**# N+1 Problem**

N+1 Problem ini terjadi, ketika aplikasi kita mengambil data dari database dan dimana didalamnya kita melakukan looping terhadap data kita. yg terjadi adalah, kita akan melakukan pemanggilan ke dalam database atau querynya berulang-ulang. sehingga nanti aplikasinya itu yg harusnya hanya melakukan 2 query, query kesemua dan query ke data yg ada didalamnya, **malah akan menambahkan (N) query.**

**sebenarnya tidak ada problem saat pemanggilan querynya di controller post,**

**contoh : 'posts' => Post::latest()->get()**

**ini tidak ada masalah yg terjadi, masalahnya ada ketika kita melakukan looping pada posts.blade.php-nya**

ketika kita melakukan looping ke semua postingan kita, kita butuh table lain, nah ini problemnya, ada table author dan table category, jadi kita query lagi didalam looping, problemnya adalah setiap kita melakukan query untuk ngecek authornya siapa, itu dia melakukan pemanggilan terhadap database, looping author($post-author>name), dia connect ke database, query lagi ke database, mau manggil category($post->category->name) juga sama. padahal query yg dilakukan itu hanya 3, ambil semua postingan, ambil semua author, dan ambil semua category.

Jadi misal didalam table posts, ada user dan category, dan misal ada 20 posts, maka ada 40 query yg akan dilakukan (20x ke table user + 20x ke table category)

Itu adalah N+1 problem, karena aplikasinya masih kecil, belum ada banyak postingan didalamnya, masih gk ada masalah mengenai performanya, tapi ketika postingannya sudah banyak misal 2000, maka itu akan memperlambat aplikasinya

ada yg namanya **lazy loading** (males, jadi dia melakukan ketika dibutuhin)

ada juga yg namanya **Eager Loading** (Eager artinya semangat), jadi lakukan semuanya diawal, sehingga kita udah punya datanya, sehingga pada saat looping nanti gk perlu melakukan pemanggilan data ke database lagi.

**penjelasan eager loading:**

pada saat kita mengakses sebuah relationship didalam sebuah eloquent (setiap pake belongsTo, hasMany, dsb) maka modelnya akan melakukan sebuah teknik yg namanya lazy loading, artinya data relationshipnya ini gk di load/dipanggil, sampe nantinya kita mengakses propertynya pada saat kita looping. tapi kita bisa meminta si eloquent-nya agar melakukan "eager load" ketika kita melakukan query pada parent-nya, ketika kita melakukan query dipostingannya, nanti dia sekalian juga nge-query langsung author dan category-nya. Eager loading ini kita lakukan untuk menghindari N+1 problem.

jadi kita kasih depannya 'with' sebelum kita ambil semua datanya

contoh : $books = Book::with('author')->get();

ini maksudnya, kita akan tambahkan with, terus table apa yg mau kita tambahin, terus ambil latest, terus ambil get. Itu tablenya hanya satu, jika ingin dua, maka harus pakai array.

contoh : $books = Book::with(['author', 'publish'])->get();

**Ada juga yg namannya Lazy Eager Loading, jadi digabung lazy loading dengan eager loading**

Penjelasan :

kadang2 kita butuh melakukan eager loading pada relationship kita, tapi setelah si parentnya sudah didapatkan. jadi gk sekalian diambil karena kita lagi melakukan Route Model Binding, jadi kita gk pake 'with', tapi kita pakeknya 'load'

contoh:

if ($someCondition) {

$books->load('author', 'publisher');

}

Jadi, pada saat manggil si authornya, kita load juga model sisanya, pada saat kita ngambil model yg sudah melakukan Route Model Binding.

**# Query Scope**

dengan menggunakan local Scopes, itu memungkinkan kita untuk mendefinisikan query2 umum yg bisa kita gunakan kembali didalam aplikasi kita, contohnya kalau misalkan kita butuh untuk secara sering mengambil data user yg kita anggap populer.

untuk bisa menjalankan itu, kita butuh model yg berisikan method scopePopular($query) => kata didepan 'scope'nya bebas mau dinamain apa aja.

contoh :

public function scopePopular($query)

{ // disini kita tinggal tulis klausul tambahannya apa, misalkan didepan udh kita cari "SELECT \* FROM User", kalau mau cari yang populer, kita tinggal tambahin/sisipkan 'where' dan misalkan yg vote dia lebih dari seratus.

return $query->where('votes','>',100);

}

cara untuk memanggilnya kita bisa langsung panggil di methodnya

Contoh :

use App\Models\User;

$users = User::popular()->active->orderBy('created\_by')->get();

#Method When (ada di collection)

when method itu akan dijalankan ketika first argumen berisi true

#null coalescing operator

null coalescing operator yaitu (??) sudah ditambahkan di php 7 supaya menambahkan syntactic sugar (pemanis, supaya nulisnya gampang) saat kita butuh untuk menggunakan ternary dan juga digunakan untuk ngecek isset()

#pagination

cara menggunakan pagination adalah kita tinggal panggil method paginate() saja, dan isinya adalah jumlah postingan yg ingin ditampilkan.

contoh : (didalam PostControllernya)

return view('posts', [

"title" => "All Posts" . $title,

"posts" => Post::latest()->filter(request(['search', 'category', 'author']))->paginate(7), // filter ini didapatkan dari Post Model

'gambar' => Controller::BASEURL,

'active' => 'posts'

]);

dan jika ingin menampilkan paginationnya di halaman view tinggal kasih {{ $posts->links() }}. ini defaultnya tampilan paginationnya adalah memakai tailwind, jika ingin mengubahnya menjadi tampilan dari bootstrap, kita pergi ke folder App->providers->appserviceprovider.php

nah nanti di method boot nya, kasih code :

use Illuminate\Pagination\Paginator;

Paginator::useBootstrap();

**# 419 | PAGE EXPIRED**

CSRF (Cross Side Request Forgery) Protection

CSRF ini adalah sebuah teknik serangan terhadap website kita, dimana serangan ini memalsukan (forgery) request dari website yg lain, jadi istilahnya ngebajak website kita, jadi bisa aja ada request yg dilakukan oleh website lain ke website kita, menggunakan url yg sesuai, jadi yg akan di kerjakan route tertentu di dalam aplikasi kita, padahal itu requestnya bukan dari website kita, melainkan dari website orang lain

Di laravel ada solusinya, namanya Preventing CSRF Requests, jadi kita akan menjaga requestnya agar selalu dikirimkan dari website kita, dengan menggunakan csrf\_token, jadi nanti website kita akan mengenerate sebuah token didalam session, dan nanti akan dicocokan dengan yg ada di requestnya, kalo sama berarti aman.

cara menggunakannnya adalah kita hanya perlu menuliskan @csrf di dalam form kita, nah nanti dibelakang layar, blade kita akan menerjemahkan @csrf ini jadi <input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

**#authentication**

=sebetulnya laravel ini menyediakan yg namanya starter kits, jadi ada semacam plugin atau aplikasi yg khusus menangani masalah authentication ini, ada yg namanya laravel breeze, dan ada satulagi namanya laravel jet-stream, dengan menggunakan aplikasi itu, kita dapat dengan mudah menangani masalah autentifikasi ini, mulai dari registrasi user, login, dan ada juga fitur verifikasi lewat email, ada remember me, ada forgot password, pokoknya semua yg ada hubungannya dengan authetication dan authorization, itu sudah ditangani oleh stater kits ini.

**baca aja dokumentasinya di laravel tentang authentication->starter kits->laravel breeze**

Sekarang kita akan pakenya yg manually authentication, laravel punya authentication services lewat facade

**#middleware**

middleware menyediakan mekanisme yang memudahkan kita untuk melakukan inspeksi dan filetering http request, jadi request kita di filter dulu sama middleware ini. contohnya laravel sudah menyediakan middleware didalamnya, yg melakukan verifikasi apakah seorang user dari aplikasi kita itu udah ter-autentikasi atau belum, jika belum maka middleware tadi akan melakukan redirect user tadi ke halaman login, atau halaman yg kita inginkan terserah. tapi kalau misalnya udah ter-autentifiksi, nanti si middlewarenya itu memperbolehkan kita untuk lanjut masuk ke dalam aplikasinya.

ada yg namanya global middleware, yg artinya secara otomatis dijalankan, setiap request http kita dijalankan, adanya di **app/Http/Kernel.php**