Fichiers Python :

"models.py"

```
class Genre(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100)
```

La classe Genre est un modèle Django qui représente un genre de film dans la base de données. Elle a deux attributs : id, un champ auto-incrémenté qui sert de clé primaire, et name, un champ de caractères avec une longueur maximale de 100 caractères pour stocker le nom du genre.

```
class Movie(models.Model):
    id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=100)
    year = models.CharField(help_text="Release year", max_length=100)
    genre = models.ManyToManyField(Genre, blank=True)
    poster = models.URLField(max_length=200, default="https://via.placeholder.com/150")
    plot = models.TextField(blank=True)
```

La classe Movie est un modèle Django qui représente un film. Elle a les attributs suivants : id (clé primaire), title (titre du film), year (année de sortie), genre (relation plusieurs-à-plusieurs avec le modèle Genre), poster (URL de l'affiche du film), et plot (intrigue du film).

```
    def set_genre(self, genre):
        genre_array = genre.split(",")
        for genre_name in genre_array:
            genre_obj, _ = Genre.objects.get_or_create(name=genre_name)
            self.genre.add(genre_obj.id)
```

La méthode set_genre(self, genre) définit le genre d'un film. Elle divise une chaîne de genres en un tableau, crée ou récupère chaque genre de la base de données, puis associe ces genres au film.

"forms.py":

```
class MovieForm(forms.ModelForm):
    class Meta:
        model = Movie
        fields = ["title"]
        widgets = {
            "title": forms.TextInput(
                attrs={
                    "class": "px-4 py-2 rounded-md focus:outline-none focus:ring-2
focus:ring-blue-500 text-black"
                }
            ),
        }
```

Classe de formulaire pour la recherche d'un film, on fait la recherche uniquement sur le titre et on lui donne un style grâce au Framework CSS Tailwind

"views.py":

```
def index(request):
    if request.method == "GET":
        context = {"form": MovieForm()}
        return render(request, "index.html", context)
    elif request.method == "POST":
        form = MovieForm(request.POST)
        if form.is_valid():
            form.save(commit=False)
            title = form.cleaned_data["title"]
            movies = get_movies(title)
            return get_movies_search(request, movies)
    return HttpResponse("Erreur")
```

Fonction pour afficher la page d'accueil, si la méthode de la requête HTTP est 'GET' on amène l'utilisateur sur la page de recherche, si la méthode est 'POST' on récupère la valeur entré par l'utilisateur sur la barre de recherche.
Puis on appelle la fonction get_movies() pour récupérer une liste de film contenant le mot cherché par l'utilisateur.
On amène alors l'utilisateur sur la page de résultat qui va afficher la liste des films.
Si la méthode est ni un 'GET' ni un 'POST' on affiche une erreur

```
def get_movies_search(request, movies):
    if movies is None or len(movies) == 0:
        return HttpResponse("No movies found")
    # MovieForm() is passed to the template to display the search form
    context = {"movies": movies, "form": MovieForm()}
    return render(request, "movies.html", context)
```

Fonction qui va vérifier si get_movies() a bien retourné une liste de films dans le cas contraire on affiche une erreur disant qu'aucun film a été trouvé

```
def movie_page(request, movie_id):
    movie = get_movie(movie_id)

    if movie:
        context = {"movie": movie, "form": MovieForm()}
```

```
        return render(request, "movie_page.html", context)
    return HttpResponse("No movie found.")
```

Fonction qui va amener sur une page détails d'un film, on a en paramètre l'id du film qui va nous permettre de faire la requête API, la variable movie est égale au film sélectionné, si cette variable n'est pas vide on amène l'utilisateur sur la page détail du film dans le cas contraire on retourne une erreur.

"url.py"

```
urlpatterns = [
    path("", views.index, name="index"),
    path('search', views.get_movies_search, name="search"),
    path("<int:movie_id>/", views.movie_page, name="movie_page"),
]
```

urlpatterns: Liste des URL de l'application.
L'URL racine est liée à la vue index, l'URL 'search' est liée à la vue get_movies_search, et l'URL avec le paramètre entier 'movie_id' est liée à la vue movie_page.
<int:movie_id> est un paramètre dans l'URL qui attend une valeur entière. Il est utilisé pour passer l'ID spécifique d'un film à la vue movie_page.

"service.py"

```
load_dotenv()
api_key = os.getenv("API_KEY")
```

La fonction load_dotenv() charge les variables d'environnement à partir d'un fichier .env, et api_key = os.getenv("API_KEY") récupère la valeur de la variable d'environnement "API_KEY" (La clé API de OMDb API).

```
def get_data_from_api(endpoint: str, title: str) -> Union[List[dict], dict]:
    url = f"https://www.omdbapi.com/?{endpoint}={title}&apikey={api_key}"
    try:
        response = requests.get(url)
        response.raise_for_status()
        if endpoint == "s":
            return response.json()["Search"]
        else:
            return response.json()
    except requests.exceptions.RequestException as e:
        logging.error(f"Request to {url} failed: {e}")
        return [] if endpoint == "s" else {}
```

La fonction get_data_from_api(endpoint: str, title: str) récupère des données de l'API OMDB. Elle construit l'URL de l'API en utilisant le point d'accès et le titre fournis, envoie une requête GET et renvoie la réponse sous forme de liste ou de dictionnaire. Si la requête échoue, elle enregistre l'erreur et renvoie une liste ou un dictionnaire vide.

```python
def get_movies(title: str) -> List[Movie]:
    try:
        db_movies = Movie.objects.filter(title__icontains=title)
        if not db_movies:
            movies = get_data_from_api("s", title)
            movies_detailed: List[Movie] = []
            for movie in movies:
                movie_data = get_data_from_api("t", movie["Title"])
                if movie_data:
                    movie: Movie = save_movie_to_db(movie_data)
                    movies_detailed.append(movie)
            return movies_detailed
        return list(db_movies)
    except Exception as e:
        logging.error(f"An error occurred while retrieving movies: {e}")
        return []
```

La fonction get_movies(title: str) récupère les films dont le titre correspond à la chaîne fournie. Elle vérifie d'abord la base de données pour trouver des films correspondants en utilisant Movie.objects.filter(title__icontains=title). Si aucun n'est trouvé, elle récupère les données de l'API OMDB en utilisant get_data_from_api("s", title), sauvegarde les films récupérés dans la base de données en utilisant save_movie_to_db(movie_data), et renvoie une liste d'objets de films détaillés. Si une erreur se produit pendant ce processus, elle enregistre l'erreur et renvoie une liste vide.

```python
def get_movie(id: int) -> Movie:
    try:
        movie = Movie.objects.get(id=id)
        return movie
    except Exception as e:
        logging.error(f"An error occurred while retrieving movie: {e}")
        return None
```

La fonction get_movie(id: int) récupère un film spécifique en utilisant son identifiant. Elle utilise la méthode Movie.objects.get(id=id) pour rechercher le film dans la base de données. Si une erreur se produit pendant ce processus, elle enregistre l'erreur et renvoie None.

```python
def save_movie_to_db(movie_data: dict) -> Movie:
    try:
        if movie_data:
            movie: Movie = Movie.objects.create(
                title=movie_data["Title"],
                year=movie_data["Year"],
                poster=movie_data["Poster"],
                plot=movie_data["Plot"],
            )
            movie.set_genre(movie_data["Genre"])
            movie.save()
```

```
            return movie
        else:
            return None
    except Exception as e:
        logging.error(f"An error occurred while saving movie to the database: {e}")
        return None
```

La fonction save_movie_to_db(movie_data: dict) sauvegarde un film dans la base de données. Elle utilise la méthode Movie.objects.create() pour créer un nouvel objet film avec les données fournies. Ensuite, elle définit le genre du film avec movie.set_genre(movie_data["Genre"]) et sauvegarde le film dans la base de données avec movie.save(). Si une erreur se produit pendant ce processus, elle enregistre l'erreur et renvoie None.

Fichiers Templates HTML :

'index.html':

```
<!DOCTYPE html>
<html>
  <head>
    <link
      href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
      rel="stylesheet"
    />
  </head>
  <body class="bg-gray-200">
    <div class="flex items-center justify-between p-4 bg-gray-900 text-white">
      <div>
        <a href="{% url 'index' %} " class="text-xl font-bold">MOVIE MERCADO / MASSE</a>
      </div>
      {% include 'form.html' %}
    </div>

    <main>{% block content %} {% endblock %}</main>
  </body>
</html>
```

Cette page HTML est le modèle principal. Elle comprend une barre de navigation avec un titre et un formulaire de recherche ({% include 'form.html' %}), et un espace pour le contenu principal ({% block content %}) qui est défini par des modèles enfants Django.. Le style de la page est géré par Tailwind CSS.

'form.html':

```
<div>
  <form method="POST" action="{% url 'index' %}">
    <div class="flex items-center">
      <div>{% csrf_token %} {{ form }}</div>
      <div>
        <button
          type="submit"
          class="ml-2 px-4 py-2 bg-blue-500 text-white rounded-md hover:bg-blue-600
focus:outline-none focus:ring-2 focus:ring-blue-500"
        >
          Search
        </button>
      </div>
    </div>
  </form>
</div>
```

Ce code HTML représente un formulaire de recherche de films. Il utilise la méthode POST pour envoyer les données de recherche à la vue 'index' de Django. Le formulaire comprend un token CSRF pour la sécurité et un bouton "Search" pour soumettre le formulaire. Ce code est inclus dans index.html.

'movies.html':

```
{% extends 'index.html' %} {% block content %} {% csrf_token %}
<body>
  <div
    class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-3 gap-4 p-4"
  >
    {% for movie in movies %}
    <a href="{% url 'movie_page' movie.id %}">
      <div
        class="bg-white rounded-lg shadow-md overflow-hidden hover:bg-gray-200 transition
duration-300"
      >
        <img
          class="w-full h-48 object-cover"
          src="{{movie.poster}}"
          alt="{{movie.title}}"
        />
        <div class="p-4">
          <h1 class="text-xl font-bold">{{movie.title}}</h1>
          <h2 class="text-gray-600">Année de sortie: {{movie.year}}</h2>
        </div>
      </div>
    </a>
    {% empty %}
    <h1>No Movie</h1>
    {% endfor %}
  </div>
  {% endblock %}
```

```
</body>
```

Page pour afficher la liste des films en grid, cette page hérite d'index.html et on vient boucler sur une liste de films pour les afficher. Sur cette page on affiche uniquement l'affiche, le titre et l'année de sortie d'un film.
Si on clique sur un film, on récupère l'id de ce dernier et on amène l'utilisateur sur la page de détail.

'movie_page.html':

```
{% extends 'index.html' %} {% block content %} {% csrf_token %}
<div class="flex justify-center items-center">
  <div class="w-2/4 h-2/4 flex flex-row items-center justify-evenly p-4 m-4 bg-white rounded-lg
shadow-md p-4 h-96 text-center">
    {% if movie.poster %}
    <img
      class="w-64 h-96 object-cover rounded-lg shadow-md"
      src="{{ movie.poster }}"
      alt="{{ movie.title }} Poster"
    />
    {% endif %}
    <div class="flex flex-col justify-center items-center">
      <h1 class="text-3xl font-bold mt-4">{{ movie.title }}</h1>
      <p class="text-lg text-gray-600 mt-2">{{ movie.year }}</p>
      {% if movie.genre %}
      <div class="flex flex-wrap mt-2">
        {% for single_genre in movie.genre.all %}
        <span class="text-sm text-gray-500 bg-gray-200 rounded-full px-2 py-1 m-1"
          >{{ single_genre.name }}</span
        >
        {% endfor %}
      </div>
      {% endif %}
    </div>
  </div>
</div>
  {%endblock%}
```

Page pour afficher les détails d'un film, cette page hérite d'index.html et on vient afficher différentes informations sur le film comme son titre, son affiche son année de sortie et son/ses genre(s).