

BAB V

THINGSPEAK

Pada praktikum MCS bab 5, praktikan akan belajar membuat sebuah aplikasi yang terhubung dengan *software* yang umum digunakan untuk *project Internet of Things* (IoT), yakni **Thingspeak**. Thingspeak merupakan sebuah *software open source* yang dapat digunakan untuk mengumpulkan, mengupdate, hingga memantau suatu data melalui *Application Programming Interface* (API). Aplikasi yang dikembangkan nantinya akan mengambil data API dalam bentuk JSON. JSON memiliki beberapa format data yang dapat digunakan, seperti *array of object*, *array with nested object* dan *object with array*.

5.1 Tujuan Praktikum

Tujuan	Penjelasan
Memperkenalkan Thingspeak	Pada bab ini, praktikan akan diberikan gambaran singkat mengenai Thingspeak
Mampu menghubungkan aplikasi yang dibangun dengan Thingspeak	Pada bab ini, praktikan akan membangun sebuah aplikasi yang nantinya akan dihubungkan dengan Thingspeak

5.2 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

5.3 Materi Praktikum

Interaksi dengan JSON memiliki dua metode yaitu ***encode*** dan ***decode***. ***Encode*** adalah proses mengubah *object* dari suatu bahasa pemrograman menjadi format JSON, contohnya adalah ketika mengirim data (biasanya ke *database* sebelum diproses dengan *query*). Sedangkan ***decode*** adalah kebalikannya, yaitu mengubah format JSON menjadi *object* yang dapat dimengerti oleh bahasa pemrograman tertentu, contohnya adalah ketika aplikasi membaca data dari API.

```
[
  {
    "id": 1,
    "name": "John",
    "age": 30
  },
  {
    "id": 2,
    "name": "Jane",
    "age": 25
  },
  {
    "id": 3,
    "name": "Bob",
    "age": 40
  }
]
```

Gambar 5.1 *Array of Objects*

```

{
  "name": "John",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zip": "12345"
  }
}

```

Gambar 5.2 Array with Nested Object

```

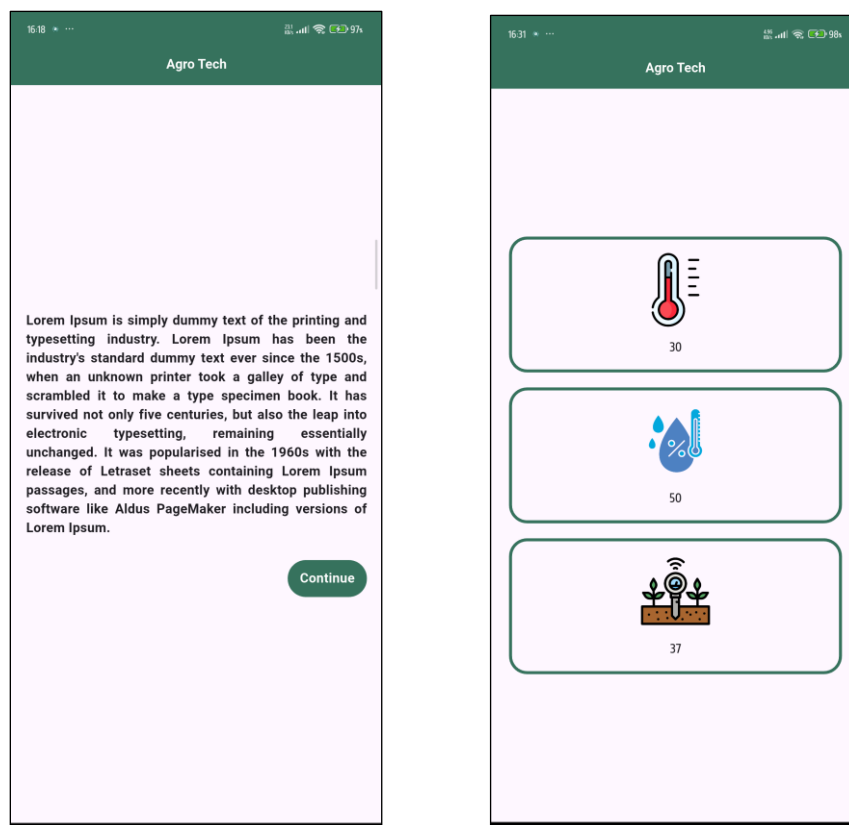
1 {
2   "name": "John",
3   "age": 30,
4   "hobbies": ["reading", "traveling", "hiking"]
5 }

```

Gambar 5.3 Object with Array

5.4 Prosedur Praktikum

5.4.1 Tampilan Aplikasi

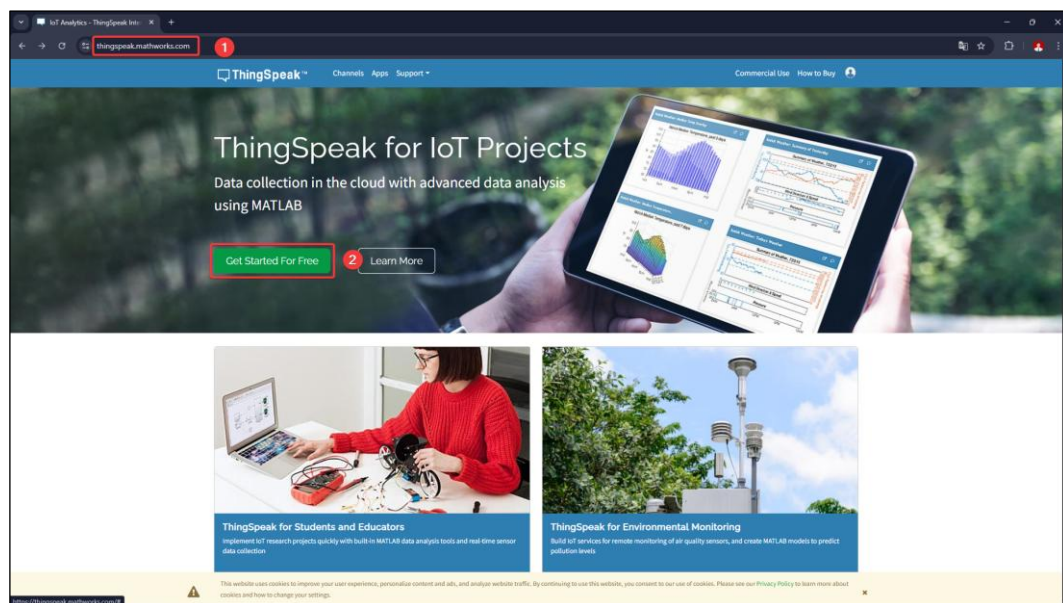


Gambar 5.4 Tampilan Halaman Aplikasi yang Akan diimplementasikan

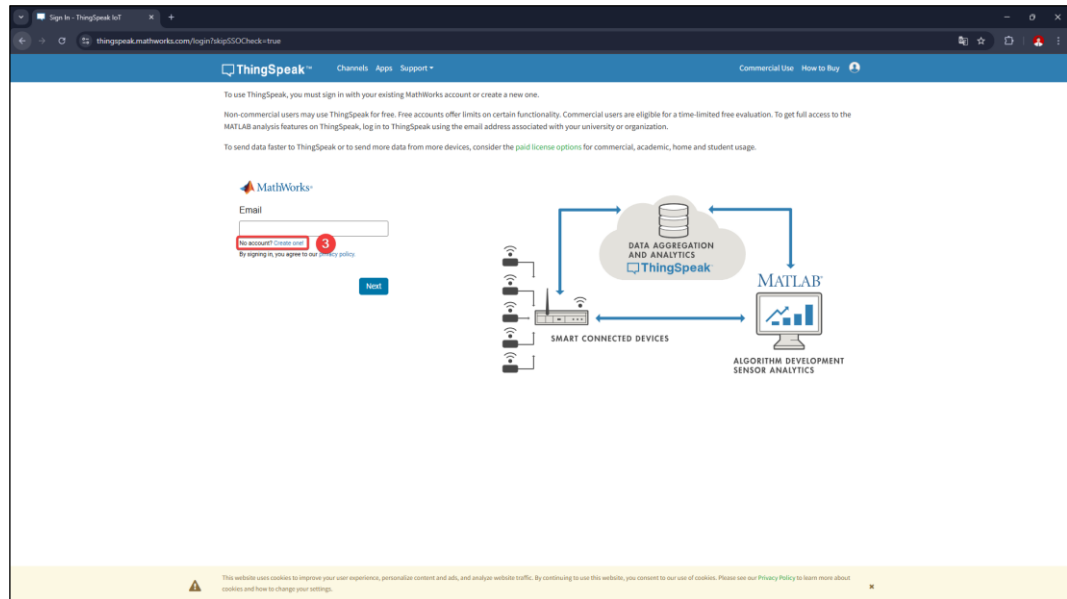
Penjelasan terkait bagaimana cara aplikasi bekerja akan diterangkan oleh asisten yang mengajar.

5.4.2 Implementasi Aplikasi

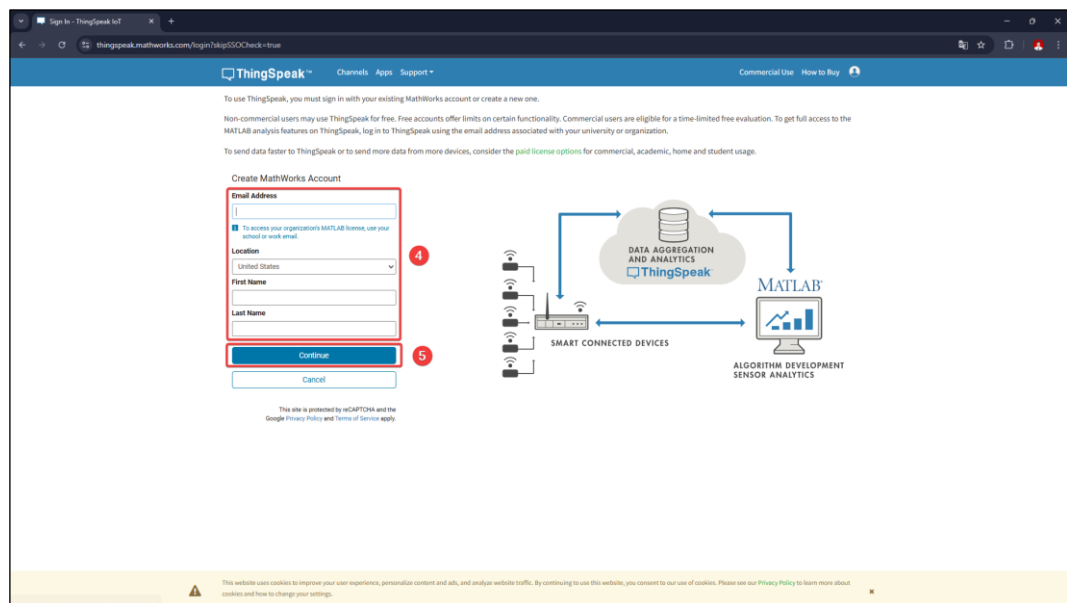
Dalam mengimplementasikan tampilan dari desain aplikasi di atas, terdapat beberapa langkah yang harus dilewati terlebih dahulu agar proses praktikum dapat berjalan dengan lancar dan terselesaikan sesuai dengan apa yang dituju. Sebelum membuat *project* flutter yang baru, praktikan diharapkan untuk membuat akun Thingspeak terlebih dahulu pada halaman *website* <https://thingspeak.mathworks.com/>. Kemudian, daftarkan email masing-masing pada halaman *website* tersebut.



Gambar 5.5 Tampilan Halaman *Website* Thingspeak



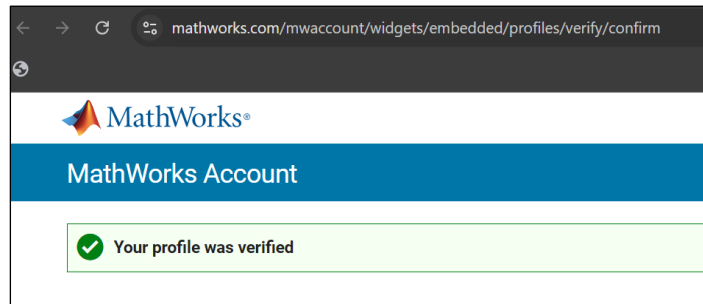
Gambar 5.6 Proses Pembuatan Akun



Gambar 5.7 Proses Pengisian Data Diri

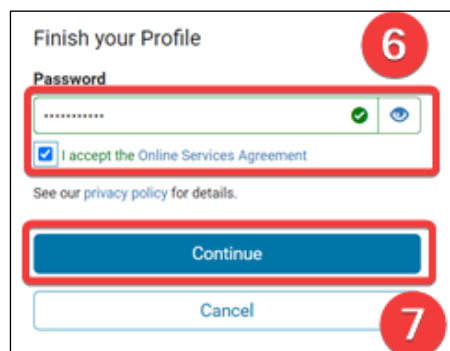
Pada saat registrasi akun, Thingspeak menyediakan 4 *field* yang harus diisikan, antara lain *email address*, *location*, *first name* dan *last name*. Pada bagian *location*, carilah *country* **Indonesia**. Setelah semua *field* diisikan, tekanlah tombol *continue* untuk melanjutkan ke tahap berikutnya. Setelah menekan tombol tersebut, pengguna akan mendapatkan *email* dari Thingspeak untuk melakukan verifikasi

akun. Tekanlah tombol yang ada pada bagian *email* dan nantinya halaman Thingspeak akan berubah, seperti yang terlihat pada Gambar 5.8.

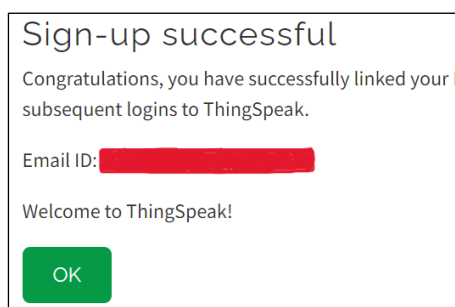


Gambar 5.8 Hasil Verifikasi Akun

Setelah mendapatkan verifikasi dari Thingspeak, pengguna dapat melakukan konfigurasi *password* terhadap akun yang telah didaftarkan. Isilah *field password* yang telah disediakan dengan ketentuan yang telah ditetapkan oleh Thingspeak dan tekanlah tombol *continue* untuk mengakhiri proses registrasi akun.



Gambar 5.9 Konfigurasi *Password* Thingspeak

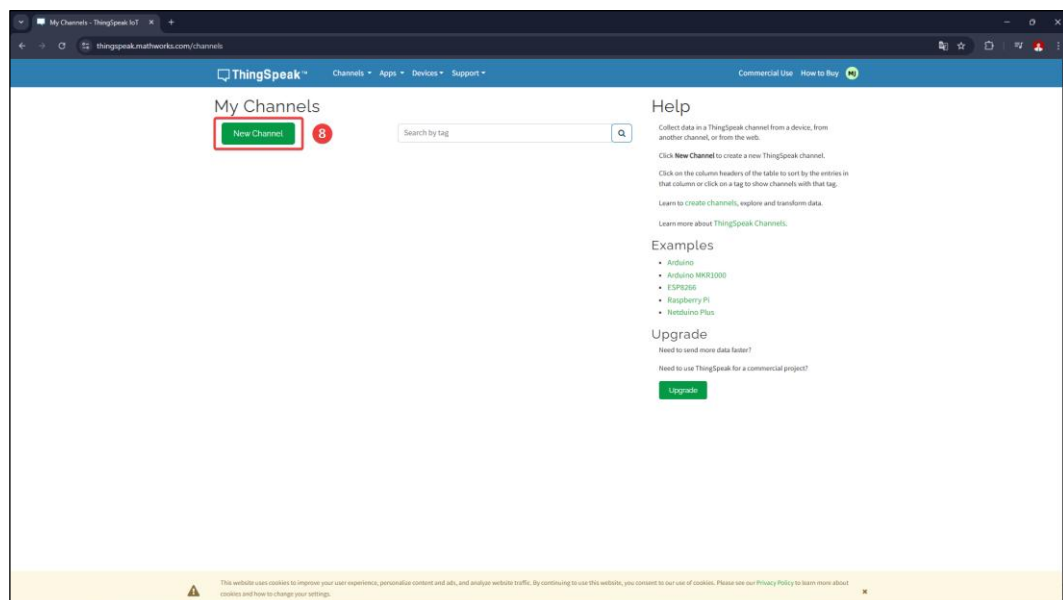


Gambar 5.10 Tampilan Ketika Proses Registrasi Akun Berhasil

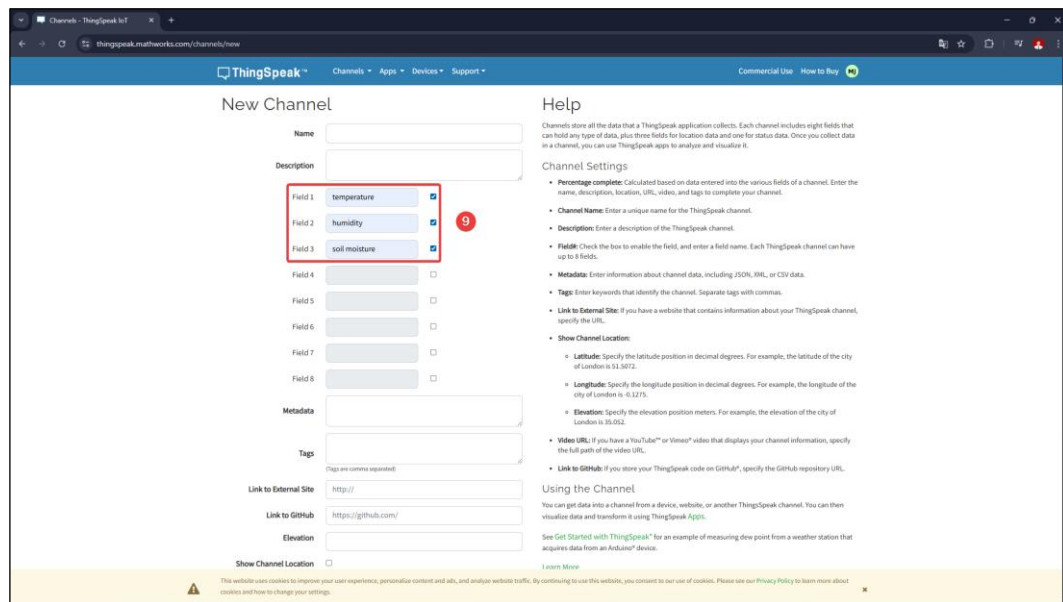
Setelah berhasil mendaftarkan akun Thingspeak, pengguna dapat menekan tombol OK dan nantinya Thingspeak akan menampilkan halaman utama kepada

pengguna. Thingspeak menyediakan 4 *channel* dan 8 *field* di setiap *channel* bagi pengguna yang menggunakan Thingspeak dengan gratis. Setiap 1 *field* dapat menyimpan 1 data, seperti suhu ruangan yang dibaca oleh sensor DHT dan masuk ke *microcontroller*.

Pada praktikum kali ini, kita hanya membutuhkan 1 *channel* saja yang di dalamnya terdapat 3 *fields*, antara lain **suhu ruangan**, **kelembaban ruangan** dan **kelembaban tanah**. Maka dari itu, setelah berada pada halaman utama, praktikan dapat menekan tombol **New Channel** untuk membentuk *channel* baru dan mengisi kolom *field* sebanyak 3 *field*, seperti yang terlihat pada Gambar 5.12.

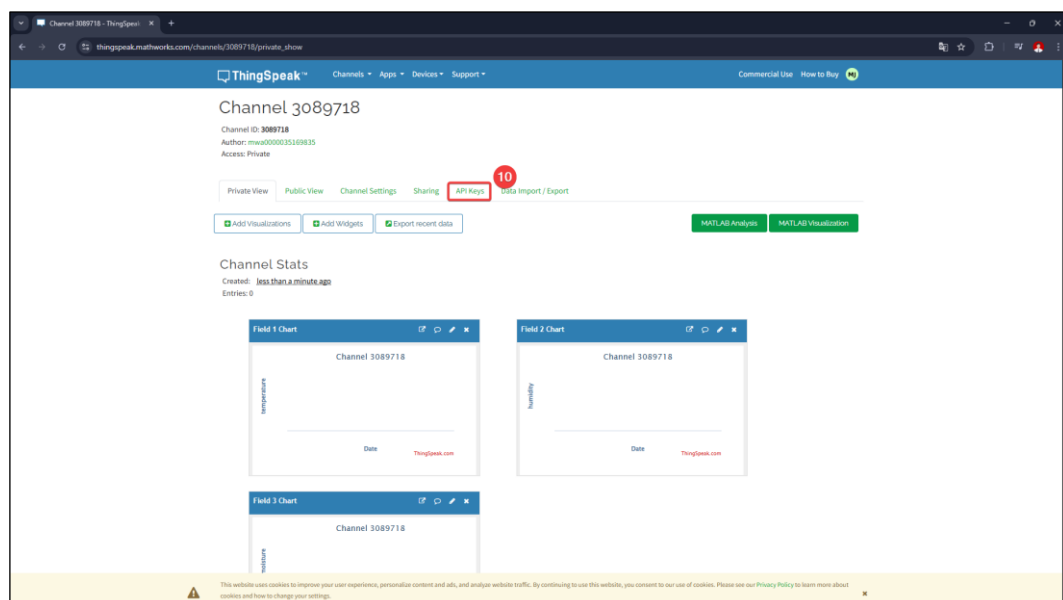


Gambar 5.11 Proses Pembuatan *Channel* Baru

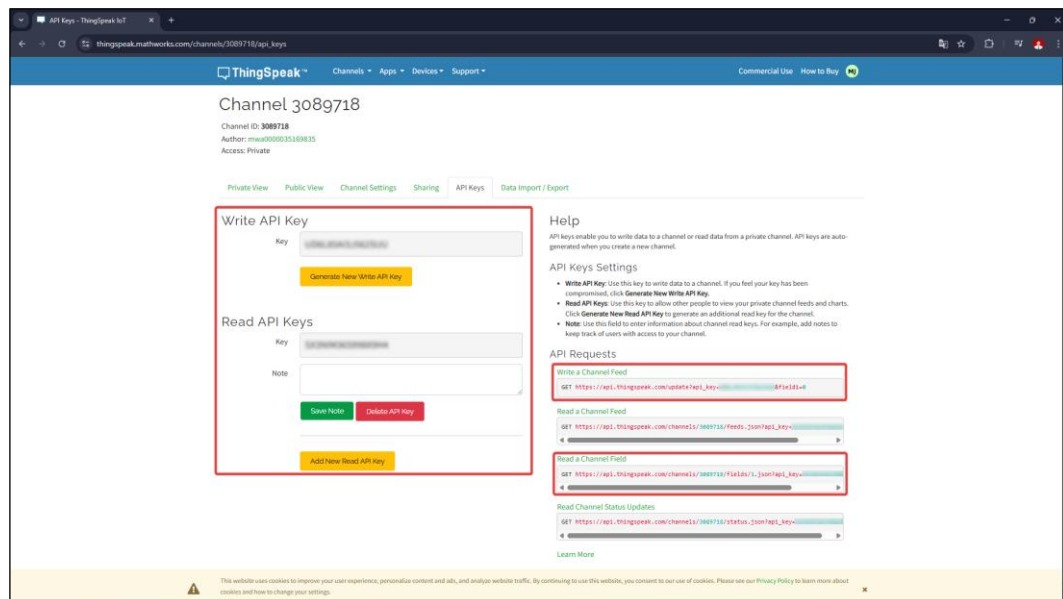


Gambar 5.12 Proses Pembuatan *Field*

Setelah mengisi *field* tersebut, *scroll* halaman tersebut hingga menemukan tombol **Save Channel** dan tekanlah tombol tersebut untuk menyimpan hasil konfigurasi. Tampilan dari halaman *website* Thingspeak akan berubah, seperti yang terlihat pada Gambar 5.13. Pada halaman tersebut, akseslah menu **API keys** untuk melihat *API keys* yang diberikan oleh Thingspeak.



Gambar 5.13 Proses Pengaksesan Menu *API Keys*

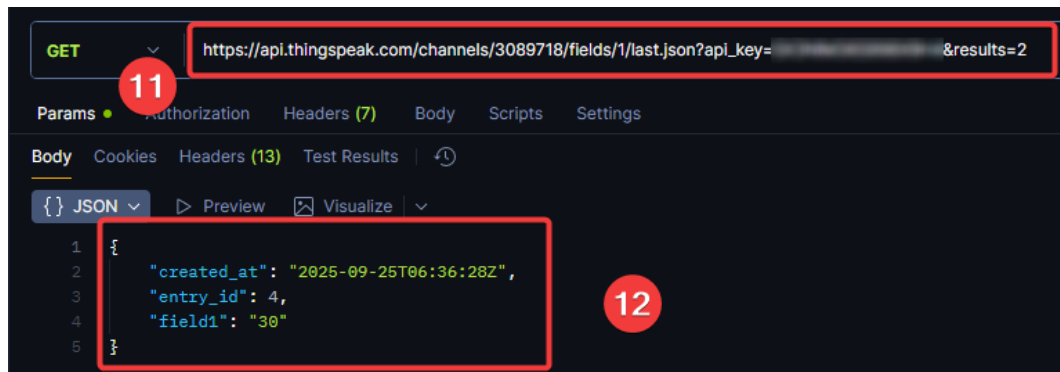


Gambar 5.14 Tampilan Menu API Keys

Write api keys dan **Read api keys** adalah kunci yang berbentuk susunan angka dan huruf yang nantinya akan digabungkan ke url utama sebagai *endpoint*. Kedua API keys tersebut digenerate secara otomatis, sehingga API keys antar pengguna tidak mungkin sama. *Write API keys* digunakan untuk mengirimkan data ke *field* yang telah disediakan, sedangkan *read API keys* digunakan untuk mengambil seluruh data yang telah tersimpan pada *field* masing-masing. Terdapat sedikit modifikasi untuk url dari **Read a Channel Fields**. Modifikasi tersebut dilakukan dengan menambahkan **/last** setelah bagian nomor *fields*. Sehingga url akan berubah, seperti berikut:

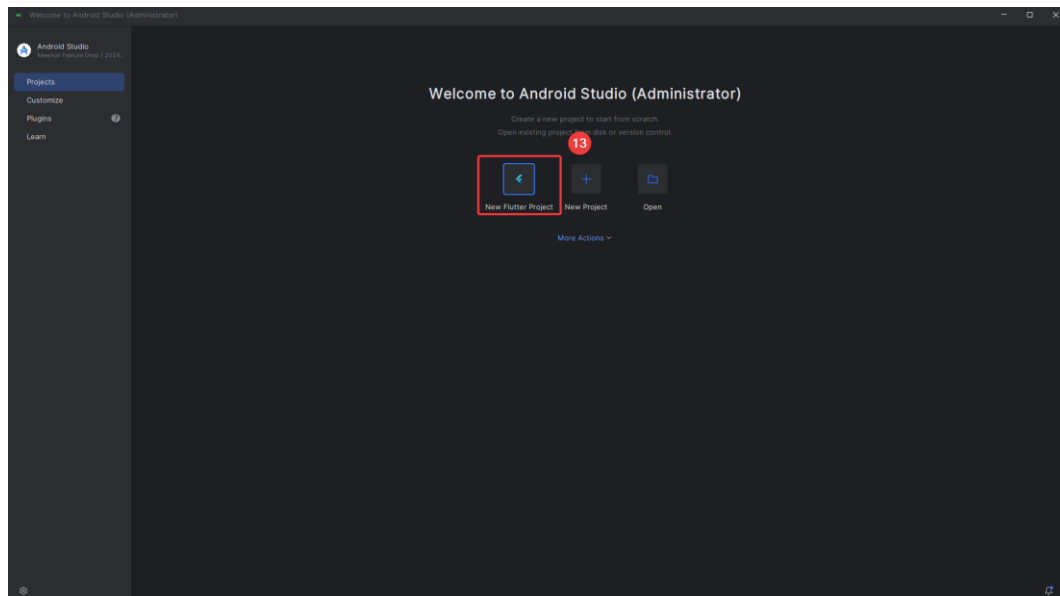
```
https://api.thingspeak.com/channels/<Channel id>/fields/<nomor field>/last.json?api_key=Read API Keys
```

Perubahan url tersebut bertujuan untuk membaca dan menampilkan data paling akhir yang masuk pada masing-masing *fields*. Jika menggunakan url tersebut, maka url akan menghasilkan data, seperti berikut:

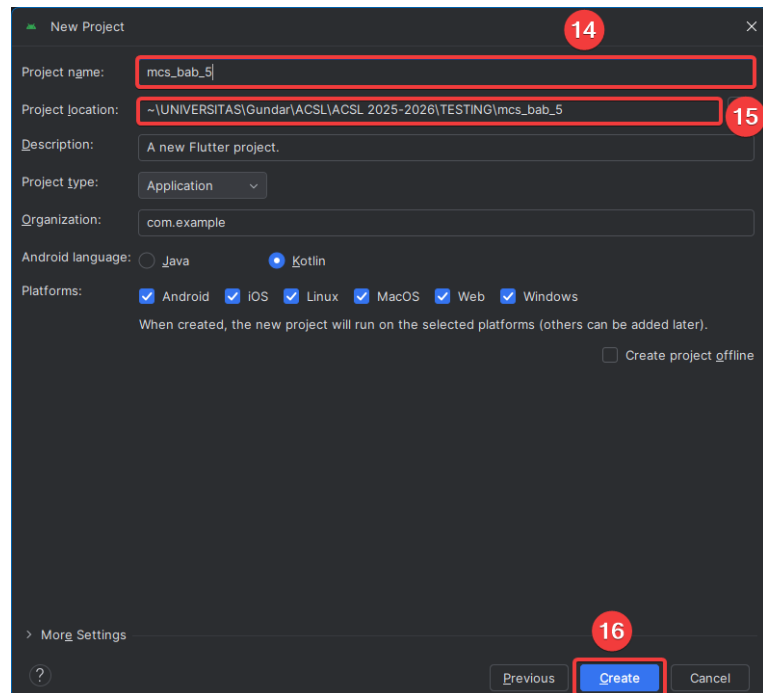


Gambar 5.15 Tampilan API Response Untuk Read API Keys

Setelah mendapatkan hasil *response* dari API, langkah berikutnya adalah membuat *project flutter* terlebih dahulu pada *software android studio*.

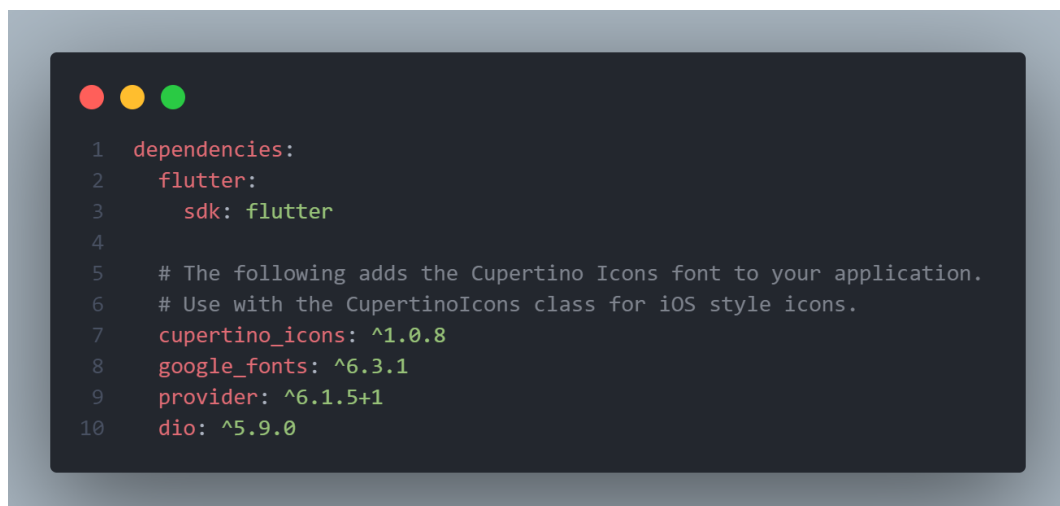


Gambar 5.16 Tampilan Awal Software Android Studio

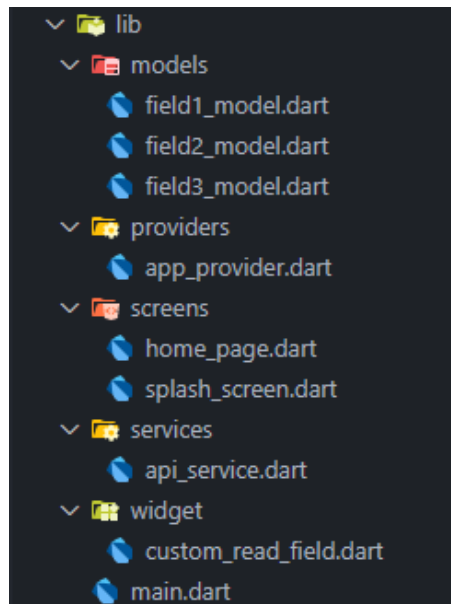


Gambar 5.17 Proses Pembuatan *Project* Baru

Setelah file *project* berhasil terbentuk, pastikan kembali bahwa tampilan yang diberikan oleh android studio sudah berada pada menu **project**. Jika sudah, masuklah ke dalam file **pubspec.yaml** dan tambahkan *package* berikut ke dalam bagian **dependencies**.

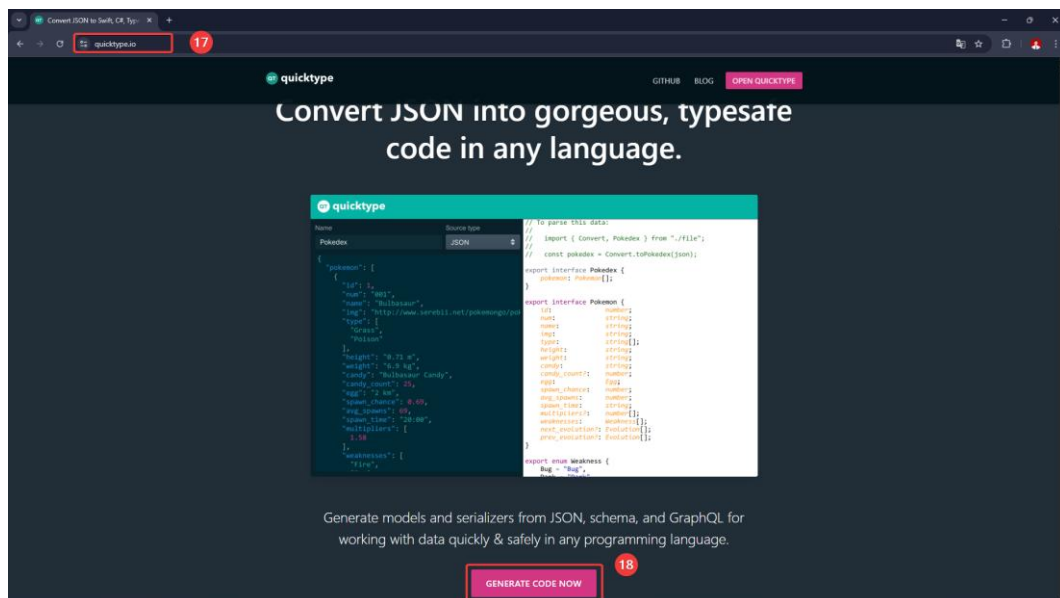


Gambar 5.18 *Package* yang digunakan



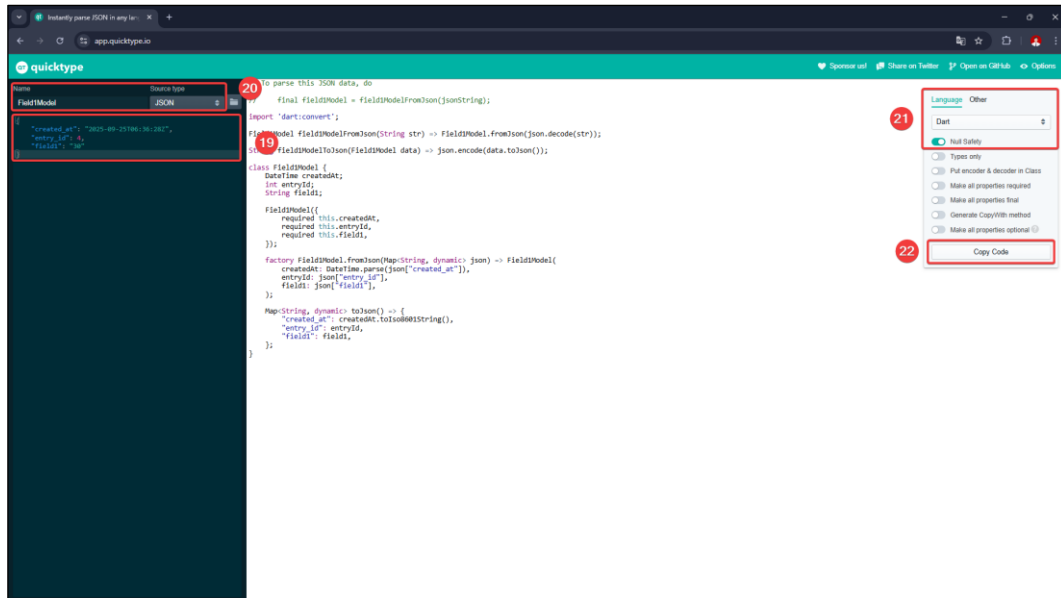
Gambar 5.19 Struktur *Tree Project*

Jika sudah membuat struktur *project*, seperti pada Gambar 5.19, kembalilah ke *API response* yang telah dihasilkan dan salinlah data tersebut kemudian bukalah halaman *website* <https://quicktype.io/> untuk *generate API response* yang diberikan secara otomatis.



Gambar 5.20 Tampilan Halaman *Website Quicktype*

Masukkanlah API *response* yang telah disalin sebelumnya ke bagian *field* sebelah kiri dan ubahlah nama *field* tersebut menjadi **Field1Model**. Pastikan bahasa pemrograman yang digunakan adalah dart dan opsi *null safety* dihidupkan.



Gambar 5.21 Hasil *Generate Model API Response*

Setelah quicktype memberikan model untuk *response API keys*, salinlah model tersebut dengan menekan tombol **Copy Code** dan masukkan ke dalam file **field1_model.dart**. Lakukanlah hal yang sama terhadap *field 2* dan *3*. Jika ketiga model *field* telah dibuat, masuklah ke dalam file **api_service.dart** dan masukkan kode program berikut:

```
class ApiService {
  Dio dio = Dio();

  final String readKey =
    ""; // SESUAIKAN DENGAN READ API KEYS PADA AKUN MASING-
    MASING

  String field1Url =
    "https://api.thingspeak.com/channels/<CHANNEL
    ID>/fields/1/last.json?api_key="; // SESUAIKAN DENGAN CHANNEL ID
    PADA AKUN MASING-MASING

  String field2Url =
```

```

        "https://api.thingspeak.com/channels/<CHANNEL
ID>/fields/2/last.json?api_key="; // SESUAIKAN DENGAN CHANNEL ID
PADA AKUN MASING-MASING

        String field3Url =
            "https://api.thingspeak.com/channels/<CHANNEL
ID>/fields/3/last.json?api_key="; // SESUAIKAN DENGAN CHANNEL ID
PADA AKUN MASING-MASING

// TEMPERATUR FIELD
Future<Field1Model> getField1() async {
    try {
        final response = await dio.get("$field1Url$readKey");
        return Field1Model.fromJson(response.data);
    } catch (e) {
        rethrow;
    }
}

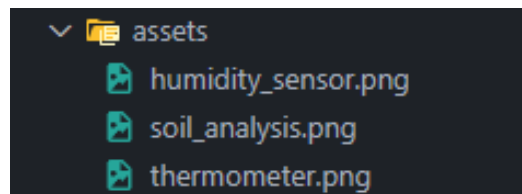
// HUMADITY FIELD
Future<Field2Model> getField2() async {
    try {
        final response = await dio.get("$field2Url$readKey");
        return Field2Model.fromJson(response.data);
    } catch (e) {
        rethrow;
    }
}

// SOIL MOISTURE FIELD
Future<Field3Model> getField3() async {
    try {
        final response = await dio.get("$field3Url$readKey");
        return Field3Model.fromJson(response.data);
    } catch (e) {
        rethrow;
    }
}
}

```

Pada praktikum ini, kita menggunakan *package* **DIO** untuk melakukan komunikasi dengan API dan mendapatkan *response* dari API tersebut yang nantinya akan disimpan ke dalam *function* yang telah disediakan. Fungsi **getField1()** akan menangkap data *response* untuk *field temperature* yang dipanggil melalui url dari variabel `field1Url`. Fungsi **getField2()** akan menangkap data *response* untuk *field humidity* yang dipanggil melalui url dari variabel `field2Url`. Fungsi **getField3()** akan menangkap data *response* untuk *field soil moisture* yang dipanggil melalui url dari variabel `field3Url`. Masing-masing fungsi akan memanggil *endpoint* yang sama, yakni `readKey` dan data dalam bentuk JSON tersebut nantinya akan melewati proses konversi terlebih dahulu menggunakan *property* **fromJson** yang ada di masing-masing *class* model.

Berikutnya tambahkan *folder* baru bernama **assets** pada *root project* dan tambahkan beberapa gambar ke dalam *folder* tersebut.



Gambar 5.22 Gambar pada *Folder Assets*

Setelah menambahkan gambar, bukalah kembali file `pubspec.yaml` dan carilah baris **assets** yang terbungkus dengan tanda *comment* (`//`). Hilangkan tanda tersebut, sehingga terlihat, seperti pada Gambar 5.23.



Gambar 5.23 Tampilan Baris Assets pada `Pubspec.yaml`

Jika sudah menghilangkan *comment* tersebut, lakukanlah `pub get` untuk memperbarui hasil konfigurasi. Jika sudah, bukalah file **app_provider.dart** dan masukkan kode program berikut:

```

class AppProvider extends ChangeNotifier {
  TextStyle roboto14Italic = GoogleFonts.roboto(
    fontSize: 14,
    fontWeight: FontWeight.w400,
  );
  TextStyle roboto14 = GoogleFonts.roboto(
    fontSize: 14,
    fontWeight: FontWeight.w500,
  );
  TextStyle roboto14SemiBold = GoogleFonts.roboto(
    fontSize: 14,
    fontWeight: FontWeight.w600,
  );
  TextStyle roboto14Bold = GoogleFonts.roboto(
    fontSize: 14,
    fontWeight: FontWeight.w700,
  );
  TextStyle roboto16Italic = GoogleFonts.roboto(
    fontSize: 16,
    fontWeight: FontWeight.w400,
  );
  TextStyle roboto16 = GoogleFonts.roboto(
    fontSize: 16,
    fontWeight: FontWeight.w500,
  );
  TextStyle roboto16SemiBold = GoogleFonts.roboto(
    fontSize: 16,
    fontWeight: FontWeight.w600,
  );
  TextStyle roboto16Bold = GoogleFonts.roboto(
    fontSize: 16,
    fontWeight: FontWeight.w700,
  );

  TextStyle whiteRoboto14Bold = GoogleFonts.roboto(
    fontSize: 14,
    fontWeight: FontWeight.w700,
  );

```



```

        color: Colors.white,
    );

    Color mainColor = const Color(0xff36725D);
    String loremIpsum =
        "Lorem Ipsum is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been the industry's standard
dummy text ever since the 1500s, when an unknown printer took a
galley of type and scrambled it to make a type specimen book. It
has survived not only five centuries, but also the leap into
electronic typesetting, remaining essentially unchanged. It was
popularised in the 1960s with the release of Letraset sheets
containing Lorem Ipsum passages, and more recently with desktop
publishing software like Aldus PageMaker including versions of
Lorem Ipsum.";

    String thermoMeterImage = "assets/thermometer.png";
    String humiditySensorImage = "assets/humidity_sensor.png";
    String soilAnalysisImage = "assets/soil_analysis.png";
    Field1Model? field1model;
    Field2Model? field2model;
    Field3Model? field3model;

    goToNextPage({required BuildContext context, required
navigationPage}) {
        Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => navigationPage),
        );
        notifyListeners();
    }

    Future getTemperature() async {
        notifyListeners();
        return field1model = await ApiService().getField1();
    }

    Future getHumidity() async {

```

```

        notifyListeners();
        return field2model = await ApiService().getField2();
    }

    Future getSoilMoisture() async {
        notifyListeners();
        return field3model = await ApiService().getField3();
    }
}

```

Kode program tersebut berisikan kumpulan deklarasi variabel dan fungsi yang nantinya dapat digunakan secara berulang. Variabel dengan tipe String yang bernama **thermoMeterImage**, **humiditySensorImage**, dan **soilAnalysisImage** adalah variabel untuk menyimpan *path* dimana gambar disimpan. Variabel dengan nama **field1model**, **field2model**, dan **field3model** adalah variabel yang nantinya menampung nilai *return* saat *function* yang berada di class ApiService() bernama getField1(), getField2() dan getField3() dipanggil. Fungsi tersebut akan dijalankan ketika fungsi getTemperature(), getHumidity() dan getSoilMoisture() dipanggil.

Kemudian bukalah file **main.dart** dan tuliskan kode program berikut:

```

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

class MyApp extends StatelessWidget {
    const MyApp({super.key});
    @override
    Widget build(BuildContext context) {
        return MultiProvider(
            providers: [
                ChangeNotifierProvider<AppProvider>(
                    create: (context) => AppProvider()
                        ..getTemperature()
                        ..getHumidity()
                        ..getSoilMoisture(),
                ),
            ],
        ),
    },
}

```

```

],
child: MaterialApp(
  title: 'MCS BAB 5',
  debugShowCheckedModeBanner: false,
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
    useMaterial3: true,
  ),
  home: const SplashScreen(),
),);
}
}

```

Pada file `main.dart` terlihat bahwa terdapat pemanggilan terhadap *function* `getTemperature()`, `getHumidity()` dan `getSoilMoisture()` yang bertujuan agar fungsi tersebut langsung dijalankan bersamaan pada saat aplikasi dijalankan. Hal tersebut juga dapat membuat data yang ingin diambil langsung disajikan tanpa menunggu untuk beberapa saat.

Berikutnya bukalah file **`splash_screen.dart`** dan masukkan kode program berikut:

```

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

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  void initState() {
    Provider.of<AppProvider>(context, listen:
false).getTemperature();
    Provider.of<AppProvider>(context, listen:
false).getHumidity();
    Provider.of<AppProvider>(context, listen:
false).getSoilMoisture();
  }
}

```

```

        super.initState();
    }

    @override
    Widget build(BuildContext context) {
        return Consumer<AppProvider>(
            builder: (context, appProvider, child) {
                return Scaffold(
                    appBar: AppBar(
                        title: Text("Agro Tech", style:
appProvider.whiteRoboto14Bold),
                        centerTitle: true,
                        backgroundColor: appProvider.mainColor,
                    ),
                    body: Center(
                        child: ListView(
                            shrinkWrap: true,
                            physics: const NeverScrollableScrollPhysics(),
                            children: [
                                Container(
                                    margin: const EdgeInsets.symmetric(horizontal:
20),
                                    width: double.infinity,
                                    child: Text(
                                        appProvider.loremIpsum,
                                        style: appProvider.roboto14Bold,
                                        textAlign: TextAlign.justify,
                                    ),
                                ),
                                const SizedBox(height: 30),
                                Row(
                                    mainAxisAlignment: MainAxisAlignment.end,
                                    children: [
                                        GestureDetector(
                                            child: Container(

```

```

        margin: const
EdgeInsets.symmetric(horizontal: 20),
        padding: const EdgeInsets.symmetric(
            vertical: 12,
            horizontal: 16,
        ),
        decoration: BoxDecoration(
            borderRadius:
BorderRadius.circular(24),
            color: appProvider.mainColor,
        ),
        child: Text(
            "Continue",
            style: appProvider.whiteRoboto14Bold,
        ),
    ),
    onTap: () {
        appProvider.goToNextPage(
            context: context,
            navigationPage: HomePage(),
        );
    },
),
],
),
],
),
),
);
},
);
}
}

```

Kode tersebut merupakan kode yang digunakan untuk membangun halaman *splash screen* aplikasi yang muncul paling awal pada saat aplikasi dijalankan. Pada kode tersebut, terdapat fungsi void bernama `initState()` yang

memanggil 3 fungsi lain yang telah didefinisikan pada *provider*, seperti **getTemperature()**, **getHumidity()**, dan **getSoilMoisture()** yang dipanggil lagi untuk memastikan bahwa data yang ingin diambil benar-benar telah siap. Meskipun saat inisialisasi *provider* ketiga *function* tersebut sudah dipanggil, hal tersebut tidak dapat menjadi jaminan bahwa *function* yang dipanggil dapat berjalan.

Berikutnya masuklah ke dalam file **custom_read_file.dart** dan masukkan kode program berikut:

```
class CustomReadField extends StatelessWidget {
  String result;
  Color borderColor;
  String image;

  CustomReadField({
    super.key,
    required this.result,
    required this.borderColor,
    required this.image,
  });

  @override
  Widget build(BuildContext context) {
    return Container(
      width: double.infinity,
      margin: const EdgeInsets.symmetric(horizontal: 24),
      padding: const EdgeInsets.symmetric(vertical: 18),
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(24),
        border: Border.all(color: borderColor, width: 4),
      ),
      child: Column(
        children: [
          SizedBox(
            width: MediaQuery.of(context).size.width / 5,
            height: MediaQuery.of(context).size.width / 5,
            child: Image.asset(image, fit: BoxFit.fill),
          ),
        ],
      ),
    );
  }
}
```

```

        const SizedBox(height: 14),
        Center(child: Text(result)),
      ],
    ),);
  }
}

```

Kode program pada file tersebut digunakan untuk memberikan tampilan yang nantinya dapat digunakan pada halaman utama untuk menampilkan data yang diambil dari Thingspeak. Dengan adanya *class* CustomReadField() kita dapat memanggil widget dengan *layout* yang sama, karena kita hanya perlu memanggil *class* CustomReadField() saja yang nantinya *class* ini akan meminta *constructor* yang perlu diisi saat *class* tersebut dipanggil. Terdapat 3 bagian di dalam *constructor* yang harus diisi, **result** adalah untuk menampilkan data dari field *channel* Thingspeak, **borderColor** adalah warna yang digunakan untuk *border* widget yang dibangun dan **image** untuk menampilkan gambar pada widget tersebut.

Berikutnya masuklah ke dalam file **home_page.dart** dan masukkan kode program berikut:

```

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

  @override
  Widget build(BuildContext context) {
    return Consumer<AppProvider>(
      builder: (context, appProvider, child) {
        return Scaffold(
          appBar: AppBar(
            title: Text("Agro Tech", style:
appProvider.whiteRoboto14Bold),
            centerTitle: true,
            automaticallyImplyLeading: false,
            backgroundColor: appProvider.mainColor,
          ),
          body: Center(
            child: ListView(

```

```

        shrinkWrap: true,
        physics: const NeverScrollableScrollPhysics(),
        children: [
            // TEMPERATUR FIELD
            CustomReadField(
                result: appProvider.field1model!.field1,
                borderColor: appProvider.mainColor,
                image: appProvider.thermoMeterImage,
            ),

            const SizedBox(height: 20),

            // HUMADITY FIELD
            CustomReadField(
                result: appProvider.field2model!.field2,
                borderColor: appProvider.mainColor,
                image: appProvider.humiditySensorImage,
            ),

            const SizedBox(height: 20),

            // SOIL MOISTURE FIELD
            CustomReadField(
                result: appProvider.field3model!.field3,
                borderColor: appProvider.mainColor,
                image: appProvider.soilAnalysisImage,
            ),
        ],
    ),
),
);
},
);
}
}

```


Kode program yang digunakan pada halaman ini akan *generate* sebuah tampilan yang akan menampilkan data yang diambil pada Thingspeak melalui JSON. Widget yang digunakan untuk menampilkan data tersebut adalah CustomReadField() dan penggunaannya hanya perlu memanggil *class*nya saja. Ketika *class* CustomReadField() dipanggil, *class* tersebut akan meminta 3 *constructor* yang harus diisi, yakni *result* yang diisi dengan nilai *field*, *borderColor* yang diisi dengan warna, dan *image* yang diisi dengan gambar yang telah disiapkan. Seluruh data tersebut diambil melalui pendefinisian yang telah dilakukan pada *provider*.