Laravel 11 Vue Inertia SPA Магазин - гипермаркет

**Урок 1: Обзор план**



**Урок 2: Основной функционал**

---

**Урок 3: Определяем сущности и атрибуты**

User, Category, Product parent и Product (м.б. Product и Product\_variant), Param (отношение м/у Product и Param многие ко многим, pivot product\_params со значениями size, color и т.д.), Card (объединение User и Product) – product user card, Order, Image.

Пока схема такая (потом может измениться):



**Урок 4: Создание проекта окружения миграций моделей**

composer create-project laravel/laravel market

**Урок 5: Атрибуты в миграциях**

Ctrl+alt+S – выбрать версию php

В таблице с параметрами (params) дополнительно задаем тип фильтра (select, диапазон, checkbox и т.п.):

$table->unsignedSmallInteger('filter\_type');

В файле Providers/AppServiceProvider.php убираем защиту для данных моделей:



Построение диаграммы (выбрать таблицы и Ctrl+alt+u)





**Урок 6: Админка контроллеры и реквесты**

В контроллерах создаем namespace для админов:

****

php artisan make:controller Admin/CategoryController -r -m Category

php artisan make:controller Admin/ParamController -r -m Param

php artisan make:controller Admin/ProductParentController -r -m ProductParent

php artisan make:controller Admin/ProductController -r -m Product

php artisan make:request Admin/Category/StoreRequest

php artisan make:request Admin/Category/UpdateRequest

…

php artisan make:resource Product/ProductResource

…

Установка:

https://bootcamp.laravel.com/inertia/installation



composer require laravel/breeze --dev

php artisan breeze:install vue

php artisan make:class Services/ProductService

…

**Урок 7: Админка роуты сервисы крад**

Создаем систему ролей (многие ко многим) у одного пользователя может быть много ролей и одной роли может быть много пользователей

php artisan make:model Role -m

php artisan make:migration create\_role\_user\_table

php artisan make:enum Enums/Role/RoleEnum

**Урок 8: Кастомный аргумент в ресурсном роуте**

Route::resource('product-parents', ProductParentController::class)  
 ->parameters(['product-parents' => 'productParents']);

Получаем список роутов (php artisan route:list):



**Урок 9: Админка сервисы для крада**

public static function update(array $data, Product $product) : Product  
{  
 $product->update($data);  
  
 return $product->fresh();  
}

fresh – нужен для того, если у продукта будут отношения и в update они будут менятся, то laravel их перезагрузит.

**Урок 10: Админка фронт заготовка**

При успешно регистрации у нас создается редирект на дашборд:



Изначальный переход:



**Урок 11: Админка фронт индексовые страницы**

Регистрируем компонент, чтобы мы могли его использовать как ссылку, как в обычном blade:

import { ***Link*** } from "@inertiajs/vue3";



**Урок 12: Админка фронт добавление категорий**

Для отображения подсказки в теге select:





**Урок 13. Админка фронт крад категорий**

Иконки (нужно на выбранной иконке нажать copy svg):

<https://heroicons.com/>

**Урок 14: Админка фронт крад характеристик**

php artisan make:enum Enums/Param/ParamFilterTypeEnum

Два разных типа фильтра диапазон (INTEGER) и список (SELECT):



**Урок 15. Админка фронт крад группы продуктов**

product\_parent переименовали в product\_group

**Урок 16. Админка фронт добавление продуктов**

---

**Урок 17. Админка фронт продукты с изображениями**

*Добавление*:

Если на html тег мы навешиваем какое-либо событие, то навешивается объект event, например, мы передаем два файла:





Для того, чтобы изображения долетели до backend необходимо просто передать данный объект события:



Также передать специальный заголовок:



В StoreRequests:



Создаем сервис для обработки ImageService:



И вызываем его:



Если мы работаем в requests, то в event прописывать ничего не надо, т.к. это одна из точек входа.

В модели:



В ресурсах:



Добавляем поле изображения в общий ресурс:



Создаем отношение в модели:



Выводим изображения:



Картинки не отображаются:



Смотрим (в пути для картинок указан localhost):



Это происходит из-за настроек .env:



Но у меня была другая ошибка

**Урок 18. Админка фронт продукты с изображениями 2**

*Удаление*:

Для каждой картинки добавляем кнопку:

<div class="flex">  
 <div v-for="image in product.images">  
 <img :src="image.url" :alt="product.title" class="mb-2">  
 <div class="text-center">  
 <a href="#" @click.prevent="deleteImage(image)" class="inline-block px-2 py-1 text-sm bg-red-800 text-gray-200 border border-red-900">Удалить</a>  
 </div>  
 </div>  
</div>

Добавляем контроллер:

php artisan make:controller Admin/ImageController

Добавляем маршрут:

Route::delete('image/{image}', [ImageController::class, 'destroy'])->name('images.destroy');

Создаем контроллер:



Добавляем метод в сервис:



*Обновление (работает как добавление)*:

В UpdateRequests:



Редактируем метод сервиса



Модифицировать метод updateProduct (для метода PATCH заголовки не работают, поэтому заменили его на POST через модификатор в data):





**Урок 19. Админка фронт характеристики обзор стратегии**

Сложность возникает в связи с тем, что нет привязки к конкретному товару и разные товары могут иметь разные характеристики. Мы имеем дело с абстракцией. Например, остатки брюк разного размера будут разными.

У нас будет родительский продукт. Добавлять будем через создание дочернего элемента.

**Урок 20. Админка фронт продукты с характеристиками**

$table->unsignedBigInteger('article')->unique();

$table->foreignId('parent\_id')->index()->nullable()->constrained('products');

Массив params хранит объекты с id, value, …

При добавлении характеристики можно сделать ее уникальной.

Структура продукта:



Дополнительная валидация:



**Урок 21. Админка фронт продукты с характеристиками 2**

Доработка изображений

Работаем с глобальным массивом $refs. Используем для поиска тега input:

<div class="mb-4">  
 <input ref="image\_input" @change="setImages" type="file" multiple class="border border-gray-200 p-2 w-1/4">  
</div>

Очищаем содержимое:

this.$refs.image\_input.value = null

При удалении продукта изображения должны также удалятся. Сделаем для этого event.

php artisan make:observer ProductObserver -m Product

public function deleting(Product $product): void  
{  
 foreach ($product->images as $image) {  
 Storage::disk('public')->delete($image->path);  
 $image->delete();  
 }  
 $product->params()->detach();  
 $product->children()->delete();  
 //$product->params()->sync([]);  
}

В модели добавляем наблюдателя:

#[ObservedBy(ProductObserver::class)]

**Урок 22. Админка фронт редактирование продукта**

**Делаем возможность удалить ненужный параметр внутри продукта**

<svg @click="removeParam(entriesParam)" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5" stroke="currentColor" class="cursor-pointer size-6">  
 <path stroke-linecap="round" stroke-linejoin="round" d="m9.75 9.75 4.5 4.5m0-4.5-4.5 4.5M21 12a9 9 0 1 1-18 0 9 9 0 0 1 18 0Z" />  
</svg>

removeParam(entriesParam) {  
 this.entries.params = this.entries.params.filter(res => res !== entriesParam)  
}

**Урок 23. Админка фронт редактирование продукта 2**

В метод обновления добавляем (делаем для пропадании записи «Успешно сохранено!» при изменении полей):

this.$nextTick(() => {  
 this.success = true  
})

Это означает, что функция колбека выполнится после обновления DOM

**Урок 24. Админка фронт дочерние продукты**

Создаем компонент CreateChild.vue на основе двух: Create.vue и Edit.vue

**Урок 25. Админка фронт дочерние продукты отрисовка**

Реализовали отображение дочерних продуктов, использовали вложенный <template>, ввели новый компонент:

<tbody class="bg-white dark:bg-slate-800">  
<template v-for="product in productsData">  
 <ProductItem :product="product" @product\_deleted="updateProductsData" @product\_children\_got="updateProductsChildrenData"></ProductItem>  
 <template v-if="product.productChild" v-for="productChild in product.productChild">  
 <ProductItem :product="productChild" :is-child="true" @product\_deleted="updateProductsData"></ProductItem>  
 </template>  
</template>  
</tbody>

**Урок 26. Админка фронт дочерние продукты отрисовка 2**

---

**Урок 27. Админка фронт дочерние продукты удаление**

---

**Урок 28. Админка фронт ошибки валидации**

---

**Урок 29. Админка фронт добавление категории доработка**

---

**Урок 30. Клиент список продуктов**

---

**Урок 31. Админка фронт доработка функционала**

Если у родителя есть 2 потомка с разными характеристиками, то при удалении будет ошибка, т.к. согласно этому коду идет запрос напрямую в SQL.



Надо всех перебрать в массиве:



В реально практике лучше использовать softdelete, тогда этого можно было бы и не делать. Можно и БД каскадное удаление сделать, но это более глубокий уровень, что не очень.

Нужно еще добавить проверку целостности транзакций:



Если еще заморочиться, то изображения надо отдельно добавлять и в случае нарушения транзакции в цикле удалять из сервера.

**Урок 32. Добавляем данные для дальнейшей работы**

Добавили разные позиции в БД

**Урок 33. Клиент продукты по категориям**

По маршруту categories/{category}/products будем видеть все продукты, которые принадлежать к данной категории.

**Урок 34. Клиент продукты по иерархии категории**

Сейчас мы видим у одной категории товары только, которые ей принадлежат, но мы хотим видеть товары дочерних категорий.

Сделали рекурсивный вызов в CategoryService:

public static function getCategoryChildren(Category $category) : array  
{  
 $arr = [];  
 $categoryChildren = Category::*where*('parent\_id', $category->id)->get();  
  
 foreach ($categoryChildren as $categoryChild) {  
 $arr = array\_merge($arr, self::*getCategoryChildren*($categoryChild));  
 }  
  
 $arr[] = $category;  
  
 return $arr;  
}

Вызов в контроллере CategoryController:

public function productIndex(Category $category) {  
 $categoryChildren = CategoryService::*getCategoryChildren*($category);  
  
 $products = ProductResource::*collection*(ProductService::*indexByCategories*($categoryChildren))->resolve();  
  
 return inertia('Client/Category/ProductIndex', compact('products'));  
}

и доп. сервис ProductService:

public static function indexByCategories($categoryChildren) {  
 return Product::*whereIn*('category\_id', array\_column($categoryChildren, 'id'))  
 ->whereNull('parent\_id')  
 ->get();  
}

Последний код делаем через scope[[1]](#footnote-1):

public static function indexByCategories($categoryChildren) {  
 return Product::*byCategories*($categoryChildren)->get();  
}

и в модели сам scope Product:

public function scopeByCategories(Builder $builder, array $categoryChildren) {  
 return $builder->whereIn('category\_id', array\_column($categoryChildren, 'id'))  
 ->whereNull('parent\_id');  
}

**Урок 35. Клиент хлебные крошки по категориям**

Для этого нужно найти все родительские категории.

public static function getCategoryParents(Category $category) : array  
{  
 $arr = [];  
  
 if ($category->parent\_id) {  
 $parentCategory = Category::*find*($category->parent\_id);  
 $arr[] = $parentCategory;  
 $arr = array\_merge($arr, self::*getCategoryParents*($parentCategory));  
 }  
  
 return $arr;  
}

**Урок 36. Клиент хлебные крошки по категориям оптимизация**

По конвенции laravel все должно быть через коллекцию

**Урок 37. Клиент характеристики по хлебным крошкам**

Задача построить фильтр для этого нужно выбрать только те характеристики, которые присущи данным товарам в хлебных крошках. Чем выше мы поднимаемся по хлебным крошкам, тем больше характеристик должно выводится.

Для этого мы должны через категорию создать отношение с param\_product через продукты. Это будет отношение один ко многим через. Т.е. одна категория имеет много param\_product. При этом если у нас присутствует только одно значение характеристики, например, есть только размер 56, то нет смысла делать фильтр.

Работу с характеристиками поделим на 3 этапа: выборка, отрисовка, работа фильтра.

Схема данных (выбрали нужные таблицы: categories, param\_product, products и через контекстное меню Diagrams… )



В модели Category:

public function paramProducts() : HasManyThrough  
{  
 return $this->hasManyThrough(ParamProduct::class, Product::class, 'category\_id', 'product\_id', 'id', 'id');  
}

Из массива характеристик выделили уникальные:

$params = Param::*whereIn*('id', $arr->pluck('param\_id'))->get();

Сгруппировали:

$arr = $arr->groupBy('param\_id');

И далее добавим существующие, но уникальные значения c сортировкой:

foreach ($params as $param) {  
 $param->param\_values = $arr[$param->id]->unique('value')->sortBy('value')->pluck('value')->toArray();  
}

**Урок 38. Клиент фильтр отрисовка**

---

**Урок 39. Клиент фильтр подготовка данных на фронте**

Будем реализовывать следующую структуру фильтра (в коде id и значения уберем):  


**Урок 40. Клиент фильтр выборка на беке**

В CategoryController делаем так:?

Если мы будем отправлять данные для асинхронного запроса, если $request хочет json. Т.к. первый раз когда мы загружаем страницу это не json-запрос.

if ($$request->wantsJson()) {  
 return $products;  
}

**Урок 41. Клиент фильтр выборка числу**

Для фильтрации по числам необходимо использовать whereRaw:



**Урок 42. Клиент фильтр шаблон**

Builder заменяет класс Model, вроде как любой:

protected function checkbox(Builder $builder, $value) {  
 if (isset($data['filters']['select'])) {  
 $builder->whereHas('paramProducts', function ($q) use ($data) {  
 foreach ($data['filters']['checkbox'] as $key => $value) {  
 $q->where('param\_id', $key)->whereIn('value', $value);  
 }  
 });  
 }  
}

php artisan make:trait Models/Traits/HasFilter

trait HasFilter  
{  
 public function scopeFilter(Builder $builder, $data) : Builder  
 {  
 $filter = new ProductFilter();  
  
 return $filter->apply($builder, $data);  
 }  
}

scope – делает какой-либо запрос в БД через модель.

Вызывается без scope:

$products = Product::*byCategories*($categoryChildren)->filter($data);

Но в трейте должна быть абстрактная реализация:

Конструкция:

$className = class\_basename($this);  
dd($className);

дает название класса:



Тогда (желательно проверить класс на существование): Good : получение класса из строки!

trait HasFilter  
{  
 public function scopeFilter(Builder $builder, $data) : Builder  
 {  
 $className = 'App\\Http\\Filters\\' . class\_basename($this) . 'Filter';  
 $filter = new $className();  
  
 return $filter->apply($builder, $data);  
 }  
}

Если захотим добавить фильтр, например, по именам. То надо добавить ключ в ProductIndexRequest, в самом ProductFilter в массив ключей и реализацию под новый ключ.

Здесь тоже нужно добавить проверку на существование метода: Good : получение метода из строки!

class AbstractFilter  
{  
 protected array $keys = [];  
  
 public function apply(Builder $builder, array $data) : Builder  
 {  
 foreach ($this->keys as $key) {  
 if (isset($data[$key])) {  
 $methodName = Str::*camel*($key);  
 $this->$methodName($builder, $data[$key]);  
 }  
 }  
  
 return $builder;  
 }  
}

**Урок 43. Реплика витринного продукта**

Изменение логики создания дочернего продукта. Идея в том, что у продуктов при нажатии на плюс будем создавать его клон и делать редирект на edit. В этом случае мы избавляемся от ненужного контроллера для создания потомков, освободит от view CreateChild и сразу решаем проблемы, связанные с изображениями, характеристиками и т.д.



**Урок 44. Реплика витринного продукта ошибка**

**---**

**Урок 45. Админка дорабатываем таблицу с продуктами**

Переключение цвета фона для дочерних и родительских компонентов:

<td :class="[product.parent\_id ? 'bg-gray-50' : 'bg-white', 'text-center border-b border-slate-100 dark:border-slate-700 p-4 text-slate-500 dark:text-slate-400']">{{ product.id }}</td>

Проверка на наличие потомков:

public function getHasChildrenAttribute() : bool  
{  
 return $this->children()->exists();  
}

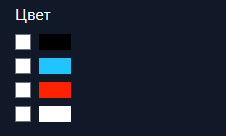
**Урок 46. Клиент фильтр стилизация**

Создаем доп. поле

php artisan make:migration add\_column\_label\_to\_params\_table

Если это поле присутствует и равно ‘color’, то отображаем в виде палитры:

<label v-if="param.label === 'color'" :style="`background: ${value}; width: 32px; height: 16px`" class="block text-sm text-gray-200" :for="value"></label>  
<label v-else class="text-sm text-gray-200" :for="value">{{ value }}</label>



**Урок 47. Клиент карточка товара**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

**Урок 5: Определяем сущности и атрибуты**

1. Использование скоупов в Laravel позволяет упростить код, избавиться от повторяющихся условий в запросах и сделать его более понятным и поддерживаемым. **Локальные скоупы**: Это методы в модели, которые добавляют условия к запросу. Они могут быть вызваны в любом месте вашего кода. [↑](#footnote-ref-1)