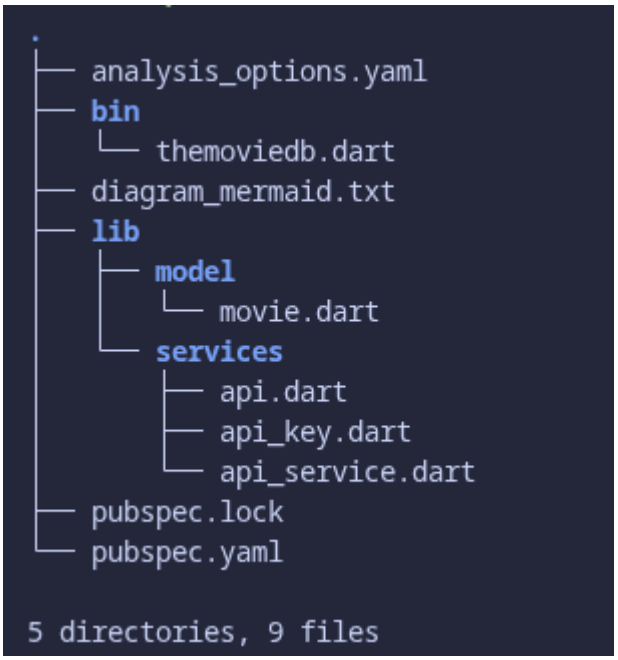
Dart - The movie DB

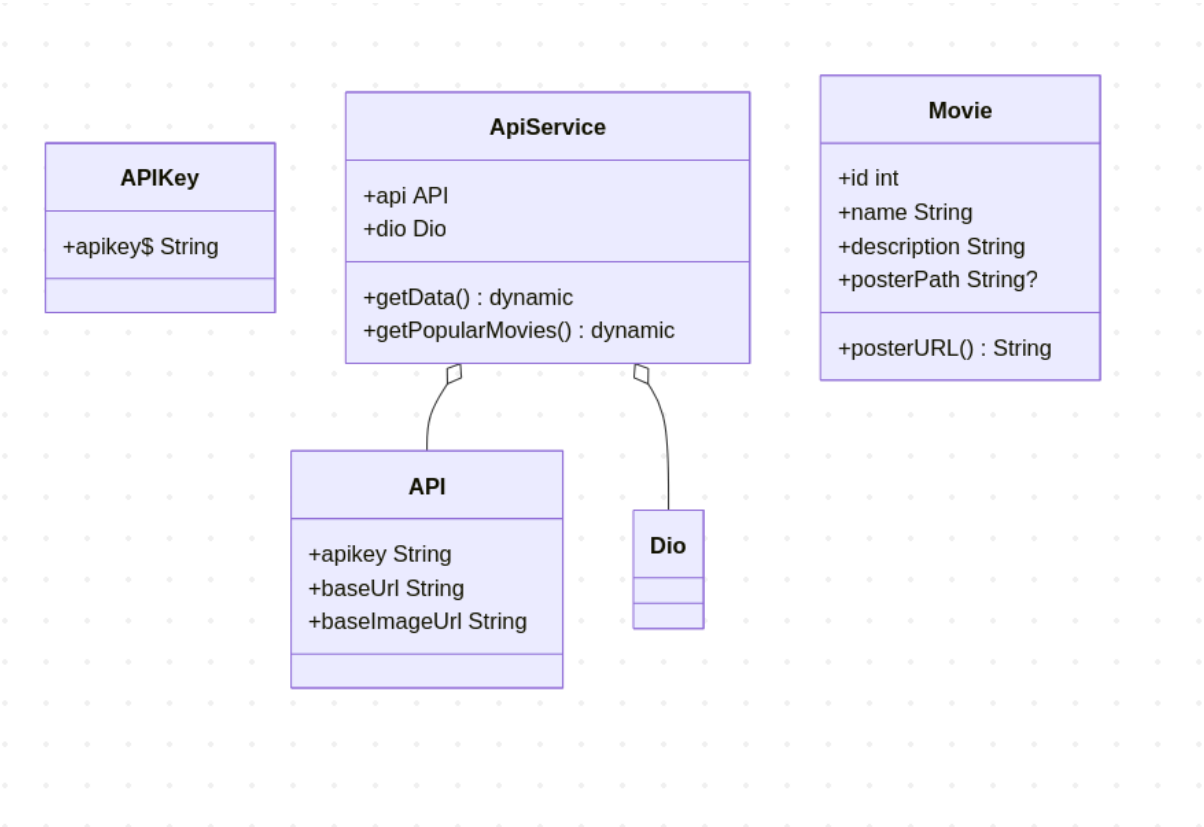
1- Analyse du projet

1.1 Analyse de l’arborescences

Suivant l’architecture de fichier du projet nous pouvons remarquer qu’il peut répondre à un point du paradigme MVC. Ce point est le Model effectivement movie.dart repond à ce point là. Par contre Controller et View sont manquante car le projet s'exécute uniquement dans le terminal et donc il n’y a rien a afficher.



1.2 Analyse des classes



Voici ci dessus le résultat du diagramme de classe mermaid du projet généré via dcdg.

On peut voir d'après le diagramme que le projet utilise la librairie DIO afin de faire les requêtes HTTP à l'API.

1.3 Analyse de l'exécution

L'erreur 401 est renvoyée car The movie DB demande une clé API, qui peut être générer sur ce site : https://www.themoviedb.org/settings/api . Une fois généré, il suffit de la renseigner dans l'attribut apiKey de la classe APIKey.

1.4 Analyse de cybersécurité

La seul classe qui n'est pas suivi en version est la classe APIKey, ce choix à été pris afin de ne pas sauvegardé la clé d'api qui est personnel au développeur et si éventuellement d'autre développeurs se mettent à travailler à leur tours sur le projet ils leurs faudra aussi une clé API.

2- API REST TMDB

2.1 A ce stade, notre code n'est pas encore fonctionnel. ➜ Dans la fonction main(), modifier le code en utilisant un bloc "try and catch" afin de traiter proprement la levée d'exception. Vous indiquerez la démarche dans votre compte rendu.

```
// Lancement de la requète
try {
  final response = await dio.get(url, queryParameters: query);

  if (response.statusCode == 200) {
    return response;
  } else {
    throw response;
  }

} catch (e) {
  throw Exception(e);
}
```

3- Gestion de bogue et refactoring

Une erreur à été commise dans la méthode getPopularMovies(), en effet dans la boucle foreach l'instanciation de l'objet Movie prend plusieur paramètre dont posterPath qui était égal à json['posterPath'] alors que la clé devrait être poster_path.

```
Movie jsonToMovie(Map<String, dynamic> json) {
  Movie movie = Movie(
      id: json['id'] as int,
      name: json['title'] as String,
      description: json['overview'] as String,
      posterPath: json['poster_path'] ?? '');
  return movie;
}
```

4- Aller plus loin

Afin de créer une application flutter il est nécessaire de modifier la classe _MyHomePageState afin d'y ajouter un future builder qui viendra appelé la méthode getFirstPopMovie() et enfin affiché l'image via internet.

```dart
class _MyHomePageState extends State<MyHomePage> {
  ApiService service = ApiService();

  Future<Movie> getFirstPopMovie() async {
    List<Movie> popMovies = await service.getPopularMovies(1);
    return popMovies[0];
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ), // AppBar
      body: Center(
        child: FutureBuilder<Movie>(
          future: getFirstPopMovie(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const CircularProgressIndicator();
            }

            if (snapshot.hasError) {
              return Text('Error: ${snapshot.error}');
            }

            if (!snapshot.hasData) {
              return const Text("No movie found");
            }

            Movie movie = snapshot.data!;

            return Image.network(movie.posterURL());
          },
        ), // FutureBuilder
      ), // Center
    ); // Scaffold
  }
}
```

Structure du projet flutter