Examen 1:

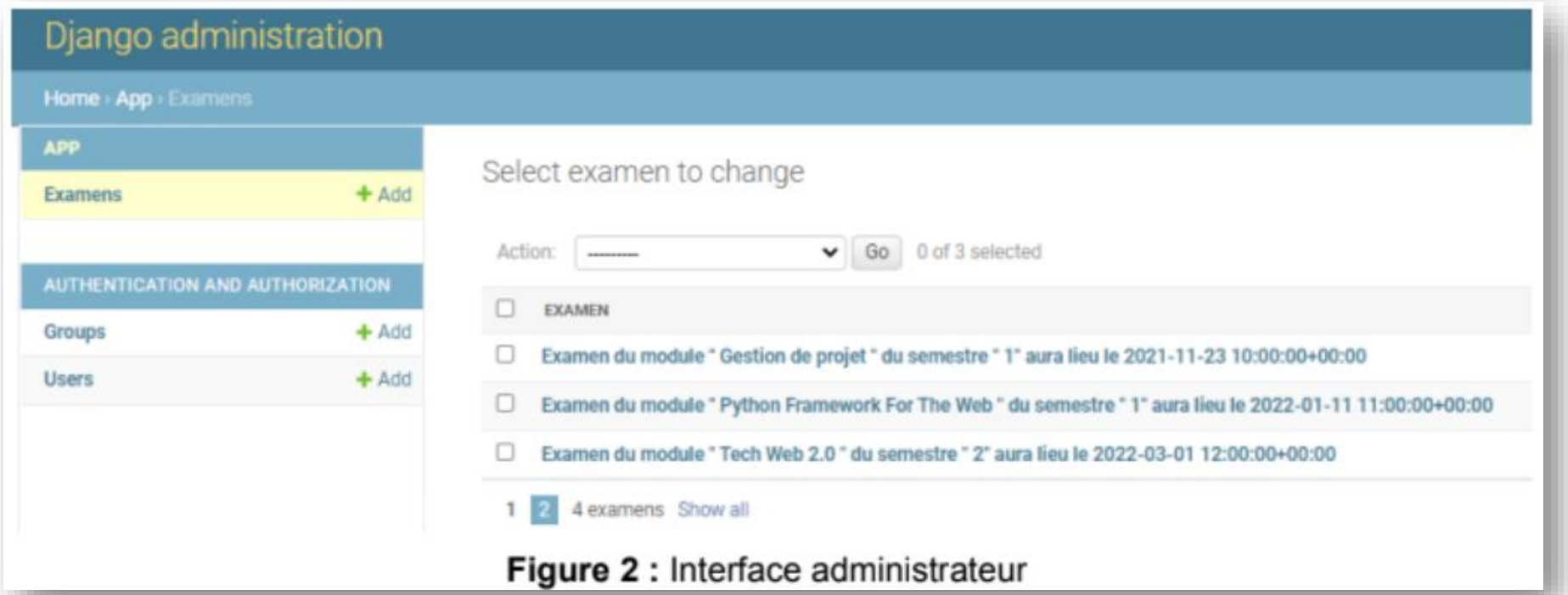

Figure 2 : Interface administrateur

models.py

```
a-(models.Model):
    nom module = models.CharField(max length=30)
b-  date = models.DateTimeField()
    semestre = models.IntegerField(default=False)
c-  def __str__(self):
        return f'Examen du module " {self.nom_module} " ' \
            f'du semestre " {self.semestre}" aura lieu le {self.date} '
```

admin.py

```
from django.contrib import admin
from App.models import *

…@admin.register(Examen)….(Examen) (1pt)
class ExamenAdmin(admin.ModelAdmin):
    ……list_per_page = 3 …… (1pt)
    ……ordering = ['date']…… (0.5pt)
```

2- Dans la partie FrontOffice on désire afficher la liste des examens dans un tableau et ordonné selon le nom du module. Pour ce faire, compléter la classe **views.py**. (**2.5pts**)

views.py

```
from django.views.generic import ..... a-ListView..... (0.5pt)
from App.models import Examen
class Affiche_Examen(..a-ListView.....):
    model=Examen
  -ordering=['-nom_module']
template_name='App/Affiche_Examen.html'
context_object_name = ....examen.. (0.75pt)
```

Affiche_Examen.html

```
<table>
    <tr><th>Module</th><th>Date</th><th>Semestre</th></tr>
    {% for e in examen %}
        <tr>
            <td>..{{e.nom_module}} /td> (0.25pt)
            <td>....{{e.date}}.....</td>(0.25pt)
            <td>..{{e.semestre}}.</td>(0.25pt)
             </tr>
    {% endfor %}
</table>
```

3- Maintenant on veut ajouter un « **Examen** » dans la BD. Une redirection vers la page d'affichage sera faite après l'exécution de l'ajout. Soit le fichier **urls.py. (3.5pts)**

urls.py

```
from django.urls import path
from App.views import Affiche_Examen, Ajout

urlpatterns=[

path('examen/ListView/',Affiche_Examen.as_view(),name='LV'),
path('examen/create/',Ajout.as_view(), name='Add'),
]
```

NB : Le formulaire d'ajout doit être affiché sous forme d'une liste

views.py

```
class Add(..... CreateView .....) :  (0.25pt)
    model = Projet
    form_class = ProjetForm
    template_name = "Ajout.html"
    success_url = ... reverse_lazy('LV') .....  (0.75pt)
```

Ajout.html

```
<form method="post" action="{% url "Add" %}..."> (1pt)
    {% csrf_token %}.. (0.5pt)
    {{ form.as_ul }}.. (1pt)
    <input type="submit" value="Ajout">

</form>
```

Etude de cas:

Inscrivez le modèle « Event » au site d'administration : `@admin.register(Person)` ou `admin.site.register(Person)`

Définissez la classe ResearchPerson qui permet de faire la recherche d'une personne selon son username :

```
search_fields = ['username']
```

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.

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

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.

```
list_display=('title', 'description' ,'image', 'evt_date','category','state', 'created_date','updated_date'
,'organisateur','event_nbr_participant')
```

Ajouter une fonction qui permet d'afficher le nombre total des participants dans Chaque évènement.

```
def event_nbr_participant(self,obj):
    count = obj.participant.count()
    return count
event_nbr_participant.short_description = 'Nombre des
participants'
```

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.

```
readonly_fields=('created_date','updated_date')
```

Activez la pagination sur le tableau de la liste des évènements via list_per_page.

```
list_per_page = 2
```

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

Un filtre selon le titre de l'évènement. **search_fields = ['title']**

Un filtre selon le nombre de participants avec deux catégories : « Noparticipants » et « There are participants ».

```
class ParticipantFilter(admin.SimpleListFilter):
    #esm el filter
    title = 'Participants'
    parameter_name = 'participants'
    def lookups(self, request, model_admin):
        return (
            ('no', 'No Participants'),
            ('yes', 'There are Participants'),
        )
    def queryset(self, request, queryset):
        if self.value() == 'no':
            return queryset.filter(participant__isnull=True)
        if self.value() == 'yes':
            return queryset.filter(participant__isnull=False)
```

Un filtre selon la date de l'évènement selon trois critères : « Past Events », «Upcoming Events » et « Today Events ».

```
class EventDateFilter(admin.SimpleListFilter):
    #esm el filter
    title = 'Event DATE'

    parameter_name = 'evt_date'

    def lookups(self, request, model_admin):
        return (
            ('past', _('Past events')),
            ('today', _('Today events')),
            ('future', _('Upcoming events')),
        )

    def queryset(self, request, queryset):
        if self.value() == 'past':
            return queryset.filter(evt_date__lt=date.today())
        elif self.value() == 'today':
            return queryset.filter(evt_date=date.today())
        elif self.value() == 'future':
            return queryset.filter(evt_date__gt=date.today())
```

II. Application Front Office (CRUD)

Créer la fonction « ListEvt » qui permet d'afficher la liste des évènements avec 2 méthodes :

a. En utilisant la queryset : Event.objects.all()

```
def ListEvt(request):
    events = Event.objects.all()
    return render(request, '../Template/event/list_eventQueryset.html', {'events': events})


path('events/', ListEvt, name='list_eventQueryset'),
```

b. En utilisant la classe générique : ListView

La liste des évènements est triès selon
la date de l'evènement.

```
class ListEventsFalse( ListView):

        model = Event
        template_name = "event/list_event.html"

        context_object_name = "events"  # par défaut object_list

        def get_queryset(self):
            eventsFalse = Event.objects.filter(state=False).order_by('evt_date')
            return eventsFalse
```

Créer la fonction « DetailEvt » qui permet d'afficher les détails d'un évènement lors de la clique sur le bouton de l'évènement en question.

```
def eventDetails(req, id):
    user = req.user
    event = Event.objects.get(id=id)
    if user:
        participant = participants.objects.filter(
            person=user, evenement=event)
        if participant:
            button_disabled = True
        else:
            button_disabled = False
    return render(req, "event/details.html", {'e': event, "button":
button_disabled})
```

Développer la fonction qui permet d'ajouter un evènement « AddEvt ».

```python
class AddEvent(CreateView):

    template_name = "event/addEvent.html"
    model = Event
    form_class = EvenementForm
    success_url = reverse_lazy('Affiche')
```

```python
def AddEv(req):
    form = EvenementForm()
    if req.method == 'POST':
        form = EvenementForm(req.POST,
req.FILES)


        if form.is_valid():
            print(form.instance)
            form.instance.organisateur=Person.o
bjects.get(cin=req.user.cin)
            # print(form.instance.organisateur)
            form.save()
            return redirect('Affiche')
    return render(req, 'event/addEvent.html',
{'form': form})
```

Dans le tableau d'évènement, ajouter un lien qui permet de modifier un évènement« UpdateEvt ».

```
<a href="{% url 'ModifierEvenement' p.id %}" class="btn btn-warning">update</a>
```

Dans le tableau d'évènement, ajouter un lien qui permet de supprimer un évènement

```
<a href="{% url 'deleteEvent' p.id  %}" class="btn btn-danger">delete</a>
```

Dans la liste des évènements, ajouter le bouton « participer ». En cliquant dessous :

- **Le nombre de participant s'incrémente ;**
- **Le bouton participer devient disabled.**
- **Un bouton annuler participation sera affiché.**

```
def participer(req, event_id):
    user = req.user

    event = Event.objects.get(id=event_id)

    participant = participants.objects.create(
        person=user, evenement=event)
    participant.save()
    event.nbr_participant += 1
    event.save()

    return redirect('Affiche')
```

```
{% if button %}

<button class="btn btn-primary">
  <a class="btn btn-primary" href="{% url 'participer' e.id %}">
    Participer</a
  >
</button>

<button class="btn btn-danger">
  <a class="btn btn-danger" href="{% url 'cancel' e.id %}">Cancel</a>
</button>

{% else %}
<button class="btn btn-primary">
  <a class="btn btn-primary" href="{% url 'participer' e.id %}">
    Participer</a
  >
</button>

<button class="btn btn-danger" disabled>
  <a class="btn btn-danger" href="{% url 'cancel' e.id %}">Cancel</a>
</button>
{% endif %}
```

Login/logout:

Vérifier l'accès à l'interface d'affichage de la liste des évènements. Une personne ne peut y accéder sans qu'elle se connecte à l'application.

```
@login_required
def list_event(request):

    list =
Event.objects.filter(state=True).order_by('evt_date
')

    Nbr = Event.objects.count()

    return render(request, 'event/list_event.html',
{'events': list})
```

```
class ListEvents(LoginRequiredMixin ,ListView):

    model = Event
    template_name = "event/list_event.html"

    context_object_name = "events"  # par défaut
object_list
    login_url = 'login'
    def get_queryset(self):
        eventsTrue =
Event.objects.filter(state=True).order_by('evt_date')
        return eventsTrue
```

Rest api: (voir workshop)