

共通・管理者

設計資料

要件定義～基本設計

企画->要件定義->設計->実装->テスト->リリース

上流工程 5割 実装 2～3割

<https://qiita.com/Saku731/items/741fcf0f40dd989ee4f8>

基本設計



画面設計(UI設計)

AdobeXD、Figma

->今回はtailblockのテンプレートで作成

<https://tailblocks.cc/>

機能設計 ・ ・ 機能名、処理内容

データ設計 ・ ・ テーブル設計、ER図

基本設計リンク

URL設計、テーブル設計、機能設計
(Googleスプレッドシート)

https://docs.google.com/spreadsheets/d/1YIDqTKH2v2-n97kb2GNhWrcMGnJD84JMqTuzD_poMqo/edit?usp=sharing

ER図(draw.io)

<https://drive.google.com/file/d/18sEk5LC-jJum-NU9JKNZibGRVX81aWE1/view?usp=sharing>

アプリ名・ロゴ

アプリ名・ロゴ

アプリ名・・・.envファイル

APP_NAME=Umarche

Config/app.php内で設定される

ロゴ (ロゴ 作成 無料 など検索)

<https://drive.google.com/file/d/1C2ooEDTFPp5cWr2B6gsYQnwKEhM3NmD3/view?usp=sharing>

ロゴ表示



publicフォルダに直接置く ・ ・ 初期ファイル
storageフォルダ ・ ・ フォルダ内画像はgitHubにアップしない

表側(public)から見れるようにリンク
php artisan storage:link
public/storage リンクが生成される

asset() ヘルパ関数でpublic内のファイルを指定

asset("images/logo.png") を
components/application-logo.blade.php に記載

リソースコントローラ

リソース (Restful) コントローラ

CRUD(新規作成、表示、更新、削除)

C(create, store) R(index, show, edit) U(update) D(destroy)

表示・・・GET、DBに保存・・・POST

動詞	URI	アクション	ルート名
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
POST<- PUT/PATCH	/photos/{photo}	update	photos.update
POST<- DELETE	/photos/{photo}	destroy	photos.destroy

URL設計を見ながら



POSTの場合は画面不要(blade不要)

オーナー登録画面・・・GET オーナー登録・・・POST

URL /admin/owners

action index

名前付きルート route('admin.owners.index')

Viewファイル(blade) view('admin.owners.index')

コントローラ Admin\OwnersController@index

リソースコントローラ

生成コマンド

```
php artisan make:controller Admin/  
OwnersController —resource
```

ルート側

```
Route::resource('owners', OwnersController::class)  
->middleware('auth:admin');
```

管理者側 コントローラ

ログイン済みユーザーのみ表示させるため
コンストラクタで下記を設定

```
public function __construct(){  
    $this->middleware('auth:admin');  
}
```

シーダー
(ダミーデータ)

シーダー(ダミーデータ)作成

```
php artisan make:seeder AdminSeeder  
php artisan make:seeder OwnerSeeder
```

database/seeder 直下に生成

シーダー(ダミーデータ)手動設定

DBファサードのinsertで連想配列で追加
パスワードがあればHashファサードも使う

```
DB::tables('owners')->insert([  
    [ 'name' => 'test1', 'email' => 'test1@test.com',  
      Hash::make('password123') ]  
]);
```

DatabaseSeeder.php内で読み込み設定

```
$this->call([  
    AdminSeeder, OwnerSeeder  
]);
```

テーブル再作成&ダミー追加

`php artisan migrate:refresh --seed`
down()を実行後up()を実行

`php artisan migrate:fresh --seed`
全テーブル削除してup()を実行

データを扱う方法

データを扱う方法 比較表

	コレクション Collection	クエリビルダ QueryBuilder	エロクアント Eloquent (モデル)
データ型	Illuminate\Support\Collection	Illuminate\Support\Collection	Illuminate\Database\Eloquent\Collection (Collection を継承)
使用方法	collect(); new Collection;	use Illuminate\Support\Facades\DB; DB::table(テーブル名)->get();	use Models\モデル名; モデル名::all();
関連マニュアル	コレクション	コレクション クエリビルダ	コレクション, クエリビルダ、 エロクアント、エロクアント のコレクション
特徴	配列を拡張	SQLに近い	ORMマッパー
メリット	多数の専用メソッド	SQLを知っているとわかりやすい	簡潔にかける リレーションが強力
デメリット	返り値に複数のパターンあり (stdClass, Collection, モデルCollection)	コードが長くなりがち	覚えることが多い やや遅い

返り値は複数のパターン

Owner::all() // 返り値はEloquentCollection
(それぞれモデルのインスタンス)

DB::table()->get() // 返り値は Collection

DB::table()->first() // 返り値は stdClass

collect(); // 返り値は Collection

Carbon(カーボン)

Carbon

PHPのDateTimeクラスを拡張した
日付ライブラリ
Laravelに標準搭載

公式サイト

<https://carbon.nesbot.com/>

個人ブログ

<https://coinbaby8.com/carbon-laravel.html>

Carbon

エロクアントのtimestampはCarbonインスタンス

```
$eloquents->created_at->diffForHumans()
```

クエリビルダでCarbonを使うなら

```
Carbon\Carbon::parse($query->created_at)  
->diffForHumans()
```



Index, Create

CRUD (Index, Create)

デザインは tailblocksを利用

Buttonタグの場合リンクは

```
<button  
  onclick="location.href='{{ route('admin.owners.create') }}'">
```

Store 保存関連

Store View側

Formタグ、method="post" action=store指定
@csrf 必須

戻るボタンは type="button"をつけておく

inputタグ name="" 属性を
Request \$requestインスタンスで取得
dd(\$request->name);

Store バリデーション1

View

バリデーションで画面読み込み後も
入力した値を保持したい場合

```
<input name="email" value="{{ old('email') }}">
```

Store バリデーション2

Model

\$fillableか\$guardedで設定

```
protected $fillable = [  
    'name',  
    'email',  
    'password',  
];
```


Store バリデーション3

Controller

簡易バリデーション or カスタムリクエスト

```
$request->validate([  
    'name' => 'required|string|max:255',  
    'email' => 'required|string|email|max:255|  
unique:owners',  
    'password' => 'required|string|confirmed|min:8',  
]);
```

Store バリデーション4

Controller
保存処理

```
Owner::create([
    'name' => $request->name,
    'email' => $request->email,
    'password' => Hash::make($request-
>password),
]);

return redirect()->route('admin.owners.index');
```

Store フラッシュメッセージ1

英語だとtoaster

Sessionを使って1度だけ表示

Controller側

```
session()->flash('message', '登録できました。');  
Session::flash('message', '登録できました。');  
redirect()->with('message', '登録できました。');
```

数秒後に消したい場合はJSも必要

Store フラッシュメッセージ2

View側(コンポーネント)
@props(['status' => 'info'])

@php
if(\$status === 'info'){ \$bgColor = 'bg-blue-300';}
if(\$status === 'error'){ \$bgColor = 'bg-red-500';}
@endphp

@if(session('message'))
 <div class="{{ \$bgColor }} w-1/2 mx-auto p-2 text-white">
 {{ session('message') }}
 </div>
@endif

View側 <x-flash-message status="info" />



Edit, Update

Edit 編集

ルート情報確認コマンド

```
php artisan route:list | grep admin.
```

Controller側

```
$owner = Owner::findOrFail($id); //idなければ404画面
```

View側/edit

```
{{ $owner->name }}
```

View側/index 名前付きルート 第2引数にidを指定

```
route('admin.owners.edit', [ 'owner' => $owner->id ]);
```

Update 更新

Controller側

```
$owner = Owner::findOrFail($id);  
$owner->name = $request->name;  
$owner->email = $request->email;  
$owner->password = Hash::make($request->password);  
$owner->save();
```

```
return redirect()->route()->with();
```

View側

```
@method('put')
```




Delete

ソフトデリート

Delete ソフトデリート

論理削除(ソフトデリート)->復元できる (ゴミ箱)

物理削除(デリート)->復元できない

マイグレーション側

```
$table->softDeletes();
```

モデル側

```
use Illuminate\Database\Eloquent\SoftDeletes;
```

モデルのクラス内

```
use SoftDeletes;
```

Delete ソフトデリート

コントローラ側

```
Owner::findOrFail($id)->delete(); //ソフトデリート
```

```
Owner::all(); // ソフトデリートしたものは表示されない
```

```
Owner::onlyTrashed()->get(); //ゴミ箱のみ表示
```

```
Owner::withTrashed()->get(); //ゴミ箱も含め表示
```

```
Owner::onlyTrashed()->restore(); //復元
```

```
Owner::onlyTrashed()->forceDelete(); //完全削除
```

ソフトデリートされているかの確認

```
$owner->trashed()
```

Delete アラート表示(JS)

```
<form id="delete_{{$owner->id}}" method="post"
action="{{ route('admin.owners.destroy', ['owner' => $owner->id]) }}">
  @csrf @method('delete')
  <a href="#" data-id="{{$owner->id }}" onclick="deletePost(this)" >削除</a>

<script>
  function deletePost(e) {
    'use strict';
    if (confirm('本当に削除してもいいですか?')) {
      document.getElementById('delete_' + e.dataset.id).submit();
    }
  }
</script>
```

フラッシュメッセージの編集

コントローラ側 (連想配列に変更
with(['message' => '削除',
 'status' => 'alert']));

ブレード側

<x-flash-message status=session('status')">

コンポーネント側

@if(session('status') === 'alert')

ソフトデリートの 利用例

ソフトデリートの使用例

月額会員・年間会員で更新期限切れ
->延滞料金を払ったら戻せる、など。
->復旧できる手段を残しておく。

View: `admin/expired-owners.blade.php`

注意：データとしては残るので
同じメールアドレスで新規登録できない。
->復旧方法などの案内が別途必要。

期限切れオーナー Route

```
Route::prefix('expired-owners')->
middleware('auth:admin')->group(function(){
Route::get('index', [OwnersController::class,
'expiredOwnerIndex'])->name('expired-
owners.index');
Route::post('destroy/{owner}',
[OwnersController::class, 'expiredOwnerDestroy'])->
name('expired-owners.destroy');
});
```

期限切れオーナー Controller

```
public function expiredOwnerIndex(){
    $expiredOwners = Owner::onlyTrashed()->get();
    return view('admin.expired-owners',
compact('expiredOwners'));
}
```

```
public function expiredOwnerDestroy($id){
    Owner::onlyTrashed()->findOrFail($id)->forceDelete();
    return redirect()->route('admin.expired-owners.index');
}
```

期限切れオーナー View

```
@foreach ($expiredOwners as $owner)
```

```
<form id="delete_{{$owner->id}}" method="post"
action="{ route('admin.expired-owners.destroy', ['owner' =>
$owner->id]) }">
```


```
@csrf
```

```
<td class="px-4 py-3 text-center">
<a href="#" data-id="{{$owner->id }}" onclick="deletePost(this)"
class="text-white bg-red-400 border-0 p-2 focus:outline-none
hover:bg-red-500 rounded">完全に削除</a>
```

```
</td>
</form>
```

ページネーション

ページネーション



オーナーの数が増えてくると
リストが長くなるのでpaginationを設定しておく

Controller側

```
$owners = Owner::select('id', 'name', 'email', 'created_at')  
    ->orderBy('created_at', 'desc')  
    ->paginate(3);
```

View側

```
{{ $owners->links() }}
```

ページネーションの日本語化



vendorフォルダ内ファイルをコピー

```
php artisan vendor:publish --tag=laravel-pagination
```

resources/views/vendor/pagination/

tailwindcss.blade.php

ファイル内を編集



その他

ルート情報の編集

新規登録はしない、ようこそ画面不要

->registration, welcome コメントアウト

OwnersController::class, show は今回使わない

```
Route::resource('owners', OwnersController::class)
```

```
->except(['show']);
```

View側の編集

<x-responsive-nav-link>にもリンク追加

レスポンシブ対応

x方向(横方向)のmargin, paddingに

md: をつける (768px以上, タブレット)