PANDUAN TUGAS 5

Membuat Tampilan MusicPlayer

Capaian

1. Mahasiswa mampu mengimplementasikan interaksi pada audio seperti memutar, menjeda, dan menghentikan musik dengan memanfaatkan package audioplayers.
2. Mahasiswa dapat mengimplementasikan interaksi pada audio baik untuk audio yang bersumber dari asset lokal maupun dari internet.
3. Mahasiswa dapat menampilkan Text widget dengan style yang memanfaat package google_fonts.
4. Mahasiswa dapat menggunakan Icon widget.
5. Mahasiswa mampu menggunakan Image widget untuk menampilkan asset gambar serta memanfaatkan package cached_network_image untuk menampilkan gambar dari internet.

Spesifikasi Hardware dan Software

Memiliki komponen perangkat keras dan perangkat lunak yang benar sangat penting untuk memastikan keberhasilan pelaksanaan tugas yang diuraikan dalam panduan ini. Konfigurasi perangkat keras dan perangkat lunak yang diperlukan untuk menyelesaikan tugas panduan ini adalah sebagai berikut:

A. Spesifikasi Minimum Hardware
    1. Minimum RAM 4 GB, disarankan RAM 8 GB
    2. Minimum 15 GB ruang disk yang tersedia (2 GB untuk Flutter SDK, 8 GB untuk Android Studio, 4 GB untuk AVD, dan 1 GB untuk proyek)
    3. Resolusi layar minimum 1280 x 800
    4. Emulator Mobile (Android/IOS)

B. Software
    1. Flutter SDK (versi 3.13.0 atau yang lebih baru) dan Dart (versi 3.1.5 atau yang lebih baru)
    2. Android Studio
    3. Visual Studio Code

Sumber Daya

1. Test File Tugas 3A
2. Test File Tugas 3B

Deskripsi Tugas

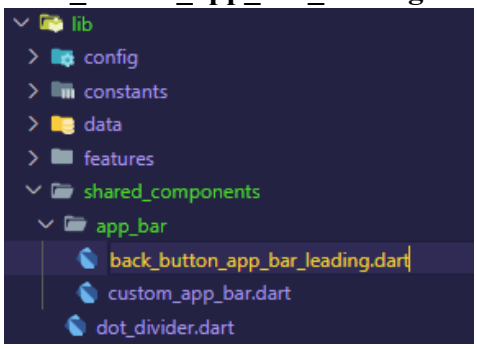
Mahasiswa akan menuliskan kode pada proyek Flutter media_player untuk mengimplementasikan komponen UI dasar multimedia. Mahasiswa perlu memahami cara pembuatan komponen yang dapat menampilkan gambar baik dari asset lokal maupun internet, teks, icon, serta berbagai dekorasi yang dapat ditambahkan. Dalam panduan ini mahasiswa

akan membuat komponen BackButtonAppBarLeading, MusicCoverImage, TimeDisplay, ControllButton, dan halaman MusicPlayer. Selain itu, mahasiswa juga akan mengimplementasikan interaksi multimedia pada audio seperti memutar, menjeda, dan menghentikan audio baik untuk audio dari asset lokal maupun dari internet.

Langkah Praktikum 1 (Membuat Komponen BackButtonAppBarLeading, MusicCoverImage, TimeDisplay, dan ControllButton)

1. Buka proyek flutter media_player pada Visual Studio Code anda.

2. Di dalam folder lib>shared_components>app_bar, buat file baru bernama **back_button_app_bar_leading.dart**.

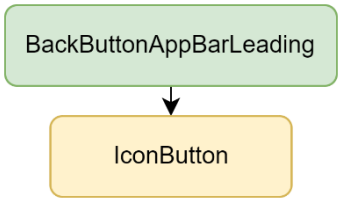


3. Berikut ini adalah gambar dari komponen yang akan anda buat

4. Buat sebuah stateless widget dengan nama **BackButtonAppBarLeading** di dalam file back_button_app_bar_leading.dart.

```dart
import 'package:flutter/material.dart';

class BackButtonAppBarLeading extends StatelessWidget {
  const BackButtonAppBarLeading({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

5. Ubah widget kembalian dari BackButtonAppBarLeading menjadi seperti berikut.



```
@override
Widget build(BuildContext context) {
  return IconButton();    The named parameter 'onPressed' is requir
}
```

The named parameter 'onPressed' is required, but there's no corresponding argument.
Try adding the required argument. dart(missing_required_argument)

The named parameter 'icon' is required, but there's no corresponding argument.
Try adding the required argument. dart(missing_required_argument)

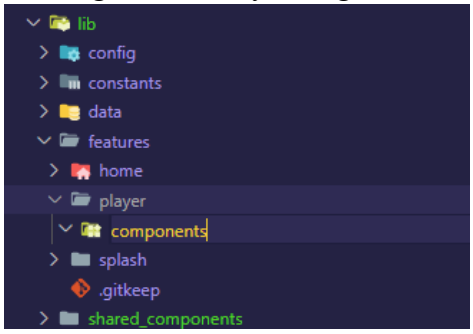
Use 'const' with the constructor to improve performance.

6. Tambahkan properti pada widget IconButton dalam BackButtonAppBarLeading tersebut sesuai dengan spesifikasi berikut.

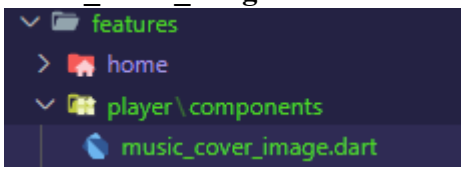
Properti	Nilai
key	const Key('back_btn')
icon	const Icon(Icons.arrow_back_ios_new_sharp, size: 18, color: MainColor.whiteFFFFFF,)
onPressed	() => Navigator.pop(context)
splashRadius	18

7. Setelah selesai membuat komponen BackButtonAppBarLeading. Anda akan membuat komponen **MusicCoverImage** seperti gambar di bawah ini.

8. Buat folder baru di dalam folder lib>features dengan nama **player**. Kemudian buat folder baru lagi di dalamnya dengan nama **components**.



9. Dalam folder components tersebut, buatlah file baru dengan nama **music_cover_image.dart**.



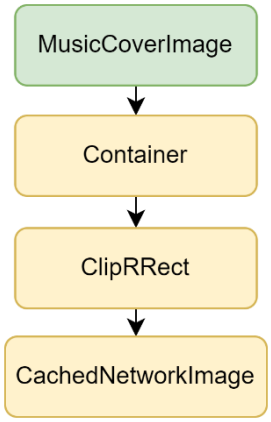
10. Buat sebuah stateless widget dengan nama **MusicCoverImage**.

```dart
import 'package:flutter/material.dart';

class MusicCoverImage extends StatelessWidget {
  const MusicCoverImage({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

11. Tambahkan properti sourceType dan cover dengan tipe String. Jangan lupa untuk menambahkannya juga ke dalam konstruktor.

```dart
const MusicCoverImage({
  super.key,
  required this.sourceType,
  required this.cover,
});

final String sourceType;
final String cover;
```

12. Ubah widget kembalian dari CustomAppBar menjadi seperti berikut.




```dart
@override
Widget build(BuildContext context) {
  return Container(
    child: ClipRRect(
      child: CachedNetworkImage(),
    ),
  );
}
```

13. Ubah properti dari setiap widget dalam MusicCoverImage tersebut sesuai dengan spesifikasi berikut.

Widget	Properti	Nilai
Container	width	double.infinity
	height	200
	decoration	BoxDecoration(borderRadius: BorderRadius.circular(20), image: DecorationImage(image: AssetImage(cover,), fit: BoxFit.cover,),)
ClipRRect	borderRadius	BorderRadius.circular(20)
CachedNetworkImage	imageUrl	cover
	progressIndicatorBuilder	(context, url, progress) => Center(child: CircularProgressIndicator(color: MainColor.purple5A579C, value: progress.progress,),)
	fit	BoxFit.cover

14. Nilai properti decoration dari Container berfungsi untuk menampilkan gambar cover musik yang bersumber dari asset lokal.

```
return Container(
  width: double.infinity,
  height: 200,
  decoration: BoxDecoration(
    borderRadius: BorderRadius.circular(20),
    image: DecorationImage(
      image: AssetImage(
        cover,
      ), // AssetImage
      fit: BoxFit.cover,
    ), // DecorationImage
  ), // BoxDecoration
```

15. Sedangkan widget CachedNetworkImage berfungsi untuk menampilkan gambar cover musik yang bersumber dari internet/network.

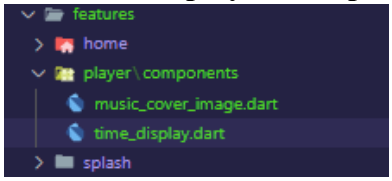
```dart
ClipRRect(
  borderRadius: BorderRadius.circular(20),
  child: CachedNetworkImage(
    imageUrl: cover,
    progressIndicatorBuilder: (context, url, progress) => Center(
      child: CircularProgressIndicator(
        color: MainColor.purple5A579C,
        value: progress.progress,
      ), // CircularProgressIndicator
    ), // Center
    fit: BoxFit.cover,
  ), // CachedNetworkImage
) // ClipRRect
```

16. Agar MusicCoverImage bisa mengatur widget mana yang akan ditampilkan berdasarkan sumber gambar dari cover musik yang diberikan (internet atau asset lokal), anda perlu menambahkan ternary operator. Sehingga properti decoration dan child dari widget Container akan terlihat seperti berikut.

```dart
@override
Widget build(BuildContext context) {
  return Container(
    width: double.infinity,
    height: 200,
    decoration: sourceType == "local"
        ? BoxDecoration( // BoxDecoration ···
        : null,
    child: sourceType != "local"
        ? ClipRRect( // ClipRRect ···
        : null,
  ); // Container
}
```

17. Selanjutnya anda akan membuat komponen TimeDisplay yang terlihat seperti gambar berikut.

18. Buat file baru dengan nama **time_display.dart** di dalam folder **lib>features>player>components**.



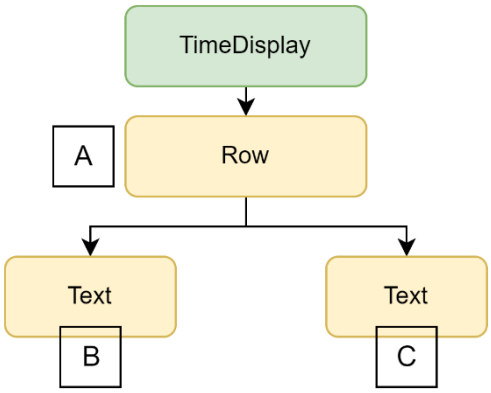
19. Buat sebuah stateless widget dengan nama **TimeDisplay** di dalam file time_display.dart.

```dart
import 'package:flutter/material.dart';

class TimeDisplay extends StatelessWidget {
  const TimeDisplay({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

20. Tambahkan properti Duration duration dan Duration position pada TimeDisplay. Jangan lupa untuk menambahkan properti tersebut ke dalam konstruktornya juga.

```
class TimeDisplay extends StatelessWidget {
  const TimeDisplay({
    super.key,
    required this.position,
    required this.duration,
  });

  final Duration position;
  final Duration duration;
```

21. Ubah widget kembalian dari TitleSection menjadi seperti berikut.



22. Tambahkan properti pada setiap widget dalam TimeDisplay tersebut sesuai dengan spesifikasi berikut.

[KODE] Widget	Properti	Nilai
[A] Row	mainAxisAlignment	MainAxisAlignment.spaceBetween
[B] Text	data	position.toString().split(".")[0]
	style	const TextStyle(color: MainColor.whiteF2F0EB,)
[C] Text	data	duration.toString().split(".")[0]
	style	const TextStyle(color: MainColor.whiteF2F0EB,)

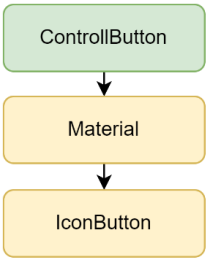
23. Komponen selanjutnya adalah ControllButton yang tampilannya seperti gambar di bawah ini.

24. Untuk membuatnya, buatlah file baru di dalam folder lib>features>player>components dengan nama **controll_button.dart**.

25. Buat sebuah stateless widget dengan nama ControllButton dan tambahkan juga properti-properti berikut ini. Jangan lupa untuk menambahkannya juga ke dalam konstruktor.

```
final IconData icon;
final Color bgColor;
final Function() onPressed;
final double splashR, icSize;
final Color? icColor;
```

26. Ubah widget kembalian dari ControllButton menjadi seperti berikut.

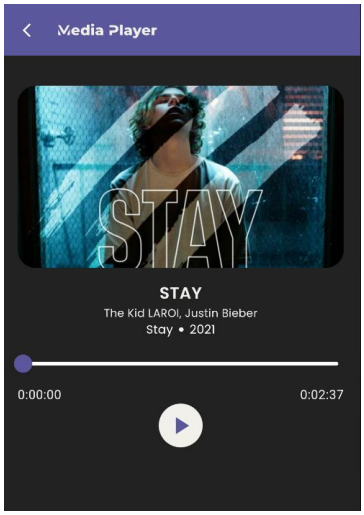


27. Tambahkan properti pada setiap widget dalam ControllButton tersebut sesuai dengan spesifikasi berikut.

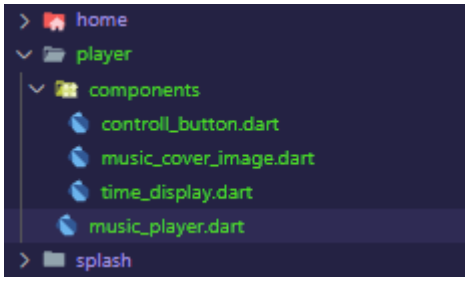
Widget	Properti	Nilai
Material	color	bgColor
	shape	const OvalBorder()
IconButton	splashRadius	splashR
	iconSize	icSize
	icon	Icon(icon, color: icColor,)
	onPressed	onPressed

Langkah Praktikum 2 (Membuat Halaman MusicPlayer)

1. Pada pratikum 2 ini, anda akan membuat halaman MusicPlayer yang tersusun dari komponen-komponen yang telah anda buat pada praktikum sebelumnya seperti CustomAppBar dengan BackButtonAppBarLeading, MusicCoverImage, TimeDisplay, dan ControllButton. Berikut adalah tampilan MusicPlayer yang akan anda buat.



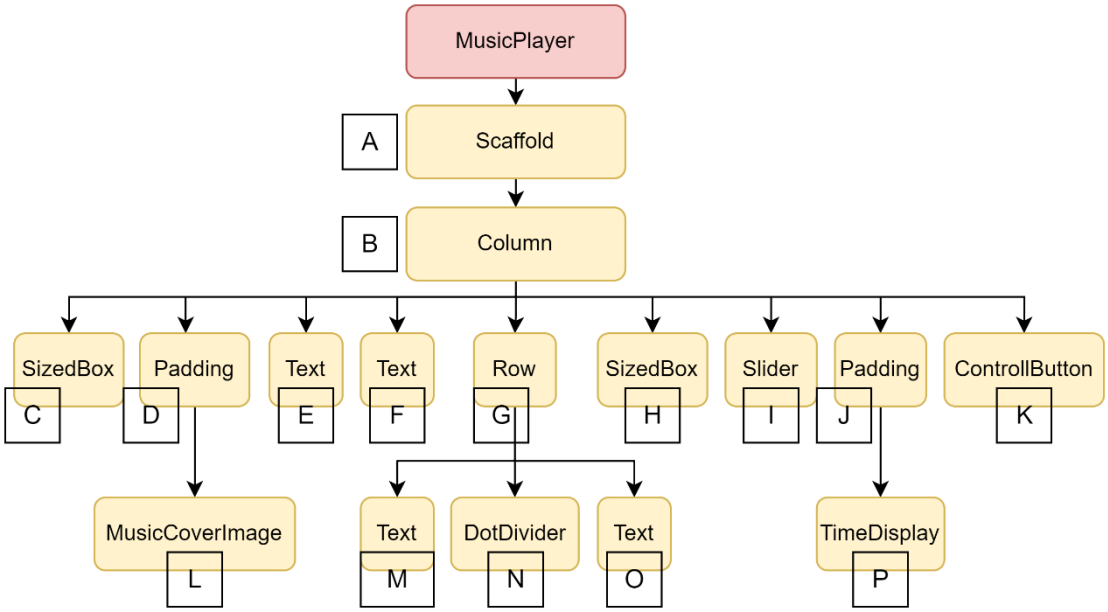
2. Buat file baru di dalam folder lib>features>player dengan nama **music_player.dart**.

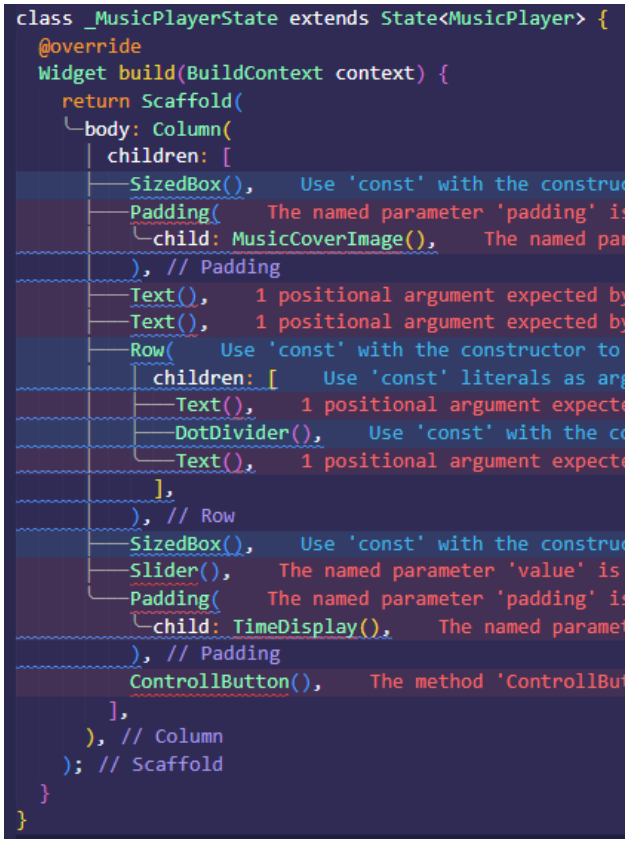


3. Pada file tersebut, buat sebuah stateful widget dengan nama **MusicPlayer**.

```dart
import 'package:flutter/material.dart';

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

  @override
  State<MusicPlayer> createState() => _MusicPlayerState();
}

class _MusicPlayerState extends State<MusicPlayer> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

4. Kemudian ubah widget kembalian dari MusicPlayer menjadi seperti berikut.





5. Untuk melengkapi kode sebelumnya, anda perlu menambahkan properti setiap widget yang ada pada halaman MusicPlayer agar sesuai dengan spesifikasi berikut.

[KODE] Widget	Properti	Nilai
[A] Scaffold	backgroundColor	MainColor.black222222
	appBar	const CustomAppBar(leading: BackButtonAppBarLeading())
[C] SizedBox	height	18
[D] Padding	padding	const EdgeInsets.symmetric(vertical: 15, horizontal: 18,)
[E] Text	data	'Music Title'
	style	MainTextStyle.poppinsW600.copyWith(fontSize: 18, color: MainColor.whiteF2F0EB,)
[F] Text	data	'Music Artist'
	style	MainTextStyle.poppinsW400.copyWith(fontSize: 12, color: MainColor.whiteF2F0EB,)
[G] Row	mainAxisAlignment	MainAxisAlignment.center
[H] SizedBox	height	4
[I] Slider	value	0.0

	min	0.0
	max	1.0
	thumbColor	MainColor.purple5A579C
	activeColor	MainColor.purple5A579C
	onChanged	(val) {}
[J] Padding	padding	const EdgeInsets.symmetric(horizontal: 18,)
[K] ControllButton	icon	Icons.pause
	bgColor	MainColor.whiteF2F0EB
	onPressed	() {}
	splashR	25
	icSize	32
[L] MusicCoverImage	sourceType	'local'
	cover	'assets/imgs/cover_one_direction_night_changes.jpeg'
[M] Text	data	'Album Name'
	style	MainTextStyle.poppinsW400.copyWith(fontSize: 13, color: MainColor.whiteFFFFFF,)
[O] Text	data	'2021'
	style	MainTextStyle.poppinsW400.copyWith(fontSize: 13, color: MainColor.whiteFFFFFF,)
[P] TimeDisplay	position	Duration()
	duration	Duration()

6. Untuk membuat tampilan MusicPlayer muncul ketika anda menekan komponen CoverMusicCard yang ada di halaman Home, anda perlu menambahkan kode berikut sebagai nilai pada properti onTap GestureDetector yang ada di CoverMusicCard.

```
() {
  Navigator.pushNamed(
    context,
    MainRoute.musicPlayer,
    arguments: music,
  );
}
```

7. Selain itu, agar halaman MusicPlayer yang anda buat dapat ditampilkan melalui route MainRoute.musicPlayer, buka file **main_pages.dart** dan ubah widget kembalian dari widget function MainRoute.musicPlayer dari yang sebelumnya `Placeholder()` menjadi `MusicPlayer()`.

8. Sekarang coba jalankan aplikasi anda. Seharusnya aplikasi akan terlihat seperti ini.

9. Karena data yang ditampilkan halaman MusicPlayer masih statis (CoverMusicCard manapun yang ditekan, halaman MusicPlayer menampilkan data yang sama), anda perlu menambahkan state dan fungsi tambahan untuk membuat data menjadi dinamis dengan mengambil objek Music yang dikirimkan argumen oleh Navigator. Untuk itu, tambahkan state `late Music music` dan fungsi `didChangeDependencies()` berikut ini pada kelas MusicPlayerState.

```
class _MusicPlayerState extends State<MusicPlayer> {
  late Music music;

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    music = ModalRoute.of(context)!.settings.arguments as Music;
  }
```

10. Jalankan kembali proyek media_player anda, maka sekarang data musik pada halaman MusicPlayer akan dinamis mengikuti data yang dikirimkan. Lihat di sini.

Langkah Praktikum 3 (Menambahkan Interaksi pada Halaman MusicPlayer)

1. Sekarang anda akan belajar cara menambahkan interaksi multimedia pada audio seperti memutar, menjeda, dan menghentikan musik. Untuk melakukannya, anda perlu membuat sebuah objek AudioPlayer dari package *audioplayers* di dalam kelas MusicPlayerState.

```
  AudioPlayer newPlayer = AudioPlayer();
```

2. Selain itu, tambahkan juga state bool play, Duration duration, Duration position, dan Source source. Semua state tersebut akan digunkan untuk menyimpan nilai dari audio yang akan dimainkan dan perubahannya

```
  bool play = false;
Duration duration = const Duration();
Duration position = const Duration();
Source? source;
```

3. Setelah itu, tambahkan fungsi `initPlayerState()` untuk menginisiasi object newPlayer dan state-state lainnya berdasarkan objek musik yang diterima dari Navigator.

```
initPlayerState() {
  setState(() {
    source = music.sourceType == "local"
        ? AssetSource(
            music.source!.replaceFirst("assets/", ""),
          )
        : UrlSource(music.source!);
  });
  newPlayer.setSource(source!).then((value) async {
    final dur = await newPlayer.getDuration() ?? const Duration();
    setState(() {
      duration = dur;
    });
  });
}
```

4. Agar fungsi tersebut dijalankan tepat saat objek music berhasil diterima, tambahkan pemanggilan `initPlayerState()`, pada body fungsi didChangeDependencies().

```
class _MusicPlayerState extends State<MusicPlayer> {
  late Music music;

  AudioPlayer newPlayer = AudioPlayer();

  bool play = false;
  Duration duration = const Duration();
  Duration position = const Duration();
  Source? source;

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    music = ModalRoute.of(context)!.settings.arguments as Music;
    initPlayerState();
  }

  initPlayerState() {
    setState(() {
      source = music.sourceType == "local"
          ? AssetSource(
              music.source!.replaceFirst("assets/", ""),
            )
          : UrlSource(music.source!);
    });
    newPlayer.setSource(source!).then((value) async {
      final dur = await newPlayer.getDuration() ?? const Duration();
      setState(() {
        duration = dur;
      });
    });
  }
```

5. Setelah menambahkan kode untuk menginisiasi AudioPlayer, anda juga perlu menambahkan fungsi dengan `initPlayer()`. Fungsi tersebut akan digunakan untuk menginisiasi dan mengawasi setiap perubahan dari nilai duration dan position (yang mana kedua state tersebut akan dijadikan argumen pada widget Slider). Sehingga komponen Slider dan TimeDisplay akan selalu update dengan posisi dari musik yang sedang dimainkan.

```
  void initPlayer() async {
    newPlayer.onPositionChanged.listen((pos) {
      setState(() {
        position = pos;
      });
      if (pos.inSeconds.toDouble() == duration.inSeconds.toDouble()) {
        setState(() {
          play = false;
        });
      }
    });
  }
```

6. Kemudian lakukan pemanggilan fungsi `initPlayer()` pada fungsi override `initState()`.

```
class _MusicPlayerState extends State<MusicPlayer> {
  late Music music;

  AudioPlayer newPlayer = AudioPlayer();

  bool play = false;
  Duration duration = const Duration();
  Duration position = const Duration();
  Source? source;

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

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    music = ModalRoute.of(context)!.settings.arguments as Music;
    initPlayerState();
  }

  initPlayerState() {
    setState(() {
      source = music.sourceType == "local"
          ? AssetSource(
              music.source!.replaceFirst("assets/", ""),
            )
          : UrlSource(music.source!);
    });
    newPlayer.setSource(source!).then((value) async {
      final dur = await newPlayer.getDuration() ?? const Duration();
      setState(() {
        duration = dur;
      });
    });
  }

  void initPlayer() async {
    newPlayer.onPositionChanged.listen((pos) {
      setState(() {
        position = pos;
      });
      if (pos.inSeconds.toDouble() == duration.inSeconds.toDouble()) {
        setState(() {
          play = false;
        });
      }
    });
  }
```

7. Tambahkan lagi fungsi playAudio() di bawah ini untuk mengimplementasikan interaksi memutar audio.

```
Future<void> playAudio() async {
  newPlayer.setVolume(1.0);
  newPlayer.play(
    source!,
    mode: PlayerMode.mediaPlayer,
  );
  setState(() {
    play = true;
  });
}
```

8. Untuk menambahkan implementasi interaksi pause dan seek to second, masukkan kedua fungsi di bawah ini ke dalam kelas MusicPlayerState.

```
Future<void> pauseAudio() async {
  newPlayer.pause();
  setState(() {
    play = false;
  });
}

void seekToSecond(double value) {
  Duration newDuration = Duration(seconds: value.toInt());

  newPlayer.seek(newDuration);
}
```

9. Tambahkan juga fungsi playPause() yang akan dijadikan callback onPressed widget ControllButton. Sehingga ControllButton dapat melakukan aksi play ataupun pause audio berdasarkan state bool play.

```
Future<void> playPause() async {
  if (play) {
    await pauseAudio();
  } else {
    await playAudio();
  }
}
```

10. Sedangkan untuk melakukan interaksi menghentikan audio, tambahkan pemanggilan `newPlayer.stop()` pada fungsi override dispose. Sehingga ketika user menekan tombol kembali (men-trigger penghapusan halaman MusicPlayer dari Stack Navigation), baik dari App Bar maupun dari tombol fisik, musik akan otomatis dihentikan.

```
@override
void dispose() {
  super.dispose();
  newPlayer.stop();
}
```

11. Kini anda telah selesai menyiapkan semua fungsi agar halaman MusicPlayer dapat melakukan interaksi-interaksi audio. Sekarang ubah properti dari widget-widget di bawah ini sesuai dengan spesifikasinya.

[KODE] Widget	Properti	Nilai
[I] Slider	position	position.inSeconds.toDouble()
	max	duration.inSeconds.toDouble()
	onChanged	seekToSecond
[P] TimeDisplay	position	position
	duration	duration
[K] ControllButton	icon	play ? Icons.pause : Icons.play_arrow
	onPressed	playPause

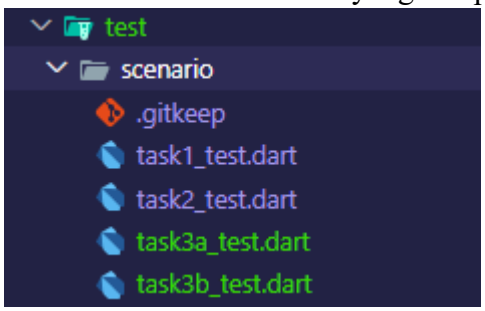
12. Coba jalankan proyek anda pada emulator dengan menekan tombol run yang ada pada fungsi main di **main.dart**. Berikut adalah tampilan emulator jika anda telah berhasil menyusun tampilan MusicPlayer dan mengimplementasikan interaksi audio. Lihat tampilan keseluruhan aplikasinya di sini.

Langkah Verifikasi Kode

1. Untuk keperluan testing, ubah kelas MusicPlayerState menjadi public dengan menghapus _ yang ada pada nama kelasnya. Selain itu tambahkan anotasi `@visibleForTesting` tepat di atas kelas MusicPlayerState. Sehingga kelas MusicPlayerState akan menjadi seperti berikut.

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

  @override
  State<MusicPlayer> createState() => MusicPlayerState();
}

@visibleForTesting
class MusicPlayerState extends State<MusicPlayer> {
  late Music music;
```

2. Unduh test file pada tautan Test File Tugas 3A dan Test File Tugas 3B. Letakkan file ke dalam folder test>scenario yang ada pada proyek.



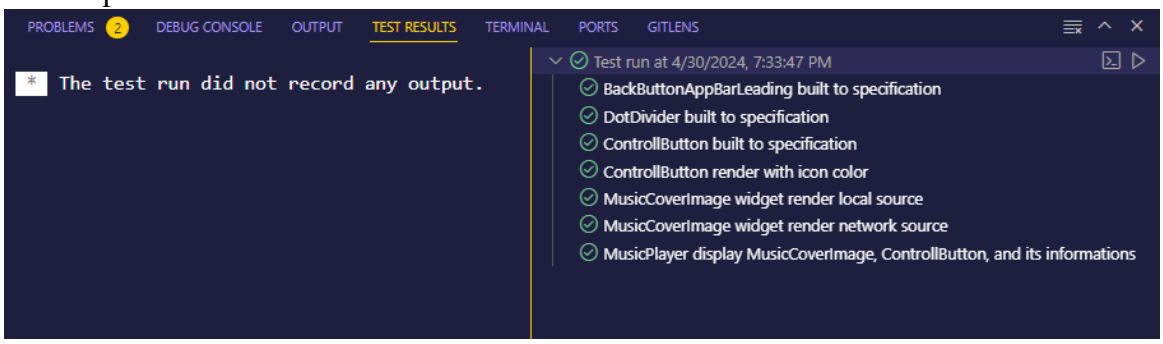
3. Buka file **task3a_test.dart**.

```dart
import 'package:cached_network_image/cached_network_image.dart';···

final localMusic = Music(···

final networkMusic = Music(···

Music? capturedMusic;
bool back = false;

class MockNavigatorObserver extends Mock···

final routes = <String, WidgetBuilder>{···

Run | Debug
void main() {
  void timeDisplayCheck(Slider slider, Duration duration) {···

  Run | Debug
  testWidgets('BackButtonAppBarLeading built to specification',···

  Run | Debug
  testWidgets('DotDivider built to specification', (WidgetTester tester) async {···
```

4. Tekan tombol Run yang ada pada fungsi main dan tunggu proses hingga selesai.

5. Jika anda mengalami error, coba periksa pesan yang muncul dan perbaiki kesalahannya. Kesalahan yang sering terjadi biasanya karena adanya perbedaan nama key atau properti yang hilang (tidak sesuai dengan spesifikasi) dari widget tertentu.

6. Jika testing berhasil (berhasil membangun halaman MusicPlayer), maka akan muncul hasil berikut pada tab Test Result Visual Studio Code anda.



7. Lakukan hal yang sama pada file **task3b_test.dart**. Berikut ini adalah hasil pada tab Test Result Visual Studio Code anda jika anda berhasil mengimplementasikan interaksi audio pada MusicPlayer.