Modul Praktikum 13

Penerapan Testing pada DevOps dalam Pengembangan Web menggunakan CodeIgnter Framework

1. Maksud

Modul praktikum ini dimaksudkan untuk memperkenalkan dan memberikan pemahaman praktis mengenai penerapan metodologi **Test-Driven Development (TDD)**, yang mencakup **Unit Testing** dan **Integration Testing**, pada framework PHP CodeIgniter dalam lingkungan pengembangan berbasis Docker dengan server web Apache.

---

2. Tujuan

Setelah menyelesaikan praktikum ini, praktikan diharapkan mampu:

a) Menyiapkan lingkungan pengembangan CodeIgniter 4 menggunakan Docker dan Docker Compose dengan server web Apache.

b) Memahami konsep dasar Test-Driven Development (TDD), Unit Testing, dan Integration Testing.

c) Menulis dan menjalankan Unit Test untuk komponen Model pada CodeIgniter.

d) Menulis dan menjalankan Integration Test untuk alur MVC (Model-View-Controller).

e) Menerapkan siklus TDD (**Red**-**Green**-**Refactor**) dalam pengembangan sebuah fitur.

---

3. Dasar Teori

a) **Docker**: Sebuah platform untuk mengembangkan, mengirim, dan menjalankan aplikasi dalam wadah (container) yang terisolasi. Docker memungkinkan kita untuk membungkus aplikasi beserta semua dependensinya, memastikan lingkungan pengembangan konsisten di antara semua pengembang.

b) **Apache2**: Salah satu server web paling populer di dunia. Dalam konfigurasi ini, Apache akan bertindak sebagai *entry point* yang menerima permintaan dari pengguna dan meneruskannya ke layanan PHP-FPM untuk diproses.

c) **CodeIgniter 4**: Sebuah framework PHP yang ringan dan cepat. CodeIgniter 4 hadir dengan *testing suite* yang sudah terintegrasi dengan **PHPUnit**, mempermudah proses penulisan tes otomatis.

d) **Test-Driven Development (TDD)**: Sebuah metodologi pengembangan di mana pengembang menulis tes otomatis *sebelum* menulis kode fungsionalnya. Alur kerjanya adalah **Red** -> **Green** -> **Refactor**.

e) **Unit Testing**: Proses pengujian unit atau komponen terkecil dari perangkat lunak secara terisolasi (misalnya, satu metode dalam sebuah class).

f) **Integration Testing**: Proses pengujian interaksi antara beberapa komponen yang terintegrasi. Dalam konteks MVC, ini berarti menguji apakah Controller, Model, dan View dapat bekerja sama dengan benar.

---

4. Prosedur Praktik

Praktikan akan menyiapkan lingkungan pengembangan dan menginstal CodeIgniter 4 di dalamnya.

Langkah 1: Persiapan Lingkungan Docker

1. Buat struktur direktori proyek seperti berikut:

```
tdd-codeigniter/
├── .docker/
│   ├── apache/
│   │   └── httpd.conf
│   └── php/
│       └── Dockerfile
├── src/
└── docker-compose.yml
```

2. Isi file **docker-compose.yml**. File ini mendefinisikan tiga layanan: app (PHP), webserver (Apache), dan db (MySQL).

```
# Versi Docker Compose
version: '3.8'

# Definisi layanan (kontainer)
services:
  # Layanan Aplikasi PHP (PHP-FPM)
  app:
      build:
        context: .
        dockerfile: .docker/php/Dockerfile
      container_name: tdd_app_ci4_apache
      volumes:
        # Sinkronisasi folder src lokal dengan folder di dalam kontainer
        - ./src:/var/www/html
      networks:
        - app-network

  # Layanan Web Server (Apache)
  webserver:
      image: httpd:2.4
      container_name: tdd_webserver_ci4_apache
      ports:
```

```
        # Hubungkan port 8080 di komputer lokal ke port 80 di kontainer
        - "8080:80"
      volumes:
        # Sinkronisasi folder src untuk diakses oleh Apache
        - ./src:/var/www/html
        # Ganti file konfigurasi default Apache dengan file lokal kita
        -
./.docker/apache/httpd.conf:/usr/local/apache2/conf/httpd.conf
      depends_on:
        # Pastikan layanan 'app' berjalan sebelum 'webserver'
        - app
      networks:
        - app-network

  # Layanan Database (MySQL)
  db:
      image: mysql:8.0
      container_name: tdd_db_ci4_apache
      restart: unless-stopped
      tty: true
      ports:
        - "3306:3306"
      volumes:
        # Volume untuk menyimpan data database secara persisten
        - db_data:/var/lib/mysql
      environment:
        MYSQL_DATABASE: ci4_test_db
        MYSQL_USER: user
        MYSQL_PASSWORD: password
        MYSQL_ROOT_PASSWORD: root_password
      networks:
        - app-network

# Definisi jaringan kustom
networks:
  app-network:
      driver: bridge

# Definisi volume persisten
volumes:
  db_data:
```

3. Isi file Dockerfile di **.docker/php/Dockerfile**. File ini digunakan untuk membangun image PHP kustom yang berisi ekstensi yang diperlukan.

```
# Gunakan image dasar PHP 8.2 FPM
FROM php:8.2-fpm

# Install dependensi sistem yang dibutuhkan
# Termasuk 'composer' jika tidak ada di base image
RUN apt-get update && apt-get install -y \
      build-essential \
      libpng-dev \
      libjpeg62-turbo-dev \
```

```
      libfreetype6-dev \
      locales \
      zip \
      vim \
      unzip \
      git \
      curl \
      composer \
      libonig-dev \
      libxml2-dev \
      libzip-dev

# Install ekstensi PHP yang dibutuhkan oleh CodeIgniter
RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd zip

# Install Xdebug untuk code coverage saat testing
RUN pecl install xdebug && docker-php-ext-enable xdebug

# Konfigurasi Xdebug 3 untuk mode coverage
RUN echo "xdebug.mode=coverage" >> /usr/local/etc/php/conf.d/docker-
php-ext-xdebug.ini

# Atur direktori kerja
WORKDIR /var/www/html
```

4. Isi file *httpd.conf* di ***.docker/apache/httpd.conf***. File ini mengonfigurasi Apache untuk meneruskan permintaan PHP ke kontainer app.

```
# Konfigurasi dasar server
ServerRoot "/usr/local/apache2"
Listen 80
ServerAdmin admin@example.com
ServerName localhost:80

# Muat modul yang diperlukan
LoadModule mpm_event_module modules/mod_mpm_event.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authn_core_module modules/mod_authn_core.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule authz_core_module modules/mod_authz_core.so
LoadModule access_compat_module modules/mod_access_compat.so
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule reqtimeout_module modules/mod_reqtimeout.so
LoadModule filter_module modules/mod_filter.so
LoadModule mime_module modules/mod_mime.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule env_module modules/mod_env.so
LoadModule headers_module modules/mod_headers.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule version_module modules/mod_version.so
LoadModule unixd_module modules/mod_unixd.so
LoadModule status_module modules/mod_status.so
```

```
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
# Modul penting untuk rewrite dan proxy
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so

<IfModule unixd_module>
      User daemon
      Group daemon
</IfModule>

# Atur DocumentRoot ke direktori public CodeIgniter
DocumentRoot "/var/www/html/public"
<Directory "/var/www/html/public">
      Options Indexes FollowSymLinks
      # Izinkan penggunaan file .htaccess untuk URL rewriting
      AllowOverride All
      Require all granted
</Directory>

# Atur file index
DirectoryIndex index.php index.html

# Teruskan semua request file .php ke layanan PHP-FPM (app:9000)
# Ini adalah bagian terpenting untuk menghubungkan Apache dan PHP
ProxyPassMatch                                    ^/(.*\.php(/.*)?)$
fcgi://app:9000/var/www/html/public/$1

# Logging
ErrorLog /proc/self/fd/2
CustomLog /proc/self/fd/1 common
```

5. Setelah semua file konfigurasi diisi, buka **PowerShell/CMD** di **C:\Users\amikom\tdd-codeigniter** dan jalankan perintah:

   docker-compose up -d –build

Langkah 2: Instalasi dan Konfigurasi CodeIgniter

1. Dari terminal **PowerShell/CMD** Anda, jalankan Composer di dalam kontainer app:

   docker-compose exec app composer create-project codeigniter4/appstarter .

2. Pindah ke direktori src dan salin file env menjadi .env menggunakan perintah Windows:

```
cd src
copy env .env
cd ..
```

3. Buka file **src\.env** dengan editor teks Anda dan sesuaikan konfigurasinya seperti pada prosedur sebelumnya.

4. Masuk ke dalam terminal kontainer app dari **PowerShell/CMD**:

   docker-compose exec app bash

Terminal Anda sekarang akan berubah menjadi terminal Linux (**root@<container_id>:/var/www/html#).**

5. Dari **dalam kontainer**, buat dan jalankan migrasi database:

```
# Perintah ini dijalankan di dalam terminal kontainer
php spark make:migration create_products_table
```

Isi file migrasi yang baru dibuat di **src/app/Database/Migrations/** dengan skema tabel produk (kolom: id, name, price).

6. Mengisi File Migrasi

Buka file migrasi yang baru saja dibuat. Anda akan melihat dua metode kosong: *up()* dan *down()*.

a) up(): Metode ini dieksekusi ketika migrasi dijalankan. Fungsinya adalah untuk membuat tabel, menambah kolom, dll.

b) down(): Metode ini dieksekusi ketika migrasi dibatalkan (rollback). Fungsinya adalah untuk menghapus tabel, menghapus kolom, dll.

Ganti isi file tersebut dengan kode berikut untuk mendefinisikan struktur tabel products.

app/Database/Migrations/xxxx_CreateProductsTable.php

```
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class CreateProductsTable extends Migration
{
    public function up()
    {
        // Mendefinisikan kolom-kolom untuk tabel 'products'
        $this->forge->addField([
            'id' => [
                'type'           => 'INT',
                'constraint'     => 5,
                'unsigned'       => true,
                'auto_increment' => true,
            ],
            'name' => [
                'type'       => 'VARCHAR',
```

```
                'constraint' => '100',
            ],
            'price' => [
                'type'       => 'DECIMAL',
                'constraint' => '10,2', // Untuk menyimpan nilai uang
            ],
            // Anda bisa menambahkan kolom created_at dan updated_at jika
perlu
            // 'created_at' => [
            //     'type' => 'DATETIME',
            //     'null' => true,
            // ],
            // 'updated_at' => [
            //     'type' => 'DATETIME',
            //     'null' => true,
            // ],
        ]);

        // Menjadikan 'id' sebagai Primary Key
        $this->forge->addKey('id', true);

        // Membuat tabel 'products'
        $this->forge->createTable('products');
    }

    public function down()
    {
        // Menghapus tabel 'products' jika migrasi di-rollback
        $this->forge->dropTable('products');
    }
}
```

7. Menjalankan Migrasi

Setelah file migrasi diisi dengan benar, kembali ke terminal kontainer dan jalankan perintah berikut:

php spark migrate

Hasil yang Diharapkan: Anda akan melihat pesan yang memberitahukan bahwa migrasi telah berhasil dijalankan.

Running all new migrations...

Running: (App) 2025-07-02-043000_CreateProductsTable

Done migrations.

Sekarang, tabel products dengan struktur yang telah kita definisikan sudah berhasil dibuat di dalam database ci4_test_db. Lingkungan kita sudah siap untuk memulai siklus TDD.

---

5. Studi Kasus dan Penyelesaian

Kasus: Membangun modul sederhana untuk menampilkan daftar produk dari database menggunakan pendekatan TDD.

Penyelesaian: Proses penyelesaian studi kasus ini **sama persis** dengan modul sebelumnya. Perubahan server web dari Nginx ke Apache tidak memengaruhi cara kita menulis atau menguji kode aplikasi di CodeIgniter.

A. Unit Testing untuk ProductModel (Panduan Detail)

Kita akan membuat dan menguji ProductModel untuk memastikan model ini dapat berinteraksi dengan database dengan benar, khususnya untuk mengambil semua data produk.

Tahap 1: 🔴 RED - Menulis Tes yang Gagal

Tujuan: Tujuan dari tahap ini adalah menulis sebuah tes untuk fungsionalitas yang *belum ada*. Tes ini **wajib gagal**. Kegagalan ini membuktikan dua hal: (1) Tes yang kita tulis sudah benar dan mampu mendeteksi ketiadaan fitur, dan (2) Fitur yang akan kita buat memang belum diimplementasikan.

Langkah 1: Buat File Tes Dari **dalam terminal kontainer**, jalankan perintah spark untuk membuat kerangka file tes.

```
# Pastikan Anda berada di dalam kontainer app
php spark make:test Models/ProductModelTest
```

Perintah ini akan membuat file baru di **src/tests/app/Models/ProductModelTest.php**.

Langkah 2: Tulis Kode Tes Buka **file ProductModelTest.php** tersebut dan ganti seluruh isinya dengan kode berikut. Kode ini mendefinisikan kasus uji untuk mengambil semua produk.

```
<?php

namespace App\Models;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait; // Trait penting untuk tes database

class ProductModelTest extends CIUnitTestCase
```

```
{
    // Mengaktifkan fitur tes database dari CodeIgniter
    use DatabaseTestTrait;

    // Variabel ini memastikan migrasi dijalankan sebelum tes di kelas ini
    protected $migrate = true;

    public function testFindAllProducts()
    {
        // ARRANGE: Persiapan data dan objek yang dibutuhkan
        // Kita akan mencoba membuat instance dari model yang belum ada
        $model = new ProductModel();

        // Masukkan beberapa data dummy ke database testing
        $model->insertBatch([
            ['name' => 'Laptop A', 'price' => 1500.00],
            ['name' => 'Mouse B', 'price' => 25.00],
        ]);

        // ACT: Panggil metode yang ingin kita uji
        $products = $model->findAll();

        // ASSERT: Lakukan penegasan atau verifikasi hasil
        // 1. Pastikan jumlah produk yang dikembalikan adalah 2
        $this->assertCount(2, $products);
        // 2. Pastikan nama produk pertama adalah 'Laptop A'
        $this->assertEquals('Laptop A', $products[0]['name']);
    }
}
```

Langkah 3: Jalankan Tes dan Verifikasi Kegagalan Kembali ke terminal kontainer dan jalankan PHPUnit hanya untuk file tes yang baru kita buat.

./vendor/bin/phpunit tests/app/Models/ProductModelTest.php
Hasil yang Diharapkan: Anda akan melihat pesan **ERROR** berwarna **MERAH** di terminal, yang menyatakan:

Error: Class "App\Models\ProductModel" not found
Penjelasan: Ini adalah hasil yang kita inginkan! Tes kita gagal karena ProductModel memang belum ada. Tahap **RED** berhasil.

Tahap 2: ✅ GREEN - Menulis Kode Minimal untuk Lulus Tes

Tujuan: Menulis kode sesederhana mungkin, tanpa hiasan, yang cukup untuk membuat tes yang sebelumnya gagal menjadi berhasil. Fokus kita bukan pada kode yang sempurna, tetapi pada kode yang *memenuhi persyaratan tes*.

Langkah 1: Buat File Model Gunakan spark untuk membuat file ProductModel.

php spark make:model ProductModel
Perintah ini akan membuat file baru di **src/app/Models/ProductModel.php**.

Langkah 2: Konfigurasi Model Buka file **ProductModel.php** dan konfigurasikan seperti berikut. Ini adalah konfigurasi minimal yang dibutuhkan oleh CodeIgniter untuk menghubungkan model ke tabel database.

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class ProductModel extends Model
{
    // Nama tabel di database
    protected $table            = 'products';
    // Nama primary key dari tabel
    protected $primaryKey       = 'id';
    // Kolom yang diizinkan untuk diisi (penting untuk insert/update)
    protected $allowedFields    = ['name', 'price'];
}
```

Langkah 3: Jalankan Tes Kembali Sekarang, jalankan lagi perintah tes yang sama.

./vendor/bin/phpunit tests/app/Models/ProductModelTest.php
Hasil yang Diharapkan: Kali ini, Anda akan melihat pesan **OK** berwarna **HIJAU**.

OK (1 test, 2 assertions)
Penjelasan: Keberhasilan ini menandakan bahwa kode yang baru kita tulis telah memenuhi semua assertion (penegasan) yang kita definisikan di dalam tes. Fungsionalitas dasar kita sudah bekerja. Tahap **GREEN** berhasil.

Tahap 3: ♻️ REFACTOR - Membersihkan dan Memperbaiki Kode

Tujuan: Setelah kita memiliki fungsionalitas yang bekerja dan dilindungi oleh tes, sekarang saatnya untuk memperbaiki kualitas internal kode tersebut. Ini bisa berarti membuat kode lebih mudah dibaca, menghilangkan duplikasi, atau meningkatkan performa, **tanpa mengubah perilakunya**. Tes yang sudah ada bertindak sebagai jaring pengaman.

Skenario Refactoring: Untuk **ProductModel** kita yang sederhana, kode yang dihasilkan oleh CodeIgniter sudah cukup bersih. Namun, dalam skenario yang lebih kompleks, tahap refactor bisa mencakup:

a) Menambahkan PHPDoc (komentar) untuk menjelaskan fungsi metode.

b) Memindahkan query yang kompleks ke dalam metode kustom di dalam model.

c) Mengoptimalkan cara data diambil dari database.

Langkah: Jalankan Tes untuk Memastikan Tidak Ada yang Rusak Meskipun kita tidak melakukan perubahan kode, praktik yang baik setelah tahap refactor adalah menjalankan **seluruh rangkaian tes** untuk memastikan perubahan kita tidak merusak bagian lain dari aplikasi.

./vendor/bin/phpunit

Penjelasan: Jika semua tes tetap hijau, artinya proses refactoring kita berhasil. Jika ada tes yang menjadi merah, kita tahu persis bahwa perubahan terakhir yang kita buat adalah penyebabnya, sehingga kita bisa memperbaikinya dengan cepat. Ini adalah kekuatan utama dari jaring pengaman TDD.

Perhatikan bahwa semua perintah **php spark** dan **./vendor/bin/phpunit** tetap sama dan dijalankan dari dalam kontainer app.

B. Integration Testing untuk ProductController (Panduan Detail)

Kita akan menguji keseluruhan alur aplikasi: dari permintaan HTTP yang masuk ke *Route*, yang kemudian dieksekusi oleh *Controller*, yang memanggil *Model* untuk mengambil data, dan akhirnya merender *View* yang menampilkan data tersebut.

Tahap 1: 🔴 RED - Menulis Tes yang Gagal

Tujuan: Menulis sebuah tes yang mensimulasikan pengguna membuka halaman daftar produk di browser. Tes ini **wajib gagal** karena *Route*, *Controller*, dan *View* untuk fitur ini belum ada sama sekali.

Langkah 1: Buat File Tes Dari **dalam terminal kontainer**, jalankan perintah spark untuk membuat kerangka file tes Controller.

php spark make:test Controllers/ProductControllerTest

Perintah ini akan membuat file baru di **src/tests/app/Controllers/ProductControllerTest.php**.

Langkah 2: Tulis Kode Tes Integrasi Buka file ProductControllerTest.php dan ganti seluruh isinya dengan kode berikut.

```
<?php

namespace App\Controllers;
```

```
use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\ControllerTestTrait; // Trait untuk mensimulasikan request HTTP
use CodeIgniter\Test\DatabaseTestTrait;
use App\Models\ProductModel;

class ProductControllerTest extends CIUnitTestCase
{
    use ControllerTestTrait, DatabaseTestTrait;

    protected $migrate = true;

    public function testIndexShowsProducts()
    {
        // ARRANGE: Siapkan data dummy di database
        $model = new ProductModel();
        $model->insert(['name' => 'Kamera C', 'price' => 750.00]);

        // ACT: Simulasikan request GET ke URL /products
             //  Ini  akan  mencoba  menjalankan  metode  'index'  di 'ProductController'
        $result = $this->withURI('http://localhost:8080/products')
                       ->controller(ProductController::class)
                       ->execute('index');

        // ASSERT: Verifikasi hasil dari request
        // 1. Pastikan response HTTP berhasil (status code 200 OK)
        $this->assertTrue($result->isOK());
       // 2. Pastikan halaman HTML yang dihasilkan mengandung teks 'Daftar Produk'
        $this->assertTrue($result->see('Daftar Produk'));
         // 3. Pastikan halaman HTML juga menampilkan nama produk dari database
        $this->assertTrue($result->see('Kamera C'));
    }
}
```

Langkah 3: Jalankan Tes dan Verifikasi Kegagalan Kembali ke terminal kontainer dan jalankan PHPUnit untuk file tes ini.

```
./vendor/bin/phpunit tests/app/Controllers/ProductControllerTest.php
```

Hasil yang Diharapkan: Anda akan melihat pesan **ERROR** berwarna **MERAH**, yang menyatakan:

```
Error: Class "App\Controllers\ProductController" not found
```

Penjelasan: Ini adalah kegagalan yang kita harapkan. Tes kita gagal karena ProductController belum kita buat. Tahap **RED** berhasil.

Tahap 2: ✅ GREEN - Menulis Kode Minimal untuk Lulus Tes

Tujuan: Membuat semua komponen yang diperlukan (Route, Controller, View) dengan kode paling minimalis agar tes yang sebelumnya gagal menjadi berhasil.

Langkah 1: Buat Controller Gunakan spark untuk membuat file Controller. Opsi --restful akan membuat kerangka metode standar (index, show, create, dll.).

php spark make:controller Product --restful
Buka file *src/app/Controllers/Product.php* dan isi metode index():

```
<?php

namespace App\Controllers;

use App\Models\ProductModel;
use CodeIgniter\RESTful\ResourceController;

class Product extends ResourceController
{
    public function index()
    {
        $model = new ProductModel();
        $data = [
            'title'    => 'Daftar Produk',
            'products' => $model->findAll()
        ];
        return view('products/index', $data);
    }
}
```

Langkah 2: Buat View Buat folder products di dalam **src/app/Views/**. Kemudian, buat file baru di dalamnya bernama index.php. Lokasi: **src/app/Views/products/index.php** Isi dengan kode HTML sederhana berikut:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title><?= esc($title) ?></title>
</head>
<body>
    <h1><?= esc($title) ?></h1>
    <ul>
        <?php foreach ($products as $product): ?>
            <li><?= esc($product['name']) ?> - $<?= esc($product['price'])
?></li>
        <?php endforeach; ?>
    </ul>
</body>
</html>
```

Langkah 3: Tambahkan Route Buka file **src/app/Config/Routes.php** dan tambahkan baris berikut untuk memetakan URL /products ke ProductController.

```
// ... (di bawah baris $routes->get('/', 'Home::index');)
```

```
$routes->resource('products', ['controller' => 'Product']);
```

Langkah 4: Jalankan Tes Kembali Dengan semua komponen sudah di tempat, jalankan lagi perintah tes.

./vendor/bin/phpunit tests/app/Controllers/ProductControllerTest.php

Hasil yang Diharapkan: Kali ini, Anda akan melihat pesan **OK** berwarna **HIJAU**.

OK (1 test, 3 assertions)

Penjelasan: Tes berhasil karena permintaan ke /products sekarang dapat diproses dengan benar oleh semua komponen yang telah kita buat. Tahap **GREEN** berhasil.

Tahap 3: ♻️ REFACTOR - Membersihkan dan Memperbaiki Kode

Tujuan: Memperbaiki kualitas internal kode Controller tanpa mengubah perilakunya. Tes yang sudah hijau akan menjadi jaring pengaman kita.

Skenario Refactoring: Saat ini, ProductModel dibuat langsung di dalam metode index(). Jika nanti kita membuat metode lain (seperti show() atau create()), kita harus membuat instance model lagi. Ini tidak efisien dan menyebabkan duplikasi kode. Praktik yang lebih baik adalah menginisialisasi model di *constructor* Controller.

Langkah 1: Perbaiki Kode Controller Buka kembali **src/app/Controllers/Product.php** dan ubah kodenya menjadi seperti ini:

```
<?php

namespace App\Controllers;

use App\Models\ProductModel;
use CodeIgniter\RESTful\ResourceController;

class Product extends ResourceController
{
    // Definisikan properti untuk menampung model
    protected $model;

    // Gunakan constructor untuk menginisialisasi model
    public function __construct()
    {
        $this->model = new ProductModel();
    }

    public function index()
    {
        $data = [
            'title'    => 'Daftar Produk',
            // Gunakan model yang sudah ada di properti
            'products' => $this->model->findAll()
```

```
        ];
        return view('products/index', $data);
    }
}
```

Penjelasan: Kode ini secara fungsional sama, tetapi strukturnya lebih baik. Properti **$this->model** sekarang dapat digunakan oleh semua metode lain di dalam kelas ini tanpa perlu membuat instance baru.

Langkah 2: Jalankan Seluruh Rangkaian Tes Setelah melakukan refactoring, sangat penting untuk menjalankan **semua tes** untuk memastikan perubahan kita tidak merusak fungsionalitas yang ada, termasuk Unit Test yang kita buat sebelumnya.

./vendor/bin/phpunit

Hasil yang Diharapkan: Semua tes (baik Unit maupun Integration) harus tetap **HIJAU**. Ini membuktikan bahwa refactoring kita berhasil dan aman.

---

6. Tugas

Berdasarkan studi kasus yang telah diselesaikan, kembangkan fitur baru untuk **melihat detail satu produk**. Terapkan siklus TDD dalam pengerjaannya.

Langkah-langkah Pengerjaan Tugas: Tugas ini juga tidak berubah. Ikuti langkah-langkah yang sama seperti pada modul sebelumnya untuk menyelesaikan fitur ini, dan pastikan semua perintah pengujian dijalankan dari **dalam terminal kontainer**.

1. **Tulis Tes Terlebih Dahulu (RED)**: Buat metode tes testShowFindsSingleProduct() di ProductControllerTest yang mensimulasikan GET ke /products/1 dan pastikan tes tersebut gagal.
2. **Implementasikan Fitur (GREEN)**: Buat metode show($id = null) di ProductController beserta view-nya (products/show.php) untuk membuat tes berhasil.
3. **Jalankan Tes Lagi**: Pastikan tes yang sebelumnya gagal kini berhasil.
4. **Refactor (Jika Perlu)**: Perbaiki kode tanpa mengubah fungsionalitasnya, dan pastikan semua tes tetap berhasil.