

<u>FilmItem</u> <ul style="list-style-type: none"> + <u>displayFavoriteImage()</u> <ul style="list-style-type: none"> • Affiche le cœur à côté du nom si le film est en favori + <u>render()</u> <ul style="list-style-type: none"> • Rend le composant • Responsable de l'agencement des différentes parties 	<u>Search</u> <ul style="list-style-type: none"> + <u>constructor (props)</u> <ul style="list-style-type: none"> • Crée le composant Search • Initialise les instance de son props et son state + <u>searchTextInputChange (text)</u> <ul style="list-style-type: none"> • Actualise l'instance 'searchText' avec ce que l'utilisateur tape. + <u>displayDetailForFilm (idFilm)</u> <ul style="list-style-type: none"> • Dirige vers la fiche détail du film + <u>displayLoading ()</u> <ul style="list-style-type: none"> • Affiche le logo chargem^t p^tt 1 chargem^t + <u>loadFilms ()</u> <ul style="list-style-type: none"> • Lance la récup^r des films dans l'API • Met à jour les instances de Search (props & state) avec la liste récupérée + <u>searchFilms ()</u> <ul style="list-style-type: none"> • Reset les instance de Search • exécute <u>loadFilms()</u> + <u>render ()</u> <ul style="list-style-type: none"> • Rend le composant, affiche une case texte et un bouton Rechercher • Affiche une FilmList de films recherchés 	<u>Navigat°</u> <ul style="list-style-type: none"> + <u>SearchStackNavigator</u> <ul style="list-style-type: none"> • Gère la navigat° entre les composants Search et FilmDetail + <u>FavoritesStackNavigator</u> <ul style="list-style-type: none"> • Gère la navigat° entre les composants Favorites et FilmDetail + <u>MoviesTabNavigator</u> <ul style="list-style-type: none"> • Gère la navigat° entre les navigat° SearchStackNavigator et FavoritesStackNavigator + <u>createAppContainer (MoviesTabNavigator)</u> <ul style="list-style-type: none"> • Détermine la navigat° principale • Exporte la navigat° principale 	<u>FilmDetail</u> <ul style="list-style-type: none"> + <u>constructor (props)</u> <ul style="list-style-type: none"> • Crée le composant • Initialise le state + <u>displayLoading ()</u> <ul style="list-style-type: none"> • Affiche le logo de chargem^t pendant un chargem^t + <u>ComponentDidMount ()</u> <ul style="list-style-type: none"> • Actualise le state de FilmDetail avec les données du film choisi • Récupère les données ou bien de la liste des favoris, ou bien de l'API + <u>displayFilm ()</u> <ul style="list-style-type: none"> • Responsable de l'agencem^t des parties dans la vue + <u>toggleFavorite ()</u> <ul style="list-style-type: none"> • Rend le film favori • Envoie une act° au Store + <u>displayFavoriteImage ()</u> <ul style="list-style-type: none"> • Affiche l'icone favori en p^tt de si le film est favori ou non + <u>render ()</u> <ul style="list-style-type: none"> • Rend le composant • Appelle la méthode <u>displayFilm()</u> + <u>connect ()</u>
<u>TMDBApi</u> <ul style="list-style-type: none"> + <u>getImageFromApi (name)</u> <ul style="list-style-type: none"> • Cherche l'image choisie dans l'API + <u>getFilmsFromApiWithSearchedText (text, page)</u> <ul style="list-style-type: none"> • Récupère la liste de films à partir du texte entré, dans l'API + <u>getFilmDetailFromApi (id)</u> <ul style="list-style-type: none"> • Récupère le détail du film id dans l'API 	<u>FilmList</u> <ul style="list-style-type: none"> + <u>constructor (props)</u> <ul style="list-style-type: none"> • Crée le composant FilmList • Initialise son state + <u>displayDetailForFilm (idFilm)</u> <ul style="list-style-type: none"> • Dirige vers la fiche détail du film + <u>render ()</u> <ul style="list-style-type: none"> • Rend le composant • Retourne les données de films entrés sous forme d'une liste de FilmItem + <u>connect ()</u> 	<u>Store</u> <div> <u>Reducers</u> <ul style="list-style-type: none"> + <u>toggleFavorite (state, act°)</u> <ul style="list-style-type: none"> • Gère les act° liées au toggle Favorite </div> <ul style="list-style-type: none"> + <u>createStore (toggleFavorite)</u> <ul style="list-style-type: none"> • Crée le store 	<u>App</u> <ul style="list-style-type: none"> + <u>render ()</u> <ul style="list-style-type: none"> • Rendu final • Appelle le Store et Navigat°
<u>Favorites</u> <ul style="list-style-type: none"> + <u>render ()</u> <ul style="list-style-type: none"> • Rend le composant • Affiche la liste des films favoris + <u>connect ()</u> <ul style="list-style-type: none"> • Abonné à l'élémt favoritesFilm du Store 			

```

Favorites
+render () {
  return (
    <FilmList
      films={this.props.favoritesFilm}
      navigat*={this.props.navigat*}
      favoritesList={true}
    />
  )
}

mapStateToProps = state => {
  return {
    favoritesFilm: state.favoritesFilm
  }
}

export default connect(mapStateToProps)(Favorites)

```

```

FilmList {
  constructor (props) {
    super (props)
    this.state = { films: [] }
  }
  // displayDefaultFilm a (idFilm) => {
  //   this.props.navigate ('FilmDetail', {idFilm: idFilm})
  // }
  render () { return (
    <FilmList
      data={this.props.films} extraData={this.props.films}
    >
      <Extractor : {film} => {film.id, filmString ()}
      <RenderItem : {film} => {
        <FilmItem film={film}
          <FilmFavorite {film.favorite, findIndex (film =>
            id === item.id, item - 1) } true false />
          <displayDefaultFilm {film.id} />
        </>
      </>
      <endReachedThreshold : (0.5) />
      <endReached : () => {
        <{film.favorite} BB : page <{film.totalPages} />
        <{film.props.loadFilms ()} />
      </>
    </>
  ) }
}
export default connect (mapStateToProps) (FilmList)

```

```

Store > Reducers > FavoriteReducer
+ const initialState = { favoriteFilm: [] }
+ toggleFavorite: (state: initialState, action) {
  let nextState
  switch (action.type) {
    case 'TOGGLE_FAV':
      FavoriteFilmIndex = s.FF.findIndex(item => item.id
        === action.value.id)
      if (FFI !== -1) {
        nextState = { ...state,
          FF: s.FF.filter((item,index) => index !== FFI) }
      } else {
        nextState = { ...state, FF: [...s.FF, action.value] }
      }
      return nextState // state
    default:
      return state
  }
}
+ export default createStore (toggleFavorite)

```

```

+ FilmDetail {
+ constructor (props) {
+   super (props)
+   this.state = { film: undefined, isLoading: false }
+ }
+ displayLoading () {
+   // idem Search
+ }
+ componentDidMount () {
+   favoriteFilmIndex: t.p.favoriteFilm.findIndex (
+     item => item.id === t.p.navgate.state.params.idFilm
+   )
+   if ( favoriteFilmIndex !== -1 ) {
+     this.setState (
+       { film: t.p.favoriteFilm [ favoriteFI ] } )
+   }
+   return
+   this.setState ( { isLoading: true } )
+   getPDFApi (t.p.nav.p.idFilm).then ( data =>
+     this.setState ( {
+       film: data
+       isLoading: false
+     })
+   )
+ }
+ displayFilm () {
+   const { film } = this.state
+   if ( film != undefined ) {
+     return (
+       <ScrollView>
+         <Img source={ {uri: getIFApi (film.backdrop)} } />
+         <Title> display: { () => this._toggleFavorite () }
+         { this._displayFavoriteImage () } (S)
+         <TouchableOpacity>
+           <Text> { film.title } </Text>
+           :
+           <Text> Note: { film.average } / 70 </Text>
+         </ScrollView>
+       )
+   }
+ }
+ _toggleFavorite () {
+   const action = { type: 'TOGGLE_FAV', value: t.s.g.film }
+   t.p.dispatch (action)
+ }
+ displayFavoriteImage () {
+   t.sImg = require ( './ic_favorite_holding.png' )
+   if ( t.p.favoriteFilm.findIndex ( item => item.id === t.s.film.id ) !== -1 ) {
+     t.sImg = require ( './ic_favorite.png' )
+   }
+   return (
+     <img source={ t.sImg } />
+   )
+ }
+ render () {
+   return (
+     <View style={ styles.main_container }>
+       { this._displayLoading () }
+       { this._displayFilm () }
+     </View>
+   )
+ }
+ }
Export default connect (mapStateToProps) (FilmDetail)

```


FilmItem.js

```

+ render ()
  const film = this.props.film
  return (
    <View>
      <Image />
    </View>
    <View>
      <Text> {film.title} </Text>
    </View>
    <View>
      <Text> {film.overview} </Text>
      <Text> {film.vote} </Text>
    </View>
  )

```

retourne
la
vue
du film F

Search.js

```

+ Constructor (props)
  * Initialise le composant
  avec les propriétés souhaitées
  - this.searchText = ''
  - this.state = { films: [] }

+ _loadFilms ()
  * Charge la liste de films
  souhaitée
  - fetchAPI(this.searchText).then(
    data => { this.setState({ films: data.results }) }
  )

+ SearchTextInputChanged (text)
  * Instancie la prop de l'objet
  au texte inséré
  - this.searchedText = text

+ render ()
  - <TI opT = { (text) => this._STIC(text) } />
  - <B op = { () => this._loadFilms() } />
  - <FL data = { this.state.films }
    idKeyF = { (item) => item.id.toString() }
    renderItem = [{ (item) => <FilmItem film={item} /> }
  ]

```

TMDBA.js

```

+ fetchAPI (text)
  * récupère films via API
  - url = '...'
  - return fetch(url)
    .then(res => res.json())
    .catch(...)

```

retourne gros
fichier json

récupère l'array appelé "results"
du json

MAS texte
dans l'appli.

lance la
recherche

retourne le
composant

App.js

```

+ render
  return
    <Search />

```