BAB IV
APPLICATION PROGRAMMING INTERFACE

Application Programming Interface (API) adalah cara agar sebuah perangkat lunak atau lebih dapat berkomunikasi dengan server yang berisikan data. Data pada server diambil dan digunakan oleh pengembang aplikasi untuk diolah, API menjadi perantara bagi para *developer* untuk mengambil data pada server. Jika dianalogikan pada kegiatan di restoran, maka seperti pelanggan yang melihat daftar menu makanan dan memesan makanan kepada restoran tersebut lalu restoran memberikannya, pelanggan tidak tahu bagaimana caranya sebuah makanan disiapkan, begitu juga *developer* tidak tahu bagaimana data tersebut disiapkan tapi yang penting adalah data didapat oleh *developer*.

4.1 Tujuan Praktikum

Tujuan	Penjelasan
Mengetahui dan memahami konsep dasar dari *application programming interface*	Pada bab ini, praktikan akan diharapkan dapat mengetahui dan memahami konsep dasar *application programming interface* (API)
Mampu memanggil data pada API ke dalam aplikasi	Pada bab ini, praktikan akan diajarkan bagaimana caranya mengambil data melalui API dan menampilkannya pada aplikasi.

4.2 Persyaratan Praktikum

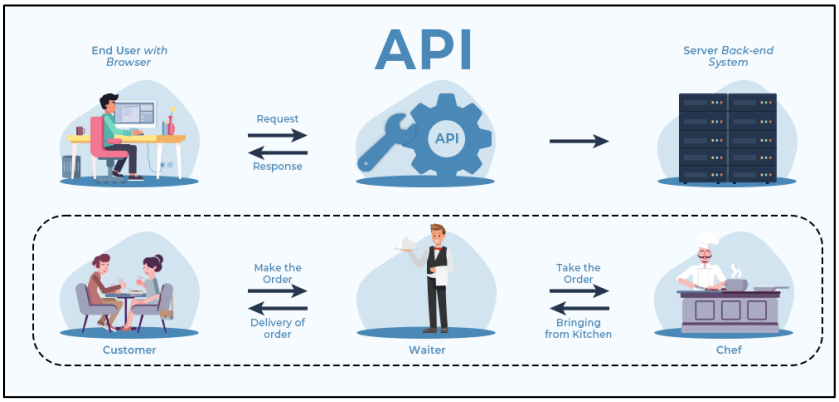
Disarankan praktikan menggunakan *hardware* dan *software* sesuai pada dokumentasi ini. Apabila terdapat versi yang lumayan lampau dari versi yang direkomendasikan atau *hardware* yang lawas maka sebaiknya bertanya kepada Asisten Mengajar Shift.

***HARDWARE* YANG DIBUTUHKAN PRAKTIKUM**	**JENIS**
PC / Laptop CPU	**≥ 4 Cores**
PC / Laptop RAM	**≥ 8 GB**
PC / Laptop *Storage*	**≥ 10 GB**

***SOFTWARE* YANG DIBUTUHKAN PRAKTIKUM**
Android *Studio* / *Visual Studio Code*

4.3 Materi Praktikum

Application Programming Interface (API) merupakan sebuah mekanisme yang memungkinkan dua komponen perangkat lunak atau lebih untuk saling berkomunikasi dengan menggunakan protokol yang ada. Dalam pengembangan aplikasi, API berperan sebagai jembatan yang menghubungkan *front end* dan *back end* dalam proses pengembangan aplikasi. *Front end* akan mengirimkan permintaan data kepada *back end* melalui API, kemudian back end akan merespon permintaan tersebut dan mengirimkan data yang diminta oleh *front end* untuk ditampilkan ke layar aplikasi.



Gambar 4.1 Ilustrasi *Application Programming Interface*

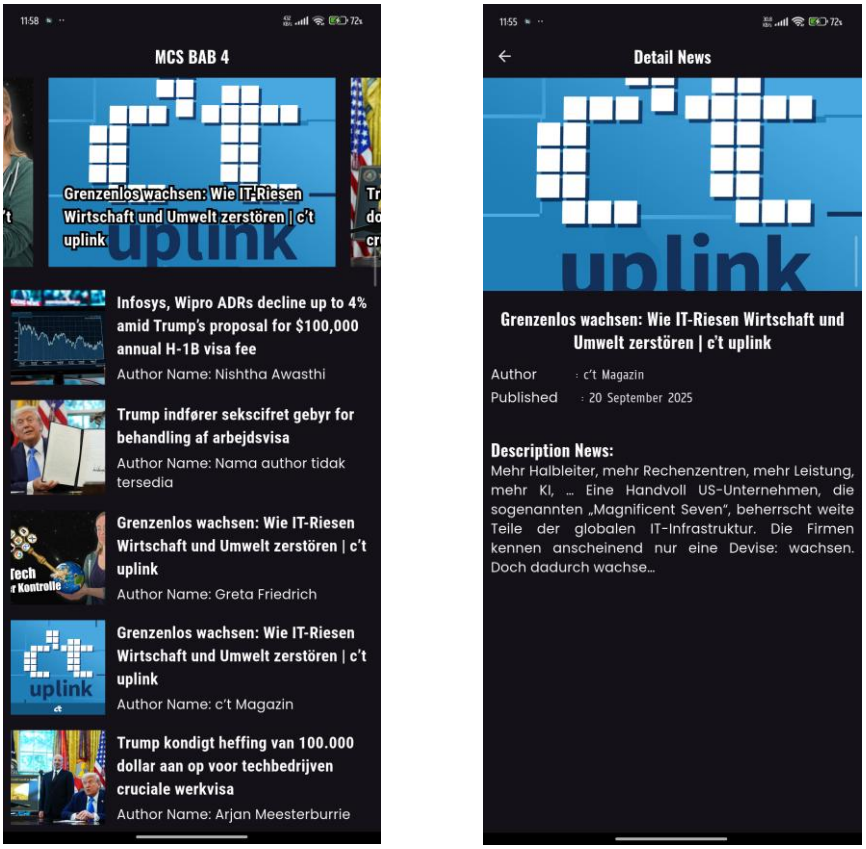
Jika dianalogikan dalam sebuah sistem restoran, **API** diibaratkan sebagai *waiter* / pelayan, ***front end*** diibaratkan sebagai *customer* / pelanggan, dan ***back end*** diibaratkan sebagai chef. Pelanggan (*front end*) akan membaca menu yang telah

disediakan oleh restoran tersebut, kemudian pelanggan akan memesan menu yang diinginkan. Setelah menerima menu yang diinginkan oleh pelanggan, *waiters* / pelayan (API) akan memberitahu menu yang diminta oleh pelanggan kepada chef (*back end*), dimana setelah menerima menu tersebut chef akan mulai memasak (server akan mencari data yang diminta). Setelah makanan jadi (data yang diminta telah didapatkan) chef akan mengirimkannya kembali kepada *waiters* / pelayan untuk dikirimkan kembali ke pelanggan

4.4 Prosedur Praktikum

4.4.1 Tampilan Aplikasi

Berikut merupakan tampilan dari aplikasi yang akan dibentuk pada praktikum bab 4.



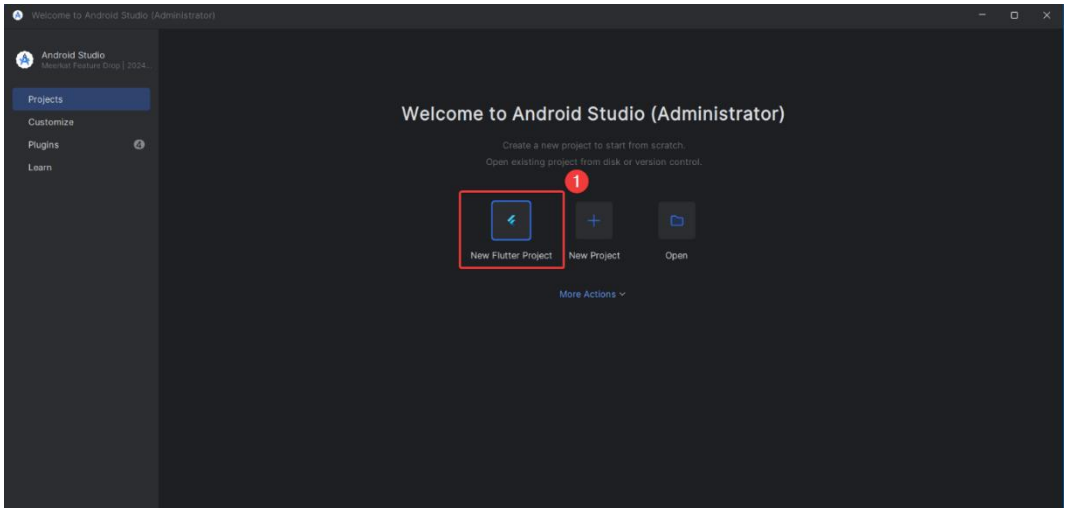
Gambar 4.2 Tampilan Halaman Aplikasi yang Akan diimplementasikan

Penjelasan terkait bagaimana cara aplikasi bekerja akan diterangkan oleh asisten yang mengajar.

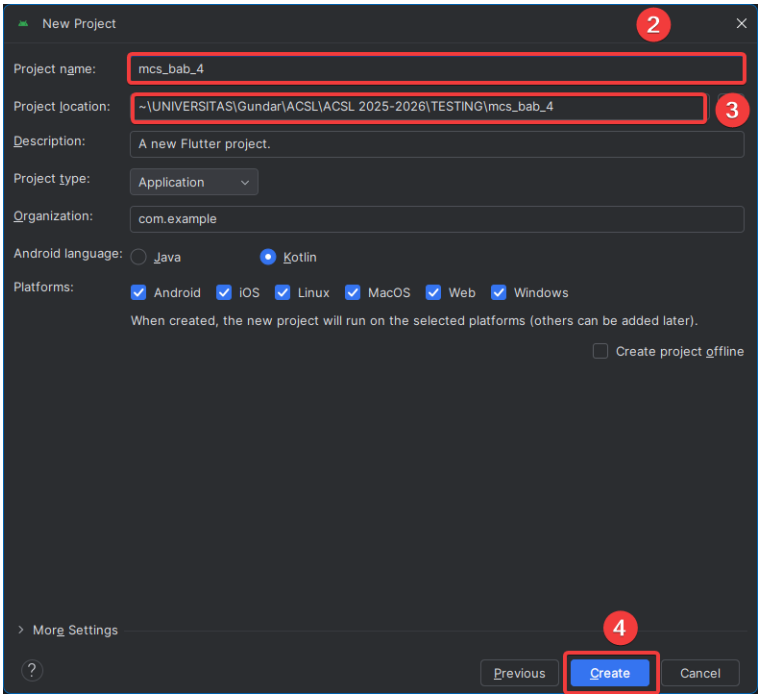
4.4.2 Implementasi Aplikasi

Dalam mengimplementasikan tampilan dari desain aplikasi di atas, terdapat beberapa langkah yang harus dilewati terlebih dahulu agar proses praktikum dapat berjalan dengan lancar dan terselesaikan sesuai dengan apa yang dituju. Sebelum membuat *project* flutter yang baru, praktikan diharapkan untuk mengakses halaman website **https://newsapi.org/** dan mendaftarkan akun gmail masing-masing untuk mendapatkan API *key* yang nantinya akan digunakan dalam proses praktikum.

Setelah akun berhasil didaftarkan dan telah mendapatkan API *key* dari NewsAPI, barulah praktikan dapat membuat *project* flutter baru pada *software* android *studio*.

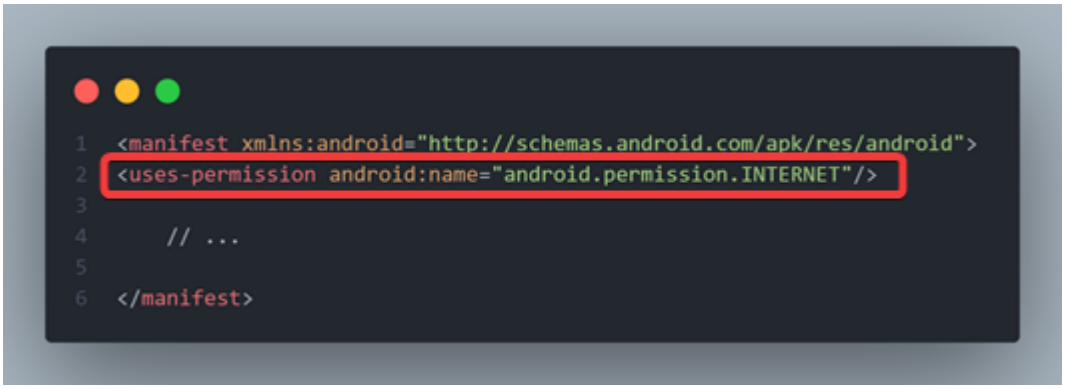


Gambar 4.3 Tampilan Awal *Software* Android *Studio*



Gambar 4.4 Proses Pembuatan *Project* Baru

Setelah file *project* berhasil terbentuk, pastikan kembali bahwa tampilan yang diberikan oleh android studio sudah berada pada menu ***project***. Jika sudah, bukalah file **AndroidManifest.xml** dan tambahkan izin berikut pada file tersebut.



Gambar 4.5 Konfigurasi Pengaksesan Internet

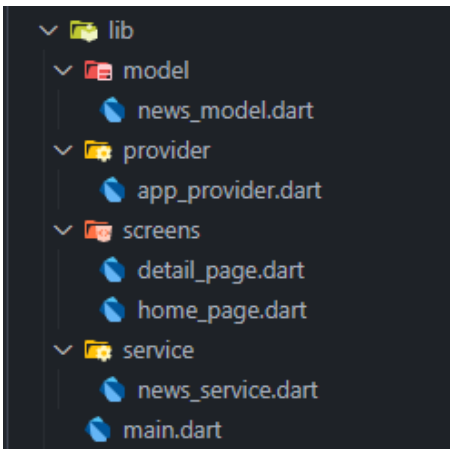
File tersebut dapat diakses melalui path **android > app > src > main > AndroidManifest.xml.** Kemudian, masuklah ke dalam file **pubspec.yaml** dan tambahkan *package* berikut ke dalam bagian **depedencies.**

```yaml
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  http: ^1.4.0
  provider: ^6.1.5
  carousel_slider: ^5.1.1
  google_fonts: ^6.2.1
  intl: ^0.20.2
```

Gambar 4.6 *Package* yang digunakan

Dalam praktikum kali ini, kita menggunakan 5 *package* tambahan, antara lain **http, provider, carousel_slider, google_fonts**, dan **intl**. Kemudian tambahkanlah beberapa file dan folder ke dalam folder **lib,** sehingga struktur *tree project* menjadi seperti pada gambar berikut:



Gambar 4.7 Struktur *Tree Project*

Masuklah ke dalam file **news_model.dart** dan masukkanlah kode program berikut:

```
class NewsSource {
  dynamic id;
  dynamic name;
  NewsSource({required this.id, required this.name});
```

```
  factory NewsSource.fromJson(Map<String, dynamic> json) {
    return NewsSource(
      id: json['id'] ?? "News doesn't have id",
      name: json['name'] ?? "News doesn't have source name",
    );
  }
}

class NewsModel {
  NewsSource source;
  String author;
  String title;
  String description;
  String url;
  String urlToImage;
  DateTime publishedAt;
  String content;

  NewsModel({
    required this.source,
    required this.author,
    required this.title,
    required this.description,
    required this.url,
    required this.urlToImage,
    required this.publishedAt,
    required this.content,
  });

  factory NewsModel.fromJson(Map<String, dynamic> json) {
    return NewsModel(
      source: NewsSource.fromJson(json["source"]),
      author: json["author"] ?? "Nama author tidak tersedia",
      title: json["title"] ?? "Judul artikel tidak tersedia",
      description:  json["description"]  ??  "Deskripsi  artikel
tidak tersedia",
      url: json["url"] ?? "-",
```

```
      urlToImage:
          json["urlToImage"] ?? "https://demofree.sirv.com/nope-
not-here.jpg",
      publishedAt: DateTime.parse(json["publishedAt"] ?? "-"),
      content: json["content"] ?? "Artikel tidak memiliki
konten",
    );
  }
}
```

File tersebut berisikan beberapa baris program yang digunakan dalam membuat model yang dapat merepresentasikan data dalam bentuk *dart* objek. Hal ini diperlukan agar proses *consume* data API menjadi lebih mudah dan lebih terstruktur dalam pengembangan aplikasi. Terdapat 2 *class* yang dibangun berdasarkan data yang diberikan oleh News API, yakni **NewsSource{}** yang di dalamnya berisikan id dan name serta **NewsModel{}** yang berisikan lebih banyak *attribute,* seperti *source, author, title, description, url, urlToImage, publishedAt,* dan *content*. Model dari NewsSource{} digunakan untuk merepresentasikan data yang tersimpan pada *array* **source**. Pada *class* tersebut terdapat *constructor* yang bersifat **required** yang mengembalikan 2 nilai yang telah didefinisikan sebelumnya. Kemudian terdapat pemanggilan terhadap **factory NewsSource.fromJson()** yang akan melakukan *mapping* nilai terhadap struktur JSON dan disimpan ke dalam bentuk objek dart yang telah dibuat sebelumnya.

Model yang dibentuk pada NewsModel{} merupakan model yang merepresentasikan data API secara umum. NewsModel{} memiliki *constructor* yang lebih banyak dibandingkan dengan NewsSource{}, karena data yang tersimpan di dalam *array* induk pada NewsAPI jauh lebih banyak dibandingkan dengan NewsSource{} yang hanya menyimpan data untuk id dan name. Model ini juga mengembalikan nilai dari setiap *key* API ke dalam bentuk dart objek dengan cara *mapping* terhadap data JSON tersebut dengan menggunakan perintah **factoryNewsModel.fromJson().**

Berikutnya masuklah ke dalam file **news_service.dart** dan masukkan kode program berikut di dalamnya:

```
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'package:mcs_bab_4/model/news_model.dart';

class NewsService {
  final universalUrl =
      "https://newsapi.org/v2/everything?q=tesla&from=2025-08-
20&sortBy=publishedAt&apiKey={API_KEY}";  //  MASUKKAN  API  KEY
MASING-MASING

  Future<List<NewsModel>> getAllNewsData() async {
    final response = await http.get(Uri.parse(universalUrl));
    try {
      if (response.statusCode == 200) {
        final responseBody = jsonDecode(response.body);
        final List articleResponse = responseBody['articles'];
        final newsResult = articleResponse
            .map((e) => NewsModel.fromJson(e))
            .toList();
        return newsResult;
      } else {
        throw Exception("[Try Error]: Failed to get data");
      }
    } catch (e) {
      throw Exception("[Catch Error]: Failed to get data");
    }
  }
}
```

Class **NewsService{}** berisikan logika program yang diperlukan untuk mengambil data dari API. Dalam *class* tersebut, terdapat variabel **universalUrl** yang mengembalikan sebuah url yang di dalamnya menyediakan sekumpulan data berita yang sudah dapat langsung dikonsumsi. Selain itu, *class* tersebut juga memiliki metode **getAllDataNews()** yang berisikan kode program untuk

menangani proses request data ke API. Dalam proses tersebut, sistem akan meminta request ke url yang telah ditetapkan dengan menggunakan metode **http.get()** dan disimpan dalam variabel response. Kemudian sistem akan mengecek statusCode yang dihasilkan pada saat proses request berlangsung. Jika statusCode menunjukkan angka 200, maka sistem akan mengambil bagian body dari data API tersebut dan disimpan ke dalam variabel responseBody. Kemudian bagian body yang diambil adalah bagian *key* **articles** yang disimpan dalam variabel articlesResponse dalam bentuk list. Kemudian body articles yang telah diambil dikonversi ke dalam bentuk dart objek dengan pemanggilan terhadap metode **NewsModel.fromJson().**

Selanjutnya dibentuklah *class* provider bernama **AppProvider** yang meng*extends* ChangeNotifier. Berikut kode yang digunakan pada *class* AppProvider:

```
class AppProvider extends ChangeNotifier {
  NewsSource? source;
  late String authorName;
  late String titleOfArticle;
  late String descOfArticle;
  late String urlOfArticle;
  late String imageOfArticle;
  late DateTime publishDateTime;
  late String contentOfArticle;
  int? index;

  final titleFontStyle = GoogleFonts.oswald(
    fontSize: 17,
    color: Colors.white,
    fontWeight: FontWeight.bold,
  );

  final subTitleFontStyle = GoogleFonts.robotoCondensed(
    fontSize: 17,
    color: Colors.white,
    fontWeight: FontWeight.bold,
```

```
  );

  final                subTitleFontWIthStrokeStyle                =
GoogleFonts.robotoCondensed(
    fontSize: 17,
    color: Colors.white,
    textStyle: TextStyle(
      fontWeight: FontWeight.bold,
      foreground: Paint()
        ..style = PaintingStyle.stroke
        ..strokeWidth = 5
        ..color = Colors.black,
    ),
  );

  final universalFontStyle = GoogleFonts.poppins(
    fontSize: 14,
    color: Colors.white,
  );

  goToDetailNews({
    required BuildContext context,
    required NewsModel newsModel,
    required int index,
    required navigationPage,
  }) async {
    this.index = index;
    source = newsModel.source;
    authorName = newsModel.author;
    titleOfArticle = newsModel.title;
    descOfArticle = newsModel.description;
    urlOfArticle = newsModel.url;
    imageOfArticle = newsModel.urlToImage;
    publishDateTime = newsModel.publishedAt;
    contentOfArticle = newsModel.content;
    Navigator.push(
      context,
```

```
      MaterialPageRoute(builder: (context) => navigationPage),
    );
  }
}
```

Dalam *class* provider tersebut, didefinisikan beberapa variabel dan juga method yang nantinya akan digunakan pada pembangunan aplikasi tersebut.

1. Variabel **source, authorName, titleOfArticle, descOfArticle, urlOfArticle, imageOfArticle, publishDateTime,** dan **contentOfArticle** merupakan variabel yang akan menyimpan *value* dari masing-masing *key* yang telah diambil pada API.
2. Variabel **index** merupakan variabel yang akan menampung index / urutan dari setiap data article yang ada.
3. Variabel **titleFontStyle, subTitleFontStyle, subTitleFontWithStrokeStyle** dan **universalFontStyle** merupakan variabel bertipe TextStyle yang akan memberikan *styling* terhadap widget Text dengan menggunakan jenis font yang disediakan oleh *package* google_fonts.
4. *Method* **goToDetailNews()** bersifat *asynchronus* yang berisikan 4 *constructor* bersifat *required*, yakni **context, newsModel, index** dan **navigationPage**. *Method* ini akan mengambil dan menyimpan seluruh data yang ada pada halaman sebelumnya ke dalam masing-masing objek dan akan menavigasikan aplikasi ke halaman berikutnya.

Berikutnya masuklah ke dalam file **main.dart** terlebih dahulu untuk memberikan konfigurasi dasar pada aplikasi yang akan dibuat.

```
void main() {
  runApp(MyApp());
}

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

```
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [ChangeNotifierProvider(create: (context) =>
AppProvider())],
      child: MaterialApp(
        title: "MCS BAB 4",
        debugShowCheckedModeBanner: false,
        theme: ThemeData.dark(),
        home: HomePage(),
      ),
    );
  }
}
```

Struktur kode yang digunakan pada file tersebut sama seperti yang digunakan pada praktikum pertemuan ke-3 yang mengembalikkan widget MultiProvider terlebih dahulu sebelum nantinya mendefinisikan konfigurasi umum untuk aplikasi.

Setelah melakukan konfigurasi umum untuk aplikasi, selanjutnya kita akan melakukan penulisan kode untuk halaman *home* (halaman awal aplikasi). Halaman ini dibentuk dengan meng*extends* StatefulWidget dan mengembalikan widget **Consumer().** Widget ini memiliki parameter appProvider yang berfungsi sebagai variabel yang akan meng*instance class* AppProvider yang telah dibuat dan mempermudah dalam penggunaan variabel atau *method* yang telah dibentuk pada *class* AppProvider.

Widget ini akan mengembalikan 3 nilai berbeda, bergantung pada kondisi yang sedang dialami. Kondisi pertama, jika data sedang dalam proses pengambilan, maka aplikasi akan mengembalikan widget **CircularProgressIndicator()** di bagian tengah halaman aplikasi. Kondisi kedua, jika terjadi *error* pada saat pengambilan data, maka aplikasi akan menampilkan pesan *error* tersebut dalam bentuk text. Kondisi ketiga akan muncul jika proses pengambilan data telah selesai dan tidak

didapati *error*. Tampilan yang akan ditampilkan oleh aplikasi dibagi menjadi ke dalam 3 bagian, yakni **appbar**, ***highlight* berita**, dan **list berita.**

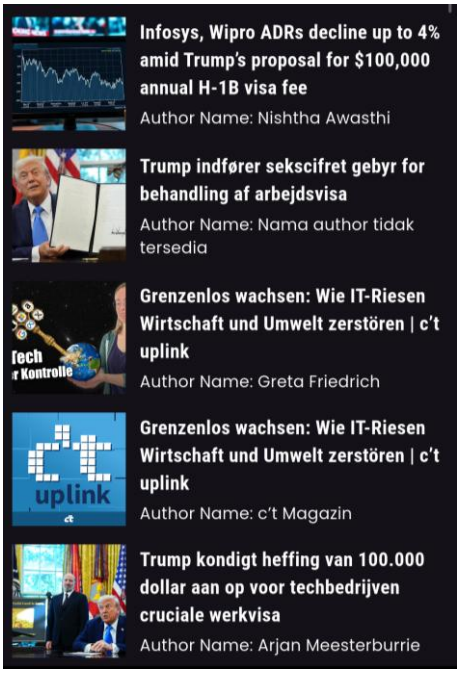
Gambar 4.8 Area *Highlight* Berita

```
CarouselSlider.builder(
  itemCount: 5,
  options: CarouselOptions(
    autoPlay: true,
    height: MediaQuery.of(context).size.width / 2,
  ),
  itemBuilder: (context, index, pageIndex) {
    final getSingleArticle = getAllArticle![index];
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 10),
      child: GestureDetector(
        child: Stack(
          children: [
            Image.network(
              width: double.infinity,
              height: double.infinity,
              getSingleArticle.urlToImage,
              fit: BoxFit.cover,
              errorBuilder: (context, error, stackTrace) {
                return Image.network(
                  width: double.infinity,
                  height: double.infinity,
                  "https://demofree.sirv.com/nope-not-here.jpg",
                  fit: BoxFit.cover,
                );
              },
            ),
```

```
            Align(
              alignment: Alignment.bottomLeft,
              child: Padding(
                padding: const EdgeInsets.all(20),
                child: Stack(
                  children: [
                    Text(
                      getSingleArticle.title,
                      style: appProvider
                          .subTitleFontWIthStrokeStyle,
                    ),
                    Text(
                      getSingleArticle.title,
                      style: appProvider.subTitleFontStyle,
                    ),
                  ],),
              ),),
          ],
        ),
        onTap: () {
          appProvider.goToDetailNews(
            context: context,
            newsModel: getSingleArticle,
            index: index,
            navigationPage: DetailPage(),
          );
        },),
    );
  },
),
```

Highlight berita dibentuk dengan menggunakan widget **CarouselSlider.builder().** Widget ini memiliki fungsi yang sama seperti widget ListView.builder(), yakni dapat menampilkan item dengan jumlah yang dinamis. Namun, dengan penggunaan CarouselSlider.builder() posisi dari item yang dibungkus akan secara otomatis berada di sumbu *horizontal*. Selain itu, widget ini

juga memiliki properti **autoPlay:** yang jika diberikan nilai *true*, maka item yang ada di dalamnya akan bergerak secara otomatis. Bagian ini mengembalikan gambar dan judul berita yang ditimpa di atas gambar tersebut dan dibungkus dengan menggunakan widget GestureDetector(), sehingga berita tersebut dapat ditekan dan menuju ke halaman yang berisikan detail berita (DetailNewsPage()).



Gambar 4.9 Area Daftar Berita

```
        ListView.builder(
          shrinkWrap: true,
          physics: NeverScrollableScrollPhysics(),
          itemCount: getAllArticle!.length,
          itemBuilder: (context, index) {
            final getSingleArticle = getAllArticle[index];
            return Padding(
              padding: const EdgeInsets.all(10),
              child: GestureDetector(
                child: Row(
                  children: [
                    Image.network(
                      width:  MediaQuery.of(context).size.width  /
4,
```

```
                        height: MediaQuery.of(context).size.width /
4,
                        getSingleArticle.urlToImage,
                        fit: BoxFit.cover,
                        errorBuilder: (context, error, stackTrace)
{
                          return Image.network(
                            width:
                                MediaQuery.of(context).size.width /
4,
                            height:
                                MediaQuery.of(context).size.width /
4,
                            "https://demofree.sirv.com/nope-not-
here.jpg",
                            fit: BoxFit.cover,
                          );
                        },
                      ),
                      SizedBox(width: 15),
                      Expanded(
                        child: Column(
                          crossAxisAlignment:
                              CrossAxisAlignment.start,
                          children: [
                            Text(
                              getSingleArticle.title,
                              style: appProvider.subTitleFontStyle,
                            ),
                            SizedBox(height: 5),
                            Text(
                              "Author                              Name:
${getSingleArticle.author}",
                              style:
appProvider.universalFontStyle,
                            ),
                          ],
```

```
                    ),
                  ),
                ],
              ),
              onTap: () {
                appProvider.goToDetailNews(
                  context: context,
                  newsModel: getSingleArticle,
                  index: index,
                  navigationPage: DetailPage(),
                );
              },),
            );
          },
        ),
```

Bagian list berita dibentuk dengan menggunakan widget **ListView.builder()** yang di dalamnya memiliki properti **shrinkWrap: true** dan **physics: NeverScrollableScrollPhysics()**. Logika program yang digunakan dalam bagian ini sama seperti yang digunakan pada bagian *highlight* berita, yang mana ketika pengguna menekan salah satu berita, maka halaman akan berubah ke bagian detail berita.

Selanjutnya bukalah file **detail_page.dart** dan masukkanlah kode program berikut:

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

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, appProvider, child) {
        return Scaffold(
          appBar: AppBar(
            title:       Text("Detail       News",       style:
appProvider.titleFontStyle),
```

```
            centerTitle: true,
          ),
          body: ListView(
            children: [
              AspectRatio(
                aspectRatio: 16 / 9,
                child: Image.network(
                  appProvider.imageOfArticle,
                  fit: BoxFit.cover,
                  errorBuilder: (context, error, stackTrace) {
                    return Image.network(
                      width: double.infinity,
                      height: double.infinity,
                      "https://demofree.sirv.com/nope-not-
here.jpg",
                      fit: BoxFit.cover,
                    );
                  },
                ),
              ),
              SizedBox(height: 20),
              Padding(
                padding:  const  EdgeInsets.symmetric(horizontal:
10),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      appProvider.titleOfArticle,
                      style: appProvider.titleFontStyle,
                      textAlign: TextAlign.center,
                    ),
                    SizedBox(height: 15),
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                      children: [
```

```
                        Text("Author",                      style:
appProvider.universalFontStyle),
                        SizedBox(width: 50),
                        Expanded(child:                  Text(":
${appProvider.authorName}")),
                      ],
                    ),
                    SizedBox(height: 5),
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                      children: [
                        Text(
                          "Published",
                          style: appProvider.universalFontStyle,
                        ),
                        SizedBox(width: 30),
                        Expanded(
                          child: Text(
                            ":       ${DateFormat("d       MMMM
yyyy").format(appProvider.publishDateTime)}",
                          ),
                        ),
                      ],
                    ),
                    SizedBox(height: 40),
                    Text(
                      "Description News:",
                      style: appProvider.titleFontStyle,
                    ),
                    Text(
                      "${appProvider.descOfArticle}",
                      style: appProvider.universalFontStyle,
                      textAlign: TextAlign.justify,
                    ),
                  ],
                ),
```

```
                ),
              ],
            ),
          );
        },
      );
    }
}
```

Halaman ini merupakan halaman lanjutan ketika pengguna menekan salah satu berita yang ada pada halaman *home*. Halaman ini dibentuk dengan menggunakan widget **ListView()** dan menampilkan beberapa informasi, seperti gambar, judul, nama penulis, tanggal publikasi, dan deskripsi dari berita tersebut.