ÉCOLE D'INGÉNIEUR-E-S
Sceaux | Troyes | Montpellier

Projet Cinémathèque

Contributeurs: JIANG Yilun, KANG Zhuodong, WANG Haoyu

Site GitHub: Marshellson/Cinematheque

Sommaire

Contexte du développement

Dans les années 1830, le cinéma entame sa période de préparation technologique prénatale, appelée aussi période d'invention, avec l'invention de la machine de cinéma optique par Renault en 1888, l'invention du cinématographe par Edison en 1889, et en 1895, alors que l'image se joue lentement, les frères Lumière réussissent à inventer la première génération d'appareils de projection cinématographique, une grande invention dans laquelle le film peut être vu par de nombreuses personnes ensemble et qu'il peut offrir à chacun une expérience à la fois inégalée et étonnante.

Et avec le développement des temps, la technologie cinématographique progresse, les effets des films sont plus réalistes, comme la 3D, la présentation des films et les appareils qui les lisent augmentent, et les films sont sauvegardés de plus en plus de manières, comme le Blu-ray, le DVD, etc., ce qui permet à un plus grand nombre de personnes de regarder des films, et la variété des films est plus abondante, mais comme de plus en plus de films sont réalisés dans le monde,la diversité dérange les gens. Nous ne pouvons pas nous souvenir des films que nous aimons, des films que nous avons regardés, des films que nous possédons, nous ne pouvons pas classer et organiser ces films de manière efficace, et il n'y a aucun programme sur le marché actuellement qui puisse à la fois intégrer sa propre bibliothèque de films et enregistrer les films qu'il possède et ceux qu'il a regardés.

Donc nous pensons qu'il y a un besoin pour un tel programme sur le marché aujourd'hui, et la conception de la programmation devient de plus en plus avancée de nos jours, et la base de données de films devient de plus en plus robuste, et les conditions sont très matures, et dans ce temps, nous avons décidé de choisir ce sujet de projet, et nous allons essayer d'utiliser ce que nous avons appris pour l'améliorer pour atteindre le but de notre projet, et nous serons les premiers à utiliser ce programme.

Ce projet consiste à créer un logiciel permettant de gérer les films/séries vues par des utilisateurs. Le projet est évolutif, c'est à dire que différentes versions peuvent être proposées, allant de la plus simple à la plus compliquée, de façon à ce que tous puissent, à la fin du semestre, proposer un logiciel en état de fonctionnement.

Analyse des exigences

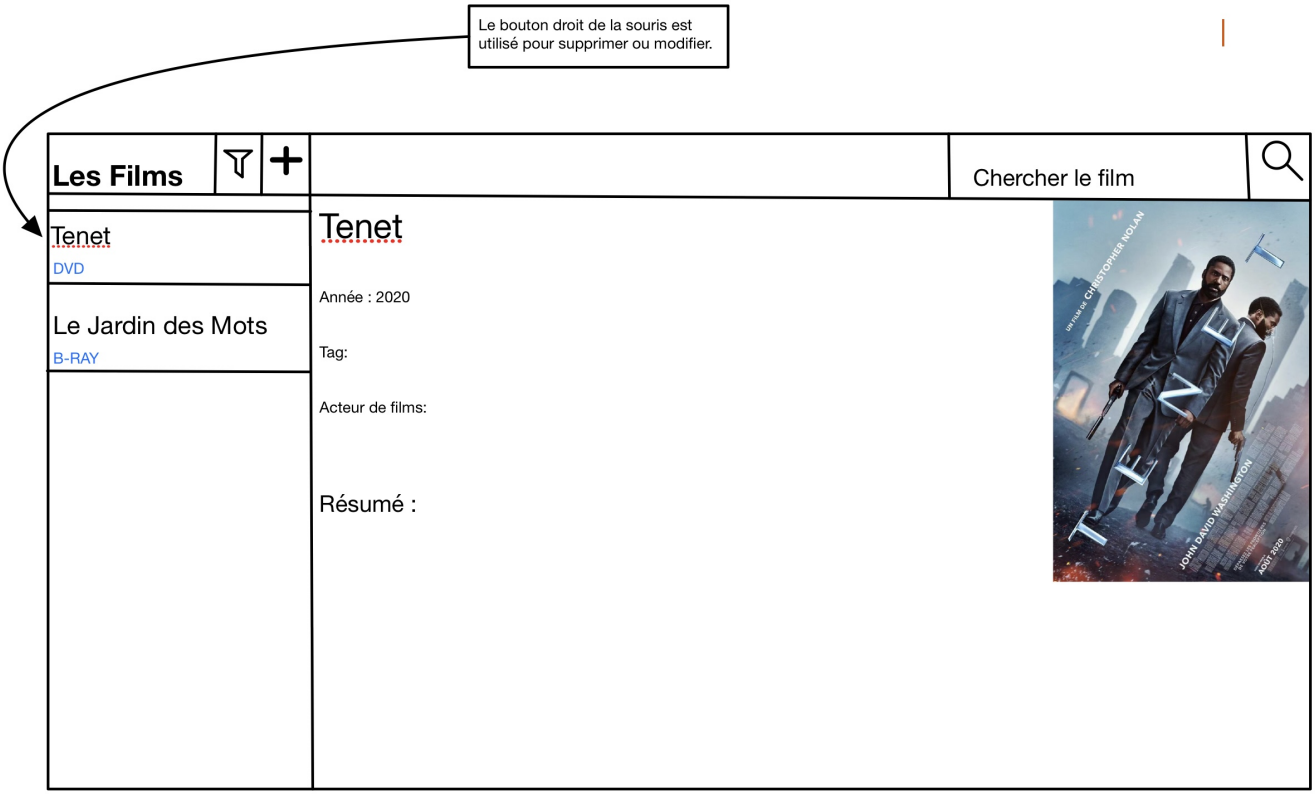
Après d'avoir décidé notre thème, nous avons eu une discussion animée sur notre sujet, et nous avons finalement déterminé nos besoins pour ce programme, et les fonctions du programme ont été résumées comme nécessitant les éléments suivants au total.

1 Gérer la bibliothèque de films

Gère les informations de base de tous les films de la bibliothèque, y compris l'ajout, la suppression, etc, et peut organiser les films en fonction de ces informations.

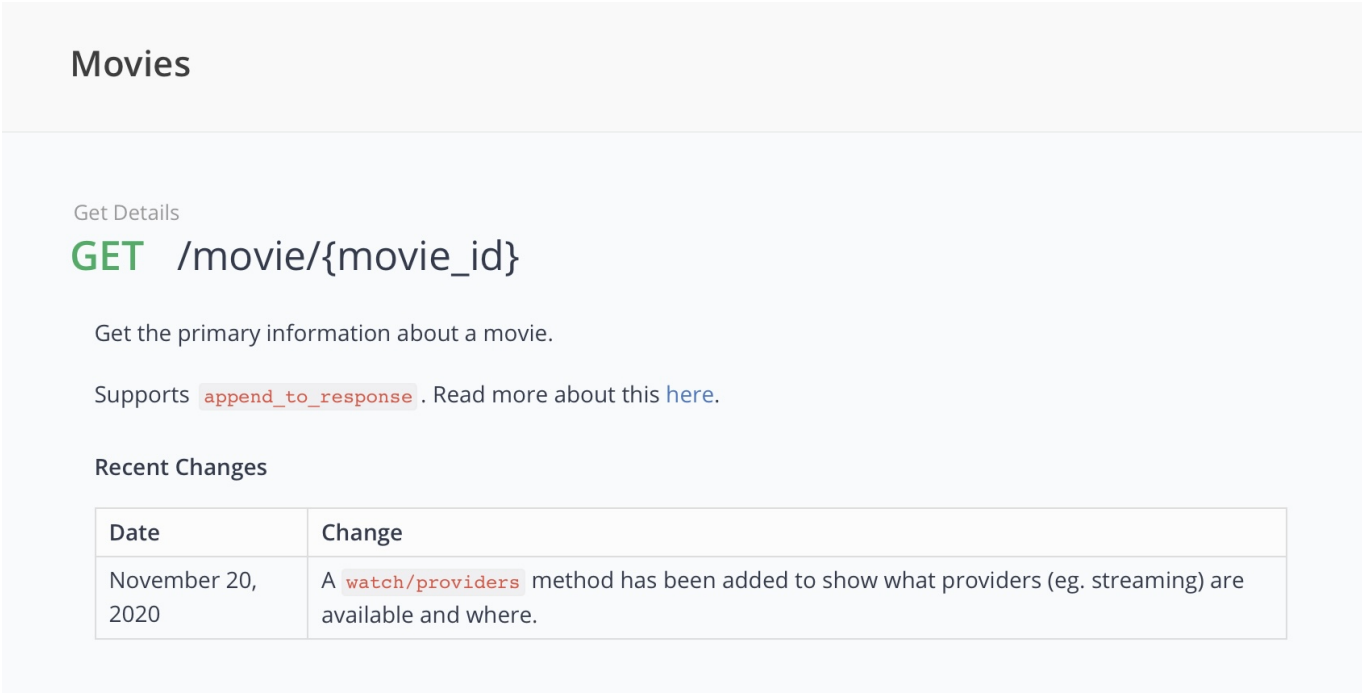
2 Gérer la base de données

Gérer les informations de base des films qui nous sont fournies par le site web, y compris les identifiants des films et la possibilité de sélectionner des films à ajouter à la bibliothèque de films. Ensuite, nous avons essayé de dessiner un aperçu de notre interface.



Cette image est l'interface principale que nous avons dessinée à l'époque, nous avons dessiné l'interface avec notre liste de films sur le côté gauche, les détails des films sur le côté droit, et le titre, le filtre et l'ajout de films sur le côté droit au-dessus, tandis que la fonction de recherche est utilisée sur le côté gauche au-dessus, et nous avons décidé à l'époque que le bouton gauche de la souris peut supprimer les films dont nous n'avons pas besoin.

Après cela, nous nous sommes rendus ensemble sur le site TheMovieDB. C'est ce site web que nous devons utiliser pour notre thème, nous avons vérifié l'API de ce site web, et après avoir regardé les instructions du site web, nous avons eu une Après avoir regardé les instructions, nous avons discuté et finalement décidé que nous devions utiliser la grande section de films (Movies)de ce site.



Movies

Get Details

GET /movie/{movie_id}

Get the primary information about a movie.

Supports `append_to_response`. Read more about this here.

Recent Changes

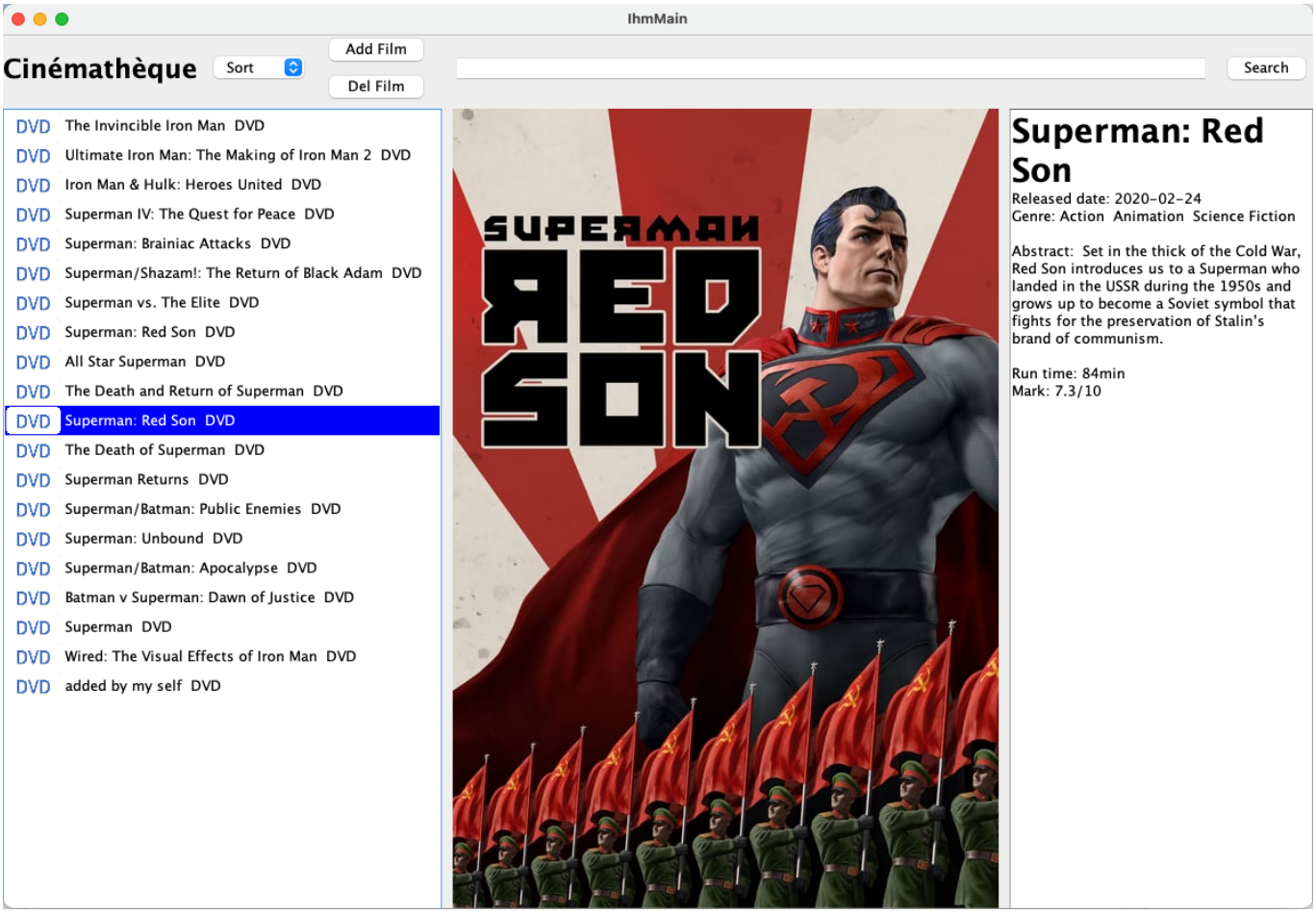
Date	Change
November 20, 2020	A `watch/providers` method has been added to show what providers (eg. streaming) are available and where.

Dans ce site, nous obtiendrons les détails de nos films et leurs identifiants. Les identifiants des films permettront d'obtenir facilement des informations sur les films. Mais pour utiliser ce site, nous avons besoin de la clé API, donc nous nous sommes inscrits sur ce site et le site nous a donné la clé API.

Conception des contours

Interface1: Interface utilisateur principale

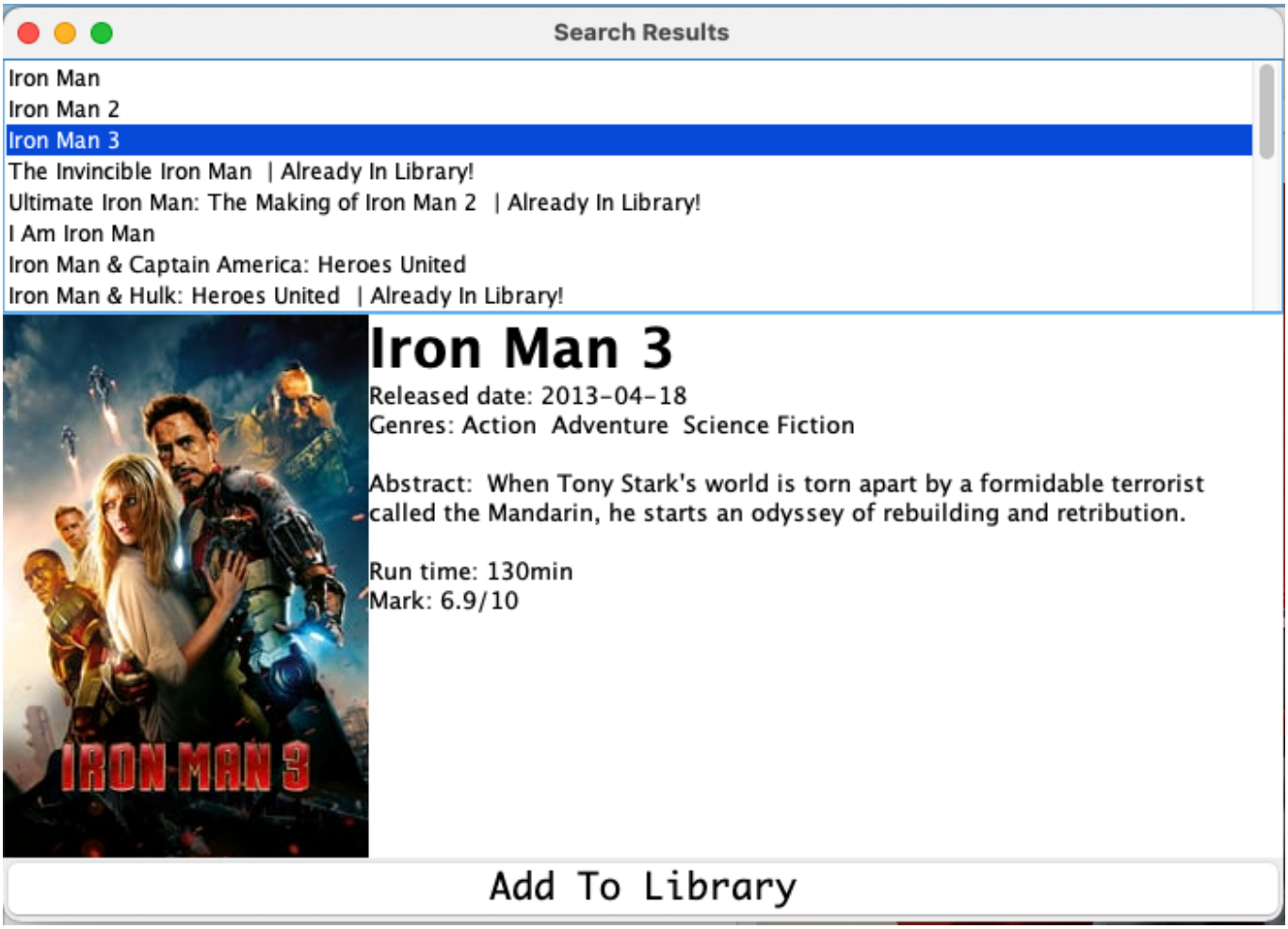
Cette interface est globalement la même que celle que nous avons conçue auparavant, à l'exception de quelques modifications. En haut à gauche de l'interface principale se trouve le nom de notre programme (Cinématique) et à côté se trouvent nos deux boutons avec (ComboBox), le bouton (Ajouter) pour ajouter les films que nous possédons, le bouton (Supprimer) pour supprimer nos films et (ComboBox) que nous utilisons pour trier les films de notre cinémathèque. Nous avons pensé qu'il y avait trop peu de fonctionnalités pouvant être ajoutées au bouton droit de la souris, et nous avons donc décidé d'utiliser le bouton à sa place. En bas à gauche de cette interface principale se trouve notre bibliothèque de films(Library), qui affichera les films que nous avons ajoutés , lorsque nous pensons au nom du film ci-dessus, le côté droit de notre interface affichera l'affiche du film avec les détails du film, et en haut à droite de l'interface se trouvent le bouton de recherche et la barre de recherche, après avoir entré le nom du film dans la barre de recherche et cliqué sur le bouton de recherche, nous obtiendrons les résultats de la recherche de films.



Interface2: Interface de recherche de film par l'utilisateur

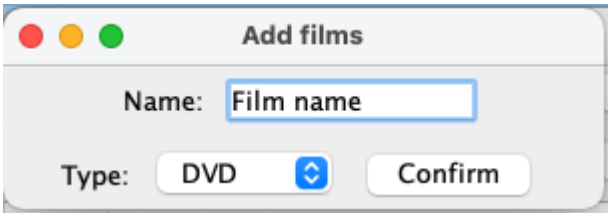
Lorsque nous cliquons sur le bouton de recherche de film, nous entrons dans cette interface - interface de recherche.Dans cette interface, le haut de notre interface est les films les plus pertinents que nous obtenons par la recherche de nom de film, lorsque nous cliquons sur le nom du film, l'affiche de ce film et les informations sommaires du film seront affichées en dessous du nom du film. En bas de l'interface de

recherche se trouve le bouton d'ajouter, lorsque nous cliquons sur ce bouton, nous entrons dans l'interface d'ajouter.



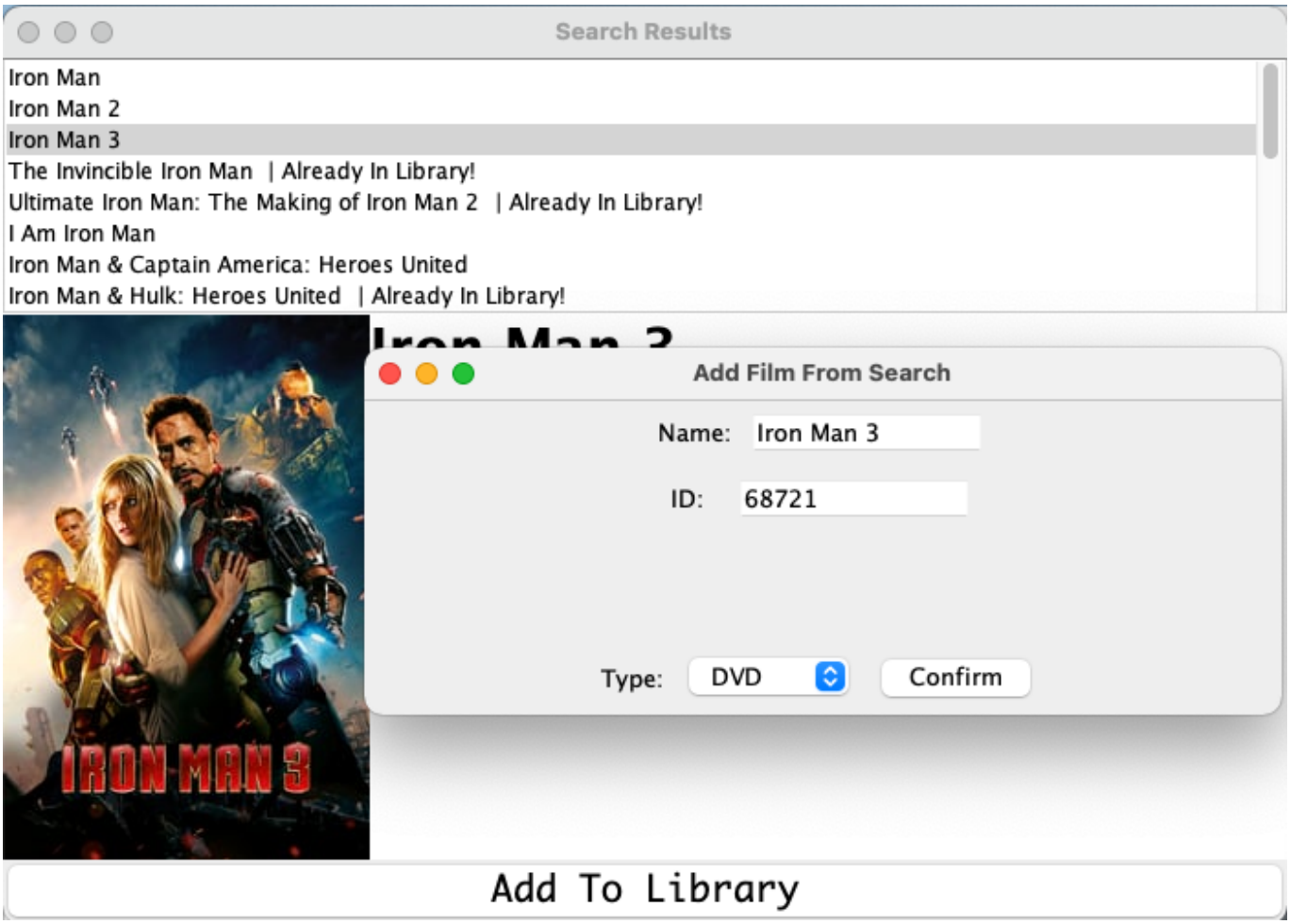
Interface3: L'interface d'ajout de film par l'utilisateur

Après avoir cliqué sur le bouton (Ajouter) dans l'interface principale, nous accédons à cette page, dans laquelle nous pouvons saisir le nom du film et choisir le type de film, lorsque nous cliquons sur le bouton(Ajouter), il sera ajouté à notre bibliothèque(Library).



Interface4: L'interface d'ajout de films après la recherche de films par les utilisateurs

Après avoir cliqué sur le bouton (Ajouter)dans l'interface Rechercher des films, nous entrerons dans cette page, cette page affichera le nom du film et l'identifiant du film, nous pouvons également choisir le type de film à ajouter à notre bibliothèque(Library). Si le film de la liste de recherche existe déjà dans la bibliothèque(Library), l'utilisateur sera invité à indiquer dans le titre qu'il existe déjà dans la bibliothèque(Library).


Search Results
Iron Man
Iron Man 2
Iron Man 3
The Invincible Iron Man | Already In Library!
Ultimate Iron Man: The Making of Iron Man 2 | Already In Library!
I Am Iron Man
Iron Man & Captain America: Heroes United
Iron Man & Hulk: Heroes United | Already In Library!
Add Film From Search
Name: Iron Man 3
ID: 68721
Type: DVD
Confirm
IRON MAN 3
Add To Library

Conception détaillée

Après cela, nous avons discuté des cinq versions de notre sujet, lorsque nous avons fait la proposition initiale du programme pour mettre en œuvre ces cinq versions.

Version 1

Dans la version 1, nous avons créé une bibliothèque locale de films(Library) dans le programme, afin de pouvoir stocker les films que nous avons ajoutés. Nous allons utiliser le format CSV pour stocker nos films sauvegardés.

Version 2 et Version 3

Nous utiliserons la clé API que nous avons obtenue, nous utiliserons la méthode du site web pour nous connecter à la bibliothèque de films (Library)sur le site web, après quoi nous obtiendrons l'identifiant du film et le stockerons dans notre propre bibliothèque de films(Library), après quoi nous utiliserons la méthode du site web pour obtenir les détails du film par l'identifiant du film, puis nous afficherons les détails dans notre bibliothèque de films.

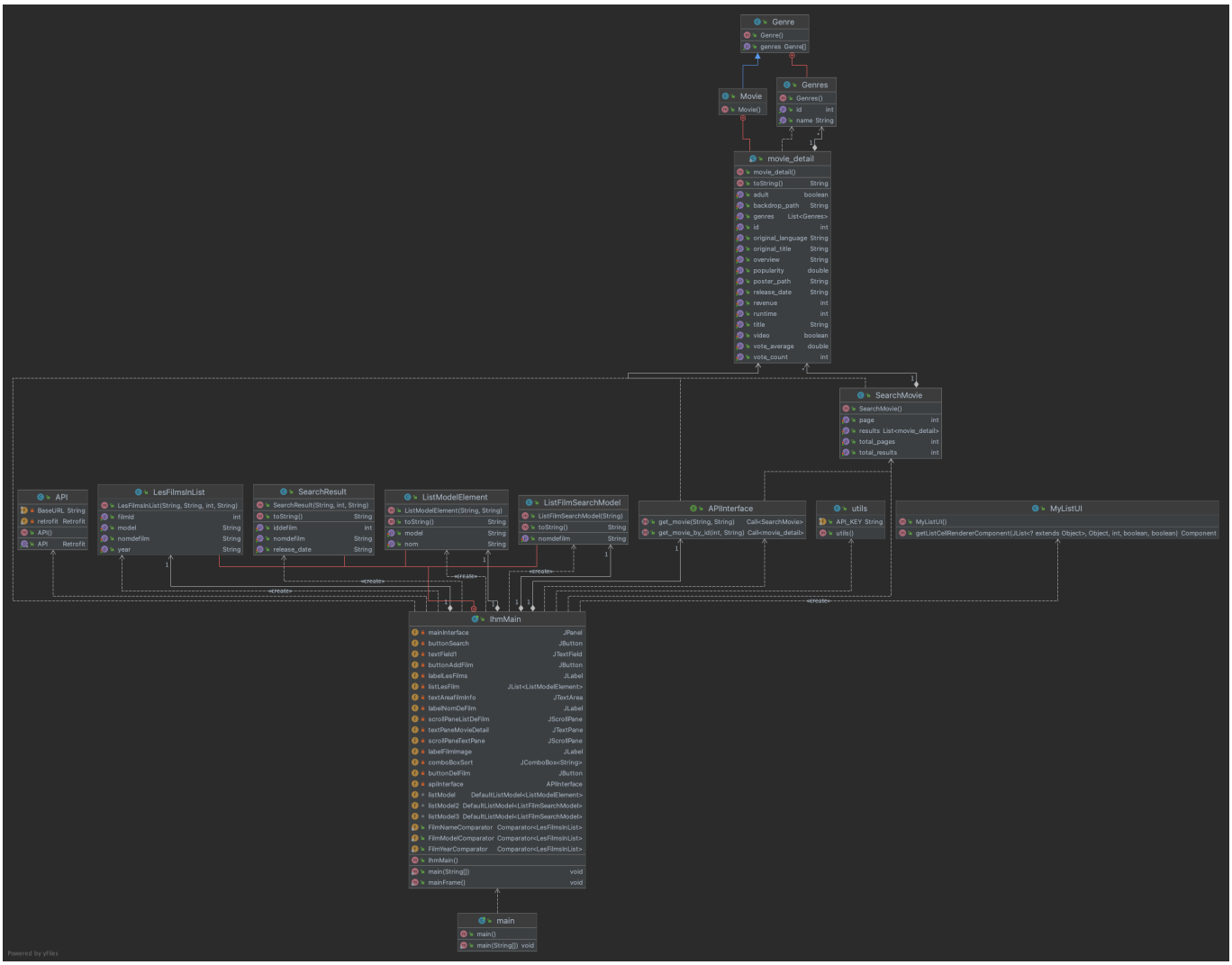
Version 4

Dans la version 4, nous utiliserons la même méthode que dans les versions 2 et 3, nous obtiendrons la date de sortie du film et le type de film par l'identifiant du film, puis nous l'afficherons dans notre bibliothèque(Library) de la même manière.

Version 5

Mais après avoir lu les livres d'apprentissage, nous n'avons pas réussi à trouver une solution.

Ensuite, voyons le graphe de dépendance relationnelle de notre programme:


Genre
Genre()
genres Genre[]
Movie
Movie()
Genres
Genres()
id int
name String
movie_detail
movie_detail()
toString() String
adult boolean
backdrop_path String
genres List<Genres>
id int
original_language String
original_title String
overview String
popularity double
poster_path String
release_date String
revenue int
runtime int
title String
video boolean
vote_average double
vote_count int
SearchMovie
SearchMovie()
page int
results List<movie_detail>
total_pages int
total_results int
API
BaseURL String
retrofit Retrofit
API() Retrofit
LesFilmsInList
LesFilmsInList(String, String, int, String)
filmid int
model String
nomdefilm String
year String
SearchResult
SearchResult(String, int, String)
toString() String
iddefilm int
nomdefilm String
release_date String
ListModelElement
ListModelElement(String, String)
toString() String
model String
nom String
ListFilmSearchModel
ListFilmSearchModel(String)
toString() String
nomdefilm String
APIInterface
get_movie(String, String) Call<SearchMovie>
get_movie_by_id(int, String) Call<movie_detail>
utils
API_KEY String
utils()
MyListUI
MyListUI()
getListCellRendererComponent(JList<? extends Object>, Object, int, boolean, boolean) Component
«create»
IhmMain
mainInterface JPanel
buttonSearch JButton
textField1 JTextField
buttonAddFilm JButton
labelLesFilms JLabel
listLesFilm JList<ListModelElement>
textAreaFilmInfo JTextArea
labelNomDeFilm JLabel
scrollPaneListDeFilm JScrollPane
textPaneMovieDetail JTextPane
scrollPaneTextPane JScrollPane
labelFilmImage JLabel
comboBoxSort JComboBox<String>
buttonDeFilm JButton
apiInterface APIInterface
listModel DefaultListModel<ListModelElement>
listModel2 DefaultListModel<ListFilmSearchModel>
listModel3 DefaultListModel<ListFilmSearchModel>
FilmNameComparator Comparator<LesFilmsInList>
FilmModelComparator Comparator<LesFilmsInList>
FilmYearComparator Comparator<LesFilmsInList>
IhmMain()
main(String[]) void
mainFrame() void
main
main()
main(String[]) void

Réaliser

Code Principale dans la Version 1

Cette section permet principalement aux utilisateurs d'entrer le film qu'ils veulent voir et de le stocker localement.

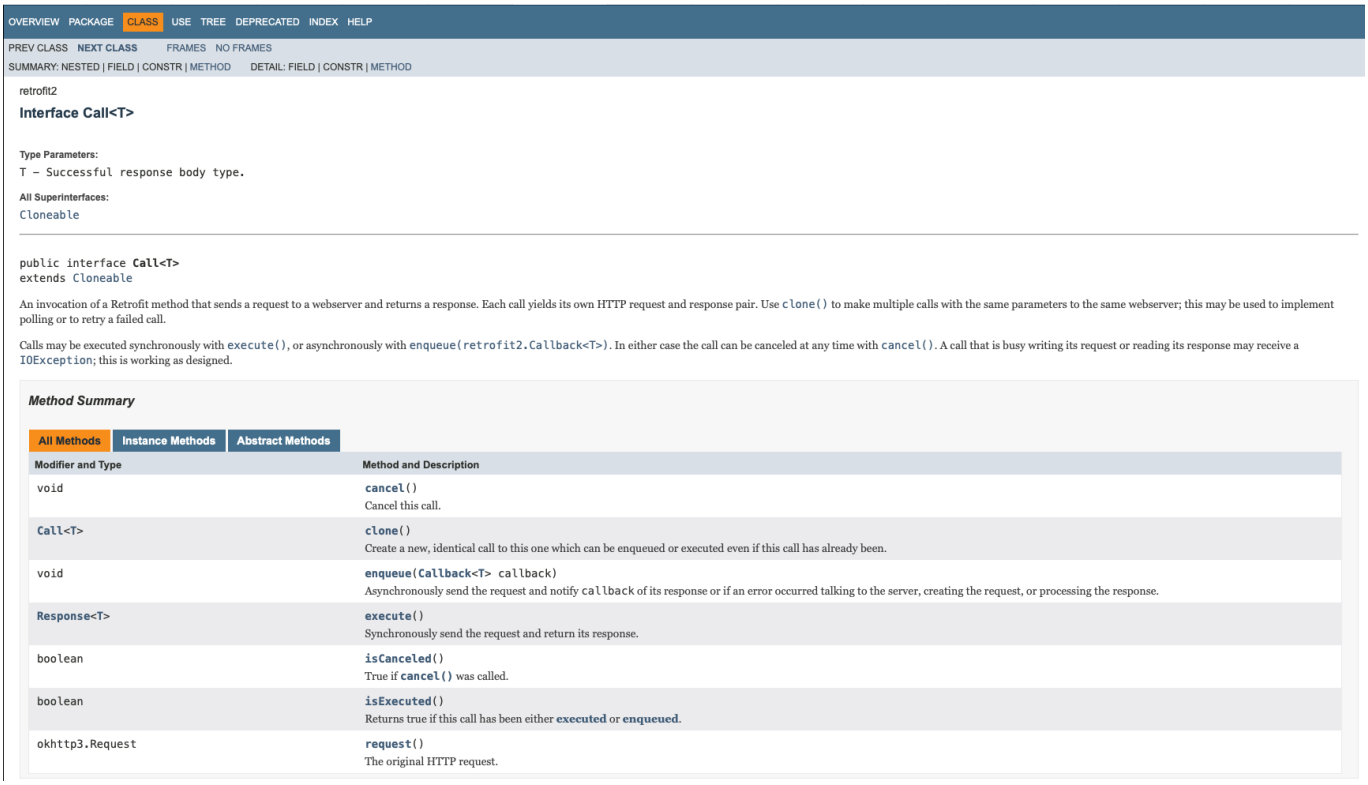
Nous voulons que les utilisateurs saisissent directement le film qu'ils veulent ajouter à la liste, nous avons éliminé la détection des films dupliqués afin que les utilisateurs puissent entrer le film qu'ils veulent voir plusieurs fois. Le film est stocké au format de document CSV, le fichier divise le film à l'aide d'un nouveau caractère de ligne `\n` et du symbole `,` divise l'information sur le film, y compris le nom du film, l'ID correspondant du film sur le site TheMovieDB, aussi le mode de stockage du film et l'année de sortie du film.

```
// textFieldFilmNameAddFilmToTxt stocke le nom du film que l'utilisateur
veut ajouter, comboBoxFilmModeAddFilmToTxt
// stocke le mode du film que l'utilisateur veut ajouter
// Check if the length of film name is 0, if then return error.

    if(textFieldFilmNameAddFilmToTxt.getText().length()==0){
        JOptionPane.showMessageDialog(null,"You must enter film
name!","ERROR",JOptionPane.PLAIN_MESSAGE);
        return;
    }

        // Write Film information into txt file, and add film into list
model to show on screen.
    try{

        FileWriter fw=new FileWriter(absoultePath,true);

fw.write("\n"+textFieldFilmNameAddFilmToTxt.getText()+","+comboBoxFilmMode
AddFilmToTxt.getSelectedItem().toString()+","+0+","+0);
        fw.close();

        listModel.addElement(new
ListModelElement(textFieldFilmNameAddFilmToTxt.getText(),comboBoxFilmModeA
ddFilmToTxt.getSelectedItem().toString()));
        listFilmInList.add(new
LesFilmsInList(textFieldFilmNameAddFilmToTxt.getText(),comboBoxFilmModeAdd
FilmToTxt.getSelectedItem().toString(),0,"0"));
        frameAddFilmToTxt.setVisible(false);

    }catch(IOException ioException){
        ioException.printStackTrace();
    }
```

Code principale pour les versions 2, 3 et 4

Les versions 2, 3 et 4 utilisent principalement l'API TheMovieDB, cette API nous aide à obtenir toutes sortes d'informations sur le film, y compris le nom du film, l'ID du film, ainsi que les genres de film, l'année de sortie etc.

Nous utilisons Retrofit pour appliquer l'API http à notre projet, nous utilisons `Call` pour envoyer notre demande



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

retrofit2

Interface Call<T>

Type Parameters:

`T` – Successful response body type.

All Superinterfaces:

`Cloneable`

```
public interface Call<T>
extends Cloneable
```

An invocation of a Retrofit method that sends a request to a webserver and returns a response. Each call yields its own HTTP request and response pair. Use `clone()` to make multiple calls with the same parameters to the same webserver; this may be used to implement polling or to retry a failed call.

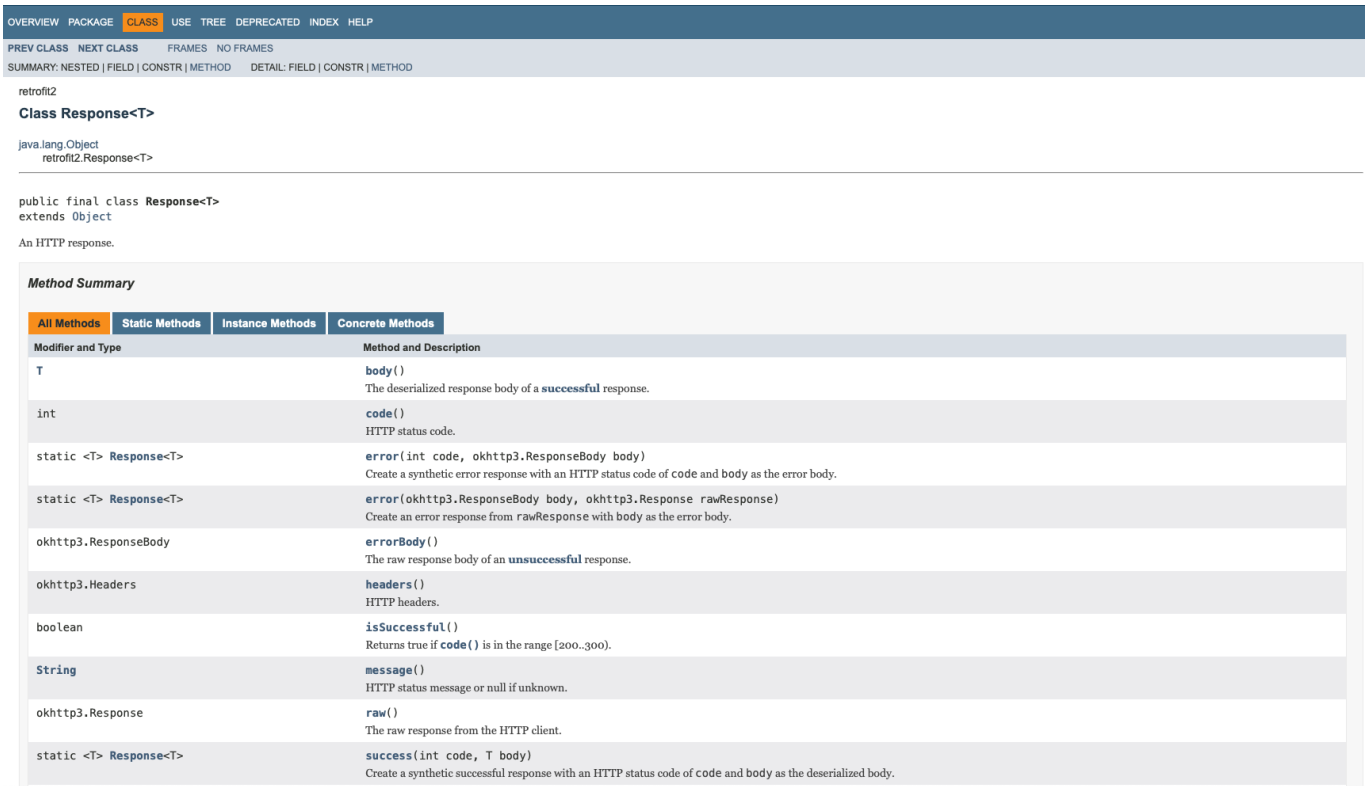
Calls may be executed synchronously with `execute()`, or asynchronously with `enqueue(retrofit2.Callback<T>)`. In either case the call can be canceled at any time with `cancel()`. A call that is busy writing its request or reading its response may receive a `IOException`; this is working as designed.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
`void`	**`cancel()`** Cancel this call.
`Call<T>`	**`clone()`** Create a new, identical call to this one which can be enqueued or executed even if this call has already been.
`void`	**`enqueue(Callback<T> callback)`** Asynchronously send the request and notify `callback` of its response or if an error occurred talking to the server, creating the request, or processing the response.
`Response<T>`	**`execute()`** Synchronously send the request and return its response.
`boolean`	**`isCanceled()`** True if **`cancel()`** was called.
`boolean`	**`isExecuted()`** Returns true if this call has been either **executed** or **enqueued**.
`okhttp3.Request`	**`request()`** The original HTTP request.

Et nous pouvons obtenir les données retournées `response.body()`



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

retrofit2

Class Response<T>

java.lang.Object
  retrofit2.Response<T>

```
public final class Response<T>
extends Object
```

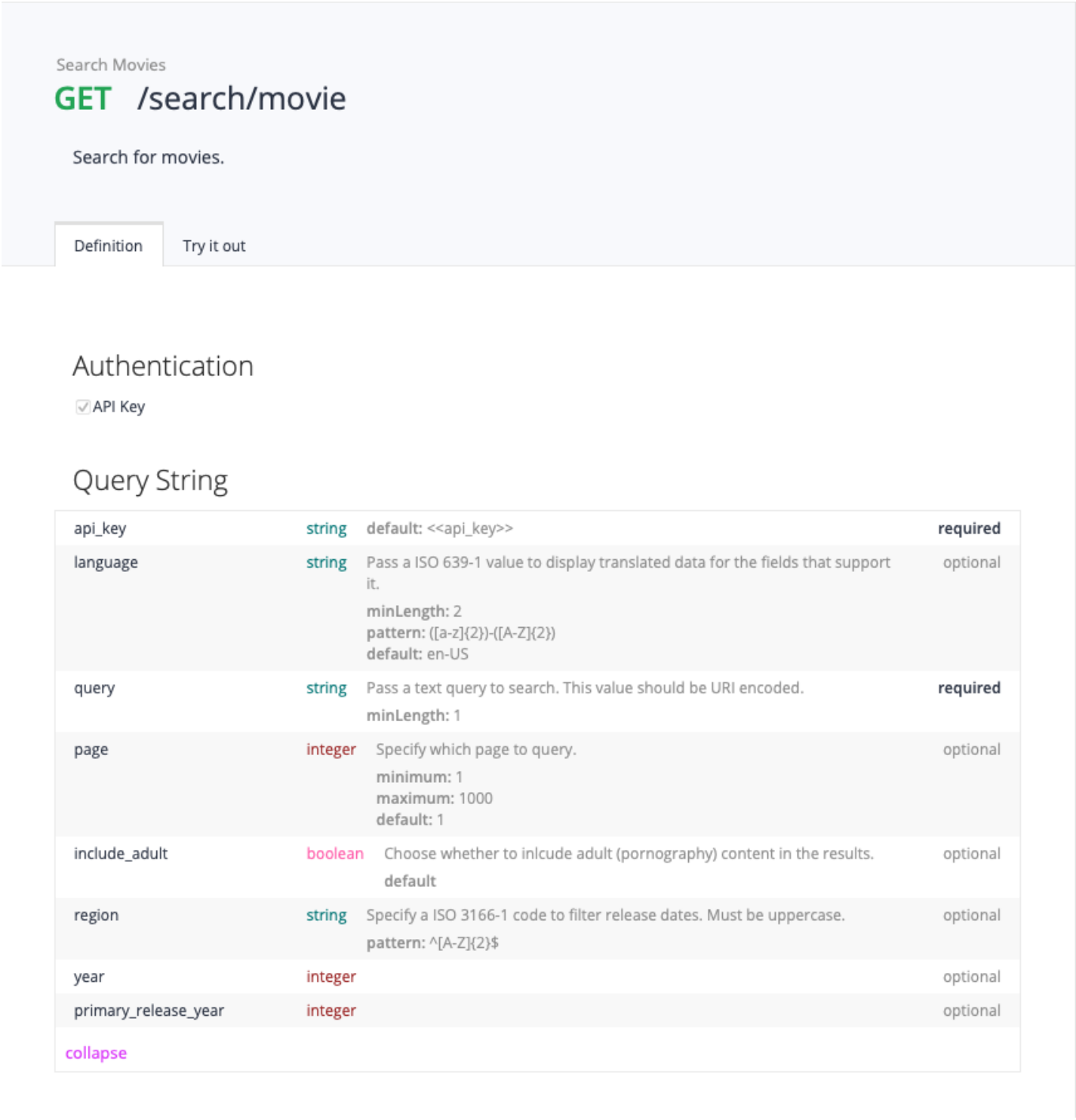
An HTTP response.

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`T`	**`body()`** The deserialized response body of a **successful** response.
`int`	**`code()`** HTTP status code.
`static <T> Response<T>`	**`error(int code, okhttp3.ResponseBody body)`** Create a synthetic error response with an HTTP status code of `code` and `body` as the error body.
`static <T> Response<T>`	**`error(okhttp3.ResponseBody body, okhttp3.Response rawResponse)`** Create an error response from `rawResponse` with `body` as the error body.
`okhttp3.ResponseBody`	**`errorBody()`** The raw response body of an **unsuccessful** response.
`okhttp3.Headers`	**`headers()`** HTTP headers.
`boolean`	**`isSuccessful()`** Returns true if **`code()`** is in the range [200..300).
`String`	**`message()`** HTTP status message or null if unknown.
`okhttp3.Response`	**`raw()`** The raw response from the HTTP client.
`static <T> Response<T>`	**`success(int code, T body)`** Create a synthetic successful response with an HTTP status code of `code` and `body` as the deserialized body.

Nous recherchons le film en utilisant le nom du film entré par l'utilisateur, avec le API key et le text query pour faire les recherche.



Search Movies

GET /search/movie

Search for movies.

Definition | Try it out

Authentication

API Key

Query String

api_key	string	default: <<api_key>>	**required**
language	string	Pass a ISO 639-1 value to display translated data for the fields that support it. **minLength:** 2 **pattern:** ([a-z]{2})-([A-Z]{2}) **default:** en-US	optional
query	string	Pass a text query to search. This value should be URI encoded. **minLength:** 1	**required**
page	integer	Specify which page to query. **minimum:** 1 **maximum:** 1000 **default:** 1	optional
include_adult	boolean	Choose whether to inlcude adult (pornography) content in the results. **default**	optional
region	string	Specify a ISO 3166-1 code to filter release dates. Must be uppercase. **pattern:** ^[A-Z]{2}$	optional
year	integer		optional
primary_release_year	integer		optional

collapse

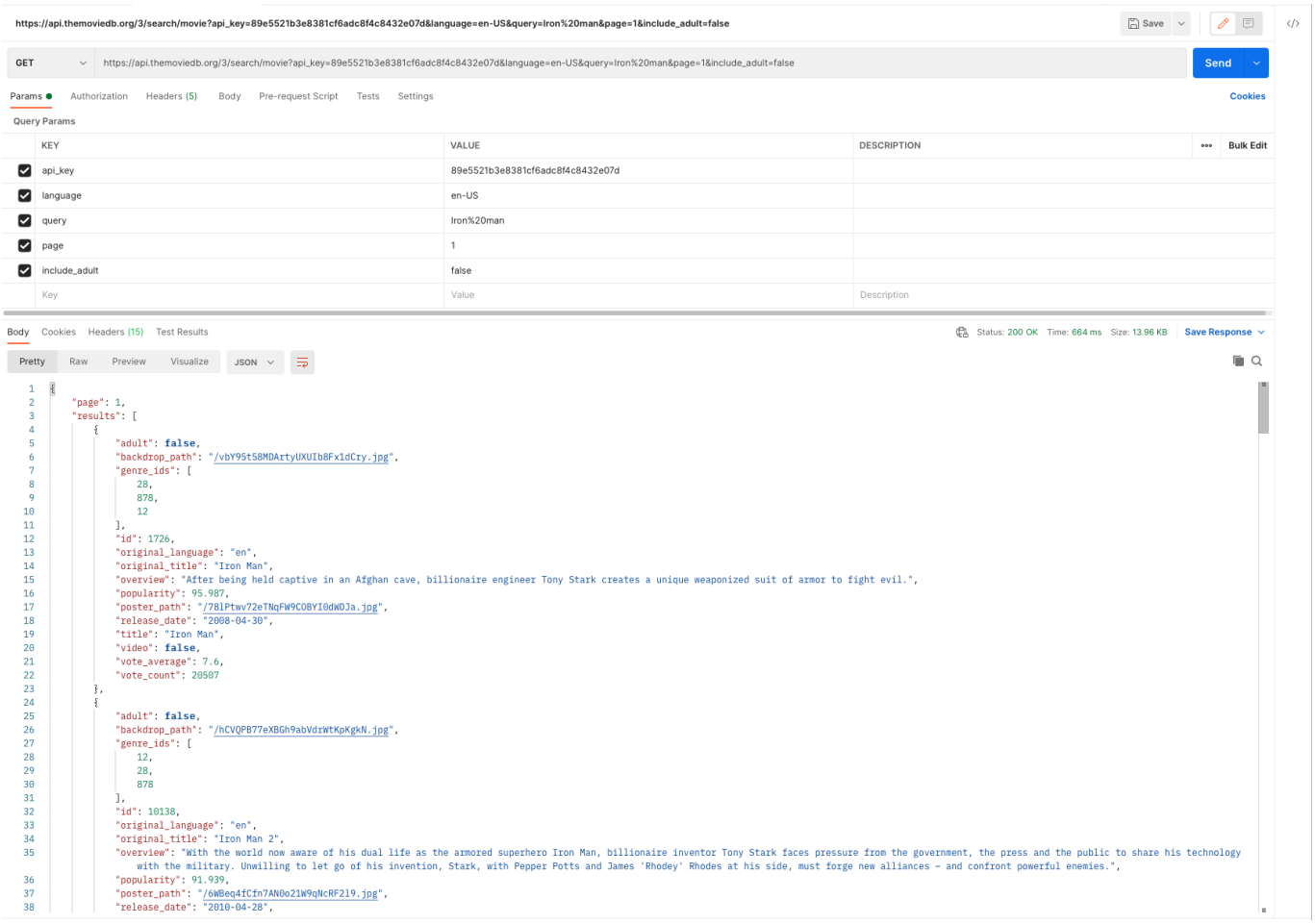
```
    @GET("3/search/movie")
    Call<SearchMovie> get_movie(
@Query("api_key") String apiKey,
@Query("query") String query
        );
```

```
Call<SearchMovie>
searchMovieCall=apiInterface.get_movie(utils.API_KEY,str);
searchMovieCall.enqueue(new Callback<SearchMovie>(){
```

```
@Override
// Invoked for a received HTTP response.
public void onResponse(Call<SearchMovie> call,Response<SearchMovie>
response){
    SearchMovie searchMovie=response.body();
}

public void onFailure(Call<SearchMovie> call,Throwable throwable){
    throwable.getMessage();
}
```

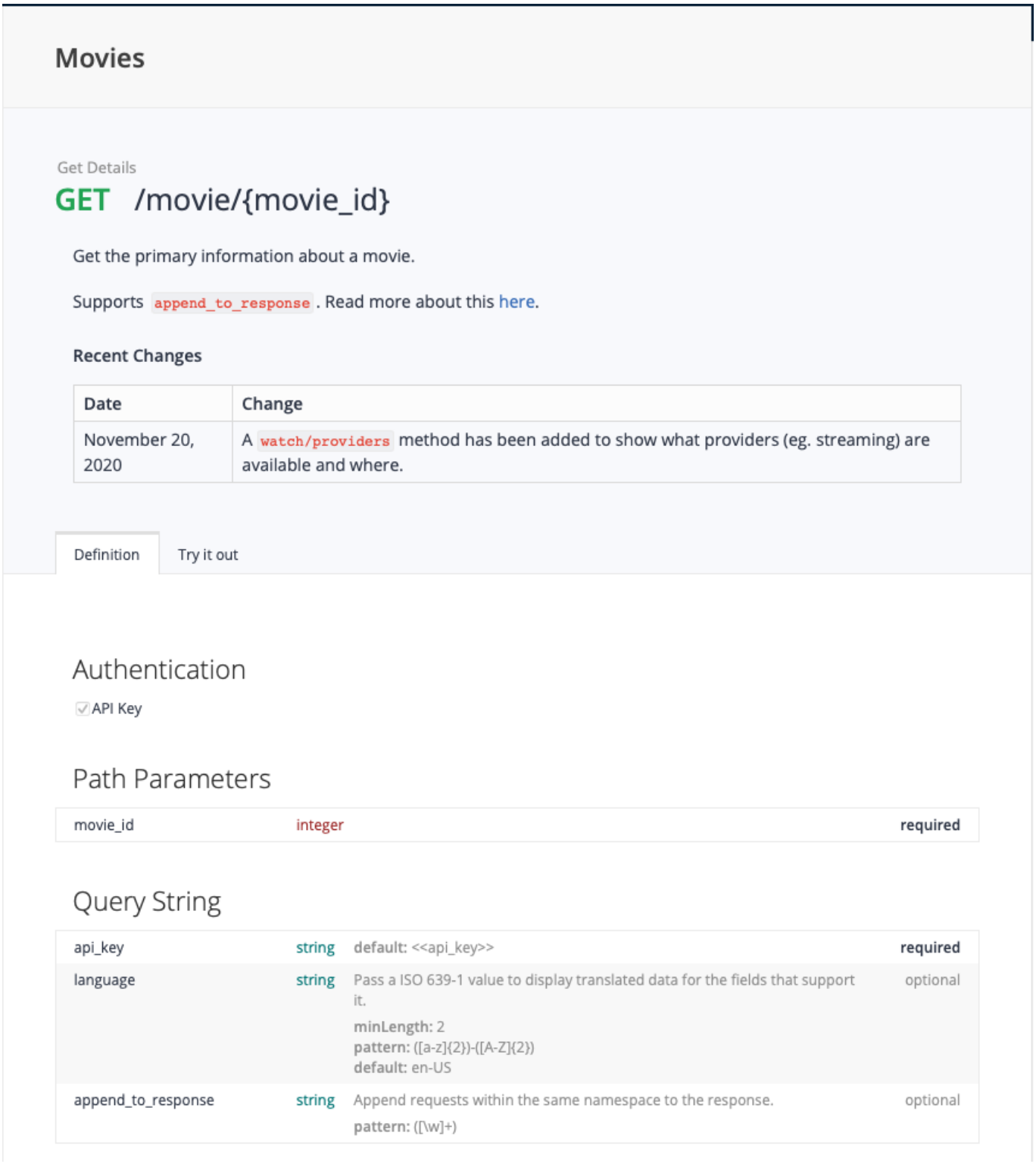
Par exemple, nous rechercherons des informations sur le film Iron Man, et donc le query_string ici est "Iron man", nous envoyons la demande avec le lien suivant:
`https://api.themoviedb.org/3/search/movie?api_key=89e5521b3e8381cf6adc8f4c8432e07d&language=en-US&query=Iron%20man&page=1&include_adult=false`

Le site Postman nous permet de Debug facilement notre réponse (`response.body()`), nous envoyons le lien que nous avons obtenue à Postman et obtenons les résultats suivants:



Donc nous pouvons facilement obtenir chaque l'ID du film que nous avons cherché.

Nous stockons ces ID dans une liste, et nous ajoutons un `ListSelectionListener` à notre liste afin que lorsque les utilisateurs changent leurs sélections, nous puissions obtenir leurs identifiants de film sélectionnés et rechercher des informations de film par rapport d'ID.



De même, nous utilisons cette méthode pour obtenir les détails du film sur l'ID du film que nous stockons dans le fichier.

```
listSearchRemote.addListSelectionListener(new ListSelectionListener(){
@Override
public void valueChanged(ListSelectionEvent e){

    // Get the value only when the left key is released.
    if(e.getValueIsAdjusting()){
        return
    };
```

```
    // Instantiate the movie info which selected.
    SearchResult searchResultSelect=searchResultArrayList.get(0);
    try{

searchResultSelect=searchResultArrayList.get(listSearchRemote.getSelectedI
ndex());
    }catch(Exception exception){
        exception.printStackTrace();
    }

    int searchResultFilmId=searchResultSelect.iddefilm;
    String searchResultFilmNom=searchResultSelect.nomdefilm;

        // Use the movie id which we got to get movie info.

Call<Movie.movie_detail>callMovieDetail=apiInterface.get_movie_by_id(searc
hResultFilmId,utils.API_KEY);

        callMovieDetail.enqueue(new Callback<Movie.movie_detail>(){
@Override
public void
onResponse(Call<Movie.movie_detail>call,Response<Movie.movie_detail>respon
se){
        Movie.movie_detail movieDetail=response.body();
        }catch(IOException ioException){
//  ioException.printStackTrace();
        jLabelPostImage.setText("This movie has no post image");
        }
@Override
public void onFailure(Call<Movie.movie_detail>call,Throwable throwable){
        JOptionPane.showMessageDialog(null,"You must enter film
name!","ERROR",JOptionPane.PLAIN_MESSAGE);
        }
```

Conclusion

Charge de travail et délai d'achèvement

// 我们从2月7号 到2月11号完成了Version1，从2月13号开始到3月7号完成了Version2,3,4 版本， 然后从3月中旬开始一直到4月20号左右就在添加代码注释和修改BUG

Name	Date
Réaliser le IHM du projet	Feb 6, 2021 → Feb 12, 2021
Réaliser la fonction du stocker de films	Feb 13, 2021 → Feb 20, 2021
Saisie, sauvegarde et supprime des films	Feb 21, 2021 → Mar 2, 2021
Info de film avec themoviedb	Mar 3, 2021 → Mar 16, 2021
Film propose	Mar 17, 2021 → Mar 29, 2021
Cherche de film par genre, année et acteur	Mar 30, 2021 → Apr 12, 2021
Fin du projet.	Apr 16, 2021 → Apr 18, 2021

Diagramme de Gantt prévu

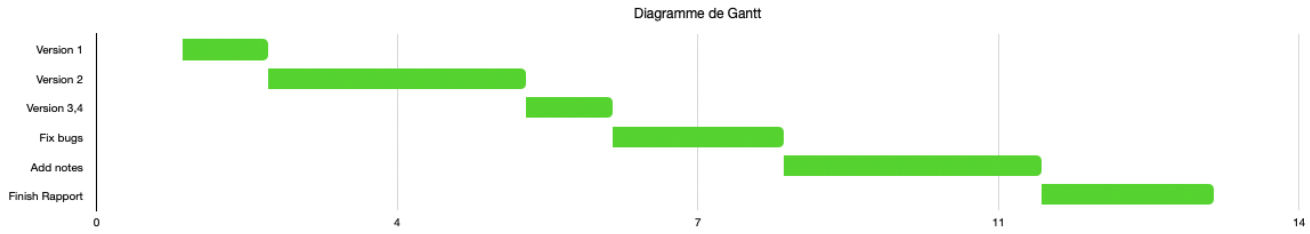


Diagramme de Gante effectif