Flutter - 웹툰 앱 만들기

상태 플러터

```
const App({super.key});
```

이 코드 많이 보는데 이건 단지 이 위젯의 key를 stateless widget이라는 슈퍼클래스에 보냈다 그런거래. 위젯은 ID같은 식별자 역할을 하는 key가 있다는 것만 알아두면 된다고 함! 플러터가 위젯을 빨리 찾을 수 있게 하는 역할이래. 식별하기 위해서!!

패키지 설치

어떤 메서드가 어떤 타입을 반환할 지 알기 어려울 때는 Future를 쓰면 되는데 일단 여기서는 넘어감.

Future : 당장 완료될 수 없는 작업이 아니다.

api를 받아와서 JSON을 콘솔에 프린트해보자.

url에서 받아오고 싶다면 http 패키지를 설치해야함

Dart packages

Pub is the package manager for the Dart programming language, containing reusable libraries & packages for Flutter and general Dart programs.

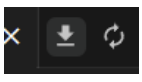
https://pub.dev/

여기는 공식 Dart, Flutter 패키지 보관소.

npm이나 PyPI랑 비슷하다.

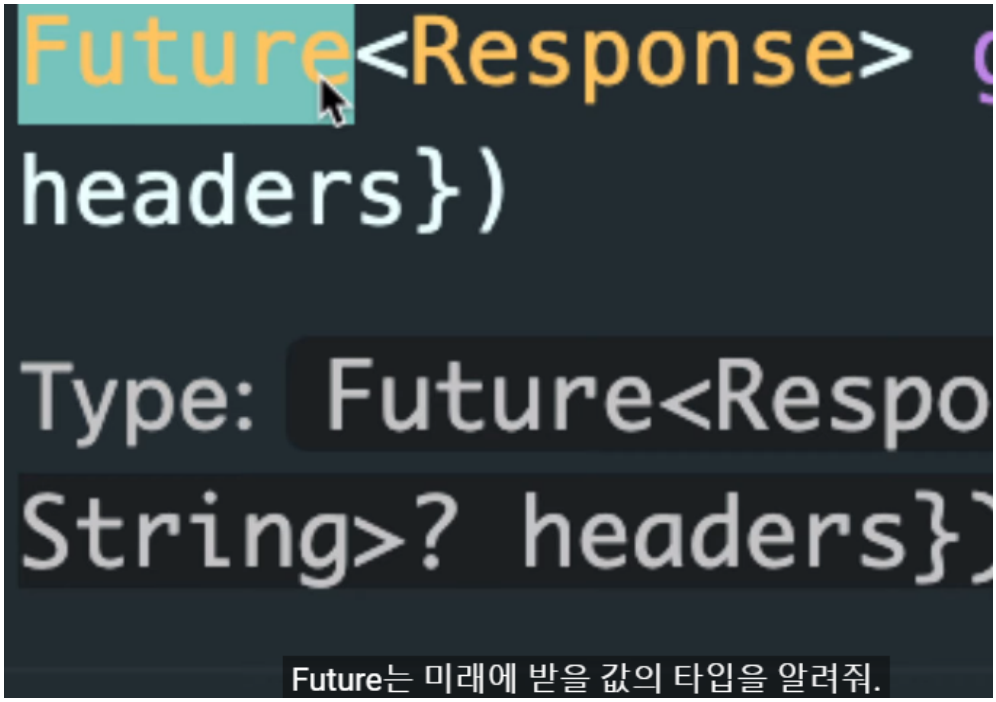
설치하는 방식은 여러가지 있는데 pubspec.yaml 파일에 붙여넣으면 버전정보도 알 수 있으니까 그렇게 하자.

짠



이거 눌러주면 됨 저 아래로 가는 화살표

▼ 데이터 붙이기




```
import 'package:http/http.dart' as http;

class ApiService {
  final String baseUrl = "https://webtoon-crawler.nomadcoders.workers.dev";
  final String today = "today";

  void getTodaysToons(){
    final url = Uri.parse("$baseUrl/$today")
    http.get(url);
  }

}
```

암튼 코드는 이렇게 생겼음. 근데 저 get(url)에 얼마나 시간이 걸릴 지 모르는데, Dart는 저 부분이 제대로 완료될 때까지 기다리려고 안함. 근데 우린 기다려서 다 실행됐으면 좋겠잖아 **비동기** 프로그래밍...

서버가 응답할 때까지 프로그램을 기다리게 하는 것이다.

그래서 `await`를 사용해야해. 비동기 함수 내에서만 사용될 수 있다는 규칙만 지켜서 사용하면 돼. 그건 `async`를 붙이면 되고.

```
void getTodaysToons()async {
    final url = Uri.parse("$baseUrl/$today");
    await http.get(url);
  }
```

정리! Future라고 뜨면 바로 처리되는 것이 아닌 나중에 완료된다는 뜻.

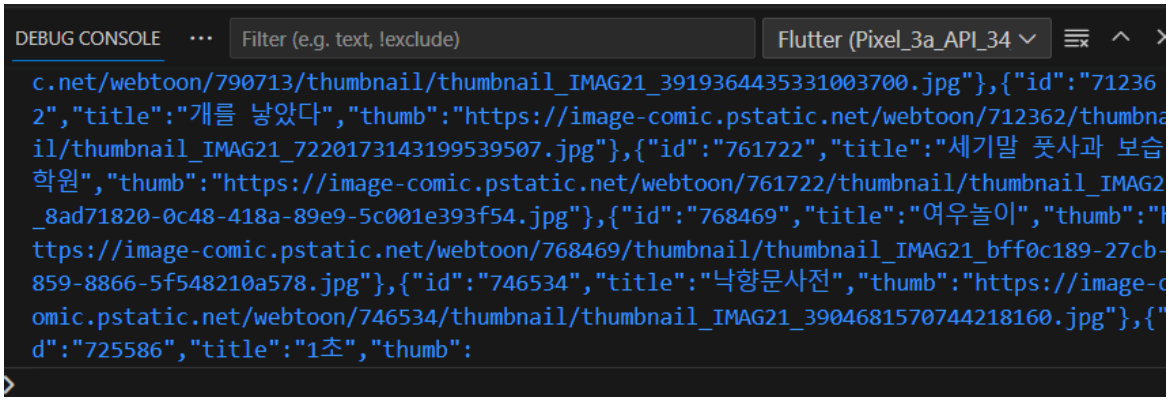
그치만 완료되면 Response(서버에 대한정보?)가 담겨있을 것이라는 것을 알 수 있어!!

우리는 그 response를 받을 거니까… 아래와 같이 수정해야해.

걍… future가 기다렸다 완료될 때 response 타입으로 준대.

```
import 'package:http/http.dart' as http;

class ApiService {
  final String baseUrl = "https://webtoon-crawler.nomadcoders.workers.dev";
  final String today = "today";

  void getTodaysToons() async {
    final url = Uri.parse("$baseUrl/$today");
    final response = await http.get(url);

    if (response.statusCode == 200) {
      print(response.body);
      return;
    }
    throw Error();
  }
}
```

이제 콘솔에 출력됨!




```
void main() {
  ApiService().getTodaysToons();
  runApp(const App());
}
```

임시방편으로 main에서 돌려본거야 이거

▼ JSON을 Dart와 Flutter에서 쓸 수 있는 데이터 형식인 클래스로 바꾸기

위에서 사용했던 api service 파일을 조금 수정한다.

```
if (response.statusCode == 200) {
      jsonDecode(response.body);
      return;
    }
```

jsonDecode! 이미 만들어져있대 좋아~ 근데 이것의 반환값은 dynamic임.

어떤 타입이든 될 수 있어서 우리가 직접 타입을 정해줘야함.

보면 우리가 받는 값은 여러개의 { } 오브젝트로 이루어진 [ ] 리스트라고 할 수 있음. 그래서 아래처럼바꿔주면

```
if (response.statusCode == 200) {
      final List<dynamic> webtoons = jsonDecode(response.body);
      for (var webtoon in webtoons) {
        print(webtoon);
      }
      return;
    }
```

하나씩 잘 떨어져서 출력된다~ 플러터는 출력할 때마다 플러터를 붙이니까 하나씩 출력됐단 것이 확인됨!

```
import 'dart:convert';

import 'package:http/http.dart' as http;
import 'package:webtoonlearn/model/webtoon_model.dart';

class ApiService {
  final String baseUrl = "https://webtoon-crawler.nomadcoders.workers.dev";
  final String today = "today";

  Future<List<WebtoonModel>> getTodaysToons() async {
    List<WebtoonModel> webtoonsInstances = [];
    final url = Uri.parse("$baseUrl/$today");
    final response = await http.get(url);

    if (response.statusCode == 200) {
      final List<dynamic> webtoons = jsonDecode(response.body);
      for (var webtoon in webtoons) {
        webtoonsInstances.add(WebtoonModel.fromJson(webtoon));
      }
      return webtoonsInstances;
    }
    throw Error();
  }
}
```

이렇게 하면 딱딱 떨어진 웹툰 정보들이 list로 저장됨!

근데 왜 Future로 감쌌을까? 그건 바로 비동기함수라서… async 붙었으니까 걍 외워두자

이제 이 클래스의 모든 메서드와 프로퍼티를 static로 바꿀거래 왜냐면 여기 state가 없어서??

▼ 여기까지 home code

home에 웹툰 리스트를 불러오고, 로딩 중인지 아닌지도 써주고 그렇기 때문에 stateful 위젯을 사용함.

```
import 'package:flutter/material.dart';
import 'package:webtoonlearn/model/webtoon_model.dart';
import 'package:webtoonlearn/services/api_service.dart';
```

```
class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  List<WebtoonModel> webtoons = [];
  bool isLoading = true;

  void waitForWebToons() async {
    //HTTP 응답 기다려야해서 비동기 함수로 처리
    webtoons = await ApiService.getTodaysToons();
    isLoading = false;
    setState(() {});
  }

  @override
  void initState() {
    super.initState();
    waitForWebToons();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        centerTitle: true,
        elevation: 2,
        title: const Text(
          "오늘의 웹툰",
          style: TextStyle(
            fontSize: 26,
            fontWeight: FontWeight.w400,
          ),
        ),
        foregroundColor: Colors.green,
        backgroundColor: Colors.white,
      ),
    );
  }
}
```

---

state를 쓰지 않는 코드로 바꿔줄거래. 왜냐면 state를 쓰지 않는 것이 좋기 때문이야.

사실 ApiService 파일은 만들지 않아도 되지만… 근데 있는게 깔끔하지 않나

필요한 데이터를 fetch하는 다른 방법은 무엇일까

stateless 위젯 상태에서 fetch를 할 수 있대!

그것은 바로 FutureBuilder라는 widget을 사용하는 것! 알아서 기다려줌~~

```
body: FutureBuilder(
        future: webtoons,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return const Text("There is data!");
          }
          return const Text("Loading...");
        },
      ),
```

이렇게 사용하면 됨.

snapshot은 future의 상태를 알 수 있다. future가 데이터를 받았는지 아니면 오류가 났는지 알 수 있다. connection state도 알 수 있고… snapshot은 이름 바꿔도 됨. future나 그런걸로 공식문서 따르면 snapshot 맞긴해

▼ 지금까지 코드

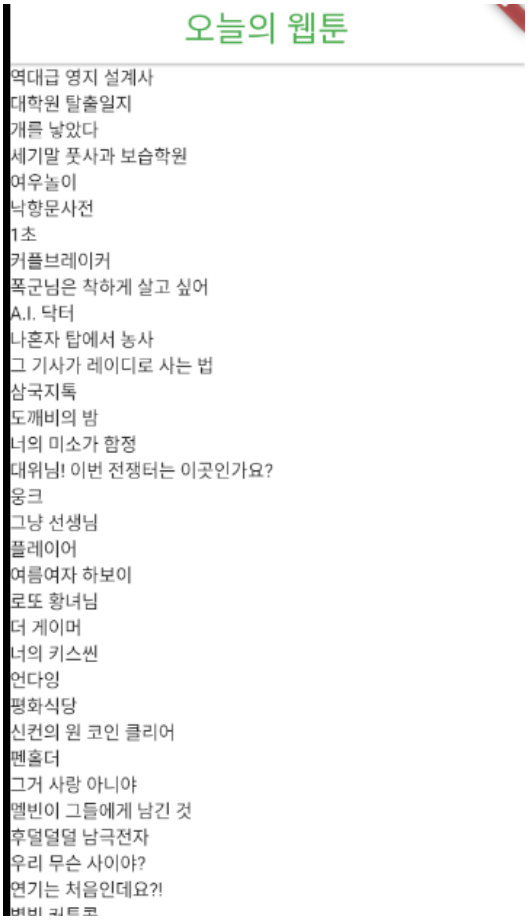
```
import 'package:flutter/material.dart';
import 'package:webtoonlearn/model/webtoon_model.dart';
import 'package:webtoonlearn/services/api_service.dart';

class HomeScreen extends StatelessWidget {
  HomeScreen({super.key});

  final Future<List<WebtoonModel>> webtoons = ApiService.getTodaysToons();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        centerTitle: true,
        elevation: 2,
        title: const Text(
          "오늘의 웹툰",
          style: TextStyle(
            fontSize: 26,
            fontWeight: FontWeight.w400,
          ),
        ),
        foregroundColor: Colors.green,
        backgroundColor: Colors.white,
      ),
      body: FutureBuilder(
        future: webtoons,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return const Text("There is data!");
          }
          return const Text("Loading...");
        },
      ),
    );
  }
}
```

많은 양의 데이터를 보여주고 싶을 때, Row나 Column은 적절하지 않다.

ListView를 쓰는 것이 좋다!

```
builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView(
              children: [
                for (var webtoon in snapshot.data!) Text(webtoon.title)
              ],
            );
          }
```

이렇게 쓰면

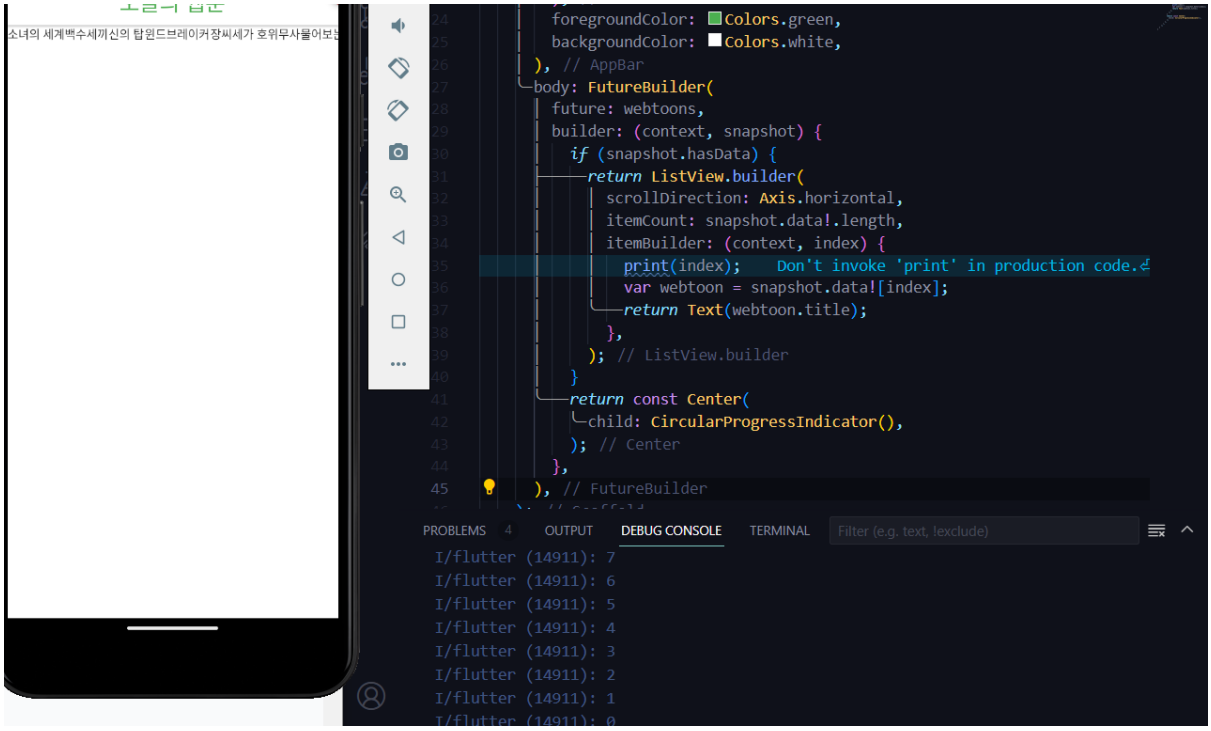


이렇게 싸악 보여주는데, 사실 이건 그렇게 좋지 않음.

인스타그램이나 틱톡같은거 미리 모든 영상 받아오면 어케되겠어? 사용자가 보고있는 사진이나 섹션만 로딩해야해.

조금 다른 ListView를 써보자. 그건 바로 builder

이건 스크롤 방향도 바꿀 수 있다!



스크롤 방향을 가로로 바꾸고, 옆으로 넘길 때마다 로딩되게 했다.

builder는 리스트의 아이템을 만들어주는데, 이 만들어야 하는 아이템의 index를 받아서 그거 만든다.

그런데 여기서 또 새로운 것을 써보자.

Separator builder

builder에서 새로운 인자를 하나 더 가진다.

위젯! 그래서 그 위젯을 리스트 아이템 사이에 렌더 시킨다.

sized box를 넣었더니 이렇게 분리가 돼~~

그리고 처음에는 sized box를 안 넣었음 짱임.

```
body: FutureBuilder(
        future: webtoons,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.separated(
              scrollDirection: Axis.horizontal,
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                print(index);
                var webtoon = snapshot.data![index];
                return Text(webtoon.title);
              },
              separatorBuilder: (context, index) => (const SizedBox(
                width: 20,
              )),
            );
```

```
          }
          return const Center(
            child: CircularProgressIndicator(),
          );
        },
```

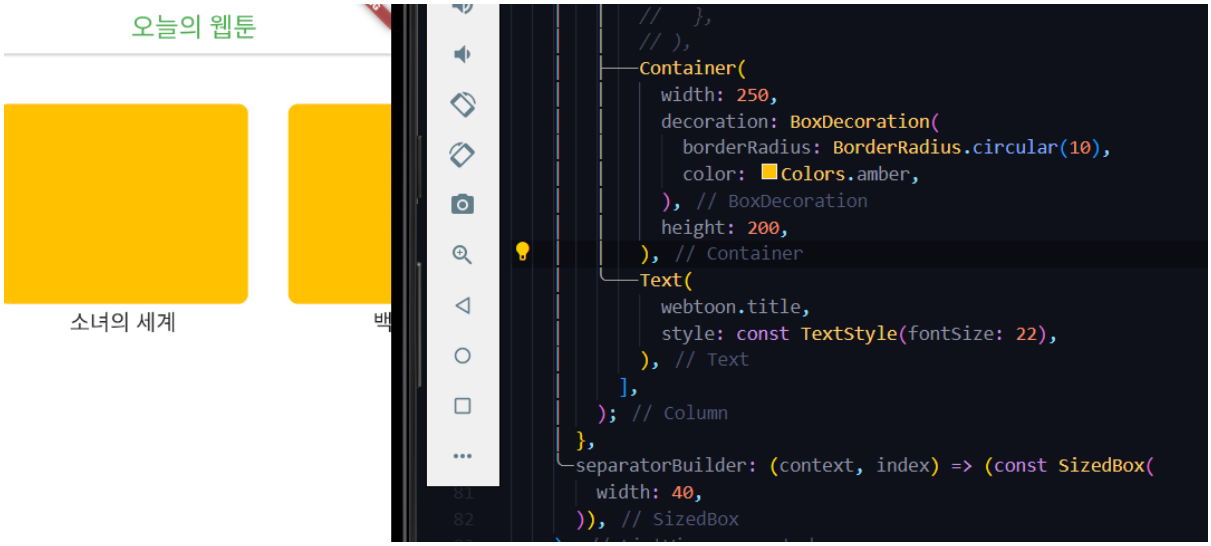
Webtoon Card

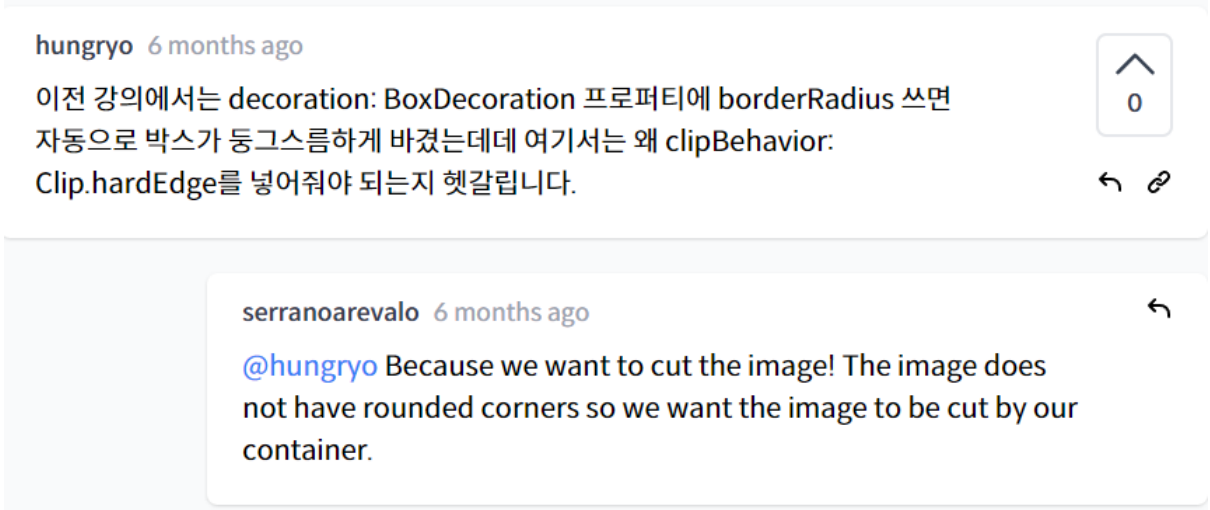
먼저 리스트를 사용할 때, Column은 안에 있는 리스트뷰의 높이가 어느 정도인지 모름. 그래서 ListView에 제한된 높이를 줘야 함

Expanded : 화면의 남는 공간을 차지하는 위젯

사진 첨부가 안되어서 임시로 노랗게 칠함.

그런데 이미지를 쓰면 이 코드만으로 모서리가 둥글게 깎이지 않는대. 이건 오버플로우에 대한 어쩌고로… clipBehavior 쓰면 됨. Clip.hardEdge로!!





그림자 넣기!

```
decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10),
                color: Colors.amber,
                boxShadow: [
                  BoxShadow(
                    blurRadius: 15,
                    offset: const Offset(10, 10),
                    color: Colors.black.withOpacity(0.5),
                  )
                ]),
```

Detail Screen

이제 썸네일 누르면 상세 페이지로 이동하도록~

GestureDetector : 대부분의 동작을 감지하게 해 줌

onTap: 버튼을 탭 했을 때 발생하는 이벤트 (onTapUp + onTapDown)

다른 페이지로 넘기려면 뭘 써야 할까 그것은 바로 Navigator.push

이걸 쓰는 이유는! 페이지 넘길 때 애니메이션 효과를 적용해서 유저가 다른 페이지로 왔다고 느끼게 해줄 수 있기 때문임!!

context:

route : stateless widget을 애니메이션 효과로 감싸서 스크린처럼 보이겠다하는 것

MaterialPageRoute는 stateless 위젯을 를 route로 감싸서 쓸 수 있게 해 줌. 왜나면 navigator.pust는 stateless 위젯을 원하지 않아서 route로 감싸야 해…

```
return GestureDetector(
      onTap: () {
        Navigator.push(context, MaterialPageRoute(builder: (context) => DetailScreen(title: title, thumb: thumb, id: id),))
      },)
```

이미지가 카드 형식이면 상단바 왼쪽에 ← 표시가 뜨는데,

```
onTap: () {
        Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) =>
                  DetailScreen(title: title, thumb: thumb, id: id),
              fullscreenDialog: true,
            ));
      }
```

fullscreenDialog:true 쓰면 X로 바뀌면서 아래에서 올라오는 애니메이션 적용

표지에도 애니메이션을 넣을 수 있다.

안드에서 옆에서 넘어오게 하려면 아래 코드

```
Navigator.push(
          context,
          PageRouteBuilder(
            transitionsBuilder:
                (context, animation, secondaryAnimation, child) {
              var begin = const Offset(1.0, 0.0);
              var end = Offset.zero;
              var curve = Curves.ease;
              var tween =
                  Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
              return SlideTransition(
                position: animation.drive(tween),
                child: child,
              );
            },
            pageBuilder: (context, anmation, secondaryAnimation) =>
                DetailScreen(id: id, title: title, thumb: thumb),
          ),
        );
      },
```

아래서 올라오게 하는 코드

```
Navigator.push(
          context,
          PageRouteBuilder(
            transitionsBuilder:
                (context, animation, secondaryAnimation, child) {
              var begin = const Offset(0.0, 1.0);
              var end = Offset.zero;
              var curve = Curves.ease;
              var tween =
                  Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
              return SlideTransition(
                position: animation.drive(tween),
                child: child,
              );
            },
            pageBuilder: (context, anmation, secondaryAnimation) =>
                DetailScreen(id: id, title: title, thumb: thumb),
          ),
        );
      },
```

https://github.com/ggamsi0418/flutter_for_beginners/blob/60b430b9b2d7d74672ecdd0fd1b9ee50afb1266a/lib/widgets/webtoon_widget.dart#L24

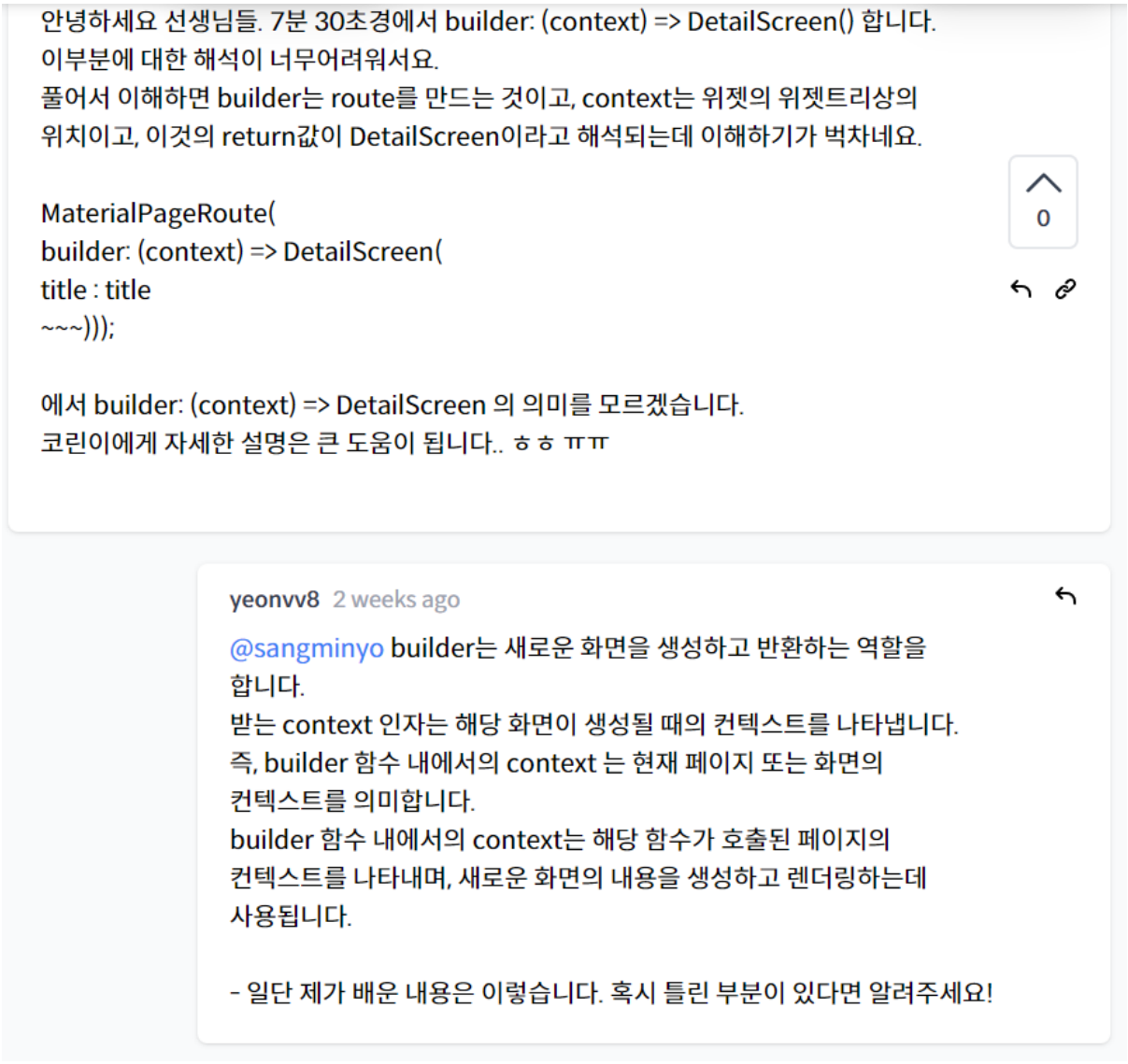
위를 참고~

운영체제에 구애받지 않고 적용하고 싶다면 아래 참고

Animate a page route transition

How to animate from one page to another.

https://docs.flutter.dev/cookbook/animation/page-route-animation



Hero widget

화면을 전환할 때 애니메이션 제공.

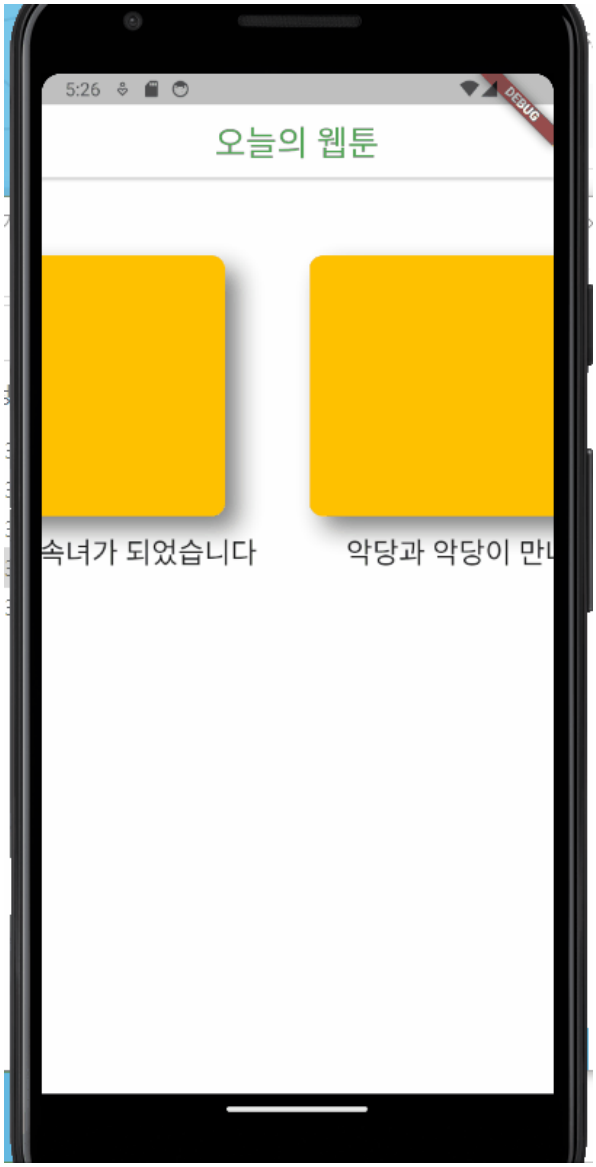
만약 두 화면에서 같은 이미지를 사용하고 있으면 이 두 화면이 전환될 때 같은이미지니까 애니메이션을 주면 멋질 것 같잖아 그래서야

각각의 이미지 위젯에 같은 태그를 주기만 하면 된다~~~~

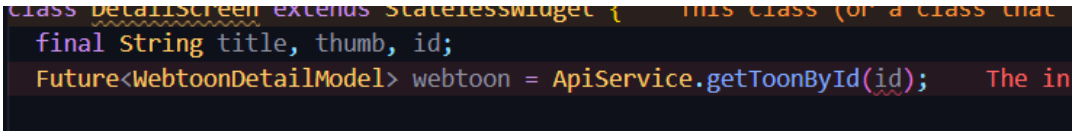
이거 너무 쩔어줌...

```
Hero(
              tag: id,
              child: Container(
                width: 250,
```

끝이야



이게 어케 되는 거야????????



이번엔 id 를 사용해서 상세 페이지에 접근해야하는데, property를 초기화 할 때 다른 property(여기서는 id)에 접근이 안돼!

우선 stateful widget이 되어야 한다…

```
class _DetailScreenState extends State<DetailScreen> {
late Future<WebtoonDetailModel> webtoon;

  @override
  void initState() {
    super.initState();
    webtoon = ApiService.getToonById(widget.id);
  }
```

그리고 이거 써줘야 함

안전하게 Future를 초기화 할 수 있게 됐다.

여기서 잠깐!!

리스트는 정말 좋지만 컨텍스트, 인덱스 등등 생각해야하는 것이 많아서 까다롭다. 렌더링 할 요소의 길이가 짧고 길이를 안다면 column 같은거 쓰는게 나을 것.

ㅋㅋ setState 적는 것 잊지 말도록 해~~

▼ detail 코드

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:url_launcher/url_launcher_string.dart';
import 'package:webtoonlearn/model/webtoon_detail_model.dart';
import 'package:webtoonlearn/model/webtoon_episode_model.dart';
import 'package:webtoonlearn/services/api_service.dart';
import 'package:webtoonlearn/widgets/episode_widget.dart';

class DetailScreen extends StatefulWidget {
  final String title, thumb, id;

  const DetailScreen({
    super.key,
    required this.title,
    required this.thumb,
    required this.id,
  });

  @override
  State<DetailScreen> createState() => _DetailScreenState();
}

class _DetailScreenState extends State<DetailScreen> {
  late Future<WebtoonDetailModel> webtoon;
  late Future<List<WebtoonEpisodeModel>> episodes;
  late SharedPreferences prefs;
  bool isLiked = false;

  Future initPrefs() async {
    prefs = await SharedPreferences.getInstance();
    final likedToons = prefs.getStringList('likedToons');
    if (likedToons != null) {
      // 최초로 실행했을 때 만들어줬으니까 이후는 계속 여기로 들어옴.
      // 그럼 이제 해당 웹툰의 ID가 이 리스트 안에 있는가??
      if (likedToons.contains(widget.id) == true) {
        // widget.id를 사용하는 이유, state가 아닌 DetailScreen의 Id를 가져오기 위해서.
        setState(() {
          isLiked = true;
          print("$likedToons $isLiked");
        });
      }
    } else {
      await prefs.setStringList('likedToons', []);
```

```
    // 사용자가 앱을 최초로 실행한 그 순간에 만들어준다.
  }
}

@override
void initState() {
  super.initState();
  initPrefs();
  webtoon = ApiService.getToonById(widget.id);
  episodes = ApiService.getLatestEpisodesById(widget.id);
}

onHeartTap() async {
  final likedToons = prefs.getStringList('likedToons');
  if (likedToons != null) {
    if (isLiked) {
      likedToons.remove(widget.id);
    } else {
      likedToons.add(widget.id);
    }
  }
  if (likedToons is List<String>) {
    await prefs.setStringList('likedToons', likedToons);
  }

  setState(() {
    isLiked = !isLiked;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
      centerTitle: true,
      elevation: 2,
      title: Text(
        widget.title,
        style: const TextStyle(
          fontSize: 26,
          fontWeight: FontWeight.w400,
        ),
      ),
      foregroundColor: Colors.green,
      backgroundColor: Colors.white,
      actions: [
        IconButton(
          onPressed: onHeartTap,
          icon:
              Icon(isLiked ? Icons.favorite : Icons.favorite_border_outlined),
        )
      ],
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(50),
        child: Column(children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Hero(
                tag: widget.id,
                child: Container(
                  width: 250,
                  decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(10),
                      color: Colors.amber,
                      boxShadow: [
                        BoxShadow(
                          blurRadius: 15,
                          offset: const Offset(10, 10),
                          color: Colors.black.withOpacity(0.5),
                        )
                      ]),
                  height: 200,
                ),
              ),
            ],
          ),
```

```
            const SizedBox(
              height: 25,
            ),
            FutureBuilder(
                future: webtoon,
                builder: (context, snapshot) {
                  if (snapshot.hasData) {
                    return Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Text(
                          snapshot.data!.about,
                          style: const TextStyle(
                            fontSize: 16,
                          ),
                        ),
                        const SizedBox(
                          height: 15,
                        ),
                        Text(
                          '${snapshot.data!.genre} / ${snapshot.data!.age}',
                          style: const TextStyle(
                            fontSize: 16,
                          ),
                        ),
                        const SizedBox(
                          height: 30,
                        ),
                        FutureBuilder(
                            future: episodes,
                            builder: (context, snapshot) {
                              if (snapshot.hasData) {
                                return Column(
                                  children: [
                                    for (var episode in snapshot.data!)
                                      Episode(
                                        episode: episode,
                                        webtoonId: widget.id,
                                      )
                                  ],
                                );
                              }
                              return Container();
                            })
                      ],
                    );
                  }
                  return const Text(". . .");
                })
          ]),
        ),
      ),
    );
  }
}
```