**M.Sc. in IT – Field of Specialization Enterprise Application Development**

**2020 – Semester 02**



**1st Year 2nd Semester**

**SE5070 – Enterprise Software Analysis Design Group Assignment**

**Group-ID : ESAD-01**

**Group-Members**:

* Thirunayan Dinesh Jeeva (MS20908188)
* H. A. M. Prasadinie (MS20907884)
* H. A. N. Ishara (MS20911980)
* R.M Chameera Laknath (MS20910990)

**Table of Contents**

[**1.** **Introduction** 2](#_Toc54473907)

[**1.1.** **Technologies and Frameworks** 3](#_Toc54473908)

[**1.2.** **Development** 4](#_Toc54473909)

[**2.** **Screenshots of working outputs** 87](#_Toc54473910)

[**3.** **REST UI outputs** 95](#_Toc54473911)

[**1.3.** **Auth Service** 95](#_Toc54473912)

[**1.4.** **Inventory Service** 102](#_Toc54473913)

[**1.5.** **Order Service** 107](#_Toc54473914)

# **Introduction**

E-Shopper is an e-commerce website. Its core focus is to provide the customers with the facility of online shopping.

**Main Features**

* User friendly
* Ability to sell products
* Ability to buy products
* Both electronic and cash payment methods
* E-delivery services

**Benefits for customers**

* Products have classified according to many classification methods. (ex:- Brands, Purpose, Price)
* Trusted sellers
* E-payment methods
* Ability to pay in cash at the doorstep after having the products to your hand
* Active customer care services (ex:- Returns and refunds, Shipping and Delivery)
* Exclusive offers (ex:- Discounts, Free shipping services)
* Bulk purchasing

**Benefits for sellers**

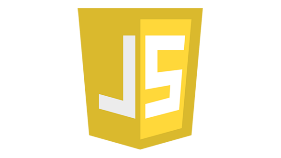
* No registration fee
* Convenient shipping methods
* Reduce the cost of doing business

**Service for the industry**

Due to the user-friendliness and attractiveness encourage customers for more online shopping.

## **Technologies and Frameworks**

Here we use below technologies and frameworks to implement this website.

Front-end development

* Framework - Angular js
* Technology – JavaScript

Back-end development

* Framework – Laravel
* Technology - PHP

Database – MySQL

## **Development**

Back-end development Flow,

API Controller Service Model Database

* + 1. **APIs**

Login

Route::post('/login/token', [App\Http\Controllers\Auth\AuthController::class, 'login']);

Route::post('/register/seller', [App\Http\Controllers\Auth\AuthController::class, 'registerSeller']);

Route::post('/register/buyer', [App\Http\Controllers\Auth\AuthController::class, 'registerBuyer']);

Route::group(['middleware' => 'auth:sanctum'], function () {

    //regular

    Route::get('/user', [App\Http\Controllers\Auth\AuthController::class, 'getUserData']);

    Route::get('/roles', [App\Http\Controllers\Auth\RolesController::class, 'getRoles']);

    Route::post('/logout', [App\Http\Controllers\Auth\AuthController::class, 'logout']);

    Route::post('/register/verify', [App\Http\Controllers\Auth\AuthController::class, 'verifySeller']);

    Route::get('/register/verify', [App\Http\Controllers\Auth\AuthController::class, 'getVerifySellerStatus']);

    //admin user management

    Route::post('/usermgt/create', [App\Http\Controllers\Admin\AdminController::class, 'adminAssistCreate']);

    Route::get('/usermgt/users', [App\Http\Controllers\Admin\AdminController::class, 'getUsers']);

    Route::get('/usermgt/users/byid/{id}', [App\Http\Controllers\Admin\AdminController::class, 'getUserById']);

    Route::put('/usermgt/update/{id}', [App\Http\Controllers\Admin\AdminController::class, 'adminAssistUpdate']);

    Route::delete('/usermgt/delete/{id}', [App\Http\Controllers\Admin\AdminController::class, 'delete']);

    //registration step 2 check

    Route::get('/usermgt/verify', [App\Http\Controllers\Admin\AdminController::class, 'pendingVerifications']);

    Route::put('/usermgt/verify/{id}', [App\Http\Controllers\Admin\AdminController::class, 'verifySellerByAdmin']);

    //role related

    Route::get('/roles', [App\Http\Controllers\Auth\RolesController::class, 'getRoles']);

    Route::get('/roles/byid/{id}', [App\Http\Controllers\Auth\RolesController::class, 'getRoleById']);

    Route::get('/roles/byname/{name}', [App\Http\Controllers\Auth\RolesController::class, 'getByName']);

    Route::post('/roles', [App\Http\Controllers\Auth\RolesController::class, 'create']);

    Route::put('/roles/{id}', [App\Http\Controllers\Auth\RolesController::class, 'update']);

    Route::delete('/roles/{id}', [App\Http\Controllers\Auth\RolesController::class, 'delete']);

});

Inventory

//category

Route::get('/inventory/category', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'getAllCategories']);

Route::get('/inventory/category/nested/{id}', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'getAllNestedCategories']);

Route::get('/inventory/category/withproducts/{id}', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'getAllWithProducts']);

Route::post('/inventory/category', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'create']);

Route::put('/inventory/category/{id}', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'update']);

Route::delete('/inventory/category/{id}', [App\Http\Controllers\Inventory\ProductsCategoryController::class, 'delete']);

//product

Route::get('/inventory/products', [App\Http\Controllers\Inventory\ProductsController::class, 'getAllProducts']);

Route::get('/inventory/products/byid/{id}', [App\Http\Controllers\Inventory\ProductsController::class, 'getById']);

Route::get('/inventory/products/byname/{name}', [App\Http\Controllers\Inventory\ProductsController::class, 'getByName']);

Route::post('/inventory/products', [App\Http\Controllers\Inventory\ProductsController::class, 'create']);

Route::put('/inventory/products/{id}', [App\Http\Controllers\Inventory\ProductsController::class, 'update']);

Route::delete('/inventory/products/{id}', [App\Http\Controllers\Inventory\ProductsController::class, 'delete']);

//productinfo

Route::post('/inventory/productinfo', [App\Http\Controllers\Inventory\ProductInfoController::class, 'create']);

Route::get('/inventory/productinfo/byid/{id}', [App\Http\Controllers\Inventory\ProductInfoController::class, 'getById']);

Route::put('/inventory/productinfo/{id}', [App\Http\Controllers\Inventory\ProductInfoController::class, 'update']);

Route::delete('/inventory/productinfo/{id}', [App\Http\Controllers\Inventory\ProductInfoController::class, 'delete']);

//cart related

Route::put('/inventory/productinfo/cart/{id}', [App\Http\Controllers\Inventory\ProductInfoController::class, 'quantityUpdate']);

Order

//invoice

Route::post('/invoice', [App\Http\Controllers\Invoice\InvoiceController::class, 'create']);

Route::get('/invoice', [App\Http\Controllers\Invoice\InvoiceController::class, 'getAllInvoices']);

Route::get('/invoice/byid/{id}', [App\Http\Controllers\Invoice\InvoiceController::class, 'getById']);

Route::get('/invoice/delivery/{id}', [App\Http\Controllers\Invoice\InvoiceController::class, 'delivered']);

Route::put('/invoice/changestatus/{id}', [App\Http\Controllers\Invoice\InvoiceController::class, 'update']);

Route::put('/invoice/{id}', [App\Http\Controllers\Invoice\InvoiceController::class, 'delete']);

//productinfo

Route::post('/invoice/invoiceitems', [App\Http\Controllers\Invoice\InvoiceItemsController::class, 'create']);

Route::get('/invoice/invoiceitems/byid/{id}', [App\Http\Controllers\Invoice\InvoiceItemsController::class, 'getById']);

Route::put('/invoice/invoiceitems/{id}', [App\Http\Controllers\Invoice\InvoiceItemsController::class, 'update']);

Route::delete('/invoice/invoiceitems/{id}', [App\Http\Controllers\Invoice\InvoiceItemsController::class, 'delete']);

//cart related

Route::put('/invoice/productinfo/cart/{id}', [App\Http\Controllers\Invoice\InvoiceItemsController::class, 'quantityUpdate']);

* + 1. **Controllers**

Login

* AuthController

<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\Services\Auth\AuthService;

class AuthController extends Controller

{

    private $user;

    private $authService;

    public function \_\_construct(Request $request)

    {

        // $this->user = $request->user();

        $this->authService = new AuthService();

    }

    public function login(Request $request)

    {

        return $this->authService->login($request);

    }

    public function getUserData(Request $request)

    {

        return $this->authService->getUserDetails($request);

    }

    public function logout(Request $request)

    {

        return $this->authService->logout($request);

    }

    public function registerSeller(Request $request)

    {

        return $this->authService->registerSeller($request);

    }

    public function registerBuyer(Request $request)

    {

        return $this->authService->registerBuyer($request);

    }

    public function verifySeller(Request $request)

    {

        return $this->authService->verifySeller($request);

    }

    public function getVerifySellerStatus(Request $request)

    {

        return $this->authService->getVerifySellerStatus($request);

    }

}

Inventory

* ProductInfoController

<?php

namespace App\Http\Controllers\Inventory;

use App\Http\Controllers\Controller;

use App\Services\Inventory\ProductInfoService;

use Illuminate\Http\Request;

class ProductInfoController extends Controller

{

    private $productInfoService;

    public function \_\_construct()

    {

        $this->productInfoService = new ProductInfoService();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create(Request $request)

    {

        return $this->productInfoService->create($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id)

    {

        return $this->productInfoService->getById($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update(Request $request, $id)

    {

        return $this->productInfoService->update($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function quantityUpdate(Request $request, $id)

    {

        return $this->productInfoService->quantityUpdate($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete(Request $request, $id)

    {

        return $this->productInfoService->delete($request, $id);

    }

}

* ProductsCategoryController

<?php

namespace App\Http\Controllers\Inventory;

use App\Http\Controllers\Controller;

use App\Services\Inventory\ProductCategoryService;

use Illuminate\Http\Request;

class ProductsCategoryController extends Controller

{

    private $productCategoryService;

    public function \_\_construct()

    {

        $this->productCategoryService = new ProductCategoryService();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create(Request $request)

    {

        return $this->productCategoryService->create($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllCategories()

    {

        return $this->productCategoryService->getCategoriesWithProducts();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllNestedCategories($id)

    {

        return $this->productCategoryService->getAllNestedCategories($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllWithProducts($id)

    {

        return $this->productCategoryService->getAllWithProducts($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update(Request $request, $id)

    {

        return $this->productCategoryService->update($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete(Request $request, $id)

    {

        return $this->productCategoryService->delete($request, $id);

    }

}

* ProductsController

<?php

namespace App\Http\Controllers\Inventory;

use App\Http\Controllers\Controller;

use App\Services\Inventory\ProductsService;

use Illuminate\Http\Request;

class ProductsController extends Controller

{

    private $productService;

    public function \_\_construct()

    {

        $this->productService = new ProductsService();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create(Request $request)

    {

        return $this->productService->create($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllProducts()

    {

        return $this->productService->getAllProducts();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id)

    {

        return $this->productService->getById($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getByName($name)

    {

        return $this->productService->getByName($name);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update(Request $request, $id)

    {

        return $this->productService->update($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete(Request $request, $id)

    {

        return $this->productService->delete($request, $id);

    }

}

Order

* InvoiceController

<?php

namespace App\Http\Controllers\Invoice;

use App\Http\Controllers\Controller;

use App\Services\Invoice\InvoiceService;

use Illuminate\Http\Request;

class InvoiceController extends Controller

{

    private $invoiceService;

    public function \_\_construct()

    {

        $this->invoiceService = new InvoiceService();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create(Request $request)

    {

        return $this->invoiceService->create($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllInvoices(Request $request)

    {

        return $this->invoiceService->getAllInvoices($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id)

    {

        return $this->invoiceService->getById($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delivered(Request $request, $id)

    {

        return $this->invoiceService->delivered($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update(Request $request, $id)

    {

        return $this->invoiceService->update($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete(Request $request, $id)

    {

        return $this->invoiceService->delete($request, $id);

    }

}

* InvoiceItemsController

<?php

namespace App\Http\Controllers\Invoice;

use App\Http\Controllers\Controller;

use App\Services\Invoice\InvoiceItemsService;

use Illuminate\Http\Request;

class InvoiceItemsController extends Controller

{

    private $invoiceItemsService;

    public function \_\_construct()

    {

        $this->invoiceItemsService = new InvoiceItemsService();

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create(Request $request)

    {

        return $this->invoiceItemsService->create($request);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id)

    {

        return $this->invoiceItemsService->getById($id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update(Request $request, $id)

    {

        return $this->invoiceItemsService->update($request, $id);

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete(Request $request, $id)

    {

        return $this->invoiceItemsService->delete($request, $id);

    }

}

* + 1. **Service**

Login

* AuthService

<?php

namespace App\Services\Auth;

use Illuminate\Http\Request;

use Illuminate\Validation\ValidationException;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Models\User;

use App\Models\SellerDetail;

class AuthService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    public function registerSeller(Request $request)

    {

        DB::beginTransaction();

        try {

            $request->validate([

                'name' => 'required|string|max:255',

                'email' => 'required|email|max:255',

                'password' => 'required|min:8',

                'device\_name' => 'required',

            ]);

            $userCheck = User::where('email', $request->email)->first();

            if ($userCheck) {

                throw ValidationException::withMessages([

                    'email' => ['You can not use this email, use a different one.'],

                ]);

            }

            $userInst = new User();

            $userInst->name = $request->name;

            $userInst->email = $request->email;

            $userInst->role\_id = roleNames('SELLER')->id;

            $userInst->password = Hash::make($request->password);

            $userInst->created\_at = now();

            $userInst->save();

            DB::commit();

            $token = $userInst->createToken($request->device\_name)->plainTextToken;

            $response = [

                'user' => $userInst,

                'token' => $token

            ];

            return response()->json([

                        'data' => $response,

                        'message' => 'REGISTERED\_AS\_SELLER\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    public function registerBuyer(Request $request)

    {

        DB::beginTransaction();

        try {

            $request->validate([

                'name' => 'required|string|max:255',

                'email' => 'required|email|max:255',

                'password' => 'required|min:8',

                'device\_name' => 'required',

            ]);

            $userCheck = User::where('email', $request->email)->first();

            if ($userCheck) {

                throw ValidationException::withMessages([

                    'email' => ['You can not use this email, use a different one.'],

                ]);

            }

            $userInst = new User();

            $userInst->name = $request->name;

            $userInst->email = $request->email;

            $userInst->role\_id = roleNames('BUYER')->id;

            $userInst->password = Hash::make($request->password);

            $userInst->created\_at = now();

            $userInst->save();

            DB::commit();

            $token = $userInst->createToken($request->device\_name)->plainTextToken;

            $response = [

                'user' => $userInst,

                'token' => $token

            ];

            return response()->json([

                        'data' => $response,

                        'message' => 'REGISTERED\_AS\_BUYER\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    public function login(Request $request)

    {

        $request->validate([

            'email' => 'required|email',

            'password' => 'required',

            'device\_name' => 'required',

        ]);

        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password, $user->password)) {

            throw ValidationException::withMessages([

                'email' => ['The provided credentials are incorrect.'],

            ]);

        }

        $token = $user->createToken($request->device\_name)->plainTextToken;

        $response = [

            'user' => $user,

            'token' => $token

        ];

        return response($response, 200);

    }

    public function getUserDetails(Request $request)

    {

        return $request->user();

    }

    public function logout(Request $request)

    {

        $request->user()->tokens()->delete();

        $response = [

            'message' => 'logout success',

        ];

        return response($response, 200);

    }

    public function verifySeller(Request $request)

    {

        DB::beginTransaction();

        try {

            $user = $request->user();

            if (!$user) {

                throw new Exception("USER\_NOT\_EXISTS", getStatusCodes('EXCEPTION'));

            }

            if (!permissionLevelCheck('SELLER\_ONLY', $request->user()->role\_id)) {

                throw new Exception("ACCESS\_DENIED", getStatusCodes('UNAUTHORIZED'));

            }

            $docExist = SellerDetail::where('document', '=', $request->br\_doc)

                    ->where('verify\_status', '=', 1)

                    ->first();

            if ($docExist) {

                throw new Exception("ALREADY\_VERIFIED", getStatusCodes('EXCEPTION'));

            }

            $detailInst = New SellerDetail();

            $detailInst->user\_id = $user->id;

            $detailInst->document = $request->br\_doc;

            $detailInst->created\_at = now();

            $detailInst->save();

            DB::commit();

            addToLog('seller verification step 2 completed: ' . $user->id, $this->enumSuccess);

            return response()->json([

                        'data' => $detailInst,

                        'message' => 'DOCUMENT\_SUBMIT\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    public function getVerifySellerStatus(Request $request)

    {

        DB::beginTransaction();

        try {

            $user = $request->user();

            if (!$user) {

                throw new Exception("USER\_NOT\_EXISTS", getStatusCodes('EXCEPTION'));

            }

            if (!permissionLevelCheck('SELLER\_ONLY', $request->user()->role\_id)) {

                throw new Exception("ACCESS\_DENIED", getStatusCodes('UNAUTHORIZED'));

            }

            $docExist = SellerDetail::where('user\_id', '=', $user->id)

                    ->first();

            if (!$docExist) {

                throw new Exception("CANNOT\_FIND\_VERIFICATION\_DETAILS", getStatusCodes('EXCEPTION'));

            }

            addToLog('seller verification step 2 checked: ' . $user->id, $this->enumSuccess);

            return response()->json([

                        'data' => $docExist,

                        'message' => 'GET\_VERIFICATION\_DETAILS\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

Inventory

* ProductInfoService

<?php

/\*\*

 \* {@inheritdoc}

 \*/

namespace App\Services\Inventory;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Services\Auth\ConnectAuthService;

use App\Models\PrProduct;

use App\Models\PrProductInfo;

class ProductInfoService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create($request)

    {

        DB::beginTransaction();

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $request->validate([

                'product\_id' => 'required|exists:pr\_product,id',

                'price' => 'required|numeric|min:0',

                'quatity' => 'required|numeric|min:0',

                'default\_image' => 'required|string|min:10|max:255',

                'product\_images' => 'required|string|min:10|max:255'

            ]);

            $userId = $userData['id'];

            $productInst = PrProduct::where('id', '=', $request->product\_id)

                    ->where('owner\_id', '=', $userId)

                    ->with('pr\_product\_infos')

                    ->get();

            if (!$productInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $totalNumberOfChild = 0;

            foreach ($productInst as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["pr\_product\_infos"]);

            }

            if ($totalNumberOfChild > 0) {

                throw new Exception("PRODUCT\_INFO\_ALREADY\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productInfoInst = New PrProductInfo();

            $productInfoInst->product\_id = $request->product\_id;

            $productInfoInst->price = $request->price;

            $productInfoInst->quatity = $request->quatity;

            $productInfoInst->default\_image = $request->default\_image;

            $productInfoInst->product\_images = $request->product\_images;

            $productInfoInst->created\_at = now();

            $productInfoInst->save();

            DB::commit();

            addToLog('product created product info: ' . $request->product\_id, $this->enumSuccess);

            return response()->json([

                        'data' => $productInfoInst,

                        'message' => 'PRODUCT\_INFO\_CREATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id, $returnType = false)

    {

        try {

            if (!isset($id)) {

                throw new Exception("PRODUCT\_INFO\_ID\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $productInfo = PrProductInfo::orderBy('created\_at', 'desc')

                    ->select(['id', 'product\_id', 'price', 'price', 'quatity', 'default\_image', 'product\_images'])

                    ->where('id', '=', $id)

                    ->first();

            if (!$productInfo) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE\_BY\_ID", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given product ' . $id, $this->enumSuccess);

            if ($returnType) {

                return $productInfo;

            }

            return response()->json([

                        'message' => 'GET\_PRODUCT\_INFO\_BY\_ID\_OK',

                        'data' => $productInfo

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $productInfoInst = PrProductInfo::where('id', '=', $id)->first();

            if (!$productInfoInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productInst = PrProduct::where('owner\_id', '=', $userId)

                    ->where('id', '=', $productInfoInst->product\_id)

                    ->first();

            if (!$productInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $request->validate([

                'product\_id' => 'required|exists:pr\_product,id',

                'price' => 'required|numeric|min:0',

                'quatity' => 'required|numeric|min:0',

                'default\_image' => 'required|string|min:10|max:255',

                'product\_images' => 'required|string|min:10|max:255'

            ]);

            $productInfoInst->product\_id = $request->product\_id;

            $productInfoInst->price = $request->price;

            $productInfoInst->quatity = $request->quatity;

            $productInfoInst->default\_image = $request->default\_image;

            $productInfoInst->product\_images = $request->product\_images;

            $productInfoInst->save();

            DB::commit();

            addToLog('PRODUCT info update product\_id: ' . $request->product\_id, $this->enumSuccess);

            return response()->json([

                        'data' => $productInfoInst,

                        'message' => 'PRODUCT\_INFO\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function quantityUpdate($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $productInfoInst = PrProductInfo::where('id', '=', $id)->first();

            if (!$productInfoInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $request->validate([

                'quatity' => 'required|numeric|min:0',

            ]);

            if ($productInfoInst->quatity < $request->quatity) {

                throw new Exception("PRODUCT\_QUANTITY\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productInfoInst->quatity = $productInfoInst->quatity - $request->quatity;

            $productInfoInst->save();

            DB::commit();

            addToLog('PRODUCT quantity update product\_id: ' . $id, $this->enumSuccess);

            return response()->json([

                        'data' => $productInfoInst,

                        'message' => 'PRODUCT\_QUANTITY\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $productInfoInst = PrProductInfo::where('id', '=', $id)->first();

            if (!$productInfoInst) {

                throw new Exception("PRODUCT\_INFO\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productInst = PrProduct::where('owner\_id', '=', $userId)

                    ->where('id', '=', $productInfoInst->product\_id)

                    ->first();

            if (!$productInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            //check with invice services

            $children = PrProduct::where('id', '=', $id)

                    ->with('pr\_product\_infos')

                    ->get();

            $totalNumberOfChild = 0;

            foreach ($children as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["pr\_product\_infos"]);

            }

            if ($totalNumberOfChild > 0) {

                throw new Exception("PRODUCT\_CAN\_NOT\_DELETE\_RELATION\_DATA\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

//            $roleExist->delete();

//            DB::commit();

            addToLog('product delete success id - ' . $id, $this->enumSuccess);

            return response()->json([

                        'message' => 'PRODUCT\_DELETE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

* ProductCategoryService

<?php

/\*\*

 \* {@inheritdoc}

 \*/

namespace App\Services\Inventory;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Services\Auth\ConnectAuthService;

use App\Models\PrCategory;

use App\Services\Inventory\ProductsService;

class ProductCategoryService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create($request)

    {

        DB::beginTransaction();

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('ADMINS\_ONLY', $userData['role\_id'])) {

                throw new Exception("ADMINS\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

//            $userId = $userData['id'];

            $request->validate([

                'super\_cat\_id' => 'nullable|exists:pr\_category,id',

                'category\_name' => 'required|string|min:6|max:255'

            ]);

            $categoryExist = PrCategory::where('category\_name', '=', $request->category\_name)

                    ->first();

            if ($categoryExist) {

                throw new Exception("PRODUCT\_CATEGORY\_EXIST", getStatusCodes('EXCEPTION'));

            }

            $categoryInst = New PrCategory();

            $categoryInst->super\_cat\_id = $request->super\_cat\_id;

            $categoryInst->category\_name = $request->category\_name;

            $categoryInst->created\_at = now();

            $categoryInst->save();

            DB::commit();

            addToLog('created category\_name: ' . $request->category\_name, $this->enumSuccess);

            return response()->json([

                        'data' => $categoryInst,

                        'message' => 'PRODUCT\_CATEGORY\_CREATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getCategoriesWithProducts()

    {

        try {

            $categories = PrCategory::select(['id', 'category\_name', 'super\_cat\_id'])

                    ->with('pr\_products:id,category\_id')

                    ->get();

            if (sizeof($categories) == 0) {

                // if no categories send empty array

                return response()->json([

                            'message' => 'GET\_CATEGORIES\_OK',

                            'data' => []

                ]);

                // throw new Exception("CATEGORIES\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productsInst = new ProductsService();

//            $fullResponse = [];

            foreach ($categories as $category) {

                $productData = [];

                foreach ($category->pr\_products as $prProduct) {

                    $productData[] = $productsInst->getById($prProduct->id, true);

                }

//                $fullResponse[] = array\_merge($category->toArray(), ['products' => $productData]);

//                $fullResponse[] =

                $category->products = $productData;

            }

            return response()->json([

                        'message' => 'GET\_CATEGORIES\_WITH\_PRODUCTS\_OK',

                        'data' => $categories

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    public function getAllNestedCategories($id)

    {

        try {

            if (!isset($id)) {

                throw new Exception("CATEGORY\_ID\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $category = PrCategory::where('id', '=', $id)

                    ->select('id', 'super\_cat\_id')//'category\_name', 'active\_status',

                    ->first();

            if (!$category) {

                throw new Exception("CATEGORY\_NOT\_AVAILABLE\_BY\_ID", getStatusCodes('EXCEPTION'));

            }

            $categories = PrCategory::select('id', 'super\_cat\_id')

                    ->get();

            $tree2 = $this->formatTree2($categories, $id);

            $result = $this->arrayFlatten($tree2);

            $category->pr\_categories = $result;

            addToLog('view all data racks', $this->enumSuccess);

            return response()->json([

                        'message' => 'GET\_NESTED\_CATEGORIES\_OK',

                        'data' => $category

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            dd($exception);

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    public function formatTree2($tree, $parent)

    {

        $tree2 = [];

        foreach ($tree as $item) {

            if ($item['super\_cat\_id'] == $parent) {

                $item->pr\_categories = $this->formatTree2($tree, $item['id']);

                $child = $this->formatTree2($tree, $item['id']);

                $tree2[] = $child;

                $tree2[] = $item['id'];

            }

        }

        return $tree2;

    }

    public function arrayFlatten($array = null)

    {

        $result = array();

        if (!is\_array($array)) {

            $array = func\_get\_args();

        }

        foreach ($array as $key => $value) {

            if (is\_array($value)) {

                $result = array\_merge($result, array\_flatten($value));

            } else {

                $result = array\_merge($result, array($key => $value));

            }

        }

        return $result;

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('ADMINS\_ONLY', $userData['role\_id'])) {

                throw new Exception("ADMINS\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

//            $userId = $userData['id'];

            $categoryInst = PrCategory::where('id', '=', $id)

                    ->first();

            if (!$categoryInst) {

                throw new Exception("PRODUCT\_CATEGORY\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $request->validate([

                'super\_cat\_id' => 'nullable|exists:pr\_category,id',

                'category\_name' => 'required|string|min:6|max:255'

            ]);

            $categoryExist = PrCategory::where('id', '!=', $id)

                    ->where('category\_name', '=', $request->category\_name)

                    ->first();

            if ($categoryExist) {

                throw new Exception("PRODUCT\_CATEGORY\_EXIST", getStatusCodes('EXCEPTION'));

            }

            $categoryInst->super\_cat\_id = $request->super\_cat\_id;

            $categoryInst->category\_name = $request->category\_name;

            $categoryInst->save();

            DB::commit();

            addToLog('update category\_name: ' . $request->category\_name, $this->enumSuccess);

            return response()->json([

                        'data' => $categoryInst,

                        'message' => 'PRODUCT\_CATEGORY\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('ADMINS\_ONLY', $userData['role\_id'])) {

                throw new Exception("ADMINS\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

//            $userId = $userData['id'];

            $categoryInst = PrCategory::where('id', '=', $id)->first();

            if (!$categoryInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $children = PrCategory::where('id', '=', $id)

                    ->with('pr\_categories')

                    ->with('pr\_products')

                    ->get();

            $totalNumberOfChild = 0;

            foreach ($children as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["pr\_categories"]);

                $totalNumberOfChild += sizeof($child->getRelations()["pr\_products"]);

            }

            if ($totalNumberOfChild > 0) {

                throw new Exception("PRODUCT\_CATEGORY\_CAN\_NOT\_DELETE\_RELATION\_DATA\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

//            $roleExist->delete();

//            DB::commit();

            addToLog('product category delete success id - ' . $id, $this->enumSuccess);

            return response()->json([

                        'message' => 'PRODUCT\_CATEGORY\_DELETE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

* ProductService

<?php

/\*\*

 \* {@inheritdoc}

 \*/

namespace App\Services\Inventory;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Services\Auth\ConnectAuthService;

use App\Models\PrProduct;

class ProductsService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create($request)

    {

        DB::beginTransaction();

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $request->validate([

                'category\_id' => 'required|exists:pr\_category,id',

                'product\_name' => 'required|string|min:6|max:255',

                'short\_desc' => 'required|string|min:10|max:255',

                'long\_desc' => 'required|string|min:10|max:255'

            ]);

            $productInst = New PrProduct();

            $productInst->owner\_id = $userId;

            $productInst->category\_id = $request->category\_id;

            $productInst->product\_name = $request->product\_name;

            $productInst->short\_desc = $request->short\_desc;

            $productInst->long\_desc = $request->long\_desc;

            $productInst->created\_at = now();

            $productInst->save();

            DB::commit();

            addToLog('product created role\_name: ' . $request->product\_name, $this->enumSuccess);

            return response()->json([

                        'data' => $productInst,

                        'message' => 'PRODUCT\_CREATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

//            dd($exception);

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllProducts()

    {

        try {

            $products = PrProduct::orderBy('created\_at', 'desc')

                    ->select(['id', 'product\_name', 'category\_id'])

                    ->with('pr\_category:id,category\_name')

                    ->with('pr\_product\_infos:product\_id,price,quatity')

                    ->paginate(getGlobalSettingByName('DEFAULT\_ITEMS\_PER\_PAGE'));

            if (sizeof($products) == 0) {

                // if no roles send empty array

                return response()->json([

                            'message' => 'GET\_PRODUCTS\_OK',

                            'data' => []

                ]);

            }

//            addToLog('view all data roles', $this->enumSuccess);

            return response()->json([

                        'message' => 'GET\_PRODUCTS\_OK',

                        'data' => $products->toArray()

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id, $returnType = false)

    {

        try {

            if (!isset($id)) {

                throw new Exception("PRODUCT\_ID\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $products = PrProduct::orderBy('created\_at', 'desc')

                    ->select(['id', 'product\_name', 'category\_id'])

                    ->with('pr\_category:id,category\_name')

                    ->with('pr\_product\_infos:product\_id,price,quatity')

                    ->where('id', '=', $id)

                    ->first();

            if (!$products) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE\_BY\_ID", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given product ' . $id, $this->enumSuccess);

            if ($returnType) {

                return $products;

            }

            return response()->json([

                        'message' => 'GET\_PRODUCT\_BY\_ID\_OK',

                        'data' => $products

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getByName($name, $returnType = false)

    {

        try {

            if (!isset($name)) {

                throw new Exception("PRODUCT\_NAME\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($name, 'string');

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $productExist = PrProduct::where('product\_name', 'like', "%{$name}%")

                    ->select(['id', 'product\_name', 'category\_id'])

                    ->with('pr\_category:id,category\_name')

                    ->with('pr\_product\_infos:product\_id,price,quatity')

                    ->get();

            if (!$productExist) {

                throw new Exception("PRODUCT\_NAME\_NOT\_EXIST", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given product ' . $name, $this->enumSuccess);

            if ($returnType) {

                return $productExist;

            }

            return response()->json([

                        'message' => 'GET\_PRODUCT\_OK',

                        'data' => $productExist

            ]);

        } catch (Exception $exception) {

            dd($exception);

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $productInst = PrProduct::where('id', '=', $id)

                    ->where('owner\_id', '=', $userId)

                    ->first();

            if (!$productInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $productExist = PrProduct::where('id', '!=', $id)

                    ->where('owner\_id', '=', $userId)

                    ->where('product\_name', '=', $request->product\_name)

                    ->first();

            if ($productExist) {

                throw new Exception("PRODUCT\_EXIST", getStatusCodes('EXCEPTION'));

            }

            $request->validate([

                'category\_id' => 'required|exists:pr\_category,id',

                'product\_name' => 'required|string|min:6|max:255',

                'short\_desc' => 'required|string|min:10|max:255',

                'long\_desc' => 'required|string|min:10|max:255'

            ]);

            $productInst->owner\_id = $userId;

            $productInst->category\_id = $request->category\_id;

            $productInst->product\_name = $request->product\_name;

            $productInst->short\_desc = $request->short\_desc;

            $productInst->long\_desc = $request->long\_desc;

            $productInst->save();

            DB::commit();

            addToLog('PRODUCT update product\_name: ' . $request->product\_name . ' ,short\_desc:' . $request->short\_desc, $this->enumSuccess);

            return response()->json([

                        'data' => $productInst,

                        'message' => 'PRODUCT\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $productInst = PrProduct::where('id', '=', $id)

                    ->where('owner\_id', '=', $userId)

                    ->first();

            if (!$productInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $children = PrProduct::where('id', '=', $id)

                    ->with('pr\_product\_infos')

                    ->get();

            $totalNumberOfChild = 0;

            foreach ($children as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["pr\_product\_infos"]);

            }

            if ($totalNumberOfChild > 0) {

                throw new Exception("PRODUCT\_CAN\_NOT\_DELETE\_RELATION\_DATA\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

//            $roleExist->delete();

//            DB::commit();

            addToLog('product delete success id - ' . $id, $this->enumSuccess);

            return response()->json([

                        'message' => 'PRODUCT\_DELETE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

Order

* InvoiceService

<?php

/\*\*

 \* {@inheritdoc}

 \*/

namespace App\Services\Invoice;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Services\Auth\ConnectAuthService;

use App\Models\InvInvoice;

class InvoiceService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create($request)

    {

        DB::beginTransaction();

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $request->validate([

                'total\_amount' => 'required|numeric',

                'payment\_procedure' => 'required|numeric|digits:1|in:1,2',

                'paid\_status' => 'required|numeric|digits:1|in:1,2',

            ]);

            $invInst = New InvInvoice();

            $invInst->buyer\_id = $userId;

            $invInst->complete\_status = 2;

            $invInst->total\_amount = $request->total\_amount;

            $invInst->payment\_procedure = $request->payment\_procedure;

            $invInst->paid\_status = $request->paid\_status;

            $invInst->created\_at = now();

            $invInst->save();

            DB::commit();

            addToLog('invoice created role\_name: ' . $invInst->buyer\_id, $this->enumSuccess);

            return response()->json([

                        'data' => $invInst,

                        'message' => 'INVOICE\_CREATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

//            dd($exception);

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getAllInvoices($request)

    {

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $invoices = InvInvoice::orderBy('created\_at', 'desc')

                    ->select(['id', 'buyer\_id', 'complete\_status', 'total\_amount',

                        'payment\_procedure', 'paid\_status', 'delivery\_date'])

                    ->with('inv\_items:inv\_id,product\_id,quantity,unit\_price,discount\_amount,total\_price')

                    ->where('buyer\_id', '=', $userId)

                    ->get();

            //->paginate(getGlobalSettingByName('DEFAULT\_ITEMS\_PER\_PAGE'));

            if (sizeof($invoices) == 0) {

                // if no invoices send empty array

                return response()->json([

                            'message' => 'GET\_PRODUCTS\_OK',

                            'data' => []

                ]);

            }

            addToLog('view all data invoices', $this->enumSuccess);

            return response()->json([

                        'message' => 'GET\_INVOICES\_OK',

                        'data' => $invoices->toArray()

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id, $returnType = false)

    {

        try {

            if (!isset($id)) {

                throw new Exception("INVOICE\_ID\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $invoice = InvInvoice::orderBy('created\_at', 'desc')

                    ->select(['id', 'buyer\_id', 'complete\_status', 'total\_amount',

                        'payment\_procedure', 'paid\_status', 'delivery\_date'])

                    ->with('inv\_items:inv\_id,product\_id,quantity,unit\_price,discount\_amount,total\_price')

                    ->where('id', '=', $id)

                    ->first();

            if (!$invoice) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE\_BY\_ID", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given invoice ' . $id, $this->enumSuccess);

            if ($returnType) {

                return $invoice;

            }

            return response()->json([

                        'message' => 'GET\_INVOICE\_BY\_ID\_OK',

                        'data' => $invoice

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getByName($name, $returnType = false)

    {

        try {

            if (!isset($name)) {

                throw new Exception("PRODUCT\_NAME\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($name, 'string');

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $productExist = PrProduct::where('product\_name', 'like', "%{$name}%")

                    ->select(['id', 'product\_name', 'category\_id'])

                    ->with('pr\_category:id,category\_name')

                    ->with('pr\_product\_infos:product\_id,price,quatity')

                    ->get();

            if (!$productExist) {

                throw new Exception("PRODUCT\_NAME\_NOT\_EXIST", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given product ' . $name, $this->enumSuccess);

            if ($returnType) {

                return $productExist;

            }

            return response()->json([

                        'message' => 'GET\_PRODUCT\_OK',

                        'data' => $productExist

            ]);

        } catch (Exception $exception) {

            dd($exception);

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \* update complete status

     \*/

    public function delivered($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $invInst = InvInvoice::where('id', '=', $id)->first();

            if (!$invInst) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            if ($invInst->complete\_status != 1) {

                throw new Exception("INVOICE\_NOT\_COMPLETED", getStatusCodes('EXCEPTION'));

            }

            $children = InvInvoice::where('id', '=', $id)

                    ->with('inv\_items')

                    ->get();

            $totalNumberOfChild = 0;

            foreach ($children as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["inv\_items"]);

            }

            if ($totalNumberOfChild == 0) {

                throw new Exception("INVOICE\_CAN\_NOT\_UPDATE", getStatusCodes('EXCEPTION'));

            }

            $invInst->delivery\_date = now();

            $invInst->save();

            DB::commit();

            addToLog('invoice delivered invoice id : ' . $id, $this->enumSuccess);

            return response()->json([

                        'data' => $invInst,

                        'message' => 'INVOICE\_DELIVERY\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \* update complete status

     \*/

    public function update($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $request->validate([

                'complete\_status' => 'required|numeric|digits:1|in:1,3'

            ]);

            $invInst = InvInvoice::where('id', '=', $id)->first();

            if (!$invInst) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            if ($invInst->complete\_status != 2) {

                throw new Exception("INVOICE\_NOT\_PENDING", getStatusCodes('EXCEPTION'));

            }

            $children = InvInvoice::where('id', '=', $id)

                    ->with('inv\_items')

                    ->get();

            $totalNumberOfChild = 0;

            foreach ($children as $child) {

                $totalNumberOfChild += sizeof($child->getRelations()["inv\_items"]);

            }

            if ($totalNumberOfChild == 0) {

                throw new Exception("INVOICE\_CAN\_NOT\_UPDATE", getStatusCodes('EXCEPTION'));

            }

            $invInst->complete\_status = $request->complete\_status;

            $invInst->save();

            DB::commit();

            addToLog('invoice complete\_status changed invoice id : ' . $request->complete\_status . ' ,invoice id' . $id, $this->enumSuccess);

            return response()->json([

                        'data' => $invInst,

                        'message' => 'INVOICE\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('SELLER\_ONLY', $userData['role\_id'])) {

                throw new Exception("SELLER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $invInst = InvInvoice::where('id', '=', $id)

                    ->where('buyer\_id', '=', $userId)

                    ->first();

            if (!$invInst) {

                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $invInst->complete\_status = 4;

            $invInst->save();

            DB::commit();

            addToLog('invoice cancel success id - ' . $id, $this->enumSuccess);

            return response()->json([

                        'message' => 'INVOICE\_CANCELLED\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

* InvoiceItemService

<?php

/\*\*

 \* {@inheritdoc}

 \*/

namespace App\Services\Invoice;

use Illuminate\Support\Facades\DB;

use Exception;

use App\Services\Auth\ConnectAuthService;

use App\Models\InvInvoice;

use App\Models\InvItem;

class InvoiceItemsService

{

    private $enumSuccess = 0;

    public function \_\_construct()

    {

        $this->enumSuccess = app('config')->get("enum.common.log\_status")['SUCCESS'];

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function create($request)

    {

        DB::beginTransaction();

        try {

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $request->validate([

                'inv\_id' => 'required|exists:inv\_invoice,id',

                'product\_id' => 'required|numeric',

                'unit\_price' => 'required|numeric|min:0',

                'discount\_amount' => 'required|numeric|min:0',

                'quantity' => 'required|numeric|min:0',

                'total\_price' => 'required|numeric|min:0'

            ]);

            $userId = $userData['id'];

            $invInst = InvInvoice::where('id', '=', $request->inv\_id)

                    ->where('buyer\_id', '=', $userId)

                    ->with('inv\_items')

                    ->first();

            if (!$invInst) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            if ($invInst->complete\_status != 2) {

                throw new Exception("ONLY\_PENDING\_INVOICE\_ITEMS\_CAN\_BE\_ADDED", getStatusCodes('EXCEPTION'));

            }

            if (sizeof($invInst->inv\_items) > 0) {

                foreach ($invInst->inv\_items as $value) {

                    if ($value->product\_id == $request->product\_id) {

                        throw new Exception("INVOICE\_ITEM\_ALREADY\_AVAILABLE", getStatusCodes('EXCEPTION'));

                    }

                }

            }

            $invoiceItemsInst = New InvItem();

            $invoiceItemsInst->inv\_id = $request->inv\_id;

            $invoiceItemsInst->product\_id = $request->product\_id;

            $invoiceItemsInst->quantity = $request->quantity;

            $invoiceItemsInst->unit\_price = $request->unit\_price;

            $invoiceItemsInst->discount\_amount = $request->discount\_amount;

            $invoiceItemsInst->total\_price = $request->total\_price;

            $invoiceItemsInst->created\_at = now();

            $invoiceItemsInst->save();

            DB::commit();

            addToLog('invoice items created for : ' . $request->inv\_id, $this->enumSuccess);

            return response()->json([

                        'data' => $invoiceItemsInst,

                        'message' => 'INVOICE\_ITEMS\_CREATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function getById($id, $returnType = false)

    {

        try {

            if (!isset($id)) {

                throw new Exception("PRODUCT\_INFO\_ID\_REQUIRED", getStatusCodes('EXCEPTION'));

            }

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $invoiceItems = InvItem::orderBy('created\_at', 'desc')

                    ->select(['id', 'inv\_id', 'product\_id', 'quantity', 'unit\_price', 'discount\_amount', 'total\_price'])

                    ->where('id', '=', $id)

                    ->first();

            if (!$invoiceItems) {

                throw new Exception("INVOCE\_ITEMS\_NOT\_AVAILABLE\_BY\_ID", getStatusCodes('EXCEPTION'));

            }

            addToLog('view data of the given invoice items ' . $id, $this->enumSuccess);

            if ($returnType) {

                return $invoiceItems;

            }

            return response()->json([

                        'message' => 'GET\_INVOICE\_ITEMS\_BY\_ID\_OK',

                        'data' => $invoiceItems

            ]);

        } catch (Exception $exception) {

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function update($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $invoiceItemsInst = InvItem::where('id', '=', $id)->first();

            if (!$invoiceItemsInst) {

                throw new Exception("INVOICE\_ITEM\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

//            $productInst = PrProduct::where('owner\_id', '=', $userId)

//                    ->where('id', '=', $invoiceItemsInst->product\_id)

//                    ->first();

//

//            if (!$productInst) {

//                throw new Exception("PRODUCT\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

//            }

            $invInst = InvInvoice::where('buyer\_id', '=', $userId)

                    ->where('id', '=', $invoiceItemsInst->inv\_id)

                    ->first();

            if (!$invInst) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            if ($invInst->complete\_status != 2) {

                throw new Exception("ONLY\_PENDING\_INVOICE\_ITEMS\_CAN\_BE\_UPDATED", getStatusCodes('EXCEPTION'));

            }

            $request->validate([

                'product\_id' => 'required|numeric',

                'unit\_price' => 'required|numeric|min:0',

                'discount\_amount' => 'required|numeric|min:0',

                'quantity' => 'required|numeric|min:0',

                'total\_price' => 'required|numeric|min:0'

            ]);

            $invoiceItemsInst->product\_id = $request->product\_id;

            $invoiceItemsInst->quantity = $request->quantity;

            $invoiceItemsInst->unit\_price = $request->unit\_price;

            $invoiceItemsInst->discount\_amount = $request->discount\_amount;

            $invoiceItemsInst->total\_price = $request->total\_price;

            $invoiceItemsInst->save();

            DB::commit();

            addToLog('Invoice item update product\_id: ' . $request->product\_id, $this->enumSuccess);

            return response()->json([

                        'data' => $invoiceItemsInst,

                        'message' => 'INVOICE\_ITEM\_UPDATE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode() == 0 ? getStatusCodes('VALIDATION\_ERROR') : $exception->getCode());

        }

    }

    /\*\*

     \* {@inheritdoc}

     \*/

    public function delete($request, $id)

    {

        DB::beginTransaction();

        try {

            $idValidation = checkDataType($id);

            if ($idValidation instanceof Exception) {

                throw new Exception($idValidation->getMessage(), $idValidation->getCode());

            }

            $authApi = new ConnectAuthService();

            $userData = $authApi->getUserDetails($request);

            if (!permissionLevelCheck('BUYER\_ONLY', $userData['role\_id'])) {

                throw new Exception("BUYER\_ONLY", getStatusCodes('UNAUTHORIZED'));

            }

            $userId = $userData['id'];

            $invoiceItemsInst = InvItem::where('id', '=', $id)->first();

            if (!$invoiceItemsInst) {

                throw new Exception("INVOICE\_ITEM\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            $invInst = InvInvoice::where('buyer\_id', '=', $userId)

                    ->where('id', '=', $invoiceItemsInst->inv\_id)

                    ->first();

            if (!$invInst) {

                throw new Exception("INVOICE\_NOT\_AVAILABLE", getStatusCodes('EXCEPTION'));

            }

            if ($invInst->complete\_status != 2) {

                throw new Exception("ONLY\_PENDING\_INVOICE\_ITEMS\_CAN\_BE\_DELETED", getStatusCodes('EXCEPTION'));

            }

            $invoiceItemsInst->delete();

            DB::commit();

            addToLog('product delete success id - ' . $id, $this->enumSuccess);

            return response()->json([

                        'message' => 'INVOICE\_ITEM\_DELETE\_OK'

            ]);

        } catch (Exception $exception) {

            DB::rollBack();

            addToLog($exception->getMessage());

            return response()->json(['message' => $exception->getMessage()], $exception->getCode());

        }

    }

}

* + 1. **Model**

Login

* User

<?php

/\*\*

 \* Class User

 \*

 \* @property int $id

 \* @property int $role\_id

 \* @property string $name

 \* @property string $email

 \* @property Carbon|null $email\_verified\_at

 \* @property string $password

 \* @property string|null $remember\_token

 \* @property Carbon|null $created\_at

 \* @property Carbon|null $updated\_at

 \*

 \* @property Role $role

 \* @property Collection|SellerDetail[] $seller\_details

 \*

 \* @package App\Models

 \*/

namespace App\Models;

use Carbon\Carbon;

//use Illuminate\Database\Eloquent\Model;

//use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

    use HasApiTokens,

        HasFactory,

        Notifiable;

    protected $table = 'users';

    protected $casts = [

        'role\_id' => 'int'

    ];

    protected $dates = [

        'email\_verified\_at'

    ];

    protected $hidden = [

        'password',

        'remember\_token'

    ];

    protected $fillable = [

        'role\_id',

        'name',

        'email',

        'email\_verified\_at',

        'password',

        'remember\_token'

    ];

    public function role()

    {

        return $this->belongsTo(Role::class);

    }

    public function seller\_details()

    {

        return $this->hasMany(SellerDetail::class);

    }

}

Inventory

* PrProduct

<?php

/\*\*

 \* Created by Reliese Model.

 \*/

namespace App\Models;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Collection;

use Illuminate\Database\Eloquent\Model;

/\*\*

 \* Class PrProduct

 \*

 \* @property int $id

 \* @property int $category\_id

 \* @property int $owner\_id

 \* @property string $product\_name

 \* @property string $short\_desc

 \* @property string $long\_desc

 \* @property Carbon $created\_at

 \* @property Carbon $updated\_at

 \*

 \* @property PrCategory $pr\_category

 \* @property Collection|PrProductInfo[] $pr\_product\_infos

 \*

 \* @package App\Models

 \*/

class PrProduct extends Model

{

    protected $table = 'pr\_product';

    protected $casts = [

        'category\_id' => 'int',

        'owner\_id' => 'int'

    ];

    protected $fillable = [

        'category\_id',

        'owner\_id',

        'product\_name',

        'short\_desc',

        'long\_desc'

    ];

    public function pr\_category()

    {

        return $this->belongsTo(PrCategory::class, 'category\_id');

    }

    public function pr\_product\_infos()

    {

        return $this->hasMany(PrProductInfo::class, 'product\_id');

    }

}

* PrProductInfo

<?php

/\*\*

 \* Created by Reliese Model.

 \*/

namespace App\Models;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Model;

/\*\*

 \* Class PrProductInfo

 \*

 \* @property int $id

 \* @property int $product\_id

 \* @property float $price

 \* @property float $quatity

 \* @property string|null $default\_image

 \* @property string|null $product\_images

 \* @property Carbon $created\_at

 \* @property Carbon $updated\_at

 \*

 \* @property PrProduct $pr\_product

 \*

 \* @package App\Models

 \*/

class PrProductInfo extends Model

{

    protected $table = 'pr\_product\_info';

    protected $casts = [

        'product\_id' => 'int',

        'price' => 'float',

        'quatity' => 'float'

    ];

    protected $fillable = [

        'product\_id',

        'price',

        'quatity',

        'default\_image',

        'product\_images'

    ];

    public function pr\_product()

    {

        return $this->belongsTo(PrProduct::class, 'product\_id');

    }

}

* PrCategory

<?php

/\*\*

 \* Created by Reliese Model.

 \*/

namespace App\Models;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Collection;

use Illuminate\Database\Eloquent\Model;

/\*\*

 \* Class PrCategory

 \*

 \* @property int $id

 \* @property int|null $super\_cat\_id

 \* @property string $category\_name

 \* @property Carbon $created\_at

 \* @property Carbon $updated\_at

 \*

 \* @property PrCategory $pr\_category

 \* @property Collection|PrCategory[] $pr\_categories

 \* @property Collection|PrProduct[] $pr\_products

 \*

 \* @package App\Models

 \*/

class PrCategory extends Model

{

    protected $table = 'pr\_category';

    protected $casts = [

        'super\_cat\_id' => 'int'

    ];

    protected $fillable = [

        'super\_cat\_id',

        'category\_name'

    ];

    public function pr\_category()

    {

        return $this->belongsTo(PrCategory::class, 'super\_cat\_id');

    }

    public function pr\_categories()

    {

        return $this->hasMany(PrCategory::class, 'super\_cat\_id');

    }

    public function pr\_products()

    {

        return $this->hasMany(PrProduct::class, 'category\_id');

    }

}

Order

* InvInvoice

<?php

/\*\*

 \* Created by Reliese Model.

 \*/

namespace App\Models;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Collection;

use Illuminate\Database\Eloquent\Model;

/\*\*

 \* Class InvInvoice

 \*

 \* @property int $id

 \* @property int $buyer\_id

 \* @property int $complete\_status

 \* @property float $total\_amount

 \* @property int $payment\_procedure

 \* @property int $paid\_status

 \* @property Carbon|null $delivery\_date

 \* @property Carbon $created\_at

 \* @property Carbon $updated\_at

 \*

 \* @property Collection|InvItem[] $inv\_items

 \*

 \* @package App\Models

 \*/

class InvInvoice extends Model

{

    protected $table = 'inv\_invoice';

    protected $casts = [

        'buyer\_id' => 'int',

        'complete\_status' => 'int',

        'total\_amount' => 'float',

        'payment\_procedure' => 'int',

        'paid\_status' => 'int'

    ];

    protected $dates = [

        'delivery\_date'

    ];

    protected $fillable = [

        'buyer\_id',

        'complete\_status',

        'total\_amount',

        'payment\_procedure',

        'paid\_status',

        'delivery\_date'

    ];

    public function inv\_items()

    {

        return $this->hasMany(InvItem::class, 'inv\_id');

    }

}

* InvItem

<?php

/\*\*

 \* Created by Reliese Model.

 \*/

namespace App\Models;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Model;

/\*\*

 \* Class InvItem

 \*

 \* @property int $id

 \* @property int $inv\_id

 \* @property int $product\_id

 \* @property float $quantity

 \* @property float $unit\_price

 \* @property float|null $discount\_amount

 \* @property float $total\_price

 \* @property Carbon $created\_at

 \* @property Carbon $updated\_at

 \*

 \* @property InvInvoice $inv\_invoice

 \*

 \* @package App\Models

 \*/

class InvItem extends Model

{

    protected $table = 'inv\_items';

    protected $casts = [

        'inv\_id' => 'int',

        'product\_id' => 'int',

        'quantity' => 'float',

        'unit\_price' => 'float',

        'discount\_amount' => 'float',

        'total\_price' => 'float'

    ];

    protected $fillable = [

        'inv\_id',

        'product\_id',

        'quantity',

        'unit\_price',

        'discount\_amount',

        'total\_price'

    ];

    public function inv\_invoice()

    {

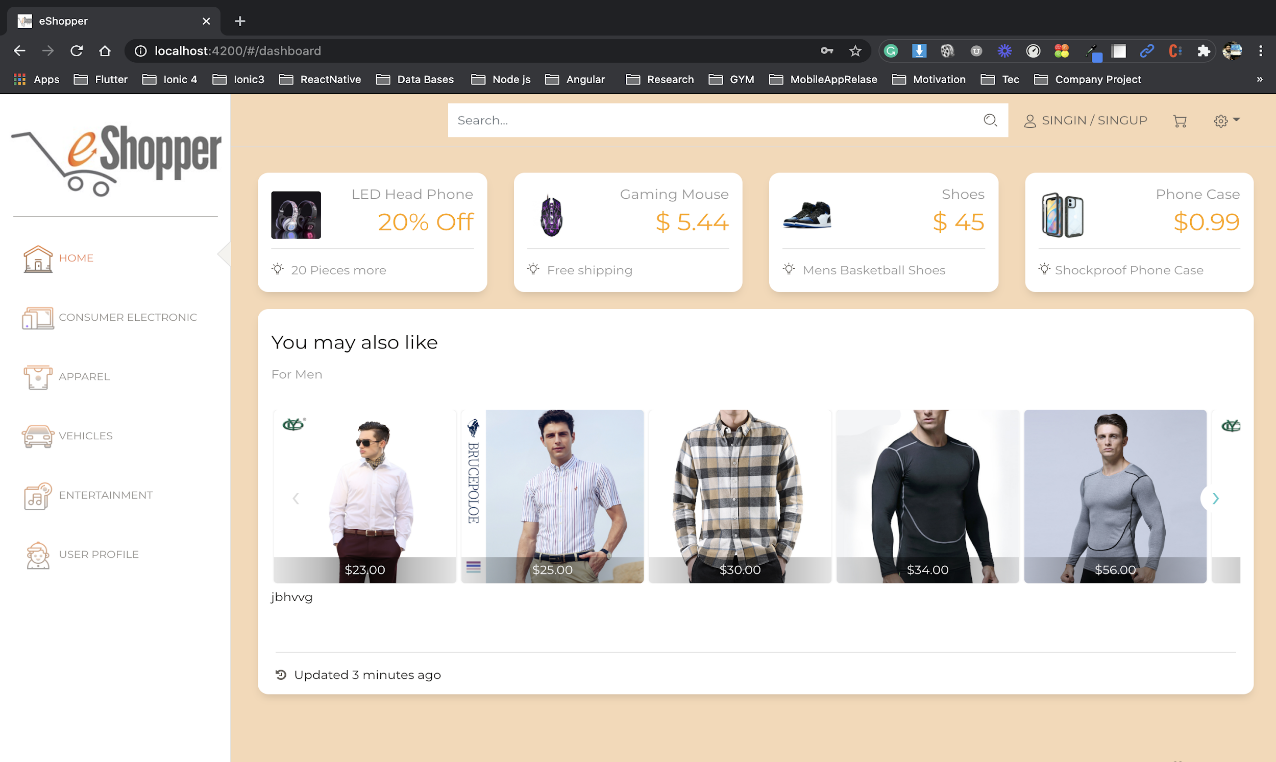
        return $this->belongsTo(InvInvoice::class, 'inv\_id');

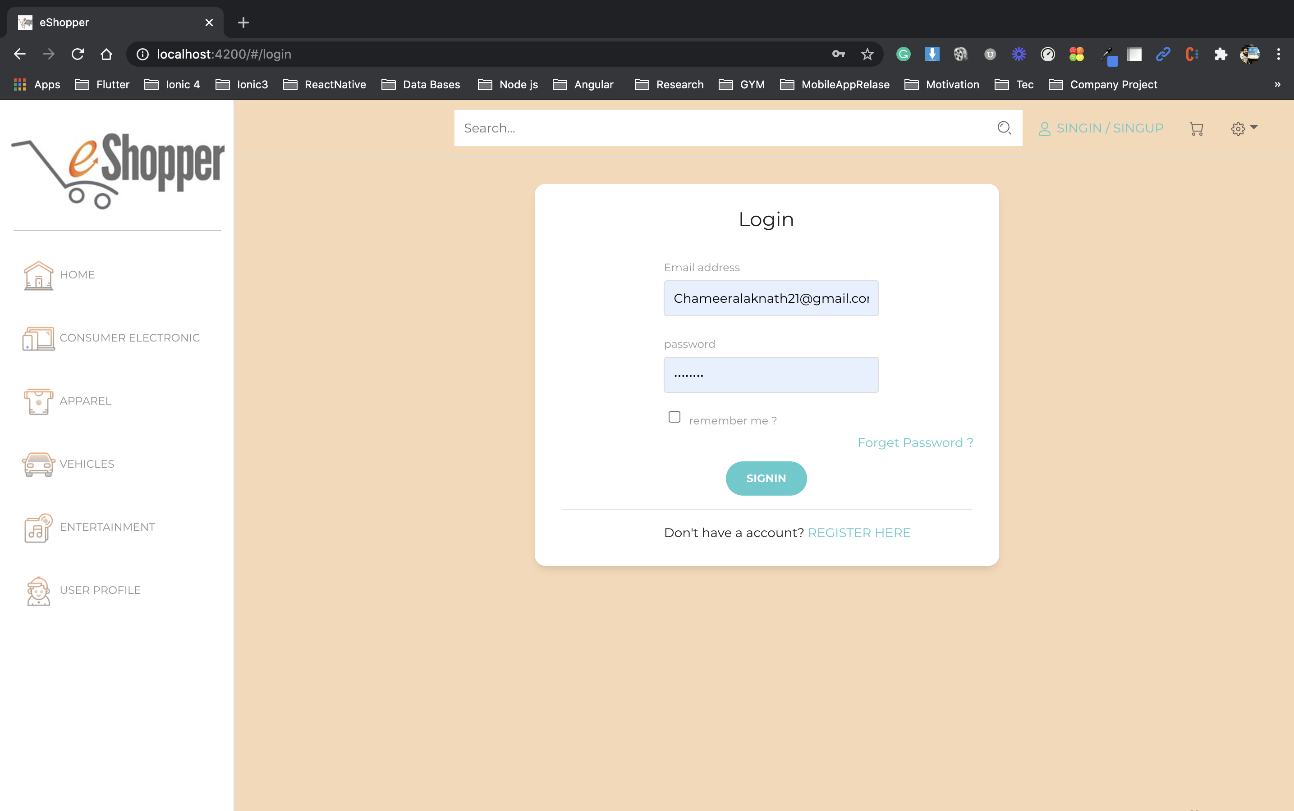
    }

}

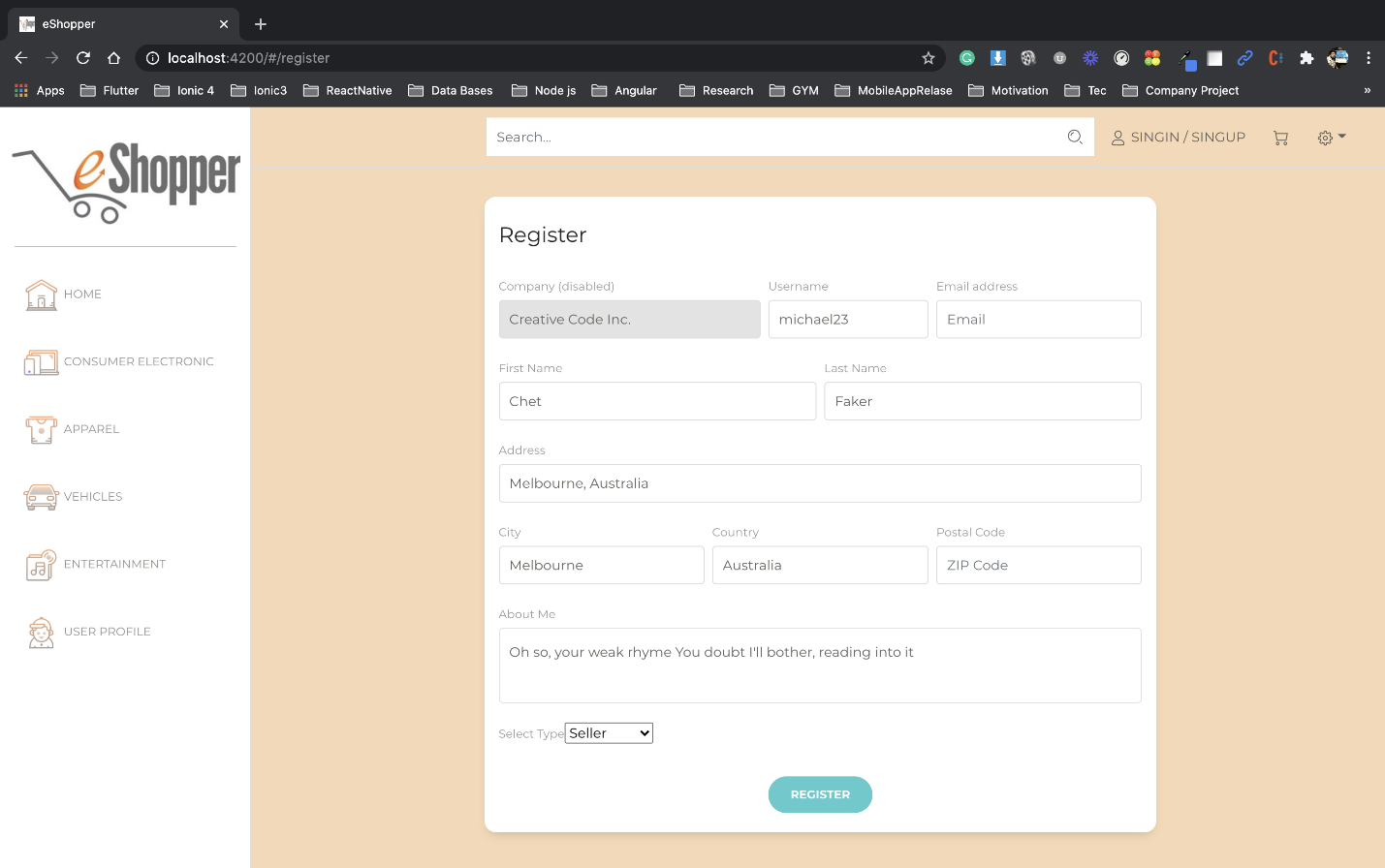
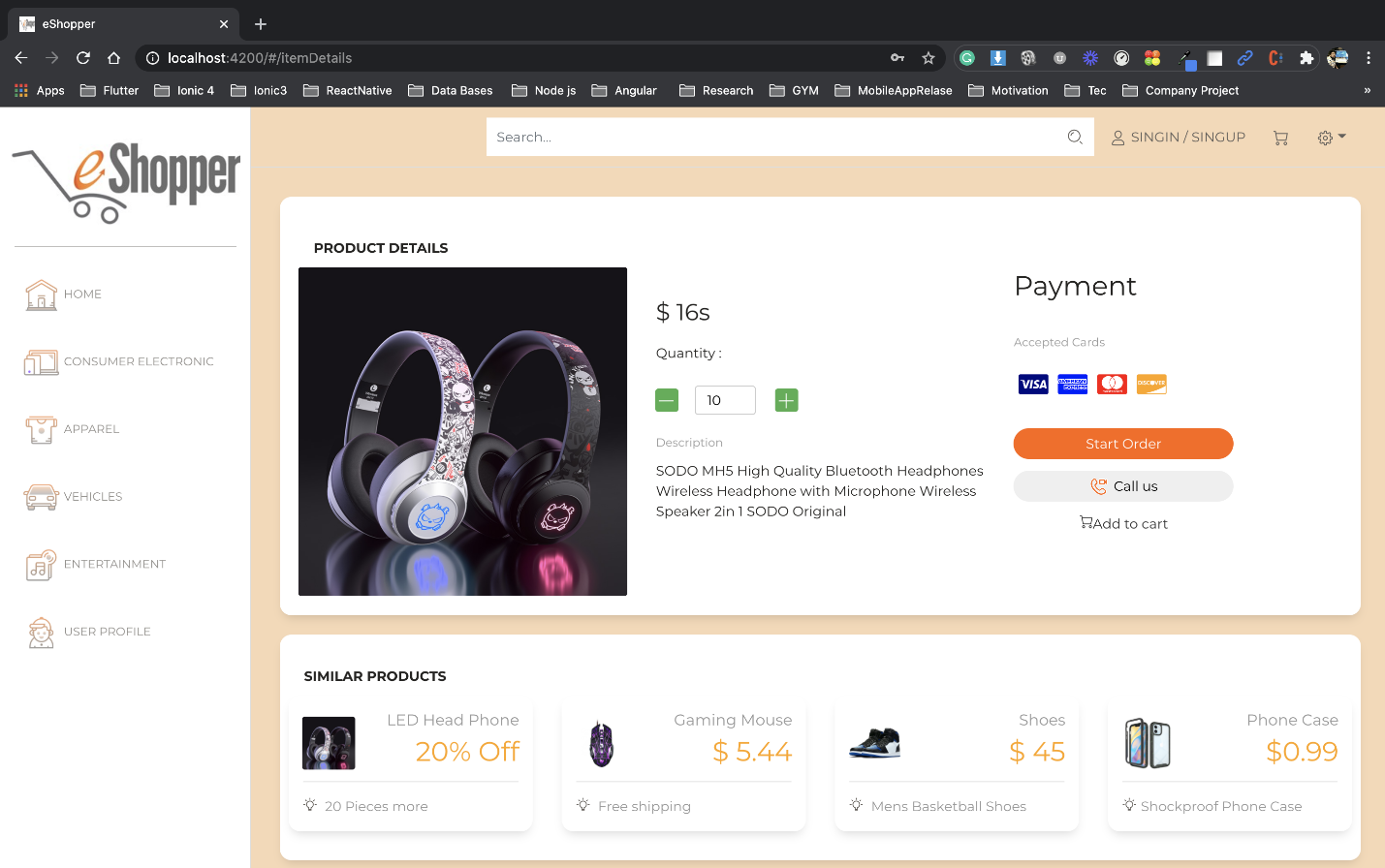
**Model connects to the Database through ORM (Object Relational Mapper)**

# **Screenshots of working outputs**

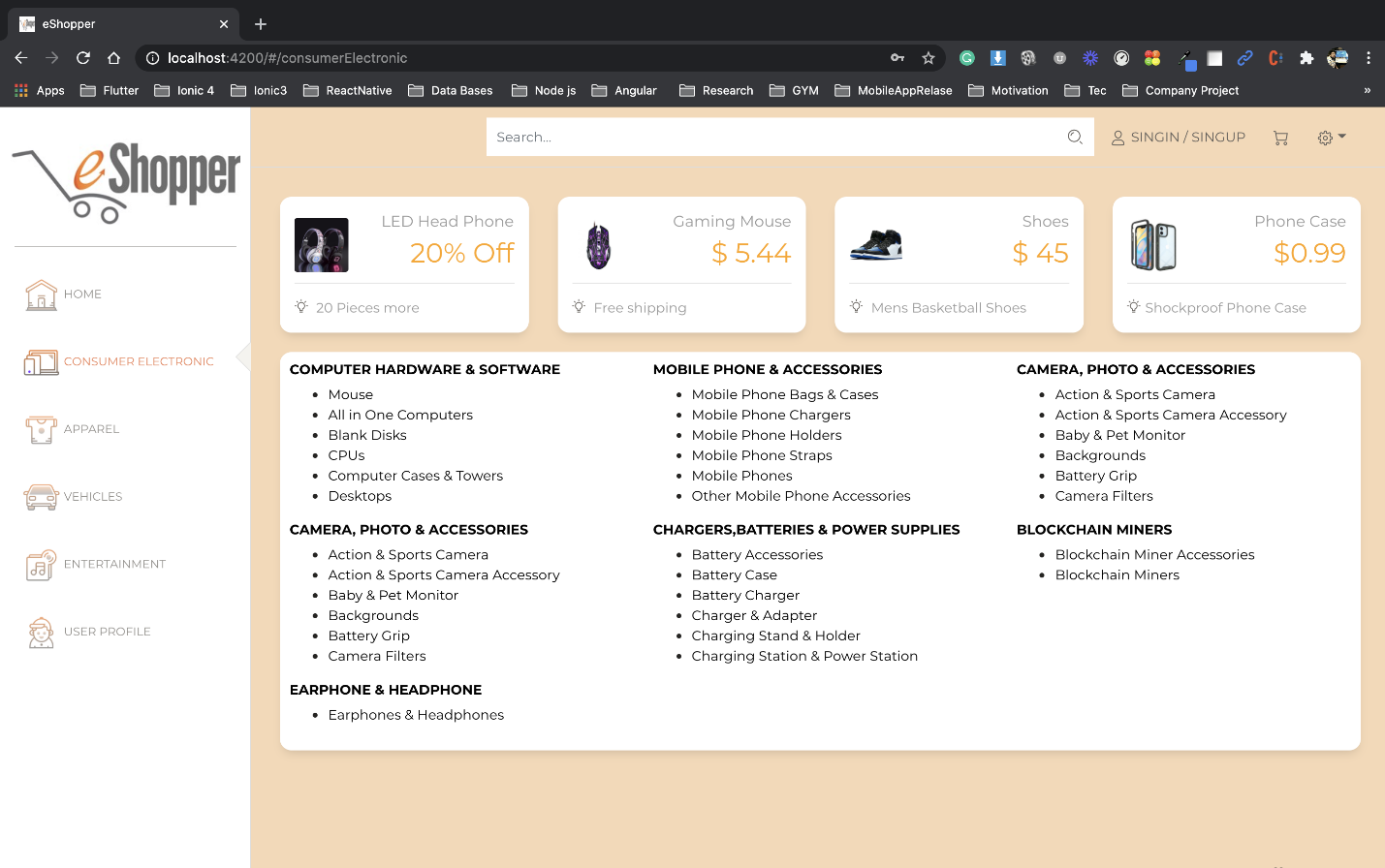
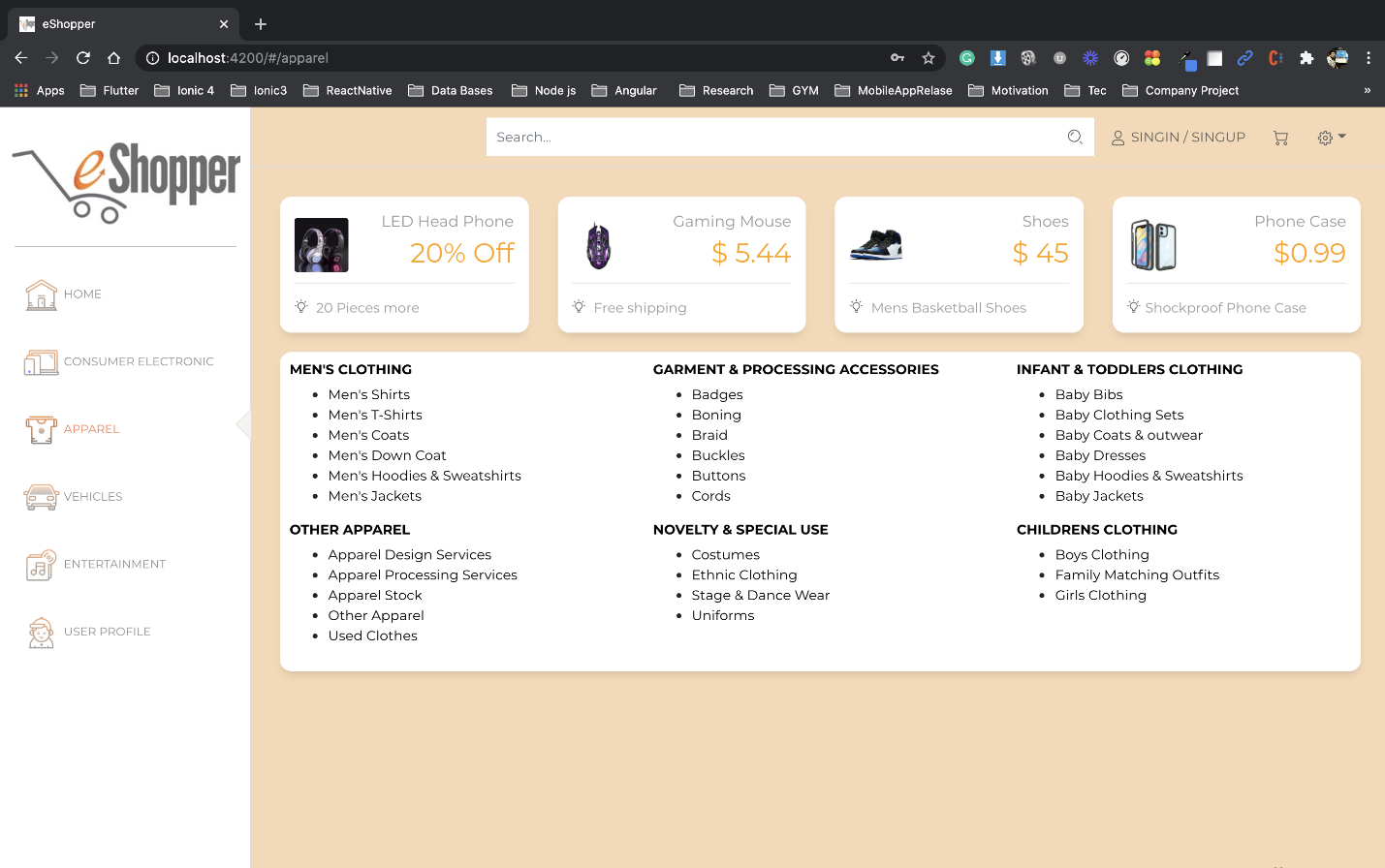
Dashboard

Login

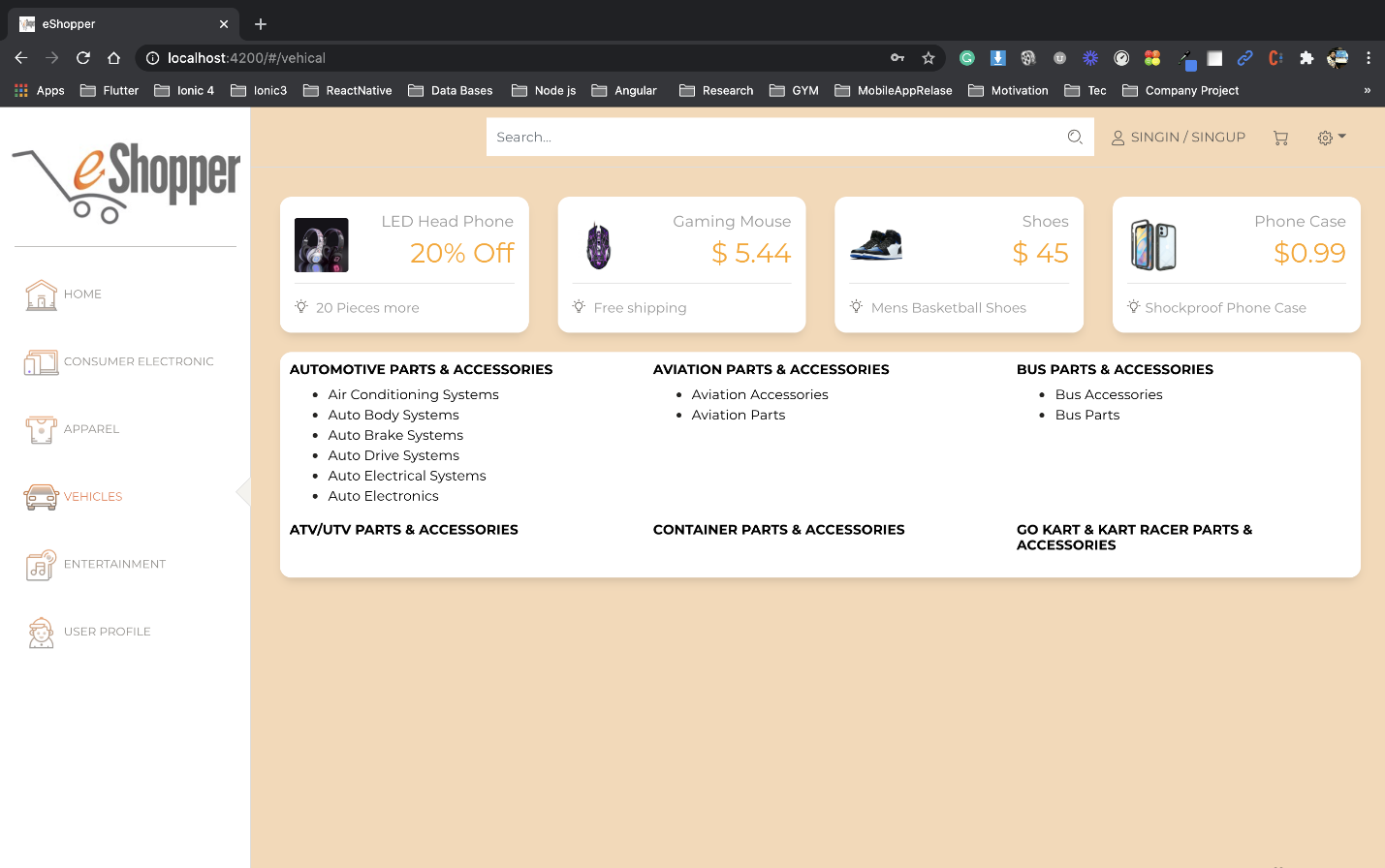
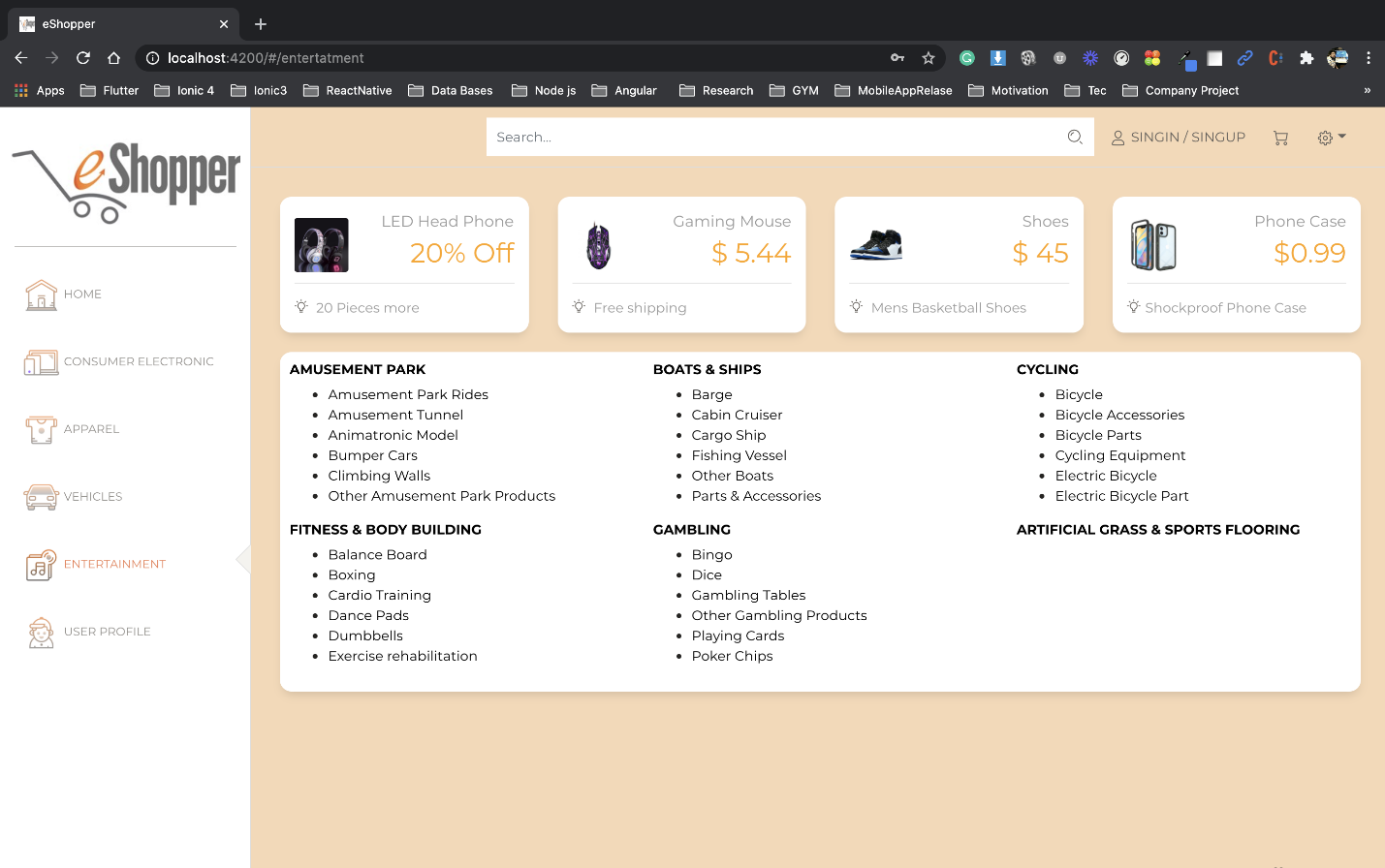
Register

Item Details

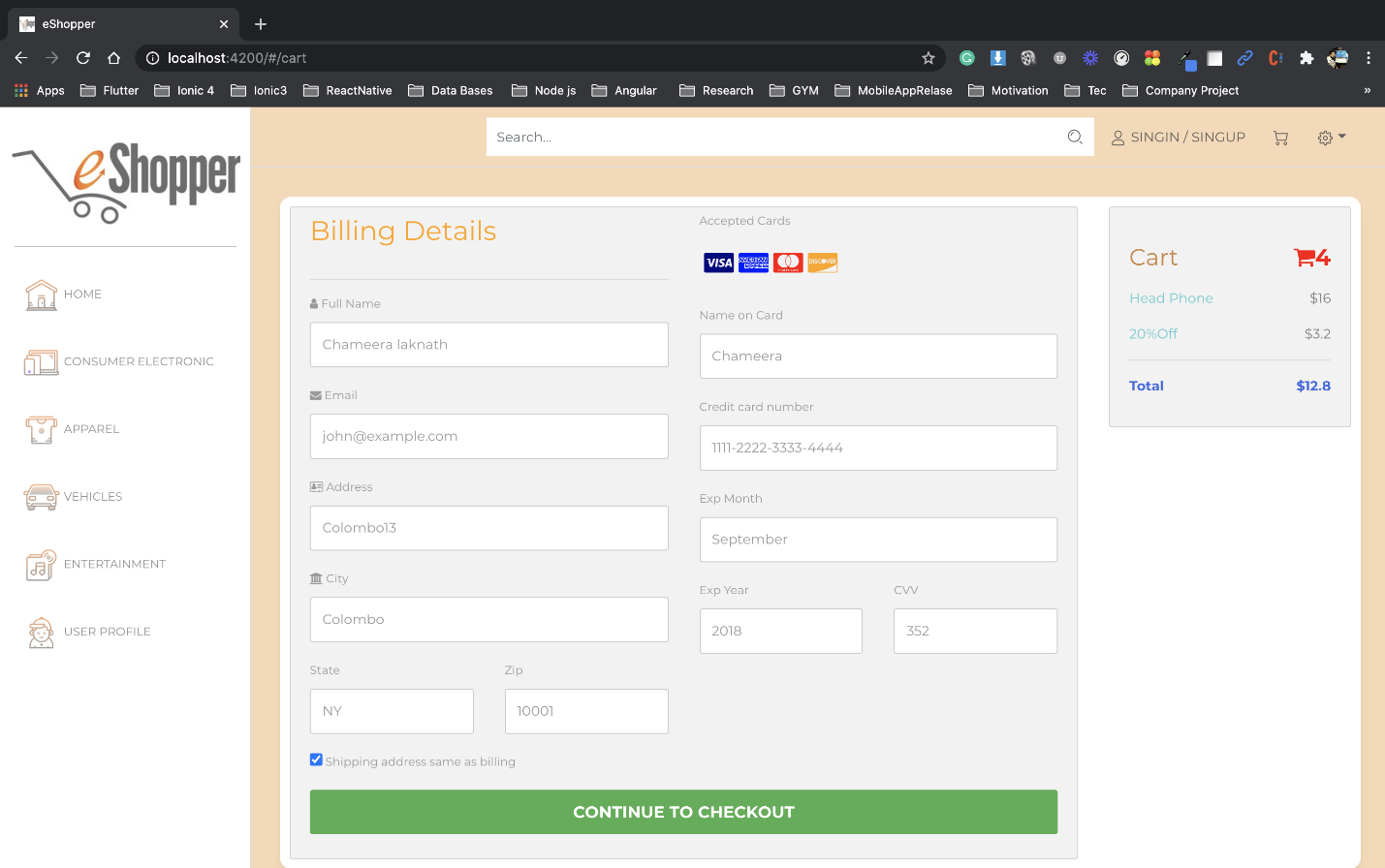
Consumer Electronics

Apparel

Vehicle

Entertainment

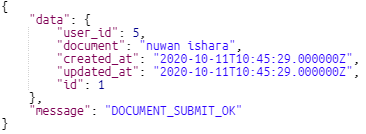
Billing Details

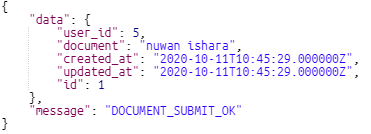


# **REST API outputs**

## **Auth Service**

* + 1. **Admin Task**
* **Admin Assist**

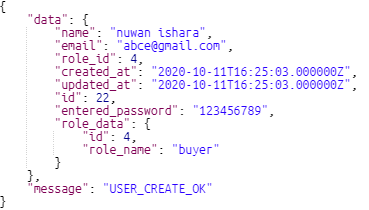
**Get Pending Verifications**

**Seller Verified by Admin**

* **User Management**

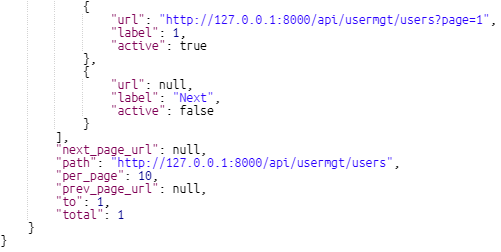
**Create User - Error Message**



**Create User - Success Message**

**Get All Users**





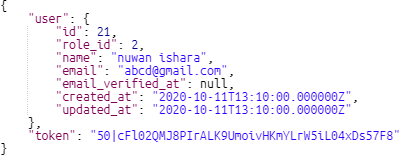
**Get User by id**



**Update User**

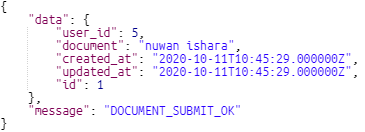
* + 1. **Login**

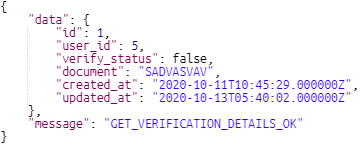
**Login**



* + 1. **Registration**
* **Seller**

**Seller Registration Step 2**

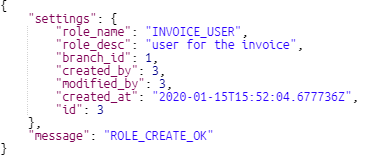


**View Seller Registration Status**

* + 1. **Role**

**Create Role - Error Message**



**Create Role - Success Message**

**Delete Role - Error Message 1**



**Delete Role - Error Message 2**



**Delete Role - Success Message**

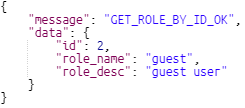


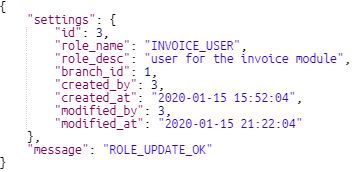
**Get all Role by Name**



**Get all Roles**

**Get Role by id**



**Update Role**

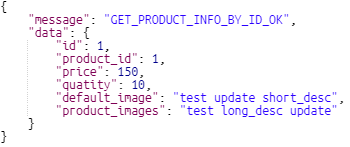
## **Inventory Service**

* + 1. **Product Info**

**Create Product - Error Message**



**Get Product info by id**

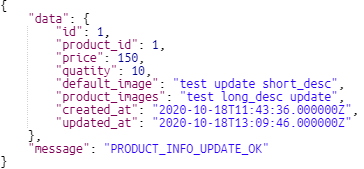


**Reduce product quantity – Error**



**Reduce product quantity – Success**



**Update Product**

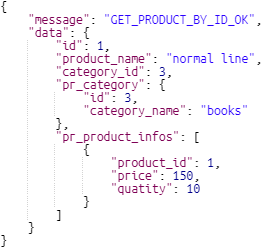
* + 1. **Products**

**Get all Products**





**Get Product by id**



**Get all Product by Name**

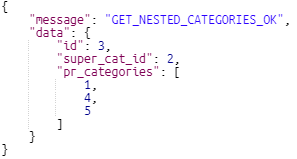


* + 1. **Products Categories**

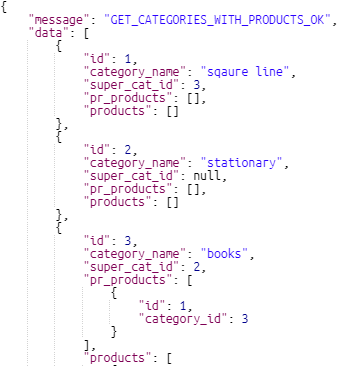
**Delete Category**

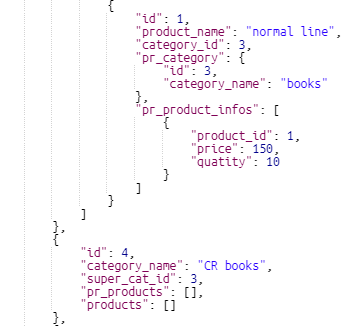


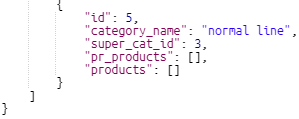
**Get all Categories Nested**



**Get all Categories with products**

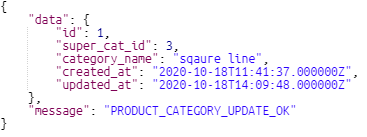






**Update Categories Error Message**

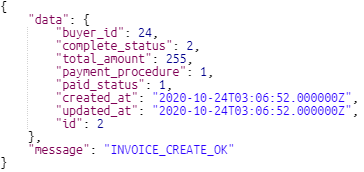


**Update Categories Success Message**

## **Order Service**

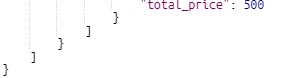
* + 1. **Invoice**

**Create invoice**

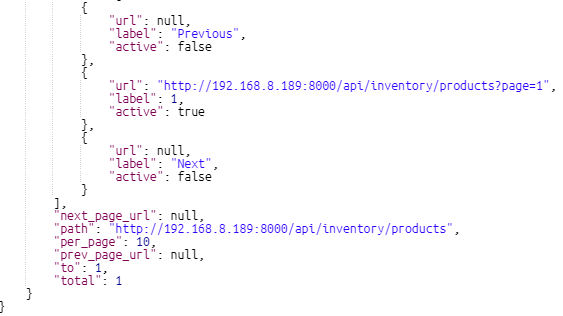


**Get all invoices of user**





**Get all invoices**



**Get Invoice by id**



**Update delivery date - Error Message**

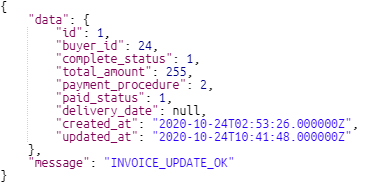


**Update delivery date - Success Message**



**Update status invoice - by seller Error Message**



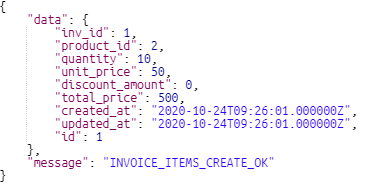
**Update status invoice - by seller Success Message**

* + 1. **Invoice Items**

**Create invoice items duplicate product add to the invoice**



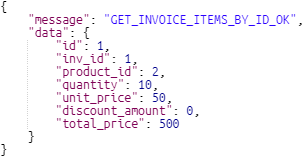
**Create invoice items**



**Delete Invoice item**



**Get Product info by id**



**Update invoice items - Error Message**



**Update invoice items - Success Message**

