Rapikan Controller

Pengantar

Controller memiliki peran besar dalam MVC(Model–View-Controller) project. Controller secara efektif sebagai penghubung yang mengambil request dari user, melakukan logic dan mengembalikan sebuah response. Jika kamu pernah mengerjakan proyek besar, kamu akan melihat di proyek tersebut memiliki banyak controller dan controller tersebut akan mulai menjadi berantakan dengan cepat tanpa anda sadari. Dibab ini kita akan melihat bagaimana merapikan controller yang bereantakan dalam laravel.

Untuk Pembaca

Bab ini ditujukan untuk siapa saja yang baru saja belajar laravel, yang mengerti dasar-dasar controller. Konsep yang dibahas relatif mudah untuk dipahami.

Masalah Controller yang Berantakan

Controller yang berantakan disebabkan oleh beberapa masalah bagi developer yaitu:

1. **Membuat controller sulit untuk melacak bagian tertentu dari kode atau kegunaannya.** Jika kamu ingin mengerjakan bagian tertentu dari sebuah kode yang ada di dalam controller yang berantakan, kamu akan menghabiskan waktu untuk mencari fungsi atau method dalam sebuah controller. Ketika kamu merapikan controller yang dipisahkan secara logic maka akan lebih mudah.
2. **Membuat controller sulit untuk menemukan bug berada secara akurat.** Seperti yang akan kita lihat dicontoh kode nanti, jika kita menangani authorization, validation, bisnis logic dan response dalam satu tempat, maka akan sulit untuk menemukan bug secara akurat.
3. **Membuat controller lebih sulit dalam menulis test untuk request yang kompleks.** Terkadang sulit menulis test untuk

method controller yang kompleks yang memiliki banyak baris kode dan melakukan banyak hal. Merapikan kode membuat testing lebih mudah. Kami akan membahas testing nanti dibab 6.

Controller yang Berantakan

Dibab ini,kita akan menggunakan sebuah contoh dari UserController:

```php
class UserController extends Controller
{
    public function store(Request $request): RedirectResponse
    {
        $this->authorize('create', User::class);

        $request->validate([
            'name'     => 'string|required|max:50',
            'email'    => 'email|required|unique:users',
            'password' => 'string|required|confirmed',
        ]);

        $user = User::create([
            'name'     => $request->name,
            'email'    => $request->email,
            'password' => $request->password,
        ]);

        $user->generateAvatar();
        $this->dispatch(RegisterUserToNewsletter::class);

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

    public function unsubscribe(User $user): RedirectResponse
    {
        $user->unsubscribeFromNewsletter();

        return redirect(route('users.index'));
    }
}
```

Agar gambar diatas tetap bagus dan mudah dibaca, saya tidak menyertakan method `index()`, `edit()`, `update()`, dan `delete()` didalam controller. Tapi kita berasumsi bahwa method tersebut tetap ada dan kami juga menggunakan teknik dibawah ini untuk merapikan method tersebut juga. Untuk sebagian besar bab ini, kita akan fokus untuk merapikan atau mengoptimalkan method `store()`

Pindahkan Validation dan Authorization ke Form Request

Salah satu yang dapat kita lakukan adalah dengan memindahkan validation dan authorization ke class form request. Jadi, ayo kita lihat bagaimana kita bisa melakukan ini dicontroller untuk `store()` method.

Gunakan artisan command untuk membuat sebuah form request baru:

```
php artisan make:request StoreUserRequest
```

Command diatas akan membuat sebuah `app\Http\Request\StoreUserRequest` baru, seperti berikut:

```
class StoreUserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return false;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

kita akan menggunakan method `authorize()` untuk menentukan jika user diizinkan untuk melakukan request. Method tersebut akan mengembalikan true jika mereka diizinkan dan jika tidak maka

mengembalikan false. Kita bisa juga menggunakan method `rules()` untuk menentukan sebuah validation rules yang akan dijalankan pada request body. Kedua method akan dijalankan secara otomatis sebelum kita berhasil menjalankan sebuah kode didalam method controller tanpa perlu memanggil salah satu dari mereka secara manual. Jadi, ayo kita pindahkan authorization dari method controller store kedalam method `authorize()`. Setelah itu, kita bisa pindahkan validation rules dari controller ke method `rules()`. Kita juga sekarang mempunyai sebuah form request seperti ini:

```php
class StoreUserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize(): bool
    {
        return Gate::allows('create', User::class);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(): array
    {
        return [
            'name'     => 'string|required|max:50',
            'email'    => 'email|required|unique:users',
            'password' => 'string|required|confirmed',
        ];
    }
}
```

controller kita seharusnya terlihat seperti ini:

```php
class UserController extends Controller
{
    public function store(StoreUserRequest $request): RedirectResponse
    {
        $user = User::create([
            'name'     => $request->name,
            'email'    => $request->email,
            'password' => $request->password,
        ]);

        $user->generateAvatar();
        $this->dispatch(RegisterUserToNewsletter::class);

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

    public function unsubscribe(User $user): RedirectResponse
    {
        $user->unsubscribeFromNewsletter();

        return redirect(route('users.index'));
    }
}
```

Perhatikan bagaimana controller kita, kita mengubah argument pertama method `store()` dari `\Illuminate\Http\Request` ke `\App\Http\Requests\StoreUserRequest`. Kita juga mengurangi beberapa kode didalam method controller dengan memisahkannya ke dalam class request.

Catatan: Agar bekerja otomatis, kamu harus memastikan bahwa controller menggunakan traits `\Illuminate\Foundation\Auth\Access\AuthorizesRequests` dan `\Illuminate\Foundation\Validation\ValidatesRequests`.
Keduanya secara otomatis ditambahkan kedalam controller yang disediakan laravel saat instalasi baru. Jadi, jika controller kamu sudah extend ke controller (base), maka controller siap digunakan. Jika tidak maka pastikan kedua traits tersebut ada di dalam controller (base).

Pindahkan Common Login ke Actions atau Services

Langkah lain yang dapat kita lakukan untuk merapikan method store() pada controller yaitu memindahkan “bisnis logic” kita ke action atau service class.

Dalam kasus ini, kita bisa melihat bahwa fungsi utama dari method store() adalah membuat sebuah user, generate avatar user dan menugaskan kepada antrian job untuk mendaftarkan news letter user. Dalam pendapat pribadi saya, action lebih cocok untuk contoh ini dibandingkan service. Saya lebih memilih untuk menggunakan action untuk tugas kecil yang hanya melakukan satu hal tertentu. Sedangkan untuk kode yang lebih banyak berpotensi membuat panjangnya ratusan baris dan melakukan banyak hal, maka akan cocok untuk menggunakan service

Jadi, ayo kita buat action dengan membuat folder Actions baru didalam folder app kemudian buat sebuah class baru `StoreUserAction.php`. Kita dapat memindahkan kode kita ke action seperti berikut:

```php
class StoreUserAction
{
    public function execute(Request $request): void
    {
        $user = User::create([
            'name'     => $request->name,
            'email'    => $request->email,
            'password' => $request->password,
        ]);

        $user->generateAvatar();
        $this->dispatch(RegisterUserToNewsletter::class);
    }
}
```

Sekarang kita akan mengubah controller menggunakan action:

```
class UserController extends Controller
{
    public function store(
      StoreUserRequest $request,
      StoreUserAction $storeUserAction
    ): RedirectResponse {
        $storeUserAction->execute($request);

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

    public function unsubscribe(User $user): RedirectResponse
    {
        $user->unsubscribeFromNewsletter();

        return redirect(route('users.index'));
    }
}
```

Seperti yang anda lihat, sekarang kita dapat memindahkan bisnis logic keluar dari controller method ke dalam action. Ini sangat berguna, seperti saya sebutkan sebelumnya, controller pada dasarnya adalah penghubung untuk request dan response. Jadi, kita mengurangi yang berhubungan dengan apa yang kode lakukan dengan mebuat kode logic terpisah. Sebagai contoh, jika kita ingin mengecek authorization atau validation, kita akan langsung mengecek pada form request. Jika kita akan mengecek apa yang sedang dilakukan dengan request data, kita dapat mengecek pada action.

Manfaat besar lainnya untuk mengabstraksi kode menjadi terpisah berdasarkan class membuat testing lebih mudah dan cepat.

Menggunakan DTOs dengan Action

Manfaat besar lainnya dari memisahkan bisnis logic ke dalam service dan class adalah dapat menggunakan logic ditempat berbeda tanpa melakukan duplikasi kode. Contohya, ayo asumsikan bahwa kita memiliki sebuah `UserController` yang menangani tradisional web request dan sebuah `Api\UserController` yang menangani API request.

Untuk argument, kita dapat berasumsi bahwa struktur umum dari method `store()` untuk kedua controller akan sama. Tapi apa yang akan kita lakukan jika API request tidak menggunakan kolom `email`, tetapi menggunakan `email_address`? Kita tidak dapat melakukan pass request object ke class `StoreUserAction` karena class tersebut membutuhkan request object yang memiliki email.

Untuk solusi dari masalah tersebut, kita dapat menggunakan DTOs (data transfer object). Ini adalah cara yang sangat berguna untuk memisahkan data dan dapat menyebarkan ke seluruh proyek tanpa terikat apapun (dalam kasus ini adalah request).
Untuk menambahkan DTOs kedalam proyek kita, dapat menggunakan paket dari spatie `spatie/data-transfer-object` dan menggunakan command berikut untuk melakukan instalasi.

```
composer require spatie/data-transfer-object
```

Sekarang paket tersebut telah terinstall, Ayo buat folder DataTransferObject didalam folder app dan buat sebuah class `StoreUserDTO.php`. Kemudian pastikan bahwa class DTO yang kita buat melakukan extend terhadap `Spatie\DataTransferObject\DataTransferObject`. Kita kemudian bisa definsikan 3 propery seperti berikut.

```
class StoreUserDTO extends DataTransferObject
{
    public string $name;

    public string $email;

    public string $password;
}
```

Jika sudah selesai, kita dapat menambahkan method baru di `StoreUserRequest` sebelumnya ditambahkan dan return sebuah class `StoreUserDTO` seperti:

```php
class StoreUserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize(): bool
    {
        return Gate::allows('create', User::class);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(): array
    {
        return [
            'name'     => 'string|required|max:50',
            'email'    => 'email|required|unique:users',
            'password' => 'string|required|confirmed',
        ];
    }

    /**
     * Build and return a DTO.
     *
     * @return StoreUserDTO
     */
    public function toDTO(): StoreUserDTO
    {
        return new StoreUserDTO(
            name: $this->name,
            email: $this->email,
            password: $this->password,
        );
    }
}
```

Sekarang kita dapat melakukan update di controller untuk melakukan pass DTO ke class action:

```php
class UserController extends Controller
{
    public function store(
      StoreUserRequest $request,
      StoreUserAction $storeUserAction
    ): RedirectResponse {
        $storeUserAction->execute($request->toDTO());

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

    public function unsubscribe(User $user): RedirectResponse
    {
        $user->unsubscribeFromNewsletter();

        return redirect(route('users.index'));
    }
}
```

Terakhir, kita harus melakukan update pada method class action untuk menerima DTO sebagai argument daripada request object:

```php
class StoreUserAction
{
    public function execute(StoreUserDTO $storeUserDTO): void
    {
        $user = User::create([
            'name'     => $storeUserDTO->name,
            'email'    => $storeUserDTO->email,
            'password' => $storeUserDTO->password,
        ]);

        $user->generateAvatar();
        $this->dispatch(RegisterUserToNewsletter::class);
    }
}
```

Hasil dari apa yang telah kita lakukan, sekarang kita telah berhasil memisahkan action dari request object. Ini artinya kita dapat menggunakan kembali action ini dalam tempat berbeda pada projek tanpa terikat terhadap spesifik struktur request. Kita sekrang akan bisa juga menggunakan pendekatan ini untuk CLI environment atau antrian job karena tidak terikat ke web request. Sebagai contoh, jika aplikasi kita memiliki fungsi untuk melakukan import user dari sebuah csv file, kita akan bisa membuat DTO dari csv data dan melakukan pass ke dalam action.

Kembali lagi kemasalah sebelumnya yaitu API request yang menggunakan `email_address` daripada `email`, kita sekarang akan dapat solusi dengan membuat DTO sederhana dan melakukan mengisi DTO `email` dengan request `email_address`. Sekarang bayangkan API request memiliki request class yang terpisah. Seperti contoh dibawah ini

```php
class StoreUserAPIRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize(): bool
    {
        return Gate::allows('create', User::class);
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(): array
    {
        return [
            'name'          => 'string|required|max:50',
            'email_address' => 'email|required|unique:users',
            'password'      => 'string|required|confirmed',
        ];
    }

    /**
     * Build and return a DTO.
     *
     * @return StoreUserDTO
     */
    public function toDTO(): StoreUserDTO
    {
        return new StoreUserDTO(
            name: $this->name,
            email: $this->email_address,
            password: $this->password,
        );
    }
}
```

Menggunakan Resource atau Single-Use Controller

Cara terbaik untuk menjaga agar controller tetap rapi adalah pastikan bahwa keduanya adalah “Resource Controller” atau “Single-use Controller”. Sebelum kita melangkah lebih jauh dan mecoba mengubah contoh controller, Mari kita lihat apa arti dari keduanya.

Resource controller adalah sebuah controller yang menyediakan fungsionalitas berdasarkan resource tertentu. Jadi dalam kasus kita, resource kita adalah user Model dan kita ingin agar dapoat melakukan semua operasi CRUD(Create Read Update Delete) dalam model user. Sebuah resource controller umumnya mempunyai method

`index()`, `create()`, `store()`, `edit()`, `update()` dan `destroy()`. Tidak semua method perlu ada, tapi tidak akan memiliki method lain dalam list diatas. Dengan menggunakan jenis controller tersebut, kita bisa membuat routing lebih RESTful.

Single-use Controller adalah sebuah controller yang hanya memiliki satu method public `__invoke()`. Controller sangat berguna jika kamu memiliki sebuah controller yang methodnya tidak cocok dengan satu dari method RESTful yang kita miliki di resource controller.

Berdasarkan informasi diatas, kita dapat melihat bahwa `UserController` mungkin bisa diperbaiki dengan memindahkan method unsubscribe ke single-use controller.

Jadi ayo buat sebuah controller baru menggunakan artisan command:

```
php artisan make:controller UnsubscribeUserController -i
```

Lihat bagaimana kita menambahkan `-i` pada command di atas yang akan membuat controller baru yang invokable, single-use. Kita sekarang memiliki sebuah controller seperti ini:

```
class UnsubscribeUserController extends Controller
{
    public function __invoke(Request $request)
    {
        //
    }
}
```

Sekarang kita dapat memindahkan kode method dan menghapus unsubscribe method dari controller lama:

```
class UnsubscribeUserController extends Controller
{
    public function __invoke(Request $request): RedirectResponse
    {
        $user->unsubscribeFromNewsletter();

        return redirect(route('users.index'));
    }
}
```

Pastikan bahwa anda ingat untuk mengubah route pada file `routes/web.php` gunakan `UnsubscribeController` daripada `UserController` untuk method ini.

Kesimpulan

Pada bab ini seharusnya memberikan kamu sebuah wawasan tentang berbagai cara yang dapat dilakukan untuk merapikan controller pada projek laravel. Harap diingat bahwa teknik yang digunakan disini adalah opini pribadi saya. Saya yakin bahwa ada developer lain yang akan menggunakan pendekatan yang benar-benar berbeda untuk membangun controller mereka. Bagian yang paling penting adalah tetap konsisten dan menggunakan pendekatan yang cocok dengan alur kerja anda (team).

Tips Cepat & Mudah Untuk Mempercepat Aplikasi

Pengantar

Diperkirakan 40% orang akan meninggakan sebuah website jika membutukan waktu lebih lama dari 3 detik saat dimuat. Jadi dari sudut pandang bisnis sangat luar biasa penting untuk memastikan bahwa waktu memuat tetap di bawah 3 detik.

Oleh karena itu, Setiap kali saya menulis kode untuk proyek laravel, saya mencoba untuk memastikan kode optimal semaksimal mungkin dalam waktu yang diberikan serta kendala biaya. Jika saya pernah bekerja untuk proyek apapun, saya selalu mencoba untuk menggunakan teknik tersebut untuk memperbarui kode yang berjalan lambat untuk meningkatkan pengalaman user.

Dalam bab ini, kita akan melihat beberapa teknik yang saya gunakan (atau saran dari developer lain) untuk mempercepat peningkatan untuk klien dan web laravel serta aplikasi pribadi.

Untuk Pembaca

Bab ini ditujukan untuk laravel developer dalam tahap apapun yang sedang mencari beberapa tips cepat dan mudah untuk diterapkan pada aplikasi yang merekan bangun.

Hanya Ambil Field Yang Dibutukan di Query Database

Salah satu cara mempercepat laravel dengan mengurangi jumlah data yang ditarik antara aplikasi dan database. Cara yang dapat kamu lakukan dengan hanya spesifik terhadap kolom yang dibutuh dalam query dengan menggunakan select. Sebagai contoh, ibaratkan kamu memiliki sebuah User model yang mempunyai 20 kolom berbeda. Sekarang, bayangkan kamu memiliki 10,000 user dalam sistem dan

kamu mencoba melakukan beberapa proses setiap dari kolom tersebut. Contoh kodenya seperti berikut:

```
$users = User::all();

foreach ($users as $user) {
    // Do something here
}
```

Query diatas akan bertanggung jawab untuk mengambil 200,000 kolom data. Tapi, bayangkan ketika kamu memproses setiap user, kamu sebenarnya hanya menggunakan kolom `id`, `first_name` dan `last_name`. Jadi ini artinya dari 20 kolom, 17 diantaranya tidak digunakan. Jadi yang bisa kita lakukan adalah secara spesifik menentukan kolom yang diperlukan dalam query. Dalam kasus ini, kodenya terlihat seperti ini:

```
$users = User::select(['id', 'first_name', 'last_name'])->get();

foreach ($users as $user) {
    // Do something here
}
```

Dengan melakukan ini, kita mengurangi jumlah kolom yang dikembalikan dalam query dari 200,000 menjadi 30,000. Meskipun mungkin ini tidak akan terlalu berefek pada beban IO database, tapi akan mengurangi network traffic antara aplikasi dan database. Ini (mungkin) dikarenakan lebih sedikat data diserialize, dikirim dan deserialize dibandigkan jika mengambil semua kolom yang ada. Dengan mengurangi network traffic serta jumlah data yang diperlukan untuk diproses, akan membantu mempercepat web laravel. Harap dicatat, dalam contoh diatas kamu sebenarnya mungkin tidak pernah melakukan sesuatu seperti ini dan kamu akan menggunakan chunk atau pagination tergantung situasinya. Contoh diatas hanya hanya untuk menunjukan kemungkinan lebih mudah dalam mengimplementasi solusi.

Solusi ini mungkin tidak akan berpengaruh besar dalam sebuah web atau aplikasi kecil. Namun, ini adalah sesuatu yang dapat membantu mengurangi waktu muat dalam aplikasi dimana performa adalah hal penting yang harus dimiliki. Kamu mungkin juga melihat peningkatan yang bagus jika anda melakukan query ke sebuah table yang memiliki kolom BLOB atau TEXT. Kedua field tersebut dapat menyimpan data megabyte dan berpotensi meningkatkan lama waktu query. Jadi jika model tabel anda memiliki salah satu dari kolom tersebut, pertimbangkan secara spesifik tabel yang kamu butuhkan dalam query untuk mengurangi waktu load.

Gunakan Eager Loading Sebisa Mungkin

Ketika kamu menggunakan sebuah model untuk mengambil data dari database dan melakukan sebuah proses dari relasi model, sangat penting untuk menggunakan eager loading. Eager loading sangat sederhana di laravel dan pada dasarnya mencegah dalam menghadapi masalah N+1 pada data. Masalah ini disebabkan karena membuat query N+1 di database, dimana N adalah jumlah item yang diambil dari database. Untuk penjelasan yang lebih baik dan beberapa konteks, mari kita lihat contoh dibawah ini.

Bayangkan kamu memiliki dua model (Comment dan Author) dengan relasi 1-ke-1 diantara mereka. Sekarang bayangkan bahwa kamu memiliki 100 comment dan kamu ini melakukan loop setiap data dan mengeluarkan nama author.

Tanpa eager loading, kodenya akan seperti berikut:

```
$comments = Comment::all();

foreach ($comments as $comment ) {
    print_r($comment->author->name);
}
```

Kode diatas akan menghasilkan 101 query database! Query pertama akan mengambil semua data comment. Dan seratus query lainnya akan

berasal dari mengeluarkan nama author pada setiap pengulangan loop. Jelas sekali, ini dapat menyebabkan masalah performa dan memperlambat aplikasi, jadi bagaimana kita memperbaiki masalah ini?

Dengan menggunakan eager loading, kita bisa mengubah kode menjadi:

```
$comments = Comment::with('authors')->get();

foreach ($comments as $comment) {
    print_r($comment->author->name);
}
```

Seperti yang kamu lihat, kode diatas terlihat sama dan masih dapat dibaca. Dengan menambahkan `::with('authors')` akan mengambil semua comment dan query lainnya untuk mengambil author hanya sekali. Jadi artinya kita memangkas query dari **101** menjadi **2**.

Bagaimana Memaksa Laravel Menggunakan Eager Loading

Dalam laravel, kamu dapat mencegah lazy loading. Fitur ini sangat berguna karena membantu untuk memastikan bahwa relasi menggunakan eager loading. Sebagai hasilnya, dapat meningkatkan performa dan mengurangi jumlah query yang dibuat database seperti contoh diatas.

Sangat sederhana untuk mencegah lazy load. Semua yang perlu kita lakukan adalah menambahkan kode dibawah ke method `boot()` pada `AppServiceProvider`:

```
Model::preventLazyLoading();
```

Jadi pada AppServiceProvider akan terlihat seperti berikut:

```php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        // ...
        Model::preventLazyLoading();
        // ...
    }
}
```

Mengizinkan Eager Loading Pada Production Environment

Anda mungkin hanya ingin mengaktifkan fitur ini saat mengembangkan aplikasi pada local environment. Dengan melakukan hal itu, kamu akan diperingatkan jika kode kamu terdapat sebuah lazy loading ketika membangun sebuah fitur baru, tapi tidak mengganggu web production. Untuk alasan ini method `preventLazyLoading()` menerima boolean sebagai argument, jadi kita bisa menggunakan baris berikut:

```php
Model::preventLazyLoading(! app()->isProduction());
```

Jadi, pada `AppServiceProvider`, akan terlihat seperti berikut:

```php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        // ...
        Model::preventLazyLoading(! app()->isProduction());
        // ...
    }
}
```

Dengan melakukan ini, fitur akan dimatikan jika APP_ENV adalah production jadi query lazy loading yang lolos tidak akan menyebabkan exceptions pada web anda.

Apa Yang Terjadi Ketika Menggunakan Lazy Load

Jika kita telah mengaktifkan fitur lazy load pada service provider dan kita mencoba melakukan lazy load terhadap relasi sebuah model, Sebuah exception `Illuminate\Database\LazyLoadingViolationException` akan ditampilkan.

Untuk memberikan sedikit konteks, ayo gunakan model kita `Comment` dan `Author` dari contoh atas. Katakanlah kita mengaktikan fitur tersebut.

Contoh kode yang akan mengembalikan exception:

```
$comments = Comment::all();

foreach ($comments as $comment ) {
    print_r($comment->author->name);
}
```

Namun, kode dibawah ini tidak akan mengembalikan exception:

```
$comments = Comment::with('authors')->get();

foreach ($comments as $comment ) {
    print_r($comment->author->name);
}
```

Singkirkan Paket Yang Tidak Dibutukan dan Diinginkan

Buka composer.json file dan lihat setiap dependecies. Untuk setiap dependencies, tanya pada diri anda “apakah saya benar-benar membutuhkan paket ini?”. Jawaban anda sebagian besar adalah ya, tapi untuk sebagian mereka, mungkin tidak.

Setiap kali, kamu menambahkan sebuah composer library baru kedalam project anda, kamu berpotensi menambahkan extra code yang kemungkinan tidak perlu dijalankan. Laravel package biasanya terdapat service providers yang dijalankan pada setiap request yang melakukan register service dan menjalankan kode.

Jadi, katakanlah jika kamu menambahkan 20 laravel package kedalam aplikasi atau project, kemungkinan minimal 20 class akan di instansiasi dan dijalankan pada setiap request. Meskipun ini tidak akan berdampak besar terhadap performa untuk situs atau aplikasi dengan traffic kecil, kamu pasti dapat melihat perbedaannya pada aplikasi yang lebih besar.

Solusinya adalah menentukan apakah kamu membutuhkan sebuah paket itu. Mungkin kamu menggunakan sebuah paket yang menyediakan berbagai fitur tapi anda sebenarnya hanya menggunakan satu fitur kecil dari package tersebut. Tanya kediri anda “Bisakah saya menulis kode ini dan menghapus package tersebut” tentu saja karena kendala waktu, tidak selalu untuk menulis kode anda sendiri, karena anda harus menulis, melakukan test dan maintain kode. Paling tidak dengan menggunakan package, kamu dapat menfaatkan komunitas open-source untuk melakukan hal tersebut. Tapi jika sebuah paket tersebut sederhana dan dapat diganti dengan kode sendiri, maka pertimbangkan untuk menghapusnya.

Cache, Cache, Cache!

Laravel datang dengan berbagai caching methods. Ini dapat membuat web anda menjadi lebih cepat atau ketikan web anda sedang aktif tanpa perlu membuat perubahan kode.

Route Caching

Karena cara laravel berjalan,menjalakan framework dan melakukan parse pada file route setiap request yang dibuat. Ini membutuhkan membaca file, parsing the content dan mempelajari serta memahami bagaimana cara aplikasi anda digunakan. Jadi laravel menyedia

sebuah perintah yang bisa digunakan untuk membuat sebuah single route yang dapat di parse lebih cepat.

```
php artisan route:cache
```

Tolong perhatikan jika anda menggunakan perintah ini dan mengubah route anda, anda perlu menjalakan perintah berikut:

```
php artisan route:clear
```

Ini akan menghapus file cached route jadi route terbaru dapat di daftarkan. Mungkin bermanfaat jika menambahkan 2 command tersebut kedalam script deploy jika belum ditambahkan. Jika tidak menggunakan sebuah script deploy, kamu mungkin akan menemukan package Laravel Executor yang berguna untuk membantu dalam menjalankan deployment.

Config Caching

Serupa dengan route caching, setiap kali sebuah request dibuat, laravel menjalankan setiap config file dalam project anda dibaca dan diparse. Jadi untuk menghentikan agar setiap file tidak perlu ditangani. Anda bisa menjalankan perintah yang dapat membuat satu file config cached.

```
php artisan config:cache
```

Seperti route caching diatas, kamu perlu untuk menjalankan perintah berikut setiap kali mengubah .env atau config file

```
php artisan config:clear
```

Event dan View Caching

laravel juga menyediakan dua perintah lainnya yang dapat digunakan untuk melakukan cache views dan events sehinggan sudah dikompilasi dan siap ketika sebuah request dibuat. Untuk cache events dan views, kamu dapat menggunakan beberapa perintah:

```
php artisan event:cache
```

```
php artisan view:cache
```

Seperti perintah caching lainnya, kamu perlu ingat untuk menghancurkan cache setiap kali anda membuat perubahan dengan menjalankan perintah:

```
php artisan event:clear
```

```
php artisan view:clear
```

Dimasa lalu, Saya melihat banyak developer melakukan cache pada config di local development environment dan menghabiskan waktu mencari tau kenapa perbuahan .env tidak muncul. Jadi, saya merekomendasikan hanya melakukan caching pada config dan routes di live sistem agar anda tidak mengalami masalah yang sama.

Caching Query dan Value

Didalam kode laravel, kamu bisa melakukan cache pada items untuk meningkatkan performa website. Sebagai sebuah contoh, bayangkan kamu memiliki query berikut:

```
$users = DB::table('users')->get();
```

Untuk membuat caching dengan query ini, kamu harus merubah code menjadi seperti berikut:

```
$users = Cache::remember('users', 120, function () {
    return DB::table('users')->get();
});
```

Kode diatas menggunakan method `remember()`. Yang dilakukan pada dasarnya adalah mengecek jika cache mempunyai sebuah item dengan key users. Jika ada, akan mengembalikan value cache. Jika tidak ada dalam cache, maka hasil yang di kembalikan adalah query `DB::table('users')→get()` dan juga di cache. Dalam kasus ini, item akan dicache untuk 120 detik.

Cache data dan query seperti ini bisa menjadi sangat efectiv untuk mengurangi pemanggilan database, mengurangi run time dan meningkatkan performa. Namun, sangat penting untuk diingat bahwa kamu mungkin terkadang perlu untuk menghapus item dari cache jika tidak valid.

Menggunakan contoh diatas, bayangkan kita telah melakukan query users cache. Sekarang bayangkan bahwa sebuah user telah dibuat, diupdate atau dihapus. Hasil dari query cached tidak akan valid dan up to date. Untuk memperbaiki ini, kita harus menggunakan laravel model observers to menghapus item dari cache. Artinya jika selanjutnya mencoba dan mengambil $users variable, sebuah database query baru akan memberikan hasil yang up to date.

Menggunakan Versi PHP Terbaru

Dengan setiap versi php keluar, performa dan kecepatan telah ditingkatkan. Kinsta menjalankan banyak test across banyak versi PHP dan platform berbeda (Laravel, Wordpress, Drupal, Joomla) dan hasilnya bahawa PHP 8.0 memberikan peningkatan performa.

Tips ini mungkin sedikit sulit diterapkan jika dibandingkan dengan tips diatas karena kamu perlu melakukan audit pada kode dan memastikan kamu bisa secara aman mengupdate versi terbaru PHP. Sebagai catatan tambahan, menggunakan sebuah rangkain test otomatis dapat membantu anda dalam melakukan upgrade.

Menggunakan Queues

Tips ini mungkin akan memakan waktu lebih lama dibandingkan beberapa tips diatas yang menggunakan kode untuk diimplementasikan. Meskipun begitu, tips ini mungkin menjadi satu yang paling bermanfaat dalam hal user experience.

Salah satu cara yang bisa digunakan adalah mengurangi waktu performa dengan menggunakan laravel queues. Jika sebuah code yang berjalan di controller atau class dalam sebuah request dimana tidak diperlukan untuk response di web browser, kamu bisa menggunakan queue.

Untuk lebih mudah dimengerti, lihat contoh berikut:

```
class ContactController extends Controller
{
    /**
     * Store a new contact.
     *
     * @param  Request  $request
     * @return JsonResponse
     */
    public function store(ContactFormRequest $request)
    {
        $request->storeContactFormDetails();

        Mail::to('mail@ashallendesign.co.uk')->send(
          new ContactFormSubmission()
        );

        return response()->json(['success' => true]);
    }
}
```

Kode diatas, ketika method store() dipanggil akan melakukan penambahan kontak form detail di database, mengirim email ke sebuah alamat untuk menginformasikan mereka mengenai form kontak submission baru dan mengembalikan response json. Masalah pada kode diatas adalah user harus menunggu sampai email terkirim sebelum menerima response di web browser. Meskipun ini mungkin hanya beberapa detik, bisa berpotensi membuat pengunjung keluar.

Untuk menggunakan sistem queue, kita harus mengubah kode kita menjadi seperti berikut:

```
class ContactController extends Controller
{
    /**
     * Store a new contact.
     *
     * @param  Request  $request
     * @return JsonResponse
     */
    public function store(ContactFormRequest $request)
    {
        $request->storeContactFormDetails();

        dispatch(function () {
            Mail::to('mail@ashallendesign.co.uk')->send(
              new ContactFormSubmission()
            );
        })->afterResponse();

        return response()->json(['success' => true]);
    }
}
```

Kode diatas pada method store() sekarang akan menambahkan kontak form detail pada database, queue pengiriman email dan mengembalikan response. Sekali response dikembalikan ke web browser user, email akan di tambahkan di queue akan diproses. Dengan melakukan ini, artinya kita tidak perlu menunggu email terkirim sebelum mengembalikan response.

Cek dokumentasi laravel untuk informasi bagaimana mengatur queue untuk web laravel atau aplikasi.

Kesimpulan

Pada bab ini seharusnya dapat memberikan gambaran bagaimana cara mempercepat project laravel tanpa perlu melakukan refactor sepenuhnya. Tentu saja, ada lebih banyak hal yang dapat anda lakukan tapi ini yang biasa saya gunakan ketika saya ingin cepat dalam meningkatkan performa.