La mission était de ***moderniser le système de gestion interne de la médiathèque.***

Pour cela, j'ai mis en place une interface (avec pour langage Python et environnement de développement intégré PyCharm) médiathèque comprenant trois applications :

- Admin
- Staff
- Client

L'application ***Admin*** permet une gestion globale de l'interface. Il y a eu la mise en place des rôles de chacun, via un système de modèles et décorateurs (Cf. photos ci-dessous).

Models →

```python
from django.contrib.auth.models import AbstractUser
from django.db import models
from .managers import CustomUserManager


class CustomUser(AbstractUser):  Ad15C
    ADMIN = 'admin'
    STAFF = 'staff'
    CLIENT = 'client'

    ROLE_CHOICES = [
        (ADMIN, 'Admin'),
        (STAFF, 'Staff'),
        (CLIENT, 'Client'),
    ]

    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default=CLIENT)

    objects = CustomUserManager()  # Utilisation du manager personnalisé

    def __str__(self):  Ad15C
        return self.username

    # Propriétés pour déterminer le rôle de l'utilisateur
    @property  1 usage (1 dynamic)  Ad15C
    def is_staff_user(self):
        return self.role == self.STAFF

    @property  Ad15C
    def is_client_user(self):
        return self.role == self.CLIENT

    @property  Ad15C
    def is_admin_user(self):
        return self.role == self.ADMIN
```

```python
from django.http import HttpResponseForbidden
from functools import wraps


def role_required(*roles):  4 usages  Ad15C
    """
    Autorise uniquement les utilisateurs appartenant aux groupes donnés dans roles.
    """
    def decorator(view_func):  Ad15C
        @wraps(view_func)  Ad15C
        def _wrapped_view(request, *args, **kwargs):
            if not request.user.is_authenticated:
                return HttpResponseForbidden("Utilisateur non authentifié.")
            if str(request.user.role) not in [str(role) for role in roles]:
                return HttpResponseForbidden("Accès limité aux utilisateurs autorisés.")
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

Decorators

Le fait de mettre en place des rôles, permet de respecter la volonté de la médiathèque quant aux autorisations accordées.

L'application permet donc d'arriver sur la page d'accueil, où l'utilisateur, quel qu'il soit, pourra se connecter/déconnecter à son espace, s'inscrire.(Cf. Photo ci-dessous).

Et on voit bien le système de connexion et déconnexion avec les vues ci-dessous :

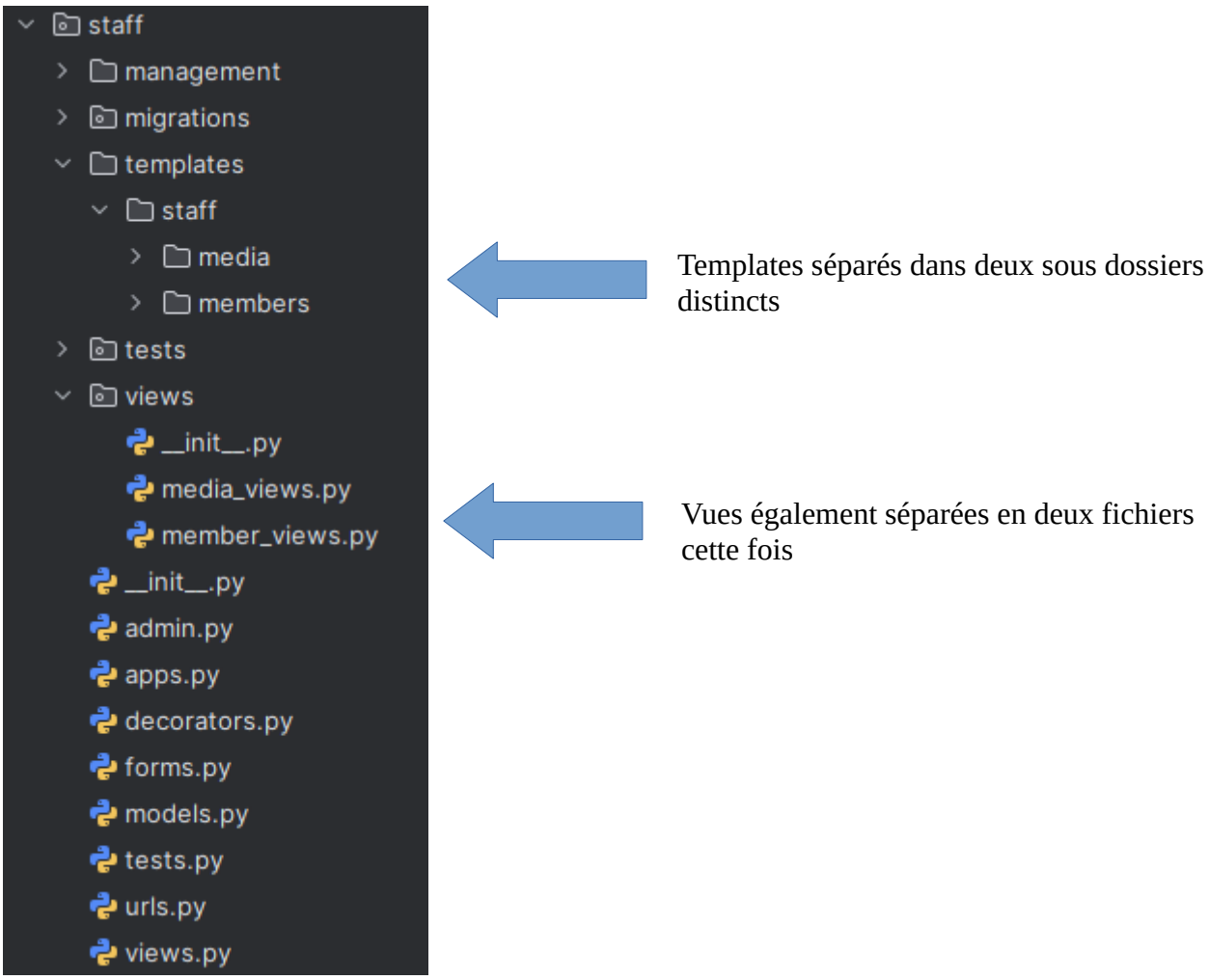
```python
def login_view(request):  1 usage  Ad15C
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
            user = authenticate(request, username=username, password=password)
            if user is not None:
                login(request, user)  # Connecter l'utilisateur
                messages.success(request, message: "Vous êtes connecté avec succès.")

                # Redirection vers la page appropriée en fonction du rôle de l'utilisateur
                if user.role == User.CLIENT:
                    return redirect('client:espace_client')
                elif user.role == User.STAFF:
                    return redirect('authentification:espace_staff')
                elif user.role == User.ADMIN:
                    return redirect('/admin/')

                # Si aucun rôle ne correspond, on redirige vers la page d'accueil par défaut
                return redirect('/')

            else:
                messages.error(request, message: "Nom d'utilisateur ou mot de passe incorrect.")
    else:
        form = LoginForm()

    # Retourne le formulaire de connexion
    return render(request, template_name: 'authentification/login.html', context: {'form': form})


def logout_view(request):  1 usage  Ad15C
    logout(request)
    messages.success(request, message: "Vous êtes maintenant déconnecté.")
    return redirect('authentification:connexion')
```

En ce qui concerne l'application ***Staff***, cette dernière va permettre au personnel de la médiathèque de gérer les aspects suivants :

- Affichage, Ajout, Emprunt et Retour d'un média.
- Création, Modification et Mise à jour d'un membre.

Dans un soucis d'optimisation et de clarté, j'ai décidé de séparer à chaque fois la partie média et membres.

Il y a eu la mise en place d'une relation Parent par rapport aux médias (Cf. photo ci dessous)



Il était également question de respecter les règles d'emprunts et j'ai également mis en place un système de relation parent/enfant, permettant de partager des champs communs entre les médias. Ici, la classe MediaStaff a le rôle de « parent » , alors que BookStaff, CDStaff, DVDStaff et BoardGameStaff ont ceux des enfants. J'ai pris parti pris de mettre les jeux de plateaux en tant qu'enfant, car même si ce dernier n'est pas empruntable, il n'en reste pas moins visible de la part de tous.

Classe Parent

```python
class MediaStaff(models.Model):  Ad15C
    name = models.CharField(max_length=200)
    media_type = models.CharField(max_length=50)
    is_available = models.BooleanField(default=True)
    can_borrow = models.BooleanField(default=True)
    description = models.TextField(blank=True, null=True)

    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField(null=True, blank=True)
    content_object = GenericForeignKey( ct_field: 'content_type',  fk_field: 'object_id')

    MEDIA_TYPE = None  # Définie dans les sous-classes

    def __str__(self):  Ad15C
        return self.name

    def save(self, *args, **kwargs):  Ad15C
        is_new = self.pk is None

        if not self.media_type and self.MEDIA_TYPE:
            self.media_type = self.MEDIA_TYPE

        if not self.content_type_id:
            self.content_type = ContentType.objects.get_for_model(self.__class__)

        super().save( *args: *args, **kwargs)

        if is_new and not self.object_id:
            self.object_id = self.id
            super().save(update_fields=['object_id'])
```

Règles d'emprunt

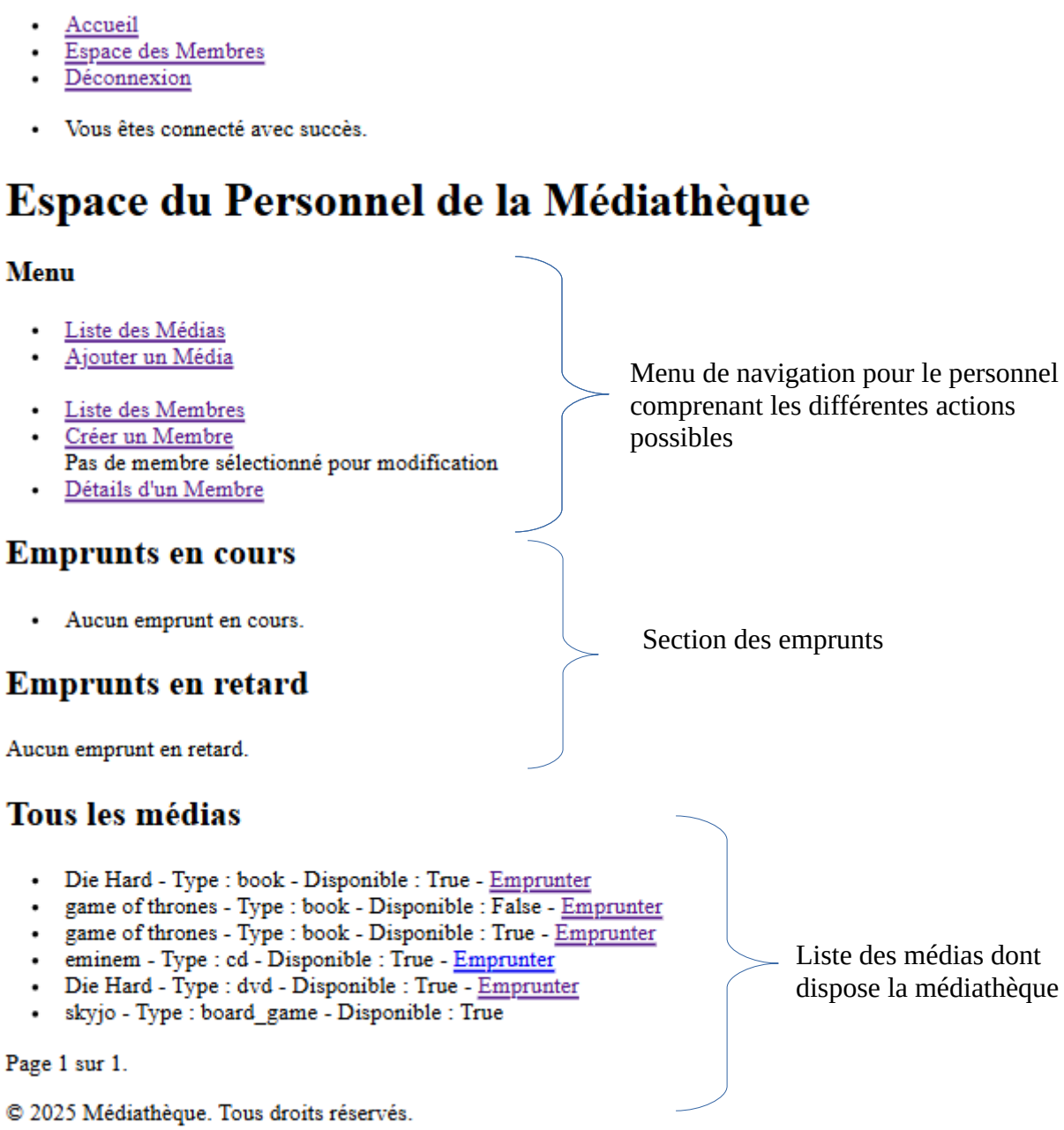
```python
    def is_borrowable_by(self, user):  2 usages (1 dynamic)  Ad15C
        if not self.is_available or not self.can_borrow:
            return False
        if StaffBorrowItem.objects.filter(media=self, is_returned=False).exists():
            return False
        if user.role not in [User.CLIENT, User.STAFF]:
            return False
        return True

    @classmethod  1 usage  Ad15C
    def get_borrowable_by(cls, user):
        return cls.objects.filter(is_available=True, can_borrow=True).exclude(
            staff_borrows_media__is_returned=False
        )

    class Meta:  Ad15C
        permissions = [
            ("can_view_media", "Peut voir les détails d'un média"),
        ]
        verbose_name = "Média"
        verbose_name_plural = "Médias"
```

Classes Enfants

Jeux de Plateaux visibles seulement

```python
class BookStaff(MediaStaff):  Ad15C
    author = models.CharField(max_length=200)
    MEDIA_TYPE = 'book'


class DVDStaff(MediaStaff):  Ad15C
    producer = models.CharField(max_length=200)
    MEDIA_TYPE = 'dvd'


class CDStaff(MediaStaff):  Ad15C
    artist = models.CharField(max_length=200)
    MEDIA_TYPE = 'cd'


class BoardGameStaff(MediaStaff):  Ad15C
    creators = models.CharField(max_length=100)
    is_visible = models.BooleanField(default=True)
    game_type = models.CharField(max_length=100, blank=True, null=True)
    MEDIA_TYPE = 'board_game'

    def __str__(self):  Ad15C
        return self.name

    def toggle_availability(self):  Ad15C
        if not StaffBorrowItem.objects.filter(media=self, is_returned=False).exists():
            self.is_visible = not self.is_visible
            self.save(update_fields=['is_visible'])

    def is_borrowable_by(self, user):  1 usage (1 dynamic)  Ad15C
        return False  # Les jeux de société ne peuvent pas être empruntés

    def save(self, *args, **kwargs):  Ad15C
        self.can_borrow = False  # Jamais empruntable
        super().save( *args: *args, **kwargs)
```

Ainsi lorsque l’utilisateur s’est connecté à son espace staff, il arrive sur la page suivante.



On voit bien les différentes actions que peut effectuer l’utilisateur. Si par exemple, ce dernier souhaite procéder à l’emprunt d’un média, voilà la vue qui a été mise en place.

```python
@login_required  1 usage  Ad15C *
@role_required(User.STAFF)
@permission_required( perm: 'authentification.can_borrow_media', raise_exception=True)
def borrow_media(request, pk):
    try:
        # Vérifie si l'utilisateur a la permission d'emprunter un média
        if not request.user.has_perm('authentification.can_borrow_media'):
            raise PermissionDenied

        # Vérifie si c'est un jeu de société
        try:
            media = BoardGameStaff.objects.get(pk=pk)
        except BoardGameStaff.DoesNotExist:
            media = get_object_or_404(MediaStaff, pk=pk)

        if isinstance(media, BoardGameStaff):
            messages.error(request,  message: "Les jeux de société ne peuvent pas être empruntés.")
            return redirect('staff:media_liste')

        # Vérifie si l'utilisateur a des emprunts en retard ou a dépassé le nombre maximum d'emprunts
        result = check_borrowing_conditions(request, request.user, media)
        if result:
            return result

        # Calcule la date limite de retour
        due_date = timezone.now() + timezone.timedelta(days=7)

        # Utilise le formulaire d'emprunt
        if request.method == 'POST':
            form = BorrowMediaForm( *args: request.POST, user=request.user)
            if form.is_valid():
                with transaction.atomic():
                    due_date = form.cleaned_data['due_date']
                    borrow_item = StaffBorrowItem.objects.create(
                        user=request.user,
                        media=media,
                        borrow_date=timezone.now(),
                        due_date=due_date
                    )
                    media.is_available = False
                    media.save()
```

```
                messages.success(request, message: "Média emprunté avec succès !")
                return redirect( to: 'staff:succes_emprunt', pk=borrow_item.pk)


        else:
            # Formulaire pré-rempli avec les informations du média
            form = BorrowMediaForm(initial={'media': media, 'due_date': due_date})

        # Retour page de confirmation avec le formulaire
        return render(request, template_name: 'staff/media/borrow_confirm.html', context: {
            'form': form,
            'media': media,
            'due_date': due_date,
        })

    except PermissionDenied:
        messages.error(request, message: "Vous n\'avez pas la permission d\'emprunter ce média.")
        return redirect('staff:espace_staff')


def has_overdue_borrowings(user):  1 usage  Ad15C
    """Vérifie si l'utilisateur a des emprunts en retard"""
    return StaffBorrowItem.objects.filter(user=user, is_returned=False, due_date__lt=timezone.now()).exists()


def has_max_active_borrows(user):  1 usage  Ad15C
    """Vérifie si l'utilisateur a atteint la limite de 3 emprunts actifs"""
    return StaffBorrowItem.objects.filter(user=user, return_date__isnull=True).count() >= 3


def check_borrowing_conditions(request, user, media):  1 usage  Ad15C
    # Vérifie si l'utilisateur a des emprunts en retard
    if has_overdue_borrowings(user):
        messages.error(request, message: "Vous avez des emprunts en retard. Impossible d\'emprunter de nouveaux médias.")
        return redirect('staff:media_liste')

    # Vérifie si l'utilisateur a déjà 3 emprunts actifs
    if has_max_active_borrows(user):
        messages.error(request, message: "Vous avez atteint la limite de 3 emprunts.")
        return redirect('staff:media_liste')

    return False
```

Le bon fonctionnement de l’emprunt a été testée via différents tests comme ceux ci-dessous, avec la tentative d’emprunt d’un jeu de plateau :

```
@pytest.mark.django_db  Ad15C
def test_borrow_media_with_board_game(staff_user, board_game, client):
    url = reverse( viewname: 'staff:emprunter', kwargs={'pk': board_game.pk})
    client.login(username='staff', password='password123')
    response = client.get(url, follow=False)
    assert response.status_code == 302
    assert response.url == reverse('staff:media_liste')
    response = client.get(response.url)
    messages = list(get_messages(response.wsgi_request))
    assert any("Les jeux de société ne peuvent pas être empruntés." in m.message for m in messages)
```

Impossibilité d'emprunter le jeu

et le succès d'un emprunt :

```python
@pytest.mark.django_db  Ad15C
def test_successful_borrow_media(staff_user, media_item, client):
    url = reverse( viewname: 'staff:emprunter', kwargs={'pk': media_item.pk})
    client.login(username='staff', password='password123')
    due_date_str = make_due_date(7)
    response = client.post(url, data={
        'media': media_item.pk,
        'due_date': due_date_str
    })
    assert response.status_code == 302
    assert StaffBorrowItem.objects.filter(user=staff_user, media=media_item).exists()
    response = client.get(response.url)
    messages = list(get_messages(response.wsgi_request))
    assert any("emprunté avec succès" in m.message.lower() for m in messages)
```

- Accueil
- Espace des Membres
- Déconnexion

Espace du Personnel de la Médiathèque

Retour au Dashboard

- Média emprunté avec succès !

Emprunt réussi

Succès

Vous avez emprunté le média : **eminem**

Type de média : cd

Date d'emprunt : 19 Jui 2025

Date limite de retour : 26 Jui 2025

Voir les détails de l'emprunt

Retour à la liste des médias

J’ai suivi également le même principe en créant une classe StaffBorrowItem, qui reprend les rôles de l’application Admin, et j’ai mis en place différentes vues permettant les actions qui nous avaient été demandées (affichage, modification, suppression).

Voici les URL de l'application Staff, montrant le fonctionnement/les routes de cette dernière.

```python
from django.urls import path
from mediatheque.authentification.views import staff_dashboard
from .views import media_views, member_views

app_name = 'staff'

urlpatterns = [
    # Dashboard
    path('espace_staff/', staff_dashboard, name='espace_staff'),

    # Médias
    path('media/liste/', media_views.media_list, name='media_liste'),
    path('media/<int:pk>/', media_views.media_detail, name='media_detail'),
    path('media/ajouter/', media_views.add_media, name='ajouter_media'),

    # Emprunts
    path('emprunter/<int:pk>/', media_views.borrow_media, name='emprunter'),
    path('emprunter/<int:pk>/detail/', media_views.borrow_detail, name='detail_emprunt'),
    path('emprunter/confirmer/<int:pk>/', media_views.confirm_borrow, name='confirmer_emprunt'),
    path('emprunter/<int:pk>/succes/', media_views.borrow_success, name='succes_emprunt'),
    path('media/<int:pk>/retourner/', media_views.return_media, name='retourner_media'),

    # Membres
    path('membres/', member_views.member_list, name='liste_membres'),
    path('membres/creer/', member_views.create_member, name='creer_membre'),
    path('membres/<int:pk>/modifier/', member_views.update_member, name='modifier_membre'),
    path('membres/<int:pk>/detail/', member_views.member_detail, name='membre_detail'),
    path('membres/<int:pk>/supprimer', member_views.delete_member, name='supprimer_membre')
]
```

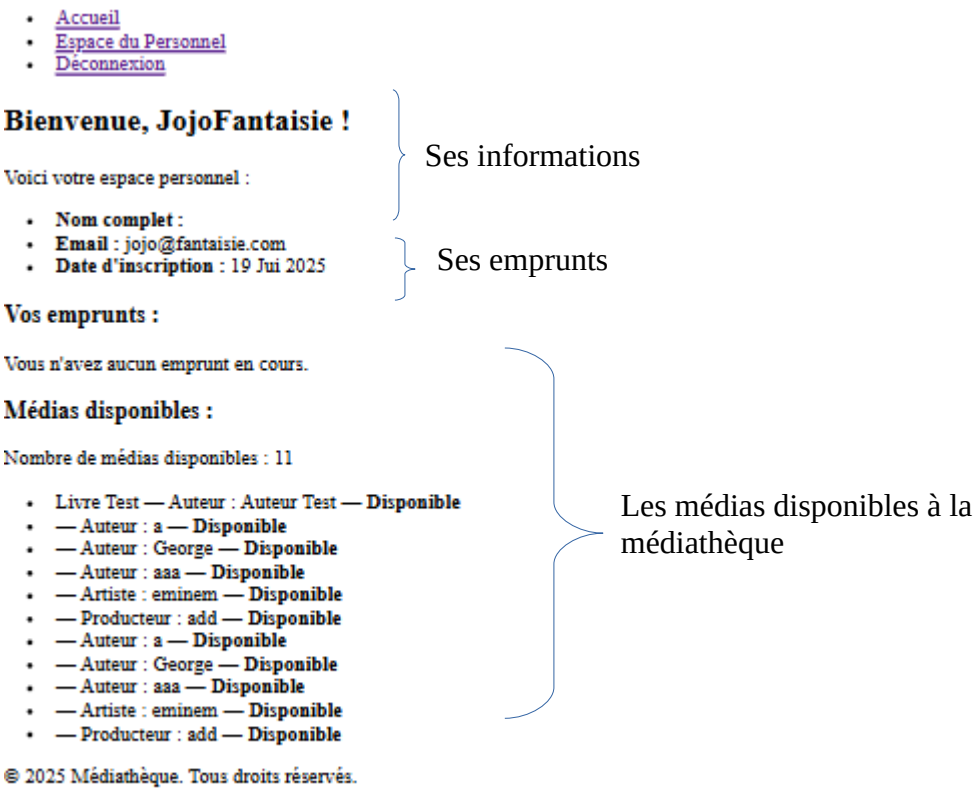
Pour chaque vue mise en place, un test a été mis en place afin de s'assurer le bon fonctionnement de cette dernière.

Enfin, pour l'application ***Client***, cette dernière a pour but de permettre aux membres de la médiathèque de voir les différents médias de la médiathèque.
L'importation des médias depuis l'application Staff vers l'application Client est assurée automatiquement via une commande personnalisée (*BaseCommand*). Pour chaque média staff, on utilise ContentType pour identifier son type (BookStaff, CDStaff, DVDStaff, BoardGameStaff), puis on cherche ou crée un objet *MediaClient* correspondant, avec les bonnes propriétés. Cette logique garantit que tous les médias créés dans l'application Staff soient visibles, à jour et bien structurés dans l'application Client, sans redondance. (Cf. photos ci-dessous)

```python
from django.core.management.base import BaseCommand
from django.contrib.contenttypes.models import ContentType
from mediatheque.staff.models import BookStaff, CDStaff, DVDStaff, BoardGameStaff
from mediatheque.client.models import MediaClient, BookClient, CDClient, DVDClient, BoardGameClient


class Command(BaseCommand):  Ad15C
    help = "Importer les médias de l'app staff vers l'app client avec content_type et object_id"

    def handle(self, *args, **options):  Ad15C
        self.import_books()
        self.import_cds()
        self.import_dvds()
        self.import_boardgames()
        self.stdout.write(self.style.SUCCESS("Import terminé avec succès."))

    def _create_or_update_media(self, staff_obj, client_model):  4 usages  Ad15C
        content_type = ContentType.objects.get_for_model(staff_obj)
        # Vérifier si MediaClient existe déjà pour ce staff_obj
        media_client = MediaClient.objects.filter(content_type=content_type, object_id=staff_obj.id).first()

        defaults = {
            "name": staff_obj.name,
            "media_type": client_model.MEDIA_TYPE,
            "is_available": getattr(staff_obj, "is_available", True),
            "can_borrow": getattr(staff_obj, "can_borrow", True),
            "description": getattr(staff_obj, "description", ""),
            "content_type": content_type,
            "object_id": staff_obj.id,
        }

        if media_client:
            # Mise à jour
            for key, value in defaults.items():
                setattr(media_client, key, value)
            media_client.save()
            created = False
        else:
            # Création
            media_client = MediaClient.objects.create(**defaults)
            created = True
```

```python
        # Maintenant on synchronise le sous-type client, lié à MediaClient
        client_obj = client_model.objects.filter(id=media_client.id).first()
        if not client_obj:
            # Crée une entrée dans la table spécifique client avec le même id
            client_obj = client_model(id=media_client.id)

        # Compléter champs spécifiques
        if client_model == BookClient:
            client_obj.author = getattr(staff_obj, "author", "")
        elif client_model == CDClient:
            client_obj.artist = getattr(staff_obj, "artist", "")
        elif client_model == DVDClient:
            client_obj.producer = getattr(staff_obj, "producer", "")
        elif client_model == BoardGameClient:
            client_obj.creators = getattr(staff_obj, "creators", "")
            client_obj.is_visible = getattr(staff_obj, "is_visible", True)
            client_obj.game_type = getattr(staff_obj, "game_type", None)
            client_obj.can_borrow = False  # Comme dans ton modèle

        client_obj.save()

        action = "Créé" if created else "Mis à jour"
        self.stdout.write(f"{action} média: {media_client.name} ({client_model.__name__})")

    def import_books(self):  1 usage  Ad15C
        for book in BookStaff.objects.all():
            self._create_or_update_media(book, BookClient)

    def import_cds(self):  1 usage  Ad15C
        for cd in CDStaff.objects.all():
            self._create_or_update_media(cd, CDClient)

    def import_dvds(self):  1 usage  Ad15C
        for dvd in DVDStaff.objects.all():
            self._create_or_update_media(dvd, DVDClient)

    def import_boardgames(self):  1 usage  Ad15C
        for game in BoardGameStaff.objects.all():
            self._create_or_update_media(game, BoardGameClient)
```

Et voici la page que le membre de la médiathèque aura lors de son utilisation :



Pour finir, le projet racine possède également un fichier avec les différentes URL (Cf. photo ci-dessous), puis un fichier de réglages (Settings), où l'on précise la langue du projet, le système de log/connexion/base de données, les chemins des templates pour chaque application, etc.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('auth/', include( arg: ('mediatheque.authentification.urls', 'authentification'), namespace='mediatheque.authentification')),
    path('', home_view, name='home'),
    path("client/", include( arg: ("mediatheque.client.urls", "client"), namespace="client")),
    path('staff/', include('mediatheque.staff.urls')),
]
```