Etude de cas: Dashboard Admin

Enoncé

L'objectif de cet atelier est de développer une application web de gestion des évènements en utilisant le Framework Django pour les deux parties FrontOffice et BackOffice. Cette application est représentée par le diagramme de classe d'analyse suivant :

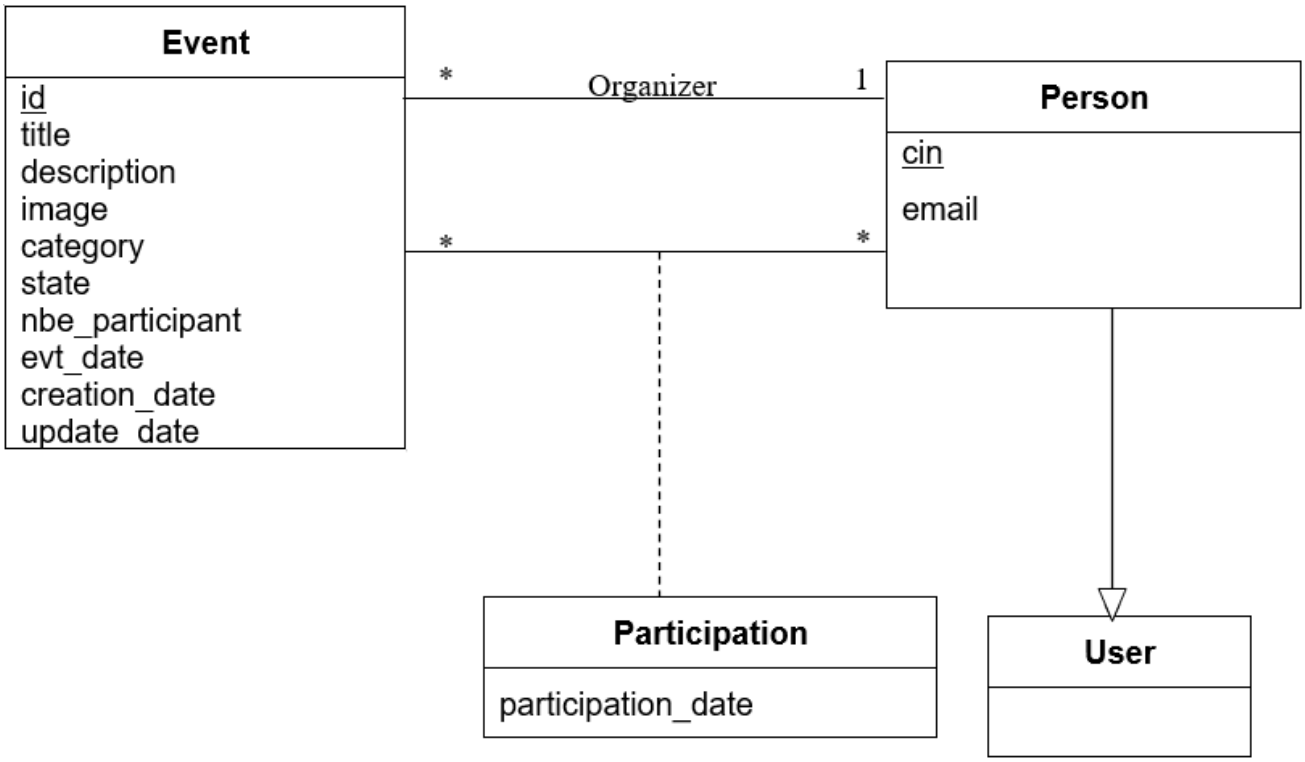


Figure: Diagramme de classe d'analyse

I. Mise en place du projet

Pour la mise en place du projet on besoin:

- Installer Python 3.11
- Installer Django 4.1
- Créer un environnement virtuel nommé « **venv** »
- Créer un projet nommé « **GestionEvenement** »
- Créer deux applications : **Event** et **Person**

II. Génération de la BD

1.Développer les entités dans le fichier « models.py » en ce basant sur le diagramme de classe ci-dessus et en appliquant les contraintes suivantes :

=>Le modèle « **Person** » hérite de l'entité « **User** » prédéfinit. Il faut changer la clé primaire « id » à « cin » qui doit avoir une longueur fixe égale à 8.

=>Le modèle « **Event** » doit respecter les contraintes suivantes :

***Description** doit être de type « TextField ».

***Image** doit être de type « ImageField », il faut installer la libraire **Pillow** via cette commande :

```
pip install Pillow
```

***State** (bool, default=False)

***Email** doit se terminer par "@esprit.tn ".

***Category** doit représenter une liste de choix comme suit:

-Musique

-Cinema

-Sport

*Un événement est organisé par une seule personne et une personne peut organiser plusieurs événements.

*Une personne peut participer à plusieurs évènements. Un évènement peut avoir plusieurs participants.

*Déclarer la classe **Meta** dans laquelle on définit la contrainte sur l'attribut « **evt_date** > à date now» **.**

=>Le modèle « **Participation** » est comme suit :

*L'attribut « **participation_date** » doit avoir comme valeur par défaut la **date système**.

*Une personne ne peut pas participer plusieurs fois au même évènement.

2.Modifier le fichier **settings.py** pour que le nom de la BD porte le nom de la classe dont vous faites partie.

3. Générer les fichiers de migration via la commande

```
python manage.py makemigrations NomApp
```

4.Générer le schéma de la BD via la commande

```
python manage.py migrate
```

III. Application Back-Office

1.Créer un nouvel utilisateur de type admin pour accéder au Dashboard admin via cette commande :

```
python manage.py createsuperuser
```

2.Dans le fichier « **admin.py** » de l'application « **Person** » apportez les modifications suivantes :

a. Inscrivez le modèle « Person » au site d'administration comme suit :

```
admin.site.register(Person)
```

b. Définissez la classe **ResearchPerson** qui permet de faire la recherche d'une personne selon son **username : search_fields=['username']**.

3.Dans le fichier « **admin.py** » de l'application « **Event** » apportez les modifications suivantes :

a. Inscrivez le modèle « **Event** » au site d'administration ;

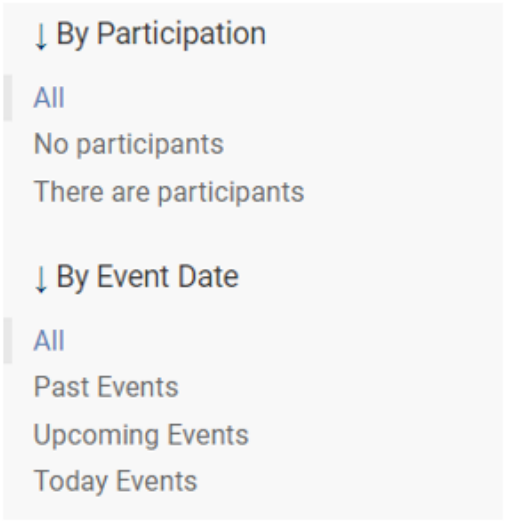
b. Créer la classe **EventAdmin**.

c. Affichez la liste des évènements comme suit:

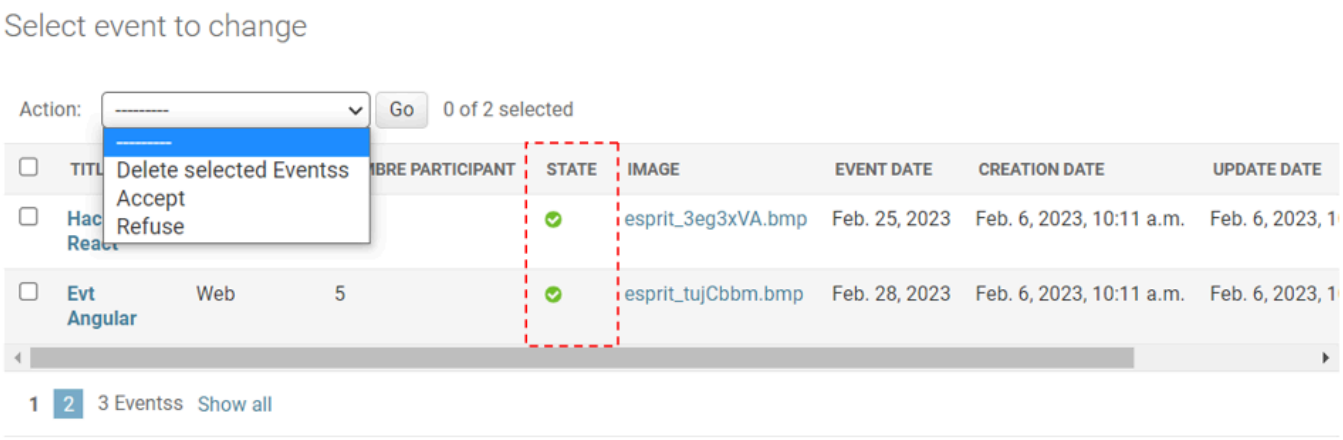
i. Modifiez l'affichage de la liste des évènements via la fonction **__str__( )** dans le fichier models.py. Le site d'administration affichera une seule colonne avec la représentation __str__() de l'objet Event.
ii. Définissez et personnaliser la variable **list_display** à fin de contrôler quels champs seront affichés sur la page de la liste de l'interface d'administration.
iii. Ajouter une fonction qui permet d'afficher le nombre total des participants dans chaque évènement.
iv. Ajoutez la caractéristique **readonly_fields** pour les deux attributs **creation_date** et **update_date** pour qu'on puisse les inclure dans la variable **list_diplay**.
v. Activez la pagination sur le tableau de la liste des évènements via list_per_page.

d. Activez les filtres dans la barre latérale droite de la page d'affichage de la liste des évènements avec list_filter :

i. Un filtre selon le titre de l'évènement.
ii. Un filtre selon le nombre de participants avec deux catégories : « No participants » et « There are participants ».
iii. Un filtre selon la date de l'évènement selon trois critères : « Past Events », «Upcoming Events » et « Today Events ».



e. Ajouter deux actions dans la liste des évènements qui permettent de rendre l'état d'un évènement « accepté » ou « refusé » comme le montre la figure ci-dessous :



f. Modifiez le formulaire de gestion d'Event :

i. Définissez **fieldsets** qui est une liste de tuples binaires pour contrôler la mise en page des pages d'administration « ajouter » et « modifier ». Chaque tuple représente une balise <fieldset> dans la page du formulaire d'administration.

```
fieldsets = (
        ('A propos', {"fields": ('title','description','image'),}),

        ('Date',{"fields":('event_date','creation_date','update_date')
}),
('Others',{"fields":('category','state','nbe_participant') }),

       ('Personal',{"fields":('organizer',) }))
```

ii. Définissez **inlines** pour créer une instance de la classe « **participation** » qui permet d'ajouter les participants pour chaque évènement. Il faut donc créer la classe **ParticipationAdmin** qui sera de type **InlineModelAdmin** et tester les deux sous classes :

- TabularInlin (https://docs.djangoproject.com/fr/4.1/ref/contrib/admin/#django.contrib.admin.TabularIn

line)
- StackedInline (https://docs.djangoproject.com/fr/4.1/ref/contrib/admin/#django.contrib.admin.StackedInline)

NB : inlines prendra comme valeur le nom de la classe **ParticipationAdmin.**

iii. Appliquez **l'auto-complete** lorsque vous cherchez une personne lors de la sélection de l'organisateur d'un évènement dans le champ « organizer » : **autocomplete_fields=['organizer']**