

# MODUL CRUD PRODUCTS

## ➤ MEMBUAT TABEL DENGAN MIGRATION

Untuk membuat tabel products dengan migration langkah pertama ialah membuat file migration itu sendiri. Pertama kita akan membuat file migration untuk tabel produk dengan menjalankan script berikut pada cmd:

**php artisan make:migration create\_tbl\_products\_table**

kemudian modifikasi file migration dengan nama **create\_tbl\_products\_table.php** yang ada pada folder database/migrations menjadi seperti berikut:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateTblProductsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('tbl_products', function (Blueprint $table) {
            $table->increments('product_id');
            $table->string('product_name', 100);
            $table->text('product_description');
            $table->string('price', 10);
            $table->integer('stock');
            $table->unsignedInteger('category_id');
            $table->timestamps();

            $table->foreign('category_id')->references('category_id')->on('tbl_categories');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('tbl_products');
    }
}
```

Kemudian sekarang jalankan perintah **php artisan migrate** pada cmd atau comand prompt yang sudah terarah pada projek laravel app\_pos\_namapanggilan yang telah dibuat, maka rancangan database yang dibuat dengan migration tadi akan dibawa ke database mysql.

## ➤ MEMBUAT MODEL PRODUCTS

Pertama jalankanlah perintah artisan berikut pada cmd yang sudah terarah ke projek laravel app\_pos\_namapanggilan:

### **php artisan make:model Products**

perintah artisan diatas akan menghasilkan file model products.php yang terletak pada folder app/ bukalah file tersebut dan modifikasi menjadi seperti dibawah:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use App\Categories;

class Products extends Model
{
    protected $primaryKey = 'product_id'; // Assuming 'ProductID' is the primary key

    protected $fillable = [
        'product_name',
        'category_id', // Foreign key for the ProductCategory relationship
        'product_description',
        'price',
        'stock',
    ];

    // Define the relationship to Category
    public function Categories()
    {
        return $this->belongsTo(Categories::class, 'category_id', 'category_id');
    }
}
```

Pada script productst.php diatas ada suatu fungsi bernama Categories, fungsi ini berfungsi untuk merelasikan tabel product dengan tbl\_categories. Langkah selanjutnya adalah modifikasi model C Categories menjadi seperti berikut:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use App\Products;
class Categories extends Model
{
    protected $table = 'tbl_categories';
    protected $primaryKey='category_id';

    protected $fillable=[
        'category_name',
        'description'
    ];

    // Example of a relationship
    public function Products()
    {
        return $this->hasMany(Products::class, 'category_id', 'category_id'); // Assuming you have a
        Product model
    }
}
```

## ➤ MEMBUAT CONTROLLER DAN ROUTES

Langkah selanjutnya adalah membuat sebuah controller untuk melakukan proses CRUD untuk tbl\_products. Untuk membuat controller tersebut jalankanlah perintah artisan dibawah pada comand prompt:

**php artisan make:controller ProductsController -r**

perintah artisan diatas akan menghasilkan controller **ProductsController.php** pada folder **app\Http\Controller** lengkap dengan 7 fungsi yang umum digunakan pada proses CRUD. Bukalah file controller tersebut dan tambahkan script berikut pada bagian atas controler.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\CATEGORIES;
use App\Products;
```

Script **use App\Proudtcs;** dan Script **use App\Categories;** tersebut berfungsi untuk mengkoneksikan ProductsContoller.php yang baru saja dibuat dengan model Products dan model Categories untuk tbl\_categories dan tbl\_products yang ada pada database.

Setelah membuat controller selanjutnya adalah mendefinisikan route untuk mengakses tiap-tiap fungsi yang ada pada controller ProductsController. Untuk mendefinisikannya bukanlah file **web.php** yang ada pada folder **routes** dan tambahkan script seperti dibawah untuk mendefinisikan route untuk mengakses fungsi-fungsi yang ada pada controller ProductsController:

```
//membuat product route
Route::get('/product', [ProductsController::class, 'index'])->name('product.index'); //untuk
menampilkan index
Route::get('/product/create', [ProductsController::class, 'create'])->name('product.create');
//untuk menampilkan form product
Route::post('/product', [ProductsController::class, 'store'])->name('product.store'); // untuk
melakukan proses simpan
Route::get('/product/{id}/edit', [ProductsController::class, 'edit'])->name('product.edit');
//untuk menampilkan form edit sesuai id
Route::put('product/{id}', [ProductsController::class, 'update'])->name('product.update');
//untuk melakukan proses update sesuai id
Route::delete('product/{id}', [ProductsController::class, 'destroy'])->name('product.destroy');
//untuk menghapus proses delete sesuai id
```

## ➤ MENAMPILKAN DATA

Untuk menampilkan data yang ada pada tabel bukanlah file ProductsController.php dan modifikasi fungsi **index()** menjadi seperti berikut:

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    $dataProducts = Products::all();
    return view('products.index', compact('dataProducts'));
}
```

Kemudian buatlah folder baru di dalam folder **resources/views** dengan nama **products** dan buatlah file dengan nama **index.blade.php** yang berisikan script sebagai berikut:

```

<table border="1">
  <thead>
    <tr>
      <th>#</th>
      <th>Product Name</th>
      <th>Category</th>
      <th>Description</th>
      <th>Price</th>
      <th>Stock</th>
      <th>
        <a href="{{route('product.create')}}">Create Products</a>
      </th>
    </tr>
  </thead>
  <tbody>
    @foreach ($dataProducts as $v)
      <tr>
        <td>{{ $loop->iteration }}</td>
        <td>{{ $v->product_name }}</td>
        <td>{{ $v->categories->category_name }}</td>
        <td>{{ $v->product_description }}</td>
        <td>{{ $v->price }}</td>
        <td>{{ $v->stock }}</td>
        <td>
          <a href="{{ route('product.edit', $v->product_id) }}">Edit</a>
          <form action="{{ route('product.destroy', $v->product_id) }}" method="POST">
            <input type="hidden" name="_method" value="delete" />
            {{ csrf_field() }}
            <button>Delete</button>
          </form>
        </td>
      </tr>
    </tbody>
  </table>

```

## ➤ MENAMBAH DATA

Untuk melakukan penambahan data produk langkah pertama yang harus kita lakukan adalah menampilkan form untuk menambah data, maka dari itu bukalalah controller ProductsController dan modifikasi fungsi **create()** yang ada di dalamnya menjadi seperti berikut:

```

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    $dataCategory = Categories::all();
    return view('products.create', compact('dataCategory'));
}

```

Kemudian buatlah satu file baru pada folder **resources/views/products/** dengan nama **create.blade.php** dan isikan script sebagai berikut:

```
<h1>FORM PRODUCTS</h1>
<form action="{{ route('product.store') }}" method="post">
    {{ csrf_field() }}
    <label>Product Name</label>
    <input type="text" class="form-control" name="product_name"></br>
    <label class="control-label col-sm-2">Product Category</label>
    <select name="category_id">
        <option value="">Select Category</option>
        @foreach ($dataCategory as $v)
            <option value="{{ $v->category_id }}">{{ $v->category_name }}</option>
        @endforeach
    </select></br>
    <label>Price</label>
    <input type="text" class="form-control" name="price"></br>
    <label>Stock</label>
    <input type="text" class="form-control" name="stock"></br>
    <label>Product Description</label>
    <textarea type="text" class="form-control" name="product_description"></textarea></br>
    <button type="submit">Save</button>
</form>
```

Langkah selanjutnya adalah memberikan script untuk menyimpan data produk ke dalam database. Pada atribut action yang ada pada form tambah produk kita menuliskan **{{route('product.store')}}** route tersebut akan membara data yang diinputkan di form ke dalam **ProductController.php** ke dalam fungsi store, jadi bukalah controller ProductController kemudian modifikasi fungsi store menjadi seperti berikut:

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'product_name' => 'required',
        'category_id' => 'required',
        'price' => 'required',
        'stock' => 'required',
        'product_description' => 'required',
    ]);

    Products::create([
        'product_name' => $request->product_name,
        'category_id' => $request->category_id,
        'price' => $request->price,
        'stock' => $request->stock,
        'product_description' => $request->product_description
    ]);

    return redirect(route('product.index'));
}

```

Pada fungsi store di bagian parameter kita memanggil fungsi bernama **Request**, fungsi tersebut digunakan untuk menangkap data inputan dari form lalu akan disimpan ke dalam **\$request**. Untuk script menambah data kita menggunakan **Eloquent ORM**, pada bagian akhir fungsi kita melakukan return nilai yakni **redirect(route('product.index'))**; script tersebut adalah fasilitas yang disediakan Laravel untuk mengarahkan kita secara otomatis ke dalam route tertentu dan pada kasus diatas kita akan diarahkan ke kembali ke halaman index dari produk dan. Sekarang form tambah produk sudah siap digunakan, silahkan lakukan uji coba pada form.

## ➤ MEMPERBAHARUI DATA

Untuk dapat memperbaharui data pada Laravel sama dengan menambah data kita harus menampilkan form edit data terlebih dahulu, namun perbedaannya kita harus menampilkan nilai lama dari data yang ingin diedit, maka dari itu bukalah file controller ProductController dan modifikasi pada bagian fungsi **edit()** menjadi seperti berikut:

```

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $dataEditproduct = Products::find($id);
    $categories = Categories::all();
    return view('products.edit', compact('dataEditproduct', 'categories'));
}

```

Pada script fungsi edit() diatas kita mengambil data produk dari tabel produk menggunakan model Eloquent dengan kondisi id dari produk sama dengan parameter yang diberikan pada link edit pada tombol edit yang ada di halaman utama produk dan mengambil data category dari table category dan kemudian kita melakukan return view untuk menampilkan view **edit.blade.php** yang ada di dalam folder **resources/views/products/**. Karena file tersebut belum ada maka buatlah satu file baru dengan nama **edit.blade.php** pada folder **resources/views/ products /** dan isikan script sebagai berikut:

```
<h1>FORM PRODUCTS</h1>
<form action="{{ route('product.update',$dataEditproduct) }}" method="post">
    {{ csrf_field() }}
    <input type="hidden" name="_method" value="PUT" />
    <label>Product Name</label>
    <input type="text" name="product_name" value="{{ $dataEditproduct->product_name }}"></br>
    <label class="control-label col-sm-2">Product Category</label>
    <select name="category_id">
        <option value="">Select Category</option>
        @foreach ($categories as $v)
            <option value="{{ $v->category_id }}" {{ $v->category_id ==
                $dataEditproduct->category_id ? 'selected' : '' }}>{{ $v->category_name }}</option>
        @endforeach
    </select></br>
    <label>Price</label>
    <input type="text" name="price" value="{{ $dataEditproduct->price }}"></br>
    <label>Stock</label>
    <input type="text" name="stock" value="{{ $dataEditproduct->stock }}"></br>
    <label>Product Description</label>
    <textarea type="text" name="product_description">{{ $dataEditproduct->product_description }}
</textarea></br>
    <button type="submit">Save</button>
</form>
```

Jika diperhatikan di dalam tag form terdapat script `<input type="hidden" name="_method" value="PUT">` field ini digunakan untuk menandakan bahwa data pada form akan dibawa ke dalam route `product.update`. Selanjutnya bukalah file controller **ProductController.php** dan modifikasi pada bagian fungsi **update()**; menjadi seperti berikut:



```

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $this->validate($request, [
        'product_name' => 'required',
        'category_id' => 'required',
        'price' => 'required',
        'stock' => 'required',
        'product_description' => 'required',
    ]);

    $dataUpdateproduct = Products::find($id);
    $dataUpdateproduct->update([
        'product_name' => $request->product_name,
        'category_id' => $request->category_id,
        'price' => $request->price,
        'stock' => $request->stock,
        'product_description' => $request->product_description
    ]);

    return redirect(route('product.index'));
}

```

Pada fungsi update kita menerima parameter berupa data dari form edit yang kita tangkap menggunakan fungsi *Request* yang kemudian juga kita simpan ke dalam variabel request serta kita juga mengirim data id dari produk yang kita edit yang juga kita tangkap pada bagian parameter fungsi edit yang kita beri nama \$id. Setelah itu kita menyimpan nilai baru dari data produk menggunakan eloquent ORM

## ➤ MENGHAPUS DATA

Pada bagian menampilkan data produk kita telah menyiapkan form untuk melakukan proses penghapusan data di bagian tag td:

```

<form action="{ route('product.destroy', $v->product_id) }" method="POST">
    <input type="hidden" name="_method" value="delete" />
    {{ csrf_field() }}
    <button>Delete</button>
</form>

```

Form tersebut akan mengirimkan data ke route produk.destroy dengan parameter \$id dari data

produk. Karena ini merupakan penghapusan data dan kita menggunakan route resource maka kita perlu mendefinisikan `_method` yang akan kita gunakan yang kita simpan dalam tag input type hidden, selanjutnya bukalah controller `productcontroller` dan modifikasi fungsi `destroy()` menjadi seperti berikut:

```
/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    Products::where('product_id', $id)->delete();
    return redirect(route('product.index'));
}
```