



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**
Membre de
HONORIS UNITED UNIVERSITIES



Rapport de Programmation Python

3^{ème} année

Ingénierie Informatique et Réseaux

Sous le thème

APPLICATION DE CINEMA

Réalisé par :

M. Akkarbich Ayman

M. Addaoui Yassir

M. Bouktib Mouhiddine

Table des matières

Introduction.....	4
Introduction :.....	5
Contexte et Problématique :.....	5
Contexte	5
Problématique	5
Chapitre 1 : Analyse et conception.....	6
Diagramme de Gant :.....	7
Chapitre 2 : Analyse et conception.....	8
Les acteurs principaux :.....	9
Diagrammes UML	9
Diagramme de classes :.....	9
Diagramme d'activité :.....	11
Diagramme de séquence :	11
Conclusion :	12
Chapitre 3 : Fonctionnalités Principales	13
Fonctionnalités Principales :	14
Accounts :	14
Models.py :.....	14
Forms .py :.....	15
urls.py :	15
Views.py :	16
Conclusion :	16
Booking :.....	17
Models.py :.....	17
Views.py :	18
Urls.py :.....	24
Conclusion :	25
Staff :	25
Models.py :.....	25
Conclusion :	28
Chapitre 4 : Démonstration.....	29
Démonstration :.....	30

Chapitre 5 : Difficultés rencontrés :	34
Chapitre 6 : Perspectives d'Évolution :	36
Perspectives d'Évolution :	37
Conclusion :	38

Introduction

Introduction :

Dans un monde de plus en plus connecté, les attentes des consommateurs en matière de simplicité, de rapidité et d'accessibilité transforment l'industrie du cinéma. Les files d'attente aux guichets, les erreurs humaines dans les réservations et le manque de visibilité sur les séances disponibles nuisent à l'expérience des spectateurs et à l'efficacité des cinémas. Une plateforme de gestion des films et des réservations en ligne répond à ces défis en automatisant les processus, en centralisant la gestion des données et en offrant une interface utilisateur intuitive.

Ce projet vise à concevoir une application web de réservation de places de cinéma, développée avec le framework Django, connu pour sa robustesse, sa sécurité et sa rapidité de développement. Cette solution améliore la satisfaction des clients tout en facilitant la gestion des cinémas, tout en mettant en pratique des compétences avancées en développement web.

Contexte et Problématique :

Contexte

Les technologies web ont transformé les habitudes des consommateurs, qui privilégient désormais les solutions en ligne pour réserver des billets ou consulter des plannings. Dans le secteur du cinéma, les réservations manuelles entraînent des files d'attente, des surréservations et une charge de travail importante pour les gestionnaires. Une plateforme numérique permet de centraliser la gestion des films, des séances et des réservations, offrant une expérience fluide pour les utilisateurs et une gestion optimisée pour les administrateurs.

Problématique

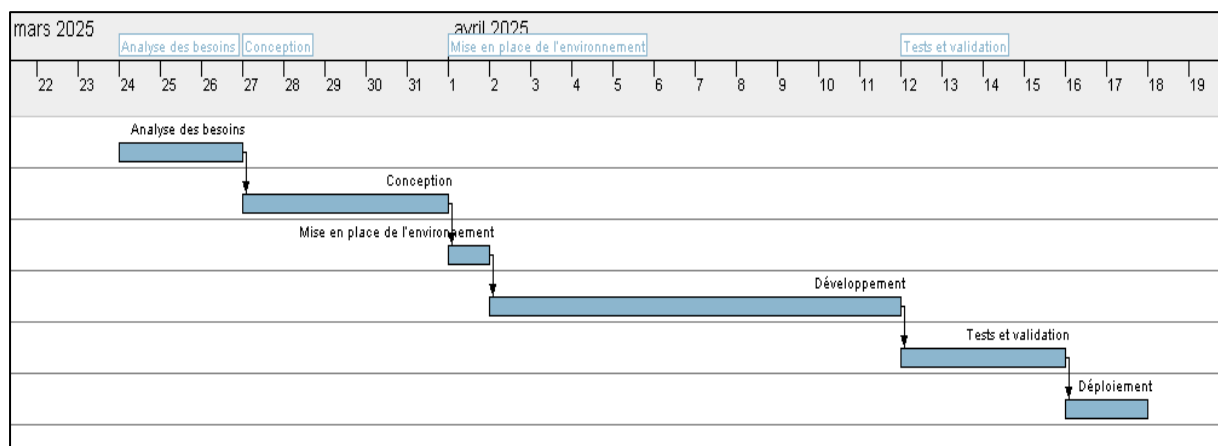
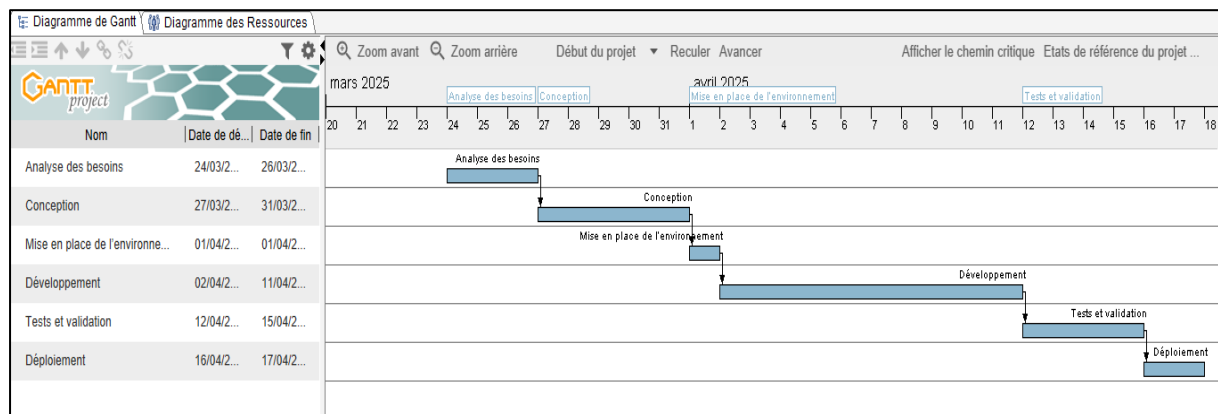
De nombreux cinémas, notamment les structures indépendantes, manquent de systèmes numériques performants, ce qui entraîne des erreurs de réservation, une mauvaise gestion des capacités et un manque de transparence pour les utilisateurs. La problématique est donc :

Comment concevoir un système en ligne simple, sécurisé et efficace pour gérer les films, les séances et les réservations, tout en garantissant une expérience utilisateur fluide et moderne ?

Ce projet répond à cette question en développant une application web avec Django, intégrant des fonctionnalités clés comme l'authentification, la réservation, et la gestion administrative.

Chapitre 1 : Analyse et conception

Diagramme de Gantt :



La planification du projet est représentée par le diagramme de Gantt (Figure 1), qui illustre la séquence et la durée des phases clés du projet, s'étendant de [date de début] à [date de fin]. Ce diagramme détaille six phases principales :

- **Analyse des besoins** : Cette phase initiale, prévue de 24/03/25 à 26/03/25 a permis d'identifier les exigences fonctionnelles et non fonctionnelles du système. Les livrables incluent les spécifications validées par les parties prenantes, servant de base aux diagrammes UML
- **Conception** : De 27/03/25 à 31/03/25, cette phase a englobé la modélisation du système, notamment via la création de diagrammes UML (cas d'utilisation, classes, séquence) pour définir l'architecture logicielle et les interactions entre composants.
- **Mise en place de l'environnement** : Planifiée de 01/04/25 à 01/04/25, cette étape a couvert la configuration de l'environnement de développement et de test, incluant l'installation des serveurs, des outils de développement, et des bases de données, conformément à l'architecture définie.
- **Développement** : De 02/04/25 à 11/04/25, cette phase a consisté en l'implémentation des fonctionnalités du système, en suivant les spécifications des diagrammes UML et les choix architecturaux. Les tâches ont été réparties entre les équipes frontend et backend.
- **Tests et validation** : Prévue de 12/04/25 à 15/04/25, cette phase a inclus des tests unitaires, d'intégration et fonctionnels pour valider la conformité du système aux exigences. Les cas d'utilisation définis dans les diagrammes UML ont servi de base aux scénarios de test.
- **Déploiement** : La phase finale, de 16/04/25 à 17/04/25, a couvert la mise en production du système, incluant la configuration finale des serveurs et la validation par les utilisateurs.

Chapitre 2 : Analyse et conception

Analyse et Conception

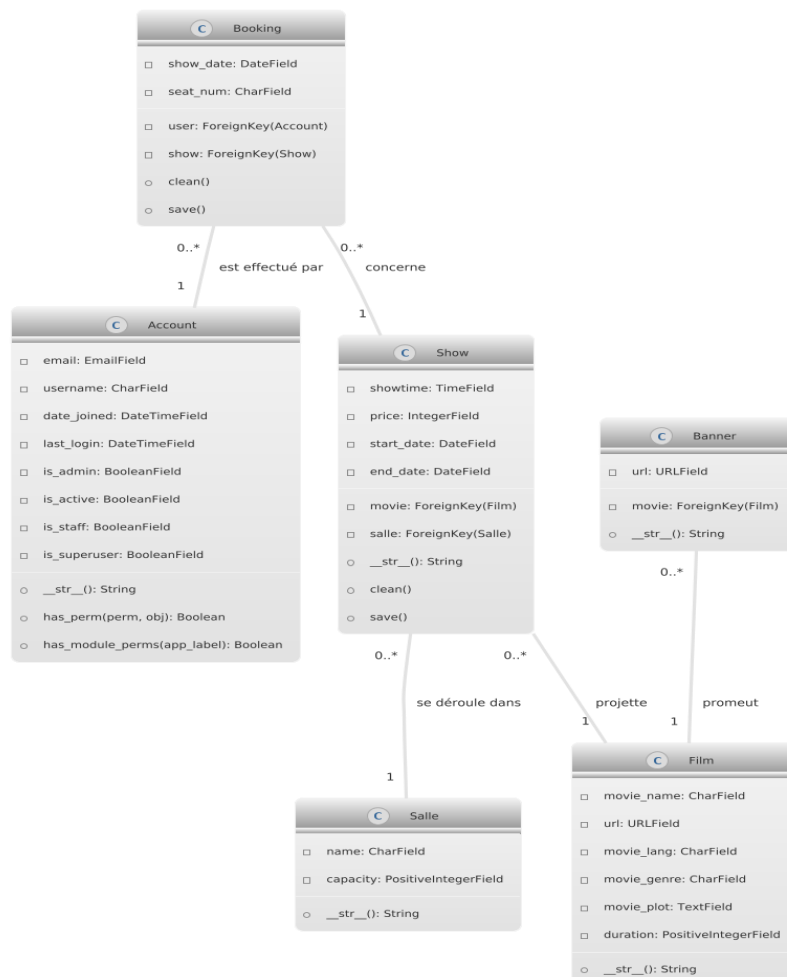
Les acteurs principaux :

Acteur	Interactions principales	Statut
Utilisateur	Réserver, annuler, consulter les séances, s'inscrire	Externe
Staff	Gérer films, séances, salles, bannières,	Externe

Ces deux acteurs ont des rôles distincts : les utilisateurs finaux (clients) effectuent des réservations, tandis que le staff gère les données du système via une interface d'administration sécurisée.

Diagrammes UML

Diagramme de classes :



Les cardinalités :

1. Booking "0..*" -- "1" Account : est effectué par

- **Cardinalité :**

- "0..*" côté Booking : Un utilisateur peut n'avoir aucune réservation ou plusieurs réservations.
- "1" côté Account : Chaque réservation doit être liée à un seul utilisateur (la clé étrangère garantit qu'une réservation ne peut pas exister sans un utilisateur)

2. Booking "0..*" -- "1" Show : concerne

- **Cardinalité :**

- "0..*" côté Booking : Une séance peut n'avoir aucune réservation (par exemple, si personne n'a réservé) ou plusieurs réservations.
- "1" côté Show : Chaque réservation est liée à une seule séance spécifique.

3. Show "0..*" -- "1" Film : projette

- **Cardinalité :**

- "0..*" côté Show : Un film peut n'avoir aucune séance programmée ou plusieurs séances (par exemple, à différentes heures ou dans différentes salles).
- "1" côté Film : Chaque séance projette un seul film.

4. Show "0..*" -- "1" Salle : se déroule dans

- **Cardinalité :**

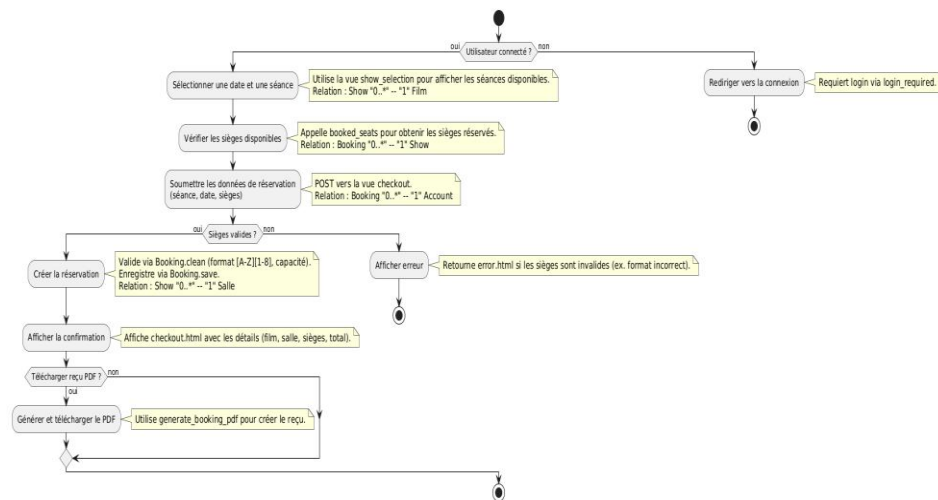
- "0..*" côté Show : Une salle peut n'avoir aucune séance programmée ou plusieurs séances à différents moments.
- "1" côté Salle : Chaque séance est associée à une seule salle.

5. Banner "0..*" -- "1" Film : promeut

- **Cardinalité :**

- "0..*" côté Banner : Un film peut n'avoir aucune bannière ou plusieurs bannières promotionnelles.
- "1" côté Film : Chaque bannière promeut un seul film.

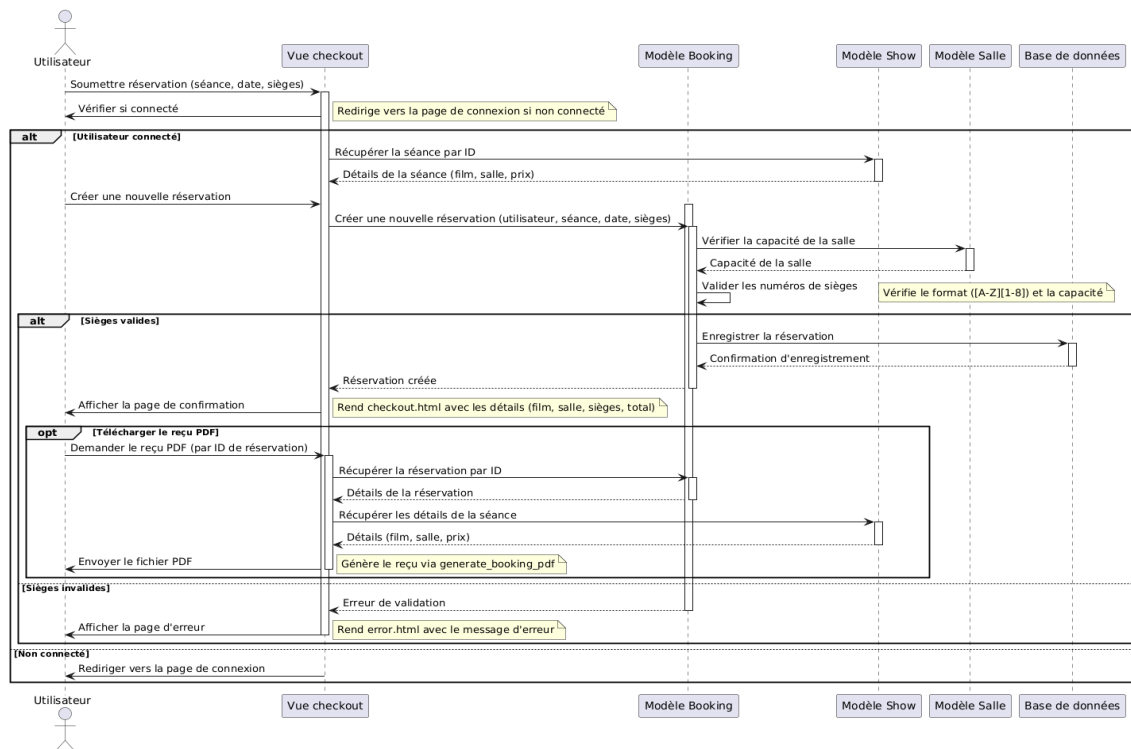
Diagramme d'activité :



Le diagramme d'activité décrit le flux de réservation :

1. L'utilisateur se connecte.
2. Il consulte les films et sélectionne une séance.
3. Il choisit des sièges (format [A-Z][1-8]).
4. Le système valide les sièges et enregistre la réservation.
5. Une confirmation est affichée, avec option de télécharger un reçu PDF.

Diagramme de séquence :



Le diagramme de séquence montre les interactions :

1. L'utilisateur envoie une requête de réservation via la vue checkout.
2. Le système vérifie l'authentification et la validité des sièges.
3. Si valide, la réservation est enregistrée dans Booking.
4. Un reçu PDF peut être généré via `generate_booking_pdf`.

Conclusion :

Les trois diagrammes — de séquence, d'activité et de classes — décrivent de manière complémentaire et cohérente le fonctionnement global du système de réservation de séances de cinéma. Le diagramme de séquence met en évidence les interactions dynamiques entre l'utilisateur, les différentes vues de l'application et les modèles métiers, depuis la soumission de la réservation jusqu'à la génération optionnelle d'un reçu PDF. Il illustre notamment les vérifications d'authentification, de capacité de la salle, de validité des sièges, et le traitement différencié selon que la réservation est valide ou non.

Le diagramme d'activité synthétise cette logique en exposant les étapes décisionnelles : connexion de l'utilisateur, sélection d'une séance, validation des sièges selon un format défini ([A-Z][1-8]) et enregistrement conditionnel de la réservation, avec en cas de succès l'affichage d'une confirmation et la possibilité de télécharger un reçu. Quant au diagramme de classes, il structure les entités principales du système et leurs relations : une réservation (Booking) est liée à un utilisateur (Account) et une séance (Show), cette dernière étant connectée à une salle (Salle) et à un film (Film), avec des attributs précis tels que le prix, la capacité, ou les horaires.

L'ensemble offre une vision claire d'un système bien conçu, à la fois fonctionnel et extensible, assurant une gestion rigoureuse des données, une expérience utilisateur fluide, et une architecture solide reposant sur les principes de la modélisation orientée objet.

Chapitre 3 : Fonctionnalités Principales

Fonctionnalités Principales :

Accounts :

Models.py :

```
from django.db import models
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager

class AccountManager(BaseUserManager):
    def create_user(self, email, username, password=None):
        if not email:
            raise ValueError('Email is required')
        if not username:
            raise ValueError('Username is required')
        user = self.model(email=self.normalize_email(email), username=username)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, username, password):
        user = self.create_user(email, username, password)
        user.is_admin = True
        user.is_staff = True
        user.is_superuser = True
        user.save(using=self._db)
        return user

class Account(AbstractBaseUser):
    email = models.EmailField(unique=True)
    username = models.CharField(max_length=30, unique=True)
    date_joined = models.DateTimeField(auto_now_add=True)
    last_login = models.DateTimeField(auto_now=True)
    is_admin = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    objects = AccountManager()

    def __str__(self):
        return self.email

    def has_perm(self, perm, obj=None):
        return self.is_admin

    def has_module_perms(self, app_label):
        return True
```

Forms .py :

```
from django import forms
from .models import Account

class RegistrationForm(forms.ModelForm): # enregistrer un nouvel utilisateur.
    password = forms.CharField(widget=forms.PasswordInput)
    confirm_password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = Account
        fields = ['email', 'username', 'password']

    def clean(self):
        cleaned_data = super().clean()
        password = cleaned_data.get('password')
        confirm_password = cleaned_data.get('confirm_password')
        if password != confirm_password:
            raise forms.ValidationError("Passwords do not match")
        return cleaned_data

class LoginForm(forms.Form): # pour connecter un utilisateur existant.
    email = forms.EmailField()
    password = forms.CharField(widget=forms.PasswordInput)
```

urls.py :

```
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
]
```

Views.py :

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login as auth_login, logout as auth_logout
from .forms import RegistrationForm, LoginForm
from django.contrib import messages

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.set_password(form.cleaned_data['password'])
            user.save()
            messages.success(request, 'Registration successful!')
            return redirect('login')
    else:
        form = RegistrationForm()
    return render(request, 'accounts/register.html', {'form': form})

def login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            user = authenticate(request, email=email, password=password)
            if user:
                auth_login(request, user)
                return redirect('index')
            else:
                messages.error(request, 'Invalid email or password')
    else:
        form = LoginForm()
    return render(request, 'accounts/login.html', {'form': form})

def logout(request):
    auth_logout(request)
    return redirect('index')
```

Conclusion :

Ce code met en place un système complet d'authentification personnalisé, en utilisant un modèle Account basé sur l'email comme identifiant principal. Le fichier `models.py` définit ce modèle en héritant de `AbstractBaseUser` avec un gestionnaire personnalisé (`AccountManager`) pour créer des utilisateurs et des superutilisateurs. Le fichier `forms.py` contient deux formulaires : un pour l'enregistrement (`RegistrationForm`) avec une vérification de confirmation de mot de passe, et un pour la connexion (`LoginForm`).

Dans `views.py`, les vues `register`, `login` et `logout` gèrent respectivement l'inscription, la connexion et la déconnexion des utilisateurs avec un retour d'information via messages. Enfin, `urls.py` relie ces vues aux chemins `/register/`, `/login/` et `/logout/`. L'ensemble constitue un système robuste et sécurisé d'inscription et d'authentification sur mesure, remplaçant le système utilisateur par défaut de Django.

Booking :

Models.py :

```
from django.db import models
from django.conf import settings
from staff.models import show, Salle
from django.core.exceptions import ValidationError
import re

class Booking(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    show = models.ForeignKey(show, on_delete=models.CASCADE)
    show_date = models.DateField()
    seat_num = models.CharField(max_length=100)

    def clean(self):
        if not self.seat_num:
            raise ValidationError("Seat numbers cannot be empty.")

        seats = self.seat_num.split(',')
        salle_capacity = self.show.salle.capacity
        seat_pattern = re.compile(r'^[A-Z][1-9][0-9]*$') # Matches A1, B12, etc.
        max_seats_per_row = 8 # indiquant qu'il y a 8 sièges par rangée
        row_labels = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'

        for seat in seats:
            seat = seat.strip()
            if not seat_pattern.match(seat):
                raise ValidationError(f"Seat '{seat}' is not in a valid format (e.g., A1, B12).")

            row = seat[0] # e.g., 'C'
            try:
                col = int(seat[1:]) # e.g., '4' from 'C4'
            except ValueError:
                raise ValidationError(f"Seat '{seat}' column must be a valid integer.")

            if row not in row_labels:
                raise ValidationError(f"Seat '{seat}' row '{row}' is invalid.")

            row_index = row_labels.index(row)
            max_rows = (salle_capacity + max_seats_per_row - 1) // max_seats_per_row

            if row_index >= max_rows:
                raise ValidationError(f"Seat '{seat}' row '{row}' exceeds hall capacity (max {max_rows} rows).")

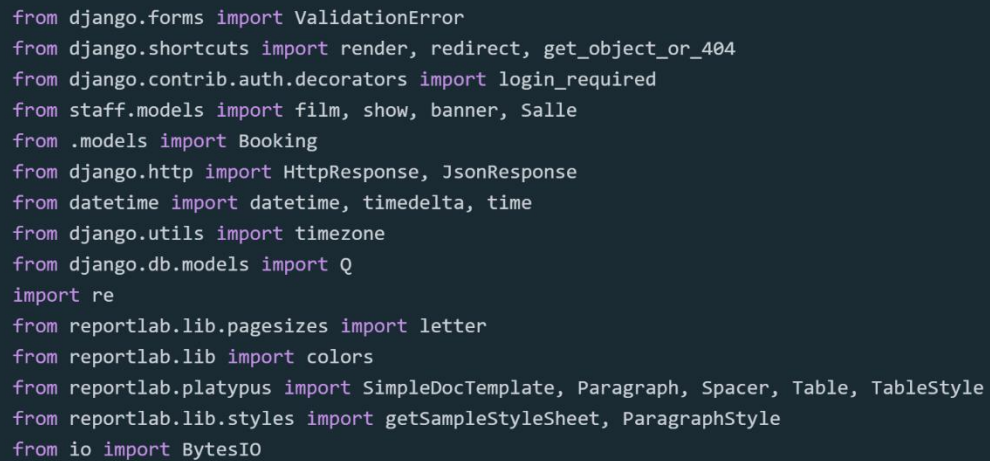
            if col < 1 or col > max_seats_per_row:
                raise ValidationError(f"Seat '{seat}' column {col} is invalid (must be 1 to {max_seats_per_row}).")

            seat_index = row_index * max_seats_per_row + col
            if seat_index > salle_capacity:
                raise ValidationError(f"Seat '{seat}' exceeds hall capacity of {salle_capacity}.")

    def save(self, *args, **kwargs):
        self.clean()
        super().save(*args, **kwargs)
```

Views.py :

Imports :



```
from django.forms import ValidationError
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from staff.models import film, show, banner, Salle
from .models import Booking
from django.http import HttpResponse, JsonResponse
from datetime import datetime, timedelta, time
from django.utils import timezone
from django.db.models import Q
import re
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from io import BytesIO
```

Homepage :

```

def index(request):
    banners = banner.objects.all()
    now = timezone.now()
    films = film.objects.filter(
        show__end_date__gte=now.date(),
        show__start_date__lte=now.date()
    ).distinct()

    tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')
    return render(request, 'booking/index.html', {
        'banners': banners,
        'films': films,
        'tomorrow': tomorrow
    })

```

Movie Detail Page :

```

def movie_detail(request, movie_id):
    film_obj = get_object_or_404(film, id=movie_id)
    now = timezone.now()
    showtimes = show.objects.filter(
        movie=film_obj,
        end_date__gte=now.date(),
        start_date__lte=now.date()
    ).select_related('salle')

    filtered_showtimes = []
    tz = timezone.get_current_timezone()
    for s in showtimes:
        show_datetime = datetime.combine(now.date(), s.showtime, tzinfo=tz)
        end_datetime = show_datetime + timedelta(minutes=film_obj.duration)
        if now.date() < s.start_date or (now.date() == s.start_date and now < end_datetime):
            filtered_showtimes.append(s)

    tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')
    return render(request, 'booking/movie_detail.html', {
        'film': film_obj,
        'showtimes': filtered_showtimes,
        'tomorrow': tomorrow
    })

```

My Bookings Page :

```
@login_required
def my_bookings(request):
    bookings = Booking.objects.filter(user=request.user).order_by('-show_date').select_related('show__movie', 'show__salle')
    tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')
    today = datetime.now().date()
    return render(request, 'booking/bookings.html', {'data': bookings, 'today': today, 'tomorrow': tomorrow})
```

Cancel Booking:

```
@login_required
def cancel_booking(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id, user=request.user)
    if booking.show_date >= datetime.now().date():
        booking.delete()
    return redirect('my_bookings')
```

Show Selection by Date :

```

def show_selection(request):
    date_str = request.GET.get('date')
    default_date = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')

    if date_str and re.match(r'^\d{4}-\d{2}-\d{2}$', date_str):
        try:
            selected_date = datetime.strptime(date_str, '%Y-%m-%d').date()
            date = date_str
        except ValueError:
            selected_date = datetime.now().date() + timedelta(days=1)
            date = default_date
    else:
        selected_date = datetime.now().date() + timedelta(days=1)
        date = default_date

    now = timezone.now()

    films = film.objects.prefetch_related('show_set__salle').filter(
        show__start_date__lte=selected_date,
        show__end_date__gte=selected_date
    ).distinct()

    films_dict = {}
    tz = timezone.get_current_timezone()
    for f in films:
        showtimes = {}
        for s in f.show_set.filter(start_date__lte=selected_date, end_date__gte=selected_date):
            show_datetime = datetime.combine(selected_date, s.showtime, tzinfo=tz)
            end_datetime = show_datetime + timedelta(minutes=f.duration)
            if selected_date > now.date() or (selected_date == now.date() and now < end_datetime):
                showtimes[s.id] = {'showtime': s.showtime, 'salle': s.salle.name}
        if showtimes:
            films_dict[f.movie_name] = {'url': f.url, 'showtimes': showtimes}

    tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')
    thirty_days_later = (datetime.now() + timedelta(days=30)).strftime('%Y-%m-%d')
    return render(request, 'booking/show_selection.html', {
        'films': films_dict,
        'date': date,
        'tomorrow': tomorrow,
        'thirty_days_later': thirty_days_later
    })

```

Checkout & PDF Generation

```

def generate_booking_pdf(booking, film, salle, show, sdate, seats, total):
    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=letter)
    elements = []
    styles = getSampleStyleSheet()

    title_style = ParagraphStyle(
        name='Title',
        fontSize=18,
        spaceAfter=20,
        alignment=1,
        textColor=colors.darkblue,
        fontName='Helvetica-Bold'
    )
    normal_style = ParagraphStyle(
        name='Normal',
        fontSize=12,
        spaceAfter=10,
        fontName='Helvetica'
    )

    elements.append(Paragraph("MovieTicket Booking Receipt", title_style))
    elements.append(Spacer(1, 12))

    data = [
        ["Movie:", film.movie_name],
        ["Hall:", salle.name],
        ["Date:", sdate],
        ["Showtime:", show.showtime.strftime('%I:%M %p')],
        ["Seats:", seats],
        ["Total:", f"${total:.2f}"],
        ["User:", booking.user.username],
        ["Booking ID:", str(booking.id)],
        ["Booking Date:", booking.show_date.strftime('%Y-%m-%d')],
    ]

    table = Table(data, colWidths=[100, 300])
    table.setStyle(TableStyle([
        ('FONTNAME', (0, 0), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 0), (-1, -1), 12),
        ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
        ('BOX', (0, 0), (-1, -1), 1, colors.black),
        ('BACKGROUND', (0, 0), (0, -1), colors.lightgrey),
    ]))
    elements.append(table)
    elements.append(Spacer(1, 20))
    elements.append(Paragraph("Thank you for booking with MovieTicket!", normal_style))
    elements.append(Paragraph("Contact us at support@movieticket.com", normal_style))

    doc.build(elements)
    pdf = buffer.getvalue()
    buffer.close()
    return pdf

```

```

@login_required
def checkout(request):
    tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')
    if request.method == 'POST':
        show_id = request.POST.get('showid')
        show_date = request.POST.get('showdate')
        seats = request.POST.get('seats')
        if show_id and show_date and seats:
            show_obj = get_object_or_404(show, id=show_id)
            seat_list = seats.split(',')
            seat_pattern = re.compile(r'^[A-Z][1-9][0-9]*$')
            for seat in seat_list:
                if not seat_pattern.match(seat.strip()):
                    return render(request, 'booking/error.html', {
                        'error': f"Seat '{seat}' is not in a valid format (e.g., A1, B12).",
                        'tomorrow': tomorrow
                    })
            try:
                booking = Booking.objects.create(
                    user=request.user,
                    show=show_obj,
                    show_date=show_date,
                    seat_num=seats
                )
            except ValidationError as e:
                return render(request, 'booking/error.html', {
                    'error': str(e),
                    'tomorrow': tomorrow
                })
            total = len(seat_list) * show_obj.price
            context = {
                'film': show_obj.movie,
                'salle': show_obj.salle,
                'sdate': show_date,
                'show': show_obj,
                'seats': seats,
                'total': total,
                'tomorrow': tomorrow,
                'booking_id': booking.id
            }
            return render(request, 'booking/checkout.html', context)

    elif request.GET.get('download_pdf') == 'true':
        booking_id = request.GET.get('booking_id')
        booking = get_object_or_404(Booking, id=booking_id, user=request.user)
        show_obj = booking.show
        film = show_obj.movie
        salle = show_obj.salle
        sdate = booking.show_date.strftime('%Y-%m-%d')
        seats = booking.seat_num
        total = len(seats.split(',')) * show_obj.price

        pdf = generate_booking_pdf(booking, film, salle, show_obj, sdate, seats, total)
        response = HttpResponse(content_type='application/pdf')
        response['Content-Disposition'] = f'attachment; filename="booking_{booking.id}_receipt.pdf"'
        response.write(pdf)
        return response

    return redirect('index')

```

AJAX/Utility Views:

```
def booked_seats(request):
    show_id = request.GET.get('show_id')
    show_date = request.GET.get('show_date')
    bookings = Booking.objects.filter(show_id=show_id, show_date=show_date)
    seats = ','.join([b.seat_num for b in bookings])
    return HttpResponse(seats)

def show_details(request):
    show_id = request.GET.get('show_id')
    try:
        show_obj = show.objects.select_related('salle').get(id=show_id)
        return JsonResponse({
            'capacity': show_obj.salle.capacity,
            'salle_name': show_obj.salle.name,
        })
    except show.DoesNotExist:
        return JsonResponse({'error': 'Show not found'}, status=404)
```

Urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('detail/<int:movie_id>/', views.movie_detail, name='movie_detail'),
    path('show/', views.show_selection, name='show_selection'),
    path('mybookings/', views.my_bookings, name='my_bookings'),
    path('checkout/', views.checkout, name='checkout'),
    path('cancelbooking/<int:booking_id>/', views.cancel_booking, name='cancel_booking'),
    path('bookedseats/', views.booked_seats, name='booked_seats'),
    path('show_details/', views.show_details, name='show_details'),
]
```


Conclusion:

Ce fichier de vues Django constitue le cœur de la logique de l'application de réservation de films. Il gère les interactions essentielles entre l'utilisateur et le système : l'affichage des films et séances disponibles, la sélection des places, la création des réservations, ainsi que la génération des reçus au format PDF.

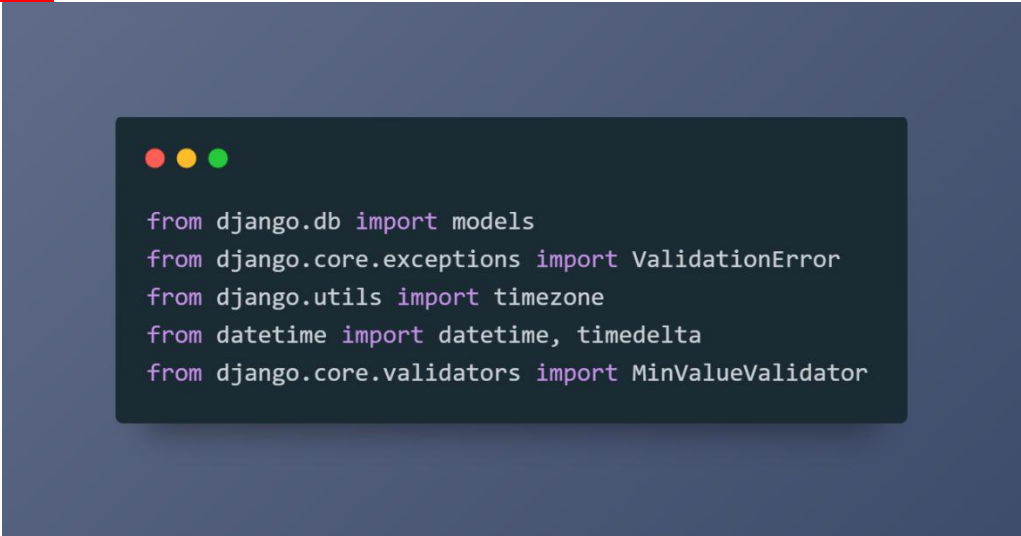
Grâce à une architecture claire et segmentée, chaque vue remplit une responsabilité bien définie, garantissant la lisibilité et la maintenabilité du code. Les contrôles de validité (des sièges, des dates, etc.) permettent de sécuriser les données, et l'intégration de bibliothèques externes comme ReportLab enrichit l'expérience utilisateur en proposant des fonctionnalités avancées.

Ce système peut encore être amélioré par l'ajout de tests unitaires, la séparation des responsabilités dans des fichiers dédiés (utils.py, pdf_utils.py, etc.), et l'intégration d'une gestion d'erreurs plus robuste. Cependant, tel quel, ce code offre une base fonctionnelle solide pour une application web moderne de billetterie cinéma.

Staff :

Models.py :

Models :



```
from django.db import models
from django.core.exceptions import ValidationError
from django.utils import timezone
from datetime import datetime, timedelta
from django.core.validators import MinValueValidator
```

Modèle banner (Bannière pour la page d'accueil)

```

class banner(models.Model):
    movie = models.ForeignKey(film, on_delete=models.CASCADE)
    url = models.URLField()

    def __str__(self):
        return f"Banner for {self.movie.movie_name}"

```

Modèle Salle (Salle de projection) :

```

class Salle(models.Model):
    name = models.CharField(max_length=50, unique=True) # e.g., "Hall 1", "Hall 2"
    capacity = models.PositiveIntegerField() # Number of seats in the hall

    def __str__(self):
        return self.name

```

Modèle film (Film à projeter) :

```

class film(models.Model):
    movie_name = models.CharField(max_length=100)
    url = models.URLField()
    movie_lang = models.CharField(max_length=50, blank=True)
    movie_genre = models.CharField(max_length=50, blank=True)
    movie_plot = models.TextField(blank=True)
    duration = models.PositiveIntegerField(
        default=120,
        help_text="Duration in minutes",
        validators=[MinValueValidator(60)] # Minimum duration = 60 min
    )

    def __str__(self):
        return self.movie_name

```

Modèle show (Séance d'un film) :

```
class show(models.Model):
    movie = models.ForeignKey(film, on_delete=models.CASCADE)
    salle = models.ForeignKey(Salle, on_delete=models.CASCADE)
    showtime = models.TimeField()
    price = models.IntegerField()
    start_date = models.DateField()
    end_date = models.DateField()

    def __str__(self):
        return f"{self.movie.movie_name} in {self.salle.name} at {self.showtime}"
    def clean(self):
        if self.start_date > self.end_date:
            raise ValidationError("Start date cannot be after end date.")

        if self.movie and self.salle:
            tz = timezone.get_current_timezone()
            current_date = self.start_date
            duration_minutes = self.movie.duration

            while current_date <= self.end_date:
                show_datetime = datetime.combine(current_date, self.showtime, tzinfo=tz)
                end_datetime = show_datetime + timedelta(minutes=duration_minutes)

                conflicting_shows = show.objects.filter(
                    salle=self.salle,
                    start_date__lte=current_date,
                    end_date__gte=current_date
                ).exclude(pk=self.pk)

                for existing_show in conflicting_shows:
                    existing_show_datetime = datetime.combine(current_date, existing_show.showtime, tzinfo=tz)
                    existing_end_datetime = existing_show_datetime + timedelta(minutes=existing_show.movie.duration)

                    if not (end_datetime <= existing_show_datetime or show_datetime >= existing_end_datetime):
                        raise ValidationError(
                            f"Show conflicts with '{existing_show.movie.movie_name}' "
                            f"on {current_date.strftime('%Y-%m-%d')} from {existing_show.showtime.strftime('%I:%M %p')} "
                            f"to {(existing_end_datetime).strftime('%I:%M %p')} in {self.salle.name}."
                        )
                    current_date += timedelta(days=1)
    def save(self, *args, **kwargs):
        self.full_clean() # Appelle clean()
        super().save(*args, **kwargs)

class Meta:
    indexes = [
        models.Index(fields=['start_date', 'end_date', 'showtime']),
    ]
```

Conclusion :

Ce code met en place un système de gestion de cinéma à l'aide de Django en définissant les modèles essentiels pour organiser les projections de films. Le modèle Salle représente les différentes salles de cinéma avec leur nom et capacité. Le modèle film contient les informations détaillées sur les films, telles que le nom, la langue, le genre, le résumé et la durée. Le modèle show gère les séances, en reliant chaque film à une salle, à une date et à une heure spécifiques, tout en intégrant une logique de validation permettant d'éviter les conflits d'horaire entre les projections. Enfin, le modèle banner permet d'associer des affiches promotionnelles aux films. L'ensemble assure une structure robuste, cohérente et extensible pour la planification et la réservation de séances dans un environnement de cinéma.

Chapitre 4 : Démonstration

Démonstration :

Login/ Signup :

The screenshot displays the MORRO_CINE website's registration interface. The header features the site logo, a 'Book Show' link, and buttons for 'LOGIN' and 'REGISTER'. The main content area contains a 'REGISTER' form with fields for Email, Username, Password, and Confirm password. The Username field is pre-filled with 'yassir2003@gmail.com' and the Password field with '*****'. A 'REGISTER' button and a link to 'Login' are at the bottom of the form. The footer includes copyright information, navigation links, and social media icons.

MORRO_CINE Book Show [LOGIN](#) [REGISTER](#)

REGISTER

Email*

Username*

Password*

Confirm password*

[REGISTER](#)

Already have an account? [Login](#)

Copyright © Morro_Cine 2025 [About](#) [Contact](#) [Privacy Policy](#) [f](#) [t](#) [i](#)

L'utilisateur peut s'inscrire avec un email et un mot de passe, se connecter, et se déconnecter. Les formulaires sont validés pour garantir la sécurité.

The screenshot displays the MORRO_CINE website's login interface. The header features the site logo, a 'Book Show' link, and buttons for 'LOGIN' and 'REGISTER'. The main content area contains a 'LOGIN' form with fields for Email and Password. A 'LOGIN' button and a link to 'Register' are at the bottom of the form. The footer includes copyright information, navigation links, and social media icons.

MORRO_CINE Book Show [LOGIN](#) [REGISTER](#)

LOGIN

Email*

Password*

[LOGIN](#)

Don't have an account? [Register](#)

Copyright © Morro_Cine 2025 [About](#) [Contact](#) [Privacy Policy](#) [f](#) [t](#) [i](#)

Main :

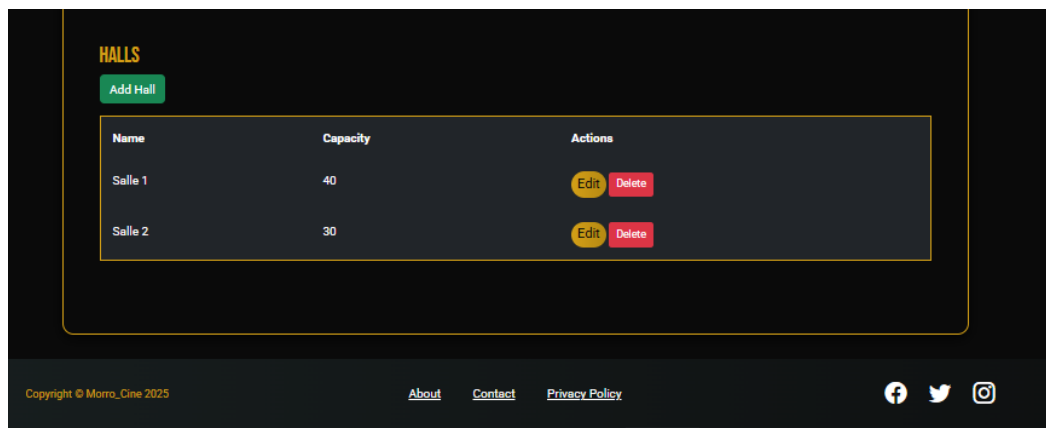
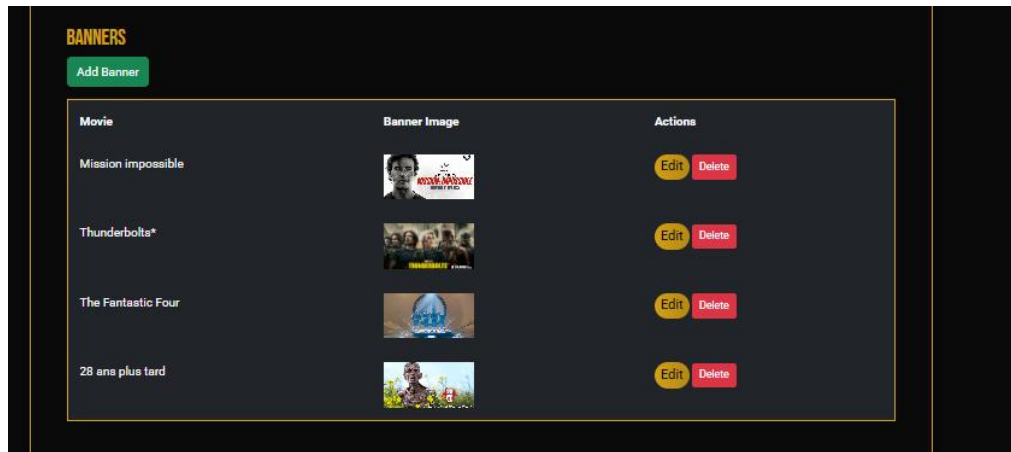
The screenshot shows the 'MORRO_CINE' Staff Dashboard. The top navigation bar includes 'Book Show', 'My Bookings', and 'Staff Dashboard'. The user is logged in as 'Hello addaoui' with a 'Logout' link. The main section is titled 'STAFF DASHBOARD' and contains a 'FILMS' subsection with an 'Add Film' button. Below this is a table listing six films with their posters, names, languages, genres, durations, and edit/delete actions.

Poster	Name	Language	Genre	Duration (min)	Actions
	Mission impossible	English	Action, Aventure, Thriller	120	Edit Delete
	The Fantastic Four	English	Sci-Fi / Superhero	145	Edit Delete
	Thunderbolts*	English	Sci-Fi / Superhero	120	Edit Delete
	Superman	English	Action, Aventure, Science-Fiction	120	Edit Delete
	M3gan 2.0	English	Science-Fiction, Horreur	120	Edit Delete
	28 ans plus tard	English	Horreur, Thriller, Science-Fiction	120	Edit Delete

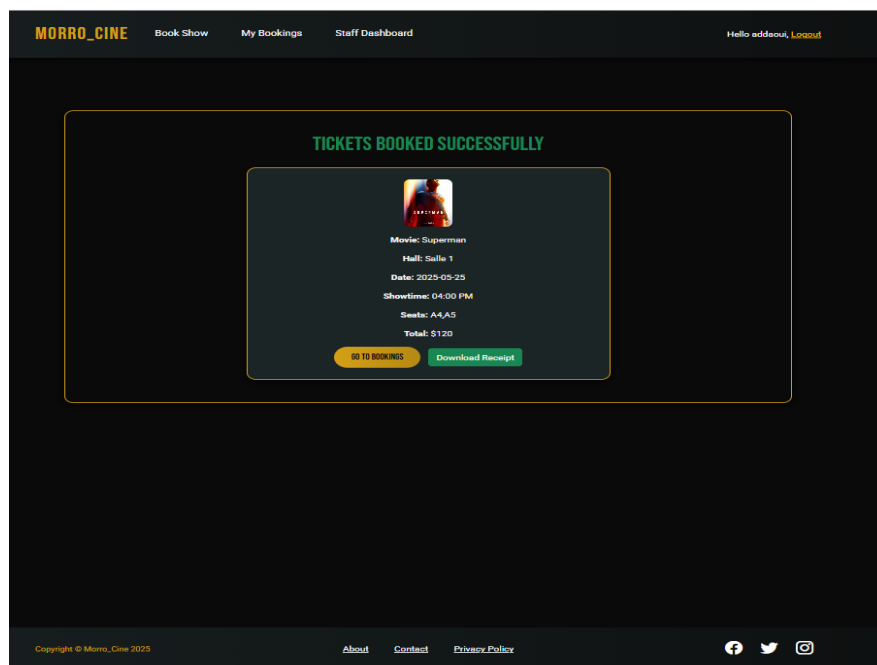
La page d'accueil affiche les bannières et les films disponibles. Les utilisateurs peuvent cliquer pour voir les détails d'un film ou choisir une séance.

The screenshot shows the 'MORRO_CINE' Shows section. It features an 'Add Show' button and a table listing movie screenings with columns for Movie, Hall, Showtime, Duration, Price, Start Date, End Date, and Actions.

Movie	Hall	Showtime	Duration (min)	Price	Start Date	End Date	Actions
Thunderbolts*	Salle 1	11:00 AM	120	\$60	May 1, 2025	June 15, 2025	Edit Delete
The Fantastic Four	Salle 2	11:00 AM	145	\$60	May 1, 2025	June 15, 2025	Edit Delete
Mission impossible	Salle 1	01:00 PM	120	\$60	May 1, 2025	May 31, 2025	Edit Delete
Superman	Salle 1	04:00 PM	120	\$60	May 2, 2025	May 31, 2025	Edit Delete
M3gan 2.0	Salle 2	04:30 PM	120	\$60	May 2, 2025	May 23, 2025	Edit Delete
28 ans plus tard	Salle 2	08:00 PM	120	\$60	May 2, 2025	May 31, 2025	Edit Delete



Booking :



MovieTicket Booking Receipt

Movie:	Superman
Hall:	Salle 1
Date:	2025-05-25
Showtime:	04:00 PM
Seats:	A4,A5
Total:	\$120.00
User:	addaoui
Booking ID:	5
Booking Date:	2025-05-25

Thank you for booking with MovieTicket!

Contact us at support@movieticket.com

L'utilisateur sélectionne une date et une séance, choisit des sièges, et valide la réservation. Un reçu PDF peut être téléchargé après confirmation.

Chapitre 5 : Difficultés rencontrés :

Difficultés rencontrés :

- Gestion **des disponibilités en temps réel** : Éviter les conflits de réservation pour un même siège a nécessité une validation rigoureuse dans Booking.clean.
- Synchronisation **front-end/back-end** : Les mises à jour dynamiques (via AJAX pour booked_seats) ont posé des défis d'intégration.
- Système **de paiement** : L'intégration d'un système de paiement sécurisé a requis une attention particulière (non implémentée dans ce code).
- Gestion **des bases de données** : Structurer les relations entre Booking, Show, Film, et Salle pour des performances optimales a été complexe.
- Interface **utilisateur** : Concevoir une interface responsive et intuitive a nécessité plusieurs itérations.

Chapitre 6 : Perspectives d'Évolution :

Perspectives d'Évolution :

- Système de paiement en ligne : Intégrer un service comme Stripe pour permettre des paiements sécurisés.
- Notifications : Ajouter des emails/SMS pour confirmer les réservations ou signaler des changements.
- Multilingue : Implémenter un système multilingue pour une accessibilité internationale.
- Optimisation des performances : Utiliser la mise en cache (ex. Redis) et optimiser les requêtes SQL pour gérer une forte affluence.
- Tests unitaires : Ajouter des tests pour garantir la robustesse du système.

Conclusion :

Conclusion :

Ce projet a permis de développer une application web complète de réservation de cinéma avec Django, répondant aux besoins des utilisateurs et des administrateurs. Le système gère l'authentification personnalisée, la réservation de sièges avec validation, et la gestion des films/séances par le staff. Les fonctionnalités comme la génération de reçus PDF et l'utilisation d'AJAX améliorent l'expérience utilisateur. Malgré des défis techniques, le projet offre une base solide, extensible, et prête à intégrer des améliorations comme un système de paiement ou des notifications. Il illustre l'application pratique des concepts de développement web et de modélisation orientée objet.