

# ユーザー その1

# ユーザーの概要

# ユーザーでできる事

---

商品一覧・商品検索  
商品詳細・店舗詳細  
商品をカートに保存  
購入(Stripe API)  
購入確認のメール

# 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>

# ユーザー情報

---

モデル・マイグレーションは  
もともと存在する

`php artisan make:seed UserSeeder`  
ダミーデータだけ作成

# 商品一覽

# 商品一覧画面の準備 その1

ルート

```
Route::middleware('auth:users')  
->group(function(){  
    Route::get('/', [ItemController::class, 'index'])->name('items.index');  
});
```

コントローラ

```
php artisan make:controller User/ItemController  
return view('user.index');
```

ビュー

```
user/index.blade.php
```

# 商品一覧画面の準備 その1

---

ルート

Dashboardは今回使わないのでコメントアウト

```
App/Providers/RouteServiceProvider.php  
public const HOME = '/'; に変更
```



# 商品一覧画面の準備 その2

ビュー

dashboard.blade.phpをコピー

products/index.blade.phpをコピー

Dashboardが残っているので

routeは user.items.index

表示名は ホームにそれぞれ変更

\$productsはダミーがないので後ほど生成



# Faker & Factory

# Faker & Factory

---

Faker ・ ・ PHPライブラリ ダミーデータ生成  
<https://fakerphp.github.io/>

Factory ・ ・ ダミーを量産する仕組み  
Laravel8からクラスベースに変更

config/app.php 日本語化対応  
'faker\_locale' => 'ja\_JP', に変更

念の為 php artisan config:clear でキャッシュ削除

# Faker & Factory

---

php artisan make:factory ProductFactory —  
model=Product

php artisan make:factory StockFactory —  
model=Stock

fakerチートシート

[https://qiita.com/tosite0345/items/  
1d47961947a6770053af](https://qiita.com/tosite0345/items/1d47961947a6770053af)

# ProductFactory

---

```
return [  
    '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),  
];
```

# StockFactory

---

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

```
return [  
    'product_id' => Product::factory(),  
    'type' => $this->faker->numberBetween(1,2),  
    'quantity' => $this->faker->randomNumber,  
];
```

# DatabaseSeederで読み込み

---

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

```
public function run()
```

```
    $this->call([
```

```
        略
```

```
    ]);
```

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

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

```
    }
```



# 見栄え調整

---

ビュー

TailblocksのEcommerceから一部拝借

カテゴリ

`$product->category->name`

金額 (カンマを表示)

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



# 商品一覧クエリ

# 商品一覧クエリ その1

---

表示する条件

Shop ・ ・ is\_selling = true

Product ・ ・ is\_selling = true

Stockの合計 ・ ・ 1以上

# 商品一覧クエリ その2

Stockの合計をグループ化->数量が1以上  
SQLの場合

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

検索条件

Where ・ ・ groupByより前に条件指定

Having ・ ・ groupByの後に条件指定

# 商品一覧クエリ その3

select内でsumを使うため  
クエリビルダのDB::rawで対応

```
$stocks = DB::table('t_stocks')  
->select('product_id',  
DB::raw('sum(quantity) as quantity'))  
->groupBy('product_id')  
->having('quantity', '>', 1);
```

# 商品一覧クエリ その4

前ページの \$stocksをサブクエリとして設定  
products、shops、stocksをjoin句で紐付けて  
where句で is\_sellingがtrue かの条件指定

```
$products = DB::table('products')
    ->joinSub($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();
```

# 商品一覧クエリ その5

Eloquent->クエリビルダに変更したためselectで指定

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

略

```
->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')
```

略 (前ページのwhere句)

```
->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

# 商品の詳細

ルート

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

コントローラ

```
public function show($id){  
    $product = Product::findOrFail($id);  
    return view('user.show', compact('product'));  
}
```

ビュー

user/show.blade.php



# 商品の詳細

---

ビュー

`user/show.blade.php`

TailblocksのEcommerceを参考に  
レイアウト調整

# Swiper.js

# Swiper.js その1

多機能・レスポンシブ対応・VanillaJS

<https://swiperjs.com/>

```
npm install swiper
```

JS記入場所

```
resources/js/swiper.js
```

Laravel Mixに追記 webpack.mix.js

```
mix.js('resources/js/app.js', 'public/js')
```

```
    .('resources/js/swiper.js', 'public/js')
```

//元ファイル、出力先(ファイル名は同じ)

# Swiper.js その2

CSSも調整

resources/css/swiper.css

app.cssで@import

CSS・JS調整後は npm run devでコンパイル

読み込む方法

```
<script src="{{ mix('js/swiper.js') }}"></script>
```

app.jsは全ページで読み込まれる

Swiper.jsとして個別で読み込む事で

app.jsを軽くしつつ他ページ表示も遅くならない

# Swiper.js その3

---

```
<div class="swiper-slide">
  @if($product->imageFirst->filename !== null)
    
  @else
    <img src="">
  @endif
</div>
```

# 店舖情報

# 店舗情報 その1

---

Shopのダミーデータに  
画像が含まれていないので追記  
README.mdにも追記  
(Public/sample1.jpgを  
Storage/app/public/shopsフォルダに  
配置してほしい)



# 店舗情報 その2

```
<div class="border-t border-gray-400 my-8"></div>
<div>この商品を販売しているショップ</div>
<div>{{ $product->shop->name }}</div>
<div>画像は@ifで設定
  
</div>
<div><button type="button">ボタン</button></div>
```



# 店舗情報 その3 詳細を見る

---

詳細を見るはMicromodal.jsで対応

HTML/CSSのサンプル

[https://gist.github.com/ghosh/  
4f94cf497d7090359a5c9f81caf60699](https://gist.github.com/ghosh/4f94cf497d7090359a5c9f81caf60699)

# 店舗情報 その4 在庫情報

コントローラ

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

```
if($quantity > 9){  
    $quantity = 9;  
}
```

ビュー

```
<select name="quantity">  
    @for($i = 1; $i <= $quantity; $i++)  
        <option value="{{ $i }}">{{ $i }}</option>  
    @endfor  
</select>
```

# 店舗情報 その5 在庫情報

在庫情報

ビュー

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

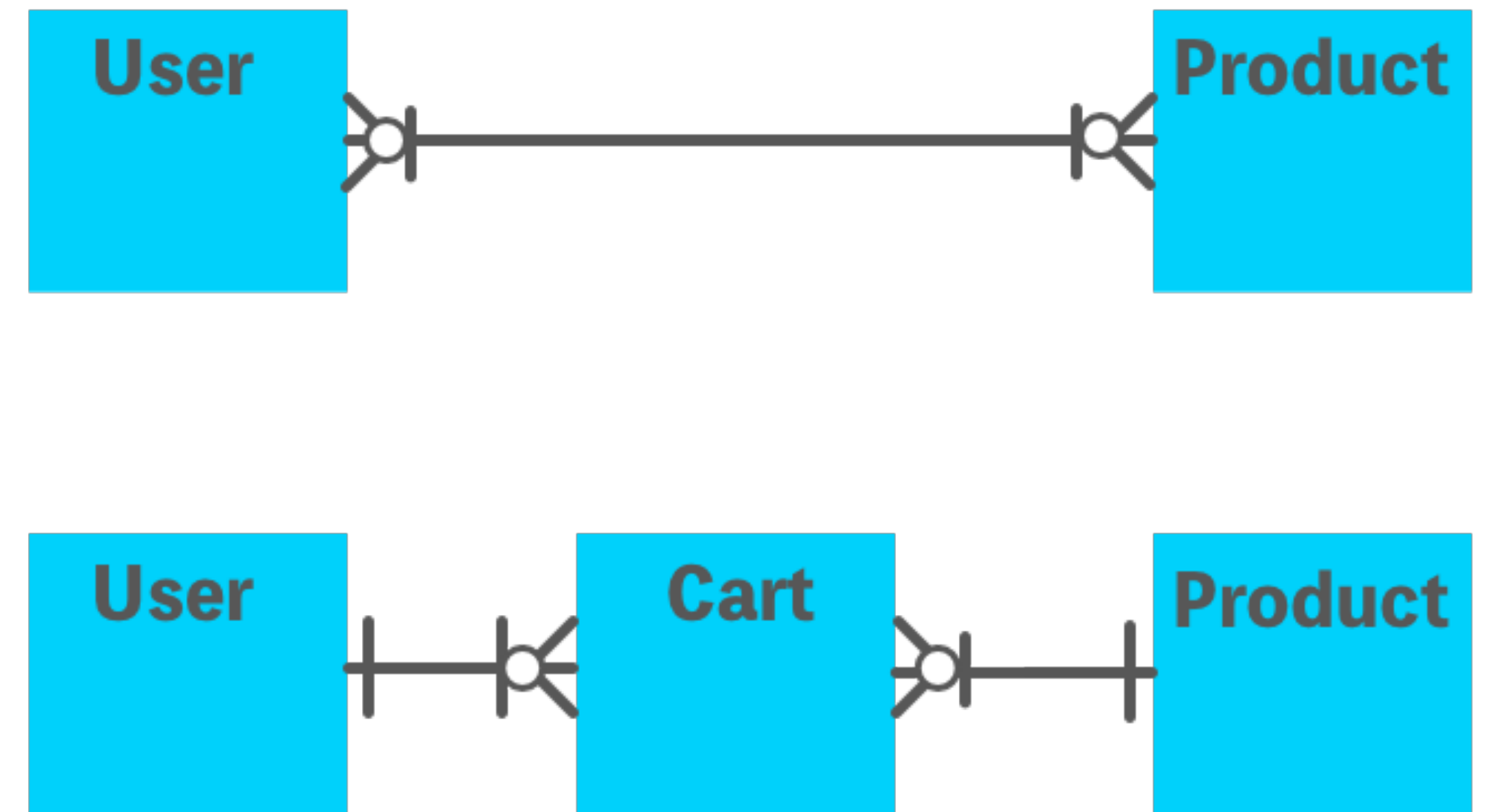
```
if($quantity > 9){  
    $quantity = 9;  
}
```

カート

# カート 多対多

複数のユーザーが  
複数の商品を持てる  
・ ・ 多対多

中間(pivot)テーブルをはさみ  
1対多



自動で生成するならproduct\_user(アルファベット順)  
今回はCartというモデルを作成し設定

# カート 多対多

---

php artisan make:model Cart -m

マイグレーションの外部キー制約はproductsなどを参考

モデル

```
protected $fillable = [  
    'user_id',  
    'product_id',  
    'quantity',  
];
```

# カート 多対多 リレーション

Userモデル

```
public function products()
{
    return $this->belongsToMany(Product::class, 'carts')
        ->withPivot(['id', 'quantity']);
    // 第2引数で中間テーブル名
    // 中間テーブルのカラム取得
    // デフォルトでは関連付けるカラム(user_idと
    product_id)のみ取得
}
```

# カート 多対多 リレーション

---

Productモデル

```
public function users()  
{  
    return $this->belongsToMany(User::class,  
'carts')  
        ->withPivot(['id', 'quantity']);  
}
```



カートに追加

# カートに追加

ルート

```
Route::prefix('cart')->middleware('auth:users')->group(function(){  
    Route::post('add', [CartController::class, 'add'])->name('cart.add');  
});
```

ビュー

User/show.blade.php のaタグにリンク記載

```
<form method="post" action="{{ route('user.cart.add') }}">  
    @csrf  
    略 (在庫情報)  
    <button>カートに追加</button>  
    <input type="hidden" name="product_id" value="{{ $product->id }}">  
</form>
```

コントローラ php artisan make:controller User/CartController

# カートに追加

```
public function add(Request $request)
{
    $itemInCart = Cart::where('user_id', Auth::id())
        ->where('product_id', $request->product_id)->first(); //カートに商品があるか確認

    if($itemInCart){
        $itemInCart->quantity += $request->quantity; //あれば数量を追加
        $itemInCart->save();
    } else {
        Cart::create([ // なければ新規作成
            'user_id' => Auth::id(),
            'product_id' => $request->product_id,
            'quantity' => $request->quantity
        ]);
    }
    return redirect()->route('user.cart.index');
}
```

# カート内 表示

# カート内を表示

ルート

```
Route::prefix('cart')->middleware('auth:users')->group(function(){  
    Route::get('/', [CartController::class, 'index'])->name('cart.index');  
});
```

コントローラ index()

```
$user = User::findOrFail(Auth::id());  
$products = $user->products; // 多対多のリレーション  
$totalPrice = 0;
```

```
foreach($products as $product){  
    $totalPrice += $product->price * $product->pivot->quantity;  
}
```

# カート内を表示

ビュー User/cart.blade.php

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

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

```
        <div>画像</div>
```

```
        <div>{{ $product->name }}</div>
```

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

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

```
    @endforeach
```

```
    合計金額: {{ $totalPrice }}
```

```
@else
```

```
    カートに商品が入っていません。
```

```
@endif
```



# カート内 削除

# カート内を削除

---

ルート

```
Route::prefix('cart')->middleware('auth:users')->group(function(){  
    Route::post('delete/{item}', [CartController::class, 'delete'])->  
name('cart.delete');  
});
```

コントローラ delete()

```
{  
    Cart::where('product_id', $id)  
->where('user_id', Auth::id())->delete();  
  
    return redirect()->route('user.cart.index');  
}
```



# カート内を削除

ビュー側

```
<form method="post" action="{{route('user.cart.delete',  
['item' => $product->id ])}}">
```

```
@csrf
```

```
<button></button>
```

```
</form>
```

アイコン

heroicons <https://heroicons.com/>



# Stripe

# Stripe



API型決済ライブラリ 手数料 3.6%

テストモードあり  
会員登録後 APIキー発行

<https://stripe.com/jp>

<https://stripe.com/docs>

# Stripe

---

APIキー(公開鍵・秘密鍵)を  
.envファイルに追記

```
STRIPE_PUBLIC_KEY=""  
STRIPE_SECRET_KEY=""
```

# Stripeの使用方法

---

Laravel Cashier (定期支払い向け)

Stripeが発行しているライブラリ

`composer require stripe/stripe-php`

<https://github.com/stripe/stripe-php>

# Stripe 決済処理

# Stripe決済処理

---

ルーティング

```
Route::prefix('cart')->middleware('auth:users')->group(function(){  
    Route::get('checkout', [CartController::class, 'checkout'])->name('cart.checkout');  
});
```



# Stripe決済処理 コントローラ1

Stripeに渡すパラメータを設定

<https://stripe.com/docs/api/checkout/sessions/create>

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

```
$line_items = [];  
foreach($user->products as $product){  
    $line_item = [  
        'name' => $product->name,  
        'description' => $product->description,  
        'amount' => $product->price,  
        'currency' => 'jpy',  
        'quantity' => $product->pivot->quantity,  
    ];  
    array_push($line_items, $line_item);  
}
```

# Stripe決済処理 コントローラ2

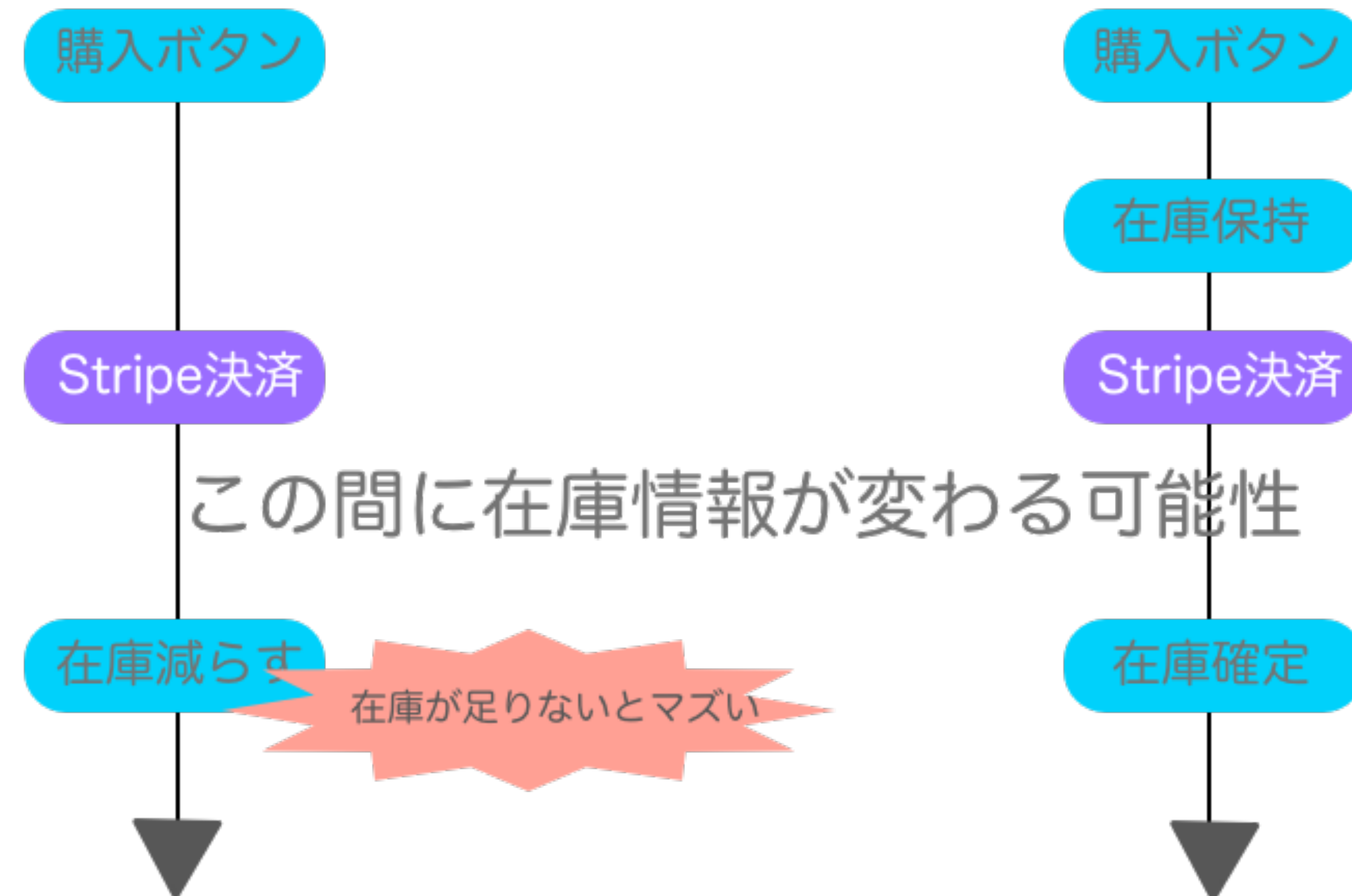
Stripeへ渡すSession情報

<https://stripe.com/docs/checkout/integration-builder>

```
\Stripe\Stripe::setApiKey(env('STRIPE_SECRET_KEY'));  
$session = \Stripe\Checkout\Session::create([  
    'payment_method_types' => ['card'],  
    'line_items'           => [$line_items],  
    'mode'                 => 'payment',  
    'success_url'          => route('user.items.index'),  
    'cancel_url'           => route('cart.cart.index'),  
]);
```

```
$publicKey = env('STRIPE_PUBLIC_KEY');  
return view('user.checkout',  
    compact('session', 'publicKey'));
```

# Stripe決済処理 在庫処理



カード情報入力で時間がかかる  
事前に在庫を保持

# Stripe決済処理 コントローラ3

在庫確認し決済前に在庫を減らしておく

```
$line_items = [];  
foreach($products as $product){  
    $quantity = "";  
    $quantity = Stock::where('product_id', $product->id)-  
>sum('quantity');  
  
    if($product->pivot->quantity > $quantity ){  
        return redirect()->route('user.cart.index');  
    } else {  
        略  
    }  
}
```

# Stripe決済処理 コントローラ4

```
foreach($products as $product)
{
    Stock::create([
        'product_id' => $product->id,
        'type' => \Constant::PRODUCT_LIST['reduce'],
        'quantity' => $product->pivot->quantity * -1
    ]);
}
```

# Stripe決済処理 ビュー 1

User/cart.blade.php

```
<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')}}'"  
>購入する
```

```
  </button>
```

```
</div>
```

# Stripe決済処理 ビュー 2

User/checkout.blade.php

<https://stripe.com/docs/checkout/integration-builder>

<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 }}'

const stripe = Stripe(publicKey)

window.onload = function() {

stripe.redirectToCheckout({

sessionId: '{{ \$session->id }}'

}).then(function (result) {

window.location.href = '{{ route('user.cart.index') }}';

});

}

</script>



# Stripe決済処理 ダミー情報

---

ダミーのカード情報

4242 4242 4242 4242 など

# Stripe 決済 成功・キャンセル 処理

# Stripe決済処理 成功時

---

カートから商品を削除

ルート

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

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

# Stripe決済処理 成功時

```
コントローラ public function success(){  
    Cart::where('user_id', Auth::id()->delete());  
  
    return redirect()->route('user.items.index');  
}
```

```
public function checkout(){  
    略  
    'success_url'      => route('user.cart.success'),  
    略  
}
```

# Stripe決済処理 キャンセル時

在庫を戻す

ルート

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

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

# Stripe決済処理 キャンセル時


```
コントローラ public function cancel(){
    $user = User::findOrFail(Auth::id());
    foreach($user->products as $product)
    {
        Stock::create([
            'product_id' => $product->id,
            'type' => \Constant::PRODUCT_LIST['add'],
            'quantity' => $product->pivot->quantity
        ]);
    }
}
```

```
public function checkout(){
    略
    'cancel_url' => route('user.cart.cancel'),
    略
}
```

# Stripe決済処理 キャンセル時

```
ビュー (user/checkout.blade.php)  
window.onload = function() {  
    stripe.redirectToCheckout({  
        sessionId: '{{ $session->id }}'  
    }).then(function (result) {  
        window.location.href =  
'{{ route('user.cart.cancel') }}';  
    });  
}
```





# Git push & pull request

# Git push & pull request

---

README.mdにインストール方法を追記後

ここまでの内容をgit push しつつ  
GitHub側でPull Requestして、  
mainブランチにマージしておきます。