

MENYELAMI FRAMEWORK Laravel^{5.2}

panduan komprehensif dan aplikatif untuk
menguasai framework laravel

RAHMAT AWALUDIN

Menyelami Framework Laravel

Panduan komprehensif dan aplikatif untuk menguasai framework Laravel.

Rahmat Awaludin

Buku ini dijual di <http://leanpub.com/bukularavel>

Versi ini diterbitkan pada 2016-12-14



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2014 - 2016 Rahmat Awaludin

Tweet Buku Ini!

Bantulah Rahmat Awaludin dengan mewartakan buku ini via [Twitter!](#)

Tagar yang disarankan untuk buku ini adalah [#bukularavel](#).

Temukan kata orang tentang buku ini dengan mengklik tautan ini untuk menampilkan tagar ini di Twitter:

<https://twitter.com/search?q=#bukularavel>

Juga Oleh **Rahmat Awaludin**

Seminggu Belajar Laravel

Untuk istriku tercinta, Irna Rahayu dan jagoan kecilku, Shidqi Abdullah Mubarak.

Contents

Konsep Dasar	1
PHP5 Autoloader	1
PHP5 Abstract dan Interfaces	4
Mengakses Database	24
Chunk, memproses banyak data dengan lebih efisien memory	24
Routing, Kendalikan Alur Aplikasi	27
Subdomain Routing	27

Konsep Dasar

PHP5 Autoloader

Sebagai seorang veteran di pemrograman PHP (*ehm*), tentu sudah sangat familiar dengan syntax `include` untuk memasukkan syntax dari file lain ke file yang sedang aktif. Biasanya, syntax ini digunakan jika kita hendak menggunakan class yang berada pada file yang lain. Perhatikan contoh syntax berikut:

~/Code/autoloader-oh-autoloader/Printer.php

```
<?php
class Printer {
    public function cetakBuku($buku) {
        echo "Class " . __CLASS__ . " : ";
        echo "Mencetak buku $buku\n";
        return "Buku $buku";
    }
}
```

~/Code/autoloader-oh-autoloader/Kurir.php

```
<?php
class Kurir {
    public function kirim($file, $tujuan) {
        echo "Class " . __CLASS__ . " : ";
        echo "Mengirim $file ke $tujuan\n";
    }
}
```

~/Code/autoloader-oh-autoloader/index.php

```
<?php
include 'Printer.php';
$printer = new Printer();
$buku = $printer->cetakBuku('Menyelami Framework Laravel');

include 'Kurir.php';
$kurir = new Kurir();
$kurir-> kirim($buku, 'Bandung');
```

Pada syntax ini, terlihat kita memiliki dua buah class yaitu Printer dan Kurir.

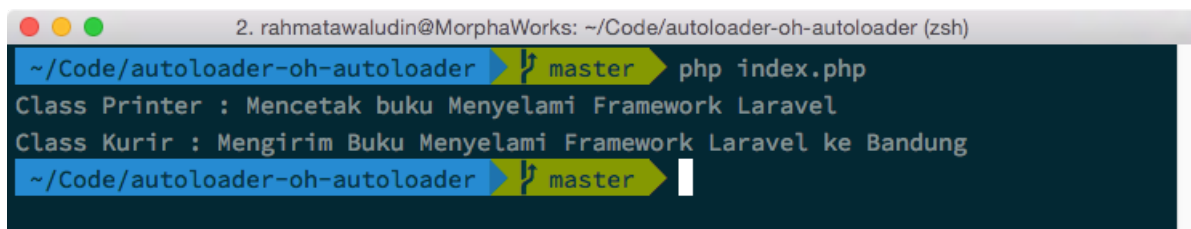
Pada Class Printer, terdapat method cetakBuku() yang berfungsi menampilkan tulisan “Mencetak Buku” dengan nama buku yang menjadi parameter. Ketika kita mengambil method ini, kita juga memanggil nama Class yang aktif dengan syntax echo "Class " . __CLASS__ . " : ".

Pada Class Kurir, terdapat method kirim() yang berfungsi menampilkan tulisan “Mengirim \$file ke \$tujuan” dengan \$file dan \$tujuan merupakan parameter dari method ini. Kita juga menampilkan nama Class yang aktif ketika memanggil method ini.

Perhatikan file index.php.

Di baris ke-2 dan ke-6 kita menggunakan syntax include untuk memasukan file Printer.php dan Kurir.php.

Ini merupakan cara lama agar kita dapat membuat object dari class Printer dan Kurir yang berada di kedua file tersebut. Syntax selanjutnya adalah syntax biasa untuk membuat object Printer dan Kurir kemudian memanggil method pada masing-masing object tersebut. Output dari syntax ini seperti ini:



Tanpa Autoloader

Sip. Seperti yang kita lihat, syntax ini berjalan sebagaimana mestinya. Dan cara inilah yang sering digunakan dari dulu (bahkan sampai sekarang masih dipakai di Wordpress) untuk memanggil Class dari file yang berbeda.



Cara ini, meskipun dapat digunakan, tetapi kurang efektif. Bayangkan, jika kita memiliki 100 Class di 100 file yang berbeda, apakah mau membuat 100 statement include?

Tentu tidak. Dan para pendahulu kita yang telah lebih dulu memahami PHP pun memikirkan hal tersebut. Maka, lahirlah fitur autoloader di PHP. Dengan fitur ini, kita tidak perlu menulis `include` untuk setiap file PHP yang akan di masukkan ke file.

Untuk menggunakan autoloader kita akan menggunakan fungsi `spl_autoload_register()`¹. Fungsi ini menerima parameter closure/fungsi yang memiliki sebuah parameter `$class` yang berisi nama class yang akan dipanggil. Di dalam closure ini, kita melakukan `include` ke class yang diinginkan.

Mari kita praktekan, ubah file `index.php` menjadi :

~/Code/autoloader-oh-autoloader/index.php

```
<?php
spl_autoload_register(function ($class) {
    include $class . '.php';
});

$printer = new Printer();
$buku = $printer->cetakBuku('Menyelami Framework Laravel');

$kurir = new Kurir();
$kurir->kirim($buku, 'Bandung');
```

Pada perubahan ini kita menghapus dua statement `include` dan menambahkan syntax:

~/Code/autoloader-oh-autoloader/index.php

```
<?php
spl_autoload_register(function ($class) {
    include $class . '.php';
});
```

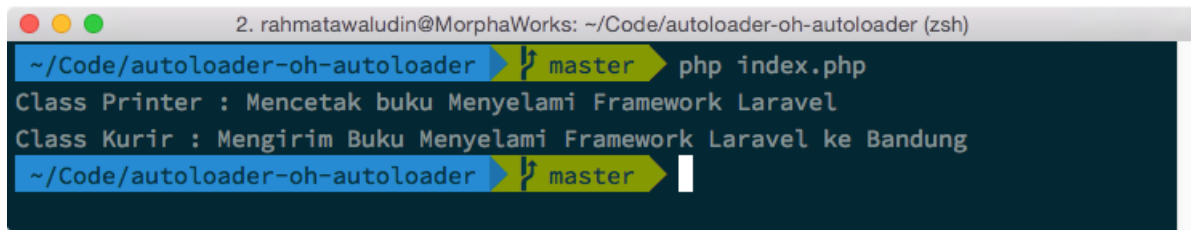
Yang dilakukan syntax ini adalah melakukan `include` setiap kali kita melakukan `new Class()`.

Terlihat di baris ke-3, kita menggunakan `include $class . '.php';`.

Ketika code berjalan (runtime), misalnya kita memanggil `new Printer()`, maka syntax ini akan berubah menjadi `include Printer.php`; yang akan memasukkan konten file tersebut ke code yang sedang aktif. Itulah alasan kita membuat sebuah file untuk sebuah class. Dengan cara ini, kita tidak perlu lagi melakukan `include` manual untuk tiap Class yang dibutuhkan. *Keren kan?*

Kalau dijalankan, outputnya akan tetap sama.

¹<http://php.net/manual/en/function.spl-autoload-register.php>



```
2. rahmatawaludin@MorphaWorks: ~/Code/autoloader-oh-autoloader (zsh)
~/Code/autoloader-oh-autoloader master php index.php
Class Printer : Mencetak buku Menyelami Framework Laravel
Class Kurir : Mengirim Buku Menyelami Framework Laravel ke Bandung
~/Code/autoloader-oh-autoloader master
```

Setelah Autoloader

Tentunya, autoloader ini dapat disesuaikan dengan kebutuhan kita. Misalnya, semua class berada di folder class dan berakhiran `.inc.php`, maka syntax autoloader berubah menjadi :

Merubah autoloader

```
<?php
spl_autoload_register(function ($class) {
    include 'class/' . $class . '.inc.php';
});
```

Laravel sangat aktif menggunakan Autoloader ini. Dengan memahami dasar dari Autoloader ini, mudah-mudahan kita tidak tenggelam dalam kebingungan ketika pembahasan tentang Laravel semakin dalam. Sip.



Source code dari latihan ini bisa didapat di <https://github.com/rahmatawaludin/autoloader-oh-autoloader>²

PHP5 Abstract dan Interfaces

Memahami Abstract dan Interfaces sangat penting untuk dapat mendalami framework Laravel. Jika pernah mempelajari mempelajari OOP di PHP, pasti telah memahami Class dan Inheritance. Jika Class dan Inheritance adalah nasi dan sayur asem, maka Abstract dan Interfaces adalah ikan asin dan sambelnya. Nah. :D

Abstract

Oke. Kita mulai dari Abstract. *Apa itu Abstract?*

Abstrak adalah tipe yang tidak dapat dipakai secara langsung. (Wikipedia)

Maksudnya, Abstract itu adalah semacam class di PHP tapi tidak bisa langsung dibuat objectnya. Misalnya, sebuah tombol. Kita semua pasti tahu, bahwa tombol apapun pasti bisa ditekan. Hanya saja, tiap tindakan yang terjadi ketika kita menekan tombol akan berbeda, tergantung jenis tombolnya. Perhatikan contoh ini:

²<https://github.com/rahmatawaludin/autoloader-oh-autoloader/commits/master>

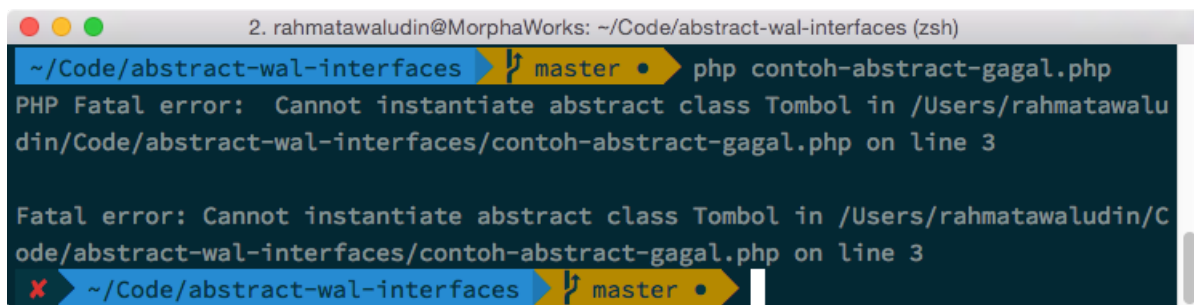
~/Code/abstract-wal-interfaces/Tombol.php

```
<?php
abstract class Tombol {
    abstract public function tekan();
}
```

~/Code/abstract-wal-interfaces/contoh-abstract-gagal.php

```
<?php
include 'Tombol.php';
$tombol = new Tombol();
$tombol->tekan();
```

Terlihat disini, kita langsung mencoba membuat object (*instantiate*) dari abstract class Tombol. Maka, akan ada error seperti ini:



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php contoh-abstract-gagal.php
PHP Fatal error: Cannot instantiate abstract class Tombol in /Users/rahmatawaludin/Code/abstract-wal-interfaces/contoh-abstract-gagal.php on line 3

Fatal error: Cannot instantiate abstract class Tombol in /Users/rahmatawaludin/Code/abstract-wal-interfaces/contoh-abstract-gagal.php on line 3
~/Code/abstract-wal-interfaces master •
```

Class Abstract tidak bisa langsung dibuat object

Ini menunjukkan bahwa abstract tidak bisa langsung di-*instantiate* menjadi object. Untuk membuat object, kita perlu membuat class baru yang merupakan turunan dari abstract class Tombol ini. Perhatikan syntax ini:

~/Code/abstract-wal-interfaces/TombolLogin.php

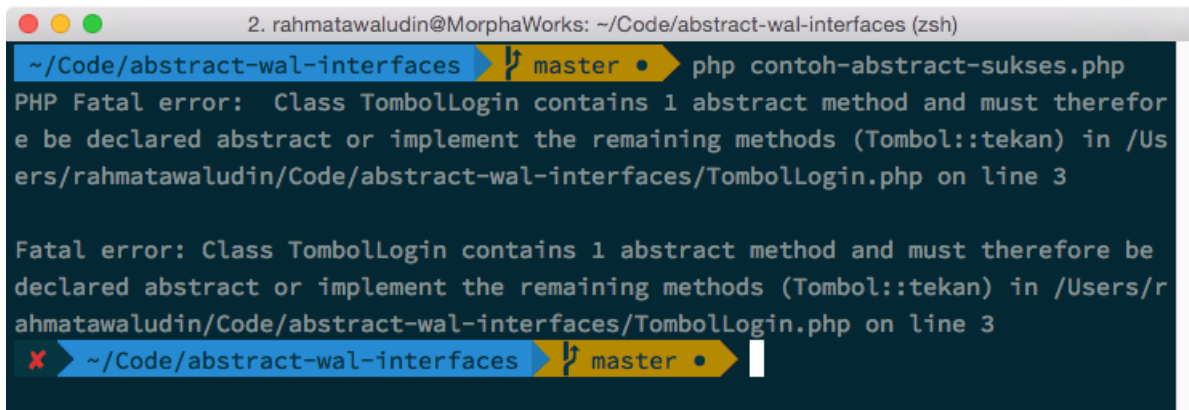
```
<?php
include "Tombol.php";
class TombolLogin extends Tombol {

}
```

~/Code/abstract-wal-interfaces/contoh-abstract-sukses.php

```
<?php
include "TombolLogin.php";
$tombol = new TombolLogin();
$tombol->tekan();
```

Disini kita membuat class baru bernama TombolLogin yang meng-*extend* abstract class Tombol. Kalau kita jalankan :



Gagal meng-extends interface, karena ada method yang belum diimplementasikan

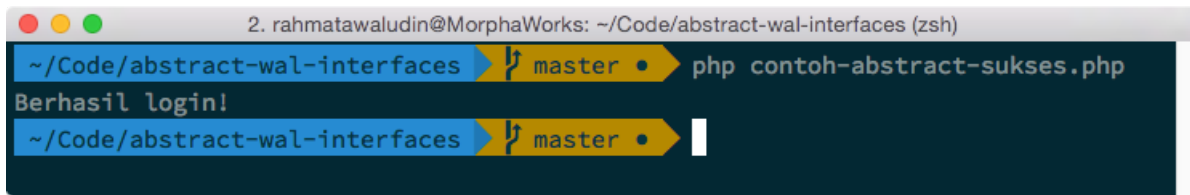
Uuuppss... Dia error lagi.

Ini terjadi karena, kita belum mengimplementasikan method abstract yaitu method tekan() yang ada di abstract class Tombol. Mari kita perbaiki:

~/Code/abstract-wal-interfaces/TombolLogin.php

```
<?php
include "Tombol.php";
class TombolLogin extends Tombol {
    public function tekan() {
        echo "Berhasil login!\n";
    }
}
```

Kalau kita jalankan lagi:



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ▶ master • php contoh-abstract-sukses.php
Berhasil login!
~/Code/abstract-wal-interfaces ▶ master •
```

Berhasil menggunakan abstract class Tombol

Tuhh.. dia berhasil kan? :)

Ini penting. Jadi, abstract itu sangat berguna kalau kita ingin memastikan bahwa suatu method selalu tersedia pada class, apapun implementasinya. Dalam contoh ini, kita selalu bisa memanggil method `tekan()` apapun jenis tombolnya. Biar lebih paham, kita tambah lagi contohnya. Kali ini, ceritanya kita mau bikin tombol untuk meluncurkan bom nuklir. Perlu dicatat, ini hanya contoh, tidak ada yang bom yang diluncurkan dalam pembuatan contoh ini. Perhatikan syntax ini:

~/Code/abstract-wal-interfaces/TombolNuklir.php

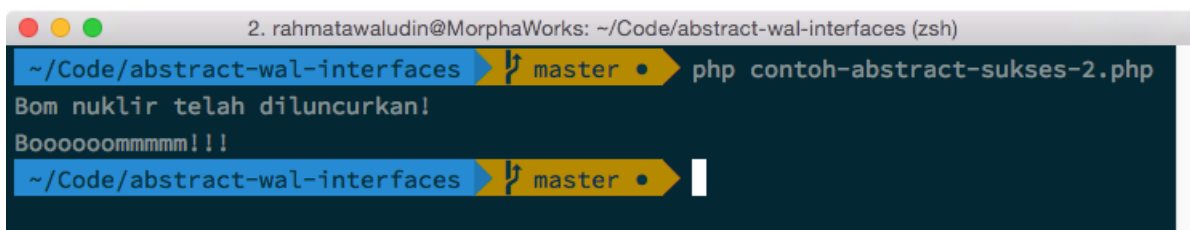
```
<?php
include "Tombol.php";
class TombolNuklir extends Tombol {
    public function tekan() {
        echo "Bom nuklir telah diluncurkan!\n";
        sleep(3);
        echo "Boooooommmmm!!!\n";
    }
}
```

Untuk menjalankan tombol ini, kita tetap menggunakan method yang sama, yaitu `tekan()`:

~/Code/abstract-wal-interfaces/contoh-abstract-sukses-2.php

```
<?php
include "TombolNuklir.php";
$tombol = new TombolNuklir();
$tombol->tekan();
```

Kalau dijalankan..



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ▶ master • php contoh-abstract-sukses-2.php
Bom nuklir telah diluncurkan!
Boooooommmmm!!!
~/Code/abstract-wal-interfaces ▶ master •
```

Berhasil membuat Tombol Nuklir

Boooooommmmm!!!

Oke, saya rasa sudah cukup penjelasannya. Pertanyaanya, *kapan biasanya teknik abstract class ini digunakan?*

Contohnya banyak, salah satunya saya contohkan dengan dependency injection. Teknik ini memudahkan kita untuk memasukkan (*inject*) class yang kita butuhkan pada class yang sedang digunakan.

Contoh dependency injection, misalnya seorang **Pembeli** *harus* punya kartu **BNI** untuk melakukan pembayaran. Bisa kita implementasikan dengan meng-*inject* class BNI ke class Pembeli. Seperti ini:

~/Code/abstract-wal-interfaces/BNI.php

<?php

```
class BNI {

    private $saldo;

    public function __construct($pin) {
        // ceritanya cek PIN ke database
        if ($pin == '12345') {
            echo "Berhasil mengaktifkan Kartu BNI!\n";
        } else {
            $pesan = "PIN yang Anda masukkan salah :(";
            throw new Exception($pesan);
        }
    }

    private function catatTransaksi($jenis, $jumlah) {
        echo "Mencatat transaksi $jenis sejumlah $jumlah ke Buku Tabungan.\n";
    }

    public function kredit($jumlah) {
        $this->catatTransaksi('transfer keluar', $jumlah);
        $this->saldo -= $jumlah;
    }

    public function deposit($jumlah) {
        $this->catatTransaksi('deposit dana', $jumlah);
        $this->saldo += $jumlah;
    }

    public function cekSaldo() {
        return $this->saldo;
    }
}
```

```

    }
}

```

Class BNI ini mempunyai:

- attribute `$saldo` yang berfungsi mencatat saldo terakhir.
- method `__construct()` yang berfungsi membangun object BNI, di method ini kita mengharuskan input PIN. Dalam prakteknya, tentu saja PIN ini disimpan di database, tapi disini kita sederhanakan dengan menyimpannya langsung di method ini.
- method `kredit()` yang berfungsi untuk mengurangi jumlah saldo.
- method `deposit()` yang berfungsi untuk menambah jumlah saldo.
- method `cekSaldo()` yang berfungsi mengecek jumlah saldo terkini.

Mari kita buat class Pembeli, Class ini akan membutuhkan class BNI :

~/Code/abstract-wal-interfaces/Pembeli-DI.php

```

<?php
include "BNI.php";
class Pembeli {
    private $nama;
    private $bni;

    public function __construct($nama = "Seseorang", BNI $bni) {
        $this->bni = $bni;
        $this->nama = $nama;
    }

    public function beli($nama = "Barang", $harga = 0) {
        $this->bni->kredit($harga);
        echo "Berhasil melakukan pembelian $nama seharga Rp$harga.\n";
        echo "Terima kasih $this->nama :)\n";
    }
}

```

Class Pembeli ini mempunyai :

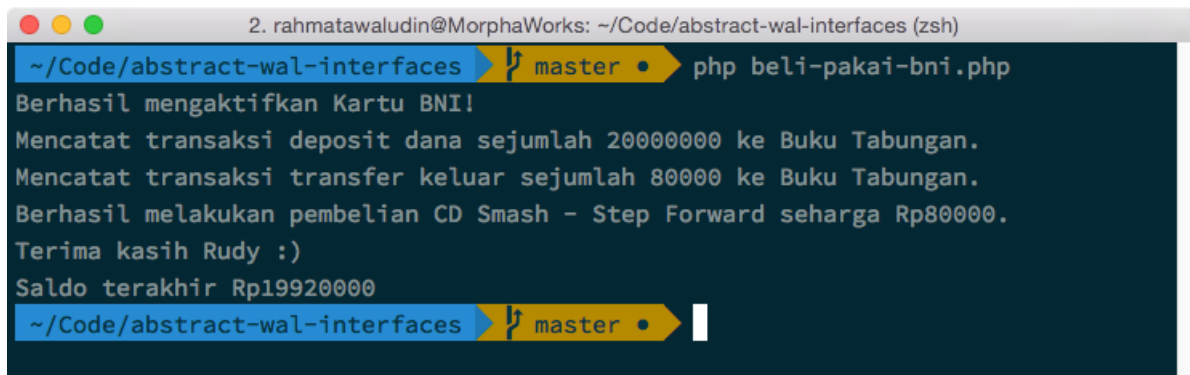
- atribut `$nama` untuk menyimpan nama pembeli
- atribut `$bni` untuk menyimpan object kartu BNI
- method `beli()` untuk melakukan pembelian barang
- method `__construct()` untuk membangun object Pembeli

Yang paling penting untuk diperhatikan disini adalah method `__construct()`. Di method ini kita menginject class BNI sebagai parameternya. Kalau didemokan syntaxnya seperti ini:

~/Code/abstract-wal-interfaces/beri-pakai-bni.php

```
<?php
require_once "Pembeli-DI.php";
// Melakukan pembelian dengan BNI
try {
    $bniKu = new BNI('12345');
    $bniKu->deposit(20000000);
    $rudy = new Pembeli("Rudy", $bniKu);
    $rudy->beli("CD Smash - Step Forward", 80000);
    echo "Saldo terakhir Rp".$bniKu->cekSaldo()."\n";
} catch (Exception $e) {
    echo $e->getMessage()."\n";
}
```

Terlihat disini, sebelum membuat object Pembeli, kita membuat object BNI dulu. Kemudian meng-*inject* object BNI itu ketika membuat object Pembeli. Jika dijalankan, hasilnya seperti berikut :



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master php beli-pakai-bni.php
Berhasil mengaktifkan Kartu BNI!
Mencatat transaksi deposit dana sejumlah 20000000 ke Buku Tabungan.
Mencatat transaksi transfer keluar sejumlah 80000 ke Buku Tabungan.
Berhasil melakukan pembelian CD Smash - Step Forward seharga Rp80000.
Terima kasih Rudy :)
Saldo terakhir Rp19920000
~/Code/abstract-wal-interfaces master
```

Berhasil meng-*inject* BNI ke Pembeli

Masalah dari dependency injection ini adalah bagaimana bila kita akan menggunakan metode pembayaran lain? Misalnya, Paypal. Tentunya, cara mengakses paypal ini pun akan berbeda dengan BNI, karena Paypal harus login dengan email dan password. Begitupun cara paypal melakukan kredit dan deposit, karena paypal perlu mengirim email setiap kali terjadi transaksi.

Kalau gitu langsung di extends dari class BNI saja gimana mas?

Memang, sekiranya implementasinya akan sama, kita cukup meng-extends class Paypal dari BNI. Namun, karena implementasi dari method kredit() dan deposit() ini bisa berbeda, maka fitur pembayaran ini cocok untuk di-abstraksi. Dengan abstraksi pula, akan lebih memudahkan jika akan ada implementasi jenis pembayaran yang baru, misalnya BitCoin.

Kita bisa membuat abstraksi dengan membuat class abstract yang berisi method apa saja yang harus ada di Class tersebut yang akan digunakan di class Pembeli. Tahapannya dimulai dari class Pembeli, ubah menjadi :

~/Code/abstract-wal-interfaces/Pembeli.php

```
<?php
class Pembeli {
    private $nama;
    private $payment;

    public function __construct($nama = "Seseorang", PaymentMethod $payment) {
        $this->nama = $nama;
        $this->payment = $payment;
    }

    public function beli($nama = "Barang", $harga = 0) {
        if ($this->payment->cekSaldo() < $harga) {
            echo "Uang tidak cukup\n";
        } else {
            $this->payment->kredit($harga);
            echo "Terima kasih $this->nama :)\n";
            echo "Berhasil melakukan pembelian $nama seharga Rp".number_format($\
harga).".\n";
        }
    }
}
```

Perubahan terbesar dari class Pembeli adalah kita mengubah atribut \$bni menjadi \$payment dan mengubah mengabstraksi class BNI menjadi PaymentMethod.

Terlihat disini, class PaymentMethod perlu menambahkan beberapa method:

- cekSaldo() untuk mengecek saldo terakhir
- kredit() untuk mengambil sejumlah uang
- deposit() untuk mengisi sejumlah uang

Untuk memudahkan pengecekan, kita akan menambah method cekNamaPembayaran() yang berfungsi menampilkan nama Class yang digunakan untuk melakukan pembayaran. Mari kita buat abstract class PaymentMethod :

~/Code/abstract-wal-interfaces/PaymentMethod.php

```
<?php
abstract class PaymentMethod {
    public function cekNamaPembayaran() {
        return "Anda melakukan pembayaran dengan ".get_class($this)."\n";
    }
    abstract public function kredit($jumlah);
    abstract public function deposit($jumlah);
    abstract public function cekSaldo();
}
```

Selanjutnya, ubah class BNI agar mengekstens PaymentMethod. Untuk memudahkan contoh, kita ubah nama classnya menjadi DebitBNI:

~/Code/abstract-wal-interfaces/DebitBNI.php

```
<?php
require_once "PaymentMethod.php";
class DebitBNI extends PaymentMethod {
    private $saldo;

    public function __construct($pin) {
        // ceritanya cek PIN ke database
        if ($pin == '12345') {
            echo "Berhasil mengaktifkan Kartu Debit!\n";
        } else {
            $pesan = "PIN yang Anda masukkan salah :(";
            throw new Exception($pesan);
        }
    }

    private function catatTransaksi($jenis, $jumlah) {
        echo "Mencatat transaksi $jenis sejumlah $jumlah ke Buku Tabungan.\n";
    }

    public function kredit($jumlah) {
        $this->catatTransaksi('transfer keluar', $jumlah);
        $this->saldo -= $jumlah;
    }

    public function deposit($jumlah) {
        $this->catatTransaksi('deposit dana', $jumlah);
    }
}
```

```

        $this->saldo += $jumlah;
    }

    public function cekSaldo() {
        return $this->saldo;
    }
}

```

Mari kita buat demo untuk metode pembayaran ini:

~/Code/abstract-wal-interfaces/beli-pakai-debitbni.php

```

<?php
require_once "DebitBNI.php";
require_once "Pembeli.php";

// Melakukan pembelian dengan DebitBNI
try {
    $paymentMethod = new DebitBNI("12345");
    $paymentMethod->deposit(20000000);
    $rahmat = new Pembeli("Morgan", $paymentMethod);
    $rahmat->beli("Sepatu Dance", 250000);
    echo "Saldo terakhir Rp".number_format($paymentMethod->cekSaldo())."\n";
    echo $paymentMethod->cekNamaPembayaran();
} catch (Exception $e) {
    echo $e->getMessage()."\n";
}

```

Jika dijalankan, hasilnya akan seperti ini :

```

1. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php beli-pakai-debitbni.php
Berhasil mengaktifkan Kartu Debit!
Mencatat transaksi deposit dana sejumlah 20000000 ke Buku Tabungan.
Mencatat transaksi transfer keluar sejumlah 250000 ke Buku Tabungan.
Terima kasih Morgan :)
Berhasil melakukan pembelian Sepatu Dance seharga Rp250,000.
Saldo terakhir Rp19,750,000
Anda melakukan pembayaran dengan DebitBNI
~/Code/abstract-wal-interfaces master •

```

Membuat DebitBNI dengan abstract PaymentMethod

Di baris terakhir output terlihat kita menggunakan implementasi PaymentMethod dengan class DebitBNI.

Untuk implementasi Paypal, kita buat seperti ini:

~/Code/abstract-wal-interfaces/Paypal.php

```
<?php
require_once 'PaymentMethod.php';
class Paypal extends PaymentMethod {
    private $balance;

    public function __construct($email, $password) {
        // Ceritanya ini akses ke database
        if ($email == "morgan@gmail.com" & $password == "12345") {
            $this->email = $email;
            echo "Berhasil login ke Paypal!\n";
        } else {
            $pesan = "User ada user dengan username/password tersebut :(";
            throw new Exception($pesan);
        }
    }

    private function kirimNotifikasi($pesan = "Informasi penting") {
        echo "Mengirim email notifikasi $pesan ke $this->email \n";
    }

    public function kredit($jumlah) {
        $this->kirimNotifikasi('pengeluaran dana');
        $this->balance -= $jumlah;
    }

    public function deposit($jumlah) {
        $this->kirimNotifikasi('penerimaan dana');
        $this->balance += $jumlah;
    }

    public function cekSaldo() {
        return $this->balance;
    }
}
```

Terlihat disini, class Paypal ini mengimplementasikan semua method dari class abstract PaymentMethod, ini diharuskan. Karena, sebagaimana saya jelaskan di pembahasan sebelumnya, class yang

meng-*ekstends* abstract class harus mengimplementasikan semua abstract methodnya. Jika tidak, aplikasi akan error.

Perbedaan lain di class Paypal adalah :

- Untuk membuat object harus menggunakan email dan password yang kita hardcode di method `__construct()`. Tentunya, di kenyataannya kita akan mengecek ini ke database.
- atribut `$balance` digunakan untuk menyimpan dana.
- Setiap kali ada transaksi uang masuk atau keluar, memanggil method `kirimNotifikasi()` yang disimulasikan akan mengirim email.

Demo dari metode pembayaran ini :

~/Code/abstract-wal-interfaces/beli-pakai-paypal.php

```
<?php
require_once "Paypal.php";
require_once "Pembeli.php";

// Melakukan pembelian dengan paypal
try {
    $paymentMethod = new Paypal("morgan@gmail.com", "12345");
    $paymentMethod->deposit(12000000);
    $pembeli = new Pembeli("Morgan", $paymentMethod);
    $pembeli->beli("Poster Smash Full Color", 100000);
    echo "Saldo terakhir Rp".number_format($paymentMethod->cekSaldo())."\n";
    echo $paymentMethod->cekNamaPembayaran();
} catch (Exception $e) {
    echo $e->getMessage()."\n";
}
```

Outputnya akan menjadi :

```

2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php beli-pakai-paypal.php
Berhasil login ke Paypal!
Mengirim email notifikasi penerimaan dana ke morgan@gmail.com
Mengirim email notifikasi pengeluaran dana ke morgan@gmail.com
Terima kasih Morgan :)
Berhasil melakukan pembelian Poster Smash Full Color seharga Rp100,000.
Saldo terakhir Rp11,900,000
Anda melakukan pembayaran dengan Paypal
~/Code/abstract-wal-interfaces master •

```

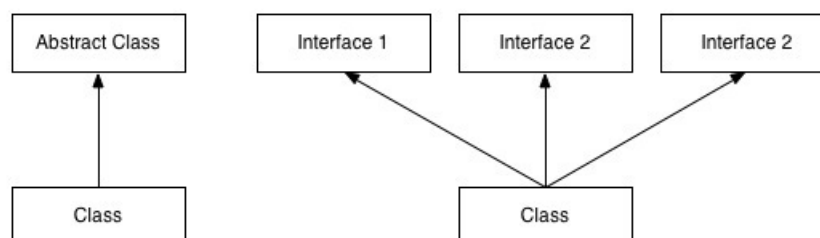
Membuat Paypal dengan abstract PaymentMethod

Jika diperhatikan, method pada class \$paymentMethod yang kita gunakan disini, sama dengan method yang kita pakai di demo dengan pembayaran DebitBNI. Inilah kekuatan dari abstract class. Kita bisa melakukan standarisasi nama method, apapun bentuk implementasinya.

Nah.. udah ah. Segini aja dulu. Silahkan istirahat, ambil snack dan minumannya ya..

Interfaces

Interface hampir mirip dengan abstract class. Dimana kita harus membuat method yang ada di Interface pada class yang mengimplementasikan interfaces tersebut. Perbedaannya adalah sebuah class hanya bisa meng-*extends* satu abstract class, tapi bisa mengimplementasikan banyak interfaces. Kira-kira seperti ini ilustrasinya:



Ilustrasi Abstract dan Interfaces

Misalnya, ada class Kucing, dia meng-*extends* abstract class HewanMamalia. Nah, dia juga bisa mengimplementasikan interface Sayap, jadi dia bisa punya method terbang(). Atau ditambah lagi dengan implementasi interface Insang, jadi dia bisa punya method menyelam() (kebayang ngga kucing yang bisa terbang dan bisa menyelam? :v). Tapi, kucing ini tidak bisa meng-*extends* abstract class HewanMelata. Karena, dia hanya boleh meng-*extends* satu abstract class (dalam contoh ini HewanMamalia). Sip.

Apa manfaat Interface?

Banyak. Salah satunya, Interface biasanya digunakan untuk kita melakukan komunikasi antar object yang tidak saling berhubungan. Contoh, misalnya kita ingin membuat sebuah Class untuk membandingkan statistik berbagai akun di media sosial, kita sebut saja SocialGraph. Class ini punya fungsi compareLike yang bisa membandingkan membandingkan jumlah Like di berbagai akun media sosial. Versi pertama dari SocialGraph, hanya bisa membandingkan jumlah like antar akun Facebook.

~/Code/abstract-wal-interfaces/SocialGraph.php

```
<?php
class SocialGraph {
    public static function compareLike(Facebook $fb1, Facebook $fb2) {
        if ($fb1->totalLike() > $fb2->totalLike()) {
            echo "Status " . $fb1->user() . " Lebih populer dari " . $fb2->user() \
. "\n";
        } elseif ($fb1->totalLike() < $fb2->totalLike()) {
            echo "Status " . $fb2->user() . " Lebih populer dari " . $fb1->user() \
. "\n";
        } else {
            echo "Status kedua user sama-sama populer";
        }
    }
}
```

Yang perlu diperhatikan disini adalah kita menggunakan *type hinting* object class Facebook pada method compareLike. Ini digunakan agar script menampilkan error jika user tidak memberikan parameter berupa object Facebook.

Kita juga menggunakan method totalLike() untuk membandingkan popularitas antar status Facebook. Dengan class Facebook :

~/Code/abstract-wal-interfaces/Facebook.php

```
<?php
class Facebook {
    private $status;
    private $user;
    private $like = 0;
    public function __construct($user, $status) {
        $this->user = $user;
        $this->status = $status;
    }
    public function status() {
        return $this->status;
    }
}
```

```

    }
    public function user() {
        return $this->user;
    }
    public function like() {
        $this->like++;
    }
    public function totalLike() {
        return $this->like;
    }
}

```

Demokan dengan syntax ini :

~/Code/abstract-wal-interfaces/demo-socialgraph.php

```

<?php
include "Facebook.php";
include "SocialGraph.php";

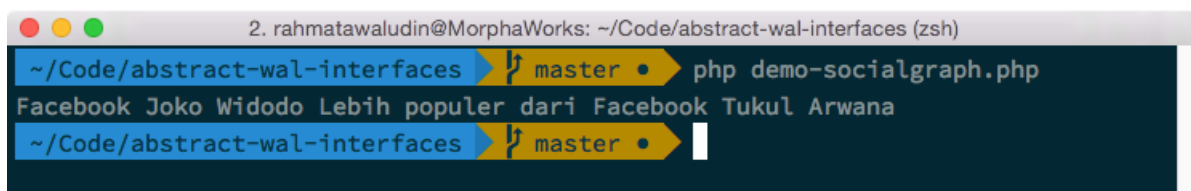
$fbTukul = new Facebook("Tukul Arwana", "Kembali ke laptop!");
$fbTukul->like();
$fbTukul->like();

$fbJokowi = new Facebook("Joko Widodo", "Aku rapopo..");
$fbJokowi->like();
$fbJokowi->like();
$fbJokowi->like();

$socialGraph = new SocialGraph();
$socialGraph->compareLike($fbTukul, $fbJokowi);

```

Terlihat disini, status Tukul mendapat 2 like, sedangkan status Jokowi mendapat 3 like. Jika dijalankan:



```

2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php demo-socialgraph.php
Facebook Joko Widodo Lebih populer dari Facebook Tukul Arwana
~/Code/abstract-wal-interfaces master • 

```

SocialGraph membandingkan popularitas status facebook

Oke, SocialGraph versi 1 sudah berjalan. Aplikasi kita terus berkembang, kini kita akan mensupport akun Twitter. Misalnya dengan class Twitter seperti berikut:

~/Code/abstract-wal-interfaces/Twitter.php

```
<?php
class Twitter {
    private $tweet;
    private $user;
    private $favorite;
    public function __construct($user, $tweet) {
        $this->user = $user;
        $this->tweet = $tweet;
    }
    public function tweet() {
        return $this->tweet;
    }
    public function user() {
        return $this->user;
    }
    public function favorite() {
        $this->favorite++;
    }
    public function totalFavorite() {
        return $this->favorite;
    }
}
```

Disinilah masalahnya muncul, cara menentukan jumlah like di twitter ternyata berbeda. Karena, twitter tidak mengenal istilah *like*, yang dikenal di twitter adalah *favorite*. Jika kita coba bandingkan popularitas akun facebook dengan twitter, pasti tidak akan bisa. Misalnya dengan syntax :

~/Code/abstract-wal-interfaces/demo-socialgraph.php

```
<?php
include "Facebook.php";
include "Twitter.php";
include "SocialGraph.php";

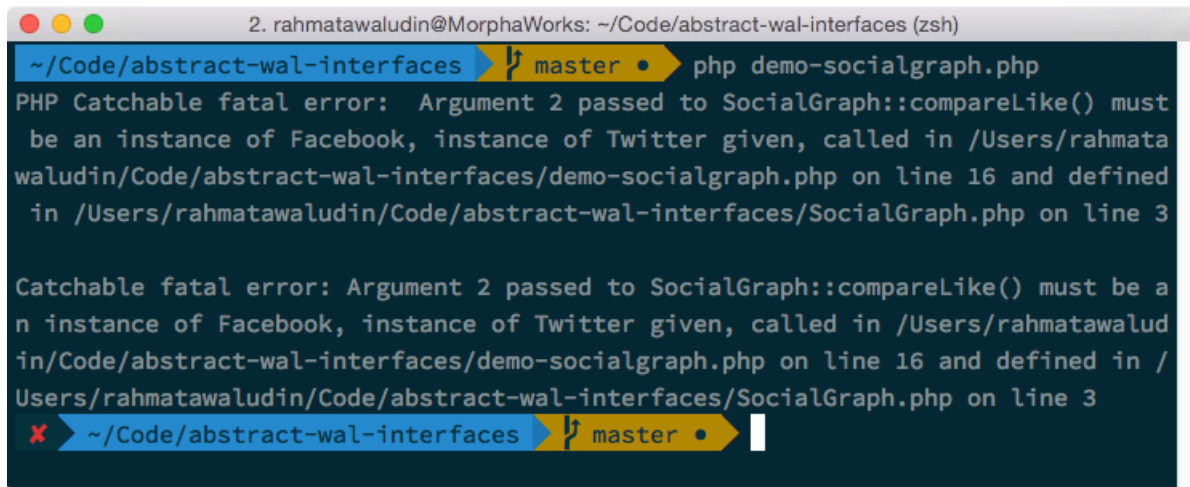
$fbTukul = new Facebook("Tukul Arwana", "Kembali ke laptop!");
$fbTukul->like();
$fbTukul->like();

$twJokowi = new Twitter("Joko Widodo", "Aku rapopo..");
$twJokowi->favorite();
$twJokowi->favorite();
```

```
$twJokowi->favorite();

$socialGraph = new SocialGraph();
$socialGraph->compareLike($fbTukul, $twJokowi);
```

Akan muncul error



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php demo-socialgraph.php
PHP Catchable fatal error: Argument 2 passed to SocialGraph::compareLike() must
be an instance of Facebook, instance of Twitter given, called in /Users/rahmata
waludin/Code/abstract-wal-interfaces/demo-socialgraph.php on line 16 and defined
in /Users/rahmatawaludin/Code/abstract-wal-interfaces/SocialGraph.php on line 3

Catchable fatal error: Argument 2 passed to SocialGraph::compareLike() must be a
n instance of Facebook, instance of Twitter given, called in /Users/rahmatawalud
in/Code/abstract-wal-interfaces/demo-socialgraph.php on line 16 and defined in /
Users/rahmatawaludin/Code/abstract-wal-interfaces/SocialGraph.php on line 3
X ~/Code/abstract-wal-interfaces master •
```

Tidak bisa membandingkan jumlah like facebook dengan favorite twitter

Ini terjadi karena method `compareLike()` hanya menerima input berupa object Facebook. Ada beberapa cara untuk menyelesaikannya:

1. Gunakan method yang berbeda untuk membandingkan facebook dan twitter misalnya `compareFacebookTwitter(Facebook $fb, Twitter tw)`. Tapi akan muncul masalah baru, bagaimana jika membandingkan Twitter dengan Twitter apa perlu dibuat method baru `compareTwitterTwitter()`? Dan bagaimana jika membandingkan Twitter ke Facebook apa dibuat `compareTwitterFacebook()`? Solusi ini tidak akan digunakan.
2. Gunakan parent class yang sama, misalnya `Social` untuk class `Facebook` dan `Twitter`. Dan ubah paramter method `compareTo` menjadi `Social`. Solusi ini mesti akan berhasil sekarang, akan merepotkan jika kita akan mengimplementasikan fitur `compareLike` ini pada object lain, misalnya `Artikel` di web, `Video` di Youtube, dll. Solusi ini tidak akan digunakan.
3. Buat interface `Likeable` untuk menunjukkan object yang bisa di Like. Solusi ini yang akan kita gunakan.

Mari kita buat interfacenya :

~/Code/abstract-wal-interfaces/Likeable.php

```
<?php
interface Likeable {
    public function platform();
    public function user();
    public function totalLike();
    public function like();
}
```

Disini, kita mendefinisikan beberapa method yang harus di buat pada Class yang mengimplemen-
tasikan Likeable yaitu platform(), user(), totalLike() dan like().

Mari kita implementasikan Likeable di class Facebook :

~/Code/abstract-wal-interfaces/Facebook.php

```
<?php
include_once "Likeable.php";
class Facebook implements Likeable {
    ....
    public function platform() {
        return "Facebook";
    }
}
```

Dan di class Twitter :

~/Code/abstract-wal-interfaces/Twitter.php

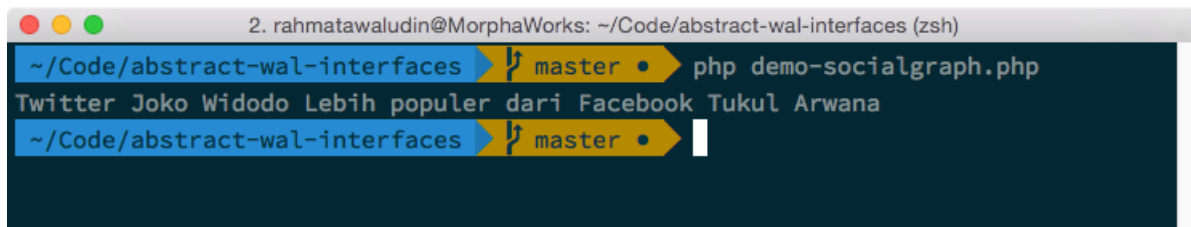
```
<?php
include_once "Likeable.php";
class Twitter implements Likeable {
    ....
    public function like() {
        $this->favorite();
    }
    public function totalLike() {
        return $this->totalFavorite();
    }
    public function platform() {
        return "Twitter";
    }
}
```

Kita ubah juga method `compareLike` di class `SocialGraph` agar parameter nya menjadi interface `Likeable` :

~/Code/abstract-wal-interfaces/SocialGraph.php

```
<?php
class SocialGraph {
    public static function compareLike(Likeable $social1, Likeable $social2) {
        if ($social1->totalLike() > $social2->totalLike()) {
            echo $social1->platform() . " ". $social1->user();
            echo " Lebih populer dari " . $social2->platform() . " ";
            echo $social2->user() . "\n";
        } elseif ($social1->totalLike() < $social2->totalLike()) {
            echo $social2->platform() . " ". $social2->user();
            echo " Lebih populer dari " . $social1->platform() . " ";
            echo $social1->user() . "\n";
        } else {
            echo "Kedua user sama-sama populer.\n";
        }
    }
}
```

Sehingga, kalau kita jalankan lagi demonya :



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces master • php demo-socialgraph.php
Twitter Joko Widodo Lebih populer dari Facebook Tukul Arwana
~/Code/abstract-wal-interfaces master •
```

Berhasil membandingkan favorite di Twitter dan like di Facebook

Berhasil kan?

Dengan cara ini kita berhasil membandingkan (mengkomunikasikan) jumlah like di facebook dan jumlah favorite di Twitter. Jika kedepannya kita akan mendukung media yang lain, misalnya video di Youtube, maka class `VideoYoutube` tersebut tinggal mengimplementasikan interface `Likeable`.

Jika saya ambil kesimpulan, ketika kita menggunakan abstract class kita fokus pada **siapa Class ini** (*instance of*). Sedangkan, ketika kita menggunakan interface kita fokus pada **apa yang bisa dilakukan Class ini** (*capable of*).

Dengan menggunakan interface, kita dapat menyeragamkan API (method `like()`) untuk aplikasi kita, meskipun jenis implementasinya beragam (facebook, twitter, youtube, dll).

Interfaces sangat membantu dalam membangun aplikasi skala besar. Penjelasan saya ini merupakan contoh sederhana penggunaan interface di lapangan, masih banyak contoh lainnya.

Laravel menggunakan banyak interfaces dalam bentuk contract untuk berbagai fiturnya. Sehingga memungkinkan kita untuk merubah implementasinya sesuai kehendak kita. Contoh penggunaan sederhananya seperti ini:

- Mau merubah implementasi view dari Blade ke Twig? Bisa.
- Mau merubah penyimpanan Cache ke MongoDB? Bisa.
- Mau merubah implementasi Queue ke driver lain? Bisa.
- dll.

Dan semua perubahan itu bisa dilakukan hanya dengan merubah implementasi *contract (interfaces)* yang berhubungan. **Serius, Keren.**

Tentunya tidak akan cukup kalau saya jelaskan contract di Laravel pada bab ini. Oleh karena itu, pembahasan lebih lanjut tentang contract akan kita bahas pada bab **Arsitektur Laravel**.



Source code dari latihan ini bisa didapat di <https://github.com/rahmatawaludin/abstract-wal-interfaces>³

³<https://github.com/rahmatawaludin/abstract-wal-interfaces>

Mengakses Database

Chunk, memproses banyak data dengan lebih efisien memory

Menggunakan chunk sangat bermanfaat ketika kita akan memproses banyak baris dalam database. Penggunaan akan meminimalkan penggunaan memory dalam eksekusi script yang kita buat. Untuk menunjukkan manfaatnya, mari kita buat 10.000 data pada table products dengan membuat seeder SampleChunkSeeder dengan isi:

database/seeds/SampleChunkSeeder.php

```
use Faker\Factory as Faker;
use Illuminate\Database\Seeder;

class SampleChunkSeeder extends Seeder {

    public function run()
    {
        $faker = Faker::create();
        $products = ["Accord", "Civic", "City", "CR-V", "Jazz", "Freed", "Mobili\
o"];
        $descriptions = ["Tipe manual", "Tipe Otomatis"];

        for ( $i=0; $i < 10000; $i++ ) {
            DB::insert('insert into products (name, description, price, stock) v\
alues (:name, :description, :price, :stock)', [
                'name' => $products[rand(0,6)] . ' ' . $faker->firstNameMale,
                'description' => $descriptions[rand(0,1)],
                'price' => rand(100,800) * 1000000,
                'stock' => rand(2,40)
            ]);
        }

        $this->command->info('Berhasil 10.000 menambah mobil!');
    }
}
```

Setelah dibuat, mari kita jalankan seeder ini hingga muncul tulisan **Berhasil menambah 10.000 mobil!**.

```
vagrant@homestead:~/Code/sample-database$ php artisan db:seed --class "SampleChunkSeeder"
Berhasil menambah 10.000 mobil!
```

Untuk mengetes penggunaan chunk, kita akan membuat route untuk menampilkan product yang memiliki stock lebih dari 20. Kita akan menggunakan 2 route, pertama tanpa chunk dan kedua dengan chunk, kemudian kita bandingkan penggunaan memory keduanya dengan fungsi `memory_get_usage()` di PHP. Mari kita buat route pertama:

app/Http/routes.php

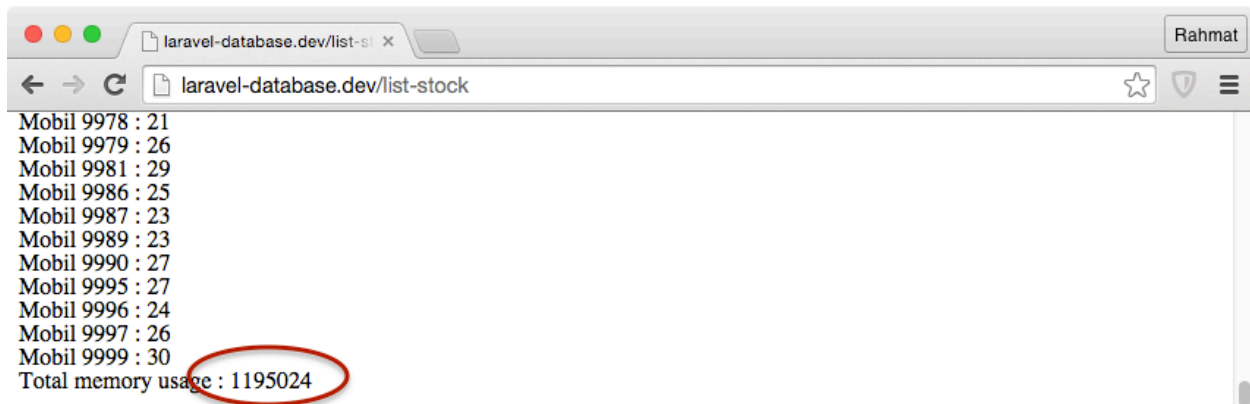
```
....
Route::get('/list-stock', function() {
    $begin = memory_get_usage();
    foreach (DB::table('products')->get() as $product) {
        if ( $product->stock > 20 ) {
            echo $product->name . ' : ' . $product->stock . '<br>';
        }
    }
    echo 'Total memory usage : ' . (memory_get_usage() - $begin);
});
```

Kita buat juga route kedua dengan chunk:

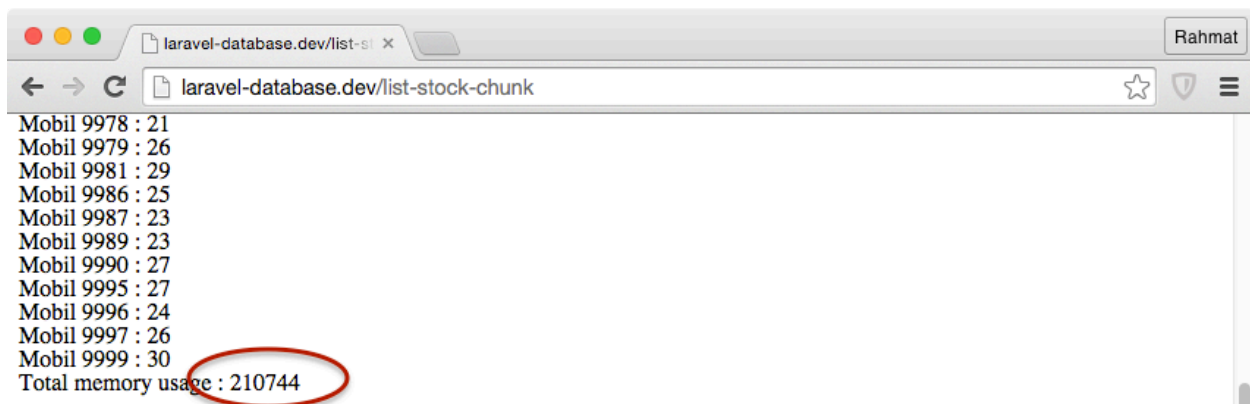
app/Http/routes.php

```
....
Route::get('/list-stock-chunk', function() {
    $begin = memory_get_usage();
    DB::table('products')->chunk(100, function($products)
    {
        foreach ($products as $product)
        {
            if ( $product->stock > 20 ) {
                echo $product->name . ' : ' . $product->stock . '<br>';
            }
        }
    });
    echo 'Total memory usage : ' . (memory_get_usage() - $begin);
});
```

Kini, kita dapat membandingkan penggunaan memory keduanya dengan mengunjungi `/list-stock` dan `/list-stock-chunk`.



Tanpa Chunk



Dengan Chunk

Terlihat disini, tanpa menggunakan chunk memory yang digunakan sebanyak 1195024 byte atau hampir 1 Mb. Sementara, menggunakan chunk memory yang digunakan sebanyak 210744 byte atau sekitar 200 Kb. **Wow!**

Penggunaan chunk ini sangat disarankan ketika kita berinteraksi dengan banyak data. Tentunya, jika kita menggunakan lebih banyak data, akan sangat terlihat penghematan memory yang kita lakukan.

Untuk memudahkan latihan selanjutnya, silahkan jalankan kembali `php artisan migrate:refresh --seed` untuk me-reset kondisi database.

Routing, Kendalikan Alur Aplikasi

Subdomain Routing

Jika kita menggunakan aplikasi misalnya Slack, biasanya setelah signup kita memiliki domain sendiri misalnya `malescast.slack.com`, `bukularavel.slack.com`, dsb. Di Laravel, kita dapat membuat fitur dengan menggunakan Subdomain Routing. Syntax dasarnya seperti berikut:

Subdomain routing

```
Route::group(['domain' => '{account}.myapp.com'], function()
{

    Route::get('user/{id}', function($account, $id)
    {
        //
    });

});
```

Misalnya, kita hendak membuat website `fakebook.dev` dan bisa menerima subdomain dengan username.

- Jika kita mengakses `joni.fakebook.dev` maka akan menampilkan halaman akun Joni
- Jika kita mengakses `joni.fakebook.dev/profile` maka akan menampilkan profile Joni
- Jika kita mengakses `joni.fakebook.dev/status/1` maka akan menampilkan status Joni dengan id 1

Mari kita buat untuk case pertama, menampilkan halaman akun. Untuk membuatnya silahkan persiapkan aplikasi laravel di homestead dengan domain ke `fakebook.dev`. Kemudian, tambahkan route berikut:

t

```

....
Route::group(['domain' => '{username}.fakebook.dev'], function()
{
    Route::get('/', function($username) {
        return 'Anda mengunjungi akun ' . $username;
    });
});
});

```

Disini, kita menggunakan **username** sebagai nama dari subdomain. Pada route `'/'` (root) kita menampilkan teks informasi akun siapa yang sedang dikunjungi.

Karena di host lokal tidak diizinkan membuat wildcard subdomain, kita perlu menambah domain untuk tiap user secara manual. Mari kita buat untuk user joni dan kiki. Tambahkan baris berikut di file host (sesuaikan IP dengan IP homestead):

`/etc/hosts` atau `C:\Windows\System32\Drivers\etc\hosts`

```

1 192.168.10.10    kiki.fakebook.dev
2 192.168.10.10    joni.fakebook.dev
3 ....

```



Wildcard Subdomin

Untuk Linux dan OSX, selain menggunakan cara manual, kita juga dapat menggunakan [dnsmasq](http://www.thekelleys.org.uk/dnsmasq/doc.html)⁴. Silahkan baca tutorial untuk [OSX](https://vinkla.com/posts/setup-wildcard-dns-on-mac-os-x/)⁵ atau [Ubuntu](https://help.ubuntu.com/community/Dnsmasq)⁶.

Di server produksi, untuk membuat wildcard subdomain ini dengan cara membuat **A Record** baru. Isi **name** dengan `*.domainkita.com`, **address** dengan IP server dan **TTL** 1 hour. Tunggu 1 jam, wildcard subdomain kitapun akan berjalan.

Setelah siap, mari kita cek:

⁴<http://www.thekelleys.org.uk/dnsmasq/doc.html>

⁵<https://vinkla.com/posts/setup-wildcard-dns-on-mac-os-x/>

⁶<https://help.ubuntu.com/community/Dnsmasq>



Mengunjungi Akun Joni



Mengunjungi Akun Kiki

Sip. Mari kita lanjut ke case kedua, menampilkan profile. Untuk memudahkan, mari kita simpan semua data user dalam array. Kemudian, kita passing array ini ke closure di route untuk menampilkan profile. Sehingga file route berubah menjadi:

app/Http/routes.php

```
1 ....
2 Route::group(['domain' => '{username}.fakebook.dev'], function()
3 {
4     ....
5     $data_user = [
6         'joni' => [
7             'profile' => 'Seorang programmer imut.',
8             'status' => ['Gue keren!', 'Gue keren bgt!', 'Top dah!']
9         ],
10        'kiki' => [
11            'profile' => 'Seorang programmer cute.',
12            'status' => ['Mantap!', 'Hari ini luar biasa!', 'Cemungut ea..']
13        ]
14    ]
15 }
```

```
14     ];
15
16     Route::get('profile', function($username) use ($data_user)
17     {
18         return $data_user[$username]['profile'];
19     });
20 });
```

Mari kita coba kunjungi profile Joni:



Mengunjungi Profile Joni

Untuk case ketiga, menampilkan status, kita juga akan menggunakan array yang telah dibuat tadi. Tambahkan route berikut:

app/Http/routes.php

```
1     ....
2     Route::group(['domain' => '{username}.fakebook.dev'], function()
3     {
4         ....
5         Route::get('status/{id}', function($username, $id) use ($data_user)
6         {
7             return $username . ' menulis : ' . $data_user[$username]['status'][$id];
8         });
9     });
```

Mari kita coba mengecek status Kiki:



Mengecek Status Kiki



Mengecek Status Kiki

Sip.