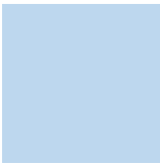


**MODUL**

**PEMROGRAMAN BERBASIS PLATFORM**



PERTEMUAN – 4

**DART**

PROGRAM STUDI INFORMATIKA  
FAKULTAS TEKNOLOGI INDUSTRI  
UNIVERSITAS ATMA JAYA YOGYAKARTA





## Daftar Isi

2		
3		
4	Gambar 1. Logo Bahasa Pemrograman Dart .....	4
5	Gambar 2. Buat project Dart baru .....	5
6	Gambar 3. Memilih Console Application.....	5
7	Gambar 4. Project Hello World .....	6
8	Gambar 5. Arahkan untuk membuka Package hello_world.dart .....	7
9	Gambar 6. Isi Package hello_world.dart .....	8
10	Gambar 7. Sintaks import dengan as .....	8
11	Gambar 8. Sintaks print.....	9
12	Gambar 9. Bentuk lain dari penggunaan sintaks <code>{...}</code> .....	9
13	Gambar 10. Print dengan variabel .....	10
14	Gambar 11. Beberapa contoh deklarasi variabel.....	10
15	Gambar 12. Deklarasi variabel menggunakan var .....	11
16	Gambar 13. Deklarasi variabel dynamic.....	11
17	Gambar 14. Struktur class beberapa tipe data di Dart.....	12
18	Gambar 15. code if else .....	14
19	Gambar 16. Ternary Operator .....	15
20	Gambar 17. Sintaks ternary operator .....	15
21	Gambar 18. Switch case .....	16
22	Gambar 19. For loop.....	17
23	Gambar 20. For each.....	18
24	Gambar 21. While dan do-while loop.....	18
25	Gambar 22. Contoh list, set dan map .....	20
26	Gambar 23. Named Parameters .....	21
27	Gambar 24. Required Parameters.....	22
28	Gambar 25. Default value parameters .....	22
29	Gambar 26. Function as Object & Anonymous Function.....	23
30	Gambar 27. Arrow Function.....	23
31	Gambar 28. Contoh class User .....	25
32	Gambar 29. Initializing formal parameters .....	26
33	Gambar 30. Named Constructor .....	27
34	Gambar 31. Getter dan setter .....	28
35	Gambar 32. Contoh penerapan Exception Handling.....	30
36	Gambar 33. Pengembangan dari Gambar 32.....	32
37	Gambar 34. Lanjutan Gambar 33.....	33
38	Gambar 35. Hasil keluaran Gambar 33.....	33
39	Gambar 36. Class LoginRepository dengan Future.delayed .....	34



40	Gambar 37. Hasil keluaran dari Gambar 36 .....	34
41	Gambar 38. Alur kasar Gambar 36 .....	35
42	Gambar 39. Class LoginRepository dengan penerapan future, async, dan await..	36
43	Gambar 40. Hasil keluaran dari Gambar 39 .....	36
44	Gambar 41. Peringatan error ketika memberikan name dan password yang salah	
45	.....	37
46	Gambar 42. Screenshot aplikasi crash .....	37
47	Gambar 43. Penerapan Null aware Operator .....	39
48	Gambar 44. Penerapan Null-coalescing operator .....	39
49	Gambar 45. File gd_modul_dart_xxxx.dart .....	40
50	Gambar 46. Class User .....	41
51	Gambar 47. Class Repository .....	41
52	Gambar 48. Class LoginController.....	42
53	Gambar 49. Isi main program login console app .....	43
54	Gambar 50. Pesan error pada stdin.write dan stdin.readLineSync.....	43
55	Gambar 51. Tampilan Solution .....	44
56	Gambar 52. Hasil setelah diimport.....	44
57	Gambar 53. Arahkan untuk menambahkan konfigurasi pada run program .....	45
58	Gambar 54. Isi dari file lauch.json.....	45
59	Gambar 55. Hasil Guided 1 tampilan awal .....	46
60	Gambar 56. Hasil Guided 1 tampilan berhasil login .....	46
61	Gambar 57. Hasil Guided 1 Tampilan Username dan Password kosong .....	47
62	Gambar 58. Hasil Guided 1 Tampilan Username atau Password salah .....	47

63



## TUJUAN

**Setelah menyelesaikan modul ini, praktikan diharapkan mampu :**

1. Memahami Bahasa Pemrograman Dart
2. Membuat aplikasi console menggunakan Bahasa Pemrograman Dart

## DASAR TEORI

### A. Dart

Dart merupakan bahasa pemrograman berbasis objek yang dikembangkan oleh google untuk pengembangan aplikasi *client-side*. Dart bertujuan untuk mendukung pengembangan aplikasi *multi-platform* secara produktif. Untuk mencapai tujuan itu, Dart dilengkapi dengan pemrosesan JIT (*just-in-time*) yang mempermudah proses pengembangan dan kompilasi AOT (*ahead-of-time*) yang meningkatkan performa aplikasi. Secara sintaks, Dart memiliki kesamaan dengan berbagai bahasa pemrograman seperti C#, Java, dan Javascript.

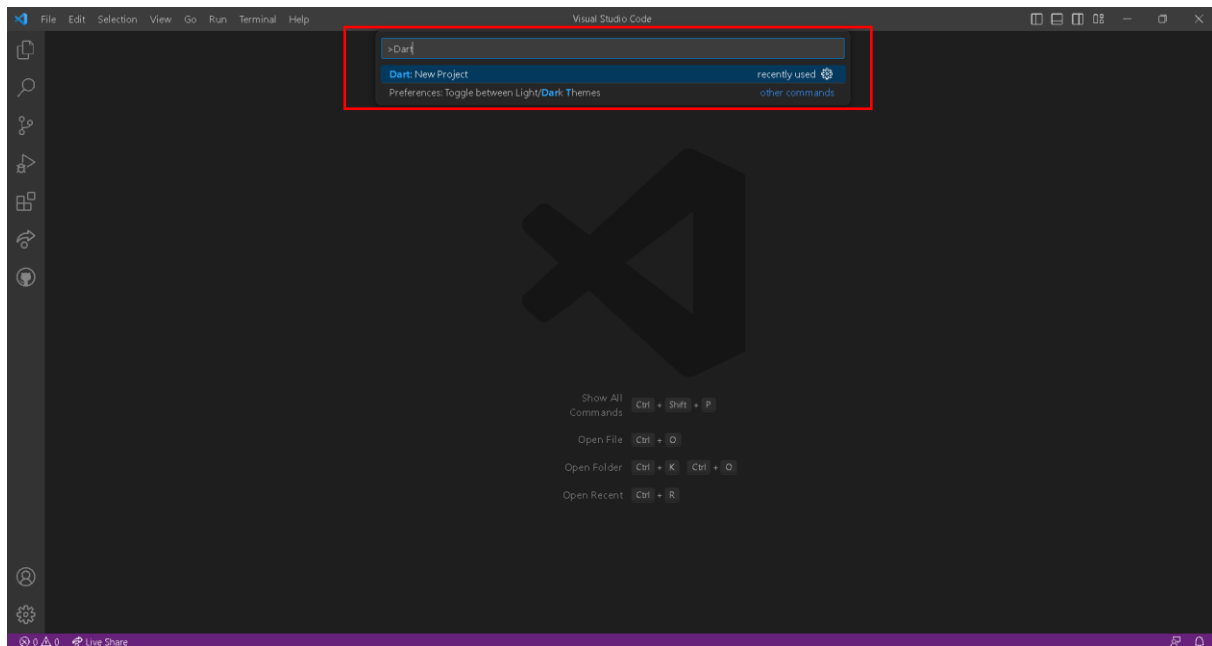


Gambar 1. Logo Bahasa Pemrograman Dart

### B. Hello World

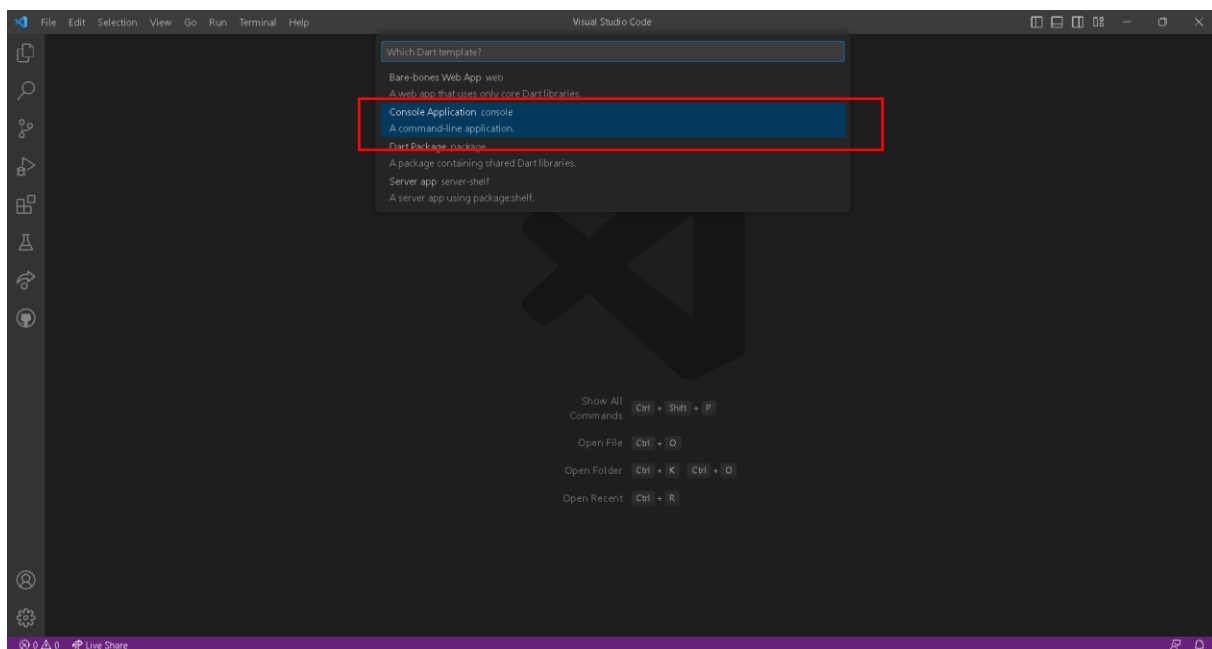
Untuk membuat project baru Dart (Bukan Flutter) di VSCode ikuti Langkah - langkah berikut ini:

1. Tekan **ctrl + shift + p** untuk Windows atau **cmd + shift + p** untuk Mac
2. Ketik **Dart** dan pilih **Dart:New Project**.



Gambar 2. Buat project Dart baru

### 3. Pilih **Console Application**.

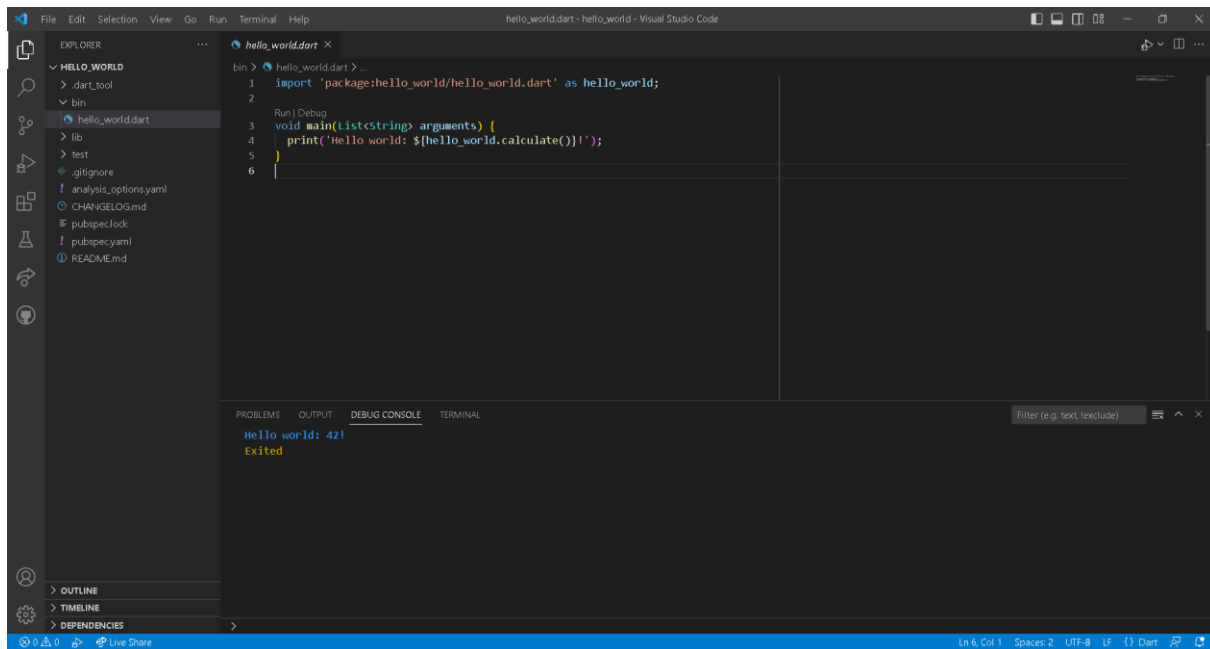


Gambar 3. Memilih Console Application

4. Kemudian tentukan dimana project kalian akan dibuat dan beri nama project. Perhatikan nama project hanya boleh menggunakan format penamaan **Snake Case**.

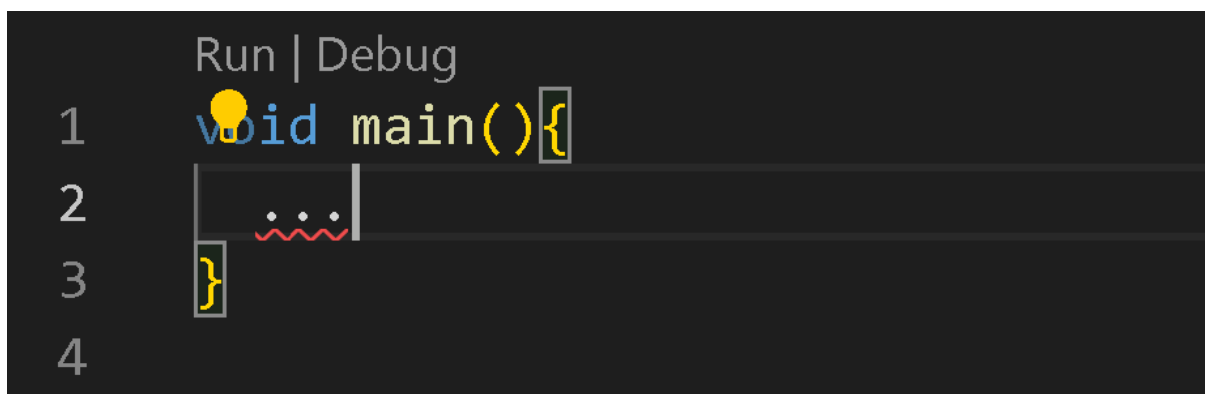


5. Kemudian kalian akan mendapatkan hasil seperti ini. Tekan **Ctrl + F5** atau **Fn + F5** atau **F5** saja sesuai konfigurasi laptop masing-masing untuk menjalankan program Dart.



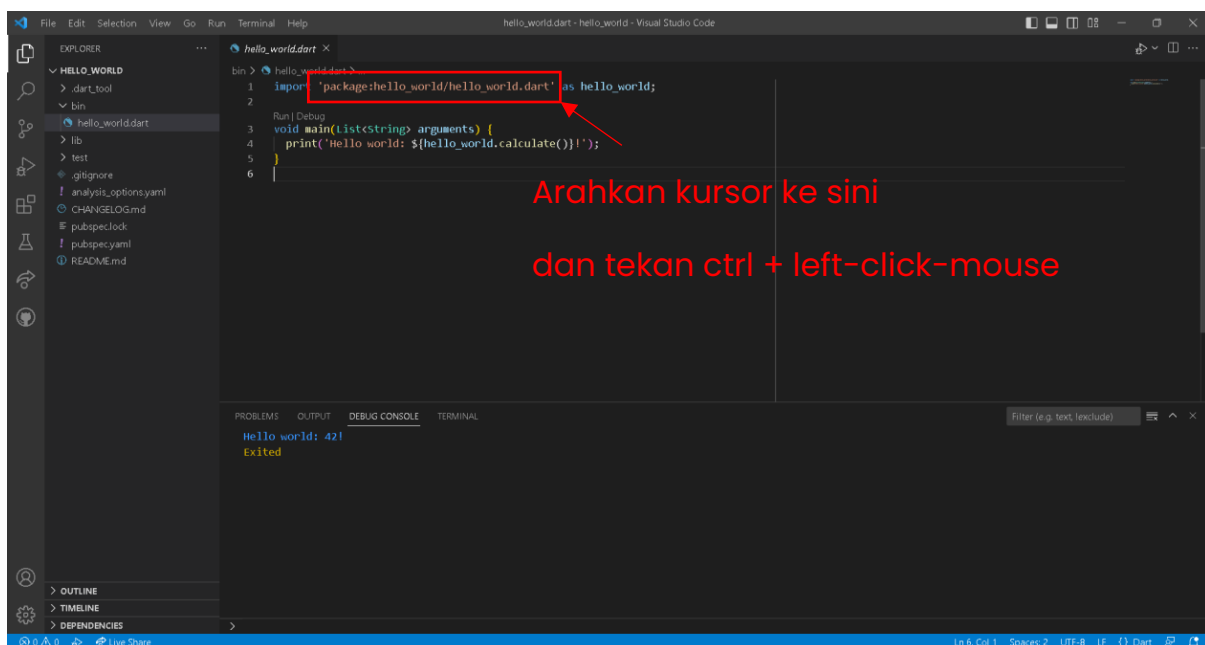
Gambar 4. Project Hello World

Seperti pada beberapa bahasa pemrograman, Dart memiliki fungsi utama yaitu **main()**, dimana eksekusi pertama kali dijalankan. Fungsi **main()** memiliki tipe data balikan **void** dan dapat menerima argument berupa **List<String>**. Akan tetapi dalam modul ini, kita tidak akan menggunakan argument tersebut sehingga kita dapat hanya menuliskannya seperti berikut (Tanpa menuliskan argumentnya).

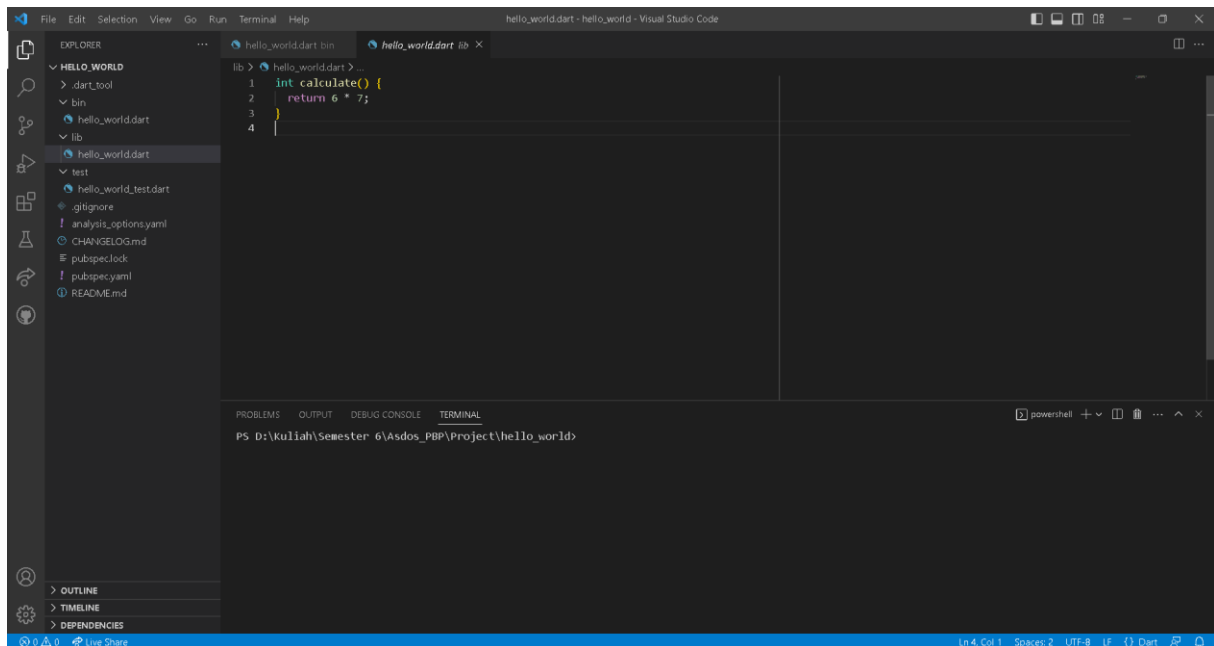


Gambar 5. Modifikasi void main

Mari kita membedah code pada Gambar 4. Pada baris pertama, program mengimpor kode dari **package hello\_world** dan memberikannya alias yaitu **hello\_world** dengan keyword **as**. Untuk melihat apa isi dari `package:hello_world/hello_world.dart` kita dapat menggunakan kombinasi **ctrl + left-click-mouse** sambil mengarahkan kursor ke *package* tersebut. Kita akan dirujuk ke file yang dirujuk.



Gambar 5. Arahan untuk membuka Package hello\_world.dart



Gambar 6. Isi Package hello\_world.dart

Berikut sintaks dari **import** menggunakan **as**

```
import 'somePathPackageOrFile' as aliasName;
```

Gambar 7. Sintaks import dengan as

Kita dapat mengimpor *library* utama, *library* dari paket eksternal atau *file* menggunakan **import** dan memberikan alias menggunakan **as**. Kita juga dapat memilih bagian isi dari library yang ingin kita impor atau kecualikan menggunakan keyword **show** dan **hide**. Baca lebih lengkap [disini](#).

Di dalam `main()` terdapat fungsi **print()** untuk mencetak sesuatu ke *console*. Fungsi `print()` menerima parameter berupa string untuk dicetak ke *console*. Dart tidak membedakan penggunaan tanda petik tunggal atau ganda untuk string.



```
Run | Debug
1 void main(){
2     print('Some String');
3     print("Some String");
4 }
5
```

Gambar 8. Sintaks print

Di dalam `print()` terdapat `hello_world.calculate()` yang merupakan pemanggilan *method* `calculate()` pada `hello_world` yang merupakan *package* yang kita impor di baris pertama. Di sini kita menggunakan sintaks `$ {...}` untuk menampilkan nilai yang dikembalikan atau dihasilkan dari apa pun di dalam tanda kurung kurawal dan dikonversi menjadi string. Hal ini sama dengan

```
Run | Debug
3 void main() {
4     print('Hello world: ' + hello_world.calculate().toString() + '!');
5 }
6
```

Gambar 9. Bentuk lain dari penggunaan sintaks `$ {...}`

Kita dapat pula memanggil variabel di dalam String menggunakan sintaks `$ {...}`. Untuk pemanggilan hanya satu variabel, kita tidak perlu menggunakan tanda kurung kurawal seperti contoh di bawah. Jangan lupa tanda **titik koma (;)** di setiap akhir *statement*.

```
Run | Debug
3  void main() {
4      var someText = 'UAJY';
5      print('Hello World: $someText!');
6  }
7
```

Gambar 10. Print dengan variabel

### C. Variabel

Cara deklarasi variabel di Dart sama dengan bahasa pemrograman lainnya. Diawali dengan tipe data, nama variabel, dan diakhiri dengan inisial value yang disimpan variabel. Variabel di dart hanya boleh menggunakan **lowerCamelCase** (hampir semua penamaan di dart menggunakan *lowerCamelCase*).

```
int varInt = 10;
String varString = 'a String';
num varNum;
```

Gambar 11. Beberapa contoh deklarasi variabel

Variabel di Dart dilengkapi dengan **type-safe**. Kita tidak wajib mendefinisikan tipe data variabel secara terpisah seperti pada Gambar 11. Kita dapat menggunakan keyword **var** untuk secara otomatis menginisialisasikan tipe data variabel berdasarkan *value* yang kita berikan pertama kali.



```
Run | Debug
1 void main() {
2     var a = 10;
3     var b = "abc";
4     var c = 10.2;
5     print("Tipe data dari variable a : ${a.runtimeType}");
6     print("Tipe data dari variable b : ${b.runtimeType}");
7     print("Tipe data dari variable c : ${c.runtimeType}");
8 }
```

Gambar 12. Deklarasi variabel menggunakan var

Bagaimana jika kita tidak memberikan *value* saat inisialisasi?

Dart memiliki tipe data **dynamic** (Dapat berubah tipe datanya selama *runtime*).

```
Run | Debug
1 void main() {
2     var a; //Variable dynamic menggunakan keyword var
3     print("Tipe data dari variable a dengan value $a : ${a.runtimeType}");
4     a = 10;
5     print("Tipe data dari variable a dengan value $a : ${a.runtimeType}");
6     a = "abc";
7     print("Tipe data dari variable a dengan value $a : ${a.runtimeType}");
8
9     dynamic b; //Variable dynamic menggunakan keyword dynamic
10    print("Tipe data dari variable b dengan value $b : ${b.runtimeType}");
11    b = 10.0;
12    print("Tipe data dari variable b dengan value $b : ${b.runtimeType}");
13    b = "abc";
14    print("Tipe data dari variable b dengan value $b : ${b.runtimeType}");
15
16    //Object c; //Akan error jika diuncomment
17    Object c = 10; //Variable dynamic menggunakan keyword Object
18    print("Tipe data dari variable c dengan value $c : ${c.runtimeType}");
19    c = "abc";
20    print("Tipe data dari variable c dengan value $c : ${c.runtimeType}");
21 }
```

Gambar 13. Deklarasi variabel dynamic

Apa perbedaan penggunaan *keyword* var, dynamic, dan Object?

## 1. Keyword var



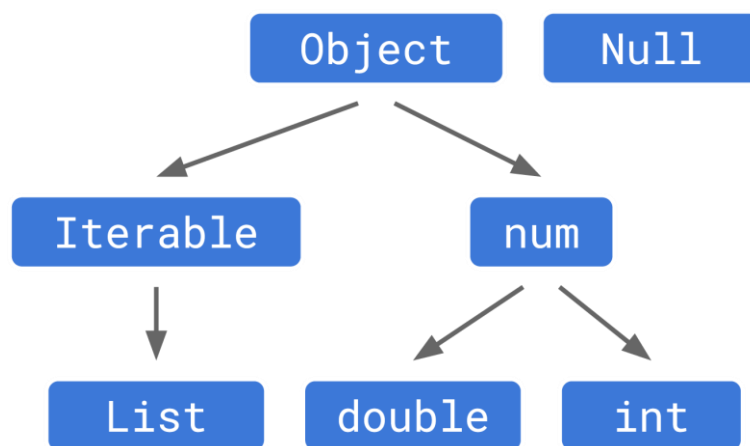
Tipe variabel secara otomatis akan didefinisikan berdasarkan *value* yang diberikan saat inisialisasi. Jika tidak diberikan *value*, maka tipe variabel yang akan diberikan berupa dynamic dengan *value* null.

## 2. Keyword dynamic

Merupakan salah satu tipe data yang dapat berubah-ubah sesuai *value* yang diberikan selama *runtime*.

## 3. Keyword Object

Di Dart, variabel merupakan sebuah *object* dari *class* berdasarkan tipe datanya. Seperti variable-variabel dengan tipe data String memiliki atribut dan *method* khusus, seperti `toUpperCase()`, `toLowerCase()`, dsb. *Object* merupakan *superclass* dari segala *class* di Dart selain Null sehingga ketika tidak diberi *value* saat inisialisasi akan error.



Gambar 14. Struktur class beberapa tipe data di Dart

## D. Built-in Type

Berikut beberapa tipe data yang tersedia di dart, untuk daftar tipe data lebih lengkap dapat merujuk ke [sini](#).



Tipe data	Keyword	Deskripsi
Number (integer, double)	num int double	Tipe data yang menyimpan <i>value</i> berupa angka atau bilangan. <ul style="list-style-type: none"><li>• int untuk bilangan bulat .</li><li>• double untuk bilangan desimal.</li><li>• Num merupakan class <i>parent</i> dari tipe data int dan double.</li></ul>
Strings	String	Tipe data yang menyimpan <i>value</i> berupa teks. Dapat menggunakan tanda petik tunggal atau ganda.
Booleans	bool	Tipe data yang menyimpan <i>value</i> berupa <i>true</i> atau <i>false</i> .
Lists	List<TypeData>	Biasa dikenal sebagai array.
Sets	Set<TypeData>	Kumpulan object unik yang tidak terurut.
Maps	Map<TypeData, TypeData>	Kumpulan object yang terhubung oleh <i>key</i> dan <i>value</i> .
null	-	Sebuah variabel yang tidak diberikan <i>value</i> akan berisi <i>value</i> null dan bertipe null.

Tabel 1. Daftar tipe-tipe data built-in di dart

## E. Control Flow 1

Untuk mengatasi berbagai kondisi tertentu, dart memiliki *if/else* seperti bahasa pemrograman pada umumnya.

```
Run | Debug
1 void main() {
2     var var1;
3     var1 = "String"; //Silahkan coba ubah nilai var1
4     if (var1 == "String" && var1 is String) {
5         print("var1 is String");
6     } else if (var1 is double || var1 is int) {
7         print("var1 is double or int");
8     } else {
9         print("var1 is not String or number");
10    }
11 }
```

Gambar 15. code if else

Pada Gambar 15, kita membuat sebuah variabel **var1** dengan tipe data **dynamic** yang ketika kita memberikan value **"String"** dia menjadi tipe data **String**. Pada kondisi **if** pertama, kita memeriksa Apakah **var1** berisi **"String"** dan **var1** merupakan tipe data **String** (*keyword is* digunakan untuk mengecek apakah tipe data atau *class* dari *object* disisi kiri sama dengan sisi kanan). Berikut beberapa operator yang dapat digunakan untuk membuat kondisi.

Deskripsi	Operator
Logika dan	&&
Logika atau	
Sama dengan, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	==
Tidak sama dengan, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	!=
Lebih besar sama dengan, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	>=



Lebih besar dari, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	>
Lebih kecil sama dengan, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	<=
Lebih kecil, membandingkan <i>value</i> dari kedua <i>object</i> yang dibandingkan	<
Sama tipe data atau <i>class</i>	is
Tidak sama data atau <i>class</i>	is!

Tabel 2. Daftar operator logika dalam Dart

```
Run | Debug
1 void main() {
2     var a;
3     a = 10;
4     //if else form
5     if (a is int) {
6         print("a is int");
7     } else {
8         print("a is not int");
9     }
10
11     //ternary form
12     a is int ? print("a is int") : print("a is not int");
13 }
```

Gambar 16. Ternary Operator

Kita juga dapat menyederhanakan penulisan if else dengan menggunakan **Ternary Operator** seperti pada Gambar 16. Ternary Operator memiliki sintaks sebagai berikut:

```
(Kondisi) ? (Jika True) : (Jika False);
```

Gambar 17. Sintaks ternary operator

```
Run | Debug
1  void main() {
2      var state = "init";
3
4      switch (state) {
5          case "init":
6              print("init");
7              break;
8          case "start":
9              print("start");
10             break;
11         case "loading":
12             print("loading");
13             break;
14         case "success":
15             print("success");
16             break;
17         case "failed":
18             print("failed");
19             break;
20         default:
21             print("State got error");
22     }
23 }
```

Gambar 18. Switch case

Untuk kondisi **if else** yang cukup banyak dan hanya membutuhkan operator **==** dapat menggunakan **Switch case**. Switch





case membanding *value* **integer**, **string**, atau **constant waktu-kompilasi** dengan operator **==**. Sintaks switch case pada dart sama seperti bahasa pemrograman pada umumnya. Salah satu hal yang harus diperhatikan bahwa keyword **break** bersifat **wajib**. Keyword break dapat digantikan dengan **continue**, **throw**, atau **return**. Hal lain yang harus diperhatikan bahwa value yang kita bandingkan harus memiliki tipe data yang sama untuk dibandingkan menggunakan operator **==**.

## F. Control Flow 2

Dalam dart terdapat tiga jenis perulangan yang dapat digunakan, yaitu **for loop**, **do-while**, dan **while**. Ketiganya memiliki sintaks yang sama seperti bahasa pemrograman pada umumnya. Untuk for loop memiliki dua sintaks yaitu **standard for loop** dan **for-in**.

```
Run | Debug
1  void main() {
2      //standard for loop
3      for (var i = 1; i <= 10; i++) {
4          print(i);
5      }
6
7      //for-in
8      var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
9      for (var n in numbers) {
10         print(n);
11     }
12 }
```

Gambar 19. For loop

Pada standard for loop terbagi menjadi tiga bagian, yaitu inisialisasi variabel, kondisi loop, dan increment. Untuk fungsi setiap



bagian sama dengan pada Bahasa Pemrograman Java. Untuk **for-in**, kita dapat melakukan loop untuk setiap elemen di object **kelas iterable (List, Set, Map)**. Pada Gambar 19, variabel *n* akan diberikan value berupa setiap elemen di dalam list numbers.

```
Run | Debug
1 void main() {
2     //for-each
3     var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
4     numbers.forEach((element) {
5         print(element);
6     });
7 }
```

Gambar 20. For each

Kita dapat pula memanfaatkan method **forEach()** yang dimiliki oleh kelas **iterable** (perhatikan Gambar 20). Cara kerja for each sama dengan for-in.

```
Run | Debug
1 void main() {
2     //while
3     var i = 1;
4     while (i <= 10) {
5         print(i);
6         i++;
7     }
8     //do-while
9     var n = 1;
10    do {
11        print(n);
12        n++;
13    } while (n <= 10);
14 }
```

Gambar 21. While dan do-while loop



Pada **while**, program akan melakukan perulangan selama kondisi yang dinyatakan terpenuhi dan berhenti ketika sudah tidak terpenuhi. Sedangkan pada **do-while** akan melaksanakannya setidaknya sekali dan melakukan perulangan selama kondisi yang dinyatakan terpenuhi dan berhenti ketika sudah tidak terpenuhi. Perbedaan while dan do-while terletak pada waktu pemeriksaan kondisi saat pertama kali dieksekusi. While akan memeriksa kondisi terlebih dahulu, sedangkan do-while akan eksekusi terlebih dahulu.

## G. List, Set, & Map

Untuk menyimpan banyak *value* dalam satu variabel kita dapat menggunakan tipe data List, Set, atau Map sesuai dengan kebutuhan.

### 1. List

List mirip seperti array di bahasa pemrograman lainnya. List menyimpan *value* secara berurutan dengan sistem indeks dimulai dari 0. Kita dapat mengakses elemen dari List menggunakan operator **[...]** dengan indeks elemen yang ingin diakses.

### 2. Set

Set menyimpan *value* unik. Hal yang membedakan Set dengan List adalah elemen dari Set harus unik. Jika kita memasukan *value* yang sudah ada di Set, maka *value* tersebut akan diabaikan. Kita dapat mengakses elemen dari Set menggunakan *method* **elemenAt(int indeks)**.

### 3. Map



267 Map mirip seperti List, tetapi indeks dari List digantikan dengan *key*.  
268 Map merupakan kombinasi dari **key** dan **value**. Kita dapat  
269 mengakses elemen dari Map melalui *key*.

270 Berikut contoh code pembuatan dan mengakses elemen List, Map,  
271 dan Set.

```
Run | Debug
1 void main() {
2     //List
3     List<int> listOfInt = [1, 2, 3];
4     print(listOfInt[1]); //List index ke-1 (Mulai dari 0)
5
6     //Map
7     Map<String, int> mapOfStringInt = {
8         "one": 1,
9         "two": 2,
10        "three": 3,
11    };
12    print(mapOfStringInt['two']); //Map key 'two'
13
14    //Set
15    Set<int> setOfInt = {1, 2, 3};
16    print(setOfInt.elementAt(1)); //Set index ke-1 (Mulai dari 0)
17 }
```

272

273 Gambar 22. Contoh list, set dan map

## 274 H. Function dan Arrow Function

275 Sintaks dari function di dart sama dengan di bahasa  
276 pemrograman pada umumnya. Berikut beberapa hal unik dari  
277 function di dart:

### 278 1. Named Parameters

279 Kita dapat menyertakan nama parameter saat pemanggilan  
280 function dengan menggunakan tanda kurung kurawal **{param1,**



**param2, ...}**. Named parameters akan bersifat **opsional** dan diberi **default value null** jika tidak diberikan arguments saat pemanggilan function. Setiap tipe data named parameter harus diberikan tanda **? (nullable)** untuk mengizinkan variabel dapat berisi *value* null. Kita juga dapat memanggil named parameter function secara tidak berurutan (Tidak harus sesuai dengan urutan di function).

Perhatikan Gambar 23, pada penggunaan parameter number di function **printHello** kita harus menambahkan tanda **! (non-null assertion operator)** untuk memastikan variabel nullable number untuk tidak null. Berhati-hati dalam penggunaan ini saat pembuatan aplikasi karena dapat menyebabkan *crash*. Untuk penjelasan lebih lanjut, lihat di [subbab null safety](#).

```
Run | Debug
1  void main() {
2    |   printHello(number: 10, text: "Hello World");
3  }
4
5  void printHello({String? text, int? number}) {
6    |   //named parameter, tanda ? untuk nullable
7    |   for (int i = 0; i < number!; i++) {
8    |       //tanda ! untuk menghilangkan nullable
9    |       print(text);
10   |   }
11   }
```

Gambar 23. Named Parameters

## 2. Required Parameters



Jika kita ingin mewajibkan variabel diisi dan tidak boleh ber-  
*value* null seperti parameter number di Gambar 24, maka kita  
dapat menambahkan keyword **required** di depannya tipe data  
parameters.

```
Run | Debug
1 void main() {
2   printHello(text: "Hello World", number: 10);
3   printHello(number: 1);
4 }
5
6 void printHello({String? text, required int number}) {
7   for (int i = 0; i < number; i++) {
8     print(text);
9   }
10 }
```

Gambar 24. Required Parameters

### 3. Default Value Parameters

Dalam beberapa kasus, kita ingin parameter tidak wajib untuk  
diisi, tetapi memiliki default value selain null. Kita dapat  
menerapkan default value seperti di Gambar 25.

```
Run | Debug
1 void main() {
2   printHello(text: "Hello World", number: 10);
3   printHello(number: 1);
4 }
5
6 void printHello({String text = "Kosong", required int number}) {
7   for (int i = 0; i < number; i++) {
8     print(text);
9   }
10 }
```

Gambar 25. Default value parameters

### 4. Function as First-class objects



Setiap variabel adalah *object*, begitu pula dengan function. Kita dapat menggunakan function sebagai tipe data dan *value* dari sebuah variabel. Perhatikan Gambar 26.

## 5. Anonymous Functions

Dalam pembuatan function, kita biasanya memberikan nama untuk dapat dipanggil. Kita juga dapat membuat function tanpa nama seperti function yang kita *assign* ke variabel `printHello` di Gambar 26.

```
Run | Debug
1 void main() {
2   var printHello = (String element) { //Menggassign sebuah function ke dalam variable
3     print(element);
4   };
5   var text = ["Hello", "World"];
6   text.forEach(printHello); //Menggunakan variable yang berisi function
7 }
```

Gambar 26. Function as Object & Anonymous Function

## 6. Arrow Function

Untuk mempersingkat penulisan function, kita dapat menuliskannya dalam sintaks **arrow function**. Arrow function hanya dapat digunakan untuk function yang hanya terdiri dari 1 baris seperti di Gambar 27.

```
2 var printHello = (String element) => print(element);
```

Gambar 27. Arrow Function

### I. Final Vs Const

Keyword `final` dan `const` sudah umum digunakan di beberapa bahasa pemrograman. Secara umum fungsi dari kedua keyword ini hampir mirip untuk deklarasi variabel yang tidak dapat diubah *value*—



nya setelah diinisialisasi. Berikut beberapa poin perbedaan antara  
final dan const.

Perbedaan	Final	Const
Inisialisasi value	Sebelum digunakan saat <i>runtime</i>	Saat kompilasi
Pendeklarasian	Dapat tidak diberikan <i>value</i> terlebih dahulu	Harus diberikan <i>value</i>
Value	Tidak ada batasan	Harus berupa <i>value</i> konstan

Tabel 3. Perbedaan Final dan Const

## J. Class dan Object

Seperti function, class di dart mirip dengan Bahasa pemrograman  
di Java. Berikut beberapa poin penting class di dart:

### 1. Format penamaan class adalah UpperCamelCase

### 2. Tidak ada keyword **protected**, **private**, dan **public**

Semua atribut dan method otomatis menjadi **public**. Untuk  
membuat atribut dan method **private**, kita dapat menambahkan  
**tanda garis bawah (\_) di awal nama variabel**. Pada Gambar 28,  
atribut `_name` dan `_password` merupakan atribut **private**,  
sedangkan *method* `User` dan `toString` merupakan *method* **public**.

Penggunaan *keyword* **late**, digunakan untuk menunda  
inisialisasi dari variabel final `_name` dan `_password` hingga  
*method* constructor dipanggil.





```
Run | Debug
1 void main() {
2     User user = User("Mahasiswa", "123");
3     print(user.toString());
4 }
5
6 class User {
7     late final String _name;
8     late final String _password;
9
10    User(String name, String password) {
11        _name = name;
12        _password = password;
13    }
14    @override
15    String toString() {
16        return 'User{name: $_name, password: $_password}';
17    }
18 }
```

Gambar 28. Contoh class User

### 3. Initializing formal parameter

Kita dapat mempersingkat sintaks constructor dengan menerapkan sintaks di Gambar 29. Di sini kita dapat meng-*assign value* dari constructor secara otomatis ke atribut dengan nama yang sama tanpa harus meng-*assign*-nya secara manual di body constructor. Di sini kita tidak perlu menambahkan *keyword* *late* karena atribut `_name` dan `_password` secara langsung di-*assign* saat pemanggilan constructor.



```
6  class User {  
7      final String _name;  
8      final String _password;  
9  
10     User(this._name, this._password);  
11  
12     @override  
13     String toString() {
```

Gambar 29. Initializing formal parameters

#### 4. Default Constructor

Jika tidak dibuat constructor, maka secara otomatis akan disediakan constructor kosong yang tidak memiliki argument.

#### 5. Constructor tidak diwariskan

Subclass yang tidak dideklarasikan constructornya tidak akan mengwarisi constructor *parent*-nya, melainkan akan memiliki default constructor.

#### 6. Named Constructor

Kita dapat membuat constructor dengan penamaan khusus seperti constructor **User.register()** di Gambar 30.



```
Run | Debug
1  void main() {
2      User user = User("Mahasiswa", "123");
3      User guest = User.register();
4      print(user.toString());
5      print(guest.toString());
6  }
7
8  class User {
9      final String _name;
10     final String _password;
11
12     User(this._name, this._password);
13
14     User.register()
15     |   : _name = "Guest",
16     |   | _password = "123";
17
18     @override
19     String toString() {
20         return 'User{name: $_name, password: $_password}';
21     }
22 }
```

Gambar 30. Named Constructor

## 7. Getters dan Setters

Di dart ada *method* khusus untuk membaca dan menulis atribut dari object. Perhatikan Gambar 31, kita menambahkan *method* getter di class User berupa `checkLogin` dan setter berupa `login` yang pemanggilannya sama seperti pemanggilan getter dan setter atribut dari class.



```
Run | Debug
1 void main() {
2     User user = User(name: "User", password: "123");
3     User guest = User.register();
4     print(user.toString());
5     print(guest.toString());
6
7     print(user.checkLogin);
8     user.login = '12345';
9     print('${user.toString()} ${user.checkLogin}');
10 }
11
12 class User {
13     final String name;
14     final String password;
15     String token;
16
17     User({required this.name, required this.password, this.token = ''});
18
19     User.register()
20         : name = "Guest",
21           password = "123",
22           token = '';
23
24     bool get checkLogin => token == '' ? false : true; //getter
25     set login(String token) => this.token = token; //setter
26
27     @override
28     String toString() {
29         return 'User{name: $name, password: $password, token: $token}';
30     }
31 }
```

Gambar 31. Getter dan setter

## K. Exception Handling

Untuk mengatasi error yang terjadi saat running, kita dapat menerapkan Exception Handling di beberapa kasus sesuai kebutuhannya. Exception Handling terdiri dari tiga bagian utama:

### 1. Throw

Di dart kita dapat men-*throw object* apapun selain null, tidak hanya class Exception atau anaknya.

### 2. Catch

Untuk menerima *throw*, kita dapat menerimanya di blok catch.

### 3. Finally



389                    Untuk menjalankan program yang tetap jalan meskipun ada  
390                    exception yang dilempar, kita dapat meletakkannya di blok finally.



```
void main() {
  User user = User(name: "User", password: "123");
  try {
    //Coba uncomment salah satu kode dibawah ini untuk melihat reaksi throw catch dan finally
    user.login('User', '123'); //tidak ada yang dithrow
    //user.login('', ''); //throw String 'Username or password cannot be empty';
    // user.login('A', '12'); //throw FailedLogin object
    print("Login Successfully");
  } on FailedLogin catch (e) {
    print(e.errorMessage()); //menangkap exception bertipe FailedLogin
  } on String catch (e) {
    print(e); //menangkap exception bertipe String
  } catch (e) {
    print(e); //menangkap semua exception
  } finally {
    print("Finally"); //finally akan selalu dijalankan
  }
}

class FailedLogin implements Exception {
  //Object yang mengimplement Exception
  String errorMessage() {
    return "Login Failed";
  }
}

class User {
  final String name;
  final String password;
  String token;

  User({required this.name, required this.password, this.token = ''});

  User.register()
    : name = "Guest",
      password = "123",
      token = '';

  bool get checkLogin => token == '' ? false : true; //getter
  set setToken(String token) => this.token = token; //setter

  void login(String name, String password) {
    if (name == this.name && password == this.password) {
      token = '123';
    } else if (name == '' || password == '') {
      throw 'Username or password cannot be empty';
    } else {
      throw FailedLogin();
    }
  }

  @override
  String toString() {
    return 'User{name: $name, password: $password, token: $token}';
  }
}
```

Gambar 32. Contoh penerapan Exception Handling

## L. Future, Async, & Await



Secara umum program berjalan secara sinkron. Tapi, terkadang kita membutuhkan pemrograman asinkronus ketika berhadapan dengan operasi yang membutuhkan waktu yang lama, seperti menunggu respon jawaban dari server. Terdapat tiga *keyword* penting untuk pemrograman asinkronus, yaitu:

### 1. **Future**

Keyword Future digunakan dengan membungkus class atau tipe data untuk menandai object atau variabel itu membutuhkan waktu untuk mengembalikan *value*-nya.

### 2. **Async**

Untuk menandai sebuah function akan melakukan operasi yang memakan waktu di dalamnya. Diletakkan di antara parameter dan body dari function. Untuk menggunakan *keyword* *await* di sebuah function, kita harus menggunakan *keyword* *async* terlebih dahulu.

### 3. **Await**

Berfungsi untuk menunggu hasil dari Future yang sedang berjalan dalam blok program *async*. Program akan berhenti hingga mendapatkan *value* balikan.

Bagaimana contoh penerapan pemrograman asinkronus? Mari kita mengembangkan program pada Gambar 32 dengan menambahkan sebuah class baru Bernama *LoginRepository* yang mengelola koneksi aplikasi dengan server. Dimana class *LoginRepository* memegang *value* *name* dan *password* user yang



418 benar untuk berhasil login dan sebuah *method* bernama login untuk  
419 memproses login aplikasi.

```
void main() {  
    User user = User(name: "User", password: "123");  
    LoginRepository loginRepository = LoginRepository();  
    try {  
        loginRepository.login(user.name, user.password);  
    } on FailedLogin catch (e) {  
        print(e.errorMessage());  
    } on String catch (e) {  
        print(e);  
    } catch (e) {  
        print(e);  
    } finally {  
        print("Finally");  
    }  
}  
  
class FailedLogin implements Exception {  
    String errorMessage() {  
        return "Login Failed";  
    }  
}  
  
class User {  
    final String name;  
    final String password;  
    String token;  
  
    User({this.name = '', this.password = '', this.token = ''});  
  
    User.register()  
        : name = "Guest",  
          password = "123",  
          token = '';  
  
    bool get checkLogin => token == '' ? false : true;  
    set setToken(String token) => this.token = token;  
  
    @override  
    String toString() {  
        return 'User{name: $name, password: $password, token: $token}';  
    }  
}
```

420

421

Gambar 33. Pengembangan dari Gambar 32



```
class LoginRepository {
  //Data Akun dummy
  String username = "User";
  String password = "123";

  User login(String username, String password) {
    print("Logging...");
    User userData = User();
    if (this.username == username && this.password == password) {
      userData = User(
        name: username,
        password: password,
        token: "12345"); //Sebuah process permintaan ke server
    } else if (this.username == '' || this.password == '') {
      throw 'Username or password cannot be empty';
    } else {
      throw FailedLogin();
    }
    print("Login Success! Here your data.. $userData");
    return userData;
  }
}
```

Gambar 34. Lanjutan Gambar 33

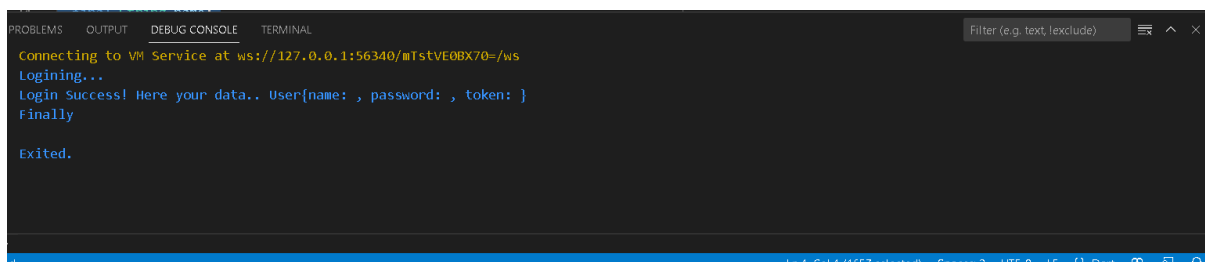
Gambar 35. Hasil keluaran Gambar 33

Jika dijalankan, kita akan mendapatkan hasil seperti gambar 35. Dapat kita lihat program berjalan secara beruntun dari proses permintaan untuk login hingga mendapatkan data user dan berhasil login. Akan tetapi proses login sebenarnya tidaklah seperti di Gambar 35, dibutuhkan waktu untuk aplikasi mengirimkan *request* ke server hingga mendapatkan *response* kembali. Mari kita mengembangkan Gambar 35 untuk lebih mensimulasikan proses login sebenarnya dengan menggunakan **Future.delayed** di Gambar 36.

```
class LoginRepository {
  //Data Akun dummy
  String username = "User";
  String password = "123";

  User login(String username, String password) {
    print("Logging...");
    User userData = User();
    Future.delayed(Duration(seconds: 3), () {
      if (this.username == username && this.password == password) {
        userData = User(
          name: username,
          password: password,
          token: "12345"); //Sebuah process permintaan ke server
      } else if (this.username == '' || this.password == '') {
        throw 'Username or password cannot be empty';
      } else {
        throw FailedLogin();
      }
    }); //Process request dan reponse
    print("Login Success! Here your data.. $userData");
    return userData;
  }
}
```

Gambar 36. Class LoginRepository dengan Future.delayed

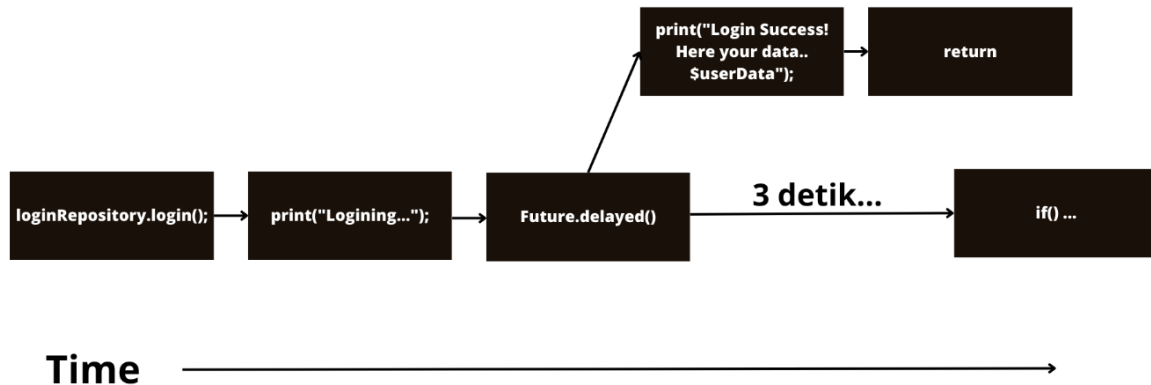


Gambar 37. Hasil keluaran dari Gambar 36

Hasil keluaran dari kedua Gambar 33 dan Gambar 34 hampir sama. Keduanya mengembalikan login berhasil. Akan tetapi, jika diperhatikan pada Gambar 35, data user yang kita dapatkan kosong. Hal tersebut terjadi karena saat kita menggunakan Future.delayed, program akan berjalan secara asinkron. Setelah Future.delayed terpicu, program akan langsung lanjut menjalankan baris berikutnya sambil tetap menjalankan isi dari Future.delayed. Hal ini yang



445 menyebabkan saat kita memanggil print data user, kita  
446 mendapatkan data user yang kosong.



447

448

Gambar 38. Alur kasar Gambar 36

449

450

451

452

Bagaimana jika kita membutuhkan data tersebut sebelum melakukan suatu aksi tertentu seperti print di Gambar 36 di atas? Kita dapat menggunakan tiga *keyword* yang telah dijelaskan sebelumnya. Berikut penerapannya



```
class LoginRepository {  
  //Data Akun dummy  
  String username = "User";  
  String password = "123";  
  
  Future<User> login(String username, String password) async {  
    print("Logging...");  
    User userData = User();  
    await Future.delayed(Duration(seconds: 3), () {  
      if (this.username == username && this.password == password) {  
        userData = User(  
          name: username,  
          password: password,  
          token: "12345"); //Sebuah process permintaan ke server  
      } else if (this.username == '' || this.password == '') {  
        throw 'Username or password cannot be empty';  
      } else {  
        throw FailedLogin();  
      }  
    }); //Process request dan reponse  
    print("Login Success! Here your data.. $userData");  
    return userData;  
  }  
}
```

Gambar 39. Class LoginRepository dengan penerapan future, async, dan await

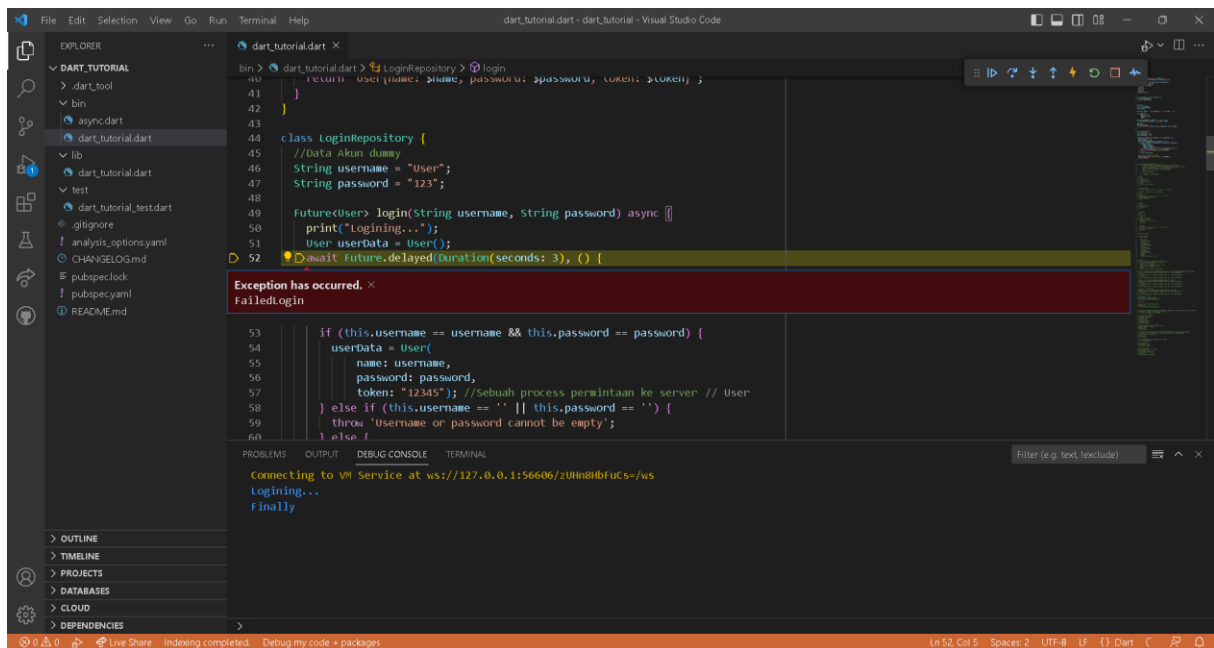
```
Connecting to VM Service at ws://127.0.0.1:56491/gi6TGC6M1wU/ws  
Logging...  
Finally  
Login Success! Here your data.. User{name: User, password: 123, token: 12345}  
Exited.
```

Gambar 40. Hasil keluaran dari Gambar 39

Perhatikan penggunaan dari ketiga *keyword* tersebut pada *method* login di class LoginRepository. Kita menggunakan *keyword* await pada Future.delayed untuk menandakan bahwa di *method* ini kita akan menunggu hingga proses dari blok Future.delayed selesai dilakukan, sebelum lanjut ke baris berikutnya. Namun, jika kita mencoba untuk memberikan name atau password yang salah, maka kita akan menerima *crash*. Sebagai tantangan, silahkan mencoba

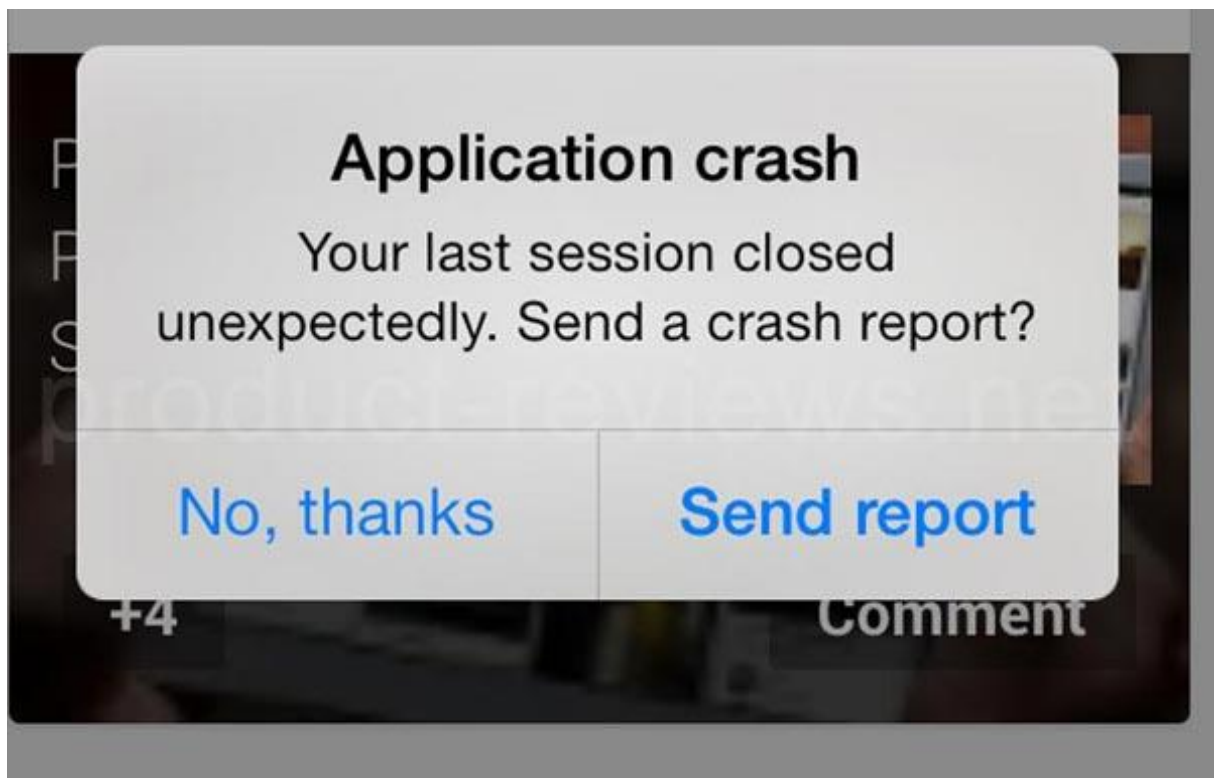


memperbaikinya terlebih dahulu sebelum melihat solusinya di Guided.



Gambar 41. Peringatan error ketika memberikan name dan password yang salah

## M. Null Safety



Gambar 42. Screenshot aplikasi crash



Apakah teman-teman pernah mengalami hal seperti di gambar 42? Salah satu hal penyebab *crash* pada aplikasi adalah null. Ketika sebuah variabel yang tidak seharusnya null diberikan value null atau variabel diakses sebelum inisialisasi dapat menyebabkan *crash* pada aplikasi. Untuk mencegah hal ini, dart memiliki null safety, yaitu sistem yang tidak akan menjalankan aplikasi yang dapat memicu *error* tersebut. Berikut beberapa poin penting mengenai null safety di dart

### 1. Tipe Nullable dan non-nullable

Semua tipe variabel secara default merupakan non-nullable, kecuali diberikan nullable operator (?) setelah tipe datanya. Untuk contoh lihat Gambar 23.

### 2. Null assertion Operator (!)

Jika ada variabel dengan tipe nullable, tetapi pada suatu bagian program kita yakin tidak bernilai null, maka kita dapat menambahkan null assertion operator. Untuk contoh lihat Gambar 23, kita yakin bahwa parameter number akan memiliki value sehingga kita menambahkan null assertion operator pada blok for. Berhati-hatilah dalam menggunakan operator ini. Jika variabel tersebut ternyata bernilai null, maka dapat menyebabkan crash pada aplikasi.

### 3. Null aware Operator (??)

Bagaimana jika kita tidak yakin Apakah sebuah variabel tipe nullable tidak akan bernilai saat kita gunakan? Kita dapat menggunakan null aware operator sebagai pengganti null assertion operator. Berbeda dengan null assertion operator yang



akan memicu error saat valuenya berupa null, null aware operator akan memberikan value yang kita berikan jika null. Seperti Gambar 43 variabel number bernilai null, maka dia akan digantikan dengan value 10 yang kita berikan di sisi kanan null aware operator.

```
void main() {  
  printHello(text: "Hello World");  
}  
  
void printHello({String? text, int? number}) {  
  for (int i = 0; i < (number ?? 10); i++) {  
    print(text);  
  }  
}
```

Gambar 43. Penerapan Null aware Operator

#### 4. Null-coalescing Operator (??=)

Null-coalescing Operator memiliki fungsi yang mirip dengan null aware operator. Namun dia juga memberikan value yang disediakan ke dalam variabel. Silahkan mencoba menjalankan program pada Gambar 44 dan perhatikan value variabel number.

```
void main() {  
  printHello(text: "Hello World");  
}  
  
void printHello({String? text, int? number}) {  
  print(number);  
  for (int i = 0; i < (number ??= 10); i++) {  
    print(text);  
  }  
  print(number);  
}
```

Gambar 44. Penerapan Null-coalescing operator

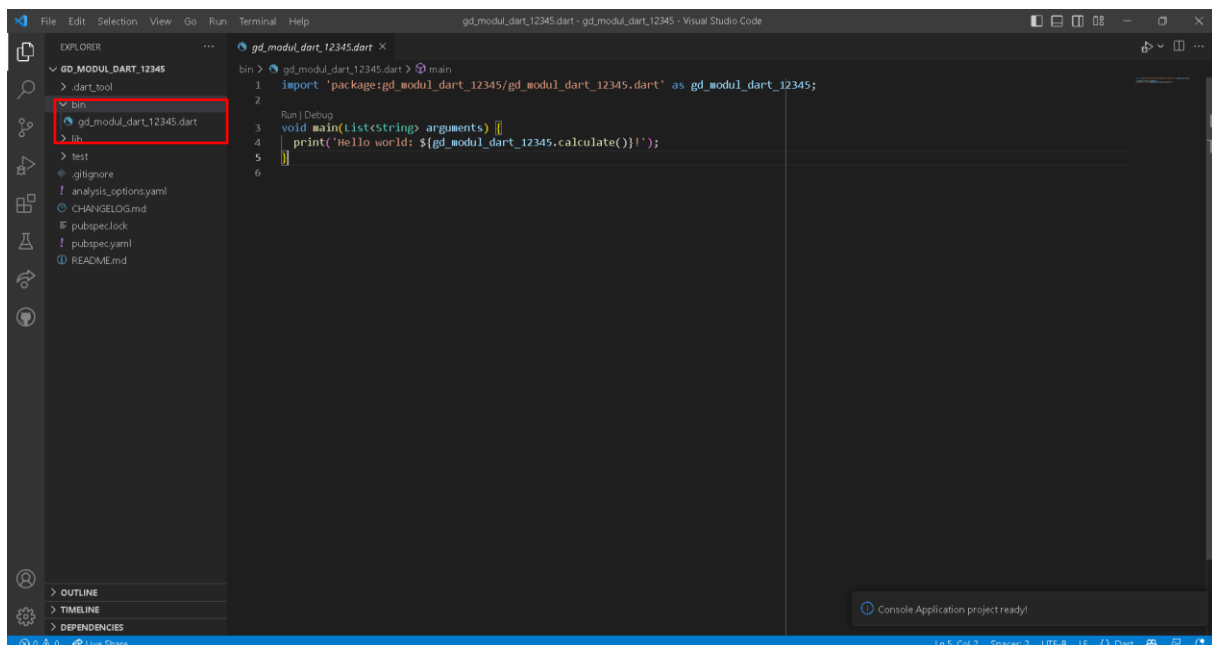
## GUIDED 1 – LOGIN CONSOLE APP

### Poin yang dipelajari dalam Guided 1 ini, yaitu :

1. Memahami cara membuat projek console dart.
2. Memahami penerapan pemrograman asikronus dalam logika bisnis Login.

Pada guided 1 ini, kita akan mencoba membuat aplikasi console login sederhana menggunakan Bahasa Pemrograman Dart. Silahkan ikuti langkah – langkah berikut ini :

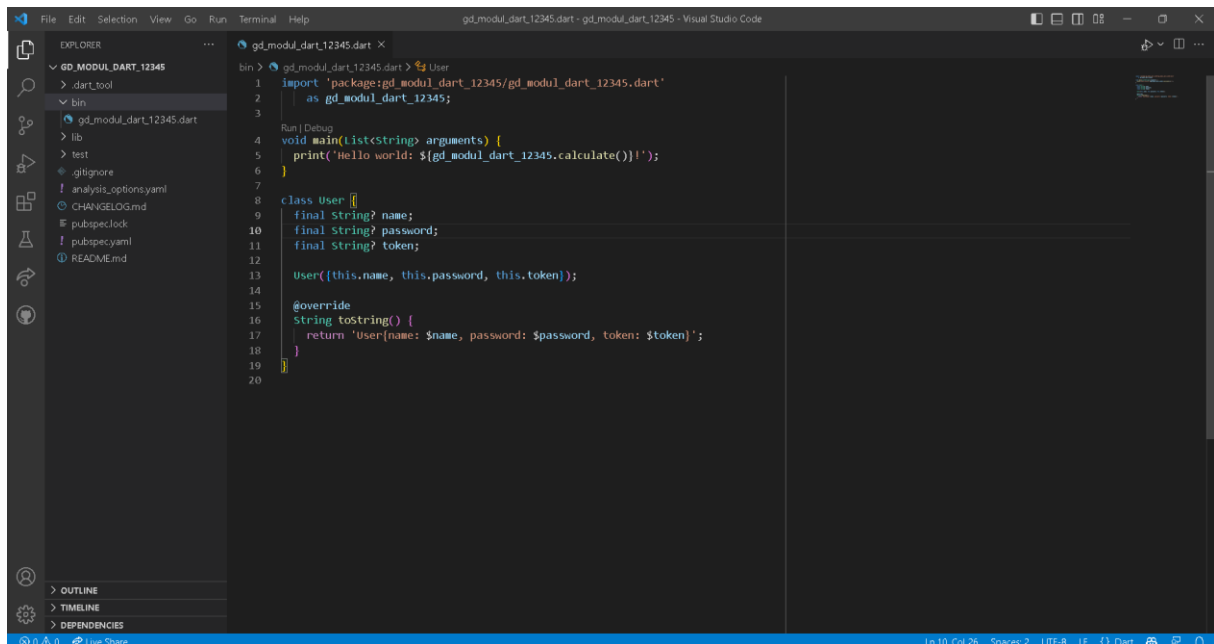
1. Buat project dart baru sesuai panduan pada **subbab Hello World** dan beri nama 'gd\_modul\_dart\_xxxx' (xxxx: empat digit npm terakhir).
2. Buka file gd\_modul\_dart\_ xxxx.dart di folder bin.



Gambar 45. File gd\_modul\_dart\_xxxx.dart

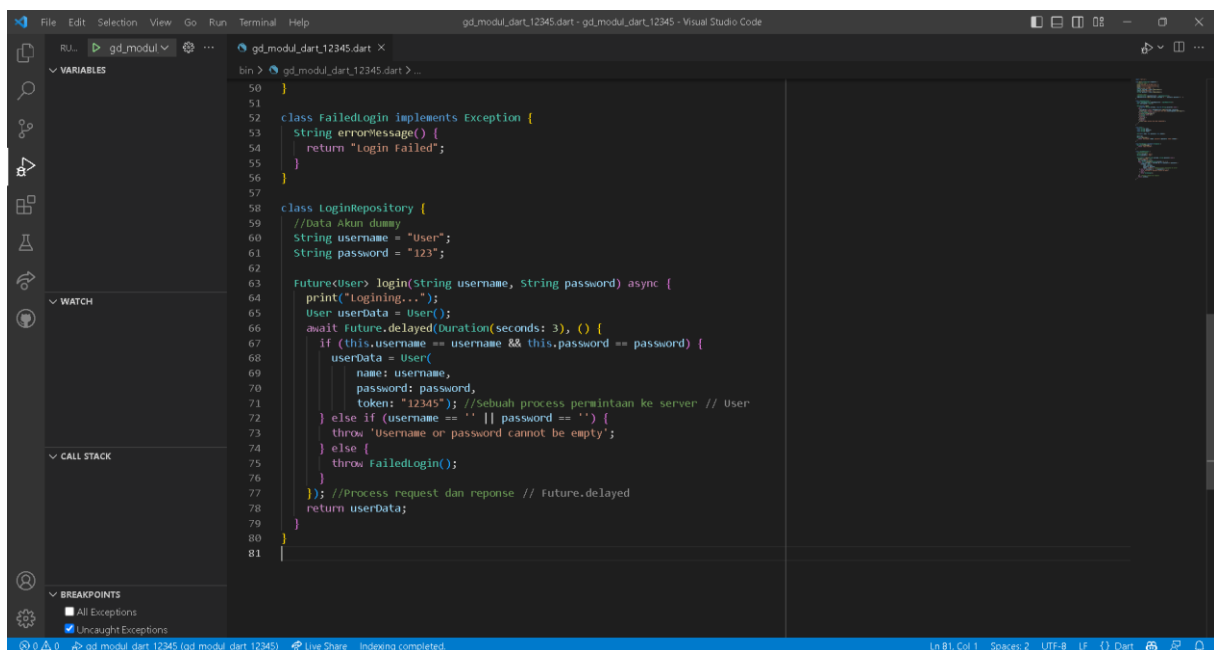
3. Sebelum mengubah main(), mari kita membuat class User sebagai model dari user yang akan kita gunakan.





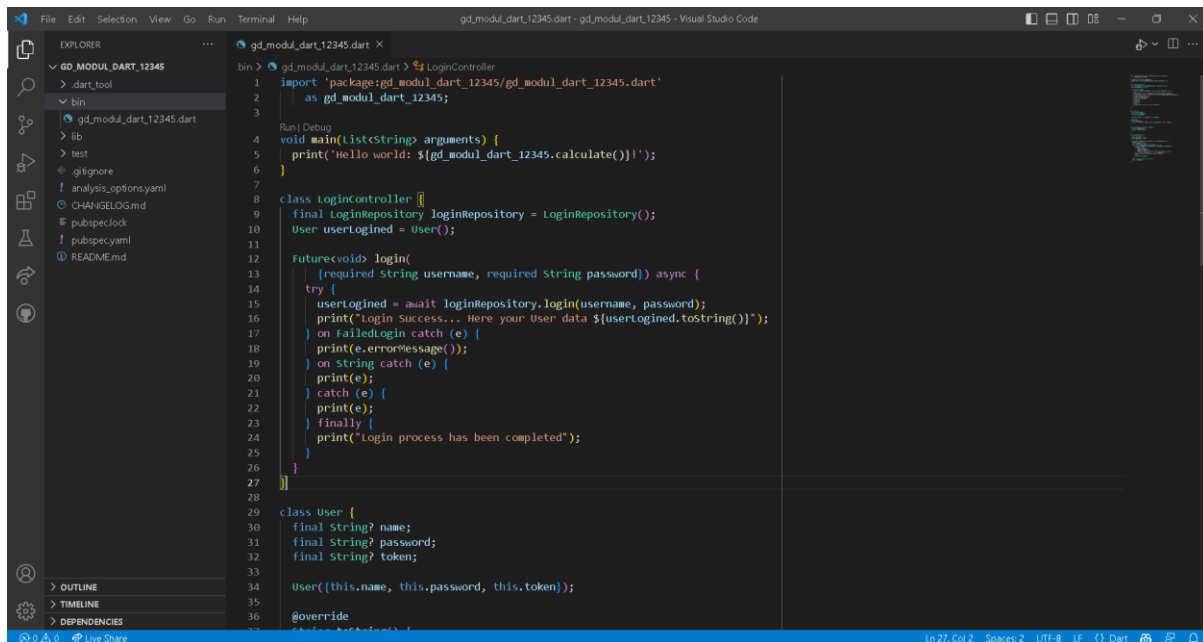
Gambar 46. Class User

4. Selanjutnya mari membuat LoginRepository yang akan mengelola koneksi kita dengan server (Dalam hal ini hanya dummy) dan class LoginFailed sebagai exception handling.



Gambar 47. Class Repository

5. Terakhir kita akan membuat LoginController sebagai penghubung antara LoginRepository dengan view kita (Dalam hal ini main()).



```
bin > gd_modul_dart_12345.dart > LoginController
1 import 'package:gd_modul_dart_12345/gd_modul_dart_12345.dart'
2   as gd_modul_dart_12345;
3
4 void main(List<String> arguments) {
5   print('Hello world: ${gd_modul_dart_12345.calculate()}!');
6 }
7
8 class LoginController {
9   final LoginRepository loginRepository = LoginRepository();
10   User userLoggedIn = User();
11
12   Future<void> login(
13     [required String username, required String password]) async {
14     try {
15       userLoggedIn = await loginRepository.login(username, password);
16       print("Login Success... Here your User data ${userLoggedIn.toString()}");
17     } on FailedLogin catch (e) {
18       print(e.errorMessage());
19     } on String catch (e) {
20       print(e);
21     } catch (e) {
22       print(e);
23     } finally {
24       print("Login process has been completed");
25     }
26   }
27 }
28
29 class User {
30   final String? name;
31   final String? password;
32   final String? token;
33
34   User([this.name, this.password, this.token]);
35
36   @override
37   String toString() {
38     return 'User{name: $name, password: $password, token: $token}';
39   }
40 }
```

Gambar 48. Class LoginController

6. Sekarang mari membuat view-nya di main(). Pertama silahkan mengosongkan isi dari main() dan menghapus import yang telah digenerate otomatis oleh dart karena kita tidak akan menggunakannya. Kemudian silahkan mengisi main() seperti gambar di bawah.
  - a. stdout.write() berfungsi untuk menampilkan *keluaran* tanpa baris baru
  - b. stdin.readLineSync() berfungsi untuk membaca masukan 1 baris dari pengguna. Perlu diperhatikan tipe balikan dari fungsi ini adalah String Nullable sehingga variabel username dan password kita tambahkan nullable operator.

```

1 void main(List<String> arguments) {
2   //Tampilan Login
3   print('Welcome to GD Modul Dart!');
4   print('=====');
5   print('-----Login-----');
6   stdout.write('Username: ');
7   String? username = stdin.readLineSync();
8   stdout.write('Password: ');
9   String? password = stdin.readLineSync();
10
11  //proses login
12  LoginController loginController = LoginController();
13  loginController.login(username: username ?? '', password: password ?? '');
14
15
16  class LoginController {
17    final LoginRepository loginRepository = LoginRepository();
18    User userLoggedIn = User();
19
20    Future<void> login(
21      {required String username, required String password}) async {
22      try {
23        userLoggedIn = await loginRepository.login(username, password);
24        print('Login Success... Here your User data ${userLoggedIn.toString()}');
25      } on FailedLogin catch (e) {
26        print(e.errorMessage());
27      } on String catch (e) {
28        print(e);
29      } catch (e) {
30        print(e);
31      } finally {
32        print('Login process has been completed');
33      }
34    }
35  }
36

```

Gambar 49. Isi main program login console app

7. Kita mendapatkan error dari menggunakan stdout.write dan stdin.readLineSync(). Jika teman-teman mencoba mengarahkan kursor ke salah satu tempat yang digaris bawahin merah. Kita dapat melihat hal yang menyebabkan error dan saran solusi yang diberikan.

```

1 vo Undefined name 'stdout'.
2 Try correcting the name to one that is defined, or defining the name. dart(unknown_identifier)
3 Type: dynamic
4 View Problem (Alt+F8) Quick Fix... (Ctrl+)
5
6 stdout.write('Username: ');
7 String? username = stdin.readLineSync();
8 stdout.write('Password: ');
9 String? password = stdin.readLineSync();
10
11 //proses login
12 LoginController loginController = LoginController();
13 loginController.login(username: username ?? '', password: password ?? '');
14
15
16 class LoginController {
17   final LoginRepository loginRepository = LoginRepository();
18   User userLoggedIn = User();
19
20   Future<void> login(
21     {required String username, required String password}) async {
22     try {
23       userLoggedIn = await loginRepository.login(username, password);
24       print('Login Success... Here your User data ${userLoggedIn.toString()}');
25     } on FailedLogin catch (e) {
26       print(e.errorMessage());
27     } on String catch (e) {
28       print(e);
29     } catch (e) {
30       print(e);
31     } finally {
32       print('Login process has been completed');
33     }
34   }
35 }
36

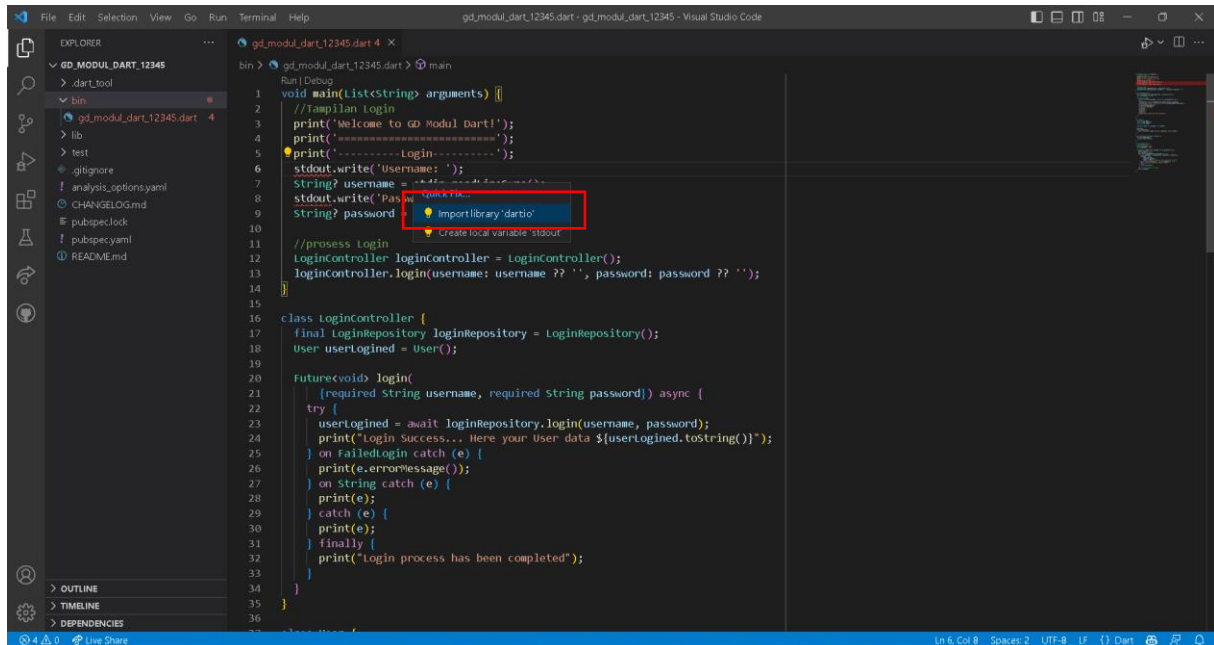
```

Gambar 50. Pesan error pada stdin.write dan stdin.readLineSync

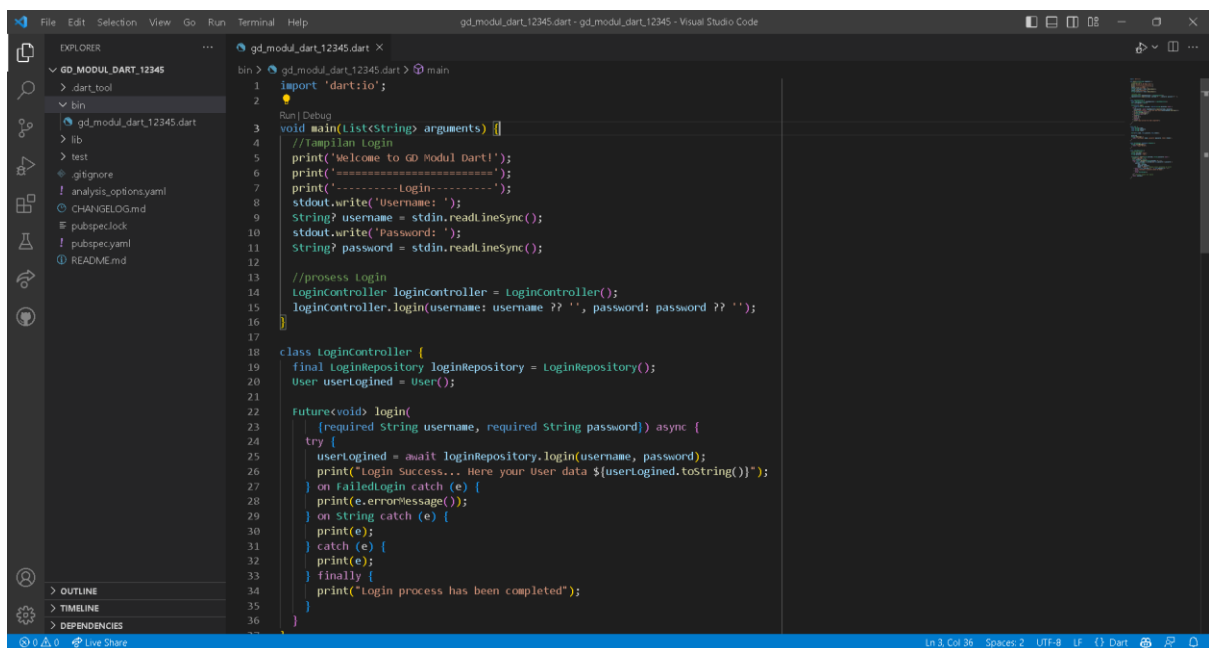
8. Jika diklik pilihan Quick Fix atau dapat menggunakan shortcut Ctrl + titik (.) akan muncul pilihan solusi yang dapat kita lakukan untuk



mengatasi error tersebut. Karena kita ingin menggunakan library  
dart.io, mari kita pilih import library 'dart.io'. maka error akan hilang.



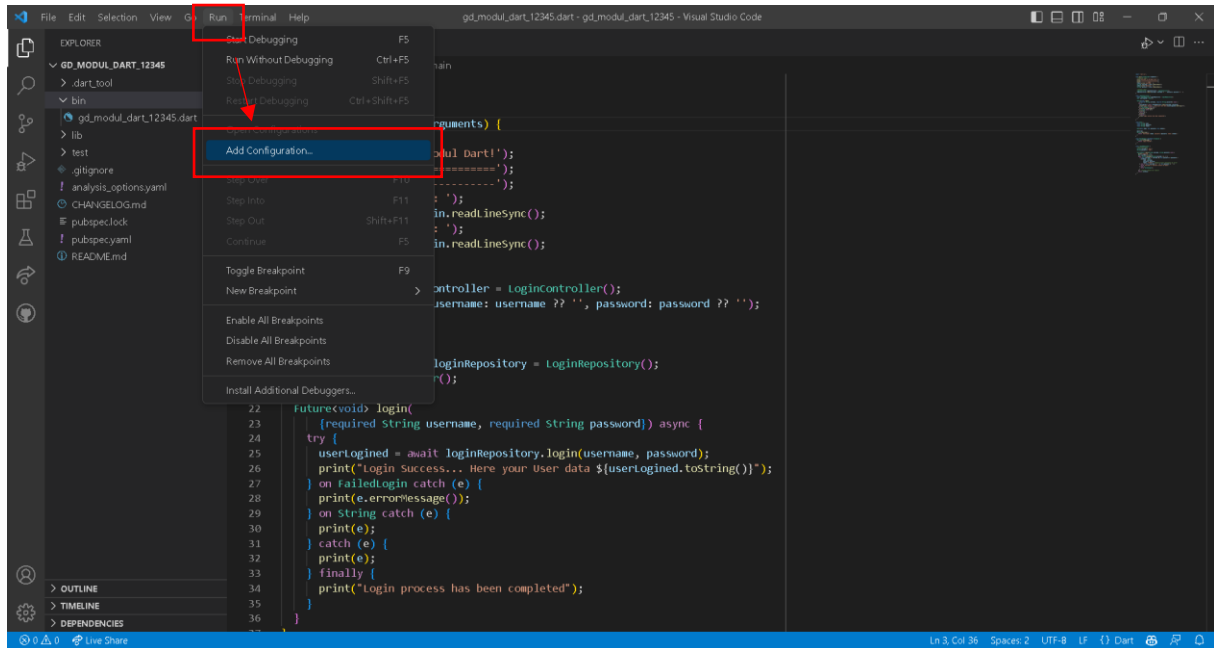
Gambar 51. Tampilan Solution



Gambar 52. Hasil setelah diimport

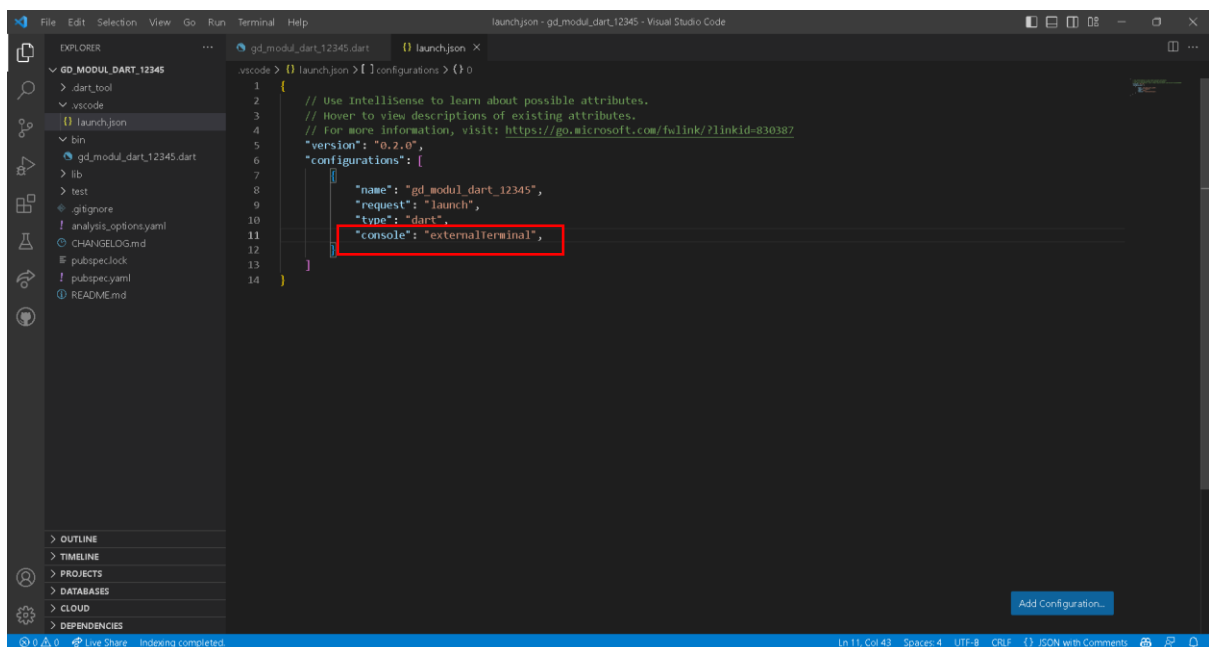
9. Jika kita langsung menjalankan program ini. Program akan dijalankan  
di Debug Console Visual Studio. Akan tetapi di Debug Console, kita  
tidak dapat menerima inputan sehingga kita harus mengantikkannya  
untuk menjalankan program menggunakan console external atau

console lainnya yang dapat menerima inputan. Untuk mengantinya  
silahkan pilih tab Run dan pilih menu Add Configuration...



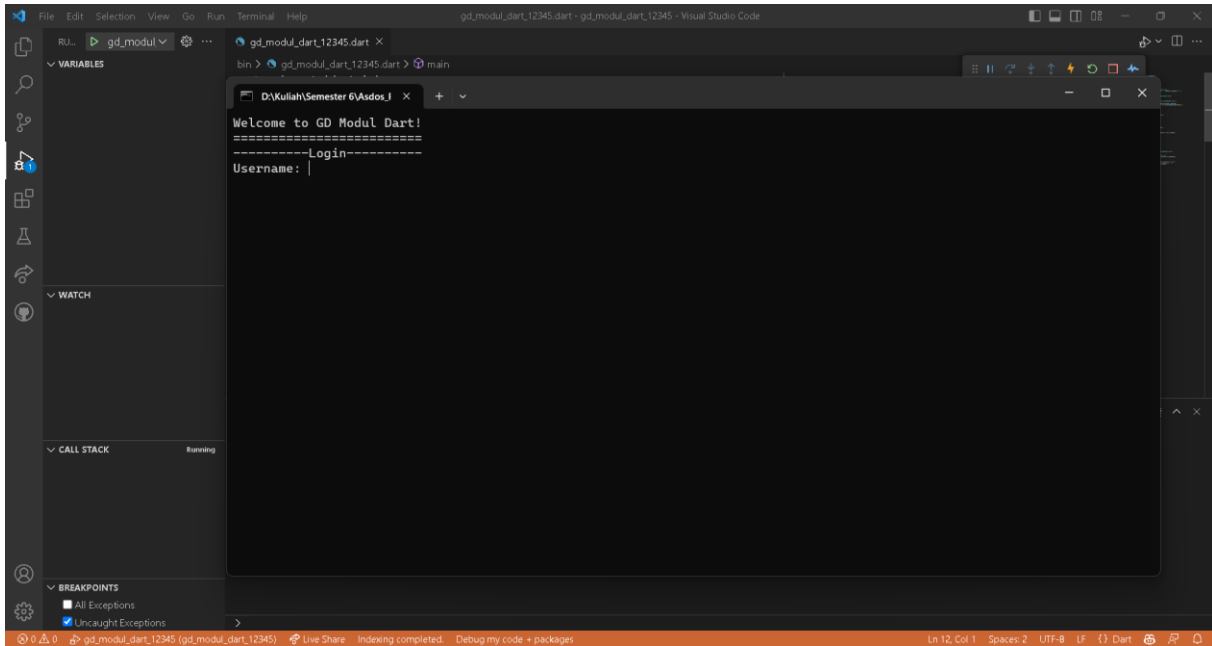
Gambar 53. Arahkan untuk menambahkan konfigurasi pada run program

10. Kita akan diarahkan ke file launch.json dan tambahkan di dalam  
configuration seperti di gambar bawah ini.



Gambar 54. Isi dari file launch.json

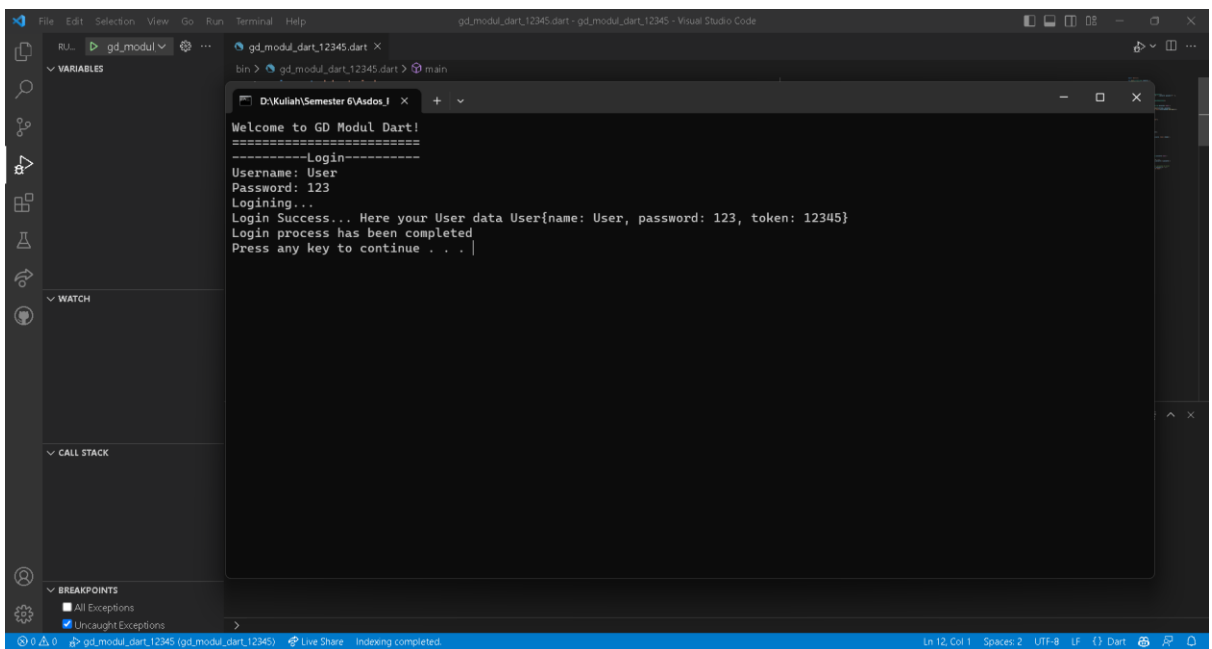
11. Setelah itu, gunakan shortcut Fn + F5 untuk menjalankan program  
dan kalian akan mendapatkan hasil seperti ini.



578

579

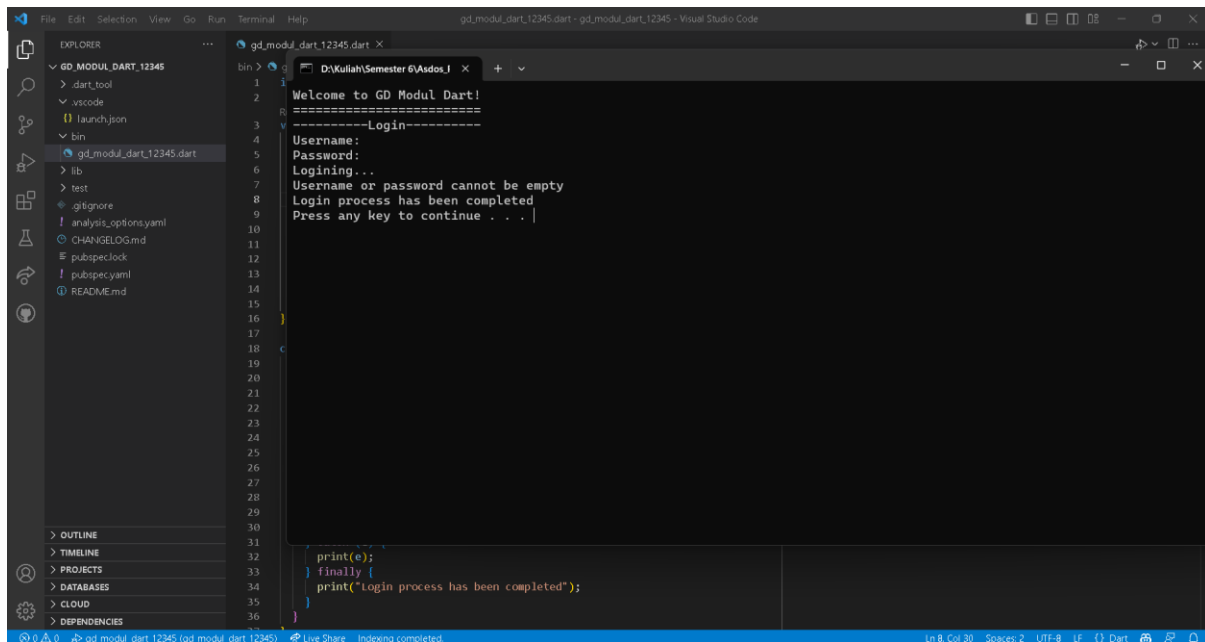
Gambar 55. Hasil Guided 1 tampilan awal



580

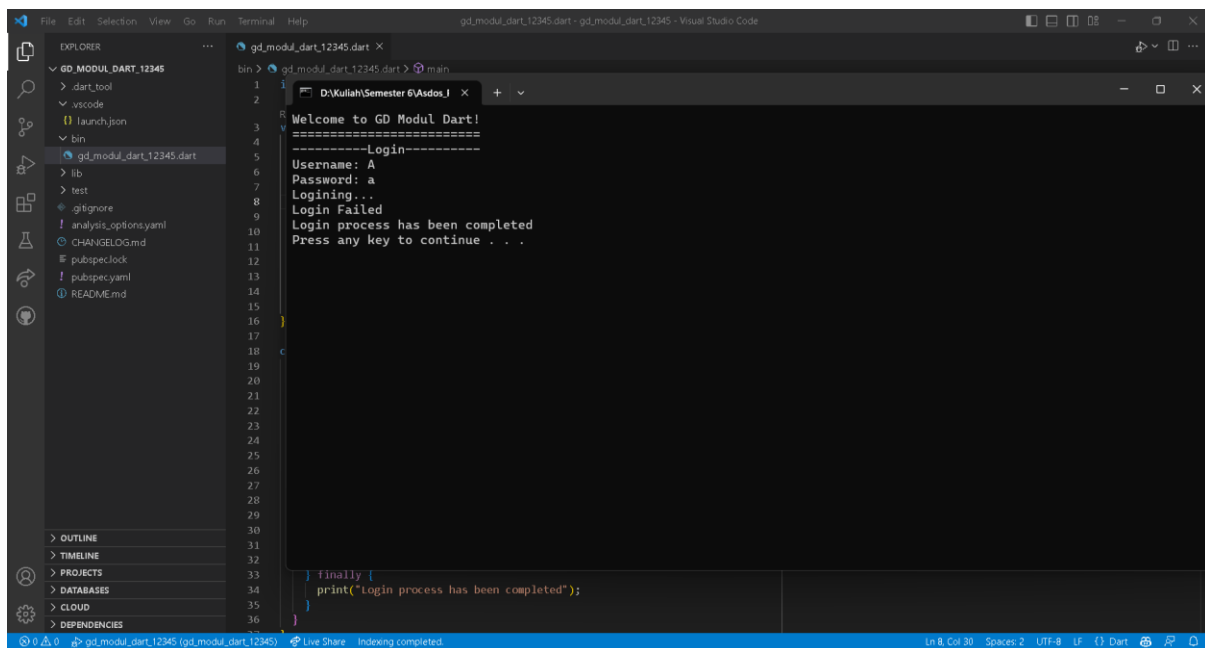
581

Gambar 56. Hasil Guided 1 tampilan berhasil login



```
1 Welcome to GD Modul Dart!
2 -----Login-----
3
4 Username:
5 Password:
6 Logging...
7 Username or password cannot be empty
8 Login process has been completed
9 Press any key to continue . . . |
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

Gambar 57. Hasil Guided 1 Tampilan Username dan Password kosong



```
1 Welcome to GD Modul Dart!
2 -----Login-----
3
4 Username: A
5 Password: a
6 Logging...
7 Login Failed
8 Login process has been completed
9 Press any key to continue . . . |
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

Gambar 58. Hasil Guided 1 Tampilan Username atau Password salah

### ATURAN PENGERJAAN GUIDED :

- Guided dikerjakan selama waktu perkuliahan berlangsung.
- Penamaan projek guided harus sesuai dengan yang sudah dicontohkan.



- 591 - **Guided** dikumpulkan melalui github dengan penamaan setiap file  
592 pada github adalah : **NAMAGUIDED\_XXXX** (contoh :  
593 **GuidedI\_Dart\_9999**)
- 594 ○ **NAMAGUIDED → SESUAI DENGAN CONTOH DALAM MODUL INI**
  - 595 ○ **XXXX → 4 DIGIT TERAKHIR NPM**
- 596 - Setelah diupload melalui github, jangan lupa untuk  
597 mengumpulkan keseluruhan link file github melalui situs kuliah.
- 598 - Cara upload ke github, silahkan melihat pada modul **"UPLOAD**  
599 **GITHUB"**.