

Conception d'une application d'affichage de films et de séries

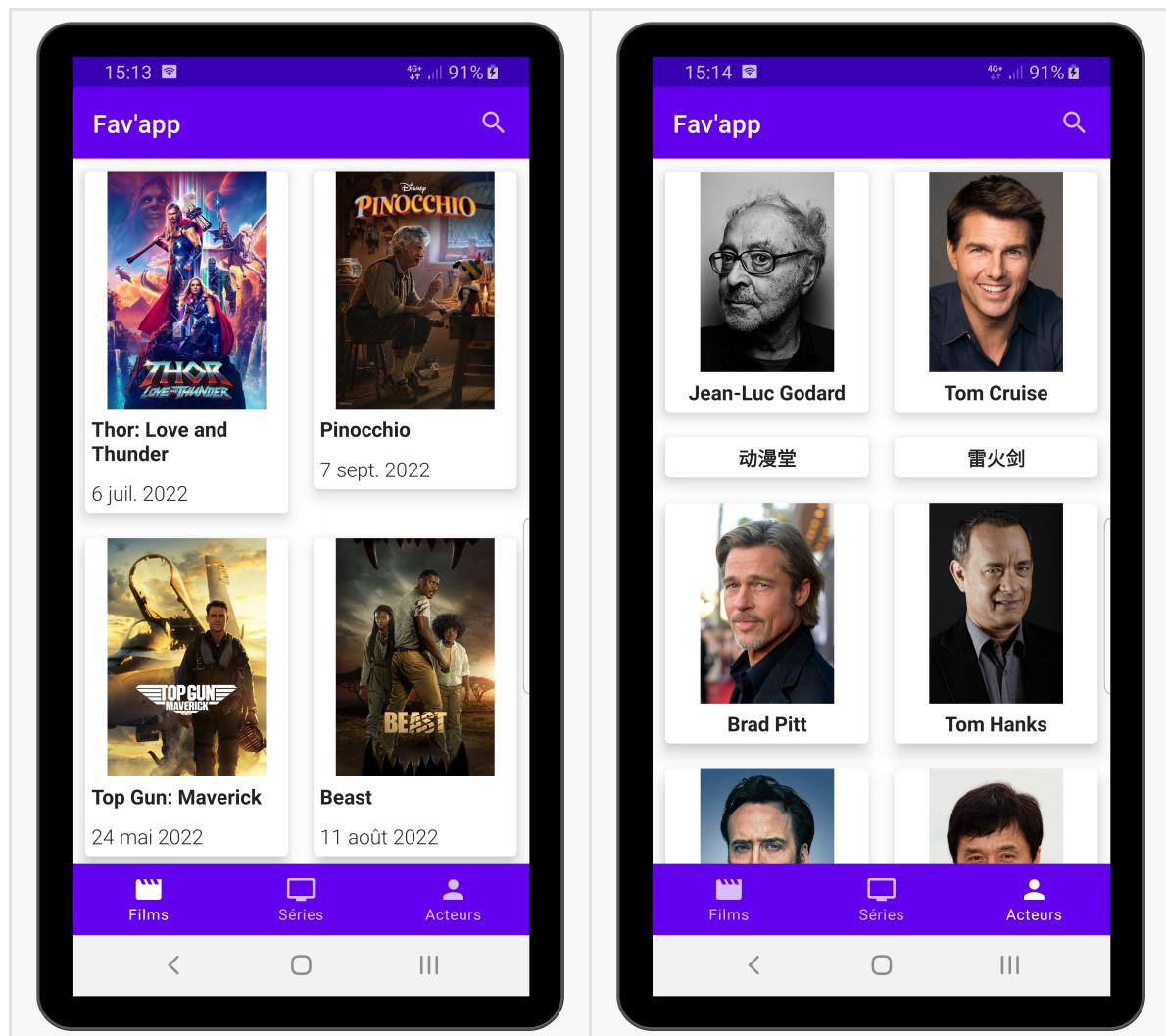
Nicolas Singer

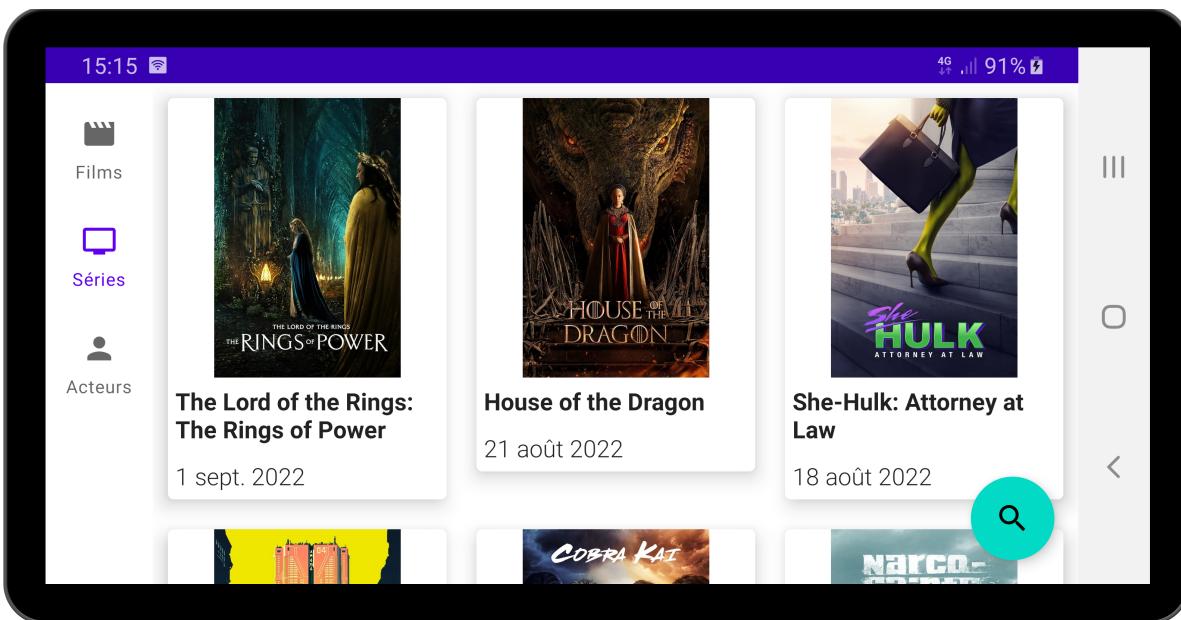
1. Objectif

On reprend l'application présentant votre profil, et on fait en sorte qu'un click sur son bouton *Démarrer* mène à un nouvel écran permettant de naviguer au sein de films et de séries dont les informations proviennent du site [The Movie Database \(TMDB\) \(themoviedb.org\)](https://www.themoviedb.org).

L'écran doit présenter une barre de navigation permettant de sélectionner l'affichage des films, séries ou personnalités du cinéma, ainsi que la possibilité de saisir un mot clé pour filtrer les médias affichés. Par défaut, ce sont les derniers médias et personnalités populaires qui sont affichés.

Les images ci-dessous illustrent les différents types de présentation. Vous noterez que le layout est adaptatif et positionne différemment la barre de navigation et le champ de recherche selon l'orientation de l'écran.





Un clic sur une série ou un film, mène à sa description détaillée comme ci-dessous. En dessous du synopsis (en scrollant), on doit voir défiler la distribution du film.

Synopsis

Divers personnages, connus et inconnus, doivent faire face à la résurgence du mal en Terre du Milieu. Des profondeurs des Montagnes de Brume aux forêts majestueuses du Lindon, de l'île des Rois de Númenor aux contrées les plus éloignées du monde, ces royaumes et personnages vont forger un héritage qui demeurera bien après leur disparition. serveur : <https://divedigital.vip/tv/84773/the-lord-of-the-rings-the-rings-of-power>

Synopsis

Divers personnages, connus et inconnus, doivent faire face à la résurgence du mal en Terre du Milieu. Des profondeurs des Montagnes de Brume aux forêts majestueuses du Lindon, de l'île des Rois de Númenor aux contrées les plus éloignées du monde, ces royaumes et personnages vont forger un héritage qui

Têtes d'affiche

Markella Kavenagh
Elanor 'Nori'
Brandyfoot

Joseph Mawle
Adar

2. Mise en place du routage entre écrans

On vous conseille de regrouper les composants nécessaires à chaque type d'écran dans un fichier distinct. Par exemple créez un fichier Profil.kt pour contenir tous les éléments de la présentation

Faites de même avec dans un premier temps un fichier `Films.kt` doté d'un composant appelé `Films()` vide.

Mettez en place le contrôleur de navigation, doté pour l'instant de deux destinations : `Profile` et `Films`. Faites en sorte que quand on clique sur le bouton *Démarrer* de votre profil, on change de destination pour atteindre le composant `Films()`. Pour réaliser cela, vous pouvez consulter la documentation officielle sur la composant de navigation de Jetpack Compose : <https://developer.android.com/jetpack/compose/navigation>. Je vous conseille de créer le composant de navigation et sa table de routage dans le `setContent { }` de `MainActivity`.

Progressivement, vous mettrez en place les autres écrans de l'application qui affichent respectivement la liste des séries, la liste des personnes, le film au format détaillé et la série au format détaillé. Ces nouvelles destinations viendront s'ajouter à votre table de routage.

3. Conception de l'écran d'affichage de la liste des films

Ici il faut utiliser la bibliothèque **Retrofit** pour aller chercher les informations concernant les films à afficher. Les étapes à réaliser sont les suivantes :

- Ajouter la dépendance de la bibliothèque Retrofit à votre projet, ainsi que son déserialisateur JSON (Moshi par exemple). Vous pouvez vous référer au cours : <https://nicolessinger.gitlab.io/revealjs/talks/Android/retrofit/>
- Modifiez le fichier `Manifest.xml` situé à la racine de votre projet pour autoriser les requêtes réseau par votre application. Pour cela ajoutez dans la balise l'élément suivant :

```
<uses-permission android:name="android.permission.INTERNET" />
```

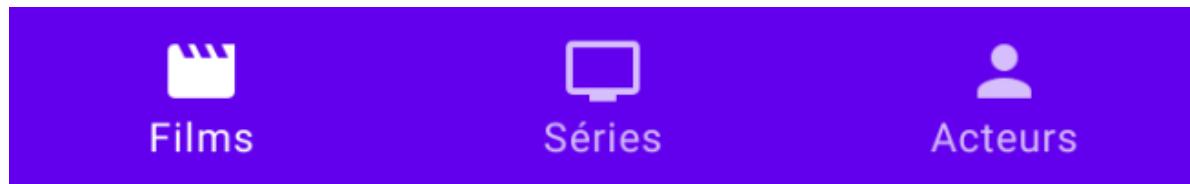
- A partir des informations au format JSON retournées par TMDb pour une liste de films, créez les classes qui correspondent en Kotlin. Pour allez plus vite, vous pouvez utiliser le plugin *JSON To Kotlin Class* qui est installable sur Android Studio. Une fois ce plugin installé, créez un nouveau fichier pour contenir vos classes modèles (vous pouvez l'appeler `Model.kt`) et dans le fichier utilisez le menu contextuel *Generate...* qui vous propose un sous-menu *Kotlin Data Classes from JSON*. Ce menu ouvre une fenêtre dans laquelle vous pouvez coller le JSON à transformer en classes Kotlin.
- Créez un nouveau fichier (vous pouvez l'appeler `TmdbAPI.kt`) pour implémenter l'interface Retrofit de requête pour la liste des films. La aussi pour définir cette interface, reportez vous au cours sur Retrofit. Au départ, elle ne contiendra que la méthode permettant d'aller chercher la liste des films de la semaine.
- Créez un nouveau fichier pour définir un ViewModel que vous relierez à votre activité (vous pouvez l'appeler `MainViewModel.kt`). Dotez ce modèle d'une propriété de type `MutableStateFlow` (appelons là `movies`) pour encapsuler la liste de vos films. Faites en sorte que le ViewModel initialise le service Retrofit et dotez le d'une méthode `getFilmsInitiaux()` qui appelle ce service pour allez chercher la liste des films et qui la place dans la propriété `value` de la propriété `movies`.
- Implémenter le composant `Films()` en lui passant en paramètre le ViewModel et en le dotant d'une propriété (on peut aussi l'appeler `movies`) obtenue par collecte de l'état `movies` du ViewModel. Si la liste des films est vide, le composant doit appeler la fonction

composant sera rafraîchi, il demandera le téléchargement de la liste, ce qui provoquera à nouveau son rafraîchissement, et donc une boucle infinie).

- Dotez le composant `Films()` d'une interface graphique présentant pour l'instant uniquement une liste des titres des films. Vous devez pour cela utiliser le composant prédefini `LazyVerticalGrid` comme expliqué dans le cours : <https://nicolassinger.gitlab.io/revealjs/talks/Android/lazyList/>
- Vous pouvez à présent tester votre application et vérifiez qu'elle affiche bien la liste des titres de films. N'oubliez pas que vous pouvez utiliser l'onglet *App inspection* et sa partie *Network Monitor* pour observer et vérifier les requêtes réseaux envoyées par votre application.
- Enfin, utilisez la bibliothèque **Coil** (<https://coil-kt.github.io/coil/>) pour raffiner l'affichage pour qu'en plus du titre, apparaisse l'image de son affiche. Référez-vous à la partie sur *Coil* dans le cours du compose pour ajouter la dépendance nécessaire et utiliser Coil : <https://nicolassinger.gitlab.io/revealjs/talks/Android/compose>
- Terminez la mise en page en ajoutant la date du film. Adaptez le design selon vos goûts.

4. Ajout de la barre de navigation

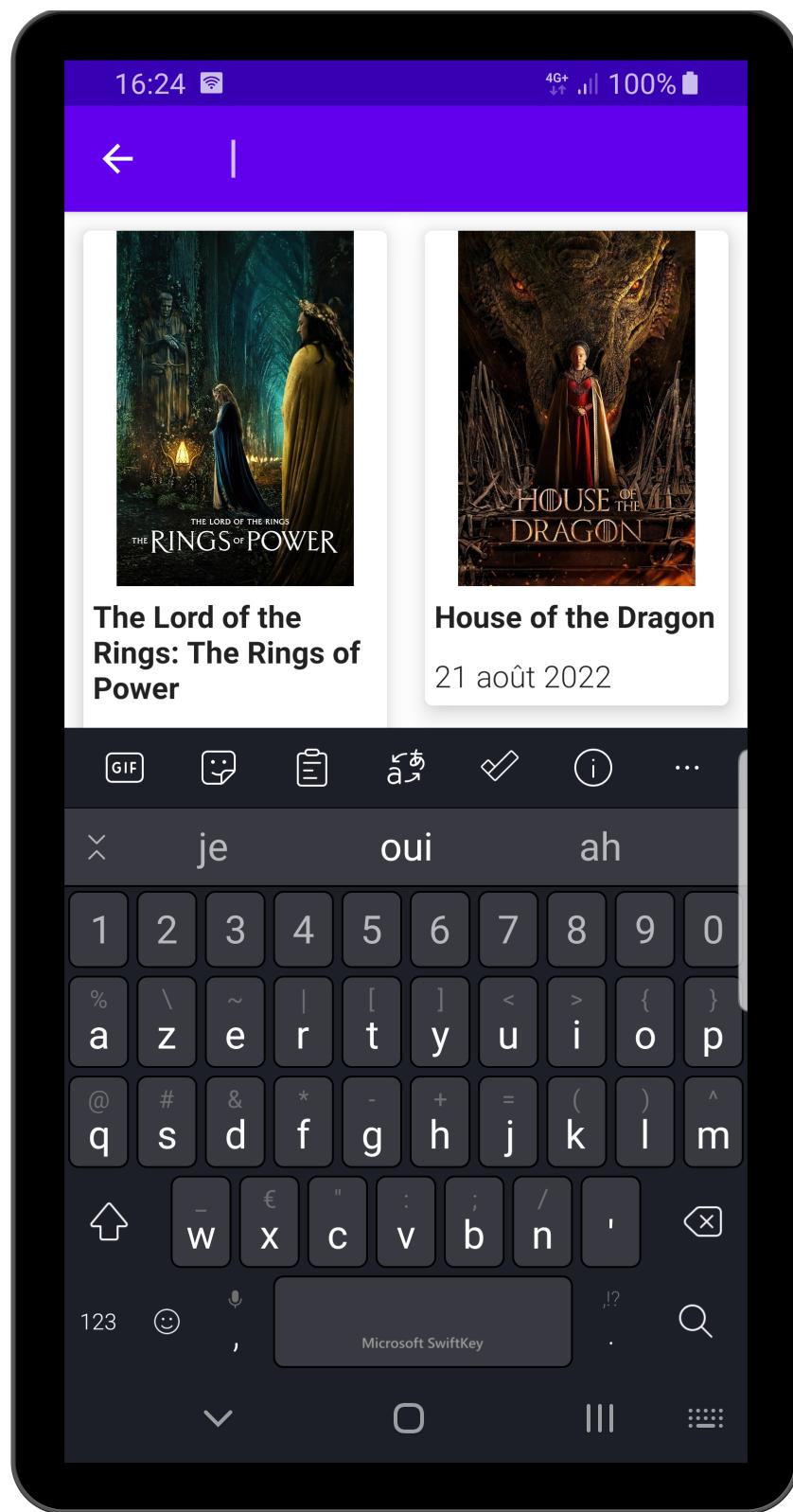
Nous voulons ici ajouter une barre de navigation en bas de l'écran d'affichage des films. Cette barre sera présente sur tous vos autres écrans, à l'exception de votre écran de profil.



En mode portatif, le plus simple pour placer cette barre est d'utiliser le composant prédefini `scaffold` qui décompose l'écran en trois parties, une TopBar, un contenu central, et une BottomBar. Dans la partie BottomBar, vous pouvez définir la barre de navigation en suivant les explications données dans la partie *Integration with the bottom nav bar* de ce lien : <https://developer.android.com/jetpack/compose/navigation>.

5. Ajout du champ de recherche

Ici nous voulons ajouter la possibilité de lancer une recherche qui filtrera la liste des films en fonction du mot clé saisi. Vous êtes libres de gérer ce champ de recherche comme vous le souhaitez. On peut par exemple créer une TopBar contenant une icône de recherche (la loupe, voir les écrans d'exemple donné au début de l'énoncé) et faire en sorte que la TopBar se transforme en champ de saisie de texte en cas de click sur l'icône loupe, comme ci-dessous :



Puis après validation du mot clé, la TopBar présente à nouveau sa forme d'origine avec la loupe. Une autre façon de faire si on ne veut pas utiliser de TopBar est d'utiliser un bouton flottant qui en cas de clic ouvre un champ de recherche (voir l'exemple en mode paysage au début de l'énoncé).

Dans tous les cas, pour gérer la recherche par mot clé, il faudra ajouter à votre interface Retrofit (`TmdbApi.kt`) une nouvelle méthode permettant d'aller chercher une liste de films par mot clé, puis ajouter la méthode correspondant dans le ViewModel, et enfin faire en sorte que la validation du mot clé dans l'interface graphique, appelle cette méthode du ViewModel. Comme votre composant observe le `MutableStateFlow` `movies`, dès que le ViewModel aura changé sa

6. Création des écrans des séries et des personnalités

Le plus dur est fait, la création de ces deux autres écrans reprend les bases de ce que nous avons vu pour la liste des films. Il faudra bien entendu créer deux fichiers supplémentaires que l'on pourra appeler `series.kt` et `personnes.kt` dotés des composants nécessaires. Il faudra aussi ajouter dans l'interface Retrofit, les méthodes pour aller chercher les dernières séries et personnalités de la semaine, puis les méthodes pour aller chercher les séries et personnalités par mot clé. Idem pour le ViewModel.

7. Crédit des écrans des détails des films et des séries

On souhaite qu'un clic sur un film ou sur une série de la liste affichée, mène à un écran affichant ce film ou cette série de façon détaillée. Il faudra donc créer deux écrans (fichiers et composants) supplémentaires pour cela. On pourra appeler les fichiers `movie.kt` et `serie.kt`. A vous de jouer pour présenter vos films le plus joliment possible.

8. Adapter la présentation à la taille de l'écran

On veut ici adapter l'interface à la taille de l'écran et à sa présentation. Parmi les adaptations possibles, on peut pour les listes augmenter le nombre de colonne de la grille en fonction de la largeur d'écran disponible. On peut aussi positionner la barre de navigation sur le coté, si l'espace vertical est étroit. Dans ce cas, vous pouvez utiliser le composant `sideNavigation` de *material 3* à la place du composant `scaffold` pour positionner la barre de navigation. On peut aussi changer le sens du scroll selon l'orientation de l'écran en utilisant tantôt une `LazyHorizontalGrid`, tantôt une `LazyVerticalGrid`, à vous de voir quelle solution est la plus agréable pour l'ergonomie de l'application .