P3. Movie

> Movie is a money spending and analysis tracking app.
> Using this app, we trace how much we spend on what items.

1. Data Structure

- Movie class
  - Conversion from/to Json

Movie class

- Movie constructor gets input from arguments and stores the information.

```
class Movie {
  late int id;
  late String title;
  late double voteAverage;
  late String releaseDate;
  late String overview;
  late String posterPath;

  Movie({required this.id, ... required this.posterPath});
```

Movie.fromJson named constructor

- This `named constructor` generates the Movie object from Json.

```
Movie.fromJson(Map<String, dynamic> parsedJson) {
  this.id = parsedJson['id'] as int;
  this.title = parsedJson['title'] as String;
  this.voteAverage = (parsedJson['vote_average'] as double);
  this.releaseDate = parsedJson['release_date'] as String;
  this.overview = parsedJson['overview'] as String;
  this.posterPath = parsedJson['poster_path'] as String;
}
```

2. Helper Functions

- TMDB API
- APIRunner

TMDB API

- In this app, we retrieve movie information from the https://image.tmdb.org API.
- Students need to create a free account on The Movie Database (TMDB) https://www.themoviedb.org/signup.
- Navigate to your account

Request the API Key

- You'll then be prompted to select your API usage type (e.g., "Developer") and agree to the terms of use.
- You'll receive an API key once you've provided the necessary information.

APIRunner (api.dart)

- You should use your API to run the application.

```
class APIRunner {
  final String api_key = 'api_key=<<YOUR API>>';
```

runAPI

- This function is the service function to use the TMDB API.
- The information is in Json format.

```
Future<List?> runAPI(API) async {
  http.Response result = await http.get(Uri.parse(API));
  if (result.statusCode == HttpStatus.ok) {
    final jsonResponse = json.decode(result.body);
    final moviesMap = jsonResponse['results'];
    return moviesMap.map((i) => Movie.fromJson(i)).toList();
  } else {
    return null;
  }
}
```

getUpcoming and searchMovie

- Using runAPI, we can make service functions to access TMDB APIs.

```
Future<List?> getUpcoming() async {
  final String upcomingAPI = urlBase + apiUpcoming + api_key + urlLanguage;
  return runAPI(upcomingAPI);
}

Future<List?> searchMovie(String title) async {
  final String search = urlBase + apiSearch + api_key + '&query=' + title;
  return runAPI(search);
}
```

search method using the service function

- The service searchMovie function we made can be used to build other functions.

```
Future search(text) async {
  movies = (await helper?.searchMovie(text))!;
  setState(() {
    moviesCount = movies?.length;
    movies = movies;
  },);
}
```

2. User Interface

- main.dart
- MovieList (movie_list.dart)
- MovieDetail (movie_detail.dart)

main.dart

- The main widget lists movie information retrieved using the TMDB API.

- It uses the MaterialApp design.
- Its theme has a simple setup in ThemeData.

```
void main() => runApp(MyMovies());
class MyMovies extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.deepOrange,
      ),
      home: MovieList(),
    );
  }
```

MovieList

Scaffold

- It has an appBar and a body.

```
return Scaffold(
  appBar: AppBar(
    title: searchBar,
    actions: <Widget>[IconButton(...),]
  ),
  body: ListView.builder(
    itemBuilder: (BuildContext context, int position) {
      return Card(...)
    }
  ),
),
```

Scaffold: appBar

- The AppBar has a search button on the right side of the bar.
- When the button is clicked, change the button icon so users can cancel the action.

```
if (this.visibleIcon.icon == Icons.search) {
  this.visibleIcon = Icon(Icons.cancel);
```

- When users give an input, the input is used to invoke the search service function we made.

```
this.searchBar = TextField(
  textInputAction: TextInputAction.search,
  onSubmitted: (String text) {
    search(text);
  },
  ...
}
```

Scaffold: body

- The movies to display are in the movies list.
- In this code, we use itemBuilder to create Cards with movie information.

```
body: ListView.builder(
  itemCount: this.moviesCount,
  itemBuilder: (BuildContext context, int position) {
    image = NetworkImage(iconBase + movies?[position].posterPath);
    return Card(
      child: ListTile(
        onTap: () {...}
        title: Text(movies?[position].title),
        subtitle: Text(...),
      ),
```

- When users click the Card, the detailed movie information is shown using the route.

```
onTap: () {
  MaterialPageRoute route = MaterialPageRoute(
    builder: (_) => MovieDetail(movies?[position]));
  Navigator.push(context, route);
}
```

search function

- The search function uses the searchMovie to get the movies.
- It uses the setState function to redraw widgets.

```
Future search(text) async {
  movies = (await helper?.searchMovie(text))!;
  setState(() {
    moviesCount = movies?.length;
    movies = movies;
  },);
}
```

initialize function

- When the program starts, it uses the getUpcoming function to get the newest movies.

```
Future initialize() async {
  movies = (await helper?.getUpcoming())!;
  setState(() {
    moviesCount = movies?.length;
    movies = movies;
  },);
}
```

MovieDetail (movie_detail.dart)

- This widget shows the detailed movie information when users click the Card widget of the Movie widget.

Widget Structure

- It is a Scaffold with an AppBar and a SingleChildScrollView body.

```
return Scaffold(
  appBar: AppBar(
    title: Text(movie.title),
  ),
  body: SingleChildScrollView(
    child: Center(
      child: Column(
        children: <Widget>[...],
      )
```

- The movie information is already given through the constructor.
- So, we can get the movie image and display it.

```
Container(
  height: height / 1.5,
  child: Image.network(path)),
Container(
  child: Text(movie.overview),
)
```

3. Program Structure

- This application uses MVC (Model-View-Controller) architecture.

```
lib
├── main.dart
├── model
|   └── movie.dart
├── util
|   └── api.dart
└── view
    ├── movie_detail.dart
    └── movie_list.dart
```

Programming Tips

Protective Code - null safety

- We should consider all the possible error cases.
- In this example, we may have a null string, so we should make the code more robust.

```
this.title = parsedJson['title'] as String? ?? '';
this.voteAverage = (parsedJson['vote_average'] as double?) ?? 0.0;
```

Self-grading for HW

- You analyze the whole code once (30%).
- You analyze the whole code twice using a different method (60%).
  - Make a summary of widgets that you did not know before (what and how to use them).
- You understand how the code works (80%).
- You can use the programming techniques in this example to make team and individual