



顧客情報 購買履歴その1



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 + headless ui

TAILBLOCKS <https://tailblocks.cc/>

headless ui <https://headlessui.com/>

機能設計 ・ ・ 機能名、処理内容

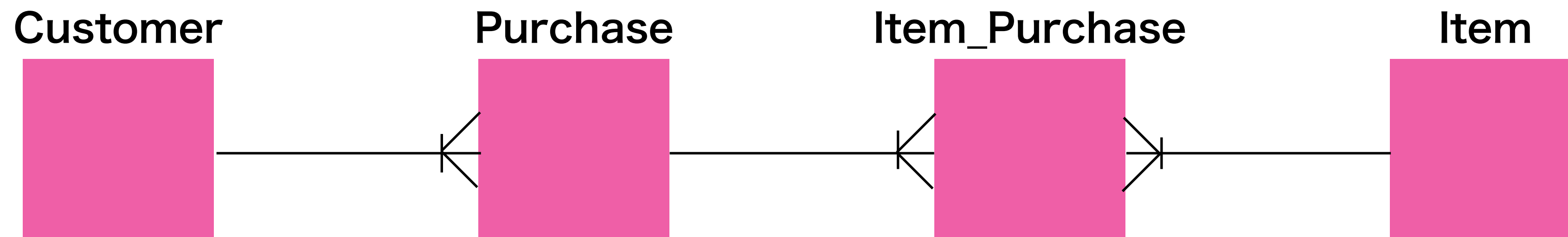
データ設計 ・ ・ テーブル設計、ER図

UI フレームワーク

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/1QsXc6rCgfNmgMDTccakNDyW2hKQzzTFqi26pM4h5BTs/edit?usp=sharing>

ER図





顧客情報

Customer

Customers 下準備



php artisan make:model Customer -a

-aはallの略

モデル、ファクトリー、マイグレーション、
シーダー、リクエスト、リソースコントロー
ラ、ポリシーをまとめて生成

マイグレーションファイル

```
databases/migrations/xxx_create_customers_table.php
public function up()
{
    Schema::create('customers', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('kana');
        $table->string('tel')->unique();
        $table->string('email');
        $table->string('postcode');
        $table->string('address');
        $table->date('birthday')->nullable();
        $table->tinyInteger('gender'); // 0男性, 1女性、2その他
        $table->text('memo')->nullable();
        $table->timestamps();
    });
}
```

php artisan migrate と実行し customers テーブルが増えていればOK

ダミーデータの日本語対応



config/app.php

'faker_locale' => 'ja_JP'; // 日本語対応

ダミーデータ生成

FakerとFactoryを使うと大量のデータを簡単に作ることができる

database/factories/CustomerFactory.php

```
public function definition()
```

```
{
    return [
        'name' => $this->faker->name,
        'kana' => $this->faker->kanaName,
        'tel' => $this->faker->phoneNumber,
        'email' => $this->faker->email,
        'postcode' => $this->faker->postcode,
        'address' => $this->faker->address,
        'birthday' => $this->faker->dateTime,
        'gender' => $this->faker->numberBetween(0, 2),
        'memo' => $this->faker->realText(50),
    ];
}
```

Fakerは「Faker チートシート」で検索

<https://qiita.com/tosite0345/items/1d47961947a6770053af>

DatabaseSeeder.php

```
public function run()
{
    $this->call([
        UserSeeder::class,
        ItemSeeder::class
    ]);

    \App\Models\Customer::factory(1000)->create();
}
```

php artisan migrate:fresh --seed と実行し
ダミーデータ1000件入っていればOK

ダミーデータ修正

電話番号-> ハイフンを削除

住所-> 郵便番号と半角スペースをカット

CustomerFactory.php

```
public function definition()  
{
```

```
    $tel = str_replace('-', '', $this->faker->phoneNumber);
```

```
    $address = mb_substr($this->faker->address, 9);
```

```
    return [  
        略
```

```
        'tel' => $tel, // 変更
```

```
        'address' => $address, // 変更
```

```
        略
```

```
    ]
```




顧客一覽 Index

顧客一覧

ルート->コントローラ->ビュー

routes/web.php

```
use App\Http\Controllers\CustomerController;
```

```
Route::resource('customers', CustomerController::class)  
->middleware(['auth', 'verified']);
```

CustomerController.php

```
use Inertia\Inertia;
```

```
public function index(){  
    return Inertia::render('Customers/Index', [  
        'customers' => Customer::select('id', 'name', 'kana', 'tel')->get();  
    ]);  
}
```


顧客一覧

Pages/Items/Index.vueをコピーして
Pages/Customers/Index.vue を作成
defineProps({ customers: Array })


商品 -> 顧客

items -> customers にそれぞれ変更

テーブル列・・・id、名前、カナ、電話番号


ShowのLinkは削除

Layouts/Authenticated.vue にナビ追加



ページネーション

ページネーション



参考URL

<https://www.positronx.io/laravel-vue-inertia-pagination-integration-tutorial/>

CustomerController.php

->get() を ->paginate(50) に変更

ページネーション

Vue側でObjectで受け取って中身を見してみる

```
<script setup>
```

```
import Pagination from '@/Components/Pagination.vue'
```

```
import { onMounted } from 'vue'
```

```
const props = defineProps({  
  'customers': Object  
})
```

```
onMounted(() => { //ページが表示されたタイミングで実行  
  console.log(props.customers)  
})
```

```
</script>
```

console.logの表示結果

Index.vue:10

[illegible]

Customers/Index.vue

必要な箇所の抜粋

```
<script setup>
```

```
import Pagination from '@/Components/Pagination.vue'
```

```
defineProps({  
  'customers': Object  
})
```

```
</script>
```

```
<template>
```

```
  <ul v-for="customer in customers.data" :key="customer.id">
```

```
    <li>{{ customer.id }}</li>
```

```
    <li>{{ customer.name }}</li>
```


```
  </ul>
```

```
  <Pagination class="mt-6" :links="customers.links"></Pagination>
```

```
</template>
```

Pagination.vue

```
<script setup>
import { Link } from '@inertiajs/inertia-vue3';
defineProps({ links: Array })
</script>
<template>
  <div v-if="links.length > 3">
    <div class="flex flex-wrap -mb-1">
      <template v-for="(link, index) in links" :key="index">
        <div v-if="link.url === null" class="mr-1 mb-1 px-4 py-3 text-sm leading-4
text-gray-400 border rounded"
          v-html="link.label" />
        <Link v-else
          class="mr-1 mb-1 px-4 py-3 text-sm leading-4 border rounded
hover:bg-white focus:border-indigo-500 focus:text-indigo-500"
          :class="{ 'bg-blue-700 text-white': link.active }" :href="link.url" v-
html="link.label" />
      </template>
    </div>
  </div>
</template>
```

ページネーション 顧客一覧に組み込む

Customers/Index.vue

```
<script setup>
import { Pagination } from '@Components/Pagination.vue';
defineProps({ customers: Object })
</script>
<template>
  略
  <tr v-for="customer in
customers.data" :key="customer.id">
  </tr>
  略
  <Pagination :links="customers.links"></Pagination>
</section>
</template>
```



顧客検索機能

顧客検索機能

顧客を探しやすくするために検索機能をつける
条件: カナ または 電話番号 前方一致

検索フォームを追加
条件

- ・ 入力されている場合
検索ヒットした場合->結果を表示
検索ヒットしない場合->全件表示
- ・ 入力されていない場合->全件表示

クエリスコープ(ローカルスコープ)

モデル用のメソッドをつくれる

app\Models\Customer;

```
public function scopeSearchCustomers($query, $input = null)
{
    if(!empty($input)){
        if(Customer::where('kana', 'like', $input . '%' )
->orWhere('tel', 'like', $input . '%')->exists())
        {
            return $query->where('kana', 'like', $input . '%' )
->orWhere('tel', 'like', $input . '%');
        }
    }
}
```

CustomerController@index

```
use Illuminate\Http\Request;
```

```
// ビューから渡ってくる情報は $requestで受け取れる
```

```
public function index(Request $request)
```

```
{
```

```
    $customers =
```

```
    Customer::searchCustomers($request->search)
```

```
    ->select('id', 'name', 'kana', 'tel')->paginate(50);
```

```
    return Inertia::render('Customers/Index', [
```

```
        'customers' => $customers
```

```
    ]);
```

```
}
```

Customer/Index.vue

```
import { ref } from 'vue'
import { Inertia } from '@inertiajs/inertia';

const search = ref('')

// ref の値を取得するには .valueが必要
const searchCustomers = () => {
  Inertia.get(route('customers.index', { search: search.value }))
}

<div>
  <input type="text" name="search" v-model="search">
  <button class="bg-blue-300 text-white py-2 px-2"
    @click="searchCustomers">検索</button>
</div>
```




顧客新規登録

顧客新規登録

CustomerController.php

```
public function create()
{
    return Inertia::render('Customers/Create');
}
```

Customers/Create.vue

Items/Create.vueをコピーして必要な情報を追記

```
const form = reactive({
  name: null, kana: null, tel: null, email: null, postcode: null,
  address: null, birthday: null, gender: null, memo: null
})
```

inputタグのtype ・ ・ telはtel、emailはemail、postcodeはnumber
birthdayはdate、genderはradio

全てv-model=form.xxx とつけてリアクティブにする



顧客保存

顧客保存

app\Models\Customer.php

```
protected $fillable = ['name','kana','tel','email','postcode','address', 'birthday','gender', 'memo'];
```

CustomerController.php

```
public function store(StoreCustomerRequest $request)
{
    Customer::create([
        'name' => $request->name,
        'kana' => $request->kana,
        'tel' => $request->tel,
        'email' => $request->email,
        'postcode' => $request->postcode,
        'address' => $request->address,
        'birthday' => $request->birthday,
        'gender' => $request->gender,
        'memo' => $request->memo,
    ]);

    return to_route('customers.index')
        ->with([
            'message' => '登録しました。',
            'status' => 'success'
        ]);
}
```

顧客保存 バリデーション

app\Http\Requests\StoreCustomerRequest.php

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

```
public function rules()  
{
```

```
    return [  
        'name' => ['required', 'max:50'],
```

```
        'kana' => ['required', 'regex:/^[ア-ヴ]+$/u','max:50'],
```

```
        'tel' => ['required', 'max:20', 'unique:customers,tel'],
```

```
        'email' => ['required', 'email', 'max:255', 'unique:customers,email'],
```

```
        'postcode' => ['required', 'max:7'],
```

```
        'address' => ['required', 'max:100'],
```

```
        'birthday' => ['date'],
```

```
        'gender' => ['required'],
```

```
        'memo' => ['max:1000'],
```

```
    ];
```

```
}  
}
```

バリデーション日本語対応

lang/ja/validation.php

```
'attributes' => [
```

```
    // 略
```

```
    'kana' => 'カナ',
```

```
    'tel' => '電話番号',
```

```
    'email' => 'メールアドレス',
```

```
    'postcode' => '郵便番号',
```

```
    'address' => '住所',
```

```
    'birthday' => '誕生日',
```

```
    'gender' => '性別'
```


```
],
```


ビュー側も変更必要

Pages/Customers/Create.vue

```
const storeCustomer = () => {  
  Inertia.post('/customers', form)  
}
```

```
<form @submit.prevent=storeCustomer>
```



郵便番号で 住所検索

郵便番号で住所検索

「Vue.js 郵便番号検索」でググる (方法はいくつかあります)
ライブラリインストール
npm i yubinbango-core2@^0.6.3

Pages/Customers/Create.vue

```
import { Core as YubinBangoCore } from "yubinbango-core2";
```

```
// 数字を文字に変換 第1引数が郵便番号、第2がコールバックで引数に住所  
const fetchAddress = () => {  
  new YubinBangoCore(String(form.postcode), (value) => {  
    form.address = value.region + value.locality + value.street  
  })  
}
```

```
<input name="postcode" @change="fetchAddress">略
```


Show以下は省略いたします

商品の時と同じような内容になるため、

顧客情報の

Show, Edit, Update, Destroyは
省略いたします。

やってみたい方はぜひ挑戦してみてください。



購買情報

Purchase

購買情報 Purchase

php artisan make:model Purchase -a

モデル、ファクトリ、マイグレーション、シー
ダー、リクエスト、リソースコントローラー、
ポリシー をまとめて生成

マイグレーション

foreignIdで外部キー設定

databases/migrations/xxx_purchases_table.php

```
public function up()
{
    Schema::create('purchases', function (Blueprint $table) {
        $table->id();
        $table->foreignId('customer_id')->constrained()-
>onUpdate('cascade');
        $table->boolean('status');
        $table->timestamps();
    });
}
```

モデル



app\Models\Purchase.php

```
protected $fillable = [  
    'customer_id',  
    'status',  
];
```

モデル:リレーションの設定

Customer 1対多 Purchase

app\Models\Customer.php

```
use app\Models\Purchase;  
public function purchases()  
{  
    return $this->hasMany(Purchase::class);  
}
```

app\Models\Purchase.php

```
use app\Models\Customer;  
public function customer()  
{  
    return $this->belongsTo(Customer::class);  
}
```


ダミーデータ

databases/factories/PurchaseFactory.php

```
use App\Models\Customer;

public function definition()
{

    return [
        'customer_id' => rand(1, Customer::count()),
        'status' => $this->faker->boolean,
    ];
}
```

databases/seeder/DatabaseSeeder.php

```
\App\Models\Purchase::factory(100)->create();
```

php artisan migrate:fresh --seed でダミー投入



中間テーブル ItemPurchase

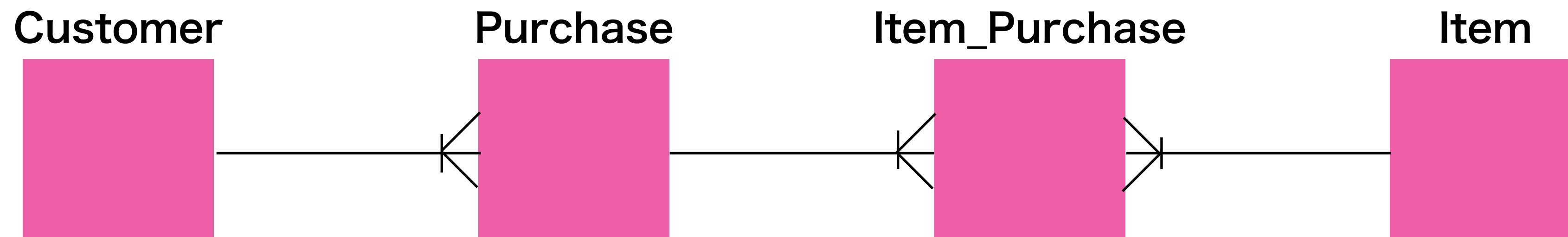
中間テーブルの作成

1回の利用で複数の商品・サービスが購入できると想定

中間テーブル(Pivotテーブル)を作成
アルファベット順(単数系)で作っておくと
Laravel側で認識してくれるので楽

```
php artisan make:migration create_item_purchase_table
```

ER図



マイグレーション

databases/migrations/xxx_item_purchase_table.php

```
public function up()
{
    Schema::create('item_purchase', function (Blueprint
$table) {
        $table->id();
        $table->foreignId('item_id')->constrained()-
>onUpdate('cascade');
        $table->foreignId('purchase_id')->constrained()-
>onUpdate('cascade');
        $table->integer('quantity');
        $table->timestamps();
    });
}
```

多対多 リレーション

中間テーブル内の情報を取得したいため、 withPivotも使用

App\Models\Purchase.php

```
use app\Models\Item;  
public function items()  
{  
    return $this->belongsToMany(Item::class)  
        ->withPivot('quantity');  
}
```

App\Models\Item.php

```
use app\Models\Purchase;  
public function purchases()  
{  
    return $this->belongsToMany(Purchase::class)  
        ->withPivot('quantity');  
}
```

ダミーデータ

Purchaseを登録時 中間テーブルにも同時に登録する
(1件の購入情報に1件～3件の商品情報を登録とする)

each・・・1件ずつ処理

attach・・・中間テーブルに情報登録

外部キー以外で中間テーブルに情報追加するには第2引数に書く

databases/seeder/DatabaseSeeder.php

```
use AppModels\Purchase;
```

略

```
$items = \App\Models\Item::all();
```

```
Purchase::factory(100)->create()
```

```
->each(function(Purchase $purchase) use ($items){
```

```
    $purchase->items()->attach(
```

```
        $items->random(rand(1,3))->pluck('id')->toArray(),
```

```
        // 1～3個のitemをpurchaseにランダムに紐づけ
```

```
        ['quantity' => rand(1, 5) ] ); });
```




購入画面

購買関連

ルート->コントローラ->ビュー

routes/web.php

```
use App\Http\Controllers\PurchaseController;
```

```
Route::resource('purchases', PurchaseController::class)  
->middleware(['auth', 'verified']);
```

PurchaseController

```
use Inertia\Inertia;
use App\Models\Customer;
use App\Models\Item;

public function create()
{
    $customers = Customer::select('id', 'name', 'kana')->get();
    $items = Item::select('id', 'name', 'price')->where('is_selling',
true)->get();

    return Inertia::render('Purchases/Create', [
        'customers' => $customers,
        'items' => $items
    ]);
}
```


ナビゲーションにも追記



Layouts/Authenticated.vue
購入画面 としてリンク追記

Purchases/Create.vue

フォームの項目

1. 購入日 ・ ・ 当日を初期値
2. 会員名 ・ ・ Customerから取得
3. 商品情報 ・ ・ (Id, 名前、金額、数量、小計)
(数量を変更すると小計 ・ 合計が計算される)
4. 合計金額

Id	名前	金額	数量	小計
1	カット	6000	1	6000
2	カラー	8000	1	8000
3	パーマ	13000	0	0

1. 購入日 その1

初期表示は当日とする JSのDateから取得
js/common.js

```
const getToday = () => {  
  const today = new Date();  
  const yyyy = today.getFullYear();  
  const mm = ("0" + (today.getMonth() + 1)).slice(-2);  
  const dd = ("0" + today.getDate()).slice(-2);  
  return yyyy + '-' + mm + '-' + dd;  
}
```

```
export { getToday }
```


1. 購入日 その2

Pages/Purchases/Create.vue

```
<script setup>
```

```
import { getToday } from '@/common' // 別ファイルをインポート
```

```
import { onMounted, reactive } from 'vue'
```

```
onMounted(() => { // ページ読み込み後 即座に実行  
  form.date = getToday()  
})
```

```
const form = reactive({ date: null })
```

```
</script>
```

```
<template>
```

```
  日付<br>
```

```
  <input type="date" name="date" v-model="form.date">
```

```
</template>
```

2. 会員名

propsで渡ってきた値をv-forで回している

Pages/Purchases/Create.vue

```
<script setup>
```

略

```
const props = defineProps({ 'customers': Array })
```

```
const form = reactive({ customer_id: null })
```

```
</script>
```

```
<template>
```

```
  会員名<br>
```

```
  <select name="customer" v-model="form.customer_id">
```

```
    <option v-for="customer in  
customers" :value="customer.id" :key="customer.id">
```

```
      {{ customer.id }} : {{ customer.name }}
```

```
    </option>
```

```
  </select>
```

```
</template>
```

3. 商品情報 その1

propsのままだと変更できないので新たに配列を作って追加
販売中のItemをv-forで全て表示 数量は初期値0

Pages/Purchases/Create.vue

```
<script setup>
```

```
import { ref } from 'vue'
```

```
const props = defineProps({ 'items': Array })
```

```
const itemList = ref([]) // リアクティブな配列を準備
```

```
onMounted(() => {
```

```
  props.items.forEach( item => { // 配列を1つずつ処理
```

```
    itemList.value.push({ // 配列に1つずつ追加
```

```
      id: item.id, name: item.name, price: item.price, quantity: 0 })
```

```
  })
```

```
})
```

```
const quantity = [ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"] // option用
```


3. 商品情報 その2

Pages/Purchases/Create.vue

商品・サービス

<table>

略

<tbody>

<tr v-for="item in itemList" >

<td>{{ item.id }}</td>

<td>{{ item.name }}</td>

<td>{{ item.price }}</td>

<td>

<select name="quantity" v-model="item.quantity">

<option v-for="q in quantity" :value="q">{{ q }}</option>

</select>

</td>

<td>

{{ item.price * item.quantity }}

</td>

</tr>

</tbody>

</table>

4. 合計金額

Computedで変更があり次第再計算する

Computedはreturnが必須

Pages/Purchases/Create.vue

```
<script setup>
```

```
import { computed } from 'vue'
```

```
const totalPrice = computed(() => {  
  let total = 0  
  itemList.value.forEach( item => {  
    total += item.price * item.quantity  
  })  
  return total  
})
```

```
</script>
```

```
<template>
```

```
  合計 {{ totalPrice }} 円
```

```
</template>
```