django.contrib.auth.urls')),
5 ]

```

Lorsqu'on essaie d'accéder à l'urls `/accounts`, on a les erreurs suivantes :



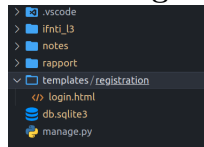
Django a parcouru tout les urls possible et n'a trouver aucune correspondance. Parmi la liste d'urls, on peut remarqué le `/accounts/login`. Essayons d'y accéder :



Cette erreur stipule que l'url existe mes il n'y a pas de template qui y ait associé. Et le template qui est recherché devrait etre `registration/login`

- **Création des templates du template de connexion** Nos allons procéder à la création du template de connexion Tout d'abord, créons le répertoire `/templates/registration` à

la racine du projet. Ensuite créons y le fichier **login.html**



```
1 {% if form.errors %}
2   <p class="text-danger">Votre username et password ne sont pas compatibles</p>
3 {% endif %}
4
5 {% if next %}
6   {% if user.is_authenticated %}
7     <p class="text-danger">Votre compte na pas accès à cette page.</p>
8   {% else %}
9     <p class="text-danger">Svp connectez vous.</p>
10  {% endif %}
11 {% endif %}
12
13 <div class="login-form">
14   <form method="post" action="{% url 'login' %}">
15     {% csrf_token %}
16     <div class="avatar">
17       <i class="material-icons">6#xE7FF;</i>
18     </div>
19     <h4 class="modal-title">Connexion</h4>
20     <div class="form-group">
21       <input type="text" class="form-control" name="username" placeholder="Username" required="required" />
22     </div>
23     <div class="form-group">
24       <input type="password" class="form-control" placeholder="Password" required="required" name="password" />
25     </div>
26     <div class="form-group small clearfix">
27       <label class="checkbox-inline"><input type="checkbox" /> Se rappeler de moi</label>
28       <a href="{% url 'password_reset' %}" class="forgot-link">Mot de passe oublié ?</a>
29     </div>
30     <input type="submit" class="btn btn-primary btn-block btn-lg" value="Login" />
31     <input type="hidden" name="next" value="{{ next }}" />
32   </form>
33   <div class="text-center small">
34     Vous n'avez pas de compte ? <a href="#">Créer un compte</a>
35   </div>
36 </div>
```

Puis dans le fichier de configuration du projet `/ifnti_l3/settings.py` nous allons rendre le répertoire template accessible. Après modification, la liste `TAMPLATE` devrait ressembler à ceci(ne pas oublier d'importer `os`) :

```
1 TEMPLATES = [
2   {
3     'BACKEND': 'django.template.backends.django.DjangoTemplates',
4     # 'DIRS': [os.path.join(BASE_DIR, 'templates')],
5     'DIRS': [],
6     'APP_DIRS': True,
7     'OPTIONS': {
8       'context_processors': [
9         'django.template.context_processors.debug',
10        'django.template.context_processors.request',
11        'django.contrib.auth.context_processors.auth',
12        'django.contrib.messages.context_processors.messages',
13      ],
14    },
15  ],
16 ]
```

Par défaut, après la connexion, django redirige vers l'url `/accounts/profile`. Nous allons changer cela

- **Redirection** Dans le même fichier spécifier l'url de redirection après la connexion : on ajoute ceci au fichier : `LOGIN_REDIRECT_URL = '/'` Maintenant les utilisateurs peuvent s'authentifier.

10. A présent, nous sommes capable de tester les contraintes ajoutées à la vue. Je me suis authentifié en tant qu'élève et voici le résultat quand je tente d'accéder à la vue :

avec un compte enseignant, aucune erreur ne survient.

2 Les problèmes de sécurité

1. Les attaques XSS permettent à un intrus d'injecter des scripts clients dans les navigateurs des utilisateurs. Le principe général est de stocker les scripts malveillants dans la base de données d'où ils seront repris et affichés pour d'autres utilisateurs, ou encore d'amener les utilisateurs à cliquer sur un lien qui va provoquer l'exécution du JavaScript de l'attaquant dans le navigateur des utilisateurs. Cependant, les attaques XSS peuvent provenir de n'importe quelle source de données non fiable, comme les cookies ou les services Web, à chaque fois que les données ne sont pas suffisamment nettoyées avant d'être incluses dans une page. Les gabarits de Django échappent des caractères spécifiques qui sont particulièrement dangereux en HTML. Bien que cela protège les utilisateurs de la plupart des saisies malveillantes, cela ne constitue pas une protection absolue.
2. Les attaques CSRF permettent à une personne malveillante d'exécuter des actions en utilisant les données d'authentification d'un autre utilisateur sans que ce dernier ne s'en rende compte. Django offre une protection intégrée contre la plupart des types d'attaques CSRF, pour autant que vous l'ayez activée et que vous l'utilisiez de manière appropriée. Toutefois, comme pour toute technique de protection, elle est limitée. Par exemple, il est possible de désactiver le module CSRF de manière globale ou pour certaines vues. Vous ne devriez le faire que si vous savez ce que vous faites. Il existe d'autres limites si votre site contient des sous-domaines qui ne sont pas sous votre contrôle.
3. L'injection SQL est un type d'attaque où un utilisateur malveillant est capable d'exécuter du code SQL arbitraire sur une base de données. Il peut en résulter des suppressions d'enregistrements ou des divulgations de données. Les jeux de requête de Django sont prémunis contre les injections SQL car leurs requêtes sont construites à l'aide de la paramétrisation des requêtes. Le code SQL d'une requête est défini séparément de ses paramètres. Comme ceux-ci peuvent provenir de l'utilisateur et donc non sécurisés, leur échappement est assuré par le pilote de base de données sous-jacent.
4. Le détournement de clic est un type d'attaque où un site malveillant intègre un autre site dans un cadre. Cette attaque peut amener un utilisateur à cliquer de manière non désirée pour effectuer des actions non volontaires sur le site ciblé. Django intègre une protection contre le détournement de clic sous la forme de l'intergiciel X-Frame-Options qui, dans un navigateur qui le prend en charge, peut empêcher un site d'être affiché à l'intérieur d'un cadre. Il est possible de désactiver cette protection par vue ou de configurer la valeur exacte de l'en-tête envoyé.
5. Le déploiement de votre site en HTTPS est toujours mieux pour la sécurité. Sans cela, il est possible que des utilisateurs malveillants du réseau interceptent des données d'authentification ou toute autre information transférée entre le client et le serveur et, dans certains cas, pour des attaquants réseau actifs, que des données soient modifiées au passage, dans l'une ou l'autre direction. Si vous souhaitez profiter de la protection ajoutée par HTTPS et l'activer sur votre serveur, il peut être nécessaire de procéder à quelques étapes supplémentaires : Utilisez Sécurité de transport HTTP stricte (HSTS) HSTS est un en-tête HTTP informant le navigateur que toute connexion future à un site particulier devra toujours utiliser HTTPS. Combiné avec la redirection des requêtes HTTP vers HTTPS, cela garantit que les connexions profiteront toujours de la sécurité ajoutée par SSL à partir du moment où une connexion réussie a eu lieu.
6. Django utilise l'en-tête Host fourni par le client pour construire les URL dans certains cas. Même si ces valeurs sont vérifiées pour empêcher les attaques Cross Site Scripting, une valeur Host contrefaite peut être exploitée pour des attaques de type Cross-Site Request Forgery, d'empoisonnement de cache et d'empoisonnement de liens dans les courriels. Comme même

des configurations de serveur Web apparemment sûres sont susceptibles d'accepter des en-têtes Host contrefaits, Django valide les en-têtes Host en les comparant avec le réglage `ALLOWED_HOSTS` dans la méthode `django.http.HttpRequest.get_host()`.

Il n'existe actuellement pas de solution technique à toute épreuve dans l'infrastructure Django pour valider de manière sûre tous les contenus de fichiers envoyés par les utilisateurs. Cependant, il existe un certain nombre de mesures à prendre pour diminuer les risques de telles attaques :

- Une catégorie d'attaques peut être évitée en servant toujours les contenus envoyés par les utilisateurs à partir d'un autre nom de domaine de premier ou de deuxième niveau.
- En plus de cette mesure, les applications peuvent choisir de définir une liste des extensions de fichiers autorisées pour les fichiers téléversés par les utilisateurs et configurer le serveur Web pour qu'il n'accepte de servir que ces fichiers.

Les sous-domaines d'un site peuvent définir des cookies pour le client valables pour tout le domaine. Cela rend possible les attaques par fixation de session si des cookies peuvent être créés par des sous-domaines qui ne sont pas sous contrôle de personnes de confiance.

Par exemple, un attaquant pourrait se connecter à `good.example.com` et obtenir une session valable pour son compte. Si l'attaquant contrôle `bad.example.com`, il peut l'utiliser pour vous envoyer sa clé de session puisqu'un sous-domaine a le droit de définir des cookies pour `*.example.com`. Lorsque vous visitez `good.example.com`, vous serez connecté sous le compte de l'attaquant et vous pourriez saisir involontairement des données personnelles sensibles (par ex. des données de carte de crédit) enregistrées dans le compte de l'attaquant.

Une autre attaque possible apparaît dans le cas où `good.example.com` définit son réglage `SESSION_COOKIE_DOMAIN` à `"example.com"`, ce qui pourrait avoir comme conséquence d'envoyer les cookies de session à `bad.example.com`.

3 Customiser le site d'administration

1. Il est intéressant de modifier le site admin pour les ajouter certaines fonctionnalités, qui n'existe pas par défaut.
2. Il est possible d'utiliser les `ModelAdmin` pour rendre les modèles accessibles dans le site admin
3. Explication des options
 - Attribuez à **`date_hierarchy`** le nom d'un champ `DateField` ou `DateTimeField` de votre modèle, et la page de liste pour modification inclura une navigation par filtres sélectifs basés sur les dates en fonction de ce champ.
 - Définissez **`list_filter`** pour activer des filtres dans la barre de droite de la page de liste pour modification de l'administration.
 - Définissez **`list_display`** pour contrôler quels champs sont affichés sur la page de liste pour modification de l'interface d'administration.
 - Utilisez **`list_display_links`** pour contrôler le cas échéant quels champs de **`list_display`** doivent faire le lien vers la page de modification d'un objet.
 - Définissez **`list_editable`** à une liste de noms de champs de modèle afin de permettre l'édition de ces champs sur la page de liste pour modification. C'est-à-dire que les champs répertoriés dans **`list_editable`** seront affichés sous forme de composants de formulaire sur la page de liste pour modification, permettant aux utilisateurs de modifier et d'enregistrer plusieurs lignes à la fois.
 - Utilisez l'option **`fields`** si vous avez besoin de réaliser de simples changements dans la disposition des formulaires des pages « ajouter » et « modifier », comme afficher uniquement un sous-ensemble des champs disponibles, modifier leur ordre ou les regrouper en lignes.
 - Cet attribut, si fourni, doit être une liste de noms de champs à exclure du formulaire.
 - **`raw_id_fields`** est une liste de champs que vous souhaitez modifier par un simple composant `Input`, que ce soit un champ `ForeignKey` ou `ManyToManyField`.
 - Définissez **`show_full_result_count`** pour définir si le nombre total d'objets doit être affiché sur une page d'administration filtrée (par exemple 99 résultats (103 au total)). Si

cette option est définie à `False`, un texte du style 99 résultats (tout afficher) s’affiche à la place.

- Si vous souhaitez désactiver le tri pour certaines colonnes, définissez **sortable_by** à une collection (par ex. list, tuple ou set) sous-ensemble de **list_display** qui sont susceptibles d’être triés. Une collection vide désactive le tri pour toutes les colonnes.
- Par défaut, un champ `ManyToManyField` est affiché dans le site d’administration avec un `<select multiple>`. Cependant, les boîtes à sélection multiple peuvent être difficiles à utiliser lors de la sélection de nombreux éléments. En ajoutant un champ `ManyToManyField` à cette liste, cette boîte sera remplacée par une discrète et astucieuse interface de « filtre » JavaScript permettant de rechercher dans les options. Les options non sélectionnés et sélectionnés apparaissent dans deux boîtes côte à côte. Voir **filter_vertical** pour utiliser une interface verticale.
- **filter_vertical** Identique à **filter_horizontal**, mais utilise un affichage vertical de l’interface de filtre avec la boîte d’options non sélectionnées apparaissant au-dessus de la boîte d’options sélectionnées.
- La chaîne à utiliser pour l’affichage de valeurs vides dans la liste pour modification du site d’administration. Il s’agit par défaut d’un tiret. La valeur peut aussi être surchargée sur la base des classes `ModelAdmin` ou sur un champ précis d’une classe `ModelAdmin` en définissant un attribut **empty_value_display** pour le champ. Voir **ModelAdmin.empty_value_display** pour des exemples.
- Définissez **list_per_page** pour contrôler le nombre d’éléments paginés apparaissant sur chaque page de la liste pour modification de l’administration. Par défaut, cette option est définie à 100.
- Définissez **save_as** pour activer la fonctionnalité « enregistrer comme nouveau » des formulaires d’édition de l’administration.
- Lorsque **save_as**=`True`, la redirection par défaut après l’enregistrement du nouvel objet se fait vers la vue pour modification de cet objet. Si vous définissez **save_as_continue**=`False`, la redirection se fera vers la vue de liste pour modification.
Par défaut, **save_as_continue** est défini à `True`.
- Si **readonly_fields** est utilisé sans ordre explicite défini par **ModelAdmin.fields** ou **ModelAdmin.fieldsets**, ils seront ajoutés en dernier après tous les champs modifiables.
Un champ en lecture seule n’est pas limité à l’affichage des données d’un champ de modèle, il peut également afficher le résultat d’une méthode de modèle ou d’une méthode de la classe **ModelAdmin**. C’est très similaire à la façon dont **ModelAdmin.list_display** fonctionne. Cela fournit un moyen pour faire présenter à l’interface d’administration des informations sur l’état des objets en cours d’édition.
- Définissez **save_on_top** pour ajouter les boutons d’enregistrement au sommet des formulaires d’édition de l’administration.
Normalement, les boutons d’enregistrement apparaissent uniquement au bas des formulaires. Si vous définissez **save_on_top**, les boutons apparaîtront à la fois en haut et en bas.
Par défaut, **save_on_top** est défini à `False`.
- Définissez **search_fields** pour activer une boîte de recherche sur la page de liste pour édition de l’administration. Cet attribut doit contenir une liste de noms de champs qui seront recherchés chaque fois que quelqu’un soumet une requête de recherche dans cette zone de texte.
- Définissez **view_on_site** pour contrôler l’affichage du lien « Voir sur le site ». Ce lien est censé diriger vers une URL où l’objet enregistré peut être affiché.

4. Nous allons tester une option, `readonly_fields`.

A screenshot of a Django admin form for a student record. The form is titled "AMONA" and contains the following fields:

- Date naissance: 2023-11-06 (with a calendar icon and "Today" label)
- User: aa (with a dropdown arrow, edit icon, and add icon)
- Id: AMO20231127163133490959
- Niveau: L3 (with a dropdown arrow, edit icon, and add icon)
- Sexe: Masculin
- Nom: AMONA
- Prenom: AUdrey

On remarque que le nom, le prenom et le sexe de l'eleve ne sont plus modifiable

5. Il est possible de modifier les templates par défaut du site adonistrateur, en créant un dossier **admin** dans un répertoire **template** qui est enregistré dans le settings.py. Ensuite il faudra copier les templates que l'on souhaite modifier depuis le répertoire **/django/contrib/admin/templates/admin** vers le répertoire créé. Il est possible de modifier les templates admin juste pour un modèles donné, dans ce cas il faudra ajouter dans le répertoire admin créé un répertoire portant le nom du mod-le mais tout en minuscule. ensuite la copie est faite vers ce répertoire.
6. Il est possible d'inclure des fichiers js et css personnalisé en les ajoutant dans le répertoire **/django/contrib/admin/static/admin/css/js** et en les ajoutant au html par le base.html des templates admins
7. Par défaut, l'entête des pages du site d'administration de Django est généré par le fichier de template admin/base_site.html. Ce template définit la structure de base de l'interface d'administration, y compris l'entête.