




購買履歴その2



購入画面->
保存処理

保存処理 View側

Controllerに渡す情報を整理する(商品数をformに追加する)

```
import { Inertia } from '@inertiajs/inertia'
const form = reactive({ status: true, items: [] })

const storePurchase = () => {
  itemList.value.forEach( item => {
    if( item.quantity > 0 ) // 0より大きいものだけ追加
      form.items.push({ id : item.id, quantity: item.quantity })
  })

  Inertia.post(route('purchases.store'), form)
}
```

```
<form @submit.prevent="storePurchase">
  <button>登録する</button>
</form>
```

保存処理 Laravel側

trueにして通信できるようにする

app\Http\Requests\StorePurchaseRequest.php

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

```
public function rules()  
{ return [ 'customer_id' => ['required'] ]; } // 顧客設定必須
```

まずはddで内容確認

app\Http\Controllers\PurchaseController.php

```
public function store(StorePurchaseRequest $request)  
{  
    dd($request);  
}
```


保存処理 (後程トランザクション)

App\Http\Controllers\PurchaseController.php

```
public function store(StorePurchaseRequest $request)
{
    $purchase = Purchase::create([
        'customer_id' => $request->customer_id,
        'status' => $request->status,
    ]);

    foreach($request->items as $item){
        $purchase->items()->attach( $purchase->id, [
            'item_id' => $item['id'],
            'quantity' => $item['quantity']
        ]);
    }
    return to_route('dashboard');
```

保存処理（トランザクション）

2つのテーブルに保存するので

途中でエラーなどあった場合はロールバックして戻せるようにする

App\Http\Controllers\PurchaseController.php

```
public function store(StorePurchaseRequest $request)
```

```
{
```

```
    DB::beginTransaction();
```

```
    try{
```

```
        // ここに処理を書く
```

```
        DB::commit();
```

```
        return to_route('dashboard');
```

```
    } catch(\Exception $e){
```

```
        DB::rollback();
```

```
}
```

購入画面のレイアウト調整

Purchases/Create.vue

Items/Create.vueを参考に
レイアウトを整える

tableはItems/Index.vueから拝借



micromodal.js

micromodal.js



シンプルなモーダルウィンドウのライブラリ

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

```
npm i micromodal@0.4.10 --save
```

package.jsonに追加される

micromodal.js 準備

resources/js/micromodal.js

```
import MicroModal from 'micromodal'  
MicroModal.init({  
  disableScroll: true  
})
```

resources/css/micromodal.css

公式ページのCSSを掲載

resources/js/app.js

```
import './micromodal'  
import '../css/micromodal.css'
```


micromodal.js View側

resources/Components/
MicroModal.vue

```
<script setup></setup>
```

```
<template>
```

公式ページからコピー

```
</template>
```

micromodal.js View側 加工

そのままだとうまく動かなかったので一部加工
micromodal-slideクラスを削除

**resources/Components/
MicroModal.vue**

```
<script setup>
import { ref } from 'vue'
const isShow = ref(false)
const toggleStatus = () => { isShow.value = !isShow.value }
</setup>
<template>
<div v-show="isShow" class="modal" id="modal-1">
  略
  <button type="button" @click="toggleStatus">ボタン</button>
</div>
```




顧客検索機能を 組み込む

APIの動作確認

Inertia::renderで返却するとページ毎 再描画される

顧客の箇所だけ更新したいのでAPIで対応 Sanctumの機能を使う

app/Http/Kernel.php

```
'api' => [ \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,  
// コメントアウトを解除
```

.env

SANCTUM_STATEFUL_DOMAINS=localhost:8000

SESSION_DOMAIN=localhost

関連記事 <https://qiita.com/ksrnnb/items/d2b73a6bf7dccde90446>

API関連ファイル

App\Providers\RouteServiceProvider.php

config/sanctum.php

config/session.php

routes/api.php

axiosの動作確認

Components/MicroModal.js

```
import axios from 'axios'
import { onMounted } from 'vue'
<script setup>
onMounted(() => {
  axios.get('/api/user')
    .then( res => {
      console.log(res)
    })
})
</script>
```

Ajax(非同期通信)の設定

以前作っていたsearchCustomersを流用する

routes/api.php

```
use App\Models\Customer;
```

```
Route::middleware('auth:sanctum')->get('/  
searchCustomers', function (Request $request){  
    return Customer::searchCustomers($request->search)  
        ->select('id', 'name', 'kana', 'tel')->paginate(50);  
});
```


Async-await 非同期関数

DBアクセスし取得するまでに若干時間がかかるので
async-awaitを使って、取得後に表示させる

Components/MicroModal.vue

```
import { reactive } from 'vue'
const search = ref("")
const customers = reactive({})
const searchCustomers = async () => {
  try{
    await axios.get(`/api/searchCustomers/?search=${search.value}`)
    .then( res => {
      console.log(res.data)
      customers.value = res.data
    })
    toggleStatus()
  } catch (e){
    console.log(e.message)
  }
}
<input name="customer" v-model="search">
<button type="button" @click="searchCustomers">検索する</button>
```

顧客情報をMicroModalに追加

Pages/Customers/Index.vue

顧客情報のtableをコピー

Components/MicroModal.vue

に貼り付け (Paginationはなし)

```
<main class="modal__content">
```

```
<div v-if="customers.value" > // 値が入っていたら表示  
略
```

```
<tbody>
```

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

```
<td class="border-b-2 border-gray-200 px-4 py-3">
```

```
</td>
```


CSS調整

Pages/Purchases/Create.vue

<div class="relative"> のrelativeを削除

<label>合計金額</label>

<select>毎削除

props customers などとも削除

コントローラー側の\$customersも削除

Components/MicroModal.vue

<div class="modal__container w-2/3">

w-2/3を追加

inputタグ、 buttonタグにclass追加

Emitで 子 -> 親 に情報アップ

Components/Create.vue (親)

```
const setCustomerId = id => { form.customer_id = id }
```

```
<MicroModal @update:customerId="setCustomerId" />
```

Components/MicroModal.vue (子)

```
const emit = defineEmits(['update:customerId'])
```

```
const setCustomer = e => {  
  search.value = e.kana  
  emit('update:customerId', e.id)  
  toggleStatus()  
}
```

```
<button type="button" @click="setCustomer({id: customer.id, kana:  
customer.kana})" >{{ customer.id }}</button>
```



大量の ダミーデータ設定

ダミーデータ調整

顧客管理システムには大量のデータが保存される事が想定されるため、ダミーデータを少し変更します。

(PC環境によってダミー数を変更必要と思われます。

(もっと大量のデータ 10万～100万・・・などになるとコマンドなり mysqlimportなどを使う必要あるかもです。)

databases/factories/PurchaseFactory.php


public function definition()

```
{
    $decade = $this->faker->dateTimeThisDecade;
    $created_at = $decade->modify('+2 years');

    return [
        // 略
        'created_at' => $created_at
    ];
}
```

databases/seeder/DatabaseSeeder.php

Purchase::factory(30000);



購買履歴一覧 の準備

購買履歴 一覧

欲しい情報

購買id、顧客名、合計金額、ステータス、購入日

- ・サブクエリ

4つのテーブルをjoin

金額 * 数量 = 小計 を表示

- ・サブクエリで生成したテーブルをもとに

group by で 購買毎の合計金額を表示

購買履歴一覧 SQL版

サブクエリ

```
select purchases.id as id, item_purchase.id as pivot_id,  
customers.name as customer_name,  
items.price * item_purchase.quantity as subtotal,  
items.name as item_name, items.price as item_price,  
item_purchase.quantity, purchases.status,  
purchases.created_at, purchases.updated_at  
from purchases  
left join item_purchase on purchases.id =  
item_purchase.purchase_id  
left join items on item_purchase.item_id = items.id  
left join customers on purchases.customer_id =  
customers.id
```

購買履歴一覧 SQL版

サブクエリを元にした購買毎の合計金額表示

```
select history.id, history.customer_name, sum(history.subtotal) as
total, history.status, history.created_at
from (select purchases.id as id, item_purchase.id as pivot_id,
customers.name as customer_name,
items.price * item_purchase.quantity as subtotal,
items.name as item_name, items.price as item_price,
item_purchase.quantity, purchases.status, purchases.created_at,
purchases.updated_at
from purchases
left join item_purchase on purchases.id = item_purchase.purchase_id
left join items on item_purchase.item_id = items.id
left join customers on purchases.customer_id = customers.id
) as history
group by history.id
```




グローバルスコープ

グローバルスコープ

先々、サブクエリを元に
クエリスコープを何度か実施する事を想定
(ex 期間指定、日別、月別、商品別・・・)

今回のサブクエリ・・・4つのテーブルをjoinし小計を出している状態

新たにモデルを作成し
グローバルスコープ(モデル全体に適用)を使い
サブクエリ実施後の状態にする

グローバルスコープの準備

```
// グローバルスコープ用のファイル作成  
php artisan make:scope Subtotal
```

```
// モデル作成  
php artisan make:model Order  
App\Models\Order.php  
use App\Models\Scopes\Subtotal;
```

```
protected static function booted()  
{  
    static::addGlobalScope(new Subtotal);  
}
```

グローバルスコープ

4つのテーブルをjoinして小計をだしているクエリをfromSubに渡す

App\Models\Scopes\Subtotal.php

```
public function apply(Builder $builder, Model $model)
{
    $sql = 'select purchases.id as id
    , item_purchase.id as pivot_id
    , items.price * item_purchase.quantity as subtotal
    , customers.name as customer_name
    , items.name as item_name
    , items.price as item_price
    , item_purchase.quantity
    , purchases.status
    , purchases.created_at
    , purchases.updated_at
    from purchases
    left join item_purchase on purchases.id = item_purchase.purchase_id
    left join items on item_purchase.item_id = items.id
    left join customers on purchases.customer_id = customers.id
    '
    ;

    $builder->fromSub($sql, 'order_subtotals');
}
```




購買履歷一覽

Index

合計金額を計算

大量データをget()で取得しようとするともメモリ不足で表示できなくなるので注意

PurchaseController.php

```
use App\Models\Order;
```

```
public function index()
```

```
{
```

```
    // dd(Order::paginate(50));
```

```
    // 合計
```

```
    $orders = Order::groupBy('id')
```

```
    ->selectRaw('id, customer_name,
```

```
    sum(subtotal) as total, status, created_at' )
```

```
    ->paginate(50);
```

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

```
        'orders' = $orders
```

```
    ]);
```

```
}
```

購買履歷一覽 Vue側表示確認

Pages/Purchases/Index.vue

```
<script setup>
import { onMounted } from 'vue'
const props = defineProps({
  orders : Object
})

onMounted(() => {
  console.log(props.orders.data)
})

</script>
```


購買履歴一覧 Vue側

Pages/Purchases/Index.vue

Pages/Customers/Index.vueを流用

```
<v-for="order in props.orders.data">
<tr>
  <td>{{ order.id }}</td>
  <td>{{ order.customer_name }}</td>
  <td>{{ order.total }}</td>
  <td>{{ order.status }}</td>
  <td>{{ order.created_at }}</td>
</tr>
<Pagination :links="props.orders.links" ></Pagination>
```

Navigation追記

js/Layouts/Authenticated.vue に追記

```
<BreezeNavLink :href="route('purchases.index')"  
:active="route().current('purchases.index')">
```

購買履歴

```
</BreezeNavLink>
```

として追記



日時の表示変更

日時の表示変更

Vue.js propsで受け取った際に表示形式が変わっているので対応

dayjs

<https://day.js.org/docs/en/installation/installation>

```
npm i dayjs@1.11.5 --save
```

Pages/Purchases/Index.vue

```
import dayjs from 'dayjs'
```

```
{{ dayjs(order.created_at).format('YYYY-MM-DD HH:mm:ss') }}
```



購買履歷

詳細 show

購買履歴 詳細 show

表示したい内容

購買ID

顧客名

商品名・数・小計

合計金額

ステータス

購入日

(キャンセルした日 updated_at)

リンク <Link :href="route('purchases.show', { purchase:
order.id })">{{ order.id }}</Link>

購買履歴 詳細 show

App\Models\Purchase.php

1件分のデータを取得したいのでwhereで絞る、1件なのでget()でok

PurchaseController.php

// 小計

```
$items = Order::where('id', $purchase->id)
->get();
```

// 合計

```
$order = Order::groupBy('id')
->where('id', $purchase->id)
->selectRaw('id, customer_name, sum(subtotal) as total, status, created_at')-
>get();
```

```
//dd($subtotals, $order);
```

```
return Inertia::render('Purchases/Show', [
    'items' => $items, 'order' => $order ]);
```

購買履歴 詳細 show

Pages/Purchases/Show.vue

Create.vueをコピー。表示させるだけなので 不要な処理は削除

```
import dayjs from 'dayjs'
```

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

```
// 表示確認
```

```
onMounted(() => {  
  console.log(props.items[0])  
  console.log(props.order[0].created_at)  
})
```

inputタグをdivタグに変更

ステータス <div v-if="props.order[0].status == true" >未キャンセル</div>

キャンセル日 <div v-if="props.order[0].status == false" >{{

dayjs(props.order[0].updated_at).format('YYYY-MM-DD')}}未キャンセル</div>

購買履歴 詳細 show

商品IDを使っていたので

グローバルスコープに商品IDも追加

App\Models\Scopes\Subtotal.php

```
$sql = 'select purchases.id as id,  
        , items.id as item_id
```

略



購買履歷
編集 edit

購買履歴 編集 edit

どの項目を編集可能にするかは検討が必要

過去の金額・販売した商品を変更すると
データ分析・売上、商品の在庫数などにも影響する

キャンセルのみにする、権限をつける(gateなど)という
手もある

今回の仕様・・・ステータスと商品(数)は変更できる

購買履歴 編集 edit

Pages/Purchases/Show.vue

```
import { Link } from '@inertiajs/inertia-vue3'
```

```
<Link
```

```
  as="button" :href="route('purchases.edit', {  
    purchase: props.order[0].id })">編集する</
```

```
Link>
```


購買履歴 編集 edit (Laravel側)

中間テーブルの数量を確認し数が入っていれば反映させたい

Vue.js側でv-ifとv-forの組み合わせは非推奨なのでPHP側で配列をつくっておく

```
$purchase = Purchase::find($purchase->id); // 購買Idで指定
```

```
$allItems = Item::select('id', 'name', 'price')->get(); // 全商品を取得
```

```
$items = []; // 空の配列を用意
```

// 販売中の商品と中間テーブルを比較し、中間テーブルに数量があれば数量を取得、なければ0で設定

```
foreach($allItems as $allItem){
```

```
    $quantity = 0; // 数量初期値 0
```

```
    foreach($purchase->items as $item){ // 中間テーブルを1件ずつチェック
```

```
        if($allItem->id === $item->id){ // 同じidがあれば
```

```
            $quantity = $item->pivot->quantity; // 中間テーブルの数量を設定 }    }
```

```
        array_push($items, [
```

```
            'id' => $allItem->id, 'name' => $allItem->name,
```

```
            'price' => $allItem->price, 'quantity' => $quantity ]); }}
```

```
dd($items);
```

購買履歴 編集 edit (Laravel側)

Vue側に顧客ID, 顧客名も渡す

App\Models\Scopes\Subtotal.php
 , customers.id as customer_id 追加

```
$order = Order::groupBy('id')
    ->where('id', $purchase->id)
    ->selectRaw('id, customer_id, customer_name,
status, created_at')
    ->get();

return Inertia::render('Purchases/Edit', [
    'items' => $items,
    'order' => $order
]);
```

購買履歴 編集 edit (Vue側)

Pages/Purchases/Edit.vue Create.vueをコピー

```
import dayjs from 'dayjs'
```

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

```
onMounted(() => {  
  props.items.forEach( item => {  
    itemList.value.push({  
      略 quantity: item.quantity }) }) })
```

```
const form = reactive({  
  date: dayjs(props.order[0].created_at).format("YYYY-MM-DD"),  
  customer_id: props.order[0].customer_id,  
  status: props.order[0].status,  
  items: []  
})
```

```
<input type="radio" name="status" v-model="form.status" value="1">未キャンセル  
<input type="radio" name="status" v-model="form.status" value="0">キャンセル済
```


購買履歴 キャンセル済みなら

キャンセル済みなら編集できなくする対応

Pages/Purchases/Show.vue

```
<div v-if="props.order[0].status == true">  
  <Link>  
</div>
```



購買履歷

更新 update

購買履歴 update

Pages/Items/Edit.vueを参考に調整

```
const form = reactive({  
  id: props.order[0].id, // 追加
```

```
const updatePurchase = id => {
```

略

```
  Inertia.put(route('purchases.update', { purchase:  
id }), form )  
}
```

```
<form @submit.prevent="updatePurchase(form.id)">
```


購買履歴 update

Request\UpdatePurchaseRequest.php

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

中間テーブルの情報を更新するにはsync()が便利
引数に配列が必要なので事前に作成しておく

PurchaseController@update

```
$purchase->status = $request->status;
```

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

```
$items = [];
```

```
foreach($request->items as $item)
```

```
{
```

```
    $items = $items + [
```

```
        // item_id => [ 中間テーブルの列名 => 値 ]
```

```
        $item['id'] => [ 'quantity' => $item['quantity']]
```

```
    ];
```

```
}
```

```
$purchase->items()->sync($items);
```

```
return to_route('dashboard');
```