ライフサイクル
マルチログイン

マルチログインの前に

マルチログインの応用例

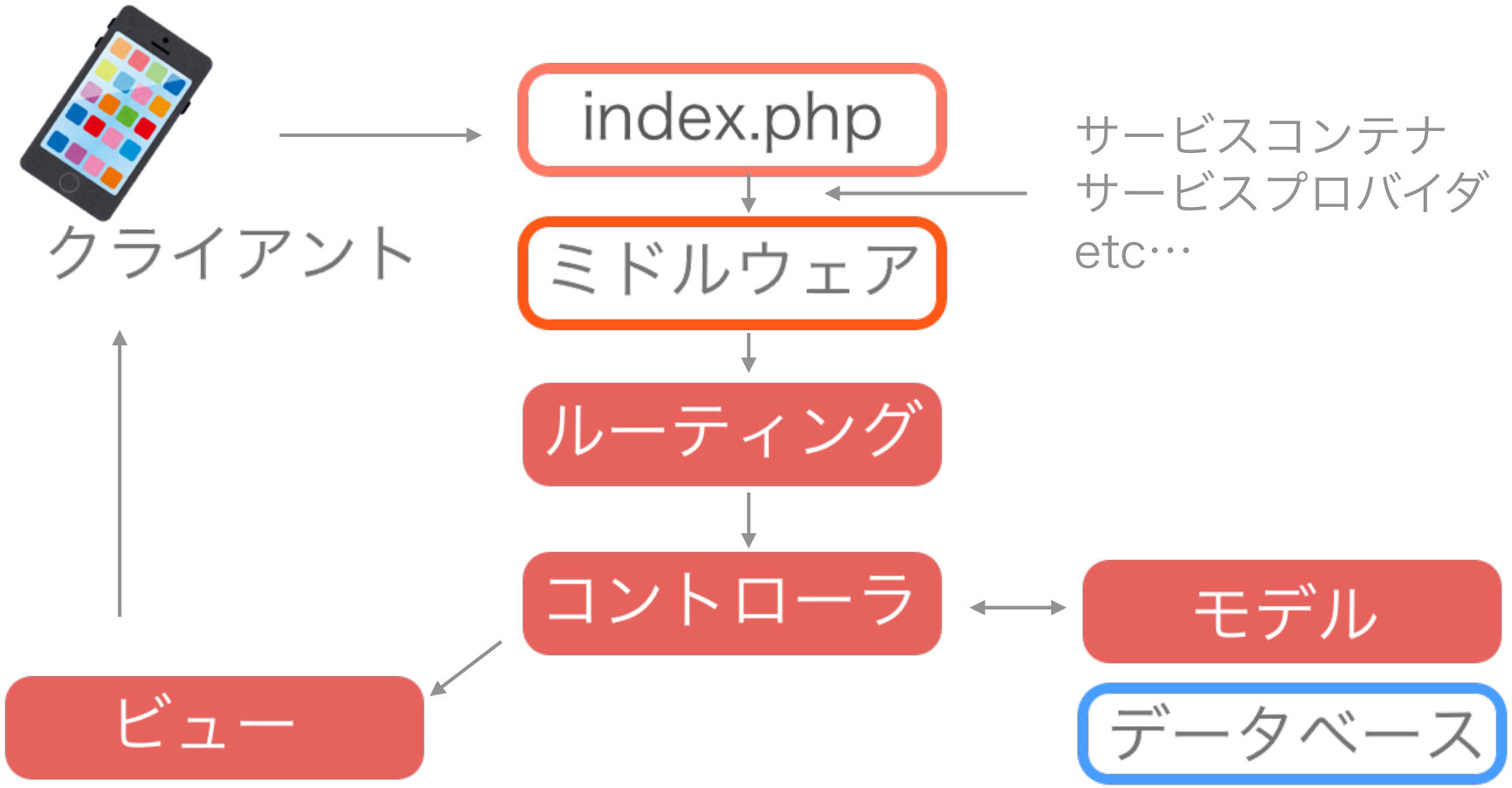
サイトの種類	提供側(販売側)	利用側(購入側)
物販	商品の登録	商品を探す・買う
不動産	物件の登録	物件を探す
求人	求人情報の登録	求人情報を探す
副業	スキルの登録	依頼する
家電修理	エアコンなどの 修理内容を登録	探す・依頼する

マルチログイン 関連ファイル

model、controller、view、route

providers/RouteServiceProvider.php
 ログイン後のURL
config/auth.php Guard設定
 ログイン方法、どのテーブルを使うか
middleware/Authenticate.php
 未認証時のリダイレクト処理
middleware/RedirectIfAuthenticated.php
 ログイン済みユーザーのリダイレクト

ブラウザに表示されるまでの流れ



MVCモデル: Model, View, Controller

ライフサイクル

全体の流れ

WEBサーバがpublic/index.phpにリダイレクト

1. autoload読み込み
2. Applicationインスタンス作成(サービスコンテナ)
3. HttpKernelインスタンス作成
4. Requestインスタンス作成
5. HttpKernelがリクエストを処理してResponse取得
6. レスポンス送信
7. terminate()で後片付け

1.autoload

Public/index.php

1.autoload
requireなしで別ファイルのクラスを利用可能

2.サービスコンテナ

Bootstrap/app.php

2. Application(サービスコンテナ)
$app = new Illuminate\Foundation\Application
Applicationインスタンスが 通称サービスコンテナ

サービスプロバイダも読み込まれる(後で解説します)
registerConfiguredProviders(){}

3.Kernel

Bootstrap/app.php
singletonでKernelをサービスコンテナに登録

Public/index.php
3.HttpKernel
$kernel = $app->make(Kernel::class);

4.Requestインスタンス作成

Public/index.php
Request::capture()
 SymfonyRequest::createFromGlobals()
→この中で$_GET, $_POSTなどを使っている

self::createRequestFromFactory()
->この中でRequestクラスをインスタンス化

5. Response取得

/vendor/laravel/framework/src/illuminate/Foundation/Http/Kernel.php

handle()
sendRequestThroughRouter()
$this->bootstrap()で
.envの環境変数の読み込み、各サービスプロバイダの処理

5. Response取得

sendRequestThroughRouter()
抜粋

```
return (new Pipeline($this->app))
->send($request)
->through($this->app->shouldSkipMiddleware() ?
[] : $this->middleware)
->then($this->dispatchToRouter());
```

6. Response送信

```
$response = tap($kernel->handle(
    $request = Request::capture()
))->send();
```

/vendor/symfony/http-foundation/
Response.php
send()の中で
sendHeaders()
sendContent()

7. terminate

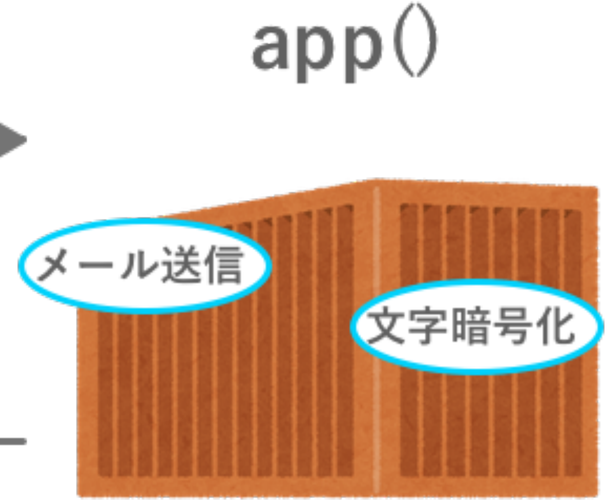
```
$kernel->terminate($request, $response);
```

サービスコンテナ

サービスコンテナ

app()->bind()
app()->singleton()

app()->make()



dd(app()); で中身を確認できる

bindings: array:65
->65のサービスが設定されている

サービスコンテナに登録する

```
app()->bind('lifeCycleTest', function(){
    return 'ライフサイクルテスト';
})
```

引数 (取り出すときの名前, 機能)

Bindings:の数が65->66に増えている

サービスコンテナから取り出す

```
$test = app()->make('lifeCycleTest');
```

他の書き方

```
$test = app('lifeCycleTest');
$test = resolve('lifeCycleTest');
$test = App::make('lifeCycleTest');
```

依存関係の解決

依存した2つのクラス
それぞれインスタンス化後に実行

```
$message = new Message();
$sample = new Sample($message);
$sample->run();
```

サービスコンテナを使ったパターン

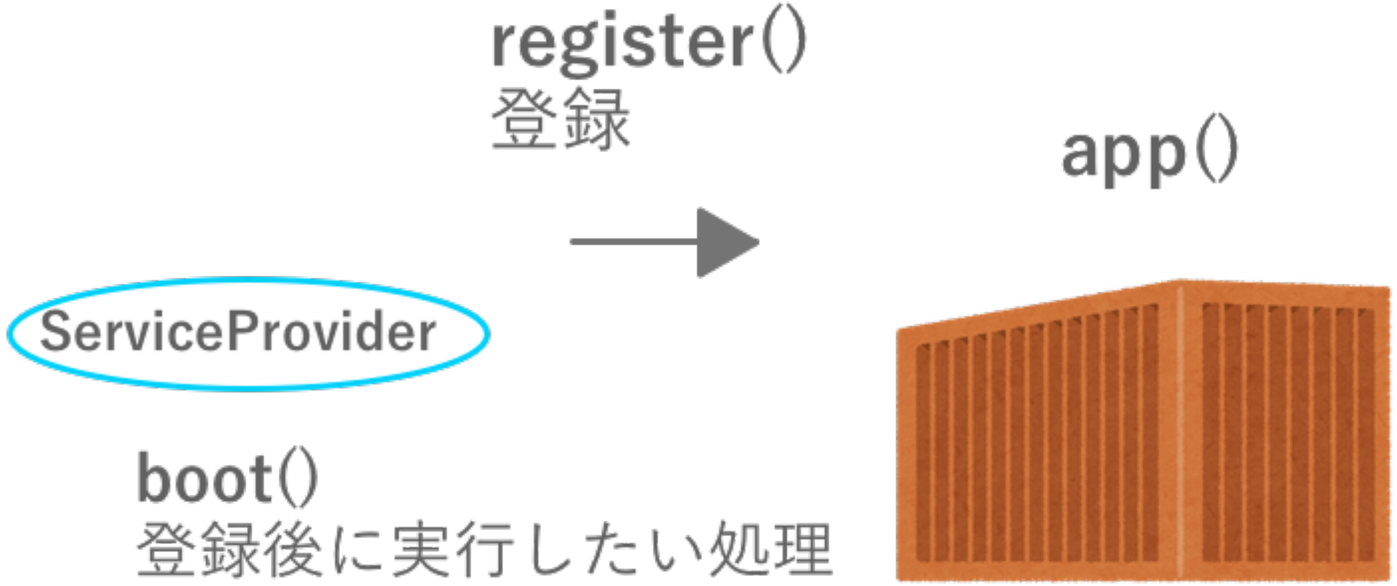
```
app()->bind('sample', Sample::class);
$sample = app()->make('sample');
$sample->run();
```

```
class Sample
{
  public $message;
  public function __construct(Message $message){
    $this->message = $message;
  }
  public function run(){ $this->message->send(); }
}

class Message
{
 public function send(){ echo('メッセージ表示'); }
}
```

サービスプロバイダ

サービスプロバイダ(提供者)



サービスコンテナに
サービスを登録する仕組み

サービスプロバイダの読込箇所

illuminate\Foundation\Application

```
registerConfiguredProviders(){
  $providers = Collection::make($this-
>config[‘app.providers’];
}
```

サービスプロバイダを使ってみる

EncryptionServiceProviderを参考に

使い方

```
$encrypt = app()->make('encrypter');
$password = $encrypt->encrypt('password');
dd($password, $encrypt->decrypt($password));
```

サービスプロバイダの生成

php artisan make:provider SampleServiceProvider
App/Providers配下に生成

```
public function register(){
    サービスを登録するコード
}

public function boot(){
    全サービスプロバイダー読み込み後に
    実行したいコード
}
```

サービスプロバイダの登録

Config/app.phpの $providersに追記

App\Providers\SampleServiceProvider;

ページ読み込み時に
サービスコンテナーに登録される

マルチログイン

マルチログインURL

ユーザー(商品を買う)・・ /
オーナー(商品を登録)・・ /owner/
管理者(オーナーの管理)・ /admin/

マルチログイン手順

1. モデル、マイグレーション作成
2. ルート設定
3. ルートサービスプロバイダ設定
4. ガード設定
5. ミドルウェア設定
6. リクエストクラス設定
7. コントローラー&ブレード作成

1.モデルとマイグレーション生成

php artisan make:model Owner -m
php artisan make:model Admin -m

-mでマイグレーションファイルも生成

app/models フォルダ以下に生成される
Authenticatable を継承

1.マイグレーション設定

create_users_tableの内容を
owner, adminにそれぞれコピーする

php artisan make:migration
create_owner_password_resets
php artisan make:migration
create_admin_password_resets
password_resetsの内容をそれぞれコピーする

2.ルート設定

Userで使ってるのはweb.phpとauth.php

Owner用の routes/owner.php
Admin用の routes/admin.php
をそれぞれ作成

use 文でUser, Owner, Adminフォルダ追記
※フォルダは後ほど生成

3.ルートサービスプロバイダ設定

App/Providers/RouteServiceProvider.php
Owner, AdminそれぞれホームURLを設定

public const OWNER_HOME = ‘/owner/dashboard’

public const ADMIN_HOME = ‘/admin/dashboard’

3.ルートサービスプロバイダ設定

App/Providers/RouteServiceProvider.php
Bootメソッドにuser, owner, adminのURLを追記
Prefixをつける, as()で名前付きルートにもprefixつける

```
Route::prefix(‘/’)->as(‘user.’)
->middleware(‘web’)
->namespace($this->namespace)
->group(base_path(‘routes/web.php’));
```

4. ガード設定

Laravel標準の認証機能
config/auth.php

```
‘guards’ => [
  ‘guard-name’ => [
      ‘driver’ => ‘session’,
      ‘provider’ => ‘users’,
  ],
],
```

```
Route::get(‘test’, function(){})->middleware(‘auth:guard-name’)
```

ルートで auth:ガード名 で認証されたユーザだけにアクセス許可

4. ガード設定

Laravel標準の認証機能
config/auth.php

guards・・今回はsession
Providers・・今回はEloquent(モデル)
passwordReset ・・生成したテーブル名
をそれぞれ設定

参考記事
https://qiita.com/tomoeine/items/40a966bf3801633cf90f

5. Middleware設定

Middleware/Authenticate.php
ユーザが未認証の場合のリダイレクト処理

URLによって条件分岐

```
if (Route::is(‘user.*”)) {
 return route($this->user_route)
}
```

APIマニュアル
https://laravel.com/api/8.x/Illuminate/Routing/Router.html

5. Middleware設定

Middleware/RedirectIfAuthenticated
ログイン済みユーザーがアクセスしてきたら
リダイレクト処理

Auth::guard(self::GUARD_USER)->check()
ガード設定対象のユーザーか

if($request->routeIs(‘user.*’)){
}
受信リクエストが名前付きルートに一致するか

6. リクエストクラス

App/Http/Requests/Auth/LoginRequest.php

ログインフォームに入力された値から
パスワードを比較し、認証する。

User, Owner, Admin 3つのフォームがあるので
routeIs() でルート確認しつつ Auth::guard() を追加

Auth::guard($guard)->attempt(以下略

7. コントローラ&ブレード作成

LaravelBreezeインストール時の
ファイルをコピーして修正

App/Http/Controllers/Auth

resources/views/auth

2.ルート設定の残り
middleware(‘auth’)->middleware(‘auth:owners’)

7-1-1. コントローラ

Authフォルダ毎場所を移動

App\Http\Controllers\Auth →
App\Http\Controllers\User\Auth
このフォルダごとコピーして、
App\Http\Controllers\Owner\Auth
App\Http\Controllers\Admin\Auth
も作成する。

7-1-2. コントローラ

コード編集 (user, owner, admin の情報を追記)

namespaceをフォルダ構成とあわせる

view(‘login’) -> view(‘owner.login’) (フォルダ名.ファイル名)

RouteServiceProvider::HOME ->
RouteServiceProvider::OWNER_HOME

Auth::guard(‘web’)->logout() -> Auth::guard(‘owners’)->logout()

7-1-3. コントローラ

RegisteredUserControllerには
モデル読み込み、
バリデーション設定もあるので注意

use App\Models\User; ->OwnerやAdminに変更

$request->validate([
'email' => ‘required|string|email|max:255|
unique:users', -> unique:owners, adminsに変更

7-2-1. ビュー

views/userフォルダ作成
authフォルダを
views/user/auth フォルダに変更

このフォルダごとコピーし
views/owner/auth
views/admin/auth
を作成

合わせてwelcome, dashboardもコピー

7-2-2. ビュー

RouteServiceProviderで
prefix とasで設定したようにroute内を変更

route(‘login’) -> route(‘user.login’)

@auth にもガードを設定する
@auth(‘users’)

7-2-3. ビュー(レイアウト)

Layoutフォルダ内も編集が必要
navigation.blade.phpをそれぞれ
user-navigation.blade.php
owner-navigation.blade.php
admin-navigation.blade.php とコピー

app.blade.phpに条件追加(auth(ガード)
@if(auth('admin')->user())
  @include('layouts.admin-navigation')
@elseif(auth('owners')->user())

補足

最後にルートの確認

php artisan route:list

ルーティング情報が表示されればOKです。
もしエラーでたらエラー内容確認しつつ
対応する必要あります。