



Dart Unit Test

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow





Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com



Sebelum Belajar

- Dart Dasar
- Dart Object Oriented Programming
- Dart Generic
- Dart Packages
- Dart Collection



Agenda

- Pengenalan Software Testing
- Testing Package
- Test dan Group Function
- Expect Function
- Matcher
- Mock Object
- Mockito
- Dan lain-lain

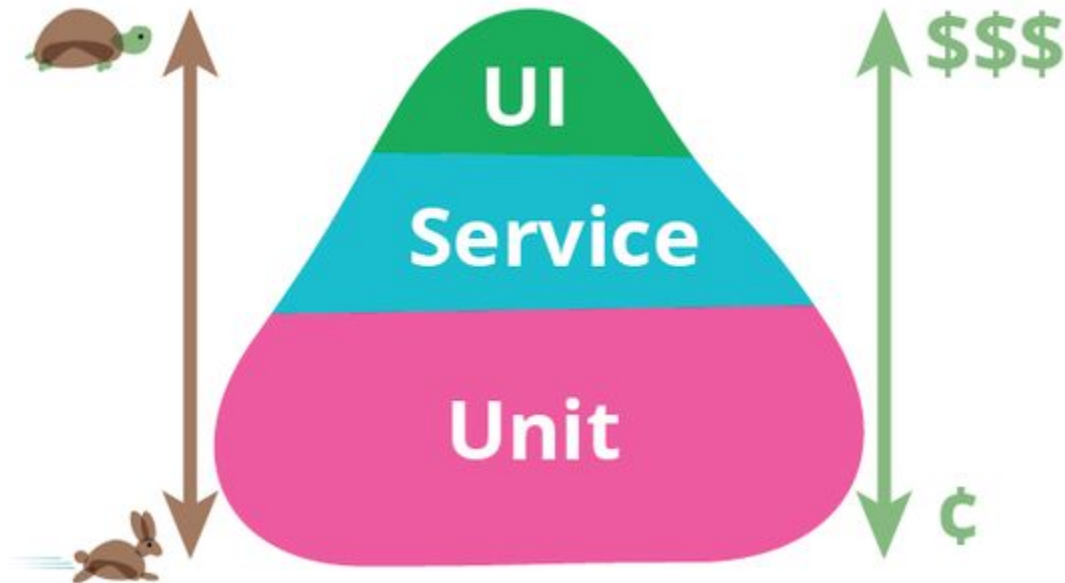
Pengenalan Software Testing



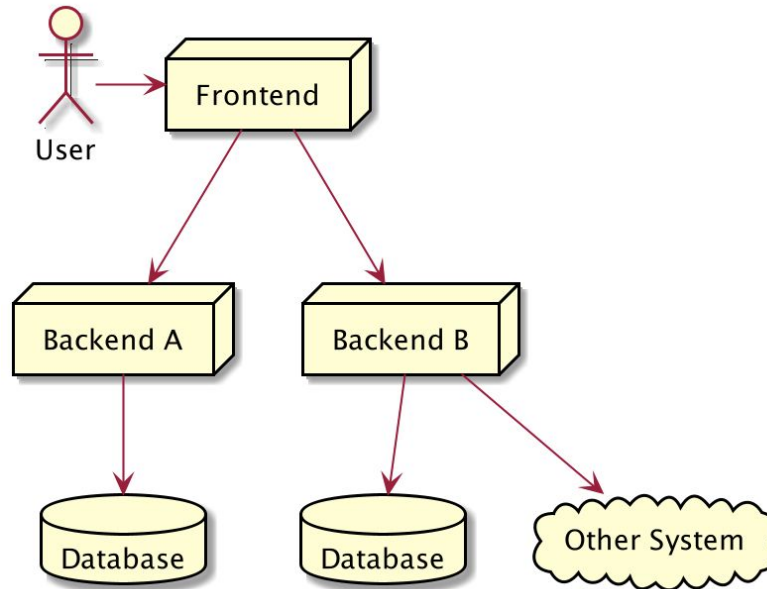
Pengenalan Software Testing

- Software testing adalah salah satu disiplin ilmu dalam software engineering
- Tujuan utama dari software testing adalah memastikan kualitas kode dan aplikasi kita baik
- Ilmu untuk software testing sendiri sangatlah luas, pada materi ini kita hanya akan fokus ke unit testing

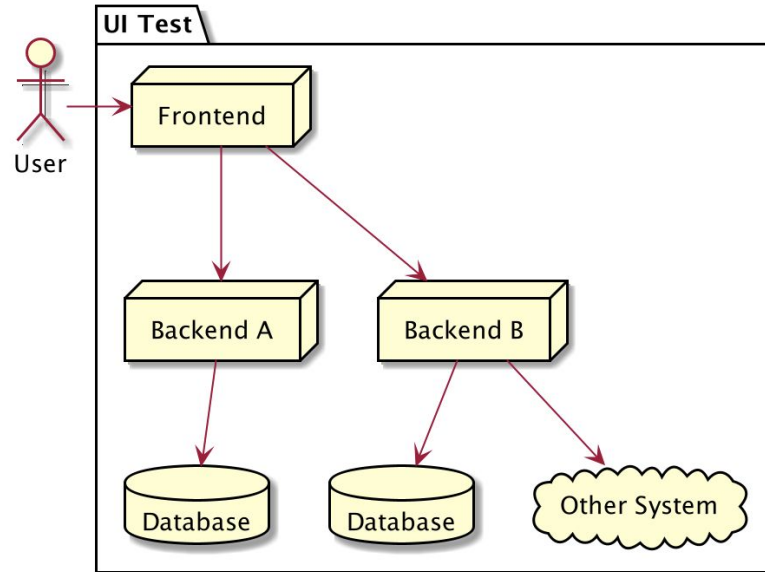
Test Pyramid



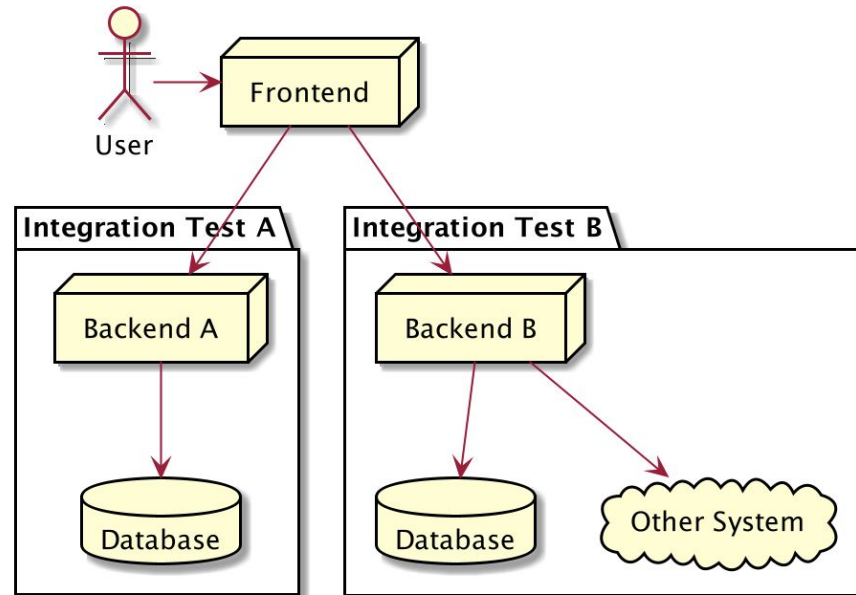
Contoh High Level Architecture Aplikasi



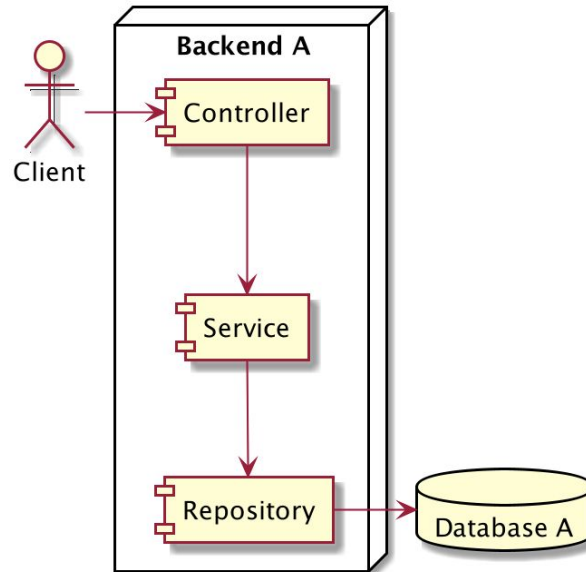
UI Test / End to End Test



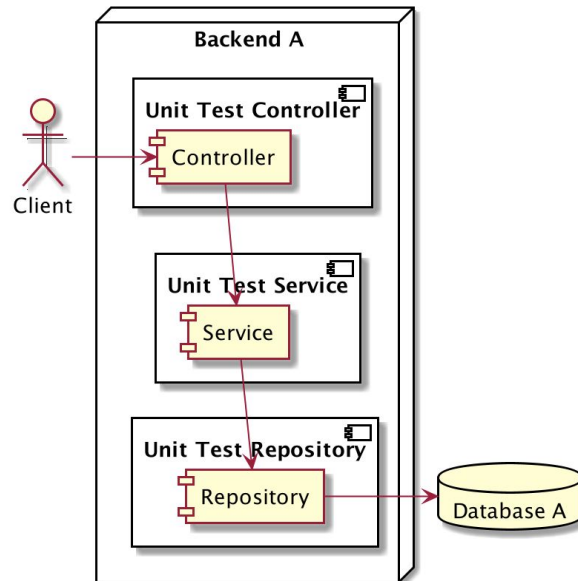
Service Test / Integration Test



Contoh Internal Architecture Aplikasi



Unit Test





Unit Test

- Unit test akan fokus menguji bagian kode program terkecil, biasanya menguji sebuah method
- Unit test biasanya dibuat kecil dan cepat, oleh karena itu biasanya kadang kode unit test lebih banyak dari kode program aslinya, karena semua skenario pengujian akan dicoba di unit test
- Unit test bisa digunakan sebagai cara untuk meningkatkan kualitas kode program kita

Membuat Project



Membuat Project

```
dart create --template console-simple belajar_dart_unit_test
```

Testing Package



Testing Package

- Dart memiliki package khusus untuk membuat unit test, yaitu package test
- Sebelum kita membuat unit test, kita perlu menambahkan package test terlebih dahulu ke project dart kita
- <https://pub.dev/packages/test>



Kode : Menambah Test Package

```
11
12     dev_dependencies:
13         lints: ^2.0.0
14         test: ^1.21.6
```

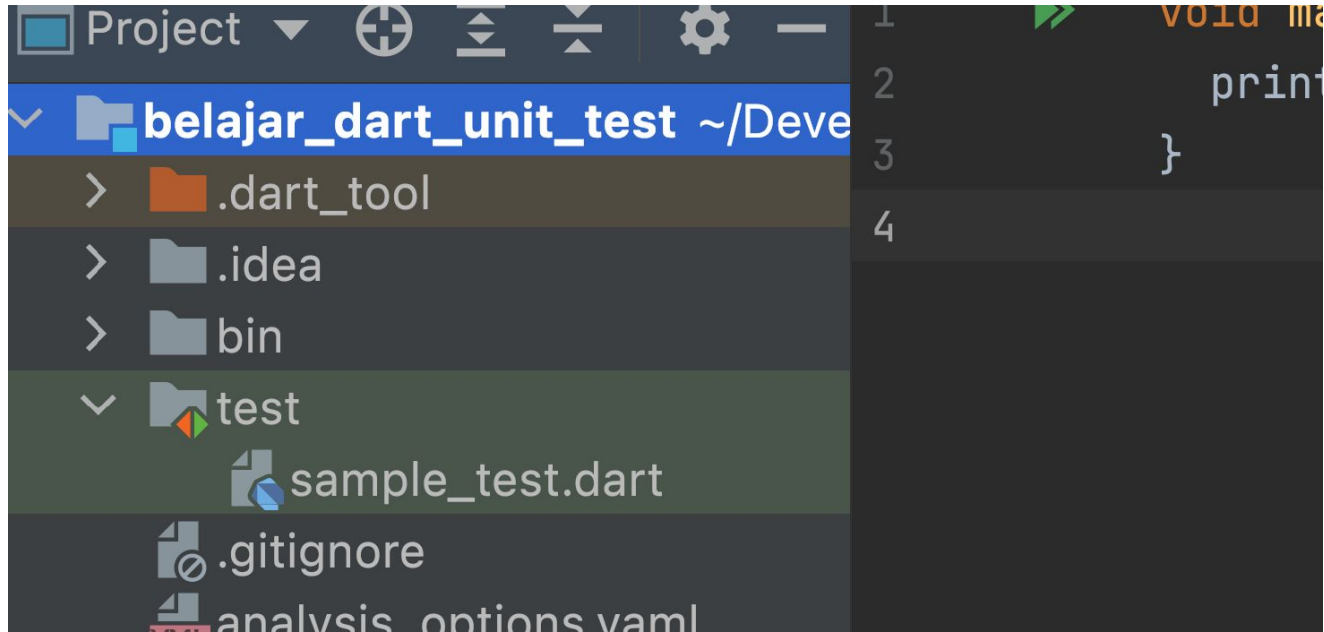
test Function



Membuat Test

- Dart memiliki standarisasi untuk struktur folder pembuatan unit test, biasanya dibuat menggunakan nama folder test
- Selanjutnya nama file dart biasanya akan memiliki akhiran `_test`, misal `contoh_test.dart`
- Sama seperti ketika membuat aplikasi, saat membuat file unit test, kita juga perlu menambahkan main function di file test tersebut

Folder Test





Test Function

- Untuk membuat unit test, kita bisa menggunakan function test yang terdapat di package test
- test function tersebut memiliki parameter nama function dan juga anonymous function yang berisikan kode unit test nya



Kode : Test Function

```
2
3     import 'package:test/test.dart';
4
5     void main() {
6         test("Contoh TEst", () => {
7             // kode unit test
8
9         });
10    }
11
```

Menjalankan Test



Menjalankan Test

- Untuk menjalankan file test, kita bisa menggunakan perintah :
`dart test test/namafile_test.dart`
- Atau jika kita ingin menjalankan semua unit file unit test, kita bisa gunakan perintah :
`dart test`
Secara otomatis dart akan menjalankan semua unit test di folder test



Kode : Menjalankan Test

Terminal: Local   

```
→ belajar_dart_unit_test dart test test/sample_test.dart
```

```
00:00 +1: All tests passed!
```

```
→ belajar_dart_unit_test dart test
```

```
00:00 +1: All tests passed!
```

```
→ belajar_dart_unit_test
```

expect Function



Expect Function

- Saat kita membuat unit test, salah satu yang paling penting adalah memastikan apakah data yang kita test benar atau salah
- Hal ini disebut assertion
- Di dart, untuk melakukan assertion, kita bisa menggunakan function expect, dimana terdapat dua parameter, pertama adalah data yang akan di test, dan kedua adalah harapan datanya
- Jika ternyata data tidak sesuai yang diharapkan, maka secara otomatis akan terjadi error yang menyebabkan unit test dianggap gagal



Kode : Expect Function

```
import 'package:test/test.dart';

String sayHello(String name){
  return "Hello $name";
}

void main() {
  test("Test Say Hello", () {
    var response = sayHello("Dart");
    expect(response, "Hello Dart");
  });
}
```

Matcher



Matcher Package

- Saat kita membuat assertion pada unit test, pengecekannya tidak hanya menggunakan kondisi equals, kadang ada kondisi lain
- Kita bisa menggunakan Matcher Package, dimana terdapat banyak sekali function dan constant yang bisa kita gunakan untuk mempermudah kita melakukan assertion
- <https://pub.dev/packages/matcher>
- Saat menginstall Test Package, Matcher Package otomatis terinstall, sehingga kita tidak perlu menginstalnya secara manual



Menggunakan Matcher

- <https://pub.dev/documentation/matcher/latest/matcher/Matcher-class.html>
- Cara penggunaan matcher, kita bisa menggunakannya pada parameter kedua pada expect function:

```
expect(value, matcher)
```



Kode : Matcher

```
4
5     test("Say Hello With Matcher", () {
6         expect(sayHello("Eko"), endsWith("Eko"));
7         expect(sayHello("Eko"), startsWith("Hello"));
8         expect(sayHello("Eko"), equalsIgnoringCase("hello eko"));
9     });
10 }
11
```

group Function



group Function

- `group()` function merupakan sebuah function yang bisa digunakan untuk melakukan grouping test function
- Melakukan grouping lebih bagus dilakukan ketika sebuah file unit test berisikan banyak sekali test function



Kode : group Function

```
int sum(int a, int b) => a + b;

>> void main() {
    group("Group", () {
        test("sum first", () {
            expect(sum(1, 2), equals(3));
        });
        test("sum second", () {
            expect(sum(10, 10), equals(20));
        });
    });
}
```

setUp Function



setUp Function

- setUp function merupakan sebuah function yang bisa kita gunakan untuk meregistrasi kode yang selalu dieksekusi setiap kali test function di eksekusi
- setUp function cocok sekali ketika kita butuh membuat sebuah kode yang berulang di awal test function
- Jika setUp function di tempatkan di dalam group function, maka setUp function ini hanya berlaku untuk test function di dalam group function tersebut



Kode : setUp Function

```
void main() {  
    var data = "Eko";  
  
    setUp(() {  
        data = "Eko";  
    });  
}
```

```
group("String", () {  
    test("string first", () {  
        data = "$data Kurniawan";  
        expect(data, "Eko Kurniawan");  
    });  
    test("string second", () {  
        data = "$data Khannedy";  
        expect(data, "Eko Khannedy");  
    });  
});
```

tearDown Function



tearDown Function

- Jika setUp function adalah function yang selalu dieksekusi di awal unit test, maka tearDown adalah function yang selalu dieksekusi di akhir unit test
- Cocok untuk melakukan hal yang harus dilakukan setelah semua unit test berjalan
- Jika tearDown function di tempatkan di dalam group function, maka tearDown function ini hanya berlaku untuk test function di dalam group function tersebut



Kode : tearDown Function

```
void main() {  
    var data = "Eko";  
  
    setUp(() {  
        data = "Eko";  
    });  
    tearDown(() {  
        print(data);  
    });  
}
```

```
group("String", () {  
    test("string first", () {  
        data = "$data Kurniawan";  
        expect(data, "Eko Kurniawan");  
    });  
    test("string second", () {  
        data = "$data Khannedy";  
        expect(data, "Eko Khannedy");  
    });  
});
```

setUpAll Function



setUpAll Function

- Jika setUp function akan dieksekusi di awal sebelum tiap test function di eksekusi
- setUpAll function hanya dieksekusi sekali saja, di awal sebelum semua test function di eksekusi
- Jika setUpAll function di tempatkan di dalam group function, maka setUpAll function ini hanya berlaku untuk test function di dalam group function tersebut



Kode : setUpAll Function

```
void main() {  
    var data = "Eko";  
  
    setUpAll(){  
        print("Start Unit Test");  
    };  
    setUp(() {  
        data = "Eko";  
    });  
    tearDown(() {  
        print(data);  
    });  
}
```

```
group("String", () {  
    test("string first", () {  
        data = "$data Kurniawan";  
        expect(data, "Eko Kurniawan");  
    });  
    test("string second", () {  
        data = "$data Khannedy";  
        expect(data, "Eko Khannedy");  
    });  
});
```

tearDownAll Function



tearDownAll Function

- Jika tearDown function akan dieksekusi di akhir setelah tiap test function di eksekusi
- tearDownAll function hanya dieksekusi sekali saja, di akhir setelah semua test function di eksekusi
- Jika tearDownAll function di tempatkan di dalam group function, maka tearDownAll function ini hanya berlaku untuk test function di dalam group function tersebut



Kode : tearDownAll Function

```
void main() {  
    var data = "Eko";  
  
    setUpAll(){  
        print("Start Unit Test");  
    };  
    setUp() {  
        data = "Eko";  
    };  
    tearDown() {  
        print(data);  
    };  
    tearDownAll() {  
        print("End Unit Test");  
    };  
}
```

```
group("String", () {  
    test("string first", () {  
        data = "$data Kurniawan";  
        expect(data, "Eko Kurniawan");  
    });  
    test("string second", () {  
        data = "$data Khannedy";  
        expect(data, "Eko Khannedy");  
    });  
});
```

Platform Selector



Platform Selector

- Pada beberapa kasus, mungkin kita membuat kode Dart yang dikhususkan untuk platform tertentu, misal untuk sistem operasi tertentu misalnya
- Kita bisa menandai sebuah unit test dengan annotation `@TestOn`, untuk memberi tahu bahwa unit test ini untuk platform yang kita sebutkan, sehingga jika kita menjalankan unit test tersebut di platform yang berbeda, secara otomatis unit test tersebut akan di skip
- Platform selector mendukung operator boolean, seperti `&&`, `||` dan `!`



Daftar Platform

- vm: test di command line Dart VM
- chrome: test di browser Chrome
- firefox: test di browser Firefox
- safari: test di browser Safari
- ie: test di browser Internet Explorer
- node: test di NodeJS
- browser: test di browser apapun
- js: test telah di compile ke JavaScript
- blink: test di browser yang memiliki blinkn render engine
- windows: test di os Windows
- mac-os: test di os Mac
- linux: test di os Linux
- android: test di os Android
- ios: test di os iOS
- posix: test di os POSIX



Kode : Platform Selector

```
@TestOn("windows || mac-os || linux")
```

```
import 'package:test/test.dart';
```

```
int sum(int a, int b) => a + b;
```

```
» void main() {  
  group("Group", () {  
    test("sum first", () {  
      expect(sum(1, 2), equals(3));  
    });  
  });  
}
```



Platform Selector Parameter

- Saat menggunakan annotation `@TestOn`, maka seluruh unit test di file tersebut akan menggunakan platform selector yang sudah ditentukan
- Kadang, kita hanya ingin beberapa test function atau group function saja misalnya, pada kasus ini kita bisa menambahkan parameter `testOn`



Kode : Platform Selector Parameter

```
void main() {  
    group("Group", () {  
        test("sum first", () {  
            expect(sum(1, 2), equals(3));  
        });  
        test("sum second", () {  
            expect(sum(10, 10), equals(20));  
        }, testOn: "windows");  
    });  
}
```

Skip Test



Skip Test

- Saat kita membuat unit test, kadang ada kalanya sebuah unit test bermasalah dan belum bisa diperbaiki
- Pada kasus ini, jangan hapus unit test tersebut, tapi tandai unit test tersebut agar tidak dijalankan, atau skip
- Kita bisa menggunakan annotation `@Skip` untuk menandai sebuah file unit test agar di skip



Kode : Skip File Unit Test

```
@Skip("This is unit test still broken")
```

```
import 'package:test/test.dart';
```



```
void main(){  
  group("Test", (){  
    test("test first", (){  
      expect(1, 2);  
    });  
    test("test second", (){  
      expect(1, 2);  
    });  
  });  
}
```



Skip Parameter

- Menggunakan Annotation `@Skip` secara otomatis akan melakukan skip seluruh test di file tersebut
- Kadang, ada kalanya kita hanya ingin melakukan skip satu buah test function atau group function
- Kita bisa menambahkan object parameter skip pada test function dan group function



Kode : Skip Parameter

```
» void main(){  
    group("Test", (){  
        test("test first", (){  
            expect(1, 2);  
        }, skip: "this test is broken");  
        test("test second", (){  
            expect(1, 2);  
        });  
        test("test third", (){  
            expect(1, 2);  
        });  
    });  
}
```

Tag



Tag

- Saat membuat unit test yang banyak, kadang kita ingin menambahkan tag terhadap unit test nya
- Biasanya ini digunakan sebagai penanda untuk tag
- Kita bisa menggunakan annotation @Tags, atau menggunakan named parameter tags di test atau group function
- Salah satu keuntungan menambahkan tag, kita bisa meminta Dart untuk menjalankan unit test dengan tag tertentu saja misalnya, dengan perintah :
dart test --tags "tag dengan boolean selector"



Kode : Tag

```
@Tags(["pzn", "eko"])  
import 'package:test/scaffolding.dart';
```

```
» void main() {  
    test("first", () {  
        print("first");  
    }, tags: ["first"]);  
    test("second", () {  
        print("second");  
    }, tags: ["second"]);  
}
```



Menjalankan Unit Test Tag Tertentu

```
→ belajar_dart_unit_test dart test --tags "first || second"
00:01 +0: loading test/tag_test.dart
Warning: Tags were used that weren't specified in dart_test.yaml.
  pzn was used in the suite itself
  eko was used in the suite itself

00:01 +0: test/tag_test.dart: first
first
00:01 +1: test/tag_test.dart: second
second
00:01 +2: All tests passed!
```

Retry Test



Retry Test

- Saat membuat unit test, kadang ada kalanya unit test tersebut tidak stabil, misal butuh konek ke database atau ke sistem lain
- Hal ini menyebabkan kadang unit test sering gagal, bukan karena kode salah, tapi karena faktor seperti koneksi network, dan lain-lain
- Dart memiliki fitur untuk melakukan retry ketika unit test gagal dilakukan
- Kita bisa menggunakan annotation `@Retry` dan secara otomatis unit test akan di retry sejumlah yang kita tentukan



Kode : Retry

```
@Retry(10)
```

```
import 'package:test/test.dart';
```

```
» void main(){  
  test("test failed", (){  
    expect(1, 2);  
  });  
}
```



Retry Parameter

- Jika kita hanya ingin melakukan retry pada test atau group function tertentu, kita juga bisa menambahkan named parameter retry



Kode : Retry Parameter

```
import 'package:test/test.dart';

>> void main(){
    test("test failed", (){
        expect(1, 2);
    }, retry: 5);
}
```

Platform Specific Configuration



Platform Specific Configuration

- Saat menggunakan `@Skip`, secara otomatis tidak akan melihat platform apapun, dia akan di skip secara otomatis
- Bagaimana jika kasusnya misalnya, kita ingin melakukan Skip, namun hanya untuk platform “mac-os” saja?
- Pada kasus seperti ini, kita bisa menggunakan annotation `@OnPlatform`



Kode : Platform Specific

```
@OnPlatform({  
  "mac-os" : Skip("This test is not supported on macOS"),  
})
```

```
import 'package:test/test.dart';
```

```
int sum(int a, int b) => a + b;
```

```
>> void main() {  
  group("Group", () {  
    test("sum", () {
```




Platform Specific Parameter

- Sama dengan annotation lainnya, jika misal kita hanya ingin menambahkan konfigurasi platform pada test atau group tertentu, kita bisa tambahkan named parameter onPlatform

Kode : Platform Specific Parameter

```
➤ void main() {  
    group("Group", () {  
        test("sum first", () {  
            expect(sum(1, 2), equals(3));  
        }, onPlatform: {  
            "mac-os" : Skip("This test is not supported on macOS"),  
        });  
        test("sum second", () {  
            expect(sum(10, 10), equals(20));  
        });  
    });  
}
```

Package Configuration



Package Configuration

- Sebelumnya, kita hanya melakukan pengaturan di file unit test dart saja
- Apa yang terjadi jika misal kita butuh pengaturan yang sama, untuk semua file unit test? Misal kita ingin semua hanya jalan di windows, atau semua memiliki tag, dan lain-lain. Maka, kita harus lakukan satu persatu di file dart nya
- Untungnya, dart memiliki package configuration file untuk unit test, dimana kita bisa gunakan file yaml untuk menambah informasi unit test di seluruh package hanya dengan satu file, yaitu `dart_test.yaml`



Configuration Format

- Banyak sekali yang bisa kita atur di package configuration file
- Kita bisa melihat detail nya di dokumentasi resminya
- <https://github.com/dart-lang/test/blob/master/pkg/test/doc/configuration.md>



Kode : Package Config File

```
1
2  test_on: "mac-os"
3
4  tags:
5      second:
6          skip: "Second tag are skipped"
7
8
```

Mock Object



Mock Object

- Saat kita membuat unit test, kadang tidak semua object bisa kita test
- Contoh, misal kita memiliki object yang harus mengirim API Request ke Payment Gateway, atau sistem lain yang diluar kontrol kita
- Pada kasus seperti ini, kita tidak bisa memaksakan untuk membuat unit test yang mengirim request ke sistem lain, karena jika kita paksakan, bisa jadi nanti hasil unit testnya tidak konsisten
- Pada kasus seperti ini, kita bisa menggunakan konsep bernama Mock Object, yaitu membuat object tiruan yang bisa kita atur tingkah lakunya agar sesuai dengan yang kita inginkan



Contoh Kasus

- Sebelum belajar Mock Object, sekarang kita akan coba buat contoh kasus, misal kita memiliki class dengan nama BookService dan BookRepository
- BookRepository merupakan class yang berisikan kode untuk memanipulasi data Book ke dalam sistem lain, misal database atau API lain
- BookService merupakan class yang berisikan kode untuk business logic aplikasi kita
- BookService memiliki method yang bisa digunakan untuk membuat Book, mengubah Book, mengambil Book dan menghapus Book



Kode : Book

```
class Book {  
    String id;  
    String name;  
    int price;  
  
    Book(this.id, this.name, this.price);  
}
```

```
@override  
bool operator ==(Object other) =>  
    identical(this, other) ||  
    other is Book &&  
        runtimeType == other.runtimeType &&  
        id == other.id &&  
        name == other.name &&  
        price == other.price;  
  
@override  
int get hashCode => id.hashCode ^ name.hashCode ^ price.hashCode;  
}
```



Kode : Book Repository

```
class BookRepository {  
    void save(Book book) {  
        print("Save $book");  
        throw UnsupportedError("s  
    }  
  
    void update(Book book) {  
        print("Update $book");  
        throw UnsupportedError("u  
    }  
}
```

```
void delete(Book book) {  
    print("Delete $book");  
    throw UnsupportedError("dele  
}  
  
Book findById(String id) {  
    print("Find by id $id");  
    throw UnsupportedError("find  
}  
}
```



Kode : Book Service (1)

```
class BookService {
    BookRepository bookRepository;

    BookService(this.bookRepository);

    void save(String id, String name, int price) {
        if (id == "" || name == "" || price <= 0) {
            throw Exception("Invalid data");
        }
        bookRepository.save(Book(id, name, price));
    }
}
```

```
void update(String id, String name, int price) {
    if (id == "" || name == "" || price <= 0) {
        throw Exception("Invalid data");
    }
    var book = bookRepository.findById(id);
    if (book == null) {
        throw Exception("Book not found");
    } else {
        book.name = name;
        book.price = price;
        bookRepository.update(book);
    }
}
```



Kode : Book Service (2)

```
Book find(String id) {  
    var book = bookRepository.findById(id);  
    if (book == null) {  
        throw Exception("Book not found");  
    } else {  
        return book;  
    }  
}
```

```
void delete(String id) {  
    var book = bookRepository.findById(id);  
    if (book == null) {  
        throw Exception("Book not found");  
    } else {  
        bookRepository.delete(book);  
    }  
}
```



Kode : Book Library

```
library belajar_dart_unit_test;  
  
export 'package:belajar_dart_unit_test/src/book.dart';  
export 'package:belajar_dart_unit_test/src/book_repository.dart';  
export 'package:belajar_dart_unit_test/src/book_service.dart';
```



Mockito



Mockito

- Dart sudah menyediakan package khusus untuk membuat mock object yaitu Mockito
- <https://pub.dev/packages/mockito>
- Mockito merupakan package yang terinspirasi dari library mock object Java dengan nama sama, yaitu Mockito <https://github.com/mockito/mockito>



Menginstall Mockito

- Sebelum kita menggunakan Mockito, silahkan tambahkan package mockito terlebih dahulu
- Selain itu, mockito juga membutuhkan package build_runner, jadi pastikan kita menambahkan package build_runner juga
- https://pub.dev/packages/build_runner



Kode : Menambah Mockito Package

```
dev_dependencies:  
  lints: ^2.0.0  
  test: ^1.21.6  
  mockito: ^5.3.1  
  build_runner: ^2.2.1
```

Membuat Mock Object



Membuat Mock Object

- Cara kerja Mockito adalah dengan membuat generated file berisi mock class yang bisa kita gunakan sebagai pengganti class aslinya
- Pertama, kita perlu memberitahu Mockito untuk membuat mock object dengan menggunakan annotation `GenerateNiceMocks`, dan lakukan import ke file dengan format `namafilename_test.mocks.dart`
- Selanjutnya, kita perlu menjalankan `build_runner`, agar file mock otomatis dibuatkan oleh Mockito



Kode : Generate Nice Mock

```
import 'package:belajar_dart_unit_test/book.dart';
import 'package:mockito/annotations.dart';
import 'package:test/test.dart';


@GenerateNiceMocks([MockSpec<BookRepository>()])
import 'mock_object_test.mocks.dart';

void main(){

}
```



Kode : Menjalankan Build Runner



```
→ belajar_dart_unit_test dart run build_runner build
[INFO] Generating build script completed, took 460ms
[INFO] Reading cached asset graph completed, took 49ms
[INFO] Checking for updates since last build completed, took 429ms
[INFO] Running build completed, took 2.8s
[INFO] Caching finalized dependency graph completed, took 20ms
[INFO] Succeeded after 2.8s with 1 outputs (1 actions)
→ belajar_dart_unit_test █
```

Kode : Hasil Mock Object File

```
/// A class which mocks [BookRepository].  
///  
/// See the documentation for Mockito's code generation for more information.  
class MockBookRepository extends _i1.Mock implements _i2.BookRepository {  
    @override  
    void save(_i3.Book? book) => super.noSuchMethod(  
        Invocation.method(  
            #save,  
            [book],  
        ),  
        returnValueForMissingStub: null,  
    );  
    @override  
    void update(_i3.Book? book) => super.noSuchMethod(  
        Invocation.method(  

```



Kode : Test Mock Object

```
void main(){
  group("BookService", (){
    var bookRepository = MockBookRepository();
    var bookService = BookService(bookRepository);

    test("Save new book must success", (){
      bookService.save("1", "Tutorial Dart", 100000);
    });
  });
}
```

Verifikasi Mock Object



Verifikasi Mock Object

- Saat kita membuat unit test menggunakan Mock Object, yang perlu diperhatikan adalah, kita harus bisa memastikan bahwa Mock Object tersebut benar-benar dipanggil, karena jika tidak, risikonya unit test kita menjadi tidak valid
- Contoh, misal kita hapus kode yang memanggil BookRepository di BookService, dan lihat apa yang terjadi



Kode : Book Service

```
class BookService {  
    BookRepository bookRepository;  
  
    BookService(this.bookRepository);  
  
    void save(String id, String name, int price) {  
        if (id == "" || name == "" || price <= 0) {  
            throw Exception("Invalid data");  
        }  
        // bookRepository.save(Book(id, name, price));  
    }  
}
```



Kenapa Unit Test Sukses?

- Walaupun kode BookRepository sudah dihapus, namun unit test tetap sukses, hal ini dikarenakan kita tidak melakukan verifikasi apapun pada unit test nya
- Oleh karena itu, ketika menggunakan Mock Object, biasakan kita selalu melakukan verifikasi interaksi yang terjadi
- Kita bisa menggunakan function verify untuk memastikan bahwa Mock Object dipanggil



Kode : Verifikasi Mock Object

```
group("BookService", (){  
    var bookRepository = MockBookRepository();  
    var bookService = BookService(bookRepository);  
  
    test("Save new book must success", (){  
        bookService.save("1", "Tutorial Dart", 100000);  
        verify(bookRepository.save(Book("1", "Tutorial Dart", 100000)));  
    });  
});
```



Called Function

- Selain melakukan verifikasi terhadap mock object, kita juga wajib memastikan jumlah eksekusi kode mock object
- Misal, saat save data Book, kita harus pastikan bahwa, `BookRepository.save()` hanya dipanggil satu kali
- Kita bisa menggunakan `called()` function setelah melakukan verify



Kode : Called Function

```
group("BookService", () {  
    var bookRepository = MockBookRepository();  
    var bookService = BookService(bookRepository);  
  
    test("Save new book must success", () {  
        bookService.save("1", "Tutorial Dart", 100000);  
        verify(bookRepository.save(Book("1", "Tutorial Dart", 100000))).called(1);  
    });  
});
```

Stubing



Stubing

- Stubing adalah mengubah tingkah laku mock object sebelum digunakan
- Saat kita membuat mock object, kadang beberapa function butuh mengembalikan value misalnya
- Kita bisa menambahkan tingkah laku ke mock object ketika mock object tersebut dipanggil, dengan menggunakan when function
- Selanjutnya, kita bisa menentukan reaksi apa yang perlu dilakukan oleh mock stub tersebut dengan menggunakan thenReturn, thenAnswer atau thenThrow function



Kode : Stubing

```
test("Find book by id", () {  
    when(bookRepository.findById("1"))  
        .thenReturn(Book("1", "Tutorial Dart", 100000));  
  
    var book = bookService.find("1");  
    expect(book, equals(Book("1", "Tutorial Dart", 100000)));  
  
    verify(bookRepository.findById("1")).called(1);  
});
```

Argument Matcher



Argument Matcher

- Mockito juga mendukung Matcher ketika membuat stubbing atau melakukan verifikasi
- Kita bisa menggunakan constant any jika ingin menerima argument apapun, atau bisa menggunakan `argThat(matcher)` jika ingin menggunakan matcher



Kode : Argument Matcher

```
test("Find book by id eko123", () {  
    when(bookRepository.findById(argThat(startsWith("eko"))))  
        .thenReturn(Book("eko123", "Tutorial Dart", 100000));  
  
    var book = bookService.find("eko123");  
    expect(book, equals(Book("eko123", "Tutorial Dart", 100000)));  
  
    verify(bookRepository.findById(any)).called(1);  
});  
});
```

Best Practice Mock Object



Best Practice Mock Object

- Testing menggunakan real object lebih baik dibandingkan menggunakan mock object
- Selama kita bisa membuat real objectnya, selalu gunakan real object, dibandingkan menggunakan mock object
- Gunakan mock object, hanya ketika kita tidak bisa membuat real object nya di unit test
- Jika menggunakan mock object, pastikan selalu melakukan verifikasi interaksi yang terjadi terhadap mock object tersebut

Materi Selanjutnya



Materi Selanjutnya

- Dart Async