



商品関連

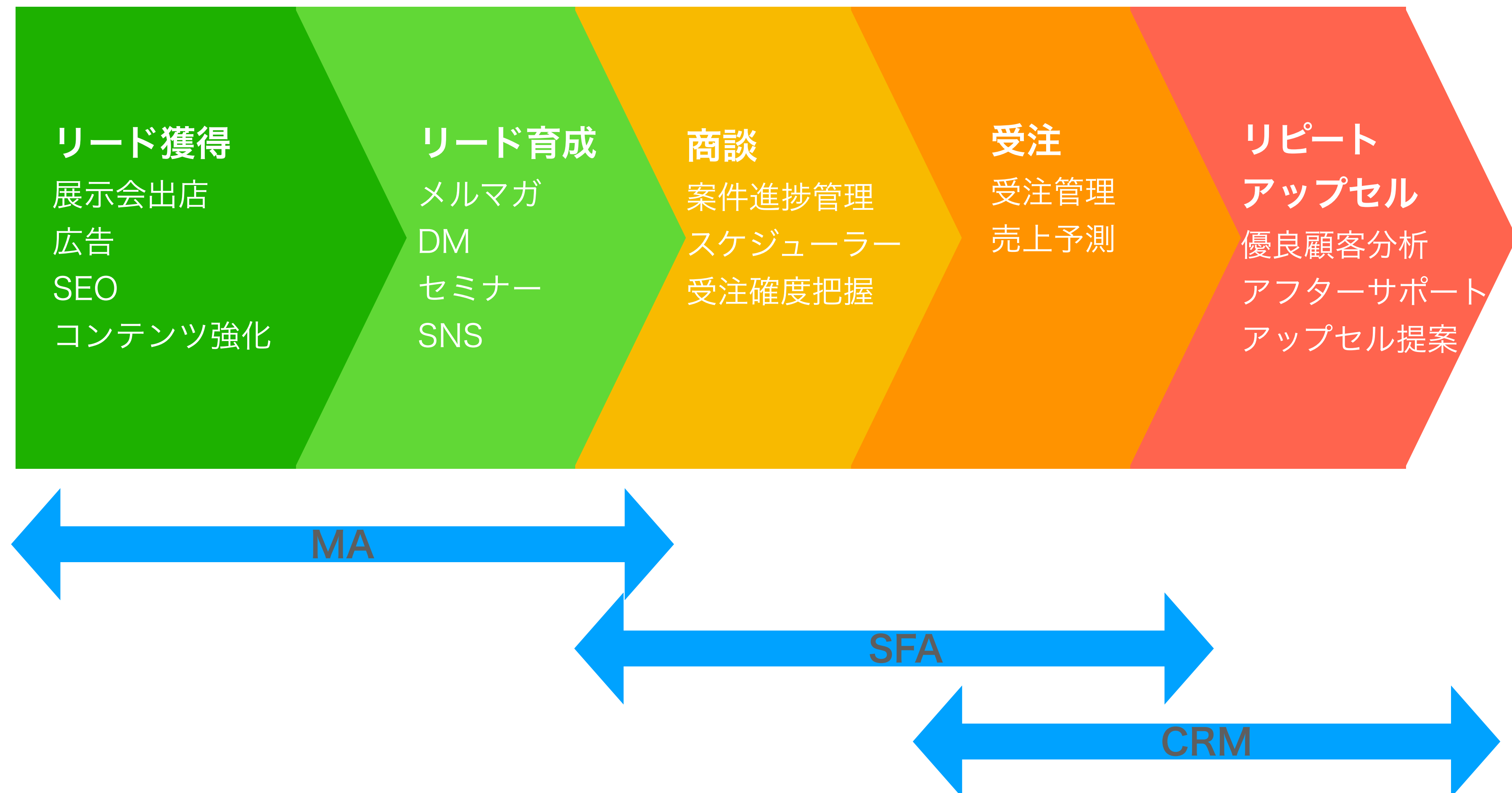





# CRMの概要

# 業務支援ツール

## 売上を上げるための施策・役割



# MA・SFA・CRMについて



## MA

Marketing Automation

新規顧客・見込み客の発掘

## SFA

Sales Force Automation

営業活動で生じる情報全般を保存、分析

## CRM

Customer Relationship Management

顧客関係管理・顧客満足度の向上



# LTVについて

LTV (LifeTimeValue 顧客生涯価値)

テクノロジーの進化により  
たくさんのサービスがつくれるようになった。

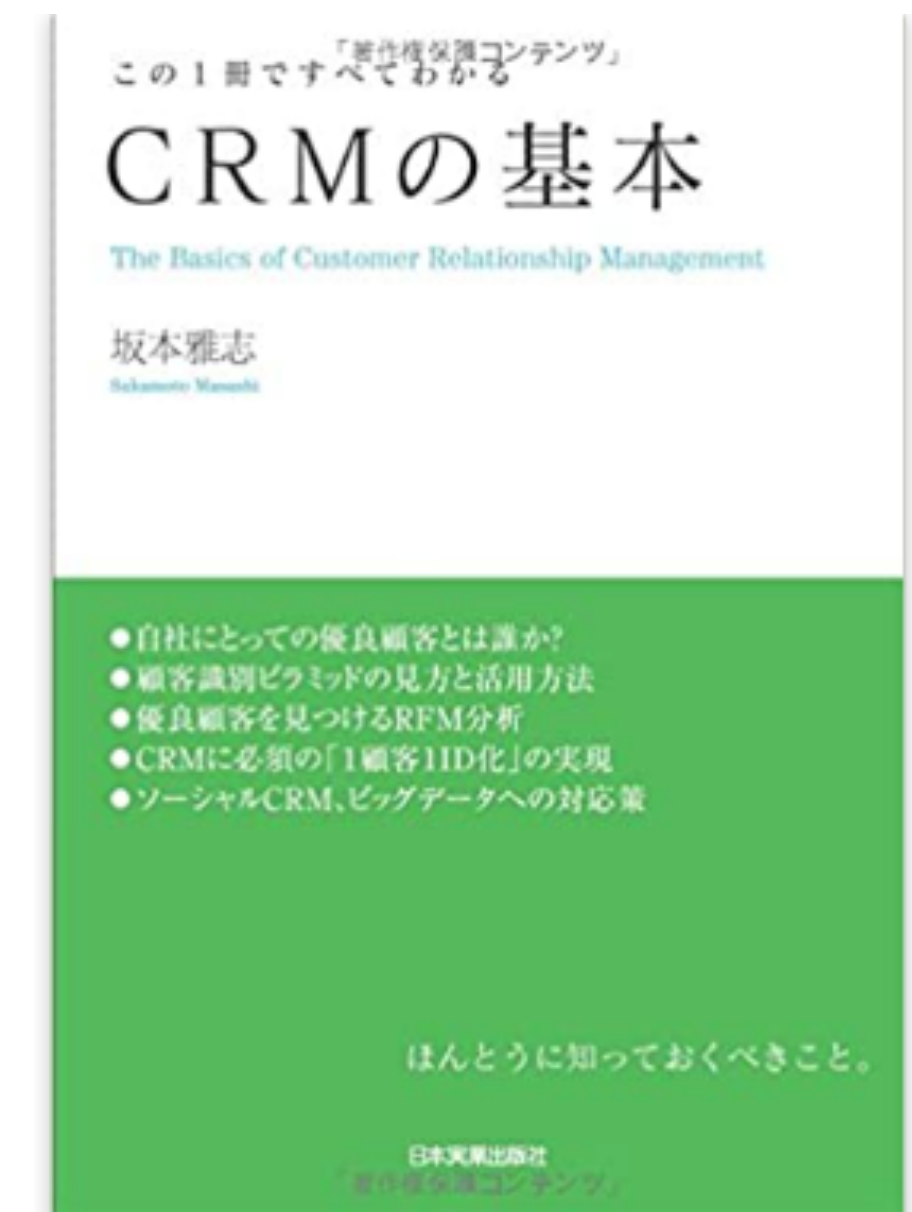
競争は激しくなり、機能のコモディティ化、  
人口減少による市場縮小などの要因により、  
新規顧客獲得の難易度は上がり、コストも上がる。

既存顧客をより重要視する時代に  
(売り切り型 -> サブスクリプション(月額制))

# CRMの目的

売上の内訳を顧客別に識別し、  
識別した優良顧客を維持・育成・獲得  
していくことにより、経費効率を上げる事

- ・顧客の識別
- ・顧客関係の構築



# CRMの例



ここ2年で20万円以上使ってくれた  
お客様限定のクーポンを発行

ここ1年で利用された方限定の特別企画案内

ex) RFM分析

(最終購入日、購入頻度、購入金額)

3つの指標で顧客グループ分け

# 今回のCRMの想定

来店してサービスを受ける  
スタジオ・サロン・ホテルなど

予約機能は別講座を参照



## 【Laravel】イベント予約システムをつくってみよう【Jetstream x Livewire】【TALLスタック】

さまざまな業界で需要のある予約管理システムの中から、日時を指定して予約できるイベント管理システムをハンズオン形式で作っていきます。Datepicker、Carbon、Livewireなどを扱い動的な予約カレンダーなどの構築方法を学べます。

世界のアオキ (Akihiro Aoki)

4.4 ★★★★★ (31)

合計12.5時間・レクチャーの数: 130・中級





# 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tailblocks

TAILBLOCKS <https://tailblocks.cc/>

機能設計 ・ ・ 機能名、処理内容

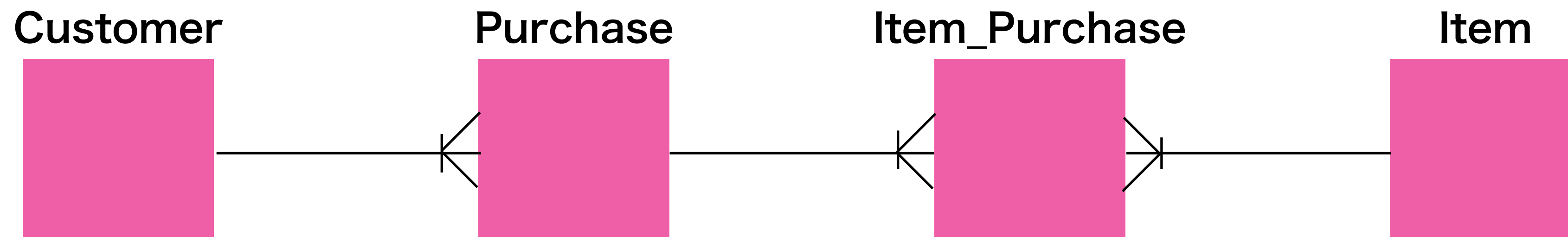
データ設計 ・ ・ テーブル設計、ER図

# 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







# アプリ名、ロゴ

# アプリ名・ロゴ



アプリ名・・・.envファイル  
APP\_NAME=uCRM

config/app.php内で設定される

ロゴ(ロゴ 作成 無料 など検索)

<https://drive.google.com/drive/u/1/folders/1QHUdgxZAitf3b0w9qCHROWZCK9EEN61R>



# ロゴ表示

publicフォルダに直接置く・・・初期ファイル  
storageフォルダ・・・フォルダ内画像はgitHubにアップしない

表側(public)から見れるようにリンク  
php artisan storage:link  
public/storage リンクが生成される

components/ApplicationLogo.vueを変更  
( / と書くと publicフォルダ配下と認識される)

```
<template>  
  <div>  
      
  </div>  
</template>
```



# Items 下準備



# Items 下準備



php artisan make:model Item -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_items_table.php
public function up()
{
    Schema::create('items', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('memo')->nullable();
        $table->integer('price');
        $table->boolean('is_selling')->default(true);
        $table->timestamps();
    });
}
```

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

# モデル app/Models/Item.php

コントローラ側 Item::create() で保存できるように  
モデル側に追記

app/Models/Item.php

略

```
protected $fillable = [  
    'name',  
    'memo',  
    'price',  
    'is_selling'  
];
```



# Items ルーティング

routes/web.php

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

略

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

php artisan route:list と入力しエラーでなければ  
OK

# UserSeeder

毎回ユーザー登録すると手間がかかるので先にダミーデータ作成しておきます。

**database/seeder/UserSeeder.php**

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

class UserSeeder extends Seeder
{
    public function run()
    {
        DB::table('users')->insert([
            'name' => 'test',
            'email' => 'test@test.com',
            'password' => Hash::make('password123')
        ]);
    }
}
```

# DatabaseSeeder

database/seeder/DatabaseSeeder.php

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

php artisan migrate:fresh --seed  
エラーでなければOK





# ItemSeeder

# ItemSeeder

毎回金額が変わるとわかりづらくなるので手動でダミーデータ作成しておきます

**database/seeder/ItemSeeder.php**

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

```
class ItemSeeder extends Seeder
```

```
{
```

```
public function run()
```

```
{
```

```
    DB::table('items')->insert([
```

```
        [
```

```
            'name' => 'カット', 'memo' => 'カットの詳細', 'price' => 6000
```

```
        ],
```

```
        [
```

```
            'name' => 'カラー',
```

```
            'memo' => 'カラーの詳細',
```

```
            'price' => 8000
```

```
        ],
```

```
        [
```

```
            'name' => 'パーマ(カット込)',
```

```
            'memo' => 'パーマ、カットの詳細',
```

```
            'price' => 13000
```

```
        ],
```

```
    ]); }
```

```
}
```

# DatabaseSeeder

database/seeder/DatabaseSeeder.php

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


php artisan migrate:fresh --seed  
エラーでなければOK





# 商品管理

# ナビメニュー



Dashboard.vueをコピーして  
Pages/Items/Index.vueを作成

Layouts/Authenticated.vueに  
商品管理のリンクを追加  
:href="route('items.index')"  
(レスポンス性もあるので2ヶ所)

# ナビメニュー

## ItemController.php

```
use Inertia\Inertia;
```

```
public function index()  
{  
    return Inertia::render('Items/Index');  
}
```

## Pages/Items/Index.vue


タイトルなどを 商品一覧 と変更





# 商品一覽 Index

# 商品管理 Index



TailBlocksのPricing 2つ目をコピーし  
Pages/Items/Index.vueにペースト

不要な箇所を削除 + CSS調整

# 商品管理 Index

## ItemController.php

```
public function index()
{
    return Inertia::render('Items/Index', [
        'items' => Item::select('id', 'name', 'price', 'is_selling')
    ]);
}
```

## Pages/Items/Index.vue

```
defineProps({ items: Array })

<tr v-for="item in items" :key="item.id">
  <td>{{ item.id }}</td>
  <td>{{ item.name }}</td>
  <td>{{ item.price }}</td>
  <td>{{ item.is_selling }}</td>
</tr>
```



# 商品管理 Index 調整

## Pages/Items/Index.vue

```
<td>
```

```
  <span v-if="item.is_selling === 1 ">販売中</span>
```

```
  <span v-if="item.is_selling === 0 ">停止中</span>
```

```
</td>
```

An abstract background on the left side of the slide, featuring bold, diagonal brushstrokes in various shades of pink, magenta, and purple. The strokes vary in intensity and direction, creating a dynamic, textured effect.

# jsconfig.json

# jsconfig.json



templateに青線がついている件

JavaScriptの設定ファイル

```
"compilerOptions": {
```

略

```
  "jsx": "preserve", // 追記
```

```
}
```





# 商品登録 Create

# 商品登録

Dashboard.vueをコピーして

Pages/Items/Create.vue を作成

タイトルなどを 商品登録に変更

Pages/Items/Index.vueに

Linkコンポーネントを追加

```
<Link as="button" :href="route('items.create')">
```

```
商品登録</Link>
```

# 商品登録

ItemController.php

```
public function create()
```

```
{
```

```
    return Inertia::render('Items/Create');
```

```
}
```



# 商品登録 Create フォーム

TailBlocksのContact 3つ目をコピーし  
Pages/Items/Create.vueにペースト

以前作成したPages/Inertia/Create.vueを  
参考にフォームを作成する

# Pages/Items/Create.vue

## Pages/Items/Create.vue

```
<script setup>
```

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

```
import { Inertia } from '@inertiajs/inertia'
```

略

```
const form = reactive({  
  name: null,  
  memo: null,  
  price: null  
})
```

```
const storeItem = () => {  
  Inertia.post('/items', form)  
}
```

```
</script>
```

# Pages/Items/Create.vue

## Pages/Items/Create.vue

```
<template>
<form @submit.prevent="storeItem">
<label for="name">
<input id="name" type="text" name="name" v-
model="form.name">
<label for="memo">
<textarea id="memo" name="memo" v-model="form.memo">
<label for="price">
<input id="price" type="number" name="price" v-
model="form.price">
</form>
</template>
```





# 商品登録 Store

# まずはそのまま保存

## Requests/StoreItemRequest.php

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

## ItemController.php

```
public function store(StoreItemRequest $request)
{
    Item::create([
        'name' => $request->name,
        'memo' => $request->memo,
        'price' => $request->price,
    ]);

    return to_route('items.index');
}
```

# バリデーション

## Requests/StoreItemRequest.php

```
public function rules()
{
    return[
        'name' => ['required', 'max:50'],
        'memo' => ['required', 'max:255'],
        'price' => ['required', 'numeric'],
    ];
}
```



# 追記: バリデーションについて

2023年2月現在

ValidationErrorsコンポーネントが  
なくなっている

対策1 1つずつ対応するパターン

対策2 ValidationErrorsを作成するパターン

# 対策1: 1つずつ対応

resources/views/Pages/Inertia/Create.blade.phpを参照

```
defineProps({  
  errors: Object // エラーがあれば受け取れる  
})
```

InputError.vueを使う場合は

:messageでpropsを渡せばOK

```
import InputError from '@/Components/InputError.vue';  
<InputError :message="errors.name" />
```

# 対策2: ValidationErrorsを作成

gitHubからコードコピー

[https://github.com/aokitashipro/laravel\\_uCRM/blob/main/resources/js/Components/ValidationErrors.vue](https://github.com/aokitashipro/laravel_uCRM/blob/main/resources/js/Components/ValidationErrors.vue)

resources/views/js/Components/ValidationErrors.vue  
を作成

usePage() で共有データ取得だが、  
props.value.errorsが含まれていないので  
definePropsで受け取るよう一部コードを変更



# 対策2: ValidationErrorsを作成

resources/views/js/Components/ValidationErrors.vue

propsで受け取れるようにする。

各値はprops.errorsをv-forで繰り返し表示

```
const props = defineProps({ errors: Object })
```

```
const hasErrors = computed(() => Object.keys(props.errors).length > 0);
```

```
<li v-for="(error, key) in props.errors" :key="key">{{ error }}</li>
```

使用するページ側

```
import BreezeValidationErrors from '@/Components/ValidationErrors.vue'
```

```
defineProps({ errors: Object })
```

```
<BreezeValidationErrors :errors="errors" />
```

# コンポーネントを使ってみる

Pages/Auth/Login.vueを参考にコンポーネントを使ってみる

Components/ValidationErrors.vue

computed() ・ ・ Vueの機能  
リアルタイムで検知、計算する

usePage() ・ ・ Inertiaの機能  
マニユアルはShared data内  
頭にuseとつくのは合成関数(関わる機能をまとめてカプセル化した関数)

# コンポーネントを使ってみる

Pages/Items/Create.vue

import BreezeValidationErrors from 略

<BreezeValidationErrors />

<section>の上

lang/ja/validation.phpのattributesに追記

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

略

```
'name' => '名前',
```

```
'memo' => 'メモ',
```

```
'price' => '価格'
```

```
],
```



# フラッシュメッセージ

app/Http/Middleware/HandleInertiaRequests.php

```
public function share(Request $request)
{
    return array_merge(parent::share($request), [
        略
        'flash' => [
            'message' => fn() => $request->session()->get('message'),
            'status' => fn() => $request->session()->get('status'), //追記
        ],
    ]);
}
```

# フラッシュメッセージ

ItemController.php@store

略

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

# コンポーネント作成

## Pages/Components/FlashMessage.vue

```
<script setup>
```

```
</script>
```

```
<template>
```

```
  <div v-if="$page.props.flash.status === 'success'" class="bg-  
blue-300 text-white p-4">
```

```
    {{ $page.props.flash.message }}
```

```
  </div>
```


```
</template>
```

## Pages.Items.Index.vue

containerクラスの下に追記

```
<FlashMessage />
```





# 商品詳細 Show

# リンク、ルート、コントローラ

## Pages/Items/Index.vue

```
<Link class="text-blue-400" :href="route('items.show', { item:  
  item.id })">  
  {{ item.id }}  
</Link>
```

## ItemController@show

```
public function show(Item $item)  
{  
  return Inertia::render('Items/Show',  
    [  
      'item' => $item  
    ]  
  );  
}
```

# コンポーネント作成

Pages/Items/Create.vue をコピーして  
Pages/Items/Show.vue作成

inputの箇所をdivタグに変更して対応

```
defineProps({  
  item: Object  
})
```

```
<label>
```

```
<div>{{ item.name }}</div>
```

```
<div v-if="item.is_selling === 1">販売中</div>
```



# textarea内は改行したい

phpにはnl2brメソッドがあるけど

JSにはないので自作必要

「nl2br php js」などで検索して  
コードをコピー

js/common.js などに作成

```
const nl2br = (str) => {  
  var res = str.replace(/\r\n/g, "<br>");  
  res = res.replace(/(\n|\r)/g, "<br>");  
  return res;  
}
```

```
export { nl2br } // 別のファイルで使えるようにexportする 56
```

# Pages/Items/Show.vue

そのままではエスケープ処理されて  
<br>と表示されるので  
v-htmlを使ってHTMLとして表示する

```
<script setup>  
import { nl2br } from '@common'  
</script>
```

```
<div v-html="nl2br(item.memo)"  
</div>
```



# 商品編集 Edit



# 商品編集 Edit

## Pages/Items/Show.vue

ボタン追加

```
<Link as="button" :href="route('items.edit', { item:  
item.id })">編集する</Link>
```

## ItemController@edit

```
public function edit(Item $item)  
{  
    return Inertia::render('Items/Edit', [  
        'item' => $item  
    ]);  
}
```

# 商品編集 Edit

Pages/Items/Create をコピーして**Pages/Items/Edit.vue** を作成

```
<script setup>
```

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

```
//コントローラから渡ってくる情報をdefinePropsで受けて、それを変数に入れておく
```

```
const props = defineProps({ item: Object })
```

```
// reactiveでリアクティブ対応にする
```

```
const form = reactive({  
  id: props.item.id,  
  name: props.item.name,  
  memo: props.item.memo,  
  price: props.item.price,  
  is_selling: props.item.is_selling  
})
```

# 商品編集 Edit

ステータスはradioボタンで設定

```
<input type="radio" name="is_selling"  
v-model="form.is_selling" value="1">
```

```
<input type="radio" name="is_selling"  
v-model="form.is_selling" value="0">
```





# 商品更新 Update

# 商品更新 Update

php artisan route:list  
をみると、updateはPUT|PUTCH  
とある

## Pages/Items/Edit.vue

```
import { Inertia } from '@inertiajs/inertia'
```

```
const updateItem = id => {  
  Inertia.put(route('items.update', { item: id }), form)  
}
```

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

# UpdateItemRequest

```
public function authorize()  
{  
    return true; // trueに換えないと保存できない  
}
```

```
public function rules()  
{  
    return [  
        'name' => ['required', 'max:255'],  
        'memo' => ['required', 'max:255'],  
        'price' => ['required', 'numeric'],  
        'is_selling' => ['required', 'boolean']  
    ];  
}
```



# ItemController@update

```
public function update(UpdateItemRequest $request, Item $item)
{
    // dd($item->name, $request->name);

    $item->name = $request->name;
    $item->memo = $request->memo;
    $item->price = $request->price;
    $item->is_selling = $request->is_selling;
    $item->save();

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





商品削除 destroy

# 商品削除 destroy

php artisan route:listでルート情報確認

## Pages/Items/Show.vue

```
import { Inertia } from '@inertiajs/inertia'
```

```
const deleteItem = id => {  
  Inertia.delete(route('items.destroy', {item: id}), {  
    onBefore: () => confirm('本当に削除しますか?')  
  })  
}
```

```
<div class="mt-20 bg-red-500">  
<button @click="deleteItem(item.id)" >削除する</button>  
</div>
```



# 商品削除 destroy

```
public function destroy(Item $item)
{
    $item->delete();

    return to_route('items.index')
->with([
        'message' => '削除しました。',
        'status' => 'danger'
    ]);
}
```

ソフトデリートに関しては別講座で扱っています。

Laravel 第2弾 マルチログインでECサイト

内容: LaravelBreeze、 BladeComponent、 Stripe(決済)、 ライフサイクル、 メール



**【Laravel】** マルチログイン機能を構築し本格的なECサイトをつくってみよう  
**【Breeze/tailwindcss】**

Laravelが搭載している認証機能を活用し、管理者、オーナー、ユーザーと3つのログイン情報を持たせ、本格的なECサイトをつくってみよう。 Bladeコンポーネント, Stripe決済, 画像アップロードなど実践形式でたっぷり解説してます。

世界のアオキ (Akihiro Aoki)

4.5 ★★★★★ (329)

合計21時間・レクチャーの数: 182・中級



# Components/FlashMessage.vue

```
<div v-if="$page.props.flash.status" ===  
'danger'" class="bg-red-300">  
  {{ $page.props.flash.message }}  
</div>
```