TP Fakeflix

Flutter

Objectifs

- Emploi de variables d'environnement
- Mise en place du pattern Master / Details
- Navigation
- Store de données
- Requêtes HTTP
- Future + async / await
- Persistance de données

Descriptif fonctionnel global

L'application Fakeflix va permettre à un utilisateur de parcourir des données provenant d'une API web et se constituer une liste de favoris / films à regarder.

Initialisation

- Créez un nouveau projet Flutter vierge nommé **fakeflix**.

API TMDB

- Créez un compte gratuit sur https://www.themoviedb.org/ afin de pouvoir utiliser l'***API The Movie DB*** *(TMDB)* (https://www.themoviedb.org/signup).
- Consultez la **documentation de l'API**
  https://developer.themoviedb.org/docs/getting-started
  et
  https://developer.themoviedb.org/reference/intro/getting-started
- Une fois connecté à votre compte, obtenez votre **token d'API** ici :
  https://www.themoviedb.org/settings/api

Variables d'environnement

- Installez le package ***flutter_dotenv*** (https://pub.dev/packages/flutter_dotenv) afin d'employer un fichier ***.env*** dans lequel vous renseignerez la variable d'environnement : ***API_TOKEN*** (dont la valeurs est communiquée dans votre compte *themoviedb*).
  - la valeur de ***API_TOKEN*** devra être transmise dans le ***Header Authorization (mode Bearer) de chaque requête HTTP***.
- Attention, pour des raisons de sécurité, **le fichier .env ne doit pas être intégré à votre dépôt Git**.

Interaction avec l'API TMDB

- Commencez par **tester l'API** avec un logiciel tel que *Postman*, *Bruno*, *Insomnia* ou en ligne de commande avec *Curl*.
- Observez la **structure des réponses fournies par l'API au format JSON** (clés, valeur, types de données, volume de données…).
- Effectuez une requête **HTTP GET sur l'URI /discover/movie** afin d'obtenir une liste de films paginés par lots de 20 (ex: https://api.themoviedb.org/3/discover/movie?include_adult=false&include_video=false&language=en-US&page=1&sort_by=popularity.desc).

  Seuls les 20 premiers objets fournis par l'API suffiront dans le cadre de cette application.

- Récupérez l'identifiant de 1 film (ex: 823464) et effectuez une requête HTTP GET sur l'URI **/movie/{movie_id}** (cf. https://developer.themoviedb.org/reference/movie-details),

  ex: https://api.themoviedb.org/3/movie/823464

- Créez une classe ***MovieService*** dans laquelle vous ajouterez des méthodes permettant d'interagir avec l'API. Chaque méthode de la classe ***MovieService*** effectuera une opération asynchrone et retournera une réponse sous forme de ***Future***.

- Dans la classe ***MovieService***, installez le package ***dio*** (https://pub.dev/packages/dio) afin de disposer d'un **client http** permettant d'obtenir des **données au format JSON fournies par l'API**.

Mapping des données

- Créez une classe Dart ***Movie*** avec les attributs utiles (ex: *id, title…*) en vous inspirant des données fournies par l'API.

- Ajoutez un constructeur de type ***factory*** à la classe ***Movie*** afin de créer des instances de ***Movie*** à partir de données JSON obtenues auprès de l'API (à l'origine, au format *Map<String, dynamic>*). Inspirez-vous de l'exemple suivant : https://docs.flutter.dev/cookbook/networking/fetch-data (cf. méthode *factory Album.fromJson*).

- Mettez en place une première méthode ***readMovies*** dans la classe ***MovieService*** permettant de retourner une valeur de type ***Future<List<Movie>>.***

Master / Details

- Mettez en place une structure de projet permettant d'organiser l'application sur le principe ergonomique ***Master / Details*** :
  - 1 écran ***Master*** contenant une liste d'items présentés sous forme de ***Preview***,
  - 1 écran ***Details*** permettant de présenter les informations détaillées d'un item sélectionné.
  - Nommez vos widgets : ***MoviesMasterScreeen***, ***MovieDetailsScreen*** et ***MoviePreview***.

    - Pour ce faire, employez le package ***go_router*** (https://pub.dev/packages/go_router) et définissez les routes utiles.

- Permettez à l'utilisateur de naviguer entre les écrans :
    - ***Master -> Details*** : clic sur un item de l'écran ***Master***,
    - ***Details -> Master*** : clic sur un élément de l'*AppBar* de l'écran ***Details***.

Ecran Master : liste d'items

- Dans l'écran ***Master***, affichez dynamiquement la liste des items récupérés à l'aide de la méthode ***readMovies*** de la classe ***MovieService***. Pour ce faire, employez un ***FutureBuilder*** et un widget ***ListView***.

- Affichez dynamiquement l'état de la ***Future*** fournie par la méthode ***readMovies*** de la classe ***MovieService***, en vous basant sur le snapshot du ***FutureBuilder*** :
    - affichage d'un message en cas d'erreur,
    - affichage d'une animation de chargement pendant l'obtention des données,
    - affichage de la liste après réception des données.

- Créez un widget ***Preview*** destiné à afficher l'item sous forme d'aperçu (ex: titre + image).

- Affichez les items du widget ***ListView*** sous forme de ***Preview***.

Ecran Details : item détaillé

- Dans la classe ***MovieService***, créez une méthode ***readMovieById*** recevant en paramètre l'***id*** d'un film et permettant de fournir des données détaillées sur le film.
- Dans l'écran Details, récupérez dynamiquement l'***id*** du film et employez la méthode ***readMovieById*** afin d'obtenir puis d'afficher une sélection de ses informations détaillées (ex: *genres, overview, popularity, poster_path*…).

  ex : https://api.themoviedb.org/3/movie/**823464**?language=en-US
- Employez le package ***percent_indicator*** (https://pub.dev/packages/percent_indicator) afin de représenter graphiquement le taux d'appréciation des utilisateurs (cf. *vote_average*).

```
GET /3/movie/823464 HTTP/1.1
Host: api.themoviedb.org
Accept: application/json

Réponse HTTP :
http/1.1 200 OK
Content-type : application/json
{
  "poster_path": "/IfB9hy4JH1eH6HEfIgIGORXi5h.jpg",
  "adult": false,
  "overview": "Jack Reacher must uncover the truth behind a major government conspiracy in order to clear his name. On the run as a fugitive from the law, Reacher uncovers a potential secret from his past that could change his life forever.",
  "release_date": "2016-10-19",
  "genre_ids": [
    53,  28,  80, 18, 9648
  ],
  "id": 343611,
```

```
  "original_title": "Jack Reacher: Never Go Back",
  "original_language": "en",
  "title": "Jack Reacher: Never Go Back",
  "backdrop_path": "/4ynQYtSEuU5hyipcGkfD6ncwtwz.jpg",
  "popularity": 26.818468,
  "vote_count": 201,
  "video": false,
  "vote_average": 4.19
}
```

Likes

- Mettez en place un ***système de "like"*** permettant à l'utilisateur de constituer une liste d'items favoris,
- Depuis l'écran ***Details*** d'un item, permettez à l'utilisateur de *"liker"* un item,
- Distinguez graphiquement les items *"likés"* dans l'écran ***Master***,
- Pour faire en sorte que les items *"likés"* persistent pendant la session courante d'utilisation de l'application, mettez en place un store de données avec le package ***provider*** (https://pub.dev/packages/provider et https://docs.flutter.dev/data-and-backend/state-mgmt/simple).
- Ajoutez un nouvel écran ***Likes*** à l'application affichant la liste des items likés. Depuis cet écran, permettez à l'utilisateur de supprimer des éléments de la liste.

UI

- Personnalisez l'interface graphique de l'application :
    - ajout d'une bannière à l'aide de l'image fournie (cf. propriétés *flexibleSpace* et *toolbarHeight* du widget ***AppBar***),
    - emploi d'une typographie personnalisée (cf. typographie *"Oswald"* fournie),

- Installez le package ***flutter_native_splash*** (https://pub.dev/packages/flutter_native_splash) et générez un splash screen pour l'application à l'aide des images fournies (crédits : FlatIcon.com).
- Installez le package ***flutter_launcher_icons*** https://pub.dev/packages/flutter_launcher_icons et générez une icône pour l'application à l'aide des images fournies (crédits : Flaticon.com).

API Fakeflix : Sign up / Sign in

Une seconde API https://fakeflix.shrp.dev, permet à un utilisateur de créer un compte afin de sauvegarder les films "likés" et d'évaluer ceux qu'il a visionné.

- **Sign up** : Création d'un compte utilisateur

```
POST /users HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "email":"john@doe.com",
  "password":"azerty",
  "first_name":"John",
  "last_name":"Doe",
  "role":"dc6fb553-1fea-4bce-9ece-67b913e8f6d6"
}
```

- **Sign in** : Connexion au compte utilisateur et obtention d'un token de connexion (JWT)

```
POST /auth/login HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "email":"john@doe.com",
  "password":"azerty"
}
```

- **Refresh** : Obtention d'un nouvel *access_token* à partir du *refresh_token* transmis au sign in

```
POST /auth/refresh HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "refresh_token":"<refresh_token>",
  "mode":"json"
}
```

- **Me** : Obtention des informations sur l'utilisateur connecté
  (ex: adresse mail, prénom, nom…). Renseigner l'*access_token* en valeur du *header authorization*, en mode *Bearer*.

```
GET /users/me HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json
```

- **Sign Out** : Invalidation du *refresh_token* transmis lors du sign in

```
POST /auth/logout HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "refresh_token":"<refresh_token>"
}
```

API Fakeflix : Likes

Le endpoint **"/items/likes"** permet à un utilisateur authentifié de **sauvegarder une liste de films**, **indiquer s'il a visionné les films** et leur **attribuer une note de 0 à 5**.

Chaque action liée au endpoint "/items/likes" sur l'API implique de communiquer un *access_token* en tant que valeur du *header authorization* de la requête *HTTP* (en mode *Bearer*).

Il est donc nécessaire de connecter l'utilisateur afin d'obtenir un ***access_token***, avant d'interagir avec le endpoint ***/items/likes***.

- **Create** : Ajout d'un film à la liste des likes

```
POST /items/likes HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "movie_id":<movie_id>,
  "title":"Kingdom of the Planet of the Apes",
  "watched":true,
  "note":3
}
```

- **Read all** : Lecture de la liste des films ajoutés à la liste des likes

```
POST /items/likes HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json
```

- **Read by id** : Lecture d'un film ajouté à la liste des likes

```
POST /items/likes/<like_id> HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json
```

- **Delete** : Suppression d'un film à la liste des likes

```
POST /items/likes/<like_id> HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json
```

- **Update** : Ajout d'un film à la liste des likes

```
PATCH /likes/<like_id> HTTP/1.1
Host: fakeflix.shrp.dev
Accept: application/json

{
  "watched":false,
  "note":null
}
```

Mode Offline

- Gérez le fonctionnement de l'application en **mode *offline*** afin de permettre à l'utilisateur d'afficher une liste de films à partir des données préalablement obtenues auprès de l'API TMDB et gérer sa liste de ***likes***.

- Pour ce faire, installez le package ***connectivity_plus*** (https://pub.dev/packages/connectivity_plus) permettant de détecter l'absence de réseau et le package ***sqflite*** (https://pub.dev/packages/sqflite) permettant de disposer d'une base de données locale de type ***SQLite*** dans laquelle vous stockerez les données obtenues auprès de l'API lorsque le réseau est disponible.

- Faites en sorte qu'un utilisateur connecté puisse employer l'application et interagir avec les données stockées localement lorsqu'une connexion au réseau est disponible, puis synchronisez les données lorsque la connexion est à nouveau disponible.