

AWSと【Laravel】で 書籍販売 E C サイトを作る【Breeze(Inertia)】

発行日：2025/09/26

著者：システム開発推進 G 畠山 慧

目次

- [目次](#)
- [開発の目的・背景](#)
 - [学習の目的](#)
 - [AWSを選んだ理由](#)
 - [Vue.jsを選んだ理由](#)
- [実装アプリ内容](#)
 - [実装したアプリ](#)
 - [実装画面・機能](#)
- [仕様について](#)
 - [共通ヘッダー](#)
 - [Bungo 商品一覧](#)
 - [Bungo 商品詳細](#)
 - [Bungo 商品削除](#)
 - [Bungo 商品登録](#)
 - [Bungo 商品編集](#)
 - [Bungo カート\(商品購入\)](#)
 - [Bungo カート\(商品\)](#)
 - [開発・工程スケジュール](#)
- [環境構築について](#)
 - [IPアドレス制御](#)
 - [アプリディレクトリ構成例](#)
- [工程で苦労した点](#)
- [詳細設計書の作成について](#)
 - [DB設計](#)
 - [ログイン](#)
 - [パスワードリセット](#)
 - [ユーザー作成](#)
 - [商品一覧・検索](#)
 - [商品登録](#)
 - [商品編集](#)
 - [商品削除](#)
 - [商品購入初期表示](#)
 - [カートへ追加](#)
 - [商品購入](#)
 - [購入履歴](#)
- [工程で苦労した点](#)
- [単体テスト仕様書の作成](#)
- [コーディング作業について](#)
 - [Laravel開発重要ポイント ~viエディタでもなんとかなった理由~](#)

- [開発例 画像表示](#)
- [工程で苦労した点](#)
- [テスト実施について](#)
- [リリース後の対応について](#)
- [かかった費用](#)

開発の目的・背景

学習の目的

- モダンな開発環境でも活躍できるエンジニアになるため
- 新卒の頃と比べて自分がどのくらい成長したか実感し、自信を付けたい

AWSを選んだ理由

- AWSが完全未経験なのでいい機会だと思ったから
- デスクトップPCしか持っていない事もあり、入社時も在宅時もオンラインで開発を完結させたかったから

Vue.jsを選んだ理由

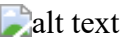
- 最近流行りのモダン言語でWebアプリ開発をしてみたかったから
- なんか名前が格好良かったから

実装アプリ内容

実装したアプリ

書籍販売ECサイト Bungo

アプリ名とアプリロゴはGeminiに考えてもらいました。



実装画面・機能

画面		主要機能	新規開発	管理者権限	Laravel URL
ログイン	ログインする		-	-	/login
パスワードリセット	パスワードリセットする		-	-	/password/reset
アカウント新規作成	ログインユーザーのアカウントを新規作成する		-	-	/register
商品一覧	WelcomおよびDashBoaad的な立ち位置で商品を表示する		○	-	/item/index
商品詳細	1商品の詳細を表示する		○	-	/item/{id}
	1商品を削除する		○	○	/item/{id}/delete
商品登録	新しい商品を登録する		○	○	/item/create
商品編集	登録されている商品情報を編集する		○	○	/item/{id}/edit

画面	主要機能	新規開発	管理者権限	Laravel URL
カート(商品購入)	カートに追加された商品情報を表示する	○	-	/purchase/index
購入履歴	購入した商品情報を表示する	○	-	/purchase/show

次ページ以降で開発した各機能の仕様について説明します

仕様について

共通ヘッダー

- ログイン ※ログイン前のみ
- アカウント新規作成 ※ログイン前のみ
- 商品一覧
- 商品登録
- カート
- 購入履歴
- ログアウト

Bungo 商品一覧

- ログイン前は詳細を見るボタンが表示されない
 - 検索ボタンを押下後にDBに登録されているデータが表示される
 - 検索条件は「商品名」「著者名」の部分一致
 - ページネーションは8件まで表示可能
 - /public/images/配下の画像を参照する
- ログイン後は詳細を見るボタンが表示される。ヘッダーも違う。
 - 管理者の場合ヘッダーが違う

Bungo 商品詳細

- 商品一覧画面で選択した書籍の詳細表示ができる
 - 「編集する」ボタン押下により商品編集画面へ遷移する
 - 「カートへ追加」ボタン押下によりカートへ商品を追加とワークテーブルへの登録をする
 - 管理者権限でログイン後のみ「削除する」ボタン押下できる
- 管理者でなければボタンは押せない

```
<div class="p-2 w-full">
  <Link v-if="$page.props.auth.user.name === 'root'" as="button" :href="route('items.edit')
</div>
<div class="mt-20 p-2 w-full">
```

```
<button v-if="$page.props.auth.user.name === 'root'" @click="deleteItem(item.id)">削除する</div>
```

アクセス制御はユーザー名でオンコーディングなので最悪です。

AWSにはアプリの特定のURLにアクセスできるユーザーを管理する方法があるはずですが、ここまで手をかけられませんでした。

Bungo 商品削除



- 「削除する」ボタン押下により商品の商品管理テーブルから削除する
- /public/images/配下の対象の画像を削除する

Bungo 商品登録



- 情報等を入力し、登録ボタンを押下後、DBにデータとして格納される
- 必須項目のバリデーションチェックを行う。
- 画像ファイルを /public/images/にYYYYmmDDhhMMss.png形式で配備
→ なぜかsvgやjpgは描画エラーで使えなかったため

Bungo 商品編集



- 商品詳細画面からの情報を編集画面に渡す
- 必須項目のバリデーションチェックを行う
- 変更がある箇所のみ書籍の更新を行う
- 画像ファイルは変更がある場合のみ更新する

Bungo カート(商品購入)



- ログインユーザのセッションIDに紐づくpurchase_statusが0の商品を表示する
- 購入するボタン押下でDB内のpurchase_statusが1に更新される
- 選択した商品の合計金額を算出表示する
- 無選択はエラーとする

Bungo カート(商品)



- DB内のpurchase_statusが1の、セッションIDに紐づく購入商品を表示する

開発・工程スケジュール

環境構築について

予定期間：2025年9月1日～9月5日 予定

実施期間：2025年9月1日～9月10日 完了

EC2上にLAMP環境を構築し、
フロントがVue.jsのMVCモデルアプリを開発しました。



項目	内容	項目	内容
開発環境	AWS	開発言語	Vue.js、php、bash
OS	Amazon Linux2	フレームワーク	Laravel
DNSサーバ	Route53	ルーティング	Inertia
SSL/TLS証明書発行	ACM	Webサーバ	Apache
https化	ALB	コーディング	9割vi、1割VScode
データベース	Amazon Aurora DB MySQL	ソース管理	Github
外部ストレージ	S3 ※今回は使用していない	ドメイン取得	お名前ドットコム

IPアドレス制御

図では表現できませんでしたが、セキュリティグループを使用しています。

インバウンドルールで (ホワイトリスト的に) アクセスするIPアドレスの制御を可能にしています。



アプリディレクトリ構成例

```
.
├── .env
├── app
│   └── Controller、Modelとか
├── artisan
├── config
├── database
│   └── テーブル定義
├── package-lock.json
├── package.json
├── public
│   ├── images
│   └── 外部公開用ページ群
├── resources
│   └── Vueファイル
├── routes
│   └── web.php
└── storage
```

工程で苦勞した点

- AWSの画面がネットに載っている情報と違う
→なんやかんやQiitaが一番参考になった
- SSL/TLS証明書の登録がなんかうまくいかない
→フレームワークでHttpsをリダイレクトする設定が必要だった
- DBのクエリエディタが使えない
→設定漏れ
- 貸し出しPCではGitHubのプルリクエストへアクセスできない
- VSCode拡張機能のSSH接続でEC2上のソースはいじれるが、接続断が多発
→すべて諦めてviエディタで開発
- npm run buildがEC2上で動かない
→突然npmが使えなくなったりするバグ、英語勉強しといてよかった...

GitHub Issue #279

Issue: "laravel-vite-plugin" resolved to an ESM file #279 [▶ GitHubで詳細を見る](#)

詳細設計書の作成について

予定期間：2025年9月5日～9月10日 予定

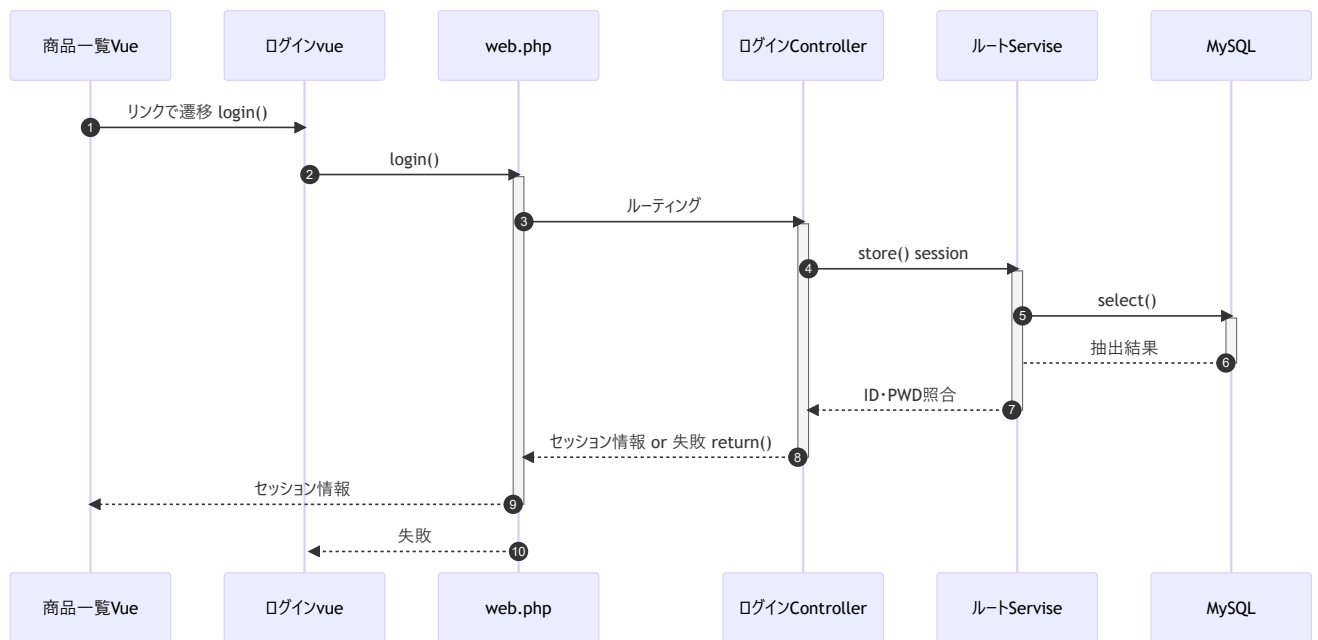
実施期間：2025年9月10日～9月11日 完了

DB設計

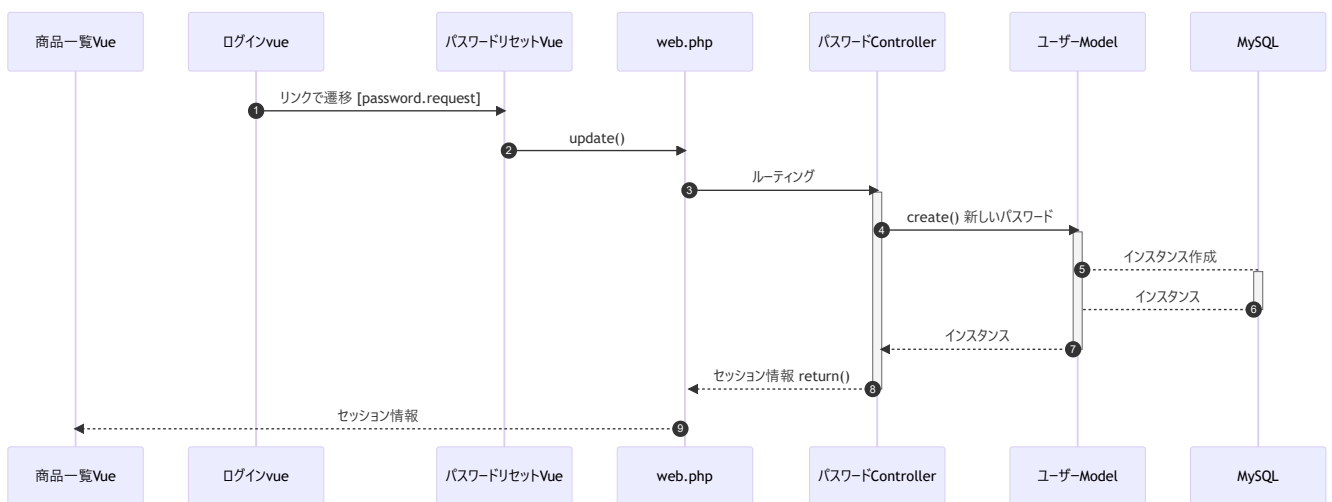
- ログインするユーザー
 - 商品
 - 購入状態
- を管理するテーブルを作成しました。



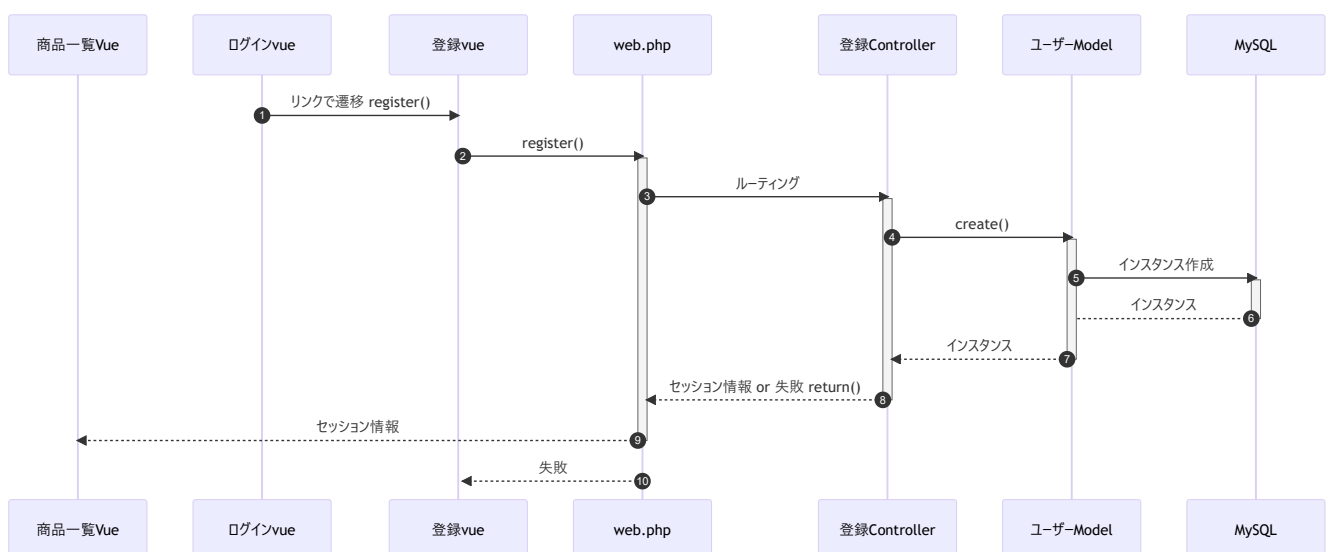
ログイン



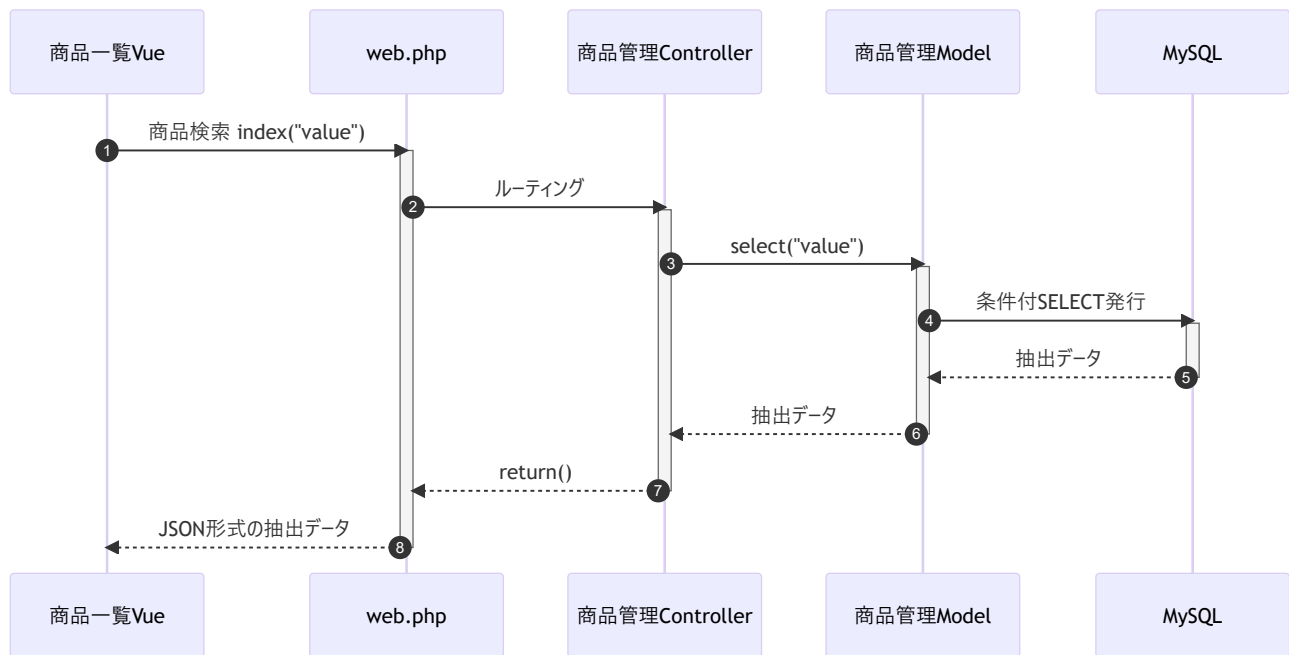
パスワードリセット



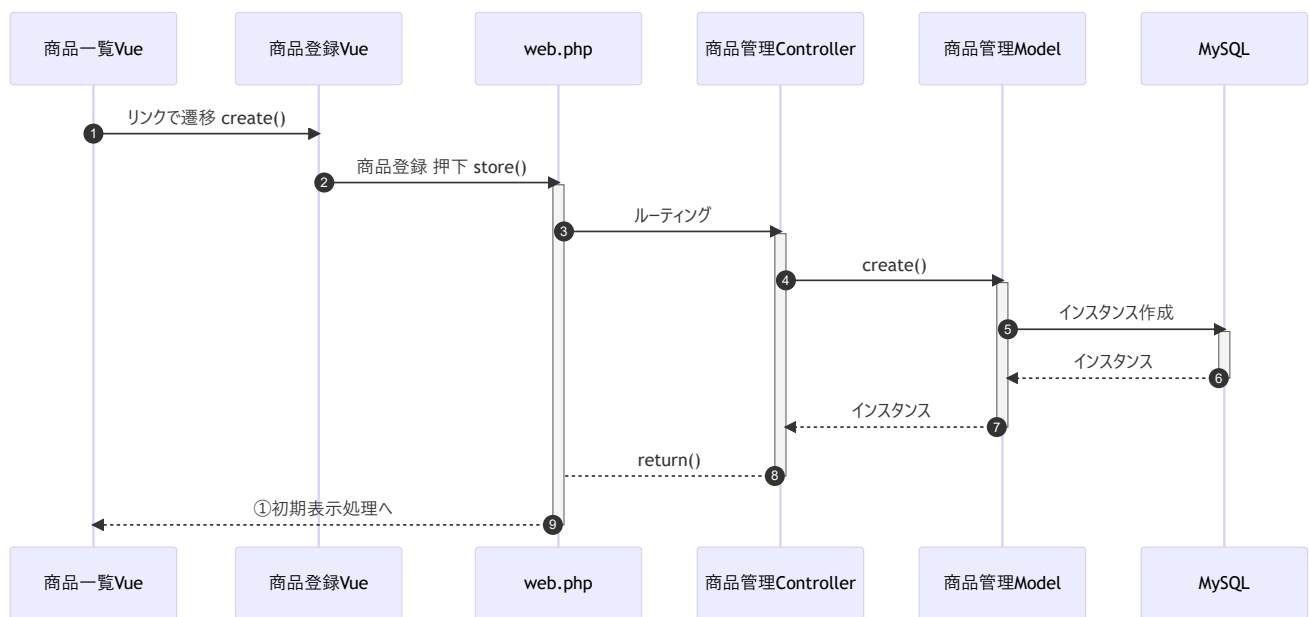
ユーザー作成



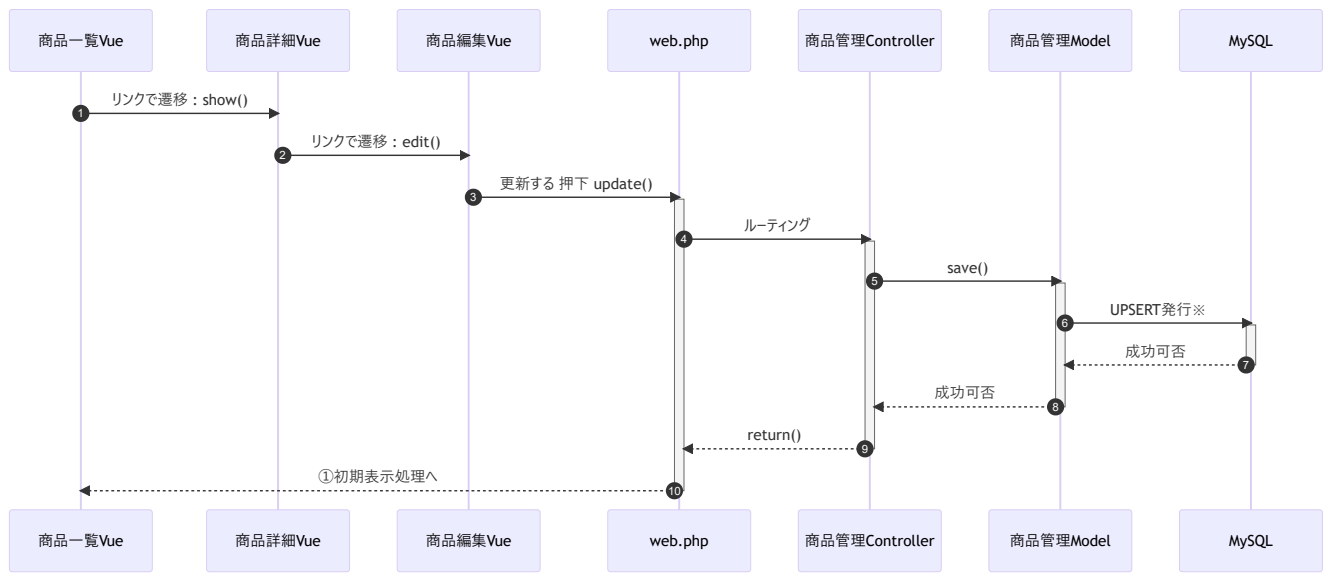
商品一覧・検索



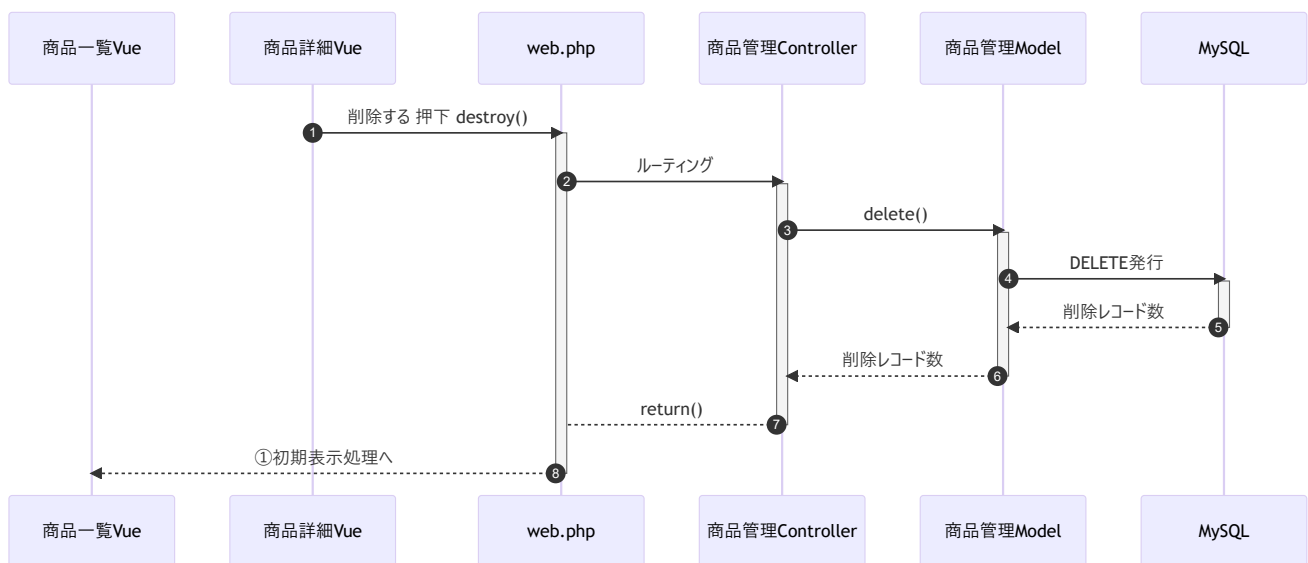
商品登録



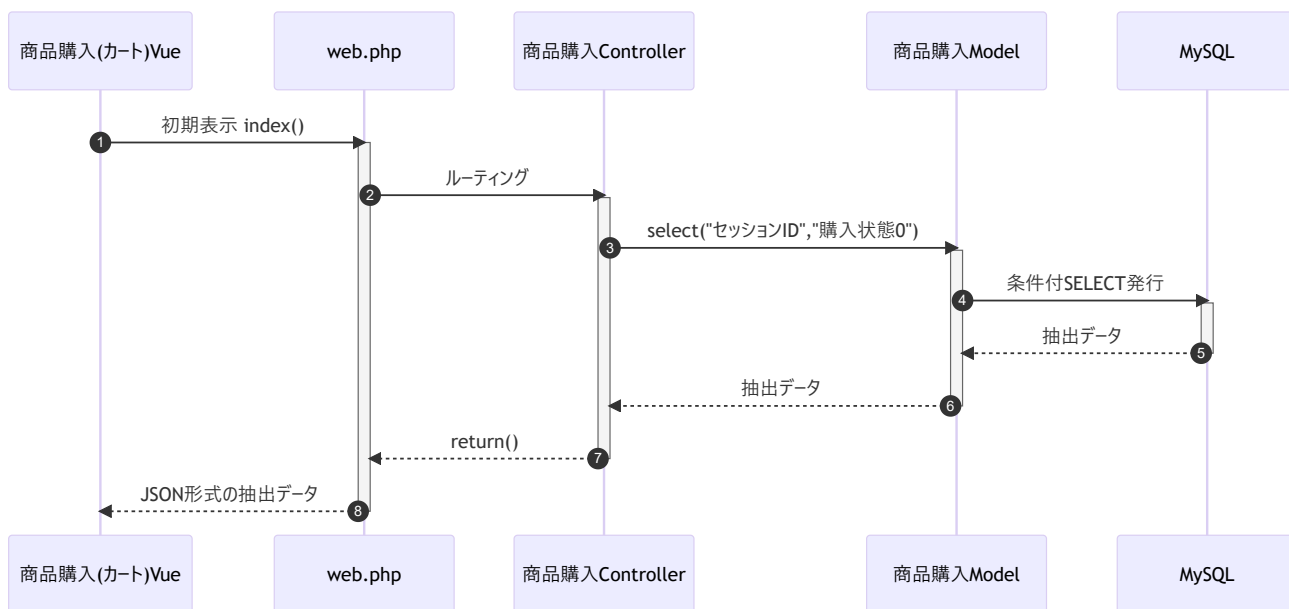
商品編集



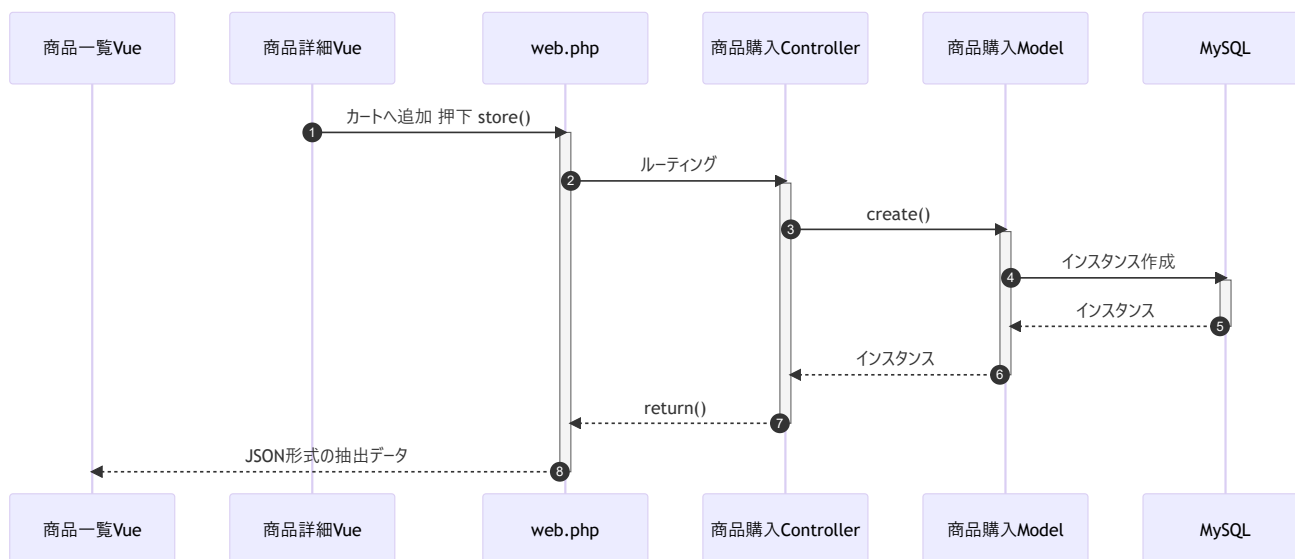
商品削除



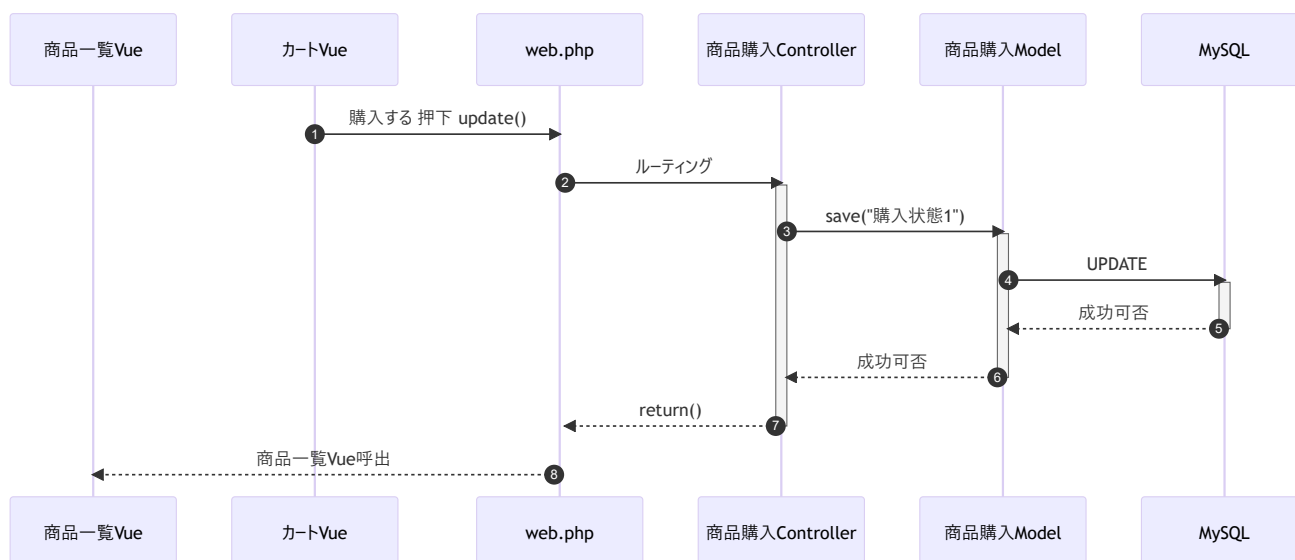
商品購入初期表示



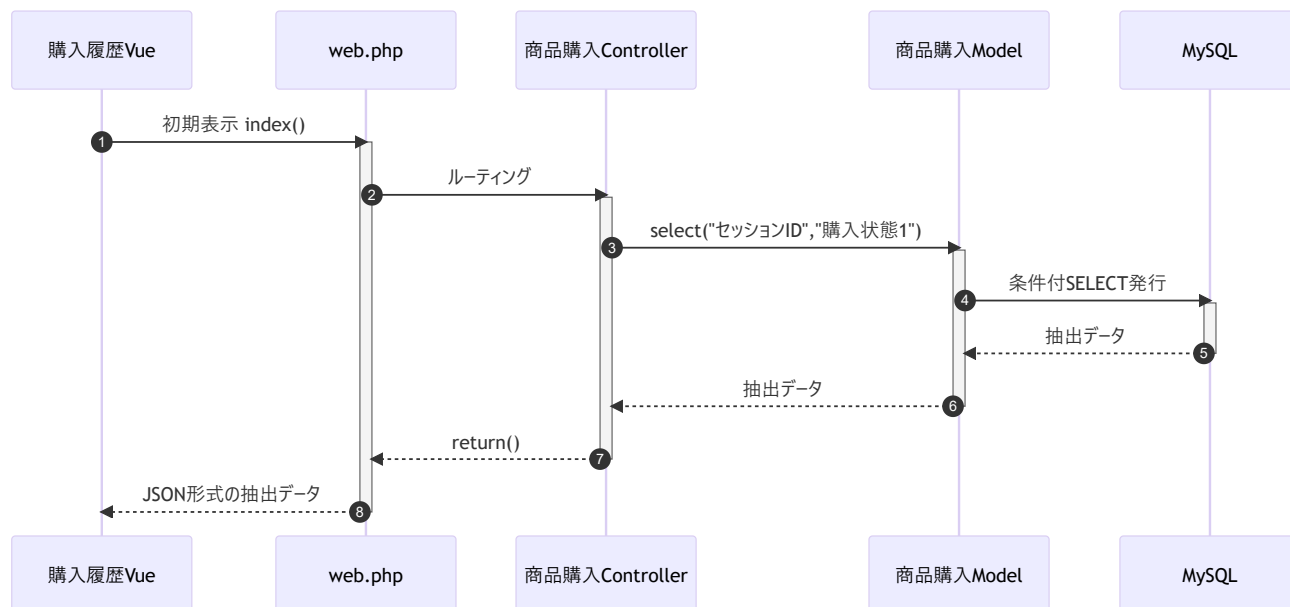
カートへ追加



商品購入



購入履歴



工程で苦労した点

- ログイン-ログアウトはLaravelフレームワークのテンプレート機能で実装済みだったが、中身の解析に時間がかかった。
図に落とし込んだが、合っているかは不安。
独自の認証機能を使うなら外すのに時間かかりそう。
- Laravelのローカルクエリスコープを利用しているのでSQLは直接書かない設計にできた
→中身で何やっているか理解しないといけなかった。
- saveメソッドがUPSERTっぽくふるまっているが厳密には違うらしい。

単体テスト仕様書の作成

正直、あまり難しいことをしてないので省略します。

バリデーションチェック、ログイン前後のメニュー確認程度です。

コーディング作業について

Laravel開発重要ポイント ~viエディタでもなんとかなった理由~

1. Artisanコマンドを使い倒す

コーディングを最速で行うための必須コマンドで、かなり強力です。

例えば、在庫管理の機能を追加するとします。

```
php artisan make:model Zaiko -a
```

このコマンドで以下のファイルが一括生成されます。

- app/Models/Post.php (モデル SQL)
- database/factories/PostFactory.php (ファクトリ)
- database/migrations/xxx_create_zaike_table.php (マイグレーション DBテーブル)
- app/Http/Controllers/PostController.php (コントローラ)
- app/Policies/PostPolicy.php (ポリシー アクセス制御とか)
- database/seeder/zaike_seeder.php (シーダー DBデータ準備)
- app/Http/Requests/StoreZaikoRequest.php (バリデーションルール)
- app/Http/Requests/UpdateZaikoRequest.php (バリデーションルール)

手作業でファイルを生成するよりも安全で、高速に新規開発が出来ます。

さらに、Controllerも見てみると標準で以下のソースが記載されています。

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreZaikoRequest;
use App\Http\Requests\UpdateZaikoRequest;
use App\Models\Zaiko;
use Illuminate\Http\Request;
use Inertia\Inertia;
use Illuminate\Support\Facades\Auth;

class ZaikoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \App\Http\Requests\StoreZaikoRequest  $request
     * @return \Illuminate\Http\Response
     */
    public function store(StoreZaikoRequest $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  \App\Models\Zaiko  $zaiko
     * @return \Illuminate\Http\Response
     */
}
```

```

    */
    public function show(Request $request)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param \App\Models\Zaiko $zaiko
     * @return \Illuminate\Http\Response
     */
    public function edit(Zaiko $zaiko)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param \App\Http\Requests\UpdateZaikoRequest $request
     * @param \App\Models\Zaiko $zaiko
     * @return \Illuminate\Http\Response
     */
    public function update(UpdateZaikoRequest $request)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param \App\Models\Zaiko $zaiko
     * @return \Illuminate\Http\Response
     */
    public function destroy(Zaiko $zaiko)
    {
        //
    }
}

```

勿論、Controllerだけ作ること可能です。

Artisanコマンドにより開発時間の大幅な短縮、コード品質の向上、チーム開発の効率化が担保されます。

一番使ったのは下記コマンドで、ルーティングの設定を見ることが出来ます。

```

php artisan route:list

GET|HEAD      / ..... We
GET|HEAD      _debugbar/assets/javascript ..... debugbar.assets.js > Barryvdh\Debugba
GET|HEAD      _debugbar/assets/stylesheets ... debugbar.assets.css > Barryvdh\Debugba
DELETE        _debugbar/cache/{key}/{tags?} debugbar.cache.delete > Barryvdh\Debugbar
GET|HEAD      _debugbar/clockwork/{id} debugbar.clockwork > Barryvdh\Debugbar > OpenH
GET|HEAD      _debugbar/open ..... debugbar.openhandler > Barryvdh\Debugbar > OpenH
POST          _debugbar/queries/explain debugbar.queries.explain > Barryvdh\Debugbar
POST          _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgni
GET|HEAD      _ignition/health-check . ignition.healthCheck > Spatie\LaravelIgnition
POST          _ignition/update-config ignition.updateConfig > Spatie\LaravelIgnition
GET|HEAD      api/user .....
GET|HEAD      confirm-password ..... password.confirm > Auth\ConfirmableI
POST          confirm-password ..... Auth\ConfirmablePa
GET|HEAD      dashboard .....
POST          email/verification-notification verification.send > Auth\EmailVerificati
GET|HEAD      forgot-password ..... password.request > Auth\PasswordRese
POST          forgot-password ..... password.email > Auth\PasswordRes

```

GET HEAD	items	items.index
POST	items	items.store
GET HEAD	items/create	items.create
GET HEAD	items/{item}	items.show
PUT PATCH	items/{item}	items.update
DELETE	items/{item}	items.destroy
GET HEAD	items/{item}/edit	items.edit
GET HEAD	login	login > Auth\AuthenticatedSe
POST	login	Auth\AuthenticatedSe
POST	logout	logout > Auth\AuthenticatedSe
PUT	password	password.update > Auth\Pas
GET HEAD	profile	profile.edit >
PATCH	profile	profile.update > Pr
DELETE	profile	profile.destroy > Pro

DBテーブル作るなら

```
php artisan migrate
```

DBデータ入れるなら

```
php artisan migrate:fresh --seed
```

ローカルでデバッグするなら

```
php artisan serve
```

2. npmコマンドはエラーチェックに有効

このコマンドはローカル上でデバッグ確認するときによく使います。

```
npm run dev
```

Vue.jsを変更したらビルドすることで/publicフォルダにjsファイルが生成され、本番リリースが出来ます。

```
npm run build
```

これらのコマンドは、Vue.jsで「{」とか、「;」とか抜けていないか構文チェックをしてくれるのでそこそこ助かりました。

その他エラーはブラウザでF12[開発者モード]で確認し、ぐりぐり進めていきました。

開発例 画像表示

以下の例では、どうやって商品一覧画面を表示しているのかについて紹介します。

routes/web.php

```
<?php

use App\Http\Controllers\ProfileController;
use Illuminate\Foundation\Application;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;
use App\Http\Controllers\ItemController;
use App\Http\Controllers\PurchaseController;

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

```

<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import { Head, Link } from '@inertiajs/vue3';
import Pagination from '@Components/Pagination.vue';
import FlashMessage from '@Components/FlashMessage.vue';
import { Inertia } from '@inertiajs/inertia'
import { ref } from 'vue'

defineProps({
  items: Object
})

const search = ref('')
const searchItems = () => {
  Inertia.get(route('items.index', { search: search.value }))
}
</script>

<template>

  <Head title="商品一覧" />

  <AuthenticatedLayout>
    <template #header>
      <h2 class="font-semibold text-xl text-gray-800 leading-tight">商品一覧</h2>
    </template>
    <FlashMessage />

    <div class="py-12">
      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
        <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
          <div class="flex w-full justify-center items-end mt-6">
            <div>
              <input input type="text" name="search" v-model="search">
            </div>
            <button @click="searchItems" focus:outline-none hover:bg-indigo-600 rounded
          </div>

          <div class="p-6 text-gray-900">
            <div class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 gap-6 p-4">
              <div v-for="item in items.data" :key="item.id"
                class="bg-white shadow rounded-lg p-4 flex flex-col items-center">
                
                <h3 class="text-gray-500 text-xs tracking-widest title-font mb-1">{{ item.
                <h2 class="text-gray-900 title-font text-lg font-medium">{{ item.name }}<
                <p class="mt-1">&yen{{ item.price.toLocaleString() }}</p>
                <Link class="text-blue-400" :href="route('items.show', { item: item.id })"
              </div>
            </div>
          </div>
          <div class="flex justify-center mb-4">
            <Pagination class="mt-6" :links="items.links"></Pagination>
          </div>
        </div>
      </div>
    </AuthenticatedLayout>
  </template>

```

app/Http/Controllers/ItemController.php

```
<?php
```

```
namespace App\Http\Controllers;
```

```

use App\Http\Requests\StoreItemRequest;
use App\Http\Requests\UpdateItemRequest;
use App\Models\Item;
use Inertia\Inertia;
use Illuminate\Support\Facades\Storage;
use Illuminate\Http\Request;

class ItemController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        $items = Item::searchItems($request->search)
            ->select('id', 'name', 'author', 'price', 'is_selling', 'image_id')->paginate(10);

        return [
            'id' => $item->id,
            'name' => $item->name,
            'author' => $item->author,
            'price' => $item->price,
            'is_selling' => $item->is_selling,
            'image_id' => $item->image_id, // 元のimage_idも残しておく
            'image_url' => asset('images/' . $item->image_id), // asset()を使ってURLを生成
        ];

        return Inertia::render('Items/Index', [
            'items' => $items
        ]);
    }
}

```

/app/Models/Item.php

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\Item;

class Item extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'author',
        'memo',
        'price',
        'is_selling',
        'image_id',
    ];

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

    public function scopeSearchItems($query, $input = null)
    {
        if(!empty($input)){
            if(Item::where('name', 'like', '%' . $input . '%')

```



```

->orWhere('author', 'like', $input . '%')->exists())
{
    return $query->where('name', 'like', '%' . $input . '%' )
->orWhere('author', 'like', $input . '%');
}
    }
}
}

```

よく見たら著者名が前方一致になってました。

工程で苦労した点

- テーブル定義はmigrationファイルで定義するのではなく、クエリエディタを使ってSQL実行したかった。
- Welcomeページをいじろうとしたら、全然うまくいかなかった。

→Laravelではよくあることらしい。フルパス表記で解決

```
<?php
```

```

use App\Http\Controllers\ProfileController;
use Illuminate\Foundation\Application;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;
use App\Http\Controllers\ItemController;
use App\Http\Controllers\PurchaseController;

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

Route::middleware(['auth', 'verified'])->group(function () {

    Route::controller(PurchaseController::class)->group(function () {
        Route::get('/purchases', 'index')->name('purchases.index');
        Route::post('/purchases', 'store')->name('purchases.store');
        Route::get('/purchases/{id}', 'show')->name('purchases.show');
        Route::put('/purchases/{id}', 'update')->name('purchases.update');
    });

});

Route::get('/', [App\Http\Controllers\WelcomeController::class, 'index']);
/*
Route::get('/', function () {
    return Inertia::render('Welcome', [
        'canLogin' => Route::has('login'),
        'canRegister' => Route::has('register'),
        'laravelVersion' => Application::VERSION,
        'phpVersion' => PHP_VERSION,
    ]);
});
*/
Route::get('/dashboard', function () {
    return Inertia::render('Dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

```

- 貸出PCをアカデミックルームで使っていると一部機能にアクセスできなかった。

テスト実施について

正直、あまり難しいことをしてないので省略します。

バリデーションチェック、ログイン前後のメニュー確認程度です。


リリース後の対応について


- 在庫管理の実施 後続で自作APIとか使ったら面白そう
- セッションID、cookieの設定


かかった費用

約5,000円かかってます。思ったよりかなり高い！が感想です。

ロードバランサの設定とかリリース直前にしたら安く抑えられたのかなと思います。

 alt text

 alt text

 alt text

ドメイン取得は無料でしたが、ネームサーバー登録が高かったです。.workとか中古ドメインにすれば良かった。

ちなみにインスタンスタイプはt3.micro(2Gib)、EBSは50Gibを使用しています。

デフォルトで進めるとapacheのインストールでフリーズするのでちょっとだけ拡張してます。