**Pengenalan Dasar-Dasar Flutter**

Sebelumnya kita telah belajar bagaimana cara menginstal Flutter pada komputer dan menjalankan aplikasi Flutter untuk pertama kalinya. Nah, kali ini kita akan mempelajari struktur *project* Flutter dan membuat aplikasi pertama kita. Mari kita simak uraian berikut ini.

**Struktur Project Flutter**

Setelah membuat *project* Flutter pertama kali, flutter akan membuatkan struktur *project*. Ketika membuka folder *project*Flutter pada berkas explorer, kita akan mendapati folder-folder seperti berikut:

Untuk tingkat pemula kita tidak perlu mengetahui seluk-beluk struktur *project* secara mendetail. Di sini Anda hanya cukup mengenal beberapa dari folder-folder tersebut.

Berikut beberapa folder yang harus Anda ketahui:

* **android/**  
  Folder ini merupakan tempat Anda untuk mengatur konfigurasi untuk aplikasi android. Di dalamnya terdapat file gradle, AndroidManifest, dan lain-lain. File-file tersebut sangat umum ketika Anda membuat aplikasi android native (menggunakan bahasa pemrograman Java atau Kotlin), nanti Anda akan melakukan beberapa setting pada folder android seiring waktu.
* **ios/**  
  Sama halnya dengan folder android, hanya saja ini untuk iOS. Folder ini merupakan tempat konfigurasi untuk aplikasi iOS. Ketika kita hendak membuat project flutter yang dapat berjalan pada iPhone, Anda akan berkutat dengan folder ini.
* **build/**  
  Ketika Anda melakukan build project flutter, hasil build akan ada pada folder ini. Sebagai contoh, ketika Anda ingin membuat file APK untuk project flutter, maka hasil file tersebut ada dalam folder ini. Folder ini hanya akan ada ketika sudah pernah mem-build project, dan akan terhapus jika menjalankan flutter clean.
* **lib/**  
  Ini merupakan folder utama ketika Anda mengerjakan project flutter. Seluruh source code flutter Anda akan berada pada folder ini.
* **test/**  
  Folder ini tempat Anda menyimpan source code testing. Untuk pemula tidak akan berkutat pada folder ini.

**Hello World App**

Ketika pertama kali membuat project Flutter, kita akan diberikan aplikasi contoh yaitu aplikasi *counter*. *Source code* utama Flutter kita ada pada file **lib/main.dart** dan kodenya kurang lebih seperti berikut:

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatelessWidget {
8. @override
9. Widget build(BuildContext context) {
10. return MaterialApp(
11. title: 'Flutter Demo',
12. theme: ThemeData(
13. primarySwatch: Colors.blue,
14. visualDensity: VisualDensity.adaptivePlatformDensity,
15. ),
16. home: MyHomePage(title: 'Flutter Demo Home Page'),
17. );
18. }
19. }
21. class MyHomePage extends StatefulWidget {
22. MyHomePage({Key key, this.title}) : super(key: key);
24. final String title;
26. @override
27. \_MyHomePageState createState() => \_MyHomePageState();
28. }
30. class \_MyHomePageState extends State<MyHomePage> {
31. int \_counter = 0;
33. void \_incrementCounter() {
34. setState(() {
35. \_counter++;
36. });
37. }
39. @override
40. Widget build(BuildContext context) {
41. return Scaffold(
42. appBar: AppBar(
43. title: Text(widget.title),
44. ),
45. body: Center(
46. child: Column(
47. mainAxisAlignment: MainAxisAlignment.center,
48. children: <Widget>[
49. Text(
50. 'You have pushed the button this many times:',
51. ),
52. Text(
53. '$\_counter',
54. style: Theme.of(context).textTheme.headline4,
55. ),
56. ],
57. ),
58. ),
59. floatingActionButton: FloatingActionButton(
60. onPressed: \_incrementCounter,
61. tooltip: 'Increment',
62. child: Icon(Icons.add),
63. ),
64. );
65. }
66. }

Kode di atas merupakan kode *starter* yang di-*generate* sebagai contoh ketika Anda membuat project. Anda dapat mempelajari kode tersebut untuk mengetahui bagaimana sebuah aplikasi Flutter disusun.

Untuk saat ini kita tidak akan menggunakan kode tersebut. Jadi, hapus semua isi berkas **main.dart** tersebut dan tulis kode untuk aplikasi kita sendiri, yaitu aplikasi sederhana untuk menampilkan teks Hello world.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatelessWidget {
8. @override
9. Widget build(BuildContext context) {
10. return MaterialApp(
11. title: 'Flutter Demo',
12. theme: ThemeData(
13. primarySwatch: Colors.blue,
14. ),
15. home: Scaffold(
16. appBar: AppBar(
17. title: Text('Hello, world!'),
18. ),
19. body: Center(
20. child: Text('Hello, world!'),
21. ),
22. ),
23. );
24. }
25. }

Mari kita bahas satu per satu kodenya!

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

Import digunakan untuk memanggil fungsi-fungsi dari berkas Flutter yang lain. Pada kode di atas, kita meng-*import* kode-kode yang terdapat dalam *library* material bawaan Flutter. Library tersebut menyediakan widget yang termasuk dalam *material design*. Pastikan kode ini ada pada bagian atas sebelum kode lain dijalankan.

1. void main() => runApp(MyApp());

main() merupakan kode dasar dari project Flutter kita. Flutter akan menjalankan fungsi main() pertama kali, yang nantinya akan menjalankan runApp() dan memanggil MyApp().

1. class MyApp extends StatelessWidget {
2. @override
3. Widget build(BuildContext context) {
4. return MaterialApp(
5. title: 'Flutter Demo',
6. theme: ThemeData(
7. primarySwatch: Colors.blue,
8. ),
9. home: Scaffold(
10. appBar: AppBar(
11. title: Text('Hello, world!'),
12. ),
13. body: Center(
14. child: Text('Hello, world!'),
15. ),
16. ),
17. );
18. }
19. }

MyApp di sini merupakan sebuah *class* yang menampilkan komponen Flutter ke layar atau dikenal dengan Widgets.

Jika aplikasi dijalankan maka akan seperti berikut:

Setiap komponen di dalam Flutter terdiri dari widget. Bahkan aplikasi Flutter itu sendiri merupakan sebuah widget. Pada contoh kode di atas kelas MyApp yang merupakan sebuah widget mengembalikan atau menampilkan widget MaterialApp. MaterialApp ini adalah widget bawaan Flutter yang akan menjadi fondasi dari aplikasi Anda. Ia umum digunakan supaya aplikasi bisa menerapkan material design. Widget ini mengatur beberapa hal, termasuk tema aplikasi, tampilan utama aplikasi, rute untuk navigasi antar halaman, dan lain-lain.

Selanjutnya Scaffold memungkinkan kita mendesain struktur layout dasar yang sesuai dengan material design. Dengan Scaffold Anda dapat mengatur App Bar, Floating Action Button, Drawer, dan lain-lain.

Ibarat rumah, MaterialApp ini adalah sebagai tanah yang umumnya digunakan hanya sekali saja, sedangkan Scaffold ini adalah sebagai tembok atau bangunannya.

Terakhir, seharusnya sudah cukup jelas kita menggunakan widget Text untuk menampilkan teks. Pada contoh di atas widget Text dibungkus dengan widget Center yang berfungsi supaya widget di dalamnya (*child*) tampil pada posisi tengah.

**Tips**: *Documentation is your best friend*.

Flutter memiliki banyak sekali widget dan tentunya akan menjadi tugas yang berat untuk menghafalkan semuanya. Namun tenang, Flutter dilengkapi dengan dokumentasi yang cukup lengkap sebagai panduan kita dalam mempelajari dan mengembangkan Flutter. Misalnya, penjelasan lebih lengkap tentang widget yang kita gunakan di atas dapat dilihat pada tautan berikut:

* <https://api.flutter.dev/flutter/material/MaterialApp-class.html>
* <https://api.flutter.dev/flutter/material/Scaffold-class.html>
* <https://api.flutter.dev/flutter/widgets/Center-class.html>
* <https://api.flutter.dev/flutter/widgets/Text-class.html>

## Menggunakan Packages

### Package Dependencies

Dalam pengembangan suatu aplikasi, kita tidak akan lepas dari package/library (selanjutnya akan disebut package). Package di sini merupakan sebuah kode yang dibuat untuk menyelesaikan suatu masalah. Contohnya ketika aplikasi yang kita buat membutuhkan fitur kalender sementara fitur tersebut tidak di-support oleh Flutter. Alih-alih membuat fitur kalender dari nol, kita dapat menggunakan package yang telah dibuat oleh developer lain. Waktu pembuatan fitur menjadi lebih singkat!

Package dependencies merupakan sekumpulan package yang dibutuhkan dalam pengembangan aplikasi. Package tersebut akan diatur oleh package manager. Setiap bahasa pemrograman memiliki package manager-nya masing-masing, contohnya NodeJS memiliki npm atau yarn, Java dengan maven atau gradle, PHP dengan composer. Begitu pula dengan Flutter yang ditulis dengan bahasa dart memiliki package manager bernama **pub**.

Kali ini kita akan membahas mengenai package manager pub, bagaimana menggunakan pub, dan di mana mencari package yang dapat digunakan untuk aplikasi kita.

### Dart Pub

Seperti yang telah kita singgung sebelumnya Pub merupakan sebuah package manager. Pub memiliki tugas untuk mengatur package apa saja yang dibutuhkan dalam pengembangan aplikasi. Pada package manager kita dapat mengatur versi package yang ingin kita gunakan. Pengaturan versi sangat penting karena ketika versi flutter/dart yang digunakan tidak cocok dengan package yang kita butuhkan akan berpengaruh pada jalannya aplikasi yang kita buat. Oleh karena itu, kita harus memastikan versi yang kompatibel dengan versi Flutter yang terinstal.

Lalu bagaimana kita menggunakan pub pada project Flutter kita? Untuk mengatur package-package yang akan kita gunakan, cukup buka berkas **pubspec.yaml** yang ada pada folder project.

Ketika membuka berkas **pubspec.yaml** kita akan melihat begitu banyak pengaturan tapi tidak perlu khawatir karena yang kita bahas hanya mengenai package dependencies-nya saja.

Coba kita fokus pada kode yang ada pada **pubspec.yaml** berikut:

1. dependencies:
2. flutter:
3. sdk: flutter
5. # The following adds the Cupertino Icons font to your application.
6. # Use with the CupertinoIcons class for iOS style icons.
7. cupertino\_icons: ^0.1.2
9. dev\_dependencies:
10. flutter\_test:
11. sdk: flutter

Kode di atas merupakan package-package yang digunakan pada project Flutter kita. Jika kita perhatikan, terdapat 2 jenis dependency yaitu dependencies dan dev\_dependencies. Fungsi dev\_dependencies digunakan untuk package-package yang berkaitan ketika proses pengembangan aplikasi Flutter, contohnya seperti flutter\_test yang digunakan untuk testing. Package di dalam dev\_dependencies tidak akan disertakan ketika aplikasi dirilis pada play store atau app store. Fungsi dependencies digunakan untuk package-package yang langsung berkaitan dengan fitur aplikasi Flutter, contohnya seperti cupertino\_icons yang digunakan untuk mendapatkan ikon-ikon cupertino (icon untuk iOS) dan contoh lainnya seperti cloud\_firestore yang merupakan package untuk firebase firestore.

Sekarang kita akan fokus pada dependencies. Untuk mendaftarkan package yang dibutuhkan kita cukup menulis seperti di bawah ini pada bagian dependencies:

1. nama\_package: versi

nama\_package merupakan nama package yang kita butuhkan, lalu disambung dengan versinya. Penulisan versi bisa langsung seperti contoh 0.1.2, atau kita menambahkan simbol caret (^) seperti ^0.1.2 . Simbol caret (^) artinya: gunakan versi patch terbaru dari versi yang telah ditentukan. Jika versi nya ^0.1.2 artinya kita akan gunakan versi minimal 0.1.2 dan maksimal versi terbaru. Karena itu, jika versi package tersebut sekarang sudah update, maka package yang digunakan merupakan versi terbaru.

**Catatan**: Hanya pada versi patch atau pada angka terakhir yaitu angka 2 jika pada contoh cupertino\_icons: ^0.1.2. Atau kita juga bisa gunakan versi minimal dan maksimal seperti contoh ‘>=0.1.2 <2.0.0’ yang artinya kita akan menggunakan versi terbaru yang ada pada saat ini yang masih berada di dalam range tersebut yaitu lebih besar sama dengan 0.1.2 dan lebih kecil dari 2.0.0.

Okay sebagai contoh kita akan menambahkan sebuah package provider yang nantinya akan kita gunakan.

1. dependencies:
2. flutter:
3. sdk: flutter
5. cupertino\_icons: ^0.1.2
6. provider: ^4.0.1

Yang perlu diperhatikan dalam menulis berkas **.yaml** adalah pada indentasinya. Indentasi atau penggunaan spasi ini sangat penting karena menunjukkan urutan dan blok kode yaml yang dibaca oleh komputer. Sebagai contoh, ketika kita menambahkan package provider, maka kita harus menuliskannya sejajar dengan package lainnya dan juga lebih menjorok ke dalam jika dibandingkan dengan dependencies. Ini menunjukkan bahwa package seperti cupertino\_icons dan provider merupakan bagian dari dependencies yang akan ditambahkan. Setiap jaraknya adalah 2 spasi, jika dependencies menempel pada ujung kiri, maka cupertino\_icons dan provider berjarak 2 spasi dari ujung kiri.

Setelah menambahkan package yang dibutuhkan, kita dapat melakukan **get package** tersebut. Jika Anda menggunakan visual studio code cukup simpan berkas **pubspec.yaml**, maka nanti akan secara otomatis mensinkronisasi pubspec tersebut. Atau, bisa dengan menekan tombol seperti gambar unduh pada pojok kanan atas paling kiri.

Bila menggunakan Android Studio Anda cukup menekan tombol "**Pub get**" pada Android Studio seperti berikut:

Kita juga bisa secara manual menggunakan terminal dengan menjalankan perintah flutter pub get di dalam project tersebut. Setelah melakukan pub get, maka package tersebut sudah dapat digunakan.

### Tempat-tempat mencari package flutter

Flutter memiliki segudang package yang telah dibuat oleh komunitas, lantas di mana kita dapat mencari package-package untuk kita gunakan? Berikut website-website yang dapat Anda gunakan untuk mencari package.

* [https://pub.dev](https://pub.dev/)  
  Website ini merupakan web official untuk Anda mencari package.
* [https://flutterawesome.com](https://flutterawesome.com/)  
  Berisi package-package yang dibuat oleh komunitas, di sini banyak sekali package UI keren yang dapat Anda coba.

### Private Packages

Selain menggunakan package yang ada pada pub.dev Anda juga bisa menggunakan package yang tidak dipublikasikan pada pub.dev tersebut dengan cara menggunakan url git package tersebut:

1. dependencies:
2. plugin1:
3. git:
4. url: git://github.com/flutter/plugin1.git

Atau path direktori package tersebut yang tersimpan secara offline di komputer Anda.

1. dependencies:
2. plugin1:
3. path: ../plugin1/

**Rangkuman Materi**

**Rangkuman Materi**

Mari kita ingat kembali apa saja yang telah dipelajari di modul ini.

* Struktur project Flutter memiliki beberapa folder utama yang penting untuk kita ketahui, yaitu:
  + android/  
    Folder berisi konfigurasi aplikasi untuk platform Android.
  + ios/  
    Folder berisi konfigurasi aplikasi untuk platform iOS.
  + build/  
    Folder berisi hasil build dari aplikasi Flutter, misalnya seperti file APK, IPA, atau folder web yang bisa di-host.
  + lib/  
    Folder utama untuk mengerjakan project Flutter. Seluruh source code Flutter berada pada folder ini.
  + test/  
    Folder berisi source code untuk melakukan testing aplikasi Flutter secara otomatis.
* Ketika pertama kali membuat project Flutter, kita akan diberikan aplikasi starter bertema counter. Kode starter ini memiliki contoh dan komentar yang menjelaskan bagaimana sebuah aplikasi Flutter disusun.
* Flutter memiliki package manager bernama pub. Dengan package manager, kita dapat menambahkan package/library yang dibuat oleh developer lain ke dalam project kita.

Pada modul selanjutnya, kita akan mulai berkenalan dengan widget dan menyusunnya menjadi sebuah tampilan halaman. Sudah siap? Sampai bertemu di modul selanjutnya!