#33 | Manipulasi Widget

| **Summary** | Mahasiswa mampu menggunakan fungsi transform widget, Matrix4, dan jenis-jenis transformasi |
| --- | --- |
| **URL** | 33-manipulasi-widget |
| **Category** | Mobile Flutter |
| **Environment** | Flutter, mobile, android, iOS, web, desktop |
| **Status** | Published |
| **Feedback Link** | https://github.com/jti-polinema/flutter-codelab/issues |
| **Author** | Tim Pengajar Mobile TSA 2022 |
| **Author LDAP** | hbb-polinema |
| **Analytics Account** | G-H7507PFEJB |

[Manipulasi Widget](#_ok7k5uux6)

[Tujuan Praktikum](#_21yzqg98x7h6)

[Tujuan Visual](#_hs6hf8vnv2d6)

[Sumber Daya yang Dibutuhkan](#_dusjnkkj26ob)

[Daftar Perintah](#_6mgk7cf2soh8)

[Bantuan](#_f4c7kiz5py9x)

[Pengetahuan yang Anda harus Miliki](#_319h1e1sjzdz)

[Transformasi widget dengan Transform class](#_t0tqoph2ohlh)

[Transform widget](#_hsinehi024rc)

[Memahami Matrix4 class](#_qioeyvn5p8uv)

[Explorasi jenis-jenis transformasi](#_rvmlbkobso2d)

[Rotate transformation](#_rt47bi9tygha)

[Scale transformation](#_nx2odt3o6ew8)

[Translate transformation](#_n37g6pyiqpff)

[Composed transformations](#_8o9jnlsqx0xf)

[Praktikum Manipulasi Widget](#_vmpz7ap0erhd)

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

[Langkah 2: Impor dart:math](#_g8csgggwssw)

[Langkah 3: Tambahkan kode Transform](#_2q85gi9dhaiq)

[Tugas Praktikum](#_dz35nqtk0wjb)

[Selamat!](#_s22qmcj4zu61)

[Apa selanjutnya?](#_9mqifffit2ew)

[Referensi](#_an95hoqxrlhg)

# Manipulasi Widget



**Terakhir diperbarui:** 18 September 2022

**Penulis:** Tim Pengajar Mobile Flutter

Pada codelab ini, Anda akan mempelajari tentang fungsi transform widget, Matrix4, dan jenis-jenis transformasi.

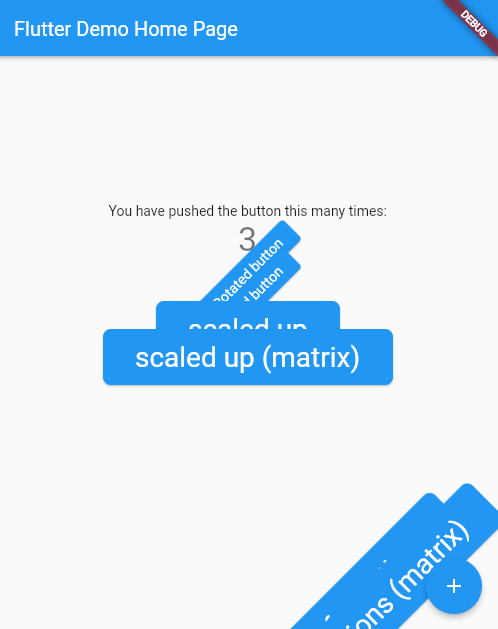
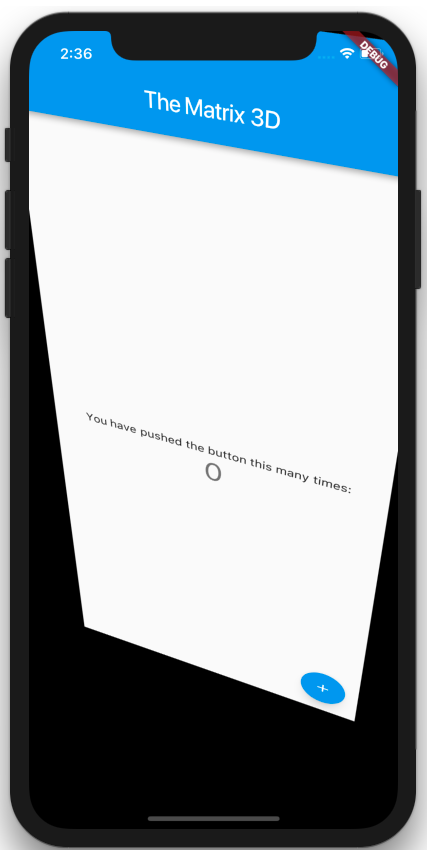
## Tujuan Praktikum

Setelah menyelesaikan codelab ini Anda akan mampu untuk:

* menggunakan fungsi transform widget, Matrix4, dan jenis-jenis transformasi

| **Catatan:** Materi ini diadaptasi dari Buku Flutter for Beginners - second edition dan Buku Ajar Pemrograman Mobile Flutter JTI Polinema serta sumber referensi lainnya. |
| --- |

## Tujuan Visual



## 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
* Chrome Browser
* Perangkat mobile (Android/iOS)

## Daftar Perintah

Tida ada.

## Bantuan

Tidak ada.

## Pengetahuan yang Anda harus Miliki

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

# Transformasi widget dengan Transform class

Duration: 45:00

Kita telah melihat banyak widget di pertemuan sebelumnya, tetapi terkadang kita mungkin perlu mengubah tampilan widget untuk meningkatkan UX. Melakukan respon terhadap input user atau untuk membuat efek keren dalam tata letak, kita mungkin perlu memindahkan widget di sekitar layar, mengubah ukurannya, atau bahkan sedikit mengubahnya.

Jika Anda pernah mencoba dalam bahasa pemrograman native, Anda mungkin merasa kesulitan. Flutter, seperti yang disebutkan sebelumnya, sangat fokus pada desain UI dan bertujuan untuk membuat hidup developer lebih mudah dengan menyederhanakan apa yang bisa dengan mudah pada area yang rumit.

Di bagian ini, pertama-tama kita akan melihat widget Transform karena ini adalah widget yang sangat berguna saat Anda melakukan manipulasi widget. Kemudian akan mempelajari lebih dalam widget untuk melihat jenis-jenis manipulasi yang mungkin bisa dilakukan.

## Transform widget

Widget Transform adalah salah satu contoh terbaik dari kekuatan dan konsistensi framework Flutter. Ini adalah widget tujuan tunggal yang hanya menerapkan transformasi grafis ke anaknya, dan tidak lebih. Memfokuskan widget pada satu tujuan adalah dasar untuk struktur tata letak yang lebih baik, dan Flutter melakukannya dengan sangat baik.

Widget Transform, seperti namanya, melakukan satu tugas—itu mengubah turunan dasarnya. Meskipun tugasnya sangat kompleks, ia menyembunyikan sebagian besar kerumitan ini dari pengembang. Mari kita lihat kodenya sebagai berikut:

| const Transform({  Key,  required Matrix4 transform,  Offset origin, AlignmentGeometry alignment,  bool transformHitTests: true,  Widget child  }) |
| --- |

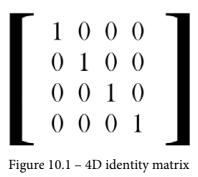
Seperti yang Anda lihat, selain properti Key, widget ini tidak memerlukan banyak argumen untuk melakukan tugasnya. Mari kita lihat penjelasan parameter masing-masing sebagai berikut:

* transform: Ini adalah satu-satunya properti yang diperlukan dan digunakan untuk menjelaskan transformasi yang akan diterapkan ke widget anak. Jenisnya adalah instance Matrix4, matriks **empat dimensi (4D)** yang menggambarkan transformasi secara matematis. Kita akan melihat lebih detail di bagian berikutnya, yaitu pada bagian memahami kelas Matrix4.
* origin: Ini adalah asal dari sistem koordinat yang menerapkan matriks transformasi. Properti asal ditentukan oleh jenis Offset, yang mewakili, dalam hal ini, titik (x,y) dalam sistem Cartesian yang relatif terhadap sudut kiri atas widget render.
* alignment: Seperti halnya origin, dapat digunakan untuk memanipulasi posisi matriks transformasi yang diterapkan. Kita dapat menggunakan ini untuk menentukan asal dengan cara yang lebih fleksibel, karena asal mengharuskan kita untuk menggunakan nilai posisi nyata. Tidak ada yang mencegah Anda menggunakan Origin dan alignment secara bersamaan.
* transformHitTests: Ini menentukan apakah tes klik (yaitu, ketukan) dievaluasi dalam versi widget yang diubah.
* child: Ini adalah widget anak tempat transformasi akan diterapkan.

Transformasi Matrix4 sangat penting untuk kelas Transform, jadi mari kita lihat lebih detail.

## Memahami Matrix4 class

Di Flutter, transformasi direpresentasikan sebagai matriks 4D. Meskipun terdengar sangat menakutkan, matriks 4D hanyalah matriks yang memiliki empat baris dan empat kolom, seperti yang ditunjukkan di sini:



Nilai matriks yang ditampilkan adalah **matriks identitas**. Ini adalah nilai khusus karena secara efektif mengatakan untuk tidak melakukan perubahan dalam transformasi. Saat nilai dalam matriks diubah, maka widget ditransformasikan dengan cara yang berbeda.

Seringkali, kita tidak perlu mengetahui nilai spesifik dari matriks untuk melakukan transformasi.

Selain metode seperti penambahan atau perkalian matriks, kelas Matrix4 berisi metode yang membantu konstruksi dan manipulasi transformasi geometris. Beberapa di antaranya tercantum di sini:

* Rotasi: rotateX(), rotateY(), dan rotateZ() adalah beberapa contoh metode yang memutar matriks melalui sumbu tertentu.
* Skala: scale(), dengan beberapa varian, digunakan untuk menerapkan skala pada matriks menggunakan nilai ganda dari sumbu yang sesuai (x, y, dan z) atau melalui representasi vektor dengan kelas Vector3 dan Vector4.
* Translasi: Sama seperti sebelumnya, kita dapat melakukan translasi matriks menggunakan metode translate() dengan nilai x, y, atau z tertentu dan instance Vector3 dan Vector4.
* Skew: Ini digunakan untuk memiringkan matriks di sekitar sumbu x dengan skewX() atau sumbu y dengan skewY().

Mari kita lihat bagaimana kita dapat menggunakan kelas Matrix4 untuk memberlakukan berbagai jenis transformasi.

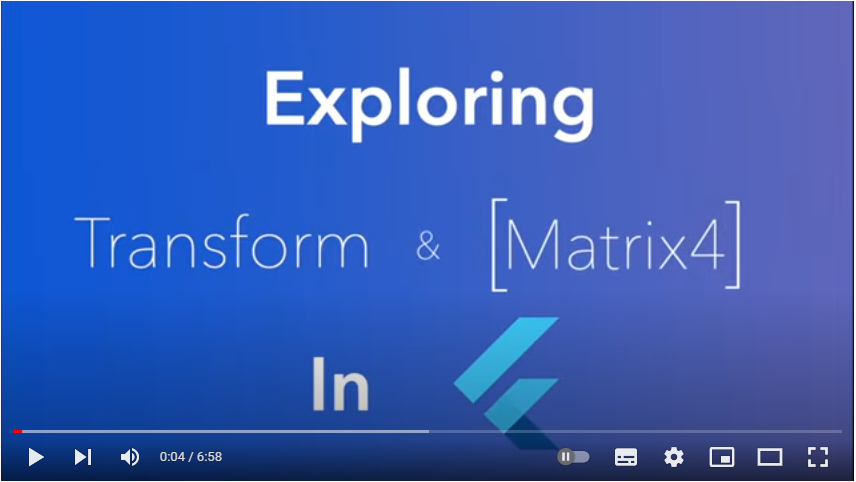
## Explorasi jenis-jenis transformasi

Kelas Transform menyediakan fasilitas kepada pengembang melalui konstruktor pabriknya.

Ada banyak dari mereka untuk setiap kemungkinan transformasi, sehingga sangat mudah untuk menerapkan transformasi ke widget tanpa pengetahuan yang lebih dalam tentang perhitungan geometris. Terdiri dari:

* Transform.rotate(): Membuat widget Transform yang memutar anaknya di sekitar pusatnya.
* Transform.scale(): Membuat widget Transform yang menskalakan anaknya secara seragam.
* Transform.translate(): Membuat widget Transform yang menerjemahkan anaknya dengan offset x,y.

Mari kita lihat masing-masing jenis transformasi secara lebih rinci.



### Rotate transformation

Transformasi putar muncul dalam situasi di mana kita hanya ingin membuat widget kita berputar. Dengan menggunakan konstruktor Transform.rotate(), kita bisa mendapatkan widget yang diputar.

Itu tidak terlalu berbeda dari konstruktor Transform default. Perbedaannya tercantum di sini:

* Tidak ada properti transform: Kita gunakan varian rotate() karena ingin menerapkan rotasi, jadi tidak perlu menentukan seluruh matriks untuk ini. Kita hanya gunakan properti angle sebagai gantinya.
* Angle: Ini menentukan rotasi yang diinginkan dalam radian searah jarum jam.
* Origin: Secara default, rotasi diterapkan relatif terhadap bagian tengah anak. Namun, kita dapat menggunakan properti origin untuk memanipulasi rotasi, seolah-olah kita menerjemahkan pusat widget dengan offset origin, menyebabkan rotasi menjadi relatif ke titik lain jika kita mau.

Kita dapat menggunakannya seperti ini:

| Transform.rotate(  angle: -45 \* (math.pi / 180.0),  child: ElevatedButton(  child: Text("Rotated button"),  onPressed: () {},  ),  ); |
| --- |

Pada contoh ini, kita spesifikkan sudut radian yang digunakan (misalnya sudut 315° searah jarum jam sama dengan sudut -45° berlawanan), dan widget child akan berotasi, dalam contoh ini yaitu widget ELevatedButton.

Hasil yang sama persis akan diperoleh ketika menggunakan widget Transform dan Matrix4, seperti yang diilustrasikan pada kode berikut:

| Transform(  transform: Matrix4.rotationZ(-45 \* (pi / 180.0)),  alignment: Alignment.center,  child: ElevatedButton (  child: Text("Rotated button"),  onPressed: () {},  ),  ); |
| --- |

Argumen yang perlu kita berikan untuk mendapatkan hasil yang sama adalah properti transform dengan rotasi melalui sumbu z, dan properti alignment dari transformasi yang menentukan pusat child widget.

### Scale transformation

Transformasi skala muncul dalam situasi di mana kita hanya ingin membuat widget kita mengubah ukurannya, baik dengan menambah atau mengurangi skalanya. Sama seperti konstruktor pabrik rotate() , varian ini tidak terlalu berbeda dari yang default. Berikut adalah beberapa rincian lebih lanjut mengenai hal ini:

* Tidak ada properti transform: Di sini, sekali lagi, kami menggunakan properti skala alih-alih seluruh matriks transformasi.
* Skala: Inilah yang kami gunakan untuk menentukan skala yang diinginkan dalam format ganda, 1.0 sebagai ukuran asli widget. Ini mewakili skalar yang akan diterapkan pada setiap sumbu x dan y.
* Alignment: Secara default, skala diterapkan relatif terhadap pusat anak. Di sini, kita dapat menggunakan properti keselarasan untuk mengubah asal skala. Sekali lagi, kita dapat menggabungkan properti alignment dan origin untuk mendapatkan hasil yang diinginkan.

Misalnya, untuk memperbesar widget, kita dapat menjalankan kode berikut:

| Transform.scale(  scale: 2.0,  child: ElevatedButton(  child: Text("scaled up"),  onPressed: () {},  ),  ); |
| --- |

Di sini, kami telah menetapkan properti skala 2.0, yang menggandakan ukuran widget anak, dan sekali lagi menetapkan bahwa widget anak adalah ElevatedButton. Jelas, Anda bisa mengatur ukuran ElevatedButton untuk menghindari transformasi.

Dan untuk mendapatkan hasil yang sama dengan menggunakan konstruktor Transform default, kami menggunakan kode berikut:

| Transform(  transform: Matrix4.identity()..scale(2.0, 2.0),  alignment: Alignment.center,  child: ElevatedButton(  child: Text("scaled up"),  onPressed: () {},  ),  ); |
| --- |

Dengan cara yang sangat mirip dengan rotasi, kita harus menentukan asal transformasi dengan properti keselarasan dan instance Matrix4 yang menjelaskan transformasi skala.

### Translate transformation

Transformasi terjemahan kemungkinan akan muncul dalam animasi, seperti yang dijelaskan nanti dalam bab di bagian Menggunakan animasi.

Di sini, kami memiliki lebih sedikit properti dibandingkan dengan transformasi sebelumnya.

Perbedaannya tercantum sebagai berikut:

* Tidak adanya properti transformasi dan penyelarasan: Transformasi akan diterapkan oleh nilai offset, jadi kita tidak memerlukan matriks transformasi.
* Offset: Kali ini, offset hanya menentukan terjemahan yang akan diterapkan pada widget anak; ini berbeda dengan transformasi sebelumnya, di mana hal itu mempengaruhi titik asal dari transformasi yang diterapkan.

Dengan menggunakan konstruktor Transform.translate(), kita memindahkan widget di sekitar layar, dengan menambahkan widget Transform sebagai induk dari widget yang ingin kita pindahkan, seperti yang diilustrasikan dalam cuplikan kode berikut:

| Transform.translate(  offset: Offset(100, 300),  child: ElevatedButton(  child: Text("translated to bottom"),  onPressed: () {},  ),  ); |
| --- |

Konstruktor default juga dapat digunakan dengan Matrix4 yang menentukan terjemahan, sebagai berikut:

| Transform(  transform: Matrix4.translationValues(100, 300, 0),  child: ElevatedButton(  child: Text("translated to bottom"),  onPressed: () {},  ),  ); |
| --- |

Kita hanya perlu menentukan properti transform dengan instance Matrix4 yang menjelaskan terjemahannya.

### Composed transformations

Kita dapat—dan kemungkinan besar akan—menggabungkan sejumlah transformasi yang terlihat sebelumnya untuk mencapai efek unik, seperti memutar pada saat yang sama saat kita memindahkan dan menskalakan widget.

Menyusun transformasi dapat dilakukan dengan dua cara, sebagai berikut:

* Menggunakan konstruktor widget Transform default dan menghasilkan transformasi yang kita inginkan menggunakan metode yang disediakan Matrix4 untuk menyusunnya.
* Menggunakan beberapa widget Transform secara bersarang dengan konstruktor pabrik rotate(), scale(), dan translate(), untuk mencapai efek yang sama.

Untuk lebih jelasnya, mari kita lihat bagaimana kita akan menyarangkan beberapa widget Transform, sebagai berikut:

| Transform.translate(  offset: Offset(70, 200),  child: Transform.rotate(  angle: -45 \* (math.pi / 180.0),  child: Transform.scale(  scale: 2.0,  child: ElevatedButton(  child: Text("multiple transformations"),  onPressed: () {},  ),  ),  ),  ); |
| --- |

Seperti yang Anda lihat, kami menambahkan widget Transform sebagai turunan ke widget Transform lainnya, yang menyusun transformasi. Meskipun lebih sederhana untuk dibaca, metode ini memiliki kelemahan:

kami menambahkan lebih banyak widget daripada yang dibutuhkan ke pohon widget.

Saat kita menambahkan beberapa transformasi ke widget secara bersamaan, kita harus memperhatikan urutan transformasi. Bereksperimen sendiri: menukar posisi widget Transform akan menghasilkan hasil yang berbeda.

Sebagai alternatif, kita dapat menggunakan konstruktor Transform default dengan transformasi tersusun dengan objek Matrix4, sebagai berikut:

| Transform(  alignment: Alignment.center,  transform: Matrix4.translationValues(70, 200, 0)  ..rotateZ(-45 \* (math.pi / 180.0))  ..scale(2.0, 2.0),  child: ElevatedButton(  child: Text("multiple transformations"),  onPressed: () {},  ),  ); |
| --- |

Sama seperti sebelumnya, kami menentukan perataan transformasi sebagai pusat widget anak dan kemudian instance Matrix4 untuk menggambarkannya. Seperti yang Anda lihat, ini sangat mirip dengan beberapa versi widget Transform tetapi tanpa widget bersarang, menyebabkan pohon widget yang lebih dalam.

Sekarang kita telah menjelajahi cara memanipulasi widget dengan cara statis, mari kita lihat membuat animasi pada pertemuan berikutnya untuk memungkinkan widget berpindah dari satu keadaan ke keadaan lain dengan cara yang lebih halus.

# Praktikum Manipulasi Widget

Duration: 30:00

Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda.

| **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 **flutter\_transform**. Lalu jadikan repository di GitHub Anda dengan nama **flutter\_transform.**

## Langkah 2: Impor dart:math

Tambahkan library dart:math pada file main.dart

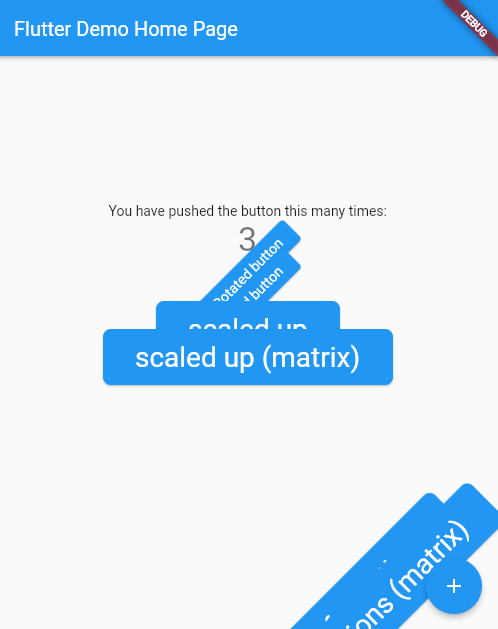
| import 'dart:math'; |
| --- |

## Langkah 3: Tambahkan kode Transform

Tetap di file main.dart di class \_MyHomePageState pada bagian widget build, letakkan kode berikut di dalam children: <Widget> setelah teks $\_counter.

| Transform.rotate(  angle: -45 \* (pi / 180.0),  child: ElevatedButton(  child: const Text("Rotated button"),  onPressed: () {},  ),  ),  Transform(  transform: Matrix4.rotationZ(-45 \* (pi / 180.0)),  alignment: Alignment.center,  child: ElevatedButton(  child: const Text("Rotated button"),  onPressed: () {},  ),  ),  Transform.scale(  scale: 2.0,  child: ElevatedButton(  child: const Text("scaled up"),  onPressed: () {},  ),  ),  Transform(  transform: Matrix4.identity()..scale(2.0, 2.0),  alignment: Alignment.center,  child: ElevatedButton(  child: const Text("scaled up (matrix)"),  onPressed: () {},  ),  ),  Transform.translate(  offset: const Offset(100, 300),  child: ElevatedButton(  child: const Text("translated to bottom"),  onPressed: () {},  ),  ),  Transform(  transform: Matrix4.translationValues(100, 300, 0),  child: ElevatedButton(  child: const Text("translated to bottom (matrix)"),  onPressed: () {},  ),  ),  Transform.translate(  offset: const Offset(70, 200),  child: Transform.rotate(  angle: -45 \* (pi / 180.0),  child: Transform.scale(  scale: 2.0,  child: ElevatedButton(  child: const Text("multiple transformations"),  onPressed: () {},  ),  ),  ),  ),  Transform(  alignment: Alignment.center,  transform: Matrix4.translationValues(70, 200, 0)  ..rotateZ(-45 \* (pi / 180.0))  ..scale(2.0, 2.0),  child: ElevatedButton(  child: const Text("multiple transformations (matrix)"),  onPressed: () {},  ),  ) |
| --- |

**Run** aplikasi tersebut dengan tekan **F5**, maka hasilnya akan seperti berikut.



# Tugas Praktikum

Duration: 90:00

1. Selesaikan Praktikum tersebut, lalu dokumentasikan dan push ke repository Anda berupa screenshot hasil pekerjaan beserta penjelasannya di file README.md!
2. Tetap di project yang sama, buatlah folder dan file dart baru lalu praktikkan kode yang ada di [tautan ini](https://medium.com/flutter/perspective-on-flutter-6f832f4d912e) terkait perspektif di flutter!
3. Tetap di project yang sama, buatlah folder dan file dart baru lalu praktikkan kode yang ada di [tautan ini](https://medium.com/flutter-community/make-3d-flip-animation-in-flutter-16c006bb3798) untuk membuat flip animasi 3D!
4. Kumpulkan laporan praktikum Anda berupa link repository GitHub ke LMS!

# Selamat!

Duration: 0:00

Selamat Anda telah menyelesaikan Codelab ini. Anda telah mempelajari terkait manipulasi widget dengan class Transform.

Pada codelab berikutnya, Anda akan mempelajari tentang Animasi.

## Apa selanjutnya?

Silakan cek beberapa sumber belajar lainnya…

* [FlutLab is a modern Flutter online IDE](https://flutlab.io/)
* [Flutter System Architecture](https://docs.google.com/presentation/d/1cw7A4HbvM_Abv320rVgPVGiUP2msVs7tfGbkgdrTy0I/edit#slide=id.p)
* [Perspective on Flutter](https://medium.com/flutter/perspective-on-flutter-6f832f4d912e)
* [Make 3D flip animation in Flutter](https://medium.com/flutter-community/make-3d-flip-animation-in-flutter-16c006bb3798)
* <https://pub.dev/packages/flip_panel>

## Referensi

* [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>
* Buku Ajar Pemrograman Mobile Dengan Flutter, Jurusan Teknologi Informasi, Politeknik Negeri Malang. 2021.