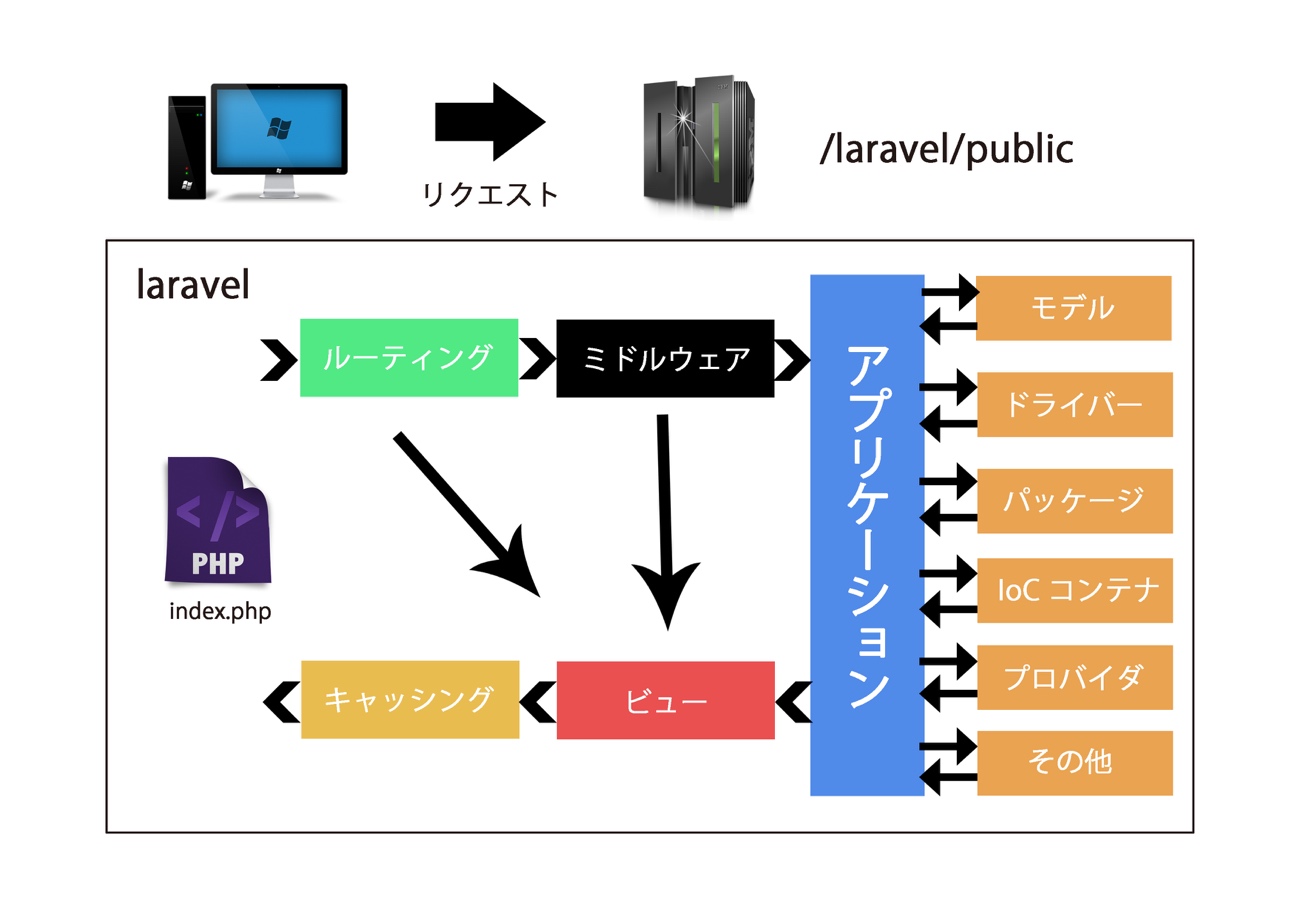
Laravel 実装イメージ（手順）



手順１：Route(ルート)作成

場所：

アプリルート -> web.php

APIルート -> api.php

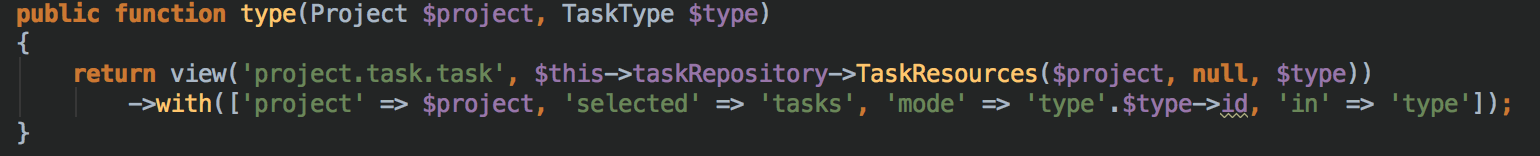
バッチ/コマンド -> console.php

注意点：

1. ルートグループの活用、出来るだけグループ化する。



1. 名前空間、Prefix、ミドルウェア（処理順番あり）
2. 認証、認可、リダイレクト、前処理　-> ミドルウェア
3. 命名ルートにする必要かどうか
4. URL値渡し
   1. ルートによるDI(依存性注入) -> 要定義（ルートサービスプロバイダ）
   2. 複数の値があった場合 -> Route::classで受ける



1. 簡単な処理はクロージャーを活用する（API叩くなど）
2. テストルートの活用 -> クロージャー

**use** Carbon\Carbon;  
  
Route::*get*('test', **function** (){  
 var\_dump(Carbon::*parse*('2017-05-15')->lt(Carbon::*now*()));  
});

クロージャー(英語: closure)

関数閉包はプログラミング言語における関数オブジェクトの一種。 いくつかの言語ではラムダ式や無名関数で実現している。 引数以外の変数を実行時の環境ではなく、自身が定義された環境（静的スコープ）において解決することを特徴とする。 - Wikipedia

1. Httpリクエストメソッド（重要）
   1. post -> 追加用
   2. put/patch -> 更新用
   3. delete -> 削除用
   4. get -> 検索用
   5. 上記ルールを必ず守ってください
   6. getとpostを同じURLにする（オススメ）
   7. put/patchとdeleteを同じURLにする（オススメ）

手順２：コントローラ/メソッド作成

機能を実現するため、どんな部品（クラス）が必要か考える。

習慣として：

データ関連　 -> [リポジトリ]::class

メール関連　 -> [メール]::class (自動生成可能)

認可（権限）関連-> [Policy]::class (自動生成可能)

定数（define） -> [定数]::class

リクエスト　　　-> [Request]::class (自動生成可能)

入力チェック -> [Request]::class (自動生成可能)

など

その他（高度な使い方）：

イベンドシステムの活用

静的クラス

抽象クラス、インタフェース、マジックメソッドなどの活用

エラー/成功の処理

注意点：

1. 名前空間
2. use書き忘れ
3. コンストラクタによりDI
4. 引数によるDI
5. ルート定義によるDI
6. PHPDocを書く
7. 曖昧な書き方はしない（型など注意）

*\** ***@param*** *User $user  
 \** ***@return*** *int  
 \*/***public static function** UserUnfinishedTaskCount(User $user)  
{  
 **return** (int)($user->Tasks()  
 ->Where('is\_finish', Definer::*TASK\_UNFINISHED*)  
 ->count());  
}

手順３：View作成

注意点：

継承、includeの活用

Blade文法

{{ }}　と　{!! !!}の使い方

Modelの活用

三項演算子の活用

<div class="form-group{{ $errors->has('type\_name') ? ' has-error' : '' }}">

Null 合体演算子（PHP７）の活用

{{ $foo ?? $bar ?? $baz ?? $qux }}

Formの書き方（重要）：

Httpリクエストメソッド使い分け

1. POST -> 追加用
2. PUT/PATCH -> 更新用
3. DELETE -> 削除用
4. GET -> 出来るだけformに使用しない

{{ csrf\_field() }}を必ず書く

PUT を使用する場合 -> {{ method\_field('PUT') }}

PATCH を使用する場合 -> {{ method\_field('PATCH') }}

DELETE を使用する場合 -> {{ method\_field('DELETE') }}

<form action="{{ url("/project/$project->id/wiki-index/$type->id") }}" method="POST" style="display: inline;">  
 {{ method\_field('PATCH') }}  
 {{ csrf\_field() }}

手順４：テストケース作成

ルートテストケース作成：

*/\*\*  
 \* プライベートルートとしてアクセスできる.  
 \*  
 \** ***@test*** *\*/***public function** it\_can\_access\_the\_admin\_user\_page()  
{  
 $response = $this->actingAs($this->user)->call('GET', '/admin/user');  
 $this->assertEquals(200, $response->status());  
 $this->assertViewHasAll(['mode', 'users']);  
}

クラス/メソッドテストケース作成：

*/\*\*  
 \* SuperAdminユーザモデル.  
 \*  
 \** ***@var*** *App\User  
 \*/***private** $admin;  
  
*/\*\*  
 \* ユーザモデル.  
 \*  
 \** ***@var*** *App\User  
 \*/***private** $user;  
  
*/\*\*  
 \* テストユーザ作成.  
 \*  
 \** ***@before*** *\** ***@return*** *void  
 \*/***public function** createTestData()  
{  
 $this->admin = \App\User::*find*(1);  
 $this->user = factory(App\User::*class*)->create();  
}  
  
*/\*\*  
 \* スーパー管理人認可失敗.  
 \*  
 \** ***@test*** *\*/***public function** testAdminPolicyFalse()  
{  
 $this->assertFalse(AdminPolicy::*is\_super\_admin*($this->user));  
}  
  
*/\*\*  
 \* スーパー管理人認可成功.  
 \*  
 \** ***@test*** *\*/***public function** testAdminPolicyTrue()  
{  
 $this->assertTrue(AdminPolicy::*is\_super\_admin*($this->admin));  
}

データベーステストケース

*/\*\*  
 \* ユーザテーブルにスーパー管理人のレコードがある.  
 \*  
 \** ***@test*** *\*/***public function** testUserTableHasSuperAdmin()  
{  
 $this->seeInDatabase('users', [  
 'id' => 1,  
 'name' => 'Admin',  
 'email' => 'admin@lindale.tk',  
 ]);  
}

作動確認テストケース作成：

*/\*\*  
 \* ログイン機能確認.*

*\** ***@test*** *\*/***public function** testLoginAction()  
{  
 $this->visit('/login')  
 ->click('English')  
 ->click('English')  
 ->type('admin@lindale.tk', 'email')  
 ->type('123456', 'password')  
 ->press('Login')  
 ->seePageIs('/home');  
}

注意点：

PHPDocに***@test***を書かないとテストケースとして認識されない。