

オーナー
その2

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

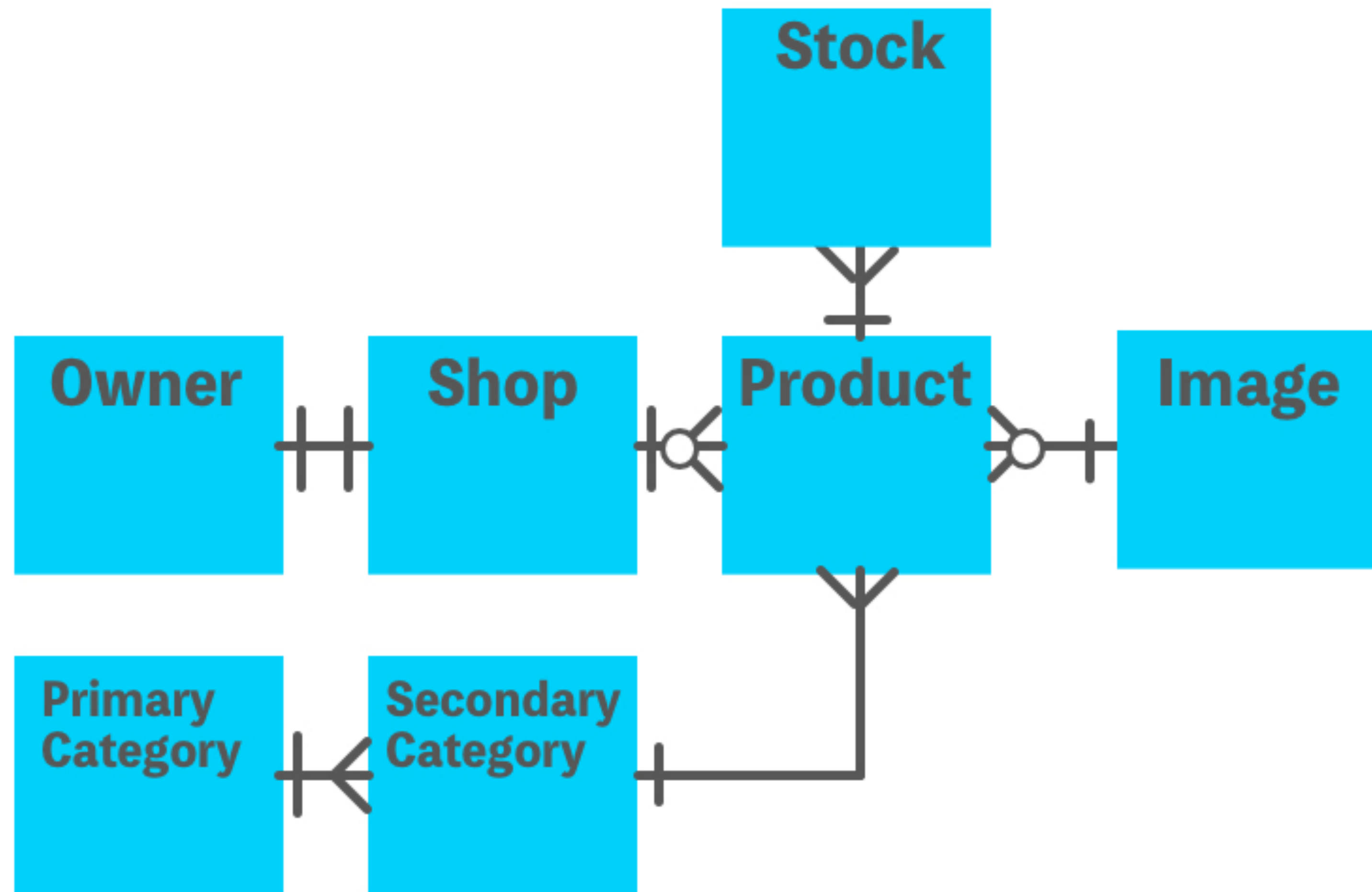
URL設計、テーブル設計、機能設計
(Googleスプレッドシート)

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

ER図(draw.io)

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

簡易ER図





Category

Category モデル

```
php artisan make:model PrimaryCategory -m  
php artisan make:model SecondaryCategory
```

モデル 1対多のリレーション
Primary

```
Public function secondary()  
{  
    return $this->hasMany(SecondaryCategory::class);  
}
```

Secondaryからは belongsTo

Category マイグレーション

1つのファイルに2つ記載

ファイル名を create_categories_table.phpに変更

クラス名も CreateCategoriesTable

```
Schema::create('primary_categories', function (Blueprint $table) {  
    略  
})
```

```
Schema::create('secondary_categories', function (Blueprint $table) {  
    略  
})
```

Downメソッドは先にsecondaryを削除する
(primaryを先に書くと migrate:refresh 時に外部キーエラー発生)

Category シーダー

```
php artisan make:seed CategorySeeder  
DB::table('primary_categories')->insert([  
    [ 略 ]  
]);
```

```
DB::table('secondary_categories')->insert([  
    [ 略 ]  
]);
```

DatabaseSeeder.php にも追記

Category シーダー

php artisan migrate:fresh --seed で投入
念の為 php artisan tinker で確認

```
$category = new App\Models\PrimaryCategory;  
$category::findOrFail(1)->secondary;
```

カテゴリーが表示されれば
リレーションOK



Product (初期設定)

Product モデル



```
php artisan make:model Product -m
```

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

```
Shop.php ・ ・ hasMany(Product::class)
```

```
Product.php ・ ・ belongsTo(Shop::class)
```

```
Product.php ・ ・ belongsTo(Image::class)->後ほど
```

```
Product.php ・ ・ belongsTo(SecondaryCategory::class)->後ほど
```

Product マイグレーション

外部キー制約

親を削除するか, 親を削除したときに合わせて削除するか

テーブル名(shops 複数形)とカラム名(shop_id 単数形_id)が一致するか

Nullを許容するか

```
$table->foreignId('shop_id') // cascadeあり
```

```
$table->foreignId('secondary_category_id') // cascadeなし
```

```
$table->foreignId('image1')->nullable()->constrained('images');
```

```
// null許可、カラム名と違うのでテーブル名を指定
```

Product シーダー

リレーションができているか確認したいので
FKのダミーデータを先に作成

```
php artisan make:seed ProductSeeder
```

```
DB::table('products')->insert([
    [
        'shop_id' => 1,
        'secondary_category_id' => 1,
        'image1' => 1,
    ],
    [
        'shop_id' => 1,
        'secondary_category_id' => 2,
        'image1' => 2,
    ]
]);
```

Product リレーション

メソッド名をモデル名から変える場合は第2引数必要
(カラム名と同じメソッドは指定できないので名称変更)
第2引数で_id が見つからない場合は 第3引数で指定必要

Product.php

```
public function category(){  
    belongsTo(SecondaryCategory::class, 'secondary_category_id');  
}
```

```
public function imageFirst(){  
    belongsTo(Image::class, 'image1', 'id');  
}
```


Product リレーション

php artisan tinker で確認

```
$product = new App\Models\Product;
```

```
$product::findOrFail(1)->category
```

```
$product::findOrFail(1)->imageFirst
```

```
$product::findOrFail(1)->shop->owner->id
```

それぞれリレーション先が取得できていればOK



Product Index

Product Index

コンストラクタを設定(ImageControllerなどを参考に)

```
Product::findOrFail($id)->shop->owner->id;
```

```
$products = Owner::findOrFail(Auth::id())->shop->product; //後ほど修正します
```

owner/images/index.blade.php を参考

\$images の箇所を \$products に変更



Eager Loading

Eager(積極的) Loading

N + 1問題の対策

リレーション先のリレーション情報を取得
withメソッド、リレーションをドットでつなぐ

```
$ownerInfo = Owner::with('shop.product.imageFirst')  
->where('id', Auth::id())->get();
```

```
foreach($ownerInfo as $owner)  
    foreach($owner->shop->product as $product)  
    {  
        dd($product->imageFirst->filename);  
    }  
endforeach
```



Stock

Stock

在庫管理・履歴用のテーブル
マスターテーブル(参照用), トランザクションテーブル(処理用)

php artisan make:model Stock -m

Product モデルから hasMany

Stockモデル protected \$table = 't_stocks';

マイグレーション

Schema::create('t_stocks', \$table->tinyInteger('type');

1・・・入庫 2・・・出庫

Upメソッド、downメソッドともにテーブル名変更

Stock

ダミーデータ

```
php artisan make:seed StockSeeder
```

合計を計算

```
php artisan tinker で確認
```

```
$product = new App\Models\Product  
$product::find(1)->stock->sum('quantity')
```



Product Create

Product Create



先に外部キーの項目を設定

```
ProductController#create
```

```
$shops = Shop::where('owner_id', Auth::id())->select('id', 'name')->get();
```

```
$images = Image::where('owner_id', Auth::id())->select('id', 'title',  
'filename')->orderBy('updated_at', 'desc')->get();
```

```
$categories = PrimaryCategory::with('secondary')->get();
```

```
return view('owner.products.create', compact('shops', 'images',  
'categories'));
```

Categoryのビュー側

owner/shops/edit.blade.php を参考

```
<select name="category">
  @foreach($categories as $category)
    <optgroup label="{{ $category->name }}">
      @foreach($category->secondary as $secondary)
        <option value="{{ $secondary->id }}" >
          {{ $secondary->name }}
        </option>
      @endforeach
    @endforeach
  @endforeach
</select>
```



Micromodal.js

Micromodal.js 画像選択

シンプル・軽量・VanillaJS

<https://micromodal.vercel.app/>

`npm install micromodal —save`

HTML/CSSのサンプル

<https://gist.github.com/ghosh/4f94cf497d7090359a5c9f81caf60699>

Micromodal.js JS/CSS設定

resources/js/bootstrap.js
に追記

```
import MicroModal from 'micromodal'; // es6 module
MicroModal.init({
  disableScroll: true
});
```

resources/css/micromodal.cssを作成し
resources/css/app.css から@import (@importはpostcssの機能)

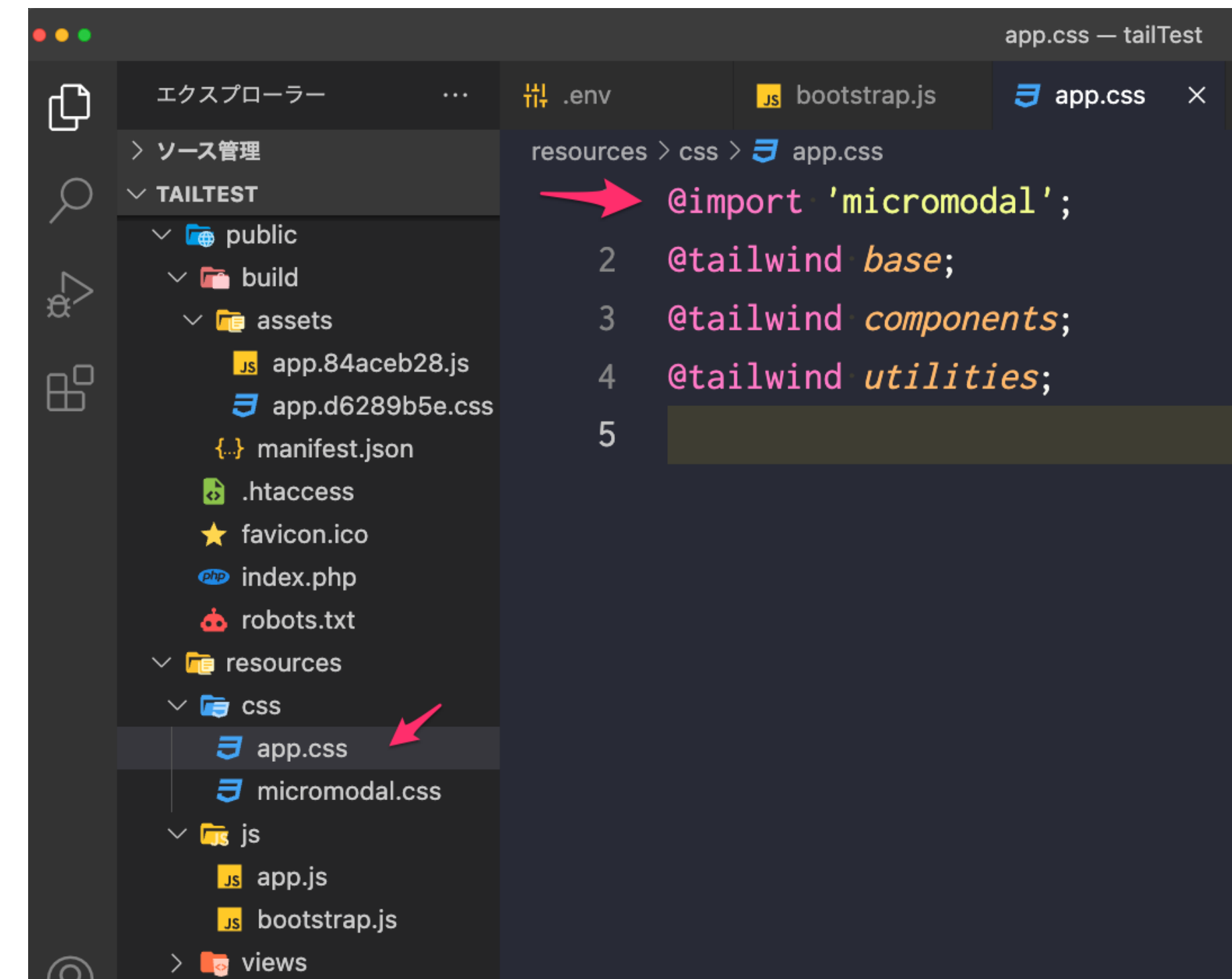
max-width: 500px; -> max-width: 1200px; に変更

npm run dev でビルド npm run prod で本番用にビルド(CSSをpurge)

Micromodal.js JS/CSS設定(追記)

resources/css/app.cssについて、
動画収録当時はtailwindcss ver2 だったため、
@import表記になっておりますが、
2022年1月以降 tailwindcss ver3 になり、@tailwind表記に変わっております。

micromodalを加えるために
@import 'micromodal' が必要なのですが、
@importはファイル冒頭に
追記する必要があるため、
1行目に追記するようにお願いいたします。



Micromodal.js HTML設定

HTML側

```
<x-select-image name="image1" />
```

コンポーネント側

submitを消すため
buttonタグは全て type="button" とつける

Micromodal.js HTML設定 1

@php // image1～4を変数で設定する

```
if($name === 'image1'){ $modal = 'modal-1'; }
```

@endphp

id="modal-1" となっている箇所を \$modal に置き換える

閉じる・ファイルを選択するに変更 Continueボタンを削除

images/index.blade.php の画像表示部分(foreach)をコピー

x-thumbnail をimgタグに変更 (次ページ)

Micromodal.js HTML設定 2

画像をクリックしたら画像を選択しつつモーダルを閉じる
JSで操作できるように共通のCSSと個別のidや属性をつける
(data-○○ とつけると、JSでe.target.dataset.○○ で取得できる)
PHPの変数をJSに渡す方法の一つ

```
<img class="image" data-id="{{ $name }}_{{ $image->id }}"  
      data-file="{{ $image->filename }}"  
      data-path="{{ asset('storage/products/') }}"  
      data-modal="{{ $modal }}"  
      src="{{ asset('storage/products/' . $image->filename) }}" >
```

Micromodal.js HTML設定 3

プレビューエリアとinputタグ(hidden)

```
<div class="flex justify-around items-center mb-4">  
  <a>開くボタン</a>  
  <div class="w-1/4">  
    <img id="{{ $name }}_thumbnail" src="">  
  </div>  
</div>  
<input id="{{ $name }}_hidden" type="hidden"  
name="{{ $name }}" value="">
```

Micromodal.js JS設定

'use strict'

const images = document.querySelectorAll('.image') //全てのimageタグを取得

images.forEach(image => { // 1つずつ繰り返す

image.addEventListener('click', function(e){ // クリックしたら

const imageName = e.target.dataset.id.substr(0, 6) //data-idの6文字

const imageld = e.target.dataset.id.replace(imageName + '_', '') // 6文字カット

const imageFile = e.target.dataset.file

const imagePath = e.target.dataset.path

const modal = e.target.dataset.modal

// サムネイルと input type=hiddenのvalueに設定

document.getElementById(imageName + '_thumbnail').src = imagePath + '/' + imageFile

document.getElementById(imageName + '_hidden').value = imageld

MicroModal.close(modal); //モーダルを閉じる

})
})

Micromodal.js 暫定対応


現象:

image1～3を選んだ後にimage4を選ぶと
以前に選んだimage1～3のモーダルが開く

暫定対応:

image5を作成
(Controller側ではimage4まで保存対象)

->対策判明次第、追収録いたします



Product 残りのCreate/ Store

Product Create

マイグレーション

```
public function up(){  
    $table->unsignedInteger('price');  
    $table->integer('sort_order')->nullable();  
}
```

DatabaseSeeder内で

ProductSeederとStockSeederは一旦コメントアウト
ビュー側

Product Store その1

モデル \$fillable を設定 (Stockにも)

バリデーション

nullable null可

外部キーが存在するか

exists:shops,id

exists:secondary_categories,id

exists:images,id

Product Store



```
try{
  DB::transaction(function() use ($request){
    $product = Product::create([
      'name' => $request->name,
      略
    ]);
    Stock::create([
      'product_id' => $product->id) // Productで作成したidを設定
      略]);
  , 2)
} catch( Throwable $e ){
  Log::error($e);
  throw $e;
}
```

Product リダイレクト後のindex

image1の画像がない場合にうまく表示されない

```
<x-thumbnail :filename="$product-  
>imageFirst->filename"
```

Null合体演算子で null判定

```
<x-thumbnail filename="{{ $product-  
>imageFirst->filename ?? '' }}" >
```



Product Edit/Update

Product Edit その1

コントローラ

```
$product = Product::findOrFail($id);  
$quantity = Stock::where('product_id', $product->id)  
->sum('quantity');
```

モデル imageのリレーション追加

```
public function imageSecond  
public function imageThird  
public function imageFourth
```


Product Edit その2

ビュー側 現在選択している項目にselectedをつける

```
<option value="{{ $secondary->id }}" @if($secondary->id === $product->secondary_category_id) selected @endif >
```

現在選択している項目をコンポーネントに渡す

```
<x-select-image :images="$images"  
currentId="{{ $product->image1 }}"  
currentImage="{{ $product->imageFirst->filename ??  
" }}" name="image1" />
```

Product Edit その3

ビュー/コンポーネント

新規登録時は値がないので 値の有無を確認

```
$cImage = $currentImage ?? " ;
```

```
$cId = $currentId ?? " ;
```

```

</div>
</div>
<input id="{{ $name }}_hidden" type="hidden" name="{{ $name }}"
value="{{ $cId }}">
```

Product Update その1

Products/edit.blade.php に @method('put') 追加 (リソースコントローラのため)

バリデーション項目が多いのでフォームリクエストを作成し分離
php artisan make:request ProductRequest

バリデーション

is_sellingに boolean

quantityに between:0,99 を追加

(products/create.blade.php にも spanタグを追記(0~99の範囲で))

```
use App\Http\Requests\ProductRequest;
```

```
public function store()とupdate()の引数に設定(Requestを置換)
```

Product Update その2

画面表示後に在庫数が変わっている可能性がある
(Edit～updateの間でユーザーが購入した場合など)
在庫が同じか確認し違っていたらeditに戻す (楽観的ロックに近い)
products/edit.blade.php 側にフラッシュメッセージ追加

```
if($request->current_quantity !== $quantity)
{
    $id = $request->route()->parameter('product');
    return redirect->route('owner.products.edit', ['product' => $id])
};
else
{
    処理
}
```

Product Update その3

ProductとStock同時更新するため
トランザクションをかけておく
ProductController@store を参考

```
DB::transaction(function() use ($request, $product){  
    $product->shop_id = $request->shop_id;  
略  
    $product->save(); // 保存処理
```

Product Update その4

在庫削減の場合は
マイナスの数値を入れるため 判定を追加

```
if($request->type === '1')  
    { $newQuantity = $request->quantity; }  
if($request->type === '2')  
    { $newQuantity = $request->quantity * -1; }
```

```
Stock::create([  
    'product_id' => $product->id,  
    'quantity' => $newQuantity,  
    'type' => $request->type,  
]);
```


定数 Constant

定数 Constant

マジックナンバー回避のため 定数クラスを作成
App/Constants/Common.php

```
<?php
```

```
namespace App\Constants;
```

```
Class Common
```

```
{
```

```
    const PRODUCT_ADD = '1';
```

```
    const PRODUCT_REDUCE = '2';
```

```
    const PRODUCT_LIST = [
```

```
        'add' => self::PRODUCT_ADD,
```

```
        'reduce' => self::PRODUCT_REDUCE
```

```
    ];
```

```
}
```

定数 Constant

Config/app.php のaliasesに追記

‘Constant’ =>

App\Constants\Common::class,

使用時は

```
\Constant::PRODUCT_LIST['add'];
```

```
\Constant::PRODUCT_LIST['reduce'];
```



Product Destroy

Product Destroy

コントローラ

ImageController@destroy をコピーして調整

ビュー

images/edit.blade.php をコピーして調整



Image Destroy の補足

Imageを削除する場合

Productで選択しているImageを
削除しようとする外部キーエラー発生

画像を使っているか確認して

対策1. Product側で画像の選択を外してとメッセージを出す

対策2. Productのimage1～image4をnullに変更

今回は対策2で対応


Imageを削除する場合

削除したい画像をProductで使っているかの確認

```
$imageInProducts = Product::where('image1', $image->id)
->orWhere('image2', $image->id)
->orWhere('image3', $image->id)
->orWhere('image4', $image->id)
->get();
```

使っていたらimage1～image4をチェックして nullに変更

```
if($imageInProducts){
    $imageInProducts->each(function($product) use($image){
        if($product->image1 === $image->id){
            $product->image1 = null;
            $product->save();
        }
    })
}
```

オーナー

その他の対応

オーナー その他

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

ログアウト後のリダイレクト修正
AuthenticatedSessionController@destroy

```
return redirect ('/owner/login');
```