#12 | Pemrograman Asynchronous

| **Summary** | Mampu melakukan pemrograman asynchronous di Flutter |
| --- | --- |
| **URL** | 12-async |
| **Category** | Mobile Flutter |
| **Environment** | web |
| **Status** | Published |
| **Feedback Link** | https://github.com/jti-polinema/flutter-codelab/issues |
| **Author** | Habibie Ed Dien |
| **Author LDAP** | hbb-polinema |
| **Analytics Account** | G-H7507PFEJB |

[Pemrograman Asynchronous di Flutter](#_ok7k5uux6)

[Tujuan Praktikum](#_21yzqg98x7h6)

[Sumber Daya yang Dibutuhkan](#_dusjnkkj26ob)

[Pengetahuan yang Anda harus Miliki](#_319h1e1sjzdz)

[Pendahuluan Pemrograman Asynchronous](#_t0tqoph2ohlh)

[Mengapa kode async itu penting?](#_daq097z5o74c)

[Menggunakan Future](#_q0781k5wmbz)

[Praktikum 1: Mengunduh Data dari Web Service](#_vmpz7ap0erhd)

[Langkah 1: Buat Project Baru](#_spr93e7x3yo2)

[Langkah 2: Cek file pubspec.yaml](#_g8csgggwssw)

[Langkah 3: Buka file main.dart](#_2q85gi9dhaiq)

[Langkah 4: Tambah method getData()](#_p0t5r1yzb5qp)

[Langkah 5: Pindah ke file main.dart](#_xmhut8m5gdv1)

[Langkah 6:](#_4b6h89e2cyxj)

[Langkah 7:](#_5krokjvrwldm)

[Praktikum 2: Menggunakan await/async untuk menghindari callbacks](#_1e543m1dm1c4)

[Langkah 1: Buat file plan\_provider.dart](#_u3l46a7g73v0)

[Langkah 2: Edit main.dart](#_gfa7d3gyzo4o)

[Langkah 3: Tambah method pada model plan.dart](#_wc7bdoy0rvka)

[Langkah 4: Pindah ke PlanScreen](#_dxl4engxhz7o)

[Langkah 5: Edit method \_buildAddTaskButton](#_dswkw4kv0st7)

[Langkah 6: Edit method \_buildTaskTile](#_y44t6huj0yys)

[Langkah 7: Edit \_buildList](#_ufiogbhfaq4m)

[Langkah 8: Tetap di class PlanScreen](#_dxub23gyz21t)

[Langkah 9: Tambah widget SafeArea](#_b28b1kisggzv)

[Praktikum 3: Melengkapi Future](#_riuqbijaf0zt)

[Langkah 1: Edit PlanProvider](#_uv7foq4euher)

[Langkah 2: Edit main.dart](#_di3ldp29lo3b)

[Langkah 3: Edit plan\_screen.dart](#_ma6gko34okso)

[Langkah 4: Error](#_oxnid84td09d)

[Langkah 5: Tambah getter Plan](#_auxtuu9crnqv)

[Langkah 6: Method initState()](#_8b58y7cyb598)

[Langkah 7: Widget build](#_5e2z95f7zwdd)

[Langkah 8: Edit \_buildTaskTile](#_beimcgaw4uuc)

[Langkah 9: Buat screen baru](#_ux10u3wh6amn)

[Langkah 10: Pindah ke class \_PlanCreatorScreenState](#_wz37fkekarcl)

[Langkah 11: Pindah ke method build](#_dognjm4cb57g)

[Langkah 12: Buat widget \_buildListCreator](#_hkjs1mhd2lno)

[Langkah 13: Buat void addPlan()](#_9pt62kmscvzi)

[Langkah 14: Buat widget \_buildMasterPlans()](#_v171501uo3db)

[Praktikum 4: Memanggil Future secara paralel](#_1c5hdgpeva1a)

[Praktikum 5: Menangani Respon Error pada Async Code](#_3fcxszkylq4h)

[Praktikum 6: Menggunakan Future dengan StatefulWidget](#_yocqwzbhogdc)

[Praktikum 7: Manajemen Future dengan FutureBuilder](#_z1vcfco4xqur)

[Praktikum 8: Navigation route dengan Future Function](#_6au90fmaoong)

[Praktikum 9: Memanfaatkan async/await dengan Widget Dialog](#_fegpi3q79bk4)

[Tugas Kelompok PBL](#_3gkrz7x5i2n)

[Selamat!](#_s22qmcj4zu61)

[Apa selanjutnya?](#_9mqifffit2ew)

[Referensi](#_an95hoqxrlhg)

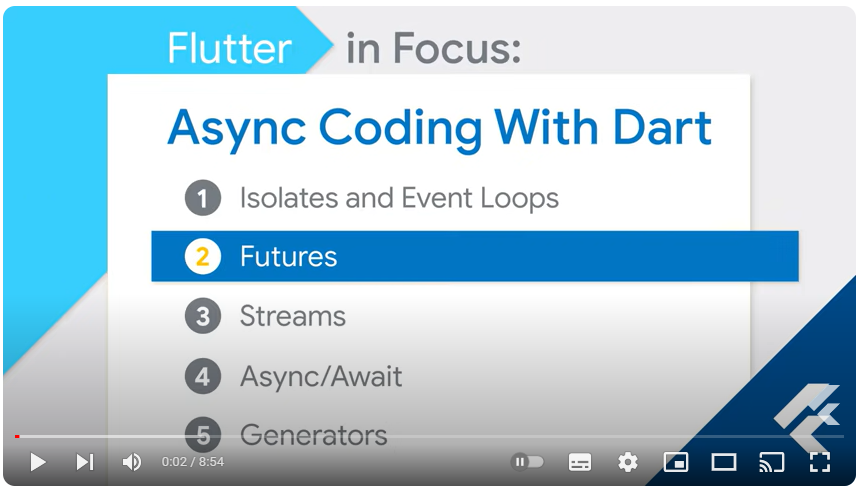
# Pemrograman Asynchronous di Flutter

**Terakhir diperbarui:** 13 November 2023

**Penulis:** Habibie Ed Dien

Pada codelab ini, Anda akan mempelajari tentang pemrograman asynchronous di Flutter beserta contoh penggunaannya. Cara kerja, manfaat, dan cara mengelola thread asynchronous.

Video berikut menjelaskan tentang apa itu Future dan bagaimana manfaatnya terhadap pemrograman asynchronous. Silakan simak dan pahami!



## Tujuan Praktikum

Setelah menyelesaikan codelab ini Anda akan mampu untuk:

* Menggunakan Future
* Menggunakan async/await untuk menghapus callbacks
* Memanggil Future secara paralel
* Mengatasi error pada kode asynchronous
* Menggunakan Future dengan StatefulWidget
* Menggunakan FutureBuilder untuk mengelola Future
* Mengubah navigasi route menjadi fungsi asynchronous
* Mengambil hasil dari widget dialog

| **Catatan:** Materi ini diadaptasi dari Buku “Flutter Cookbook, 100+ step-by-step recipes for building cross-platform, professional grade apps with Flutter 3.10.x and Dart 3.x - Second Edition” oleh Simone Alessandria (2023). |
| --- |

## Sumber Daya yang Dibutuhkan

Berikut merupakan sumber daya yang diperlukan untuk menyelesaikan praktikum ini:

* PC atau Laptop dengan spesifikasi minimum RAM 4GB Processor Core i3
* Koneksi internet minimal kecepatan 1 Mbps
* Chrome Browser
* Perangkat mobile (Android/iOS)

## Pengetahuan yang Anda harus Miliki

* Algoritma dan Pemrograman Dasar
* Pemrograman Berorientasi Objek
* Bahasa Pemrograman Dart Dasar

# Pendahuluan Pemrograman *Asynchronous*

Duration: 10:00

Pemrograman *asynchronous* (*async*) dimanfaatkan untuk aplikasi menyelesaikan tugas-tugas yang membutuhkan waktu (user harus menunggu hasilnya). Contohnya seperti mengunduh gambar dari web, menulis data ke web server, mengerjakan tugas paralel lainnya, hingga menangani inputan dari user. Dengan pemrograman async ini akan meningkatkan kualitas UX dan perangkat lunak kita.

Di Dart dan Flutter Anda dapat menulis program async dengan memanfaatkan **Future** dan pola async/await. Pola ini tersedia hampir di semua bahasa pemrograman modern. Flutter memiliki cara yang efisien untuk membangun UI async dengan class FutureBuilder.

Pada pertemuan 13 berikutnya, kita akan menggunakan streams yaitu lanjutan dari pemrograman async.

| **Catatan:** Secara umum ada beberapa perbedaan antara **Future** dan **Stream**, yaitu:   * **Future** digunakan untuk melakukan **satu operasi asynchronous**, seperti mengunggah data ke web server. * Sedangkan **stream** digunakan untuk melakukan **multi operasi asynchronous**, seperti mendapatkan titik-titik koordinat GPS dari berbagai device IoT atau bereaksi terhadap beberapa operasi update database. |
| --- |

## Mengapa kode async itu penting?

Operasi async banyak digunakan dalam program untuk menyelesaikan pekerjaan paralel yang memerlukan waktu tunggu sampai operasi lain selesai. Beberapa operasi async yang sering dilakukan diantaranya:

* Proses pengambilan data melalui jaringan internet
* Menulis data ke database
* Membaca data dari sebuah file

## Menggunakan *Future* untuk operasi *Asynchronous*

Ketika Anda menulis kode program, pada umumnya berjalan secara sekuensial (baris per baris). Sebagai contoh kode berikut:

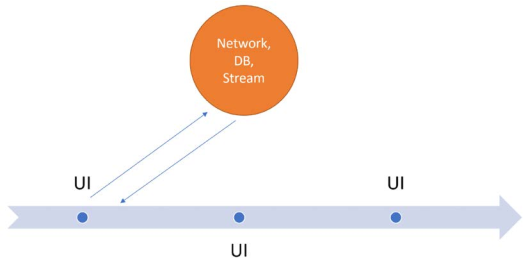
| int x = 3;  int y = x \* 7; |
| --- |

Anda akan menebak bahwa nilai y sama dengan 21 karena instruksi kode x = 3 telah dieksekusi **sebelum** baris berikutnya. Dengan kata lain, baris kedua menunggu untuk instruksi baris pertama diselesaikan terlebih dahulu.

Pada banyak kasus, pola seperti itu sangatlah sempurna, namun pada beberapa kasus, khususnya ketika Anda butuh waktu untuk menyelesaikan instruksi kode yang lebih lama, hal itu tidak direkomendasikan karena akan membuat tampilan aplikasi menjadi tidak responsif. Itulah mengapa hampir semua bahasa pemrograman modern termasuk Dart dapat melakukan operasi *asynchronous* (async).

Operasi async tidak akan menghentikan eksekusi kode pada baris utama, sehingga dapat menyelesaikan operasi secara paralel dengan operasi di kode baris utama.

Perhatikan gambar diagram berikut ini.



Pada diagram tersebut, Anda dapat melihat baris eksekusi utama, yaitu pada bagian UI, dapat melakukan pemanggilan operasi async (misalnya akses data via jaringan, database atau Stream). Namun, UI tetap dieksekusi tanpa harus menunggu hasil operasi async selesai.

Dart adalah bahasa *single-thread*, namun Anda dapat melakukan operasi async agar dapat membuat aplikasi yang responsif. Di Dart dan Flutter Anda dapat menggunakan class Future untuk melakukan operasi async.

| **Catatan:** Future memiliki 3 kondisi yaitu:   * **Uncompleted**: Anda memanggil Future, namun respon belum tersedia. * **Completed successfully**: ketika fungsi then() dipanggil. * **Completed with an error**: ketika fungsi catchError() dipanggil. |
| --- |

Dalam codelab ini, kita akan membangun aplikasi **books** menggunakan plugin http untuk membaca data JSON dari **Google Books API.**

# Praktikum 1: Mengunduh Data dari Web Service (API)

Duration: 20:00

Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil melakukan setup environment Flutter SDK, VS Code, Flutter Plugin, dan Android SDK pada pertemuan pertama. |
| --- |

## Langkah 1: Buat Project Baru

Buatlah sebuah project flutter baru dengan nama **books** di folder **src week-12** repository GitHub Anda.

Kemudian Tambahkan dependensi http dengan mengetik perintah berikut di terminal.

| flutter pub add http |
| --- |

## Langkah 2: Cek file pubspec.yaml

Jika berhasil install plugin, pastikan plugin http telah ada di file pubspec ini seperti berikut.

| dependencies:  flutter:  sdk: flutter  http: ^1.1.0 |
| --- |

**Jika Anda menggunakan macOS**, Anda harus mengaktifkan fitur networking pada file macos/Runner/DebugProfile.entitlements dan macos/Runner/Release.entitlements dengan menambahkan kode berikut:

|  |
| --- |

## Langkah 3: Buka file main.dart

Ketiklah kode seperti berikut ini.

| Soal 1 Tambahkan **nama panggilan Anda** pada title app sebagai identitas hasil pekerjaan Anda. |
| --- |



| **Catatan:**  Tidak ada yang spesial dengan kode di main.dart tersebut. Perlu diperhatikan di kode tersebut terdapat widget CircularProgressIndicator yang akan menampilkan animasi berputar secara terus-menerus, itu pertanda bagus bahwa aplikasi Anda responsif (tidak freeze/lag). Ketika animasi terlihat berhenti, itu berarti UI menunggu proses lain sampai selesai. |
| --- |

## Langkah 4: Tambah method getData()

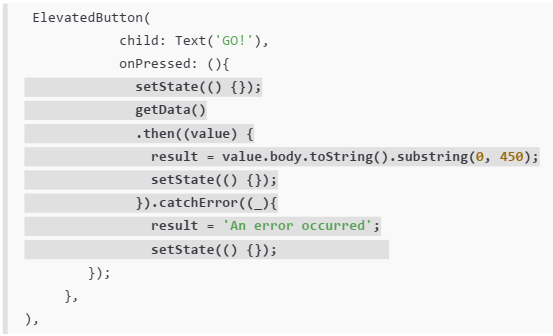
Tambahkan method ini ke dalam class \_FuturePageState yang berguna untuk mengambil data dari API Google Books.

|  |
| --- |

| **Soal 2**  * Carilah judul buku favorit Anda di Google Books, lalu ganti ID buku pada variabel path di kode tersebut. Caranya ambil di URL browser Anda seperti gambar berikut ini.      * Kemudian cobalah akses di browser URI tersebut dengan lengkap seperti ini. Jika menampilkan data JSON, maka Anda telah berhasil. Lakukan *capture* milik Anda dan tulis di README pada laporan praktikum. Lalu lakukan commit dengan pesan “**W12: Soal 2**”. |
| --- |

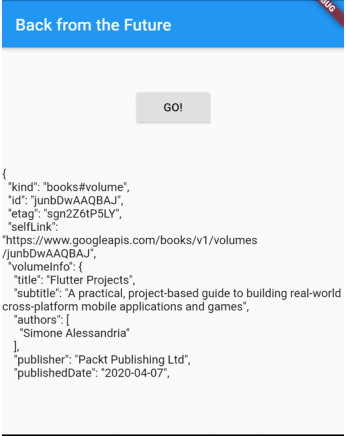
## Langkah 5: Tambah kode di ElevatedButton

Tambahkan kode pada onPressed di ElevatedButton seperti berikut.



Lakukan run aplikasi Flutter Anda. Anda akan melihat tampilan akhir seperti gambar berikut. Jika masih terdapat error, silakan diperbaiki hingga bisa running.

| Soal 3  * Jelaskan maksud kode langkah 5 tersebut terkait substring dan catchError! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 3**”. |
| --- |



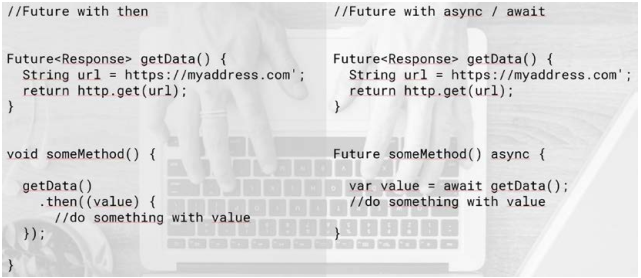
# Praktikum 2: Menggunakan *await/async* untuk menghindari *callbacks*

Duration: 15:00

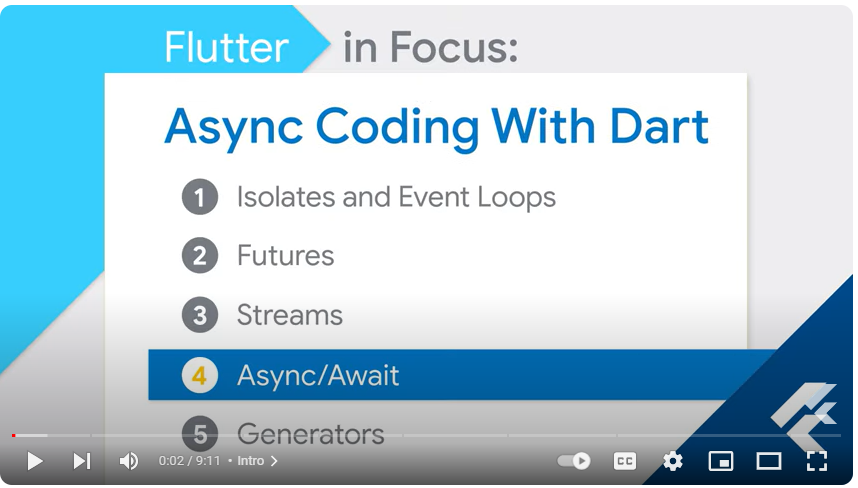
Ada alternatif penggunaan Future yang lebih clean, mudah dibaca dan dirawat, yaitu pola **async/await**. Intinya pada dua kata kunci ini:

* async digunakan untuk menandai suatu method sebagai asynchronous dan itu harus ditambahkan di depan kode function.
* await digunakan untuk memerintahkan menunggu sampai eksekusi suatu function itu selesai dan mengembalikan sebuah value. Untuk then bisa digunakan pada jenis method apapun, sedangkan await hanya bekerja di dalam method async.

Berikut ini contoh kode perbedaan Future dengan then dan async/await.



Untuk memahami lebih dalam penggunaan **async/await**, simaklah video berikut.



Setelah Anda menyelesaikan praktikum 1, Anda dapat melanjutkan praktikum 2 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 1. |
| --- |

Pada codelab ini, kita akan menambah kode dari aplikasi **books** di praktikum sebelumnya.

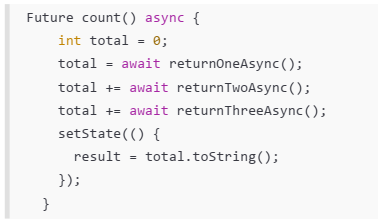
## Langkah 1: Buka file main.dart

Tambahkan tiga method berisi kode seperti berikut di dalam class \_FuturePageState.

| Future<int> returnOneAsync() async {  await Future.delayed(const Duration(seconds: 3));  return 1;  }  Future<int> returnTwoAsync() async {  await Future.delayed(const Duration(seconds: 3));  return 2;  }  Future<int> returnThreeAsync() async {  await Future.delayed(const Duration(seconds: 3));  return 3;  } |
| --- |

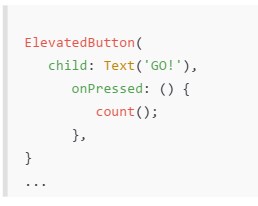
## Langkah 2: Tambah method count()

Lalu tambahkan lagi method ini di bawah ketiga method sebelumnya.



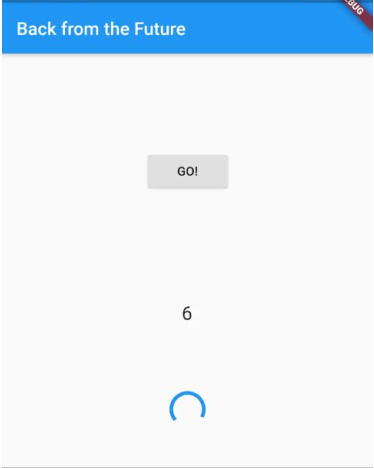
## Langkah 3: Panggil count()

Lakukan *comment* kode sebelumnya, ubah isi kode onPressed() menjadi seperti berikut.



## Langkah 4: Run

Akhirnya, **run** atau tekan **F5** jika aplikasi belum running. Maka Anda akan melihat seperti gambar berikut, hasil angka 6 akan tampil setelah delay 9 detik.



| Soal 4  * Jelaskan maksud kode langkah 1 dan 2 tersebut! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 4**”. |
| --- |

# Praktikum 3: Menggunakan Completer di Future

Duration: 15:00

Menggunakan Future dengan then, catchError, async, dan await mungkin sudah cukup untuk banyak kasus, tetapi ada alternatif melakukan operasi async di Dart dan Flutter yaitu dengan class Completer.

**Completer** membuat object Future yang mana Anda dapat menyelesaikannya nanti (late) dengan return sebuah value atau error.

Setelah Anda menyelesaikan praktikum 2, Anda dapat melanjutkan praktikum 3 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 2. |
| --- |

## Langkah 1: Buka main.dart

Pastikan telah impor package async berikut.

| import 'package:async/async.dart'; |
| --- |

## Langkah 2: Tambahkan variabel dan method

Tambahkan variabel late dan method di class \_FuturePageState seperti ini.

| late Completer completer;  Future getNumber() {  completer = Completer<int>();  calculate();  return completer.future;  }  Future calculate() async {  await Future.delayed(const Duration(seconds : 5));  completer.complete(42);  } |
| --- |

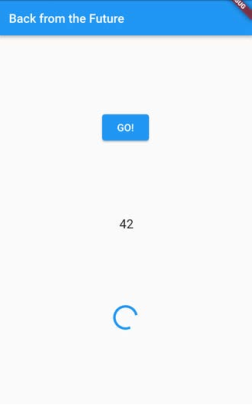
## Langkah 3: Ganti isi kode onPressed()

Tambahkan kode berikut pada fungsi onPressed(). Kode sebelumnya bisa Anda *comment*.



## Langkah 4:

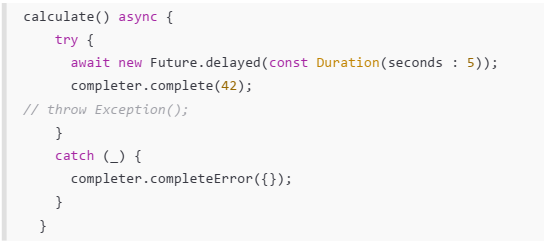
Terakhir, **run** atau tekan **F5** untuk melihat hasilnya jika memang belum running. Bisa juga lakukan **hot restart** jika aplikasi sudah running. Maka hasilnya akan seperti gambar berikut ini. Setelah 5 detik, maka angka 42 akan tampil.



| Soal 5  * Jelaskan maksud kode langkah 2 tersebut! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 5**”. |
| --- |

## Langkah 5: Ganti method calculate()

Gantilah isi code method calculate() seperti kode berikut, atau Anda dapat membuat calculate2()



## Langkah 6: Pindah ke onPressed()

Ganti menjadi kode seperti berikut.

| getNumber().then((value) {  setState(() {  result = value.toString();  });  }).catchError((e) {  result = 'An error occurred';  }); |
| --- |

| Soal 6  * Jelaskan maksud perbedaan kode langkah 2 dengan langkah 5-6 tersebut! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 6**”. |
| --- |

# Praktikum 4: Memanggil Future secara paralel

Duration: 15:00

Ketika Anda membutuhkan untuk menjalankan banyak Future secara bersamaan, ada sebuah class yang dapat Anda gunakan yaitu: FutureGroup.

FutureGroup tersedia di package async, yang mana itu harus diimpor ke file dart Anda, seperti berikut.

| import 'package:async/async.dart'; |
| --- |

| **Catatan:** Package dart:async dan async/async.dart merupakan library yang berbeda. Pada beberapa kasus, Anda membutuhkan kedua lib tersebut untuk me-*run* *code async*. |
| --- |

**FutureGroup** adalah sekumpulan dari Future yang dapat run secara paralel. Ketika run secara paralel, maka konsumsi waktu menjadi lebih hemat (cepat) dibanding run method async secara single setelah itu method async lainnya.

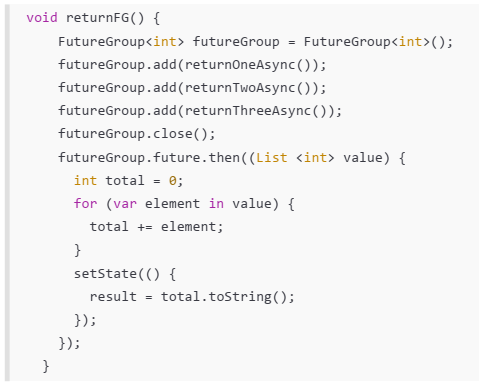
Ketika semua code async paralel selesai dieksekusi, maka FutureGroup akan return value sebagai sebuah List, sama juga ketika ingin menambahkan operasi paralel dalam bentuk List.

Setelah Anda menyelesaikan praktikum 3, Anda dapat melanjutkan praktikum 4 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 3. |
| --- |

## Langkah 1: Buka file main.dart

Tambahkan method ini ke dalam class \_FuturePageState



## Langkah 2: Edit onPressed()

Anda bisa hapus atau comment kode sebelumnya, kemudian panggil method dari langkah 1 tersebut.



## Langkah 3: Run

Anda akan melihat hasilnya dalam 3 detik berupa angka 6 lebih cepat dibandingkan praktikum sebelumnya menunggu sampai 9 detik.

| Soal 7  * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 7**”. |
| --- |

## Langkah 4: Ganti variabel futureGroup

Anda dapat menggunakan FutureGroup dengan Future.wait seperti kode berikut.

| final futures = Future.wait<int>([  returnOneAsync(),  returnTwoAsync(),  returnThreeAsync(),  ]); |
| --- |

| Soal 8  * Jelaskan maksud perbedaan kode langkah 1 dan 4! |
| --- |

# Praktikum 5: Menangani Respon Error pada Async Code

Duration: 13:00

Ada beberapa teknik untuk melakukan *handle error* pada code async. Pada praktikum ini Anda akan menggunakan 2 cara, yaitu then() callback dan pola async/await.

Setelah Anda menyelesaikan praktikum 4, Anda dapat melanjutkan praktikum 5 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 4. |
| --- |

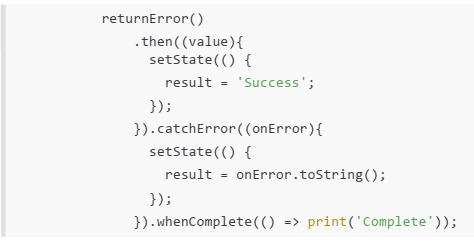
## Langkah 1: Buka file main.dart

Tambahkan method ini ke dalam class \_FuturePageState



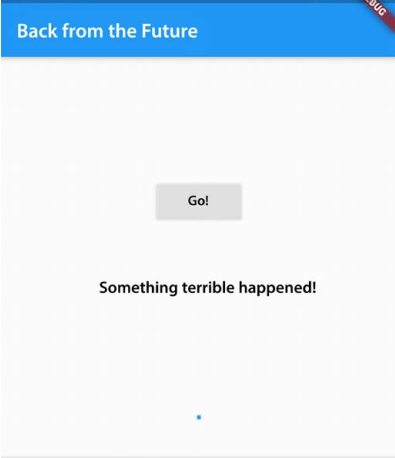
## Langkah 2: ElevatedButton

Ganti dengan kode berikut



## Langkah 3: Run

Lakukan run dan klik tombol **GO!** maka akan menghasilkan seperti gambar berikut.



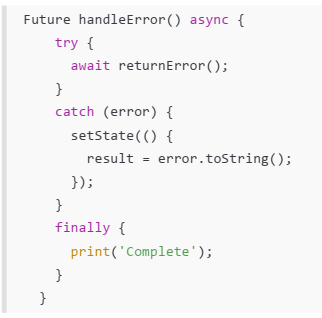
Pada bagian debug console akan melihat teks Complete seperti berikut.



| Soal 9  * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 9**”. |
| --- |

## Langkah 4: Tambah method handleError()

Tambahkan kode ini di dalam class \_FutureStatePage



| Soal 10  * Panggil method handleError() tersebut di ElevatedButton, lalu run. Apa hasilnya? Jelaskan perbedaan kode langkah 1 dan 4! |
| --- |

# Praktikum 6: Menggunakan Future dengan StatefulWidget

Duration: 15:00

Seperti yang Anda telah pelajari, Stateless widget tidak dapat menyimpan informasi (state), StatefulWidget dapat mengelola variabel dan properti dengan method setState(), yang kemudian dapat ditampilkan pada UI. State adalah informasi yang dapat berubah selama life cycle widget itu berlangsung.

Ada **4 method utama** dalam life cycle StatefullWidget:

* initState(): dipanggil sekali ketika state dibangun. Bisa dikatakan ini juga sebagai konstruktor class.
* build(): dipanggil setiap kali ada perubahan state atau UI. Method ini melakukan destroy UI dan membangun ulang dari nol.
* deactive() dan dispose(): digunakan untuk menghapus widget dari tree, pada beberapa kasus dimanfaatkan untuk menutup koneksi ke database atau menyimpan data sebelum berpindah screen.

Setelah Anda menyelesaikan praktikum 5, Anda dapat melanjutkan praktikum 6 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 5. |
| --- |

## Langkah 1: install plugin geolocator

Tambahkan plugin geolocator dengan mengetik perintah berikut di terminal.

| flutter pub add geolocator |
| --- |

## Langkah 2: Tambah permission GPS

Jika Anda menargetkan untuk platform **Android**, maka tambahkan baris kode berikut di file android/app/src/main/androidmanifest.xml

| <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/> |
| --- |

Jika Anda menargetkan untuk platform **iOS**, maka tambahkan kode ini ke file Info.plist

| <key>NSLocationWhenInUseUsageDescription</key>  <string>This app needs to access your location</string> |
| --- |

## Langkah 3: Buat file geolocation.dart

Tambahkan file baru ini di folder lib project Anda.

## Langkah 4: Buat StatefulWidget

Buat class LocationScreen di dalam file geolocation.dart

## Langkah 5: Isi kode geolocation.dart



| Soal 11  * Tambahkan **nama panggilan Anda** pada tiap properti title sebagai identitas pekerjaan Anda. |
| --- |

## Langkah 6: Edit main.dart

Panggil screen baru tersebut di file main Anda seperti berikut.

| home: LocationScreen(), |
| --- |

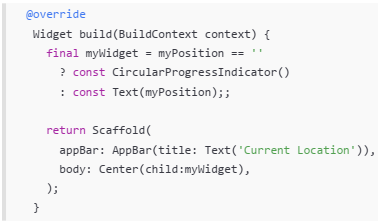
## Langkah 7: Run

Run project Anda di **device** atau **emulator** (**bukan browser**), maka akan tampil seperti berikut ini.



## Langkah 8: Tambahkan animasi loading

Tambahkan widget loading seperti kode berikut. Lalu hot restart, perhatikan perubahannya.



| Soal 12  * Jika Anda tidak melihat animasi loading tampil, kemungkinan itu berjalan sangat cepat. Tambahkan delay pada method getPosition() dengan kode await Future.delayed(const Duration(seconds: 3)); * Apakah Anda mendapatkan koordinat GPS ketika run di browser? Mengapa demikian? * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 12**”. |
| --- |

# Praktikum 7: Manajemen Future dengan FutureBuilder

Duration: 15:00

Pola ketika menerima beberapa data secara async dan melakukan update pada UI sebenarnya itu tergantung pada ketersediaan data. Secara umum fakta di Flutter, ada sebuah widget yang membantu Anda untuk memudahkan manajemen future yaitu widget FutureBuilder.

Anda dapat menggunakan FutureBuilder untuk manajemen future bersamaan dengan update UI ketika ada update Future. FutureBuilder memiliki status future sendiri, sehingga Anda dapat mengabaikan penggunaan setState, Flutter akan membangun ulang bagian UI ketika update itu dibutuhkan.

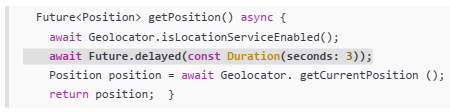
Untuk lebih memahami widget FutureBuilder, mari kita coba dengan praktikum ini.

Setelah Anda menyelesaikan praktikum 6, Anda dapat melanjutkan praktikum 7 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 6. |
| --- |

## Langkah 1: Modifikasi method getPosition()

Buka file geolocation.dart kemudian ganti isi method dengan kode ini.



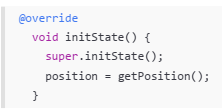
## Langkah 2: Tambah variabel

Tambah variabel ini di class \_LocationScreenState



## Langkah 3: Tambah initState()

Tambah method ini dan set variabel position



## Langkah 4: Edit method build()

Ketik kode berikut dan sesuaikan. Kode lama bisa Anda comment atau hapus.



| Soal 13  * Apakah ada perbedaan UI dengan praktikum sebelumnya? Mengapa demikian? * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 13**”. * Seperti yang Anda lihat, menggunakan FutureBuilder lebih efisien, clean, dan reactive dengan Future bersama UI. |
| --- |

## Langkah 5: Tambah handling error

Tambahkan kode berikut untuk menangani ketika terjadi error. Kemudian hot restart.

| else if (snapshot.connectionState == ConnectionState.done) {  if (snapshot.hasError) {  return Text('Something terrible happened!');  }  return Text(snapshot.data.toString());  } |
| --- |

| Soal 14  * Apakah ada perbedaan UI dengan langkah sebelumnya? Mengapa demikian? * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 14**”. |
| --- |

# Praktikum 8: Navigation route dengan Future Function

Duration: 15:00

Praktikum kali ini Anda akan melihat manfaat Future untuk Navigator dalam transformasi Route menjadi sebuah function async. Anda akan melakukan push screen baru dan fungsi await menunggu data untuk melakukan update warna background pada screen.

Setelah Anda menyelesaikan praktikum 7, Anda dapat melanjutkan praktikum 8 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 7. |
| --- |

## Langkah 1: Buat file baru navigation\_first.dart

Buatlah file baru ini di project lib Anda.

## Langkah 2: Isi kode navigation\_first.dart



| Soal 15  * Tambahkan **nama panggilan Anda** pada tiap properti title sebagai identitas pekerjaan Anda. * Silakan ganti dengan warna tema favorit Anda. |
| --- |

## Langkah 3: Tambah method di class \_NavigationFirstState

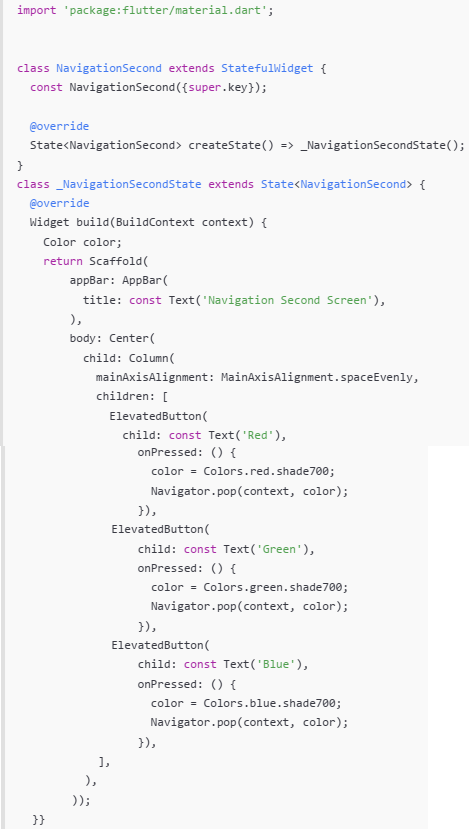
Tambahkan method ini.

| Future \_navigateAndGetColor(BuildContext context) async {  color = await Navigator.push(context,  MaterialPageRoute(builder: (context) => const NavigationSecond()),) ?? Colors.blue;  setState(() {});  });  } |
| --- |

## Langkah 4: Buat file baru navigation\_second.dart

Buat file baru ini di project lib Anda. Silakan jika ingin mengelompokkan view menjadi satu folder dan sesuaikan impor yang dibutuhkan.

## Langkah 5: Buat class NavigationSecond dengan StatefulWidget



## Langkah 6: Edit main.dart

Lakukan edit properti home.

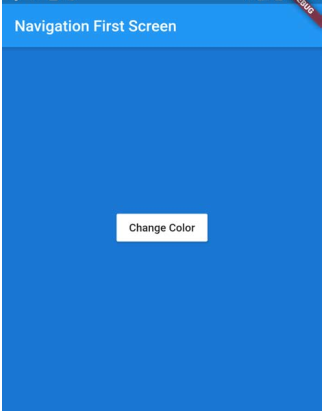
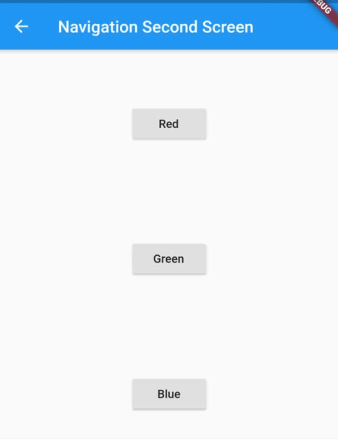
| home: const NavigationFirst(), |
| --- |

## Langkah 8: Run

Lakukan run, jika terjadi error silakan diperbaiki.

| Soal 16  * Cobalah klik setiap button, apa yang terjadi ? Mengapa demikian ? * Gantilah 3 warna pada langkah 5 dengan warna favorit Anda! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 16**”. |
| --- |

Hasilnya akan seperti gambar berikut ini.

# Praktikum 9: Memanfaatkan async/await dengan Widget Dialog

Duration: 15:00

Pada praktikum ini, Anda akan memanfaatkan widget AlertDialog. Anda bisa manfaatkan widget ini misal untuk memilih operasi Save, Delete, Accept, dan sebagainya.

Setelah Anda menyelesaikan praktikum 8, Anda dapat melanjutkan praktikum 9 ini. Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda. Jawablah di laporan praktikum Anda pada setiap soal yang ada di beberapa langkah praktikum ini.

| **Perhatian:** Diasumsikan Anda telah berhasil menyelesaikan Praktikum 8. |
| --- |

## Langkah 1: Buat file baru navigation\_dialog.dart

Buat file dart baru di folder lib project Anda.

## Langkah 2: Isi kode navigation\_dialog.dart



## Langkah 3: Tambah method async



## Langkah 4: Panggil method di ElevatedButton



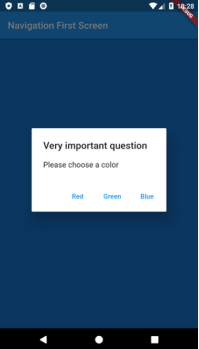
## Langkah 5: Edit main.dart

Ubah properti home



## Langkah 6: Run

Coba ganti warna background dengan widget dialog tersebut. Jika terjadi error, silakan diperbaiki. Jika berhasil, akan tampil seperti gambar berikut.



| Soal 17  * Cobalah klik setiap button, apa yang terjadi ? Mengapa demikian ? * Gantilah 3 warna pada langkah 3 dengan warna favorit Anda! * Capture hasil praktikum Anda berupa GIF dan lampirkan di README. Lalu lakukan commit dengan pesan “**W12: Soal 17**”. |
| --- |

# Selamat!

Duration: 0:00

Selamat Anda telah menyelesaikan Codelab ini. Anda telah mempelajari terkait pemrograman async dan contoh penggunaannya.

Pada codelab berikutnya, Anda akan mempelajari tentang Pemrograman *Asynchronous* lanjutan dengan *streams* dan *state BLoC*.

Jangan sungkan jika Anda menemukan kesalahan pada codelab ini untuk merevisi atau sekedar melaporkan issue melalui tautan di pojok kiri bawah ([Report a mistake](https://github.com/jti-polinema/flutter-codelab/issues)).

## Apa selanjutnya?

Silakan cek beberapa sumber belajar lainnya…

* <https://dart.dev/codelabs/async-await>
* <https://api.flutter.dev/flutter/dart-async/Completer-class.html>
* [Video tentang Async, Await, Future (Bahasa Indonesia)](https://www.youtube.com/watch?v=AJQuw-fnooA)
* <https://dart.dev/guides/libraries/futures-error-handling>
* <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>
* <https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html>
* <https://api.flutter.dev/flutter/material/AlertDialog-class.html>

## Referensi

* [Buku “Flutter Cookbook, 100+ step-by-step recipes for building cross-platform, professional grade apps with Flutter 3.10.x and Dart 3.x - Second Edition” oleh Simone Alessandria (2023)](https://drive.google.com/file/d/1W9uSLC9wlf9fFudmCWFHTx9dan5SXII0/view?usp=sharing)
* [Flutter for Beginners Second Edition](https://drive.google.com/file/d/1hJbctx-_O3B6kUeG70bJD5xfbnsWwBXY/view?usp=sharing)
* <https://www.youtube.com/flutterdev>
* <https://github.com/PacktPublishing/Flutter-for-Beginners-Second-Edition>