

MODUL 7. LARAVEL LANJUT

Tujuan Praktikum

1. Mahasiswa mampu memahami konsep dan implementasi CodeIgniter pada pembuatan aplikasi berbasis *web*.
2. Mahasiswa mampu menerapkan CRUD menggunakan Laravel dan konsep MVC.
3. Mahasiswa mampu mengimplementasikan fitur-fitur yang sering digunakan pada Laravel.

7.1. CRUD

Pada modul ini kita akan membuat aplikasi yang dapat melakukan create, read, update, dan delete terhadap suatu data/tabel. Aplikasi yang akan kita bangun yaitu aplikasi e-commerce dimana kita akan fokus melakukan CRUD terhadap tabel produknya saja.

7.1.1. Konfigurasi dan Skema

Hal pertama yang harus dilakukan agar aplikasi dapat terhubung dengan database adalah mengatur konfigurasi koneksi. Secara detail, konfigurasi database berada pada file **config/database.php**. Tetapi untuk konfigurasi standar, kita dapat mengubahnya di file **.env** pada baris yang berisi **DB_CONNECTION** hingga **DB_PASSWORD**. Ubah nilainya dengan pengaturan yang kita inginkan.

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ecommerce
DB_USERNAME=root
DB_PASSWORD=
...
```

Buat database bernama **ecommerce** di DBMS. Untuk membuat tabelnya, dapat dilakukan secara otomatis (di-generate) oleh Laravel dari command prompt / console / terminal dengan perintah. Perintah yang pertama yaitu membuat file migrasi sebagai skema dari tabel:

```
php artisan make:migration create_products_table
```

File migrasi **xxx_create_products_table.php** akan terbuat pada folder **database/migrations**. Edit file ini pada fungsi **Schema::create()** untuk mendefinisikan kolom yang kita inginkan.

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->integer('price');
    $table->timestamps();
});
```

Penjabaran dari kode ini:

- Fungsi **id()** yaitu kita mendefinisikan kolom **id** sebagai PK dengan tipe int dan auto-increment.
- Fungsi **string('name')** yaitu kita mendefinisikan kolom **name** dengan tipe varchar
- Fungsi **integer('price')** yaitu kita mendefinisikan kolom **price** dengan tipe int
- Fungsi **timestamps()** yaitu kita mendefinisikan kolom **created_at** yang akan terisi jika data dibuat melalui ORM dan **updated_at** yang akan terisi jika data diubah melalui ORM

Untuk membuat tabel ke database dari skema ini, maka ketikkan perintah berikut pada command prompt / console / terminal:

```
php artisan migrate
```

7.1.2. Model

Laravel memberikan tiga cara untuk mengakses ataupun manipulasi data ke database: query langsung, query builder, dan ORM. Query langsung dan query builder menggunakan library Illuminate (**Illuminate\Support\Facades\DB**) sedangkan ORM menggunakan Eloquent, yang merupakan kelas extend dari Illuminate. File Model diletakkan pada folder **app/Models**. Untuk membuat Model dalam Laravel, dapat dilakukan dengan manual dengan menambahkan namespace "App\Models" dan meng-extend kelas base Model dari Eloquent (**Illuminate\Database\Eloquent\Model**) ataupun di-generate oleh Laravel dari command prompt / console / terminal dengan perintah:

```
php artisan make:model Product
```

Jika sebelumnya belum membuat Controller atau file migration-nya, perintah generate Model ini dapat ditambahkan parameter sehingga bisa sekaligus me-generate file Controller-nya (-c), file migration-nya (-m), ataupun keduanya langsung (-cm).

Eloquent memiliki beberapa konvensi / aturan yang harus diperhatikan yaitu:

- Nama sebuah Model "X" secara otomatis merepresentasikan tabel database bernama "xs". Contoh, Model Product akan merepresentasikan tabel **products**. Jika tabel yang direpresentasikan berbeda, maka harus ditambahkan atribut **protected \$table = 'nama_tabel'**; pada kelas Model.
- Kolom PK pada tabel bernama "id". Jika kolom PK bukan "id", maka harus ditambahkan atribut **protected \$primaryKey = 'nama_kolom_PK'**; pada kelas Model.
- Kolom PK pada tabel di-set auto-increment. Jika kolom PK tidak auto-increment, maka harus ditambahkan atribut **public \$incrementing = false**; pada kelas Model.
- Kolom PK pada tabel bertipe integer. Jika kolom PK bukan integer, maka harus ditambahkan atribut **protected \$keyType = 'string'**; pada kelas Model.
- Ada kolom "created_at" dan "updated_at" pada tabel. Jika tidak ada, maka harus ditambahkan atribut **public \$timestamps = false**; pada kelas Model.

Beberapa aturan lain juga dapat diatur seperti date format, koneksi DB yang berbeda, dan nilai default untuk atribut tertentu.

7.1.3. Controller

Setelah Model siap, kita tinggal memanggilnya pada Controller. Berikut kode lengkapnya dalam ProductController:

```
...
use App\Models\Product;

class ProductController extends Controller {
    public function index()
    {
        $prods = Product::get();
        return view('product.index', ['list' => $prods]);
    }
}
```

```

public function create()
{
    return view('product.form', [
        'title' => 'Tambah',
        'method' => 'POST',
        'action' => 'product'
    ]);
}

public function store(Request $request)
{
    $prod = new Product;
    $prod->name = $request->name;
    $prod->price = $request->price;
    $prod->save();
    return redirect('/product')->with('msg', 'Tambah berhasil');
}

public function show($id)
{
    return Product::find($id);
}

public function edit($id)
{
    return view('product.form', [
        'title' => 'Edit',
        'method' => 'PUT',
        'action' => "product/$id",
        'data' => Product::find($id)
    ]);
}

public function update(Request $request, $id)
{
    $prod = Product::find($id);
    $prod->name = $request->name;
    $prod->price = $request->price;
    $prod->save();
    return redirect('/product')->with('msg', 'Edit berhasil');
}

public function destroy($id)
{
    Product::destroy($id);
    // atau
    /* $prod = Product::find($id);
    $prod->delete(); */
    return redirect('/product')->with('msg', 'Hapus berhasil');
}
}

```

7.1.4. View

Untuk tampilannya, buat file dengan nama **index.blade.php** yang diletakkan di folder **resources/views/product** (folder product dibuat dahulu agar rapi).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Daftar Produk</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css"
integrity="sha384-zCbKRCUGaJDkqS1kPbPd7TveP5iyJE0EjAuZQTgFLD2ylzuqKfdKlfG/eSrtx
Ukn" crossorigin="anonymous">
    <script src="{{ asset('assets/jquery.js') }}"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.
js"
integrity="sha384-fQybjgWLrvvRgtW6bF1B7jaZrFsaBXjsOMm/tB9LTS58ONXgqbR9W8oWht/am
npF" crossorigin="anonymous"></script>
  </head>
  <body style="width:95%">
    <div class="row justify-content-center" style="margin-top:13%">
      <div class="col-4">
        <span class="float-left">{{ session('msg') }}</span>
        <a href="/product/create" class="btn btn-secondary
float-right">Tambah</a><br /><br />
        <table class="table table-bordered table-striped">
          <tr>
            <th>Nama</th>
            <th>Harga</th>
            <th>Aksi</th>
          </tr>
          @foreach($list as $d)
            <tr>
              <td>{{ $d->name }}</td>
              <td>{{ $d->price }}</td>
              <td>
                <a href="/product/{{ $d->id }}/edit" class="btn
btn-primary">Edit</a>
                <form method="post" action="/product/{{ $d->id }}"
style="display:inline" onsubmit="return confirm('Yakin hapus?')">
                  @csrf
                  @method('DELETE')
                  <button class="btn btn-danger">Hapus</button>
                </form>
              </td>
            </tr>
          @endforeach
        </table>
      </div>
    </div>
  </body>
</html>
```

Dan untuk tampilan form tambah dan edit, dapat dibuat satu halaman saja karena hampir sama. Hanya perlu tambahan pengkondisian di beberapa tempat.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form {{ $title }} Produk</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css"
integrity="sha384-zCbKRCUGaJDkqS1kPbPd7TveP5iyJE0EjAuZQTgFLD2ylzuqKfdKlfG/eSrtx
Ukn" crossorigin="anonymous">
```

```

<script src="{{ asset('assets/jquery.js') }}"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.
js"
integrity="sha384-fQybjgWLrvvRgtW6bFlB7jaZrFsaBXjsOMm/tB9LTS58ONXgqbR9W8oWht/an
npF" crossorigin="anonymous"></script>
</head>
<body style="width:95%">
<div class="row justify-content-center" style="margin-top:13%">
<div class="col-3">
<h4>Form {{ $title }} Produk</h4>
<form class="border" style="padding:20px" method="POST" action="/{{
$action }}">
@csrf
<input type="hidden" name="_method" value="{{ $method }}" />
<div class="form-group">
<label>Nama</label>
<input type="text" name="name" class="form-control" value="{{
isset($data)?$data->name:'' }}" />
</div>
<div class="form-group">
<label>Harga</label>
<input type="number" name="price" class="form-control" value="{{
isset($data)?$data->price:'' }}" />
</div>
<div style="text-align:center">
<button class="btn btn-success">Simpan</button>
</div>
</form>
</div>
</div>
</body>
</html>

```

7.1.5. Tampilan Halaman

Jalankan aplikasi. Jika *kode* yang dibuat sesuai dengan semua gambar di atas pada modul ini, maka tampilan aplikasi *webnya* akan seperti gambar dibawah ini.

1. <http://localhost:8000/product>

Tambah

Nama	Harga	Aksi

Gambar 7.1 Tampilan halaman view

2. <http://localhost:8000/product/create>

Form Tambah Produk



A form titled "Form Tambah Produk" with two input fields: "Nama" and "Harga". Below the fields is a green button labeled "Simpan".

Gambar 7.2 Tampilan halaman form tambah produk

3. <http://localhost:8000/product>

Tambah berhasil

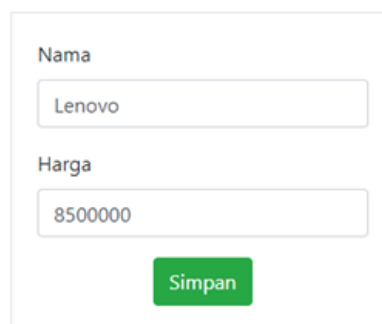
Tambah

Nama	Harga	Aksi
Lenovo	8500000	Edit Hapus

Gambar 7.3 Tampilan halaman view setelah tambah data

4. [http://localhost:8000/product/\[id\]/edit](http://localhost:8000/product/[id]/edit)

Form Edit Produk



A form titled "Form Edit Produk" with two input fields: "Nama" (containing "Lenovo") and "Harga" (containing "8500000"). Below the fields is a green button labeled "Simpan".

Gambar 7.4 Tampilan halaman form ubah data

7.2. Templating Halaman

Sebelumnya kita sudah menggunakan beberapa directive Blade. Directive ini juga dapat digunakan untuk templating halaman, yaitu bagian-bagian dari halaman yang sama / berulang untuk semua halaman dapat dibuat menjadi satu file Blade, dan semua halaman itu dapat me-load-nya tanpa harus ditulis ulang.

Langkah pertama dalam templating adalah membuat file template-nya, yang dimiliki oleh tiap halaman. Untuk contoh kasus pada modul ini, kita membuat file template dengan nama **template.blade.php** dengan isi:

```
<!DOCTYPE html>
<html>
  <head>
    <title>@yield('title')</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css"
integrity="sha384-zCbKRCUGaJDkqS1kPbPd7TveP5iyJE0EjAuZQTgFLD2ylzuqKfdKlfG/eSrtx
Ukn" crossorigin="anonymous">
    <script src="{{ asset('assets/jquery.js') }}"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.
js"
integrity="sha384-fQybjgWLrvvRgtW6bFlB7jaZrFsaBXjsOMm/tB9LTS58ONXgqbR9W8oWhT/am
npF" crossorigin="anonymous"></script>
  </head>
  <body style="width:95%">
    @if(session('email'))
      <div class="row justify-content-end" style="margin-top:2%">
        <div class="col-3">
          {{ session('name') }}
          <a href="/logout" class="btn btn-warning">Logout</a>
        </div>
      </div>
    @endif
    <div class="row justify-content-center" style="margin-top:13%">
      @yield('content')
    </div>
  </body>
</html>
```

Directive **@yield** digunakan untuk menandakan bahwa di bagian itulah yang akan berbeda untuk tiap halamannya. Langkah selanjutnya adalah mengubah tampilan di tiap halaman seperti contoh **index.blade.php** di bawah ini:

```
@extends('template')

@section('title', 'Daftar Produk')

@section('content')
  <div class="col-4">
    <span class="float-left">{{ session('msg') }}</span>
    <a href="/product/create" class="btn btn-secondary
float-right">Tambah</a><br /><br />
    <table class="table table-striped table-bordered">
      <tr>
        <th>Nama</th>
        <th>Harga</th>
        <th>Aksi</th>
      </tr>
      @foreach($list as $d)
```

```

<tr>
    <td>{{ $d->name }}</td>
    <td>{{ $d->price }}</td>
    <td>
        <a href="/product/{{ $d->id }}/edit" class="btn btn-primary">Edit</a>
        <form method="post" action="/product/{{ $d->id }}" onsubmit="return
confirm('Yakin hapus?')" style="display:inline">
            @csrf
            @method('DELETE')
            <button class="btn btn-danger">Hapus</button>
        </form>
    </td>
</tr>
@endforeach
</table>
</div>
@endsection

```

Directive **@extend** digunakan untuk menentukan file template mana yang digunakan oleh halaman ini. Sedangkan directive **@section** digunakan untuk mengisi halaman sesuai dengan nama **yield** yang di-define dalam file template. Directive **@section** dapat diisi dengan string ataupun tag HTML.

7.3. Form Validation

Laravel dapat ditambahkan form validation dari sisi server. Di controller, sebelum kita memanggil fungsi untuk menambahkan (fungsi **store**) / mengubah (fungsi **update**), kita dapat menambahkan kode:

```

...
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required|min:4',
        'price' => 'required|integer|min:1000000'
    ]);
    ...
public function update(Request $request)
{
    $this->validate($request, [
        'name' => 'required|min:4',
        'price' => 'required|integer|min:1000000'
    ]);
    ...

```

Di kode validasi ini, kita mengatur nama tidak boleh kosong dan minimal 4 karakter, sedangkan harga tidak boleh kosong, harus integer, dan nilai minimal 1.000.000. Kemudian di halaman **form.blade.php** kita harus menambahkan directive **@error** seperti ini:

```

@extends('template')

@section('title')
    Form {{ $title }} Produk
@endsection

@section('content')
    <div class="col-3">
        <h4>Form {{ $title }} Produk</h4>
        <form class="border" style="padding:20px" method="POST" action="/{{ $action
    }}">
            @csrf
            <input type="hidden" name="_method" value="{{ $method }}" />
            <div class="form-group">
                <label>Nama</label>

```



```

        <input type="text" name="name" class="form-control @error('name')
is-invalid @enderror" value="{{ isset($data)?$data->name:old('name') }}" />
        @error('name')
        <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>
    <div class="form-group">
        <label>Harga</label>
        <input type="number" name="price" class="form-control @error('price')
is-invalid @enderror" value="{{ isset($data)?$data->price:old('price') }}" />
        @error('price')
        <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>
    <div style="text-align:center">
        <button class="btn btn-success">Simpan</button>
    </div>
</form>
</div>
@endsection

```

Directive **@error** diletakkan di atribut class pada tag `<input>` untuk validasinya dan diletakkan di bawah tag `<input>` untuk menampilkan pesan validasinya. Kemudian agar data pada form di-repopulate, maka isikan atribut value pada tag `<input>` dengan **old('value-di-atribut-name')**.

7.4. Session

Session memiliki fungsi menyimpan suatu variabel pada server dan variabel ini dapat diakses dan diubah dimanapun: Routing, Controller, ataupun halaman manapun. Session default pada Laravel dikelola sendiri oleh Laravel dan disimpan dalam bentuk file. Selain file, Session pada Laravel juga dapat diubah menjadi disimpan ke database atau menggunakan mekanisme lain.

Laravel memiliki dua jenis Session, yaitu Session biasa dan Session flash. Session biasa akan selalu ada selama belum dihapus, time-out, atau browser ditutup. Sedangkan Session flash adalah session yang hanya berlaku untuk satu request saja, setelah itu Laravel akan menghapusnya secara otomatis. Contoh Session flash sudah pernah kita gunakan melalui fungsi **with('x', y)** yang digunakan bersamaan dengan fungsi **redirect**. Untuk Session biasa, berikut contoh penggunaannya di Routing:

```

...
Route::get('/login', function () {
    if (session()->has('email')) return redirect('/product');
    return view('login');
});

Route::get('/logout', function () {
    session()->flush();
    return redirect('/login');
});
...

```

Fungsi **session()->has('x')** digunakan untuk memeriksa apakah ada Session bernama "x". Sedangkan **session()->flush()** digunakan untuk menghapus semua Session. Contoh lain, berikut penggunaan Session di SiteController:

```

...
public function auth(Request $req) {
    $u = User::where([
        ['email', $req->em],
        ['password', $req->pwd],
    ])->first();
    if (isset($u)) {
        session()->put('email', $u->email);
    }
}

```

```

    session()->put('name', $u->name);
    return "<script>
    alert('Welcome, " . session('name') . "');
    location.href='/product';
    </script>";
    }
    return redirect('/login')->with('msg', 'Email / password salah');
}
...

```

Fungsi **session()->put('x', y)** digunakan untuk membuat Session bernama "x" dengan nilai y. Sedangkan **session('x')** digunakan untuk mengambil nilai dari Session "x".

7.5. Middleware

Middleware pada Laravel adalah suatu mekanisme untuk menyaring request yang masuk ke suatu Routing. Sebagai contoh, kita bisa membuat Middleware untuk memverifikasi apakah seorang user sudah terautentikasi atau memiliki hak akses untuk mengakses URL. Jika user belum terautentikasi atau tidak memiliki akses, maka Middleware dapat me-redirect user tersebut ke URL lain. Sebaliknya jika user terautentikasi atau memiliki hak akses, maka Middleware akan meneruskan request ke aksi yang dipetakan di Routing.

Middleware dapat melakukan hal lain selain untuk autentikasi, misalnya untuk menulis log atau error yang terjadi. Dan Middleware dapat kita atur mekanisme yang dilakukannya, yaitu bisa sebelum request diteruskan atau sesudah request diteruskan. Middleware yang kita buat harus berada di folder **app/Http/Middleware**.

Sekarang kita akan membuat autentikasi dengan menggunakan library **Auth**. Library ini sudah termasuk di dalam paket instalasi Laravel, sehingga kita tidak perlu menambahkannya lagi melalui composer. Dalam library ini sudah mencakup semua fitur untuk autentikasi, registrasi, dan middleware-nya. Dengan menggunakan **Auth**, autentikasi menjadi lebih simpel, aman, dan bisa menjadi alternatif pengganti yang lebih baik dari Session.

Langkah pertama adalah pengaturan pada Routing. Ganti Routing untuk login, logout, dan product:

```

...
Route::get('/login', function () {
    if (Auth::check()) return redirect('/product');
    return view('login');
})->name('login');

Route::get('/logout', function () {
    Auth::logout();
    return redirect('/login');
});

Route::resource('product', ProductController::class)->middleware('auth');

```

Fungsi **Auth::check()** digunakan untuk memeriksa apakah user telah terautentikasi. Routing login kita berikan nama alias karena kebutuhan dari Middleware **auth**. Sedangkan **Auth::logout()** digunakan untuk menghapus autentikasi. Untuk product, Middleware kita ganti dengan Middleware **auth** yang berkolaborasi dengan library Auth.

Lalu langkah kedua ubah isi dari fungsi **auth** di **SiteController**:

```

...
use Illuminate\Support\Facades\Auth;

```

```

class SiteController extends Controller
{
    public function auth(Request $req) {
        if (Auth::attempt(['email'=>$req->em, 'password'=>$req->pwd])) {
            //jika ada nilai lain selain data user yang ingin disimpan di session, baru
            gunakan session disini
            return redirect('/product');
        }
        return redirect('/login')->with('msg', 'Email / password salah');
    }
    ...
}

```

Fungsi **Auth::attempt()** digunakan untuk proses autentikasi, yaitu apakah email dan password yang di-submit ada dalam database pada tabel **users**. Jika email dan password cocok maka akan menghasilkan **true**. Hal yang harus diperhatikan adalah ketika menambahkan data User ke database, pastikan isi dari kolom password di-hash dengan metode **Bcrypt** agar dapat menggunakan library Auth ini. Laravel sudah menyediakan fungsi **bcrypt('x')** untuk mempermudahnya.

Langkah ketiga, dalam View yang sebelumnya menggunakan **@if(session('x'))** dapat kita ganti menjadi directive **@auth** untuk pengecekan apakah user telah terautentikasi, dan dapat menggunakan **Auth::user()** untuk mengakses data user yang login. Berikut contohnya dalam **template.blade.php**:

```

...
<body style="width:95%">
    @auth
        <div class="row justify-content-end" style="margin-top:2%">
            <div class="col-3">
                {{ Auth::user()->name }}
                <a href="/logout" class="btn btn-warning">Logout</a>
            </div>
        </div>
    @endauth
    <div class="row justify-content-center" style="margin-top:10%">
        @yield('content')
    </div>
...

```

7.6 Model Relasi

Model Eloquent menyediakan fasilitas agar dua Model yang berelasi dapat langsung memanggil satu sama lain. Kita akan mencoba salah satunya yaitu relasi one to many, dengan menggunakan contoh kasus relasi **Product** dengan **Variant**. Tiap **Product** dapat memiliki banyak **Variant**, tetapi tiap **Variant** hanya dimiliki oleh satu **Product**. Langkah awal yaitu membuat Model dan file migration-nya, maka perintahnya:

```
php artisan make:model Variant -m
```

Setelah itu edit file **xxx_create_variants_table.php** pada folder **database/migrations** untuk mendefinisikan kolomnya:

```

...
Schema::create('variants', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->text('description');
    $table->string('processor');
    $table->string('memory');
    $table->string('storage');
    $table->foreignId('product_id')->constrained();
    $table->timestamps();
});

```

```
});  
...
```

Untuk kebutuhan foreign key, kita menggunakan **foreignId** dan **constrained** jika tabel entitas yang diacu mengikuti konvensi Model Eloquent (PK bernama "id", tabel database ditambah akhiran "s", dsb). Jika berbeda, maka pendefinisian foreign key harus menggunakan **foreign**, **references**, dan **on**. Contoh:

```
...  
$table->integer('product_id');  
$table->foreign('product_id')->references('id_product')->on('product');  
...
```

Kemudian generate tabel **variants** dengan perintah seperti sebelumnya (php artisan migrate). Selanjutnya edit file Model **Variants**, tambahkan fungsi berikut:

```
...  
public function product() {  
    return $this->belongsTo(Product::class);  
}  
...
```

Fungsi **belongsTo()** secara otomatis akan memanggil object **Product** yang berelasi dengan dirinya berdasarkan **product_id**. Kemudian tambahkan juga fungsi ke dalam Model **Product**:

```
...  
public function variants() {  
    return $this->hasMany(Variant::class);  
}  
...
```

Fungsi **hasMany()** secara otomatis akan memanggil semua object **Variant** yang berelasi dengan dirinya berdasarkan **product_id**. Untuk mencobanya kita akan menambahkan satu kolom pada tampilan tabel di **index.blade.php** untuk menampilkan data variant dari tiap product:

```
...  
<th>Harga</th>  
<th>Variant</th>  
<th>Aksi</th>  
</tr>  
<@foreach($list as $d)>  
<tr>  
<td>{{ $d->name }}</td>  
<td>{{ $d->price }}</td>  
<td>  
<ul>  
<@foreach($d->variants()->get() as $var)>  
<li>{{ $var->name }}</li>  
<Desc: {{ $var->description }}<br />  
<Proc: {{ $var->processor }}<br />  
<RAM: {{ $var->memory }}<br />  
<Strg: {{ $var->storage }}<br />  
<Product: {{ $var->product->name }}</td>  
</ul>  
</td>  
<td align="center">  
...</td>  
</tr>  
</@foreach>
```

Tiap data product dapat langsung mengakses semua data variant-nya dengan fungsi **variants()** dan tiap variant dapat mengakses product yang berelasi dengannya dengan menggunakan atribut **product**. Uji dengan memasukkan data **variants** di database dan membuat form untuk menambahkan data **variants** ke database.