

Userができること:

database/seeder/UserSeeder.php:

```
use Illuminate\Support\Facades\DB;
```

```
use Illuminate\Support\Facades\Hash;
```

database/seeder/UserSeeder.phpのrunメソッド:

```
DB::table("users")->insert([
```

```
"name" => "test",
```

```
"email" => "test@test.com",
```

```
"password" => Hash::make("Password123"),
```

```
"created_at" => "2023/12/05 11:11:11",
```

```
"updated_at" => "2023/12/06 11:11:11",
```

```
]);
```

商品一覧:



たいらのエンジニアノート



【Laravel】Routeのprefixについて解説します

こんにちは。たいら(@tairaengineer2)です。転職を繰り返し現在4社経験している、10年目エンジニアです。この記事では、PHPのフレ...

routes/web.php:

```
Route::middleware("auth:users")->group(function(){
```

```
Route::get("/",[ItemController::class,"index"])->name("items.index");
```

```
});
```

```
use App\Http\Controllers\User\ItemController;
```

app/Http/Controller/User/ItemController:

```
public function index(){
```

```
return view('user.index');
```

```
}
```

Viewの調整1:

views/user/index:

```
<div class="flex flex-wrap">
```

```
@foreach ($products as $product)
```

```
<div class="w-1/4 p-2 md:p-2">
```

```
<a href ="">
```

```
<div class="border rounded-md p-2 md:p-4">
```

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

```
type="products" /> <div class="text-xl"> {{ $product->name }} </div>
```

```
</div></a></div> @endforeach </div>
```

app/Http/Controllers/User/ItemController.php:

```
use App\Models\Product;
```

Faker&Factory:

```
php artisan make:factory ProductFactory --model=Product
```

```
php artisan make:factory StockFactory --model=Stock
```



database/factories/ProductFactory:

```
'name' => $this->faker->name,
```

```
'information' => $this->faker->realText,
```

```
'price' => $this->faker->numberBetween (10, 100000),
```

```
'is_selling' => $this->faker->numberBetween (0,1),
```

```
'sort_order' => $this->faker->randomNumber,
```

```
'shop_id' => $this->faker->numberBetween (1,2),
```

```
'secondary_category_id' => $this->faker->numberBetween (1,6),
```

```
'image1' => $this->faker->numberBetween (1,6),
```

```
'image2' => $this->faker->numberBetween(1,6),
```

```
'image3' => $this->faker->numberBetween (1,6),
```

```
'image4' => $this->faker->numberBetween(1,6),
```

database/factories/StockFactory:

```
use App\Models\Product;
```

```
"Product_id" => Product::factory(),
```

```
"type" => $this->faker->numberBetween(1,2),
```

```
"quantity" => $this->faker->randomNumber,
```

database/Seeders/DatabaseSeeder.php:

```
Product::factory(100)->create();
```

```
Stock::factory(100)->create();
```


Viewの調整2:

 tailblocks.cc



Tailblocks — Ready-to-use Tailwind CSS blocks

Ready-to-use Tailwind CSS blocks

SQL: 在庫が1以上のものを取得

```
SELECT product_id,sum(quantity)as quantity FROM t_stocks GROUP BY product_id HAVING quantity >= 1
```

ItemController:

```
use Illuminate\Support\Facades\DB;
```

```
//table('t_stocks')でテーブル名を取得する
```

```
// DB::rawでSQLをそのままファイルに記述できる。(クエリービルダー)
```

```
// ->groupBy('product_id')->having('quantity','>',1);でproduct_id毎の合計在庫が1個以上あるかを指定
```

```
$stocks = DB::table('t_stocks')
```

```
->select('product_id',
```

```
DB::raw('sum(quantity)as quantity'))
```

```
->groupBy('product_id')
```

```
->having('quantity','>',1);
```

```
dd($stocks);
```

```
//->joinSub($stocks,'stock', function ($join) {で $stocksをstockに置き換えて、置き換えたproduct_idとproductsテーブルのproducts.idを合体！これでproductsのテーブルとstockのテーブルが紐付いた。
```

```
//さらに->join ('shops', 'products.shop_id', '=', 'shops.id')で productsのテーブルとshopsのテーブルを紐付かせる
```

```
// ->where('shops.is_selling', true)->where('products.is_selling', true)で 両方販売中の物を取得
```

```
$products = DB::table('products')
```

```
->joinSub($stocks,'stock', function ($join) {
```

```
$join->on ('products.id', '=', 'stock.product_id');
```

```
})
```

```
->join ('shops', 'products.shop_id', '=', 'shops.id')
```

```
->where('shops.is_selling', true)
```

```
->where('products.is_selling', true)
```

```
->get();
```

Eloquentからクエリビルダで変更してるので。。:

今回使用したshops,productsテーブルにはnameやinformationが同じ名前で読み取れないなのでproductsテーブルとsecondary_categoriesテーブルを結合↓

```
->join('secondary_categories','products.secondary_category_id','=','secondary_categories.id')
```

```
->join('images as image1','products.image1','=','image1.id')
```

```
->join('images as image2','products.image2','=','image2.id')
```

```
->join('images as image3','products image3','=','image3.id')
```

```
->join('images as image4','products image4','=','image4.id')
```

imagesテーブルをimage1としてエイリアスして結合しています。結合の基準はproductsテーブルのimage1カラムとimagesテーブル（ここではimage1として参照）のidカラムです。この結合により、製品に関連付けられた最初の画像についての情報を取得できます。

```
->select('products.id as id', 'products.name as name', 'products. price','products.sort_order as sort_order', 'products.information','secondary_categories.name as category','image1.filename as filename') ->get();
```

商品の詳細:

routes/web.php:

```
Route::get("show/{item}",[ItemController::class,"show"]->name("items.show"));
```

User/ItemController:

```
public function show($id){
```

```
$product = Product::findOrFail($id);
```

```
return view('user.show',compact('product'));} 
```

User/ItemController:

```
public function __construct()
```

```
{//ユーザーかどうかの確認
```

```
$this->middleware("auth:users");}
```

resources/views/user/index:

```
{{ route('user.items.show', ['item' => $product->id]) }}
```

商品詳細レイアウト:

※ここは個人のレイアウト自由で!

```
<div class="flex md:flex md:justify-around">
```

```
<div class="md:w-1/2">
```

```
<x-thumbnail filename="{{ $product->imageFirst->filename ?? " }}" type="products" />
```

```
</div>
```

```
<div class="md:w-1/2 ml-4">
```

```
<h2 class="mb-4 text-sm title-font text-gray-500 tracking-widest">{{ $product->category->name }}</h2>
```

```
<h1 class="mb-4 text-gray-900 text-3xl title-font font-medium">{{ $product->name }}</h1>
```

```
<p class="mb-4 leading-relaxed">{{ $product->information }}</p>
```

```
<div class="flex justify-around items-center">
```

```
<div>
```

```
<span class="title-font font-medium text-2xl text-gray-900">{{ number_format($product->price) }}
```

```
</span><span class="text-sm text-gray-700">円(税込)</span>
```

```
</div>
```

```
<div class="flex items-center">
```

```
<span class="mr-3">数量</span>
```

```
<div class="relative">
```

```
<select
```

```
class="rounded border appearance-none border-gray-300 py-2 focus:outline-none focus:ring-2
```

```
focus:ring-indigo-200 focus:border-indigo-500 text-base pl-3 pr-10">
```

```
<option>SM</option>
```

```
<option>M</option>
```

```
<option>L</option>
```

```
<option>XL</option>
```

```
</select>
```

```
</div>
```

```
</div>
```

```
<button
```

```
class="flex ml-auto text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600
```

```
rounded">カートに入れる</button>
```

```
</div>
```

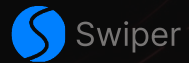
```
</div>
```

```
</div>
```

Swiper.js:



Swiper
The Most Modern
Mobile Touch Slider



Swiper



Swiper - The Most Modern Mobile Touch Slider

The Most Modern Mobile Touch Slider

```
npm install swiper@6.7.0
```


Swiper.jsでスライド画像埋め込み:

```
@if($product->imageFirst->filename !== null)
```

```

```

```
@else
```

```
<img src="">
```

```
@endif
```

shop情報1:

README.md:

shopの画像を表示する場合はstorage/app/public/shopsフォルダを作成し画像を保存してください。

views/user/show:

```
<div class="border-t border-gray-400 my-8"></div>
```

```
<div class="mb-4 mt-4 text-center">この商品を販売してるショップ</div>
```

```
<div class="mb-4 text-center">{{ $product->shop->name }}</div>
```

```
<div class="mb-4 text-center">
```

```
@if ($product->shop->filename !== null)
```

```

```

```
@endif
```

```
<img src="">
```

```
</div>
```

```
<div class="mb-4 text-center">
```

```
<button type="button" class="text-white bg-gray-400 border-0 py-2 px-6 focus:outline-none hover:bg-gray-500 rounded">ショップの詳細も見る</button>
```

```
</div>
```

マイクロモーダルを使おう:



Micromodal.js

Tiny javascript library for creating
accessible modal dialogs



Micromodal



Micromodal - Tiny javascript library for creating accessib...

Accessible modal dialogs with minimal configuration. Just 1.9kb
minified and gzipped, its a tiny library for big change.



Made with Gamma

数量の箇所を動的設定:

app/Http/Controllers/user/itemControllerのshowメソッド:

```
$quantity = Stock::where("product_id", $product->id)->sum("quantity"); //一つの商品の在庫情報を取るために
```

```
if($quantity>9){$quantity=9;}//9より大きかったら 9
```

```
use App\Models\Stock;
```

resource/views/user/show:

```
@for($i = 1; $i <= $quantity; $i++)
```

```
<option value="{{ $i }}">{{ $i }}</option>
```

```
@endfor
```


カート:

database/migrations/create_carts_table.php:

```
//数量と外部キーを結ぶ

$table->foreignId("user_id")

->constrained()

->onUpdate("cascade")

->onDelete("cascade");

$table->foreignId("product_id")

->constrained()

->onUpdate("cascade")

->onDelete("cascade");

$table->integer("quantity");
```

app/Models/Cart.php:

```
protected $fillable = [

    'user_id',

    'product_id',

    'quantity',

];
```

app/Models/User.php:

```
use\App\Models\Product;

public function products(){

    return $this->belongsToMany(Product::class,'carts')

->withPivot(['id','quantity']);

}
```

app/Models/Product.php:

```
use\App\Models\User;

public function users(){

    return $this->belongsToMany(user::class,'carts')

->withPivot(['id','quantity']);

}
```

カートに追加:

routes/web.php:

```
use App\Http\Controllers\User\CartController;
```

```
Route::prefix("cart")->middleware("auth:users")->group(function(){
```

```
Route::post("add",[cartController::class,"add"])->name("cart.add");
```

```
});
```

resources/views/user/show.blade.php:

```
<form method="post" action="{{route('user.cart.add')}}">
```

```
@csrf
```

```
コードが入る
```

```
<input type="hidden" name="product_id" value="{{ $product->id }}">
```

```
</form>
```

app/Http/Controller/User/cartController:

```
public function add( Request $request ){
```

```
dd($request);
```

```
}
```

カートに保存処理:

app/Http/Controller/User/CartController.php:

```
use App\Models\Cart;
```

```
use Illuminate\Support\Facades\Auth; //ユーザーidを取得するために
```

```
$itemInCart = Cart::where('product_id',$request->product_id)//渡ってくるproductを取得
```

```
->where('user_id', Auth::id())->first();//違うユーザーかもしれないのでuserでログインしてるユーザーとする
```

```
if($itemInCart){
```

```
$itemInCart->quantity += $request->quantity; //カートに1入ってるとして更に追加されると合計される
```

```
$itemInCart->save();//保存
```

```
}else{
```

```
Cart::create([
```

```
"user_id" => Auth::id(),
```

```
"product_id" => $request->product_id,
```

```
"quantity" => $request->quantity
```

```
]);dd("テスト");
```

エラー対応します！：

SQLSTATE[42S02]: Base table or view not found: 1146 Table 'laravel_umarche.carts' doesn't exist (SQL: select * from carts where product_id = 108 and user_id = 1 limit 1)

このようなエラーが出た時。これはDBテーブルが作られていない（存在しない） そうなので

`php artisan migrate`をしてあげましょう！

注意:

綴ミスに気をつけてここから先の動画も頑張ってください。

何かエラーが出た時はコピペしてGoogleで調べるのが一番ベストです！

エラー出ずにアプリ作成できるプログラマーはいないので挫折せずに頑張りましょう

カート内を表示:

routes/web.php:

```
Route::get('/', [CartController::class, 'index'])->name('cart.index');
```

app/Http/Controller/User/CartController:

```
use App\Models\User;
```

app/Http/Controller/User/CartController:

```
public function index(){
```

```
$user = User::findOrFail(Auth::id()); //ログインしてるユーザー情報を取得
```

```
$products = $user->products;
```

```
$totalPrice = 0;
```

```
foreach($products as $product){
```

```
$totalPrice += $product->price * $product->pivot->quantity; //金額と数量を掛けたものをtotalPriceに
```

```
}
```

```
dd($products,$totalPrice);
```

```
return view("user.cart.index",compact("product","totalPrice"));
```

```
};
```

app/Http/Controller/User/CartController addメソッド:

```
return redirect()->route("user.cart.index");
```

カート内を表示2:

```
@if (count($products) > 0)
```

```
@foreach ($products as $product)
```

```
<div class="md:flex md:items-center mb-2">
```

```
<div class="md:w-3/12">{{-- 画像 --}}
```

```
@if ($product->imageThird->filename !== null)
```

```

```

```
@else
```

```
<img src="">
```

```
@endif
```

```
</div>
```

```
<div class="md:w-4/12 md:ml-2 ml-4">{{ $product->name }}</div>{{-- 商品名 --}}
```

```
<div class="md:w-3/12 flex justify-around">
```

```
<div>{{ $product->pivot->quantity }}個</div>{{-- 数 --}}
```

```
<div>{{ number_format($product->pivot->quantity * $product->price) }}<span
```

```
class="text-sm text-gray-700">円（税込） </span> </div>{{-- 金額 --}}
```

```
</div>
```

```
<div class="md:w-2/12">削除ボタン</div>
```

```
</div>
```

```
@endforeach
```

```
@else
```

```
カートに商品が入っていません。
```

```
@endif
```



Getting Started

Installation

The simplest and fastest way to get up and running with Tailwind CSS from scratch is with the Tailwind CLI tool.



Installation - Tailwind CSS

The simplest and fastest way to get up and running with Tailwind CSS from scratch is with the Tailwind CLI tool.



エラー対応:

count(): Argument #1 (\$value) must be of type Countable|array, null given (View: /Applications/MAMP/htdocs/laravel/umarche/resources/views/user/cart.blade.php)

このようなエラーが出た場合綴のミスです

compactの中に変数を書きますが定義している変数と一致してるか確認しましょう、
sなど忘れていることが多々あります。

```
return view("user.cart",compact("products","totalPrice"));
```

エラー対応:

フレームワークcssが反応ない時ターミナルでキャッシュ削除してあげましょう

```
npm run dev
```



カート内削除:


routes/web.php:

```
Route::post('delete/{item}', [CartController::class, 'delete'])->name('cart.delete');
```

app/Http/Controller/User/CartController addメソッドの下:

```
public function delete($id){  
  
    Cart::where("product_id",$id)->where("user_id",Auth::id())->delete();//ログインしているユーザーidを削除  
  
    return redirect()->route("user/cart.index");  
  
}
```



 Heroicons

Heroicons

Beautiful hand-crafted SVG icons, by the makers of Tailwind CSS.



 fontawesome

Font Awesome

The internet's icon library + toolkit. Used by millions of designers, devs, & content creators. Open-source. Always free. Always...



 getbootstrap

Bootstrap

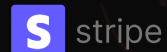
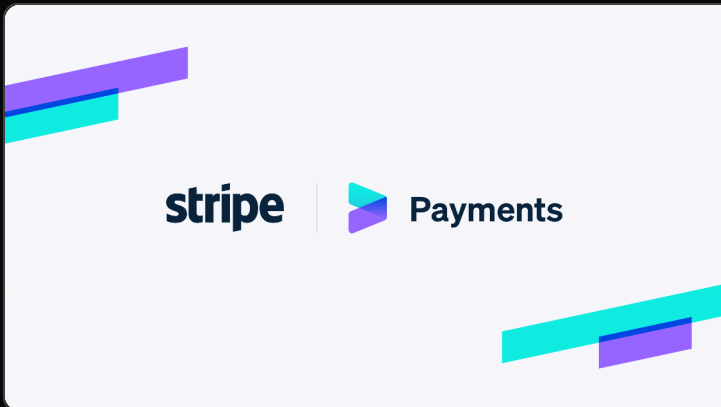
パワフルで拡張性の高い、機能満載のフロントエンドツールキットです。Sassでビルドしてカスタマイズし、あらかじめ用意されたグリッ...

resources/views/user/cart.blade.php:

```
<form method="post" action="{{ route('user.cart.delete', ['item' => $product->id]) }}">  
  
    @csrf  
  
    <button>  
  
        <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5"  
  
        stroke="currentColor" class="w-6 h-6">  
  
            <path stroke-linecap="round" stroke-linejoin="round"  
  
            d="m14.74 9-.346 9m-4.788 0L9.26 9m9.968-3.21c.342.052.682.107 1.022.166m-1.022-.165L18.16  
19.673a2.25 2.25 0 0 1-2.244 2.077H8.084a2.25 2.25 0 0 1-2.244-2.077L4.772 5.79m14.456 0a48.108  
48.108 0 0 0-3.478-.397m-12 .562c.34-.059.68-.114 1.022-.165m0 0a48.11 48.11 0 0 1 3.478-.397m7.5  
0v-.916c0-1.18-.91-2.164-2.09-2.201a51.964 51.964 0 0 0-3.32 0c-1.18.037-2.09 1.022-2.09  
2.201v.916m7.5 0a48.667 48.667 0 0 0-7.5 0" />  
  
        </svg>  
  
    </button>  
  
</form>
```

決済機能:

- stripeは実際に決済されると3,6%の手数料がかかる。
- 登録料はかからない。
- テストモードあり。
- API型決済ライブラリになっていて使いやすさ抜群



stripe



Stripe Payments | グローバル決済処理プラットフォーム

成長中のスタートアップからグローバル企業まで、あらゆるビジネスに対応できる決済ソリューションを利用して、オンライン、対面、世...

Stripeの使用方法:

1. **LaravelCasher(定期支払い・サブスクリプション)**
2. **stripeが発行しているライブラリ**

```
composer require stripe/stripe-php
```

route/web.php:

```
Route::get("checkout",[CartController::class,"checkout"]->name("cart.checkout"));
```

app/Http/Controller/User/CartController:

```
public function checkout()
```

```
{
```

```
$user = User::findOrFail(Auth::id()); //1ログインしているユーザー情報を取る
```

```
$products = $user->products; //2ユーザーから商品を取得する
```

```
$lineItems = []; //3stripeではカートに入っている情報をlineItemsと呼び配列を作り中に入れていく
```

```
foreach ($products as $product) { //4foreachで全てのカートに入っている情報をとる
```

```
$quantity = ""; //在庫情報の処理
```

```
$quantity = Stock::where('product_id', $product->id)->sum('quantity'); //商品の現在の在庫数を調べる
```

```
if ($product->pivot->quantity > $quantity) {
```

```
//カート内の数量より現在の在庫数が多かったら買えない処理に****
```

```
return redirect()->route('user.cart.index'); //user.cart.indexに戻す
```

```
} else {
```

```
// 買える時の処理
```

```
$price_data = ([
```

```
'unit_amount' => $product->price, //商品価格
```

```
'currency' => 'jpy', //通貨
```

```
'product_data' => $product_data = ([
```

```
'name' => $product->name, //商品名
```

```
'description' => $product->information, //商品情報
```

```
]),
```

```
]);
```

```
$lineItem = [
```

```
'price_data' => $price_data, //$price_dataの事
```

```
'quantity' => $product->pivot->quantity, //在庫情報
```

```
];
```

```
array_push($lineItems, $lineItem);
```

```
}
```

```
} // $lineItemsの中に商品名と商品情報と商品価格と通貨と在庫情報を入れていく
```

```
// もし買える状態でstripeに渡す前に在庫情報を減らすので
```

```
foreach ($products as $product) {
```

```
Stock::create([
```

```
'product_id' => $product->id, //その商品に対して選択
```

```
'type' => \Constant::PRODUCT_LIST['reduce'],
```

```
//商品を減らす以前使った定数(app/Http/Controller/Owner/ProductController)
```

```
'quantity' => $product->pivot->quantity * -1 //カートの在庫数を減らす
```

```
]);
```

```
}
```

```
dd("test");
```

```
\Stripe\Stripe::setApiKey(env('STRIPE_SECRET_KEY')); //シークレットキー(envファイルに書いていたから env('STRIPE_SECRET_KEY')このようになります)
```

```
$checkout_session = \Stripe\Checkout\Session::create([
```

```
'payment_method_types' => ['card'],
```

```
'line_items' => [$lineItems], //22行目の配列が入ってくる
```

```
'mode' => 'payment', //一回払い(モード)
```

```
'success_url' => route('user.items.index'), //支払い成功したらuser.items.indexに戻す
```

```
'cancel_url' => route('user.cart.index'), //支払い失敗したらuser.cart.indexに戻す
```


```
]);
```

```
$publicKey = env('STRIPE_PUBLIC_KEY'); //公開キー
```

```
return view('user.checkout',
```

```
compact('checkout_session', 'publicKey')); //checkout_sessionに情報が全て入って、publicKeyと渡す！
```

```
}}
```


 docs.stripe.com

Create a Session | Stripe API Reference

Complete reference documentation for the Stripe API. Includes code snippets and examples for our Python, Java, PHP, Node.js, Go, Ruby, and .NET libraries.

 docs

Stripe-hosted page

 docs.stripe.com

Stripe-hosted page

Check Outへのボタン作成:

```
<div class="my-2">
```

```
小計:{{number_format($totalPrice)}}<span class="text-sm text-gray-700">円(税込)</span>
```

```
</div>
```

```
<div>
```

```
<button onclick="location.href='{{route('user.cart.checkout')}}'" class="flex ml-auto text-white bg-indigo-500 border-0 py-2 px-6 focus:outline-none hover:bg-indigo-600 rounded">購入する</button>
```

```
</div>
```


上手にいかない人は。。

Cart.Controller.php:

```
// Stock::create([

// 'product_id' => $product->id, //その商品に対して選択

// 'type' => \Constant::PRODUCT_LIST['reduce'],

// //商品を減らす以前使った定数(app/Http/Controller/Owner/ProductController)

// 'quantity' => $product->pivot->quantity * -1 //カート の在庫数を減らす

// ]);

と

//4foreachで全てのカートに入っている情報をとる

// $quantity = ""; //在庫情報の処理****

// $quantity = Stock::where('product_id', $product->id)->sum('quantity'); //商品の現在の在庫数を調べる ****

// if ($product->pivot->quantity > $quantity){

//カート内の数量より現在の在庫数が多かったら買えない処理に

// return redirect()->route('user.cart.index'); //user.cart.indexに戻る

// } else {

// 買える時の処理****

// コメントアウトしておこう
```

購入ボタンからstripeへリダイレクト:

<p>決済ページへリダイレクトします</p>

<script src="https://js.stripe.com/v3/"></script>

<script>

const publicKey = "{{ \$publicKey }}"//公開キー

const stripe = Stripe(publicKey)//Stripe(publicKey)で初期化してstripe変数に

// 画面読み込み処理

window.onload = function(){

stripe.redirectToCheckout({//Checkoutに飛ばす

sessionId:'{{ \$session->id }}'//stripeで作成した情報(idは作成時、自動で作られている)

}).then(function (result){

window.location.href="{{ route('user.cart.index') }}"//NGだったらuser.cart.indexに戻す

});

}

</script>

stripe DOCS

Stripe-hosted page



docs.stripe.com

Stripe-hosted page



決済成功時の処理:

```
Route::get("success",[CartController::class,"success"]->name("cart.success"));
```

```
public function success(){
```

```
    Cart::where('user_id',Auth::id())->delete();
```

```
    return redirect()->route('user.items.index'); }
```

Stripe決済キャンセル時の在庫処理:

```
Route::get("cancel",[CartController::class,"cancel"])->name("cart.cancel");
```

```
public function cancel($id){
```

```
    $user = User::findOrFail(Auth::id());
```

```
    foreach($user->products as $product){
```

```
        Stock::create([
```

```
            'product_id' => $product->id,//その商品に対して選択
```

```
            'type' => \Constant::PRODUCT_LIST['add'],//商品を増やす。以前使った定数  
            (app/Http/Controller/Owner/ProductController)
```

```
            'quantity' => $product->pivot->quantity //カートの商品数を減らす
```

```
        ]); }
```

```
    return redirect()->route('user.cart.index');
```

```
}
```


決済機能バージョン変更訂正:

上記資料でうまくいかない場合こちらお使いください。

Cart.Controller.php:

```
<?php

namespace App\Http\Controllers\User;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\Models\Cart;

use App\Models\Stock;

use Illuminate\Support\Facades\Auth; //ユーザーidを取得するために

use App\Models\User;

use App\Constants\Common; // Correctly import the Common class

class CartController extends Controller
{

    public function index()
    {

        $user = User::findOrFail(Auth::id()); //ログインしてるユーザー情報を取得

        $products = $user->products;

        $totalPrice = 0;

        foreach ($products as $product) {

            $totalPrice += $product->price * $product->pivot->quantity; //金額と数量を掛けたものをtotalPriceに

        }

        return view("user.cart", compact("products", "totalPrice"));

    }

    public function add(Request $request){

        $itemInCart = Cart::where('product_id', $request->product_id) //渡ってくるproductを取得

        ->where('user_id', Auth::id())->first(); //違うユーザーかもしれないのでuserでログインしてるユーザーとする

        if ($itemInCart) {

            $itemInCart->quantity += $request->quantity; //カートに1入ってるとして更に追加されると合計される

            $itemInCart->save(); //保存

        } else {

            Cart::create([

                "user_id" => Auth::id(),

                "product_id" => $request->product_id,

                "quantity" => $request->quantity

            ]);

        }

        return redirect()->route("user.cart.index");

    }

    public function delete($id)

    {

        Cart::where("product_id", $id)->where("user_id", Auth::id())->delete(); //ログインしているユーザーidを削除

        return redirect()->route("user.cart.index");

    }

    public function checkout()

    {

        $user = User::findOrFail(Auth::id()); //1ログインしているユーザー情報を取る

        $products = $user->products; //2ユーザーから商品を取得する

        $lineItems = []; //3stripeではカートに入っている情報をlineItemsと呼び配列を作り中に入れていく

        foreach ($products as $product) {

            $price_data = ([

                'unit_amount' => $product->price, //商品価格

                'currency' => 'jpy', //通貨

                'product_data' => $product_data = ([

                    'name' => $product->name, //商品名

                    'description' => $product->information, //商品情報

                ]),

            ]);

            $lineItem = [

                'price_data' => $price_data, // $price_dataの事

                'quantity' => $product->pivot->quantity, //在庫情報

            ];

            array_push($lineItems, $lineItem);

        } // $lineItemsの中に商品名と商品情報と商品価格と通貨と在庫情報を入れていく

        // もし買える状態でstripeに渡す前に在庫情報を減らすのでforeach ($products as $product) {

        Stock::create([

            'product_id' => $product->id, //その商品に対して選択

            'type' => Common::PRODUCT_LIST['reduce'], // Correctly refer to the constant

            'quantity' => $product->pivot->quantity * -1 //カートの在庫数を減らす

        ]);

        }

        // dd("test");

        \Stripe\Stripe::setApiKey(env('STRIPE_SECRET_KEY')); //シークレットキー(envファイルに書いていたからenv('STRIPE_SECRET_KEY')このようになります)

        $checkout_session = \Stripe\Checkout\Session::create([

            'payment_method_types' => ['card'],

            'line_items' => [$lineItems], //22行目の配列が入ってくる

            'mode' => 'payment', //一回払い(モード)

            'success_url' => route("user.cart.success"), //支払い成功したらuser.items.indexに戻す

            'cancel_url' => route("user.cart.cancel"), //支払い失敗したらuser.cart.indexに戻す

        ]);

        $publicKey = env('STRIPE_PUBLIC_KEY'); //公開キー

        return view(

            'user.checkout',

            compact('checkout_session', 'publicKey')

        ); //checkout_sessionに情報が全て入って、publicKeyと渡す！

    }

    public function success()

    {

        Cart::where('user_id', Auth::id())->delete();

        return redirect()->route("user.items.index");

    }

    public function cancel()

    {

        $user = User::findOrFail(Auth::id());

        foreach ($user->products as $product) {

            Stock::create([

                'product_id' => $product->id, //その商品に対して選択

                'type' => Common::PRODUCT_LIST['add'], // Correctly refer to the constant

                'quantity' => $product->pivot->quantity //カートの在庫数を減らす

            ]);

        }return redirect()->route('user.cart.index'); }

}
```


決済機能バージョン変更訂正2:

```
<?php
```

```
namespace App\Constants;
```

```
class Common{
```

```
const PRODUCT_ADD = "1";//追加
```

```
const PRODUCT_REDUCE = "2";//削減
```

```
//上のままでもOK下のように連想配列も書いても良し
```

```
const PRODUCT_LIST = [
```

```
"add" => self::PRODUCT_ADD,//クラスの中でconstを指定するならself::が必要!
```

```
"reduce" => self::PRODUCT_REDUCE
```

```
];
```

```
}
```

README.mdに追記:

##ダウンロード方法

git clone git clone

https://github.com/aokitashipro/laravel_umarche.git

git clone ブランチを指定してダウンロードする場合 git clone -b ブランチ名

https://github.com/aokitashipro/laravel_umarche.git

もしくはzipファイルでダウンロードしてください

##インストール方法

cd laravel_umarche composer install npm install npm run dev

.env.example をコピーして .env ファイルを作成

.envファイルの中の下記をご利用の環境に合わせて変更してください。

DB_CONNECTION=mysql DB_HOST=127.0.0.1 DB_PORT=3306 DB_DATABASE=laravel_umarche
DB_USERNAME=umarche DB_PASSWORD=password123

XAMPP/MAMPまたは他の開発環境でDBを起動した後に

php artisan migrate:fresh --seed

と実行してください。(データベーステーブルとダミーデータが追加されればOK)

最後に Php artisan key:generate と入力してキーを生成後、

Php artisan serve で簡易サーバーを立ち上げ、表示確認してください。

画像のダミーデータを扱う場合は php artisan storage:link でリンク作成し

public/images内の画像ファイルを storage/app/public内に shopsフォルダと productsフォルダを作成し それぞれに画像をコピーしてください。

画像コピー後

php artisan migrate:fresh --seed と実施してください。

商品の一覧:

app/Models/Product:

```
public function scopeAvailableItems($query)

{

    $stocks = DB::table('t_stocks')

->select(

    'product_id',

    DB::raw('sum(quantity) as quantity'))

->groupBy('product_id')

->having('quantity', '>', 1);

return $query

->joinSub($stocks, 'stock', function ($join) {

    $join->on('products.id', '=', 'stock.product_id');

})

->join('shops', 'products.shop_id', '=', 'shops.id')

->join('secondary_categories', 'products.secondary_category_id', '=', 'secondary_categories.id')

->join('images as image1', 'products.image1', '=', 'image1.id')

->where('shops.is_selling', true)

->where('products.is_selling', true)

->select(

    'products.id as id',

    'products.name as name',

    'products.price',

    'products.sort_order as sort_order',

    'products.information',

    'secondary_categories.name as category',

    'image1.filename as filename'

);}
```

use Illuminate\Support\Facades\DB;

app/Http/Controller/User/ItemController:

```
$products = Product::availableItems()->get();
```

商品の詳細:

app/Http/Controller/User/ItemController:

```
$this->middleware(function ($request, $next) {
```

```
$id = $request->route()->parameter("item"); //itemのidを取得
```

```
// routes/web.phpのRoute::get("show/{item}",[ItemController::class,"show"])の事
```

```
if (!is_null($id)) { //itemのidが存在するなら ↓
```

```
$itemId = Product::availableItems()->where('product.id',$id)->exists(); //productのidが入ってきた値idと一致してるか。入ってきた値が存在するかどうか確認。
```

```
// ↓ itemIdが存在していなかったら
```

```
if (!$itemId) {
```

```
abort(404); //404の画面表示
```

```
}
```

```
}
```

```
return $next($request);
```

```
});
```


表示順1:

ECサイト表示順について。

- おすすめ順、高い順、安い順、新しい順、古い順などあります。
- 大規模なサイトになる程上位表示にお金がかかる。
- 広告枠と売れた時の手数料と上位表示などを週 1 万円とか。

app/Constants/common.php:

```
// 表示順の定数を作成

const ORDER_RECOMMEND = '0';//おすすめ順

const ORDER_HIGHER = '1';//高い順

const ORDER_LOWER = '2';//安い順

const ORDER_LATER = '3';//新しい順

const ORDER_OLDER = '4';//古い順

// 各定数を配列に入れていく、同じファイルの定数にはself::必須

const SORT_ORDER = [

'recommend' => self::ORDER_RECOMMEND,

'higherPrice' => self::ORDER_HIGHER,

'lowerPrice' => self::ORDER_LOWER,

'later' => self::ORDER_LATER,

'older' => self::ORDER_OLDER

];
```

app/Models/Product:

```
public function scopeSortOrder($query,$sortOrder)

{

// 特定のソート順が指定されていない場合や、推奨されるソート順が選択された場合のデフォルト動作の条件。

if($sortOrder === null || $sortOrder === \Constant::SORT_ORDER['recommend']){

return $query->orderBy('sort_order', 'asc') ;

}

// 高い価格から低い価格へ並び替えるための条件(desc)

if($sortOrder === \Constant::SORT_ORDER['higherPrice']){

return $query->orderBy('price', 'desc') ;

}

// 低い価格から高い価格へ並び替えるための条件(asc)

if($sortOrder === \Constant::SORT_ORDER['lowerPrice']){

return $query->orderBy('price', 'asc') ;

}

// 商品が追加された日付が新しい順に並び替えるための条件(desc)

if($sortOrder === \Constant::SORT_ORDER['later']){

return $query->orderBy('products.created_at', 'desc') ;

}

// 商品が追加された日付が古い順に並び替えるための条件(asc)

if($sortOrder === \Constant::SORT_ORDER['older']){

return $query->orderBy('products.created_at', 'asc') ;

}

}
```


セレクトボックスを貼る:

resources/views/user/index:

```
<div class="flex justify-between items-center">

<h2 class="font-semibold text-xl text-gray-800 leading-tight">

商品一覧

</h2>

<form method="get" action="{{ route('user.items.index') }}">

<div class="flex">

<div>

<span class="text-sm">表示順</span>

<br>

<select name="sort" class="mr-4" id="sort">

<option value="{{ \Constant::SORT_ORDER['recommend'] }}"

@if(\Request::get('sort') === \Constant::SORT_ORDER['recommend'] )

selected

@endif>おすすめ順

</option>

<option value="{{ \Constant::SORT_ORDER['higherPrice'] }}"

@if(\Request::get('sort') === \Constant::SORT_ORDER['higherPrice'] )

selected{{--option value値と\Request::get('sort')が同じなら selected --}}

@endif>料金の高い順

</option>

<option value="{{ \Constant::SORT_ORDER['lowerPrice'] }}"

@if(\Request::get('sort') === \Constant::SORT_ORDER['lowerPrice'] )

selected

@endif>料金の安い順

</option>

<option value="{{ \Constant::SORT_ORDER['later'] }}"

@if(\Request::get('sort') === \Constant::SORT_ORDER['later'] )

selected

@endif>新しい順

</option>

<option value="{{ \Constant::SORT_ORDER['older'] }}"

@if(\Request::get('sort') === \Constant::SORT_ORDER['older'] )

selected

@endif>古い順

</option>

</select>

</div>

<span class="text-sm">表示件数</span>

</div>

</form>

</div>
```

resources/views/user/index:

```
<script>

const select = document.getElementById('sort')//id="sort"を取得

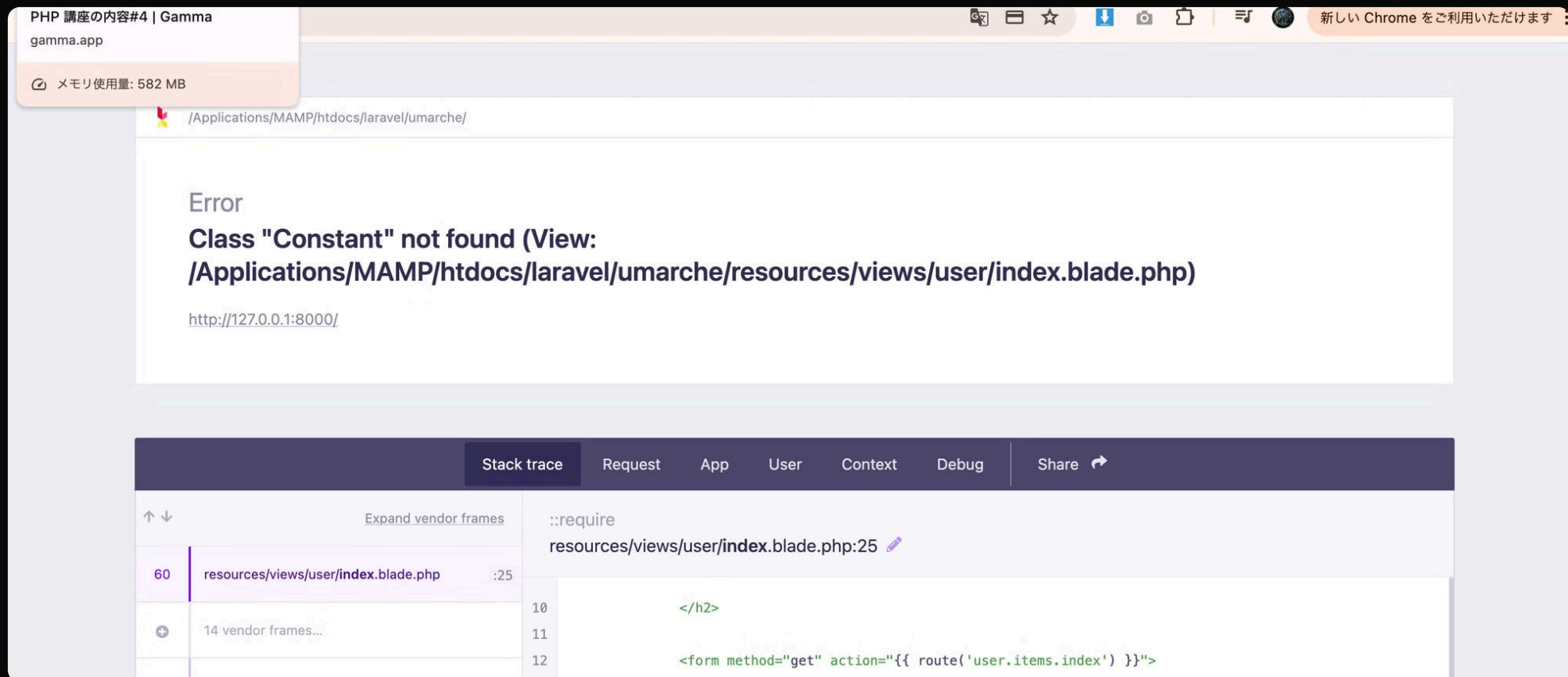
select.addEventListener('change', function(){

//イベントが発生(change)した瞬間submiする

this.form.submit()

})

</script>
```



このようなエラーになられた方はuser/index.blade.phpの先頭に

```
<?php use App\Constants\Common; ?>
```

と記入して

\Constant:: の部分を Common:: に変更

```
ex...<option value="{{ Common::SORT_ORDER['recommend'] }}"
```

```
@if (\Request::get('sort') === Common::SORT_ORDER['recommend']) selected @endif>おすすめ順こんな  
感じ！これは全てに。。。。
```

表示件数:

resources/views/user/index:

```
<span class="text-sm">表示件数</span>
```

```
<br><select id="pagination" name="pagination">
```

```
<option value="20"
```

```
@if (\Request::get('pagination') === '20')
```

```
selected
```

```
@endif>20件
```

```
</option>
```

```
<option value="50" @if (\Request::get('pagination') === '50') selected @endif>50件
```

```
</option>
```

```
<option value="100" @if (\Request::get('pagination') === '100') selected @endif>100件
```

```
</option>
```

```
</select>
```

resources/views/user/index:

```
const paginate = document.getElementById('pagination')
```

```
paginate.addEventListener('change', function() {
```

```
this.form.submit()
```

```
})
```

```
})
```

resources/views/user/index:

```
{{ $products->appends([
```

```
'sort' => \Request::get('sort'),
```

```
'pagination' => \Request::get('pagination'),
```

```
}}}
```

検索フォーム:

resources/views/user/index:

```
<div class="lg:flex items-center">

<select class=" mb-2 lg:mb-0 lg:mr-2" name="category">
{{-- owner/products/createと同じ --}}

<option value="0" selected>全て</option>

@foreach ($categories as $category)

<optgroup label="{{ $category->name }}">

@foreach ($category->secondary as $secondary)

<option value="{{ $secondary->id }}">

{{ $secondary->name }}

</option>

@endforeach

@endforeach

</select>

<div class="space-x-2 items-center flex">

<div>

<input name="keyword" class="border-gray-500 mr-2 py-2"

placeholder="キーワードを入力">

</div>

<div>

<button class=" ml-auto text-white bg-indigo-500 border-0 py-2 px-6

focus:outline-none hover:bg-indigo-600 rounded">

検索する

</button>

</div>

</div>

</div>

</div>
```

app/Http/Controllers/User/itemController:

```
use App\Models\PrimaryCategory;

dd($request);

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


検索フォーム（カテゴリー）：

resources/views/user/index:

```
@if(\Request::get('category') === '0')@endif
```

```
@if(\Request::get('category') == $secondary->id)@endif
```

app/Http/Controllers/User/itemController:

```
->selectCategory($request->category ?? "0");//選んでいない場合初期値0に！
```

app/Models/Product:

```
public function scopeSelectCategory($query,$categoryId)
```

```
{
```

```
if($categoryId !=="0"){
```

```
return $query->where("secondary_category_id",$categoryId);
```

```
}else{
```

```
return ;
```

```
}
```

```
}
```


indexを正しくコード書いてみた

```
@foreach ($categories as $category)
```

```
<optgroup label="{{ $category->name }}">
```

```
@foreach ($category->secondary as $secondary)
```

```
<option value="{{ $secondary->id }}"
```

```
@if (\Request::get('category') == $secondary->id) selected @endif>
```

```
{{ $secondary->name }}
```

```
</option>
```

```
@endforeach
```

```
</optgroup>
```

```
@endforeach
```

検索フォーム（キーワード）：

app/Http/Controllers/User/itemController:

```
->searchKeyword($request->keyword)
```

app/Models/Product:

```
public function scopeSearchKeyword($query, $keyword)
```

```
{
```

```
if(!is_null($keyword))//与えられたキーワードがnullでないことを確認します
```

```
{
```

```
$spaceConvert = mb_convert_kana($keyword,'s'); //キーワード内の全角スペースを半角スペースに変換にして検索キーワードの一貫性を保証
```

```
$keywords = preg_split('/[\s]+/', $spaceConvert,-1,PREG_SPLIT_NO_EMPTY); //変換されたキーワードを空白で区切って配列にPREG_SPLIT_NO_EMPTYフラグは、空の文字列を結果から除外するために使用されます。
```

```
foreach($keywords as $word) //単語をループで回す
```

```
{
```

```
$query->where('products.name','like','%'.$word.'%');
```

```
}
```

```
return $query;
```

```
} else {
```

```
return ;
```

```
}
```

```
}
```

mailtrap.io:

メール関連

商品を購入時、ユーザーやオーナーそれぞれに通知

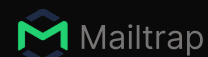
ログインパスワード忘れた時の連絡など

メールサーバーや

パブリッククラウド(AWS)側にメールサーバーが存在する。

環境によって設定が変わる(.envを編集)

- `MAIL_DRIVER, MAIL_HOST`
- `MAIL_PORT, MAIL_USERNAME,`
- `MAIL_PASSWORD` など.




Mailtrap: Email Delivery Platform

Email Delivery Platform to test, send and control email infrastructure in one place. Great for dev teams. Start today - Sig...



※バージョンにより動画とボタンの位置などの変わることがあります

メール設定:

 readouble.com



メール 8.x Laravel

メールの送信は複雑である必要はありません。Laravelは、人気のあるSwiftMailerライブラリを利用したクリーンでシンプルなメールAPIを提供します。LaravelとSwiftMailerは、SMTP、Mailgun、Postmark、AmazonSES、およ...

テストメール:

app/Mail/TestMail.php:

```
return $this
```

```
->subject('テスト送信完了') //タイトル
```

```
->view('emails.test'); //本文
```

app/Http/Controllers/User/itemController:

```
use Illuminate\Support\Facades\Mail;
```

```
use App\Mail\TestMail;
```

app/Http/Controllers/User/itemControllerのindexメソッド:

```
Mail::to('test@example.com') //受信者の指定
```

```
->send(new TestMail()); //Mailableクラス
```


非同期処理:

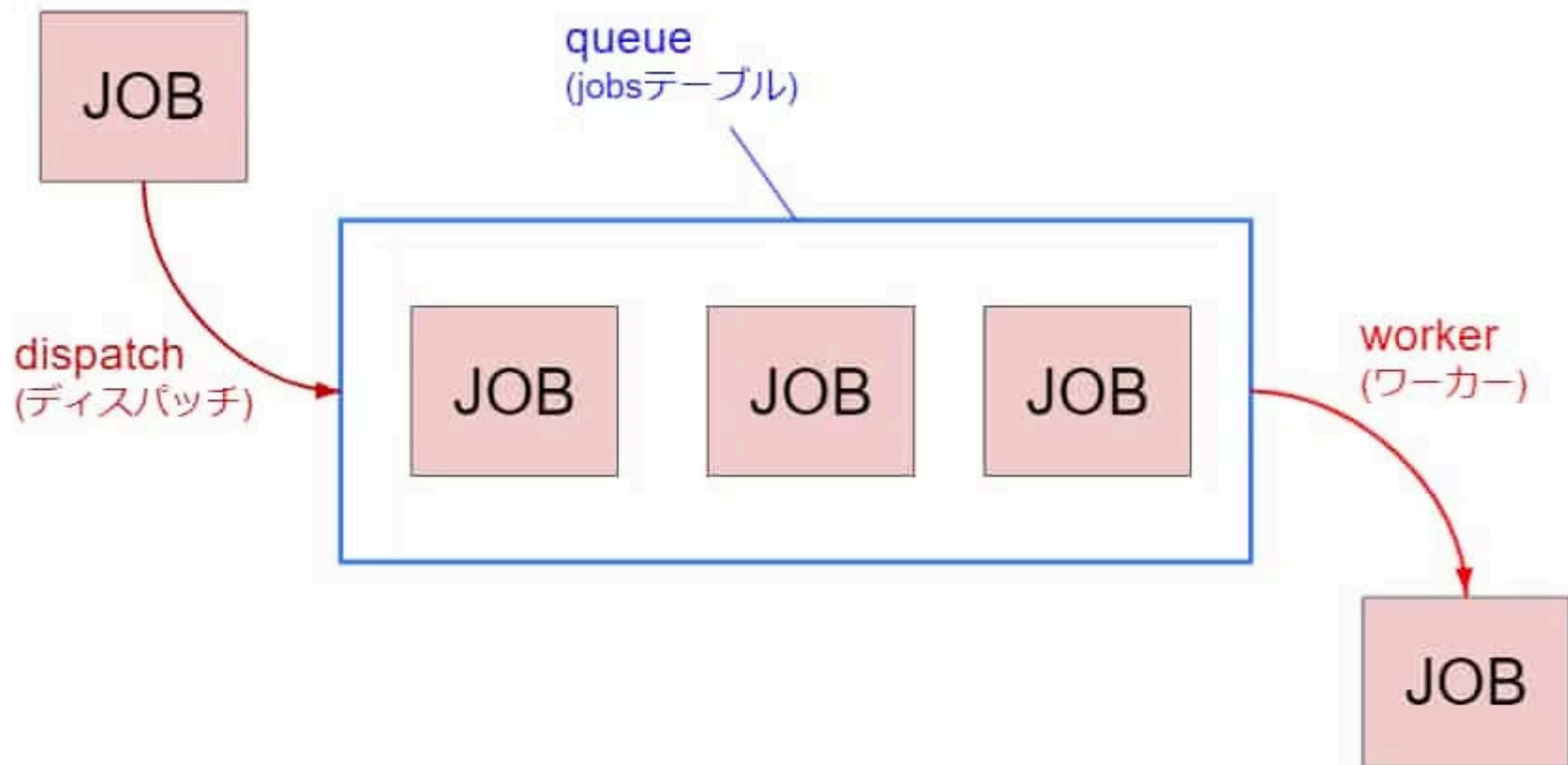
同期処理・・・送信してから画面更新(3~4秒)

非同期処理・・・画面上は更新しつつ(0.1秒~)裏側でメール送信

Queue(キュー)・・・待ち行列

Job(ジョブ)・・・1つ1つの処理

Worker(ワーカー)・・・処理をする人



```
Php artisan make:job SendThanksMail
```

```
app/Http/Controllers/User/ItemController:
```

```
App\Jobs\SendThanksMail.php
```

```
SendThanksMail:dispatch();
```

```
php artisan queue:work
```

CartService作成(メール設定):

商品購入後の流れ:

カート情報を取得

カートから商品を削除

ユーザー向けのメール(Job)

オーナー向けのメール(複数Job)

カート情報から下記を取得

(商品情報、在庫、オーナー(名前・メールアドレス))

-> コード量が増えるので

App\Services\CartService.phpを作成

app/Services/CartService.php:

```
<?php

namespace App\Services;

use App\Models\Product;

use App\Models\Cart;

class CartService
{

    public static function getItemsInCart($items)
    {

        $products = []; //空の配列を準備

        dd($items);

        foreach($items as $item){ // カート内の商品の一つずつ処理

        }

        略(次ページ);

        return $products; // 新しい配列を返す

    }

}
```

app/Http/Controllers/Use/CartController:

```
use App\Services\CartService;

...

$items = Cart::where('user_id', Auth::id())->get(); //カートの中でログインしているユーザーの商品情報が設定されている

$products = CartService::getItemsInCart($items);
```

カート情報から新規配列:

app/Services/CartService.php:

```
use App\Models\Product;
```

```
use App\Models\Cart;
```

```
$pro = Product::findOrFail($item->product_id);//商品を取得
```

```
$owner = $pro->shop->owner->select('name','email')->first()->toArray();//オーナー情報を取得
```

```
$values = array_values($owner);//値のみ取得
```

```
$keys = ['ownerName','email'];//キーのみ名前を変える
```

```
$ownerInfo = array_combine($keys,$values);//$keysと$valuesをくっつける！
```

```
dd($ownerInfo);
```

追加:

```
$product = Product::where('id', $item->product_id)
```

```
->select('id', 'name', 'price')->get()->toArray(); // 商品情報の配列
```

```
$quantity = Cart::where('product_id', $item->product_id)
```

```
->select('quantity')->get()->toArray(); // 在庫数の配列
```

```
dd($ownerInfo,$product,$quantity );
```

```
$result = array_merge($product[0], $ownerInfo, $quantity[0]); // 配列の結合
```

```
dd($result);
```

```
array_push($products, $result); //配列に追加
```

ユーザー向け商品購入メール:

```
php artisan make:mail ThanksMail
```

```
app/Http/Controllers/User/CartController:
```

```
use App\Jobs\SendThanksMail;
```

```
$user = User::findOrFail(Auth::id());
```

```
SendOrderedMail::dispatch($products, $user);
```

```
dd("メール送信test");
```

```
app/Jobs/sendThanksMail:
```

```
use App\ThanksMail;
```

```
//CartControllerのdispatch($product, $user)を受け取れるように
```

```
public $products;
```

```
public $user;
```

```
public function __construct($products,$user)
```

```
{
```

```
//引数で入ってきた$products,$userを$this->productsと$this->userにしてhandleメソッドで使えるように
```

```
$this->products = $products;
```

```
$this->user = $user;
```

```
}
```

```
//user情報の中でemailまで探してくれる
```

```
Mail::to($this->user)->send(new ThanksMail($this->products, $this->user));
```

```
app/Mail/ThanksMail.php:
```

```
public $products;
```

```
public $user;
```

```
public function __construct($products, $user)
```

```
{
```

```
$this->products=$products;
```

```
$this->user=$user;
```

```
}
```

```
<p class="mb-4">{{ $user->name }}様</p>
```

```
<p class="mb-4">下記のご注文ありがとうございました!!</p>
```

```
商品内容
```

```
@foreach($products as $product)
```

```
<ul class="mb-4">
```

```
<li>商品名:{{ $product['name'] }}</li>
```

```
<li>商品金額:{{ number_format($product['price']) }}円</li>
```

```
<li>商品数:{{ $product['quantity'] }}</li>
```

```
<li>合計金額:{{ number_format($product['price'] * $product['quantity']) }}円</li>
```

```
</ul>
```

```
@endforeach
```


オーナー向け商品販売メール:

ユーザーとほぼ同じですけど複数の商品買う際にオーナーが別なのでそれぞれにメール送信!

app/Http/Controllers/User/CartController:

```
//複数メールを送るのでそれぞれの商品とユーザーを処理する
```

```
foreach($products as $product)
```

```
{
```

```
SendOrderedMail::dispatch($product, $user);
```

```
}
```

app/Jobs/SendOrderedMail.php:

```
use Illuminate\Support\Facades\Mail;
```

```
use App\Mail\OrderedMail;
```

app/Jobs/SendOrderedMail.php:

```
public $product;
```

```
public $user;
```

```
public function __construct($product, $user)
```

```
{
```

```
$this->product=$product;
```

```
$this->user=$user;
```

```
}
```

app/Jobs/SendOrderedMail.php:

```
Mail::to($this->product['email'])
```

```
->send(new OrderedMail($this->product, $this->user));
```

app/Mail/OrderedMail:

```
public $product;
```

```
public $user;
```

```
public function __construct($product, $user)
```

```
{
```

```
$this->product=$product;
```

```
$this->user=$user;
```

```
}
```

resources/views/emails/Ordered.blade.php:

```
<p class="mb-4">{{ $product["ownerName"] }}様の商品が注文されました。</p>
```

```
<div class="mb-4">商品情報</div>
```

```
<ul class="mb-4">
```

```
<li>商品名:{{ $product['name'] }}</li>
```

```
<li>商品金額:{{ number_format($product['price']) }}円</li>
```

```
<li>商品数:{{ $product['quantity'] }}</li>
```

```
<li>合計金額:{{ number_format($product['price'] * $product['quantity']) }}円</li>
```

```
</ul>
```

```
<div class="mb-4">購入者情報</div>
```

```
<ul>
```

```
<li>{{ $user->name }}様</li>
```

```
</ul>
```


その他1:

app/Http/Controllers/Admin/Auth/Authenticated~:

```
use Illuminate\Support\Facades\Log;
```

```
Log::debug('admin',$request->session()->all());
```

その他2:

.envと.env.example:

```
SESSION_COOKIE=user
```

```
SESSION_COOKIE_OWNER=owner
```

```
SESSION_COOKIE_ADMIN=admin
```

config/session.php:

```
'cookie' => env('SESSION_COOKIE',
```

```
Str::slug(env('APP_NAME', 'laravel'), '_').'_session'),
```

```
'cookie_owner' => env('SESSION_COOKIE_OWNER',
```

```
Str::slug(env('APP_NAME', 'laravel'), '_').'_session_owner'), // 追記
```

```
'cookie_admin' => env('SESSION_COOKIE_ADMIN',
```

```
Str::slug(env('APP_NAME', 'laravel'), '_').'_session_admin'), // 追記
```

app/Providers/AppServiceProviders:

```
// ownerから始まるURL
```

```
if (request()->is('owner*')) {
```

```
    config(['session.cookie' => config('session.cookie_owner')]);
```

```
}
```

```
// adminから始まるURL
```

```
if (request()->is('admin*')) {
```

```
    config(['session.cookie' => config('session.cookie_admin')]);
```

```
}
```

その他3:

resources/views/layouts/app.blade.php:

```
@if (request()->is('admin*'))
```

```
@include('layouts.admin-navigation')
```

```
@elseif (request()->is('owner*'))
```

```
@include('layouts.owner-navigation')
```

```
@else
```

```
@include('layouts.user-navigation')
```

```
@endif
```