

オーナー  
その1

# オーナーの概要

# オーナーでできる事

---

オーナープロフィール編集 (管理側と同じ)  
店舗情報更新(1オーナー 1店舗)  
画像登録  
商品登録・  
(画像(4枚)、カテゴリ選択(1つ)、在庫設定)

# 基本設計リンク

---

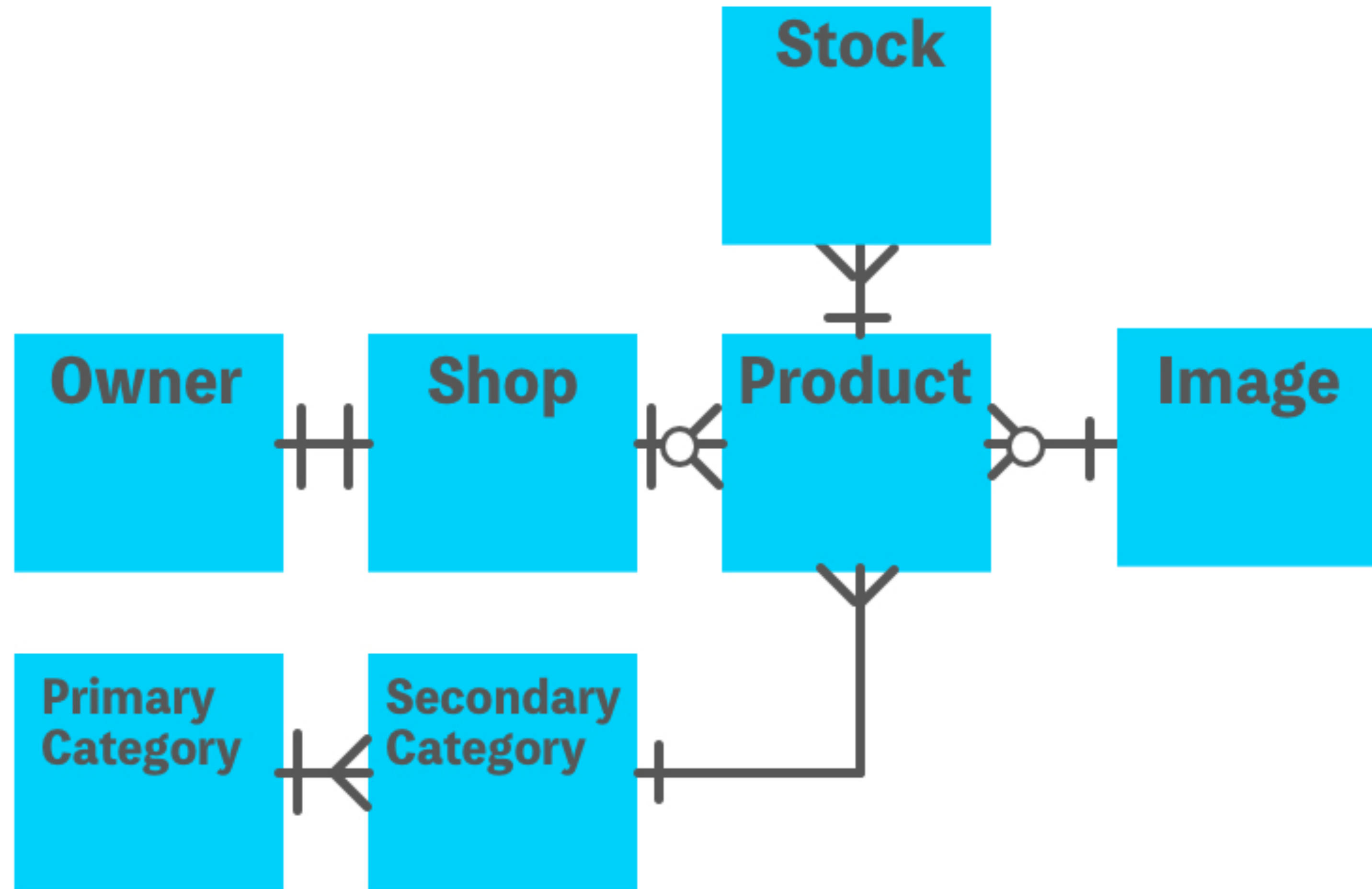
URL設計、テーブル設計、機能設計  
(Googleスプレッドシート)

[https://docs.google.com/spreadsheets/d/1YIDqTKH2v2-n97kb2GNhWrcMGnJD84JMqTuzD\\_poMqo/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1YIDqTKH2v2-n97kb2GNhWrcMGnJD84JMqTuzD_poMqo/edit?usp=sharing)

ER図([draw.io](https://draw.io))

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

# 簡易ER図



# Git branchを新しい順で表示

---

git branch —sort=authordate //古い順  
git branch —sort=-authordate //新しい順



# Shop



# 外部キー制約(FK)

php artisan make:model Shop -m

マイグレーション

//紐づくモデル名\_id

```
$table->foreignId('owner_id')->constrained();  
$table->string('name');  
$table->text('information');  
$table->string('filename');  
$table->boolean('is_selling');
```



# ダミーデータ Seeder

php artisan make:seed ShopSeeder

```
DB::table('shops')->insert([
    [
        'owner_id' => 1,
        'name' => 'お店の名前が入ります。',
        'information' => 'ここにお店の情報が入ります。ここ
    にお店の情報が入ります。ここにお店の情報が入ります。',
        'filename' => '',
        'is_selling' => true
    ]
]);
```

# ダミーデータ Seeder

---

DatabaseSeeder

外部キー制約がある場合は、  
事前に必要なデータ (Owner) を設定する

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

# Eloquent リレーション設定

---

## Owner

```
use App\Models\Shop;  
public function shop()  
{  
    return $this->hasOne(Shop::class);  
}
```

## Shop

```
use App\Models\Owner;  
public function owner()  
{  
    return $this->belongsTo(Owner::class);  
}
```

# Laravel Tinkerで確認

```
php artisan tinker
```

```
App\Models\Owner::find(1)->shop;
```

```
App\Models\Owner::find(1)->shop->name;
```

- ・ ・ Ownerに紐づく Shop情報を取得

```
App\Models\Shop::find(1)->owner;
```

```
App\Models\Shop::find(1)->owner->email;
```

- ・ ・ Shopに紐づく Owner情報を取得

※public functionで設定していますが  
動的プロパティとして()が不要なので注意

# Shopの作成/削除

# Shopの作成

---

Admin/OwnersController@store

外部キー向けにidを取得

```
$owner = Owner::create();
```

```
$owner->id;
```

Shop::create で作成する場合は  
モデル側に \$fillableも必要

# トランザクション

複数のテーブルに保存する際は

トランザクションをかける

無名関数内で親の変数を使うには useが必要

```
DB::transaction(function() use ($request){
```

```
    DB::create($request->name);
```

```
    DB::create($request->owner_id);
```

```
}, 2) // NG時2回試す
```



# 例外 + ログ

トランザクションでエラー時は例外発生  
PHP7 から Throwableで例外取得  
ログは storage/logs 内に保存  
use Throwable;  
use Illuminate\Support\Facades\Log;

```
try{  
    トランザクション処理  
} catch( Throwable $e ){  
    Log::error($e);  
    throw $e;  
}
```

# Shopの削除 カスケード

---

Owner->Shop と  
外部キー制約を設定しているため  
追加設定が必要。

```
$table->foreignId('owner_id')  
    ->constrained()  
    ->onUpdate('cascade')  
    ->onDelete('cascade');
```

# Git branch名 変更

---

git branch -m 旧ブランチ名 新ブランチ名

git branch -m 新ブランチ名

// 現在のブランチを変更

# Shopの 一覧/編集/更新

# Shop 表示までの設定

---

## Route

Index, edit, updateの3つ  
owner.shops.index など

## View

ロゴサイズ調整, owner-navigation

# Shop 表示までの設定

Controller ・ ・ ShopController

\_\_construct で \$this->middleware('auth:owners'):

index メソッド

```
use Illuminate\Support\Facades\Auth;
```

```
$ownerId = Auth::id(); // 認証されているid
```

```
$shops = Shop::where('owner_id', $ownerId)->get();
```

```
// whereは検索条件
```

# Shop ルートパラメータの注意

---

/owner/shops/edit/2/

edit, updateなど URLにパラメータを使う場合  
URLの数値を直接変更すると

他のオーナーのShopが見れてしまう・・・NG

ログイン済みオーナーのShop URLでなければ 404を  
表示



# Shop コントローラミドルウェア

コントラクタ内

```
$this->middleware(function($request, $next){  
    $id = $request->route()->parameter('shop'); //shopのid取得  
    if(!is_null($id)){ // null判定  
        $shopsOwnerId = Shop::findOrFail($id)->owner->id;  
        $shopId = (int)$shopsOwnerId; // キャスト 文字列→数値に型変換  
        $ownerId = Auth::id();  
        if($shopId !== $ownerId){ // 同じでなかったら  
            abort(404); // 404画面表示  
        }  
    }  
    return $next($request);  
});
```

# 404画面をカスタマイズするなら

---

Vendorフォルダ内ファイルは  
更新がかかると上書きされてしまう可能性がある

下記コマンドでresources/views/errorsに  
関連ファイル作成

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

# Shop Index画面

---

初期設定 NO IMAGE画像

<https://drive.google.com/file/d/1fLW38ueg4cRR2W8NWQaRz7u0cOhk5QoH/view?usp=sharing>

無料画像サイト

<https://pixabay.com/ja/>

# Shop Index画面

---

```
@if(empty($shop->filename))  
      
@else  
      
@endif
```

# 画像アップロード

# 画像アップロード



バリデーション->後ほど

画像サイズ(今回は1920px x 1080px (FullHD))

比率は 16:9

->ユーザ側でリサイズしてもらう

->サーバー側でリサイズする

-> Intervention Imageを使う

重複しないファイル名に変更して保存

# 画像アップロード ビュー側

---

ビュー側

```
<form method="post" action=""  
enctype="multipart/form-data">
```

```
<input type="file" accept="image/  
png,image/jpeg,image/jpg">
```



# 画像アップロード コントローラ側

リサイズしないパターン (putFileでファイル名生成)

```
use Illuminate\Support\Facades\Storage;
```

```
public function update(Request $request, $id)
```

```
{
```


```
    $imageFile = $request->image; //一時保存
```

```
    if(!is_null($imageFile) && $imageFile->isValid() ){
```

```
        Storage::putFile('public/shops', $imageFile);
```

```
    }
```

```
}
```



# Intervention Image

# Intervention Image

---

PHP 画像ライブラリ

<http://image.intervention.io/>

(もし無効になっていたら有効化する php.ini)

FileInfo Extension

GD 画像ライブラリ

```
composer require intervention/image
```

# Intervention Image 設定

config/app.php

```
$providers = [  
    Intervention\Image\ImageServiceProvider::class  
];
```

// Imageだとバッティングするので変更

```
$alias = [  
    'InterventionImage' =>  
    Intervention\Image\Facades\Image::class  
];
```

# Intervention Image リサイズ

```
use InterventionImage;
```

```
$resizedImage = InterventionImage::make($imageFile)-  
>resize(1920, 1080)->encode();
```

Storage::putFile はFileオブジェクト想定  
InterventionImageでリサイズすると画像  
(型が変わる)

今回は Storage:put で保存  
(フォルダは作成、ファイル名は指定)

# Intervention Image リサイズ

---

```
$fileName = uniqid(rand().'_');  
$extension = $imageFile->extension();  
$fileNameToStore = $fileName. '.' . $extension;  
  
Storage::put('public/shops/' . $fileNameToStore,  
$resizedImage );
```

# フォーム リクエスト



# フォーム(カスタム)リクエスト 1

```
php artisan make:request UploadImageRequest
```

```
App\Http\Requests\UploadImageRequest.php  
が生成
```

```
public function authorize()  
{ return true; }
```

```
public function rules()  
{  
    return [  
        'image'=>'image|mimes:jpg,jpeg,png|max:2048',  
    ];  
}
```

# フォーム(カスタム)リクエスト 2

---

```
public function messages()
{
    return [
        'image' => '指定されたファイルが画像ではありません。',
        'mines' => '指定された拡張子 (jpg/jpeg/png) ではありません。',
        'max' => 'ファイルサイズは2MB以内にしてください。',
    ];
}
```

# フォーム(カスタム)リクエスト 3

---

コントローラ側

```
use App\Http\Requests\UploadImageRequest;

public update(UploadImageRequest $request, $id)
{

}
```

# サービスへの 切り離し

# サービスへの切り離し

重複を防ぎ、ファットコントローラを防ぐため  
App\Services\ImageService.php ファイルを作成

```
<?php
namespace App\Services;
Use InterventionImage;
Use Illuminate\Support\Facades\Storage;

Class ImageService
{
    public static function upload($imageFile, $folderName)
    {
        省略
        Storage::put('public/' . $folderName . '/' . $fileNameToStore, $resizedImage);
        return $fileNameToStore;
    }
}
```



# Shopの Edit/Update

# Shop Edit残りのフォーム抜粋

店名 <input type="text">{{ \$shop->name}}

店舗情報 <textarea rows="10">{{ \$shop->information}}</textarea>

画像のサムネイル

<div class="w-32">

<x-shop-thumbnail />

</div>

販売中/停止中

<input type="radio" name="is\_selling" value="1" @if(\$shop->is\_selling = true){ checked } @endif>販売中

<input type="radio" name="is\_selling" value="0" @if(\$shop->is\_selling = false){ checked } @endif>停止中



# Shop Update残りのコード抜粋

```
$request->validate([  
    'name' => 'required|string|max:50',  
    'information' => 'required|string|max:1000',  
    'is_selling' => 'required',  
]);
```

# Shop Update残りのコード抜粋

```
$shop = Shop::findOrFail($id)
$shop->name = $request->name;
$shop->information = $request->information;
$shop->is_selling = $request->is_selling;
```

```
If(!is_null($imageFile) && imageFile->isValid())
{ $shop->filename = $fileNameToStore; }
```

```
$shop->save();
```

```
redirect()->route()->with([]); //フラッシュメッセージ
```



Image

# Imageのモデル, マイグレーション

```
php artisan make:model Image -m  
モデル
```

```
$fillable = ['owner_id', 'filename'];
```

```
マイグレーション
```

```
$table->foreignId('owner_id')->constrained()
```

```
->onUpdate('cascade')
```

```
->onDelete('cascade');
```

```
$table->string('filename');
```

```
$table->string('title')->nullable();
```

# Imageのコントローラ

---

```
php artisan make:controller Owner/  
ImageController —resource
```

ルート

```
Route::resource('images',  
ImageController::class)  
->middleware('auth:owners')->except('show');
```

# ImageのIndex

# ImageのIndex その1

コントローラ

constructはShopControllerを参考に

```
public function index()
{
    $images = Image::where('owner_id', Auth::id())
        ->orderBy('updated_at', 'desc') // 降順 (小さくなる)
        ->paginate(20);
}
```

以下略

```
}
```



# ImageのIndex その2

---

ビュー

shops/index.blade.phpを参考に

コンポーネントをまとめるために変更

```
<x-thumbnail 略 type="products" />
```

# ImageのIndex その3

components/thumbnail.blade.php

```
@php
if($type === 'shops'){
    $path = 'storage/shops/';
}
if($type === 'products'){
    $path = 'storage/products/';
}
@endphp

<div>
    @if(empty($filename))
        
    @else
        
    @endif
</div>
```

# ImageのIndex その4

---

ページネーションのリンクを追加  
`admin/owners/index.blade.php`

モデルのリレーションも追加  
`App/Models/Shop.php`を参考に

`App/Models/Owner.php` に `hasMany`  
`App/Models/Image.php` に `belongsTo` を追記



# ImageのCreate, バリデーション Store

# ImageのCreate とバリデーション

Shops/edit.blade.phpを参考

画像の複数アップロード対応

<input type="file" name="files[image]" multiple 略>

フォームリクエストのrulesに下記を追加

App/Http/Requests/UploadImageRequest.php

```
'files.*.image' => 'required|image|mimes:jpg,jpeg,png|max:2048',
```

# ImageのStore ImageController

ShopController@updateを参考

```
$imageFiles = $request->file('files'); //配列でファイルを取得
if(!is_null($imageFiles)){
    foreach($imageFiles as $imageFile){ // それぞれ処理
        $fileNameToStore = ImageService::upload($imageFile, 'products');
        Image::create([
            'owner_id' => Auth::id(),
            'filename' => $fileNameToStore
        ]);
    }
}
```



# ImageのStore ImageService

```
if(is_array($imageFile)){  
    $file = $imageFile['image']; // 配列なので['key'] で取得  
} else {  
    $file = $imageFile;  
}  
  
$fileName = uniqid(rand().'_');  
$extension = $file->extension();  
$fileNameToStore = $fileName . '.' . $extension;  
$resizedImage = InterventionImage::make($file)->resize(1920,  
1080)->encode();  
Storage::put('public/' . $folderName . '/' . $fileNameToStore,  
$resizedImage );
```





# Imageの Edit, Update


# ImageのEdit, Update

---

ShopController@edit, updateを参考に

shop の箇所は image に変更

リソースコントローラを使っているので  
updateがputメソッド  
-> @method('put') をつける



# Imageの Destroy

# ImageのDestroy (Delete)

admin/OwnersController@destroy と  
admin/owners/index.blade.php を参考に

テーブル情報を削除する前に  
Storageフォルダ内画像ファイルを削除

```
$image = Image::findOrFail($id);  
$filePath = 'public/products/'. $image->filename;  
if(Storage::exists($filePath)){  
    Storage::delete($filePath);  
}
```

削除・リダイレクトは省略



# Imageの ダミーデータ

# Imageのダミーデータ

php artisan make:seed ImageSeeder

画像はリサイズ・リネーム後 storage/productsフォルダに保存

いくつかのファイル名を書き換えつつダミーとして登録  
sample1.jpg ~ sample6.jpg

Storage内ファイルはgitにアップすると消えるので  
public/images内に保存しつつ

README.md に明記しておきます