RAPPORT DE TRAVAIL

MÉDIATHÈQUE -
“NOTRE LIVRE, NOTRE MÉDIA”

ANNE VILLETTE

SOMMAIRE

I. L’ étude et les correctifs du code fourni

État initial du code

```
def menu():
    print("menu")

if __name__ == '__main__':
    menu()


class livre():
    name = ""
    auteur = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class dvd():
    name = ""
        realisateur = ""
        dateEmprunt = ""
        disponible = ""
    emprunteur = ""


class cd():
    name = ""
    artiste = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class jeuDePlateau :
    name = ""
    createur = ""

class Emprunteur():
    name = ""
        bloque = ""


def menuBibliotheque() :
    print("c'est le menu de l'application des bibliothéquaire")




def menuMembre():
    print("c'est le menu de l'application des membres")
    print("affiche tout")
```

Problèmes identifiés

- Structure du code
  - Absence d'héritage alors que livre, dvd, cd partagent des attributs commun
  - Classes mal définies avec le manque de constructeur
  - Non-respect du norme de nommage Python, elle devrait être en PascalCase
- Code minimal et non fonctionnel
  - Simple affichage de texte dans les menus
  - Aucune logique
  - Pas de gestion de base de données
  - Pas de gestion des emprunts
- Manque de structure
  - Pas de séparation des responsabilités
  - Pas d'utilisation de framework
  - Pas de gestion des erreurs

Solutions apportées

- Restructuration avec Django
  - Création de 3 applications distinctes
    - Librarian : interface bibliothécaire
    - Member : interface publique
    - Home : page d'accueil
- Amélioration des Modèles
  - Création d'un model Media, parent des modèles Livre, DVD, CD.

```python
class Media(models.Model):  AnneVi
    title = models.CharField(max_length=200)
    is_available = models.BooleanField(default=True)

    class Meta:  AnneVi
        abstract:True

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

- Hiérarchie des médias

```python
class Book(Media):  AnneVi
    author = models.CharField(max_length=200)

class DVD(Media):  2 usages  AnneVi
    director = models.CharField(max_length=200)

class CD(Media):  AnneVi
    artist = models.CharField(max_length=200)


class BoardGame(models.Model):  AnneVi
    name = models.CharField(max_length=200)
    creator = models.CharField(max_length=200)

    def __str__(self):  AnneVi
        return self.name
```

- Gestion des membres

```python
class Member(models.Model):  2 usages  AnneVi
    first_name = models.CharField(max_length=100, null=False, default="Prénom")
    last_name = models.CharField(max_length=100, null=False, default="Nom")

    def __str__(self):  AnneVi
        return f'{self.last_name}, {self.first_name}'

    def can_borrow(self):  4 usages (4 dynamic)  AnneVi
        """
        Vérifie si le membre peut emprunter un nouveau livre.
        Retourne True si le membre a moins de 3 emprunts en cours, False sinon.
        """
        current_loans = self.loans.filter(return_date__isnull=True).count()
        return current_loans < 3

    def has_overdue_loans(self):  3 usages (3 dynamic)  AnneVi
        current_date = timezone.now().date()
        overdue_loans = [
            loan for loan in self.loans.filter(return_date__isnull=True)
            if (current_date - loan.loan_date).days > 7
        ]
        return len(overdue_loans) > 0

    class Meta:  AnneVi
        ordering = ['last_name', 'first_name']
```

- Système d'emprunt

```python
class Loan(models.Model):  2 usages  AnneVi
    member = models.ForeignKey('Member', on_delete=models.CASCADE, related_name='loans')
    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    item = GenericForeignKey('content_type', 'object_id')
    loan_date = models.DateField(default=timezone.now)
    return_date = models.DateField(null=True, blank=True)

    def __str__(self):  AnneVi
        return f"{self.member} a emprunté {self.item}"

    def due_date(self):  2 usages (1 dynamic)  AnneVi
        return (self.loan_date + timezone.timedelta(days=7))

    def is_overdue(self):  2 usages (2 dynamic)  AnneVi
        if self.return_date:
            return False
        current_date = timezone.now().date()
        due = self.due_date()
        # Convertir en date si c'est un datetime
        if hasattr(due, 'date'):
            due = due.date()
        return current_date > due

    class Meta:  AnneVi
        ordering = ['-loan_date']
```

➢ Mise en place de fonctionnalités
  - Système d'authentification pour les bibliothécaires
  - Gestion complète des emprunts
  - Interface de consultation publique
  - Système de logs pour le suivi des activités

Structure finale

➢ Architecture MVC avec Django
➢ Base de données relationnelle
➢ Tests unitaires pour chaque fonctionnalité
➢ Système de logs pour le suivi des actions
➢ Interfaces distinctes pour le publique et les bibliothécaires

II. La mise en place des fonctionnalités demandée

Application bibliothécaire

➢ Gestion des membres
  - Affichage de la liste des membres

```python
@login_required  1 usage  AnneVi
def member_list(request):
    members = Member.objects.all().order_by('last_name')
    return render(request, 'librarian/member_list.html', {'members': members})
```

- Création de nouveaux membres

```python
@login_required  1 usage  AnneVi
def add_member(request):
    if request.method == 'POST':
        form = AddMemberForm(request.POST)
        if form.is_valid():
            member = form.save(commit=False)
            member.save()
            messages.success(request, f"Le membre '{member.last_name}', '{member.first_name}' a été ajouté avec succès!"
            return redirect('dashboard')
    else:
        form = AddMemberForm()

    return render(request, 'librarian/add_member.html', {'form': form})
```

- Modification ou suppression de membre

```python
@login_required  1 usage  AnneVi
def member_detail(request, member_id):
    member = get_object_or_404(Member, id=member_id)
    if request.method == 'POST':
        if 'update' in request.POST:
            form = MemberForm(request.POST, instance=member)
            if form.is_valid():
                form.save()
                messages.success(request, f"Le membre '{member.last_name}' '{member.first_name}' a été mis à jour avec
                return redirect('member_detail', member_id=member.id)
        elif 'delete' in request.POST:
            member.delete()
            messages.success(request, f"Le membre '{member.last_name}' '{member.first_name}' a été supprimé avec succès
            return redirect('member_list')
    else:
        form = MemberForm(instance=member)

    return render(request, 'librarian/member_detail.html', {'form': form, 'member': member})
```

- Gestion des médias
  - Affichage des médias

```python
@login_required  1 usage  AnneVi
def media_list(request, media_type):
    # Mapping des types de médias vers leurs modèles et noms
    media_mapping = {
        'book': {'model': Book, 'name': 'Livres', 'name_singular': 'Livre'},
        'dvd': {'model': DVD, 'name': 'DVDs', 'name_singular': 'DVD'},
        'cd': {'model': CD, 'name': 'CDs', 'name_singular': 'CD'},
    }

    if media_type not in media_mapping:
        messages.error(request, "Type de média non valide.")
        return redirect('dashboard')

    media_info = media_mapping[media_type]
    items = media_info['model'].objects.all().order_by('title')

    context = {
        'items': items,
        'media_type': media_type,
        'media_name': media_info['name'],
        'media_name_singular': media_info['name_singular']
    }

    return render(request, 'librarian/media_list.html', context)
```

- Création de nouveaux médias

```python
@login_required  1 usage  AnneVi                                                    26 3
def media_add(request, media_type):
    # Mapping des types de médias vers leurs formulaires et noms
    form_mapping = {
        'book': {'form': BookForm, 'name': 'Livre'},
        'dvd': {'form': DVDForm, 'name': 'DVD'},
        'cd': {'form': CDForm, 'name': 'CD'},
    }

    if media_type not in form_mapping:
        messages.error(request, "Type de média non valide.")
        return redirect('dashboard')

    form_info = form_mapping[media_type]

    if request.method == 'POST':
        form = form_info['form'](request.POST)
        if form.is_valid():
            item = form.save()
            messages.success(request, f"Le {form_info['name']} '{item.title}' a été ajouté avec succès!")
            return redirect('media_list', media_type=media_type)
    else:
        form = form_info['form']()

    context = {
        'form': form,
        'media_type': media_type,
        'media_name': form_info['name']
    }

    return render(request, 'librarian/media_add.html', context)
```

- Modification ou suppression de médias

```python
@login_required  1 usage  AnneVi
def media_detail(request, media_type, item_id):
    # Mapping des types de médias vers leurs modèles, formulaires et noms
    media_mapping = {
        'book': {'model': Book, 'form': BookForm, 'name': 'Livre'},
        'dvd': {'model': DVD, 'form': DVDForm, 'name': 'DVD'},
        'cd': {'model': CD, 'form': CDForm, 'name': 'CD'},
    }

    if media_type not in media_mapping:
        messages.error(request, "Type de média non valide.")
        return redirect('dashboard')

    media_info = media_mapping[media_type]
    item = get_object_or_404(media_info['model'], id=item_id)

    if request.method == 'POST':
        if 'update' in request.POST:
            form = media_info['form'](request.POST, instance=item)
            if form.is_valid():
                form.save()
                messages.success(request, f"Le {media_info['name']} '{item.title}' a été mis à jour avec succès.")
                return redirect('media_detail', media_type=media_type, item_id=item.id)
        elif 'delete' in request.POST:
            title = item.title
            item.delete()
            messages.success(request, f"Le {media_info['name']} '{title}' a été supprimé avec succès.")
            return redirect('media_list', media_type=media_type)
    else:
        form = media_info['form'](instance=item)

    context = {
        'form': form,
        'item': item,
        'media_type': media_type,
        'media_name': media_info['name']
    }

    return render(request, 'librarian/media_detail.html', context)
```

- Système d'emprunt

```python
@login_required  1 usage  AnneVi
def borrow_item(request):
    # Si c'est une requête AJAX pour obtenir les items
    if 'item_type' in request.GET:
        item_type = request.GET.get('item_type')
        items = []
        if item_type == 'book':
            items = Book.objects.filter(is_available=True)
        elif item_type == 'dvd':
            items = DVD.objects.filter(is_available=True)
        elif item_type == 'cd':
            items = CD.objects.filter(is_available=True)

        return JsonResponse({
            'items': [{'id': item.id, 'text': str(item)} for item in items]
        })

    # Traitement normal du formulaire
    if request.method == 'POST':
        form = LoanForm(request.POST)
        if form.is_valid():
            member = form.cleaned_data['member']
            item_type = form.cleaned_data['item_type']
            item = form.cleaned_data['item']
            if member.has_overdue_loans():
                messages.error(request,
                               "Ce membre a des emprunts en retard et ne peut pas emprunter de nouveaux items.")
            elif not member.can_borrow():
                messages.error(request, "Ce membre a déjà atteint la limite de 3 emprunts")
            else:
                content_type = ContentType.objects.get_for_model(item.__class__)
                loan = Loan(
                    member=member,
                    content_type=content_type,
                    object_id=item.id
                )
                loan.save()

                item.is_available = False
                item.save()

                messages.success(request, f"Emprunt de '{item}' par {member} enregistré avec succès!")
                return redirect('borrow_item')
    else:
        form = LoanForm()

    context = {
        'form': form,
        'loans': Loan.objects.filter(return_date__isnull=True),
    }
    return render(request, 'librarian/borrow_item.html', context)
```

- Système de retour

```python
@login_required  1 usage  AnneVi
def return_list(request):
    # Récupérer tous les emprunts non retournés
    active_loans = Loan.objects.filter(return_date__isnull=True)
    context = {
        'loans': active_loans
    }
    return render(request, 'librarian/return_list.html', context)


@login_required  1 usage  AnneVi
def confirm_return(request, loan_id):
    loan = get_object_or_404(Loan, id=loan_id)

    if request.method == 'POST':
        # Marquer l'emprunt comme retourné
        loan.return_date = timezone.now()
        loan.save()

        # Rendre l'item disponible
        item = loan.item
        item.is_available = True
        item.save()

        messages.success(request, f"Retour de '{item}' par {loan.member} enregistré avec succès!")
        return redirect('return_item')

    context = {
        'loan': loan
    }
    return render(request, 'librarian/confirm_return.html', context)
```

- Tableau de bord
  - Vue d'ensemble des statistiques
  - Listes des emprunts en cours
  - Alertes pour les retards

```python
@login_required  AnneVi
def dashboard(request):
    # Statistiques générales
    total_members = Member.objects.count()
    active_loans = Loan.objects.filter(return_date__isnull=True)
    overdue_loans = [loan for loan in active_loans if loan.is_overdue()]

    # Statistiques par type de média
    media_stats = {
        'books': {
            'total': Book.objects.count(),
            'available': Book.objects.filter(is_available=True).count(),
            'name': 'Livres'
        },
        'dvds': {
            'total': DVD.objects.count(),
            'available': DVD.objects.filter(is_available=True).count(),
            'name': 'DVDs'
        },
        'cds': {
            'total': CD.objects.count(),
            'available': CD.objects.filter(is_available=True).count(),
            'name': 'CDs'
        }
    }
    context = {
        'total_members': total_members,
        'media_stats': media_stats,
        'current_loans': active_loans,
        'overdue_loans': overdue_loans,
        'total_loans': active_loans.count(),
        'total_overdue': len(overdue_loans)
    }

    return render(request, 'librarian/dashboard.html', context)
```

Application Membre

- Consultation du catalogue

```python
def available_media(request):  1 usage  AnneVi
    # Récupérer tous les médias disponibles
    available_books = Book.objects.filter(is_available=True)
    available_dvds = DVD.objects.filter(is_available=True)
    available_cds = CD.objects.filter(is_available=True)
    board_games = BoardGame.objects.all()

    context = {
        'books': available_books,
        'dvds': available_dvds,
        'cds': available_cds,
        'boardgames': board_games,
    }

    return render(request, 'member/available_media.html', context)
```

Sécurité et contrôle d'accès

- Authentification des bibliothécaires

```python
def user_login(request):  1 usage  AnneVi
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None and user.is_librarian:
            login(request, user)
            return redirect('dashboard')
        else:
            messages.error(request, "Identifiant ou mot de passe invalide")
    return render(request, 'librarian/login.html')
```

- Protections des Vues

  Toutes les vues de l'application bibliothécaire sont protégées par le décorateur « @login_required ». Seuls les utilisateurs authentifiés et ayant le statut de « bibliothécaire » peuvent y avoir accès. Les tentatives d'accès non autorisées sont automatiquement redirigées vers la page de connexion

- Gestion de la déconnexion

```python
def user_logout(request):  1 usage  AnneVi
    logout(request)
    return redirect('home')
```

- Système de logs

```
            logger.info(f"Nouveau membre créé: {member.last_name}, {member.first_name} par {request.user.username}")
            messages.success(request, f"Le membre '{member.last_name}', '{member.first_name}' a été ajouté avec succès!"
            return redirect('dashboard')
        else:
            logger.warning(f"Tentative échouée de création de membre par {request.user.username}")
```

```
                logger.warning(f"Tentative d'emprunt refusée - retards existants: {member}")
                messages.error(request, "Ce membre a des emprunts en retard...")
            elif not member.can_borrow():
                logger.warning(f"Tentative d'emprunt refusée - limite atteinte: {member}")
                messages.error(request, "Ce membre a déjà atteint la limite de 3 emprunts")
```

Cela sert à la traçabilité des actions, ça identifie l'utilisateur effectuant l'action. Et cela affiche les tentatives d'accès non autorisées

Contraintes métiers à implémenter

- Limitation des emprunts
  - Maximum 3 emprunts par membres
  - Durée limité d'emprunt à 7 jours sinon le membre ne peut plus emprunter (même s'il n'est pas à 3 emprunts consécutifs)
- Gestion des Jeux de Sociétés
  - Consultation uniquement, pas de possibilité d'emprunt
- Validation des données
  - Formulaires avec validation côté serveur
  - Messages d'erreurs explicites
  - Protection contre les données invalides

III. Stratégie des tests

Les tests sont développer afin d'assurer la fiabilité et la robustesse de l'application. La méthode adoptée teste 3 aspects principaux : les modèles, les vues et les règles métiers.

Les Modèles

Les tests des modèles vérifient l'intégrité des données et le bon fonctionnement des méthodes métier. Par exemple, pour la classe Member, je teste la limitation des 3 emprunts simultanés et la détection des retards. Pour les médias, la vérification de leur création et de leur disponibilité.

Les Vues

Chaque vue de l'application est testée pour s'assurer :

- De la protection des accès (authentification requise)
- Du bon traitement des formulaires (création, modification, suppression)
- Des redirections appropriées
- Des messages de succès ou d'erreur

Les règles métiers

- La limite de 3 emprunts par membre
- La durée d'emprunt de 7 jours

- Le blocage des membres ayant des retards
- La consultation unique des jeux de plateau

IV. Une base de données avec des données test

Pour permettre une utilisation et une démonstration immédiate de l'application, une base de données de test a été créée avec un ensemble de données représentatives. Ces données couvrent tous les cas d'utilisation de l'application.

Données initiales

- Comptes bibliothécaires
  2 comptes bibliothécaires ont été créé avec accès complet au système. Les identifiants sont :
  bilbiothécaire1 / django123
  bibliothécaire2 / django123
- Membres tests
  - 5 membres avec différents statuts :
    - Membre sans emprunt
    - Membre avec emprunts en cours
    - Membre ayant atteint la limite d'emprunts
    - Membre avec retard
    - Membre avec historique d'emprunt
- Catalogue de Médias
  - Livres : 4
  - DVDs : 3
  - CDs : 3
  - Jeux de Plateau : 2
- Emprunts
  - Emprunts en cours
  - Emprunts en retard
  - Emprunts retournés

Objectifs des Données de Test

Les données de test ont été choisies spécifiquement pour valider l'ensemble des fonctionnalités de l'application. Elles permettent notamment de vérifier que le système gère correctement les emprunts et leurs contraintes. Les scénarios incluent des membres ayant atteint leur limite de trois emprunts, des emprunts en retard et des retours de médias. Cette diversité de situations garantit que le suivi des disponibilités et l'application des règles de gestion fonctionnent correctement.