Laporan Modul 7: Eloquent Relationship & Pagination

Mata Kuliah: Workshop Web Lanjut
Nama: Adha Gusti Harmadhan
NIM: 2024573010009
Kelas: 2B TI

Abstrak

Laporan ini menjelaskan hasil praktikum pada Modul 7: *Eloquent Relationship & Pagination* dalam mata kuliah Workshop Web Lanjut. Fokus praktikum adalah memahami bagaimana Laravel mendefinisikan relasi antar tabel menggunakan Eloquent ORM (One-to-One, One-to-Many, dan Many-to-Many), serta bagaimana membagi data menjadi beberapa halaman menggunakan fitur pagination bawaan Eloquent. Praktikum dibagi menjadi dua: (1) membangun aplikasi *complex relationships* yang menghubungkan model User, Profile, Post, dan Tag dengan berbagai jenis relasi; (2) membangun aplikasi sederhana daftar produk yang menampilkan data Product dengan pagination menggunakan Tailwind. Melalui praktikum ini mahasiswa diharapkan mampu memetakan konsep relasi pada ERD ke dalam Eloquent Relationship, serta memahami pentingnya pagination untuk performa aplikasi dan pengalaman pengguna.

1. Dasar Teori

1.1 Eloquent Relationship

Dalam aplikasi berbasis database, tabel-tabel biasanya saling berhubungan. Di Laravel, hubungan antar tabel tersebut direpresentasikan melalui **Eloquent Relationship** di level model. Dengan relationship, kita bisa mengakses data terkait cukup dengan memanggil properti pada objek model tanpa menulis query SQL manual.

Beberapa jenis relasi yang umum:

- **One-to-One** → satu data di tabel A hanya berpasangan dengan satu data di tabel B.
- **One-to-Many** → satu data di tabel A berpasangan dengan banyak data di tabel B.
- **Many-to-Many** → banyak data di tabel A bisa berpasangan dengan banyak data di tabel B melalui tabel pivot.

Relationship memudahkan pengembangan karena:

- Sintaks lebih deklaratif dan mudah dibaca.
- Mendukung *eager loading* (`with()`) untuk menghemat query.
- Mengurangi penggunaan query mentah atau join yang rumit.

1.2 Relasi One-to-One

Relasi **One-to-One** digunakan ketika satu baris di sebuah tabel hanya terkait dengan satu baris di tabel lain. Contoh klasik: `users` dan `profiles` — satu user hanya punya satu profile, dan sebaliknya.

Skema:

- `users` → menyimpan data user
- `profiles` → menyimpan data profil, memiliki kolom `user_id` unik

Contoh di model:

```
// User.php
public function profile()
{
    return $this->hasOne(Profile::class);
}

// Profile.php
public function user()
{
    return $this->belongsTo(User::class);
}
```

Dengan ini, cukup memanggil `$user->profile` untuk mendapatkan profil user tersebut.

1.3 Relasi One-to-Many

Relasi **One-to-Many** dipakai ketika sebuah data di tabel A bisa memiliki banyak data di tabel B. Contoh: `users` dan `posts` — satu user bisa menulis banyak postingan, sedangkan satu post hanya dimiliki satu user.

Skema:

- `users`
- `posts` (punya `user_id` sebagai foreign key)

Contoh di model:

```
// User.php
public function posts()
{
    return $this->hasMany(Post::class);
}

// Post.php
public function user()
{
    return $this->belongsTo(User::class);
}
```

Di view kita bisa looping `@foreach($user->posts as $post)` untuk menampilkan semua post milik user.

1.4 Relasi Many-to-Many dan Tabel Pivot

Relasi **Many-to-Many** terjadi saat banyak data di tabel A bisa berhubungan dengan banyak data di tabel B. Contoh: `posts` dan `tags` — satu post bisa punya banyak tag, dan satu tag bisa digunakan di banyak post.

Untuk ini dibutuhkan **tabel pivot**, misalnya `post_tag` yang menyimpan pasangan `post_id` dan `tag_id`. Konvensi Laravel: nama tabel pivot menggunakan **bentuk tunggal** dan urut alfabetis, misalnya `post_tag` (bukan `posts_tags`).

Contoh di model:

```
// Post.php
public function tags()
{
    return $this->belongsToMany(Tag::class);
}

// Tag.php
public function posts()
{
    return $this->belongsToMany(Post::class);
}
```

Dengan ini, `$post->tags` akan mengembalikan semua tag milik sebuah post.

1.5 Eager Loading dengan `with()`

Jika kita memanggil relasi di dalam loop (misalnya `$user->profile` berulang-ulang), bisa terjadi masalah **N+1 query**. Untuk mencegah ini, Laravel menyediakan *eager loading* dengan method `with()`.

Contoh:

```
$users = User::with('profile', 'posts')->get();
```

Perintah ini mengambil data user sekaligus data profile dan posts-nya dalam beberapa query yang lebih efisien.

1.6 Pagination di Laravel

Pagination adalah proses membagi data menjadi beberapa halaman, misalnya 10 item per halaman. Di Laravel, pagination bisa dilakukan dengan method `paginate()` pada query Eloquent.

Contoh:

```
$products = Product::orderBy('id', 'desc')->paginate(10);
```

Di view, kita cukup memanggil:

```
{{ $products->links() }}
```

Laravel akan otomatis menghasilkan link pagination dengan styling Tailwind CSS. Pagination membantu:

- Mengurangi beban server dan waktu loading karena data tidak diambil semua.
- Membuat tampilan lebih rapi dan mudah dinavigasi.
- Meningkatkan UX terutama pada data yang jumlahnya besar.

---

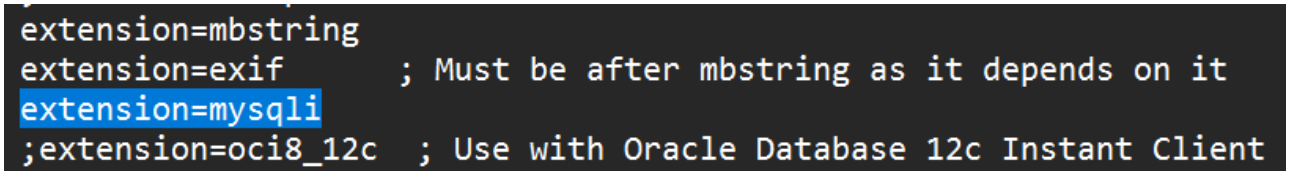
2. Langkah-Langkah Praktikum

2.1 Praktikum 1 — Eloquent ORM Relationships: One-to-One, One-to-Many, Many-to-Many

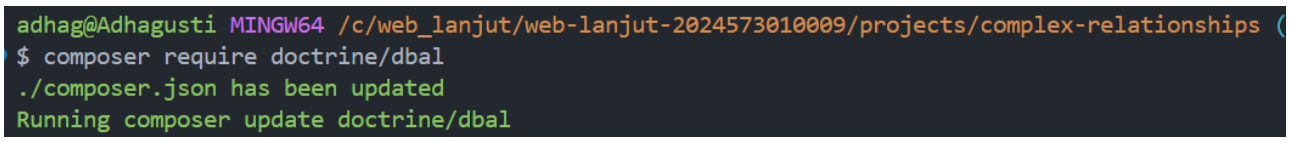
Langkah-langkah:

1. Buat dan buka proyek Laravel baru:

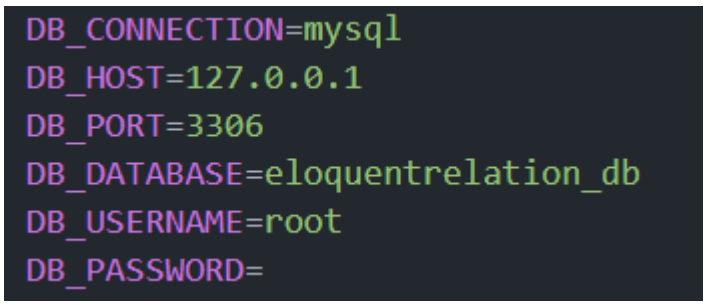
2. Pastikan ekstensi MySQL aktif (`mysqli`, `pdo_mysql`) dan buat database baru `eloquentrelation_db` di MySQL atau phpMyAdmin.



3. Install dependency database dan konfigurasi `.env`:



Atur koneksi database di `.env`:



4. Buat migration untuk `profiles`, `posts`, `tags`, dan `post_tag`:

```
php artisan make:migration create_profiles_table
php artisan make:migration create_posts_table
```

```
php artisan make:migration create_tags_table
php artisan make:migration create_post_tag_table
```

Lalu perbarui masing-masing file migration untuk menambahkan foreign key dan struktur tabel.

profiles_table

projects > complex-relationships > database > migrations > 2025_11_05_012048_create_profiles_table.php

```php
<?php

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

return new class extends Migration {
    public function up()
    {
        Schema::create('profiles', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id')->unique();
            $table->text('bio')->nullable();
            $table->string('website')->nullable();
            $table->timestamps();

            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
        });
    }

    public function down()
    {
        Schema::dropIfExists('profiles');
    }
};
```

posts_table

projects > complex-relationships > database > migrations > 2025_11_05_012049_create_posts_table.php

```php
<?php

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

return new class extends Migration {
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id');
            $table->string('title');
            $table->text('content');
            $table->timestamps();

            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
        });
    }

    public function down()
    {
        Schema::dropIfExists('posts');
    }
};
```

tags_table

projects > complex-relationships > database > migrations > 2025_11_05_012050_create_tags_table.php

```php
<?php

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

return new class extends Migration {
    public function up()
    {
        Schema::create('tags', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('tags');
    }
};
```

post_tag_table

projects > complex-relationships > database > migrations > 2025_11_05_012051_create_post_tag_table.php

```php
<?php

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

return new class extends Migration {
    public function up()
    {
        Schema::create('post_tag', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('post_id');
            $table->unsignedBigInteger('tag_id');
            $table->timestamps();

            $table->foreign('post_id')->references('id')->on('posts')->onDelete('cascade');
            $table->foreign('tag_id')->references('id')->on('tags')->onDelete('cascade');
        });
    }

    public function down()
    {
        Schema::dropIfExists('post_tag');
    }
};
```

5. Jalankan migrasi untuk membuat tabel di database:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/complex-relationships (main)
$ php artisan make:migration create_profiles_table
php artisan make:migration create_posts_table
php artisan make:migration create_tags_table
php artisan make:migration create_post_tag_table

  INFO  Migration [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\database\migr
ations\2025_11_05_012048_create_profiles_table.php] created successfully.


  INFO  Migration [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\database\migr
ations\2025_11_05_012049_create_posts_table.php] created successfully.


  INFO  Migration [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\database\migr
ations\2025_11_05_012050_create_tags_table.php] created successfully.


  INFO  Migration [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\database\migr
ations\2025_11_05_012051_create_post_tag_table.php] created successfully.
```

6. Buat model `Profile`, `Post`, dan `Tag`:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/complex-relationships (main)
$ php artisan make:model Profile
php artisan make:model Post
php artisan make:model Tag

  INFO  Model [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\app\Models\Profile.ph
p] created successfully.


  INFO  Model [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\app\Models\Post.php]
created successfully.


  INFO  Model [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\app\Models\Tag.php] c
reated successfully.
```

`profiles.php`

projects > complex-relationships > app > Models > Profile.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use App\Models\User;

class Profile extends Model
{
    use HasFactory;

    /** Kolom yang dapat diisi secara massal*/
    protected $fillable = ['user_id', 'bio', 'website'];

    /*** Relasi inverse One-to-One dengan User*/
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

posts.php

projects > complex-relationships > app > Models > Post.php

```php
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use App\Models\User;
use App\Models\Tag;

class Post extends Model
{
    use HasFactory;
    /** Kolom yang dapat diisi secara massal */
    protected $fillable = ['user_id', 'title', 'content'];
    /** Relasi inverse One-to-Many dengan User*/
    public function user()
    {
        return $this->belongsTo(User::class);
    }
    /**Relasi Many-to-Many dengan Tag*/
    public function tags()
    {
        return $this->belongsToMany(Tag::class);
    }
}
```

tag.php

projects > complex-relationships > app > Models > Tag.php

```php
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use App\Models\Post;

class Tag extends Model
{
    use HasFactory;

    /**
     * Kolom yang dapat diisi secara massal
     */
    protected $fillable = ['name'];

    /**
     * Relasi Many-to-Many dengan Post
     */
    public function posts()
    {
        return $this->belongsToMany(Post::class);
    }
}
```

7. Buat dan atur seeder di `DatabaseSeeder` untuk mengisi data awal:

DatabaseSeeder.php U

projects > complex-relationships > database > seeders > DatabaseSeeder.php

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Profile;
use App\Models\Post;
use App\Models\Tag;

class DatabaseSeeder extends Seeder
{
    use WithoutModelEvents;
    /**     * Seed the application's database. */
    public function run(): void
    {
        // Membuat 10 user menggunakan factory
    User::factory(10)->create();
    // Membuat profile untuk setiap user
    foreach (User::all() as $user) {
        $user->profile()->create([
            'bio' => 'Ini adalah bio untuk user ' . $user->id,
            'website' => 'https://ilmudata.id/user/' . $user->id,
        ]);
    }
    // Membuat post untuk setiap user
    foreach (User::all() as $user) {
        $user->posts()->create([
            'title' => 'Judul Post untuk user ' . $user->id,
            'content' => 'Ini adalah konten dari post untuk user ' . $user->id,
        ]);
    }
    // Membuat tag dan mengasosiasikannya dengan posts
    foreach (Post::all() as $post) {
        $tag = Tag::create(['name' => 'Tag untuk post ' . $post->id]);
        $post->tags()->attach($tag->id);
    }
    }
}
```

Jalankan seeder:

```
php artisan db:seed
```

8. Buat controller `UserController` dan `PostController`:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/complex-relationships (main)
$ php artisan make:controller UserController
php artisan make:controller PostController
  INFO  Controller [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\app\Http\Controllers\UserController.php] created successfully.


  INFO  Controller [C:\web_lanjut\web-lanjut-2024573010009\projects\complex-relationships\app\Http\Controllers\PostController.php] created successfully.
```

`

Tambahkan method `index` dan `show` untuk masing-masing:

- `UserController` memanggil `User::with('profile','posts')->get()`.

projects > complex-relationships > app > Http > Controllers > UserController.php

```php
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\User;

class UserController extends Controller
{
    /** Menampilkan daftar semua user dengan profile dan posts terkait   */
    public function index()
    {
        $users = User::with('profile', 'posts')->get();
        return view('users.index', compact('users'));
    }

    /** Menampilkan detail user tertentu  */
    public function show(User $user)
    {
        // Menggunakan route model binding, Laravel akan otomatis
        // mencari user berdasarkan ID yang diberikan
        return view('users.show', compact('user'));
    }
}

```

- `PostController` memanggil `Post::with('user','tags')->get()`.

Post.php U | UserController.php U | PostController.php U | Tag.php U

projects > complex-relationships > app > Http > Controllers > PostController.php

```php
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Post;

class PostController extends Controller
{
    /**     * Menampilkan daftar semua posts dengan user dan tags terkai*/
    public function index()
    {
        $posts = Post::with('user', 'tags')->get();
        return view('posts.index', compact('posts'));
    }

    /**     * Menampilkan detail post tertentu*/
    public function show(Post $post)
    {
        // Menggunakan route model binding, Laravel akan otomatis
        // mencari post berdasarkan ID yang diberikan
        return view('posts.show', compact('post'));
    }
}
```

9. Daftarkan route di `routes/web.php`:

projects > complex-relationships > routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\UserController;
use App\Http\Controllers\PostController;

// Routes untuk User
Route::get('/users', [UserController::class, 'index'])->name('users.index');
Route::get('/users/{user}', [UserController::class, 'show'])->name('users.show');

// Routes untuk Post
Route::get('/posts', [PostController::class, 'index'])->name('posts.index');
Route::get('/posts/{post}', [PostController::class, 'show'])->name('posts.show');

Route::get('/', function () {
    return view('welcome');
});

```

10. Buat layout `resources/views/layouts/app.blade.php` yang memuat Bootstrap dan menu navigasi Users & Posts.

projects > complex-relationships > resources > views > layouts > app.blade.php

```blade
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield('title')</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container mt-4">

    <h1 class="text-center mb-4">Laravel 12 Complex Relationships</h1>

    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @endif

    <nav class="mb-4">
        <a href="{{ route('users.index') }}" class="btn btn-primary">Users</a>
        <a href="{{ route('posts.index') }}" class="btn btn-secondary">Posts</a>
    </nav>

    @yield('content')

</body>
</html>
```

11. Buat view untuk users:

    - `resources/views/users/index.blade.php` untuk daftar user.

projects > complex-relationships > resources > views > users > index.blade.php

```blade
@extends('layouts.app')

@section('title', 'Users')

@section('content')
    <h2>Users List</h2>
    <ul class="list-group">
        @foreach($users as $user)
            <li class="list-group-item">
                <a href="{{ route('users.show', $user->id) }}">{{ $user->name }}</a> ({{ $user->email }})
            </li>
        @endforeach
    </ul>
@endsection
```

    - `resources/views/users/show.blade.php` untuk detail user, profile, dan daftar post miliknya.

projects > complex-relationships > resources > views > users > show.blade.php

```blade
@extends('layouts.app')

@section('title', 'User Profile')

@section('content')
    <h2>{{ $user->name }}'s Profile</h2>
    <p>Email: {{ $user->email }}</p>

    <h3>Profile Details</h3>
    <p>Bio: {{ $user->profile->bio ?? 'No bio available' }}</p>
    <p>Website: <a href="{{ $user->profile->website ?? '#' }}">{{ $user->profile->website ?? 'No website' }}</a></p>

    <h3>Posts</h3>
    <ul class="list-group">
        @foreach($user->posts as $post)
            <li class="list-group-item">
                <a href="{{ route('posts.show', $post->id) }}">{{ $post->title }}</a>
            </li>
        @endforeach
    </ul>
@endsection
```

12. Buat view untuk posts:

- `resources/views/posts/index.blade.php` untuk daftar post dan nama penulisnya.

projects > complex-relationships > resources > views > posts > index.blade.php

```blade
@extends('layouts.app')

@section('title', 'Posts')

@section('content')
    <h2>All Posts</h2>

    <ul class="list-group">
        @foreach($posts as $post)
            <li class="list-group-item">
                <a href="{{ route('posts.show', $post->id) }}">{{ $post->title }}</a> by {{ $post->user->name }}
            </li>
        @endforeach
    </ul>
@endsection
```

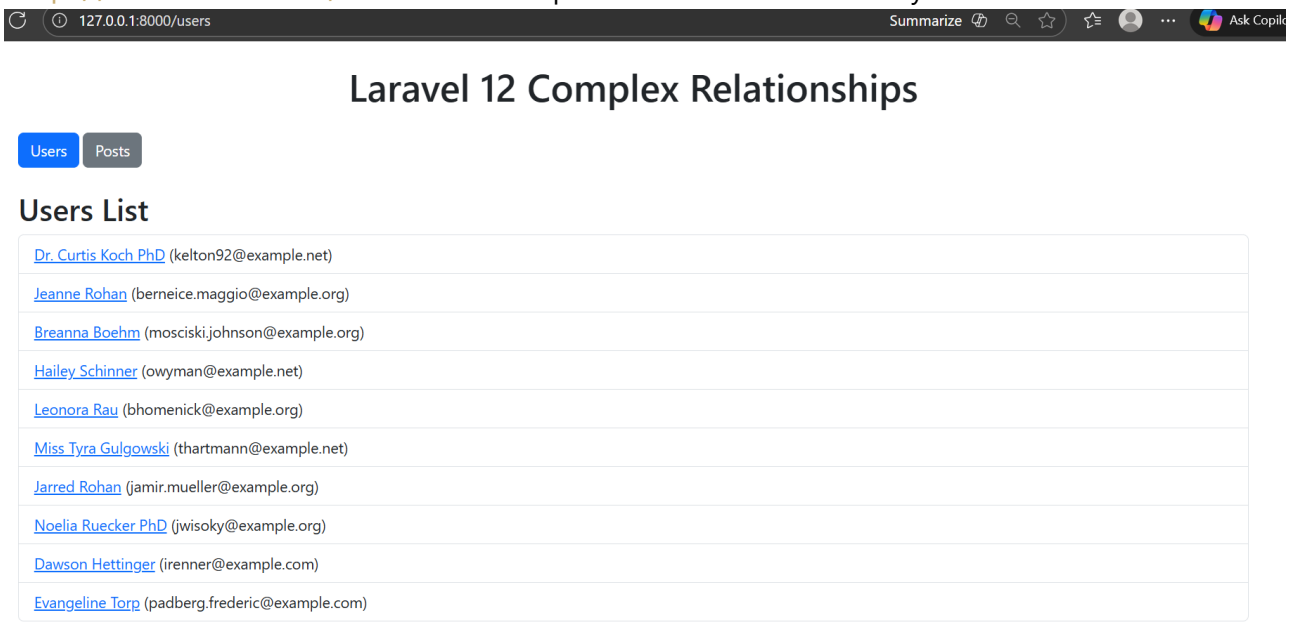
- `resources/views/posts/show.blade.php` untuk detail post, author, dan daftar tag.

projects > complex-relationships > resources > views > posts > show.blade.php

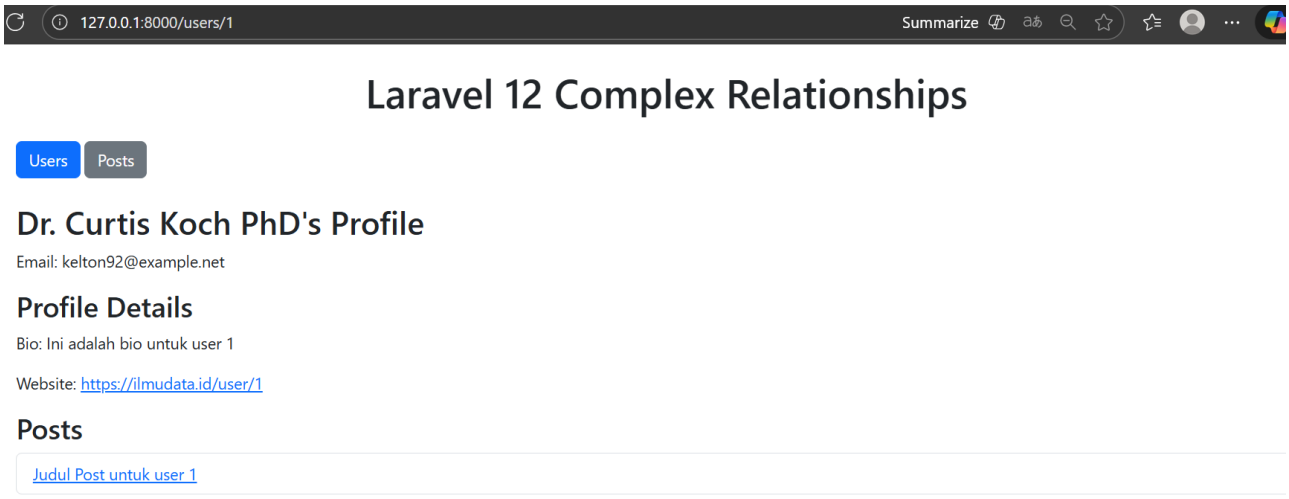
```blade
@extends('layouts.app')

@section('title', 'Post Details')

@section('content')
    <h2>{{ $post->title }}</h2>
    <p><strong>Author:</strong> {{ $post->user->name }}</p>
    <p>{{ $post->content }}</p>

    <h3>Tags</h3>
    <ul class="list-group">
        @foreach($post->tags as $tag)
            <li class="list-group-item">{{ $tag->name }}</li>
        @endforeach
    </ul>

    <a href="{{ route('posts.index') }}" class="btn btn-secondary mt-3">Back to Posts</a>
@endsection
```

Hasil Pengujian:

- `http://127.0.0.1:8000/users` → menampilkan daftar user dan emailnya.

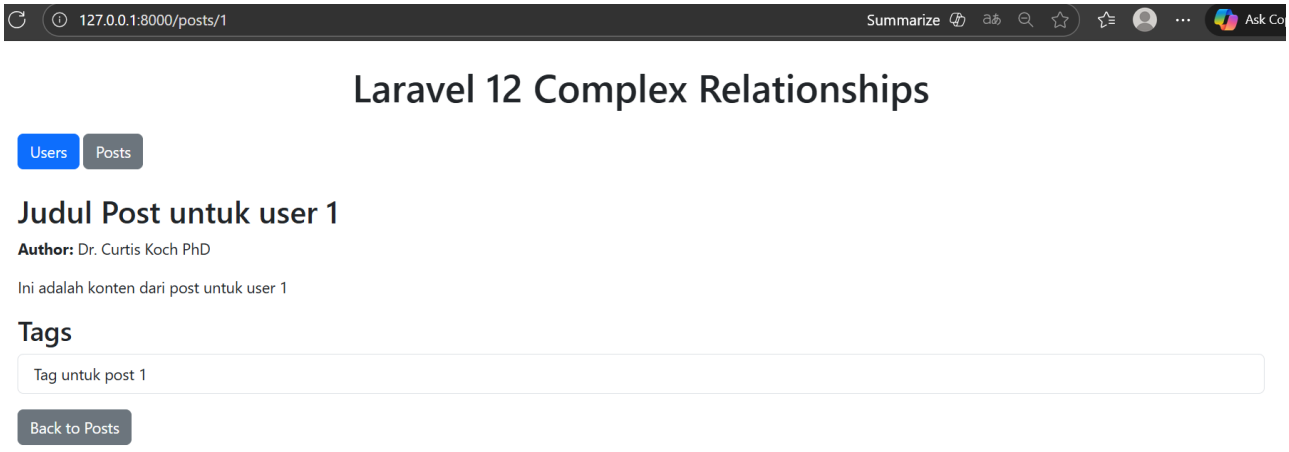


- Klik salah satu user → `http://127.0.0.1:8000/users/{user}` menampilkan informasi user, bio & website dari `profile`, serta daftar post milik user tersebut.



- `http://127.0.0.1:8000/posts` → menampilkan daftar post dan nama user yang menjadi author.

- Klik salah satu post → `http://127.0.0.1:8000/posts/{post}` menampilkan detail post, author, dan daftar tag Many-to-Many.



2.2 Praktikum 2 — Pagination dengan Eloquent ORM

Pada praktik kedua, fokusnya adalah cara melakukan pagination daftar produk menggunakan model `Product` dan method `paginate()` dari Eloquent, dengan tampilan yang memanfaatkan Tailwind CSS.

Tujuan praktik (implisit):
Latihan menampilkan data dalam jumlah banyak dengan pagination Laravel, agar tampilan lebih rapi, loading lebih cepat, dan UX tetap nyaman.

Langkah-langkah:

1. Buat proyek baru:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects (main)
$ laravel new productpagination
```

2. Buat database `pagination_db` di MySQL atau phpMyAdmin dan konfigurasi koneksi di `.env`, mirip praktikum pertama:

```
DB_DATABASE=pagination_db
```

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

Install dependency database dan clear config:

```
composer require doctrine/dbal
php artisan config:clear
```

3. Buat model dan migration untuk `Product`:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan make:model Product -m

  INFO  Model [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\app\Models\Product.php] c
reated successfully.

  INFO  Migration [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\database\migrations\2
025_11_05_021207_create_products_table.php] created successfully.
```

Perbarui migration `create_products_table` dengan kolom `name` dan `price`, kemudian jalankan migrasi:

```
php artisan migrate
```

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan migrate

  INFO  Preparing database.

  Creating migration table ..................................................... 36.86ms DONE

  INFO  Running migrations.

  0001_01_01_000000_create_users_table ........................................ 212.16ms DONE
  0001_01_01_000001_create_cache_table ......................................... 54.65ms DONE
  0001_01_01_000002_create_jobs_table ......................................... 194.44ms DONE
```

4. Buat seeder dan factory untuk menghasilkan data dummy:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan make:seeder ProductSeeder

  INFO  Seeder [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\database\seeders\ProductSeeder.php] created successfully.
```

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan make:factory ProductFactory --model=Product

  INFO  Factory [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\database\factories\ProductFactory.php] created successfully.
```

- Di `Product.php`, tambahkan `HasFactory` dan `$fillable`.

projects > productpagination > app > Models > Product.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Product extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'price'];
}
```

- Di `ProductFactory`, generate nama produk dan harga acak.

projects > productpagination > database > factories > ProductFactory.php

```php
<?php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class ProductFactory extends Factory
{
    public function definition(): array
    {
        return [
            'name' => fake()->word(),
            'price' => fake()->randomFloat(2, 10, 1000),
        ];
    }
}

```

- Di `ProductSeeder`, panggil `Product::factory()->count(50)->create();`

projects > productpagination > database > seeders > ProductSeeder.php

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Product;

class ProductSeeder extends Seeder
{
    public function run(): void
    {
        Product::factory()->count(50)->create();
    }
}
```

- Di `DatabaseSeeder`, panggil `ProductSeeder`.

projects > productpagination > database > seeders > DatabaseSeeder.php

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    use WithoutModelEvents;

    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        $this->call([
            ProductSeeder::class,
        ]);
    }

}
```

Jalankan seeder:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan db:seed

  INFO  Seeding database.

  Database\Seeders\ProductSeeder ..... RUNNING
  Database\Seeders\ProductSeeder ..... 363 ms DONE
```

5. Buat `ProductController`:

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan make:controller ProductController

  INFO  Controller [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\app\Http\Controllers
\ProductController.php] created successfully.
```

Tambahkan method `index` untuk mengambil data produk dengan pagination:

projects > productpagination > app > Http > Controllers > ProductController.php

```php
<?php

namespace App\Http\Controllers;

use App\Models\Product;

class ProductController extends Controller
{
    public function index()
    {
        $products = Product::orderBy('id', 'desc')->paginate(10);
        return view('products.index', compact('products'));
    }
}
```

6. Daftarkan route di `routes/web.php`:

projects > productpagination > routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;

Route::get('/products', [ProductController::class, 'index'])->name('products.index');

Route::get('/', function () {
    return view('welcome');
});
```

7. Buat view `resources/views/products/index.blade.php` untuk menampilkan daftar produk dengan Tailwind dan link pagination:

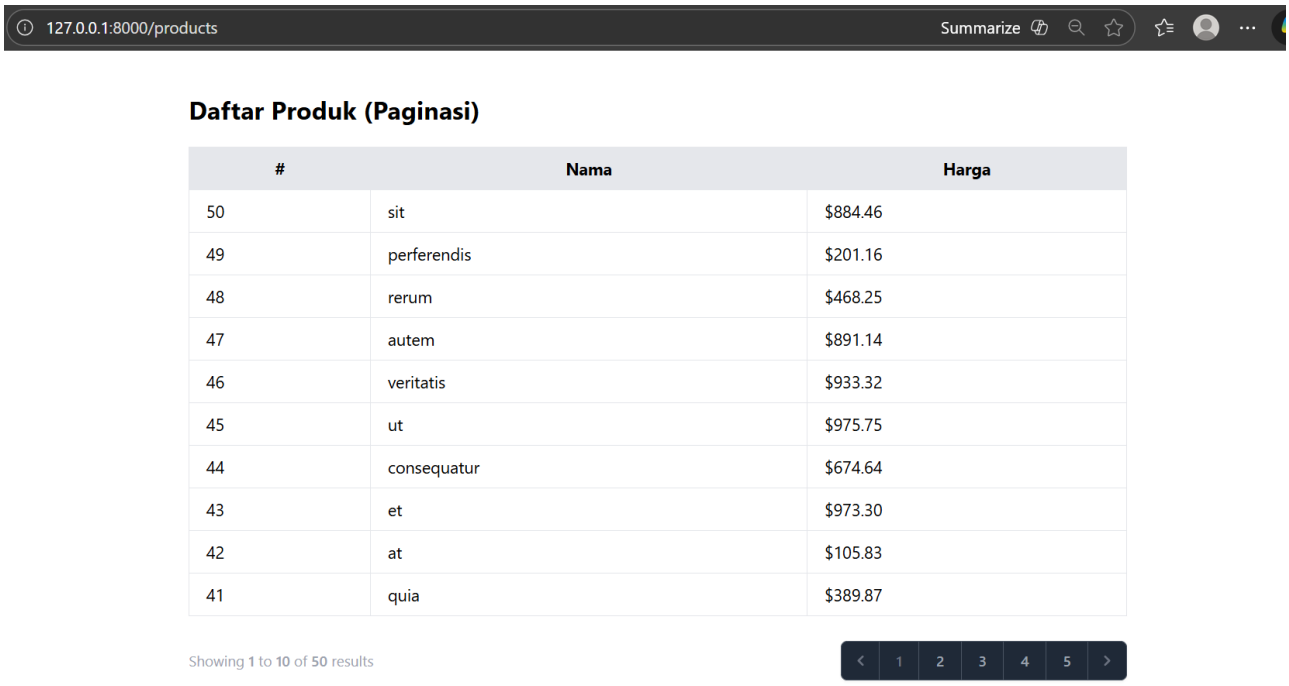
projects > productpagination > resources > views > products > index.blade.php

```blade
<!DOCTYPE html>
<html>
<head>
    <title>Paginated Products</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="max-w-4xl mx-auto py-10">
    <h1 class="text-2xl font-bold mb-5">Daftar Produk (Paginasi)</h1>

    <table class="table-auto w-full border-collapse border border-gray-300 mb-6">
        <thead>
            <tr class="bg-gray-200">
                <th class="border px-4 py-2">#</th>
                <th class="border px-4 py-2">Nama</th>
                <th class="border px-4 py-2">Harga</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($products as $product)
            <tr>
                <td class="border px-4 py-2">{{ $product->id }}</td>
                <td class="border px-4 py-2">{{ $product->name }}</td>
                <td class="border px-4 py-2">${{ number_format($product->price, 2) }}</td>
            </tr>
            @endforeach
        </tbody>
    </table>

    <div>
        {{ $products->links() }}
    </div>
</body>
</html>

```

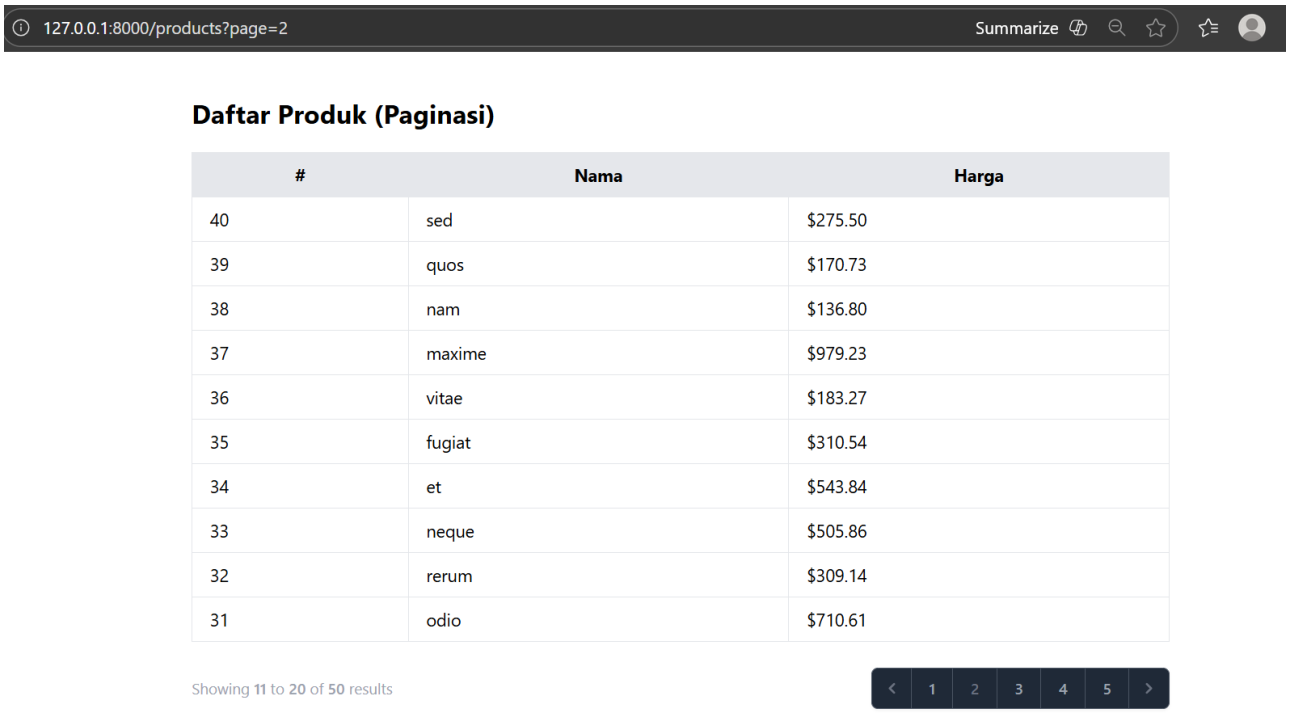
View ini menampilkan tabel produk dan memanggil `{{ $products->links() }}` untuk menampilkan navigasi pagination.

Hasil Pengujian:

- `http://127.0.0.1:8000/products` → menampilkan daftar produk dari database dalam bentuk tabel, 10 item per halaman.



- Link pagination (nomor halaman, Next, Previous) muncul otomatis di bawah tabel dan dapat diklik untuk berpindah halaman.



---

3. Hasil dan Pembahasan

1. **Relasi One-to-One (User–Profile)** pada praktikum pertama memperlihatkan bagaimana satu user dikaitkan dengan tepat satu profile melalui foreign key `user_id` yang unik di tabel `profiles`. Dengan `hasOne()` dan `belongsTo()`, pengambilan data menjadi sangat sederhana dan tidak membutuhkan join manual.

2. **Relasi One-to-Many (User–Posts)** menunjukkan bahwa satu user dapat memiliki banyak post. Di controller dan view, kita cukup memanfaatkan `$user->posts` untuk menampilkan seluruh tulisan user.

Ini sangat mirip dengan kasus nyata seperti sistem blog atau forum.

3. **Relasi Many-to-Many (Post–Tags)** memperkenalkan penggunaan tabel pivot `post_tag`. Dengan `belongsToMany()` di kedua model, proses menempelkan tag ke post dan mengambil ulang data tag menjadi jauh lebih mudah. Praktikum ini juga menegaskan pentingnya konvensi penamaan tabel pivot di Laravel.

4. Penggunaan ***eager loading*** (`with('profile', 'posts')` dan `with('user', 'tags')`) membuat akses data relasi menjadi lebih efisien, menghindari masalah N+1 query yang sering muncul jika relasi dipanggil berulang di dalam loop.

5. Pada **pratikum pagination**, mahasiswa melihat bagaimana data yang jumlahnya banyak sebaiknya tidak ditampilkan sekaligus. Dengan `Product::orderBy(...)->paginate(10)`, sistem otomatis membagi data per halaman dan menghasilkan link navigasi. Ini tidak hanya meningkatkan performa, tetapi juga membuat tampilan lebih user-friendly.

6. Secara keseluruhan, kedua praktikum ini memperlihatkan bagaimana **relasi data** dan **pagination** saling melengkapi pada aplikasi Laravel yang riil: relasi untuk memetakan struktur data yang kompleks, dan pagination untuk mengatur cara penyajian data tersebut kepada pengguna.

---

4. Kesimpulan

Dari praktikum Modul 7 ini dapat disimpulkan bahwa:

1. **Eloquent Relationship** menyediakan cara yang deklaratif dan rapi untuk mendefinisikan hubungan antar model seperti One-to-One, One-to-Many, dan Many-to-Many tanpa perlu query SQL yang rumit.
2. **Tabel pivot** adalah komponen penting dalam relasi Many-to-Many dan Laravel menyediakan dukungan langsung melalui method `belongsToMany()`.
3. Penggunaan ***eager loading*** dengan `with()` sangat membantu mengoptimasi query dan mencegah terjadinya N+1 query problem ketika bekerja dengan banyak relasi.
4. **Pagination** dengan `paginate()` mempermudah pembagian data ke beberapa halaman, meningkatkan performa dan pengalaman pengguna, serta otomatis terintegrasi dengan Tailwind CSS untuk tampilan link pagination.
5. Dengan menggabungkan relasi Eloquent dan pagination, mahasiswa dapat membangun aplikasi Laravel yang *scalable*, efisien, dan tetap mudah dipelihara baik dari sisi struktur kode maupun dari sisi tampilan data.

---

5. Referensi

- Modul 7 - Eloquent Relationship & Pagination — (https://hackmd.io/@mohdrzu/ryIIM1a0ll)
- Dokumentasi Resmi Laravel 12 — (https://laravel.com/docs/12.x/eloquent-relationships)
- Dokumentasi Pagination Laravel — (https://laravel.com/docs/12.x/pagination)