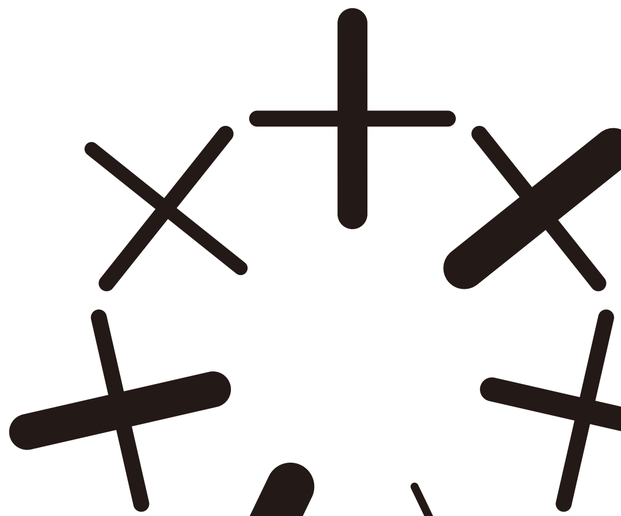
Laravelにおける
後悔しないための
アプリケーション設計

Laravel Osaka 2016

@localdisk



自己紹介

松尾 大

まつおまさる

@localdisk

エンジニア

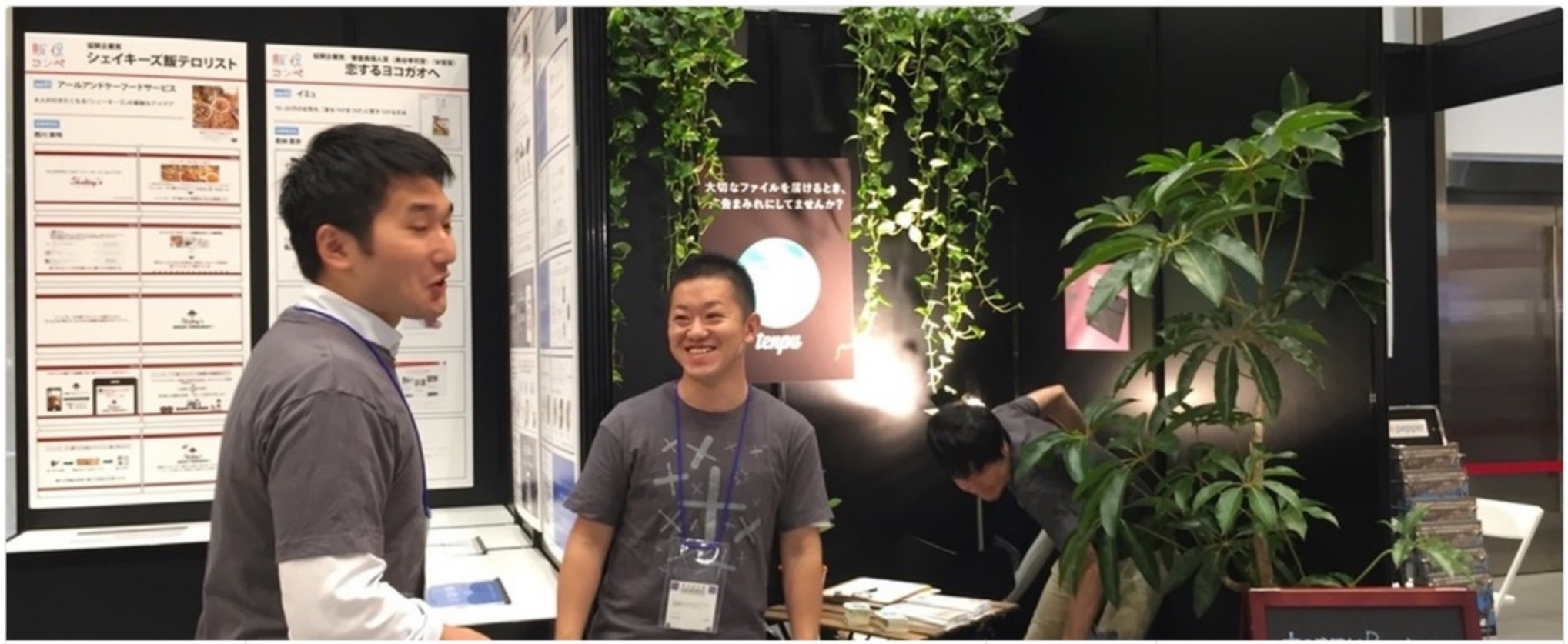
イノベータージャパン所属(エンジニア・デザイナ・ディレクタを募集中!)

好評発売中!!

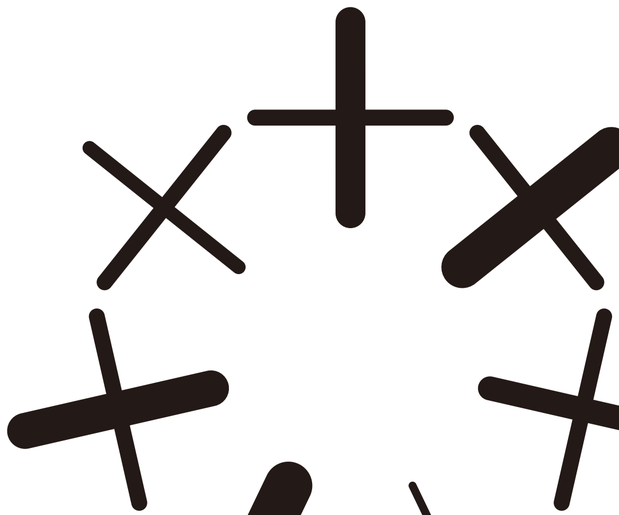
募集してます！！！

ゼロからサービスを開発し育てていきたい。そんなwebエンジニアを募集します

株式会社イノベーター・ジャパン

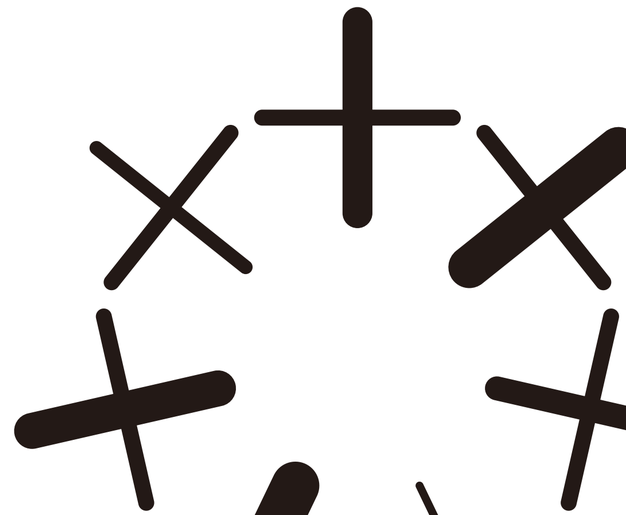


後悔しないアプリケーション設計とは?

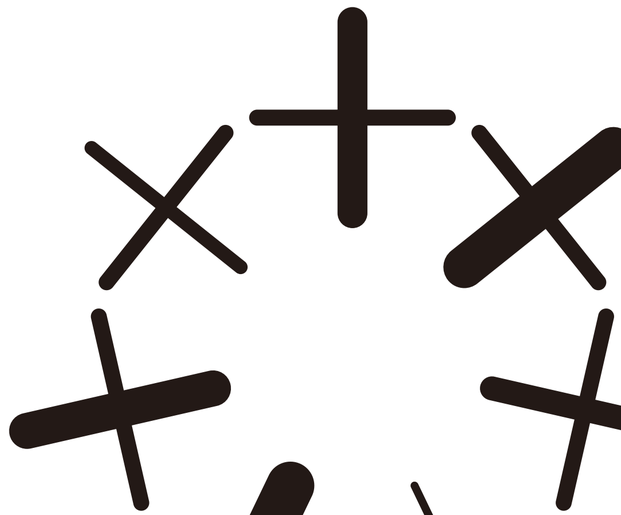


バグを出さない唯一の方法はプログラムを書かないことだ

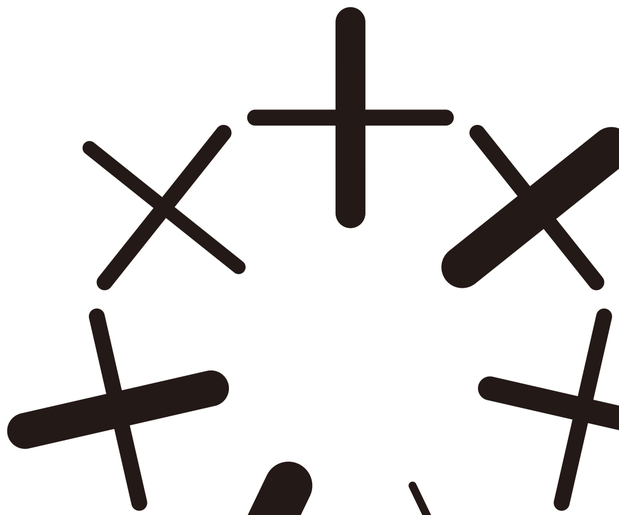
—プログラマの格言—



そういうわけにもいかない

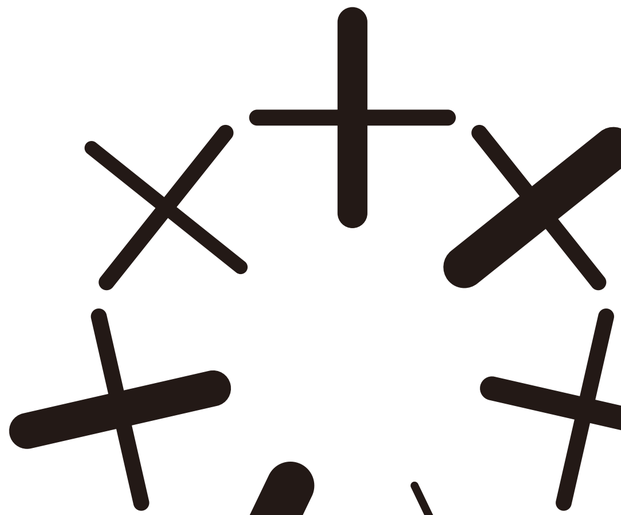


現実は厳しい



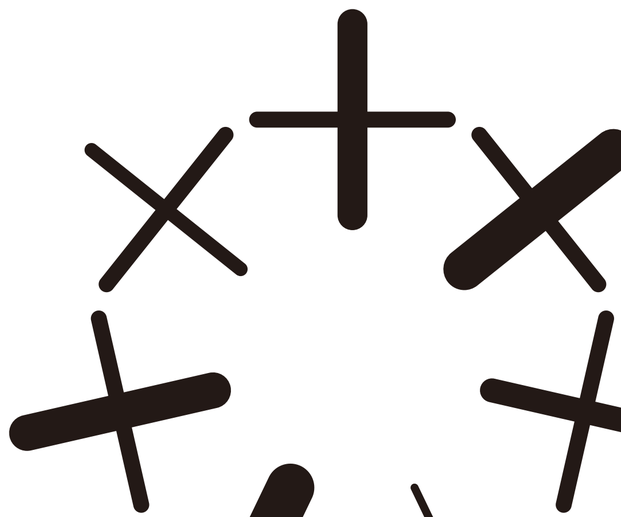
~~妥協して~~

シンプルさをたもつ

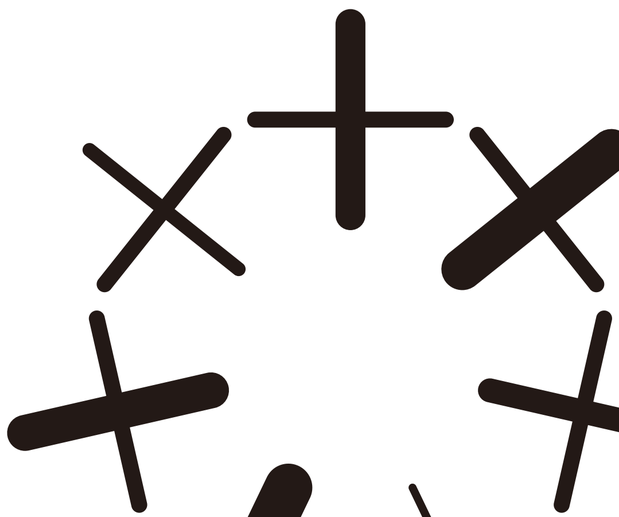


Agenda

- 後悔しないためにどうするか?
  - Facadeを控えてみる
  - Named Routes
  - FormRequest
  - ControllerでEloquentのメソッドチェイン禁止
  - Repository Pattern

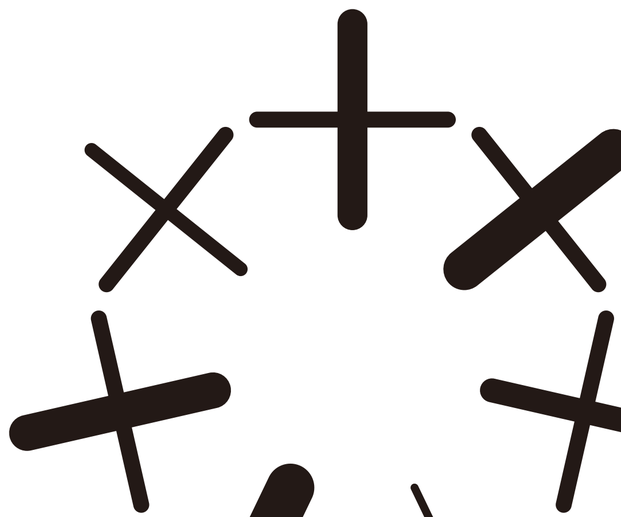


Facadeを控えてみる



Facade は便利だけど...

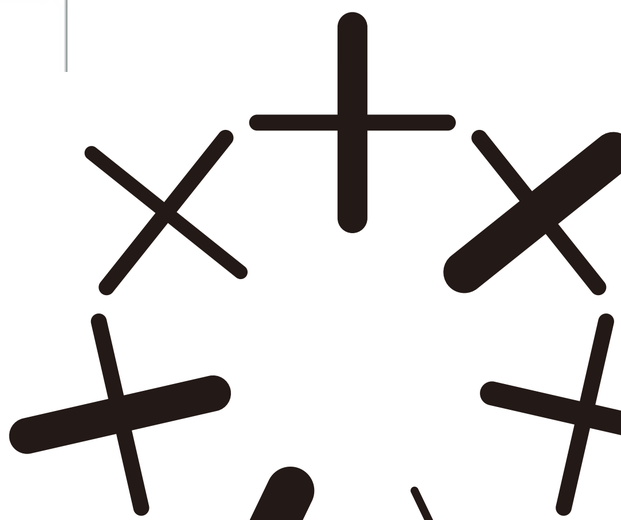
- 静的メソッドのように使える
  - 実際はサービスコンテナから取り出している
- どこからでも使える
  - Viewの中でも
  - \Auth::check()
- 結果、クラスの責務が大きくなる



DIを使おう

ファサード	クラス	サービスコンテナ結合
App	Illuminate\Foundation\Application	app
Artisan	Illuminate\Contracts\Console\Kernel	artisan
Auth	Illuminate\Auth\AuthManager	auth
Blade	Illuminate\View\Compilers\BladeCompiler	blade.compiler
Bus	Illuminate\Contracts\Bus\Dispatcher	
Cache	Illuminate\Cache\Repository	cache
Config	Illuminate\Config\Repository	config
Cookie	Illuminate\Cookie\CookieJar	cookie
Crypt	Illuminate\Encryption\Encrypter	encrypter
DB	Illuminate\Database\DatabaseManager	db
DB (Instance)	Illuminate\Database\Connection	

https://readouble.com/laravel/5.3/ja/facades.html#facade-class-reference



DIを使おう(2)

なるべく Contract 使おう

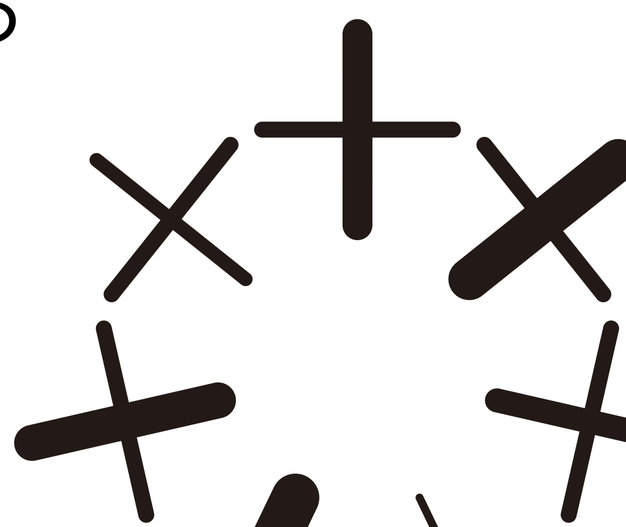
```php
/**
 * Register the core class aliases in the container.
 *
 * @return void
 */
public function registerCoreContainerAliases()
{
    $aliases = [
        'app'                  => ['Illuminate\Foundation\Application', 'Illuminate\Contracts\Container\Container', 'Illuminate\Contracts\Foundation\Application'],
        'auth'                 => ['Illuminate\Auth\AuthManager', 'Illuminate\Contracts\Auth\Factory'],
        'auth.driver'          => ['Illuminate\Contracts\Auth\Guard'],
        'blade.compiler'       => ['Illuminate\View\Compilers\BladeCompiler'],
        'cache'                => ['Illuminate\Cache\CacheManager', 'Illuminate\Contracts\Cache\Factory'],
        'cache.store'          => ['Illuminate\Cache\Repository', 'Illuminate\Contracts\Cache\Repository'],
        'config'               => ['Illuminate\Config\Repository', 'Illuminate\Contracts\Config\Repository'],
        'cookie'               => ['Illuminate\Cookie\CookieJar', 'Illuminate\Contracts\Cookie\Factory', 'Illuminate\Contracts\Cookie\QueueingFactory'],
        'encrypter'            => ['Illuminate\Encryption\Encrypter', 'Illuminate\Contracts\Encryption\Encrypter'],
        'db'                   => ['Illuminate\Database\DatabaseManager'],
        'db.connection'        => ['Illuminate\Database\Connection', 'Illuminate\Database\ConnectionInterface'],
        'events'               => ['Illuminate\Events\Dispatcher', 'Illuminate\Contracts\Events\Dispatcher'],
```

なぜ DI か?

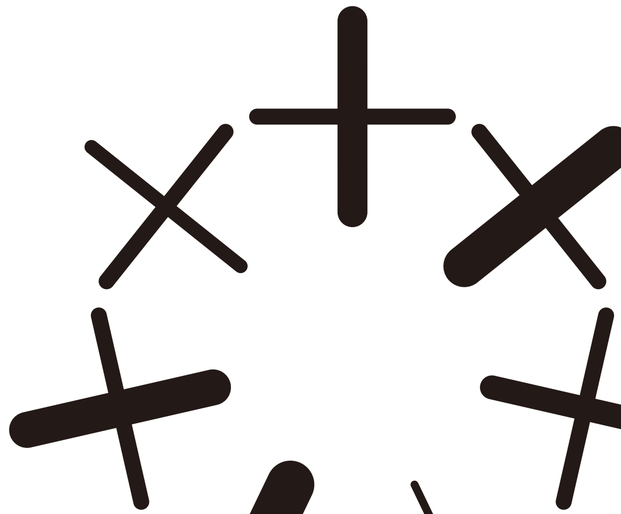
- Facade を使うかわりに DI をすると使用するとメソッドの引数に使用するクラスが並ぶ

```
public function store(User $user, Request $request, SessionManager $session, Factory $cache)
{
    ....
}
```

- あ、やばい。とわかる。
- Facadeを経由しないので若干早くなる



Named Routes

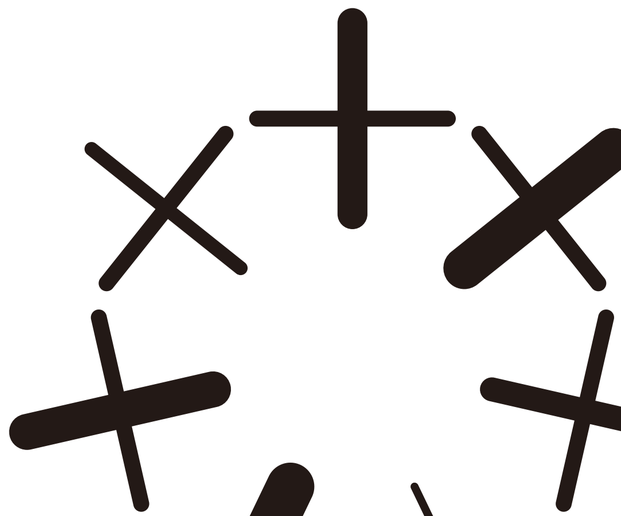


Named Routes

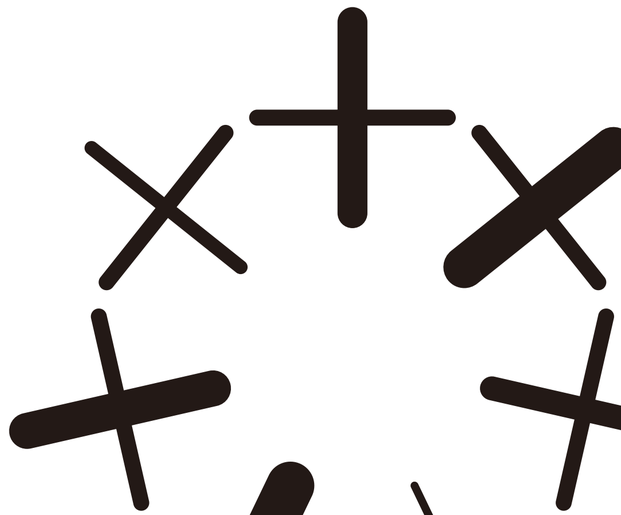
- Routeに名前をつける

```
// name メソッドで URL と 名前を関連付ける
Route::get('laravel/osaka', 'OsakaController@index')->name('osaka');
```

- route(‘osaka’); でURLが取得できる
- URLが変わっても安心



FormRequest

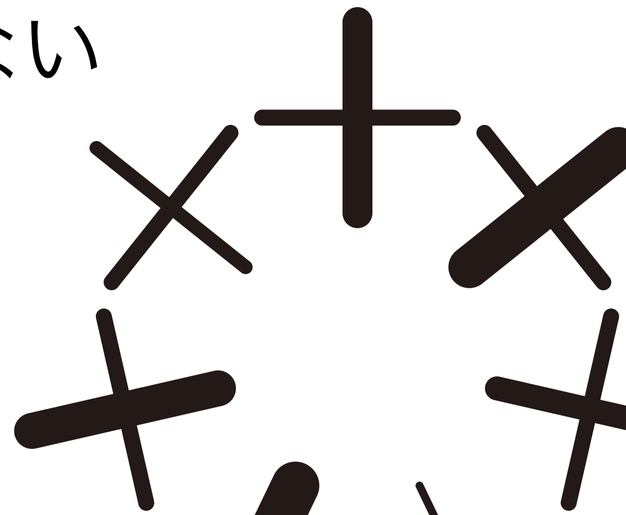


Validation の変遷

```php
    // Validation
    public function index(OsakaRequest $request)
    {
        $rules = [
            'name' => 'required',
            'email' => 'required|email',
        ];
        // Laravel 4
        \Validator::make($request->all(), $rules);

        // Laravel 5
        // ValidatesRequests trait により validate メソッドが実装された
        $this->validate($request, $rules);
    }
```

少しずつ記述量が減っている

しかしノイズになっているのは否めない



FormRequest

- php artisan make:request クラス名
- rulesメソッドを実装

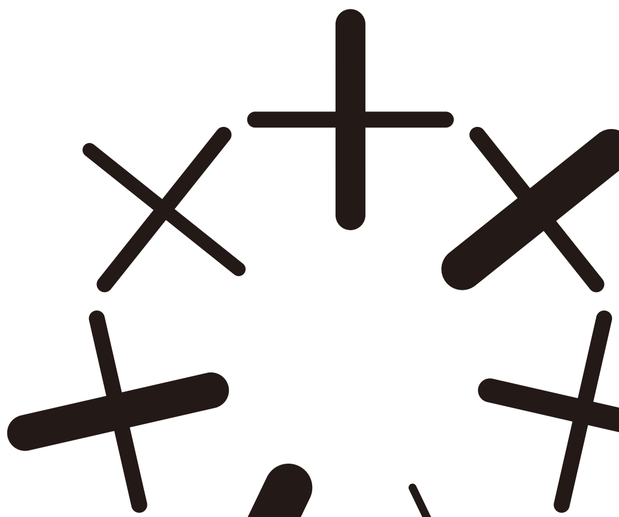
```
class OsakaRequest extends FormRequest
{

    public function rules()
    {
        return [
            'name' => 'required',
            'email' => 'required|email',
        ];
    }
}
```

FormRequest

- Controllerでメソッドインジェクションするだけ
- ノイズが消えてロジックに集中できる

```
class OsakaController extends Controller
{

    public function index(OsakaRequest $request)
    {
        //
    }

}
```

ControllerでEloquentのメソッドチェイン禁止



Model Binding

```php
Route::get('user/{id}', function($id) {
    $user = \App\User::findOrFail($id);
    dd($user);
});
```

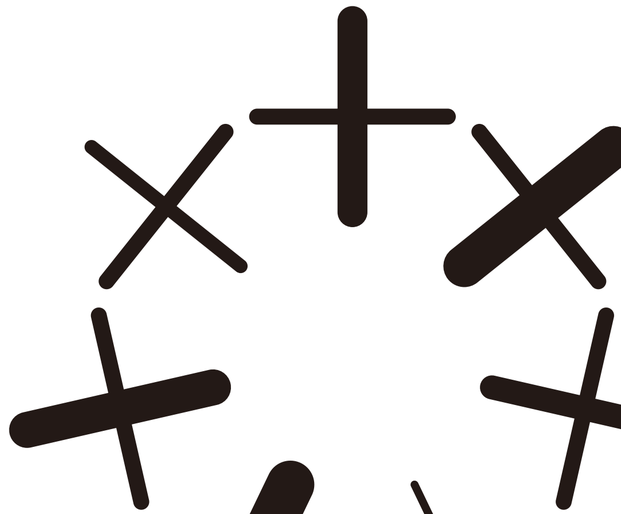
```php
Route::get('user/{user}', function(\App\User $user) {
    dd($user);
});
```

Controllerでメソッドチェインはダメ！

```php
public function search(Request $request)
{
    $users = User::where('name', 'like', "%$request->name%")
        ->where('status', '=', 1)
        ->latest()
        ->get();

    return view('users', compact('users'));
}
```

「同じ結果をAPIで返せるようにしといてね」

と言われたら…？

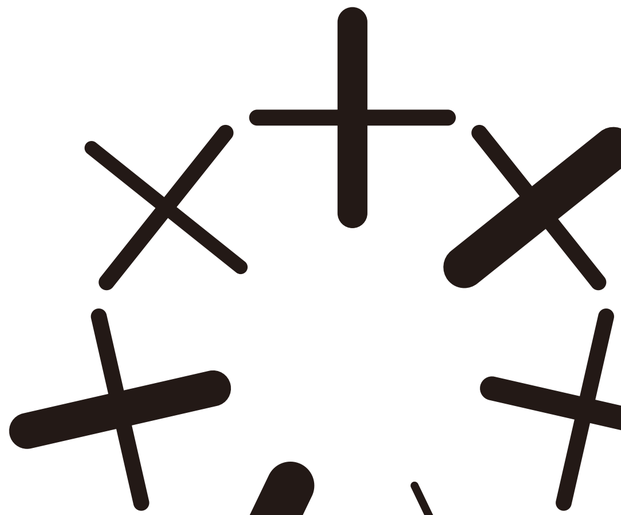


Modelにクエリを閉じ込める

```php
class User extends Authenticatable
{
    // App\User にメソッドを定義する
    public function search(array $query)
    {
        return $this->where('name', 'like', '%' . $query['name'] . '%')
            ->where('status', '=', $query['status'])
            ->latest()
            ->get();
    }
}
```

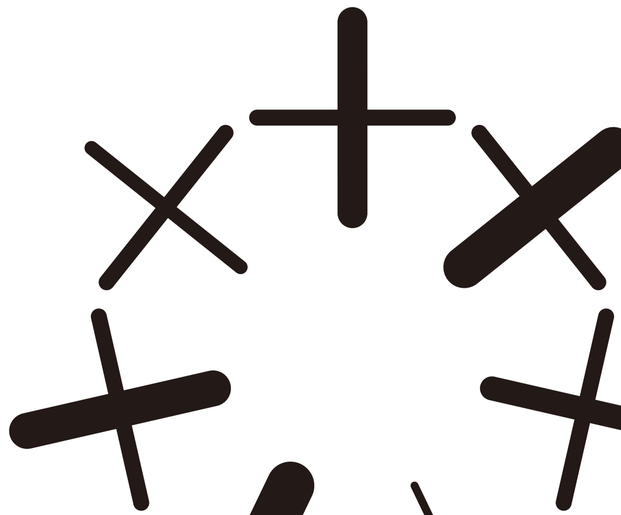
```php
class OsakaController extends Controller
{
    // コントローラーが簡潔になって再利用性もあがった
    public function search(Request $request, User $user)
    {
        $users = $user->search($request->only('name', 'query'));

        return view('users', compact('users'));
    }

}
```

Repository Pattern



なぜ Repository Pattern?

- Eloquentは User::find のように静的に呼べますが、Facadeではない。
- よってshouldRecieveというメソッドは存在しない
- EloquentはDBに依存している
- テストが困る



Repository Patternの作り方

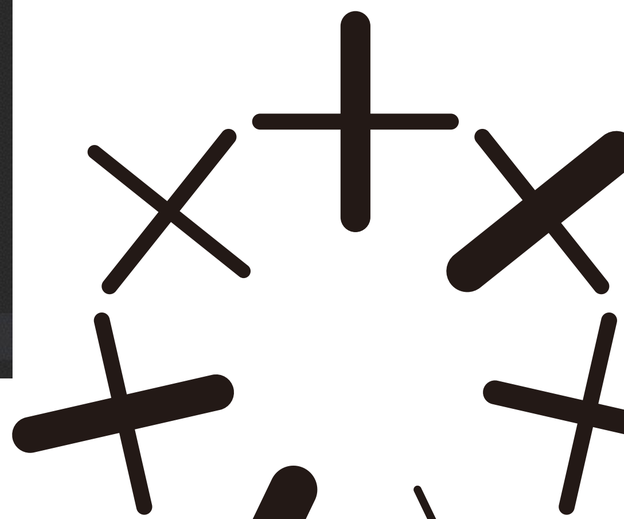
- インターフェースを定義

```php
interface UserRepository
{

    /**
     * Get the user with the given ID.
     *
     * @param  int  $id
     * @return \Illuminate\Contracts\Auth\Authenticatable|null
     */
    public function find($id);

    /**
     * Perform a basic user search by name or e-mail address.
     *
     * @param  string  $query
     * @param  \Illuminate\Contracts\Auth\Authenticatable|null  $excludeUser
     * @return \Illuminate\Database\Eloquent\Collection
     */
    public function search($query, $excludeUser = null);

}
```

Repository Patternの作り方

- インターフェースを実装。あとはコントローラでインジェクションする

```php
class UserRepository implements UserRepositoryContract
{

    /**
     * {@inheritdoc}
     */
    public function find($id)
    {
        //
    }

    /**
     * {@inheritdoc}
     */
    public function search($query, $excludeUser = null)
    {
        //
    }

}
```



質問とか？

おわり