

Проектна задача

по предметот

Имплементација на системи со отворен код

Веб продавница за нарачка на храна

Имплементација на веб апликација во Laravel

Имплементација на административски портал со примена на
Laravel NOVA

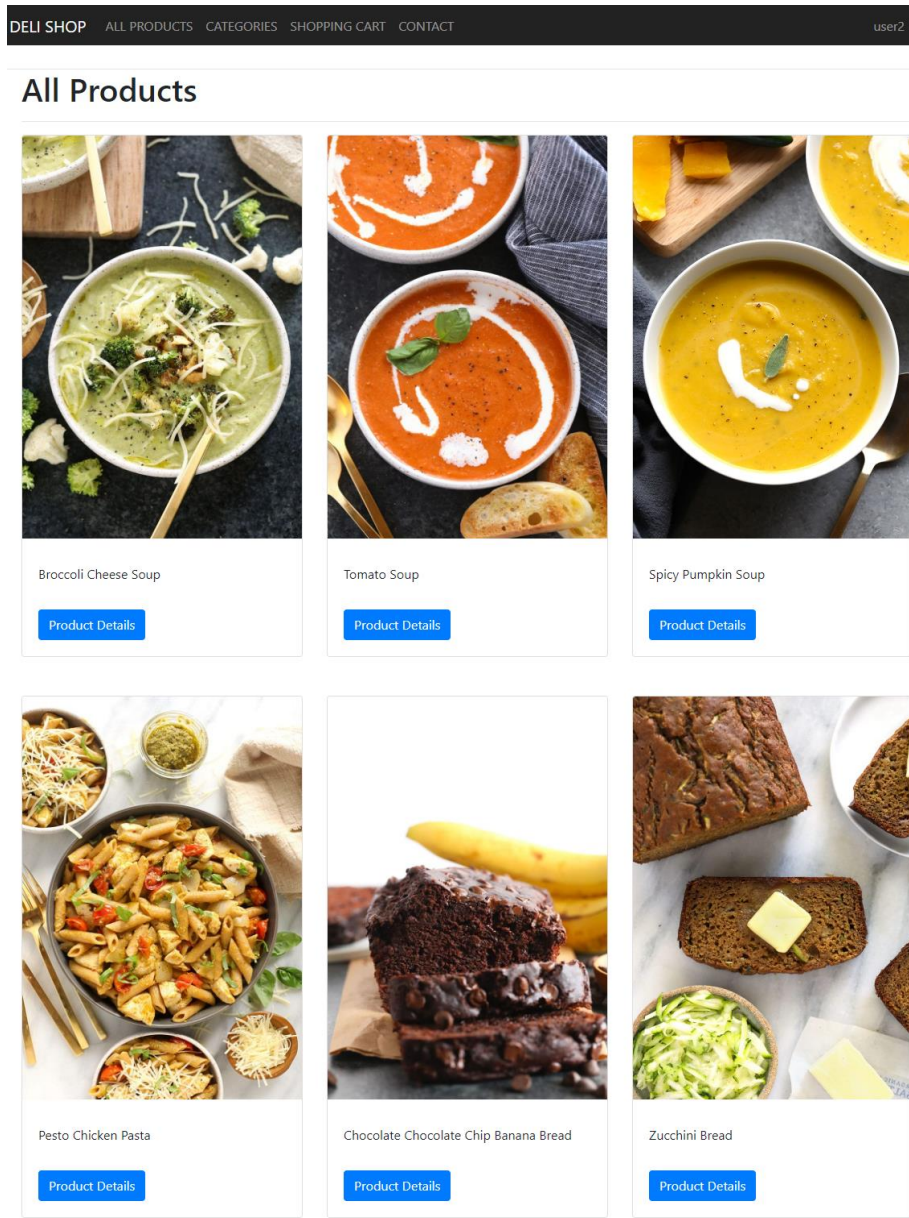
Технологии: Laravel 8 Laravel NOVA 3.27.0

Изработено од Сандра Неделчева 173097

Јуни 2022

1. Вовед

За потребите на оваа проектна задача, создадена е веб страница за нарачка на оброци. Корисникот може ги прелиста сите продукти одеднаш или да ги прелистува по категорија. Тој има можност да селектира и детално да разгледа даден продукт, како и да остави коментар и оцена за него. И се разбира, корисникот може да поставува продукти во сопствена кошничка за купување и подоцна да ги наплати со помош на кредитна картичка.



За сопствениците на бизнисот, имплементиран е административен портал кој им овозможува менаџмент на веб страницата и базата на податоци преку интерфејс кој е лесен за користење, особено за лица без програмерско знаење.

Веб апликацијата е изградена со Laravel ver.8 рамка поради компатибилноста со расположливата верзија на Laravel NOVA 3.x. При нејзина изработка користена е алатката Composer.

2. Конфигурација за базата на податоци (Модели, табели и seeders)

Оваа веб апликација користи Postgres база на податоци. Проектот е приспособен за оваа база на податоци, со тоа што првично во php.ini документот одкоментирани се соодветните линии (**extension=pdo_pgsql** и **extension=pgsql**). Потоа, во .env се внесуваат податоците потребните за конектирање кон базата за податоци.

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=php_test
DB_USERNAME=postgres
DB_PASSWORD=
```

На крај, во config/database.php, се прави промена на која ќе е default-ната/основната база за податоци?

'default' => env('DB_CONNECTION', 'pgsql')

Со сите овие направени промени во конфигурацијата на оваа апликација, сега е можно да се создаваат модели, табели и seeders.

За потребите на оваа веб апликација, создадени се следниве модели и табели:

- Product
 - Модел кој го претставува продуктот кој може да биде купен
 - Табелата за овој модел во себе содржи:
 - Име на продуктот
 - Краток опис на продуктот
 - Листа на состојки зачувана како String
 - Цена на продуктот
 - URL адресе за слика
- Category
 - Модел кој ги претставува категориите во кои може да припаѓаат продуктите
 - Табелата за овој модел во себе содржи:
 - Име на категоријата
 - URL адресе за слика
- Category_Product
 - Пивот табела која ја опишува M-N релацијата помеѓу продуктот и категоријата (еден продукт може да припаѓа на повеќе категории, една категорија може да припаѓа содржи повеќе продукти)
 - Во неа се сместени надворешни клучеви кои покажуваат на претходно споменати модели/табели
- User
 - Модел кој го претставува корисникот т.е купувачот

- Се користи за да овозможува коментирање на продукти и наплата
- Табелата за овој модел во себе содржи:
 - Корисничко име
 - Email адресе на корисникот
 - лозинка
- Comment
 - Модел кој претставува коментар кој може да се постави оддаден корисник
 - Табелата за овој модел во себе содржи:
 - Содржина на самиот коментар
 - Оцена за продуктот
 - Датум кога тој е создаден
 - Надворешен клуч кој покажува кон корисникот кој го има оставено коментарот.
- Cart
 - Модел кој ја претставува самата кошничка во која корисникот поставува продукти.
 - Табелата за овој модел во себе содржи:
 - Датум кога таа е создаден
 - Надворешен клуч кој покажува кон корисникот на кого припаѓа количката
- CartItem
 - Модел кој претставува инстанца од еден продукт во дадена кошничка. Овој модел е создаден со цел да се овозможи поставување на поголема количина на даден продукт.
 - Табелата за овој модел во себе содржи:
 - Надворешен клуч кој покажува кон количката во која е поставен
 - Надворешен клуч кој покажува кон продуктот поставен во количката
 - Количина на продуктот/елементот
- Order
 - Модел кој претставува една нарачка.
 - Се користи за увид на нарачките во администрацискиот панел
 - Табелата на овој модел содржи:
 - Надворешен клуч кој покажува кон количката која е нарачана
 - Надворешен клуч кој покажува кон корисникот кој ја нарачал
 - Валута
 - Вкупна цена на нарачката

Моделите измеѓусебно се поврзани на следниот начин:

Главна страна	Инверзна страна	Тип на релација
Product	Category	M-N
User	Cart	1-1
Cart	CartItem	1-N
Product	CartItem	1-N
Product	Comment	1-N
Order	Cart	1-N
User	Order	1-N

Сите овие модели и релации се креирани со помош на Eloquent ORM (Object Relational Mapper), софтверски пакет кој овозможува менаџмент со бази на податоци, претставувајќи ги зачуваните информации во базата како објекти од модел класи. За создавање на модел се користи следнава artisan команда **php artisan make:model Product**.

Со цел релациите практично да се искористат во функционалноста на веб апликацијата, во моделите кои учествуваат во релацијата се напишани функции кои ја враќаат инстанцата на другиот соучесник поврзан со тој модел. Во моделот кој се наоѓа на главната страна на релацијата, се создава функција со името на елементот од инверзната страна, која враќа инстанца од неговата класа преку функцијата `hasMany()` или `hasOne()`, зависно од типот на релацијата. Истото се извршува и кај моделот од инверзната страна на релацијата, но со функцијата `belongsTo()` (за 1-1 и 1-N релации) или `belongsToMany()` (M-N релации).

Пр. Product- Comment (1-N релација) и Product-Category(M-N релација)

Во Product моделот

```
public function categories()
{
    return $this->belongsToMany(Category::class);
}

public function comments()
{
    return $this->hasMany(Comment::class);
}
```

Во Comment моделот

```
public function product()
{
    return $this->belongsTo(Product::class);
}
```

Во Category моделот

```
public function products()
{
    return $this->belongsToMany(Product::class);
}
```

За да се создаде табела во базата на податоци, првично се создава миграција со следнава артисан команда:

php artisan make:migration create_comment_table --create="comments"

Во миграцијата се дефинира изгледот на табелата. Овде мора да се обрати внимание на типот на информациите кои се чуваат во табелата, како и надворешните клучеви кои ги градат релациите.

```
Schema::create('comments', function (Blueprint $table) {
    $table->bigIncrements('id');
    $table->string('FullComment');
    $table->unsignedInteger('CommentRating');
    $table->dateTime('createDate');
    $table->integer('product_id')->unsigned()->nullable();
    $table->foreign('product_id')->references('id')->on('products')->onDelete('cascade');
    $table->timestamps();
});
```

Табелите крајно се создаваат со артисан командата **php artisan migrate**.

Самите табели се пополнети преку Seeders. При сам почеток на апликацијата, на располагање се четири категории (супи, главни јадења, десерти и лебови), 14 различни продукти и девет коментари. Даден seeder се создава првично со следнава команда:

php artisan make:seeder CommentsTableSeeder

Во seeder-от се поставуваат посакуваните почетни вредности за дадената табела/модел во листата

```
DB::table('comments')->insert(array (
    0 =>
        array (
            'id' => 12,
            'FullComment' => 'tasty but only when it\'s warm',
            'CommentRating' => 3,
            'createDate' => '2020-10-11 12:50:29',
            'product_id' => 1,
            'created_at' => new DateTime,
            'updated_at' => new DateTime,
        ),
```

На крај, табелата се пополнува со командата **php artisan db:seed**.

3. Контролери

За да може да се пристапи до различните веб страни, како и да се извршуваат основните операции со податоците во базата на податоци, потребно е да се создадат контролери. Тие може да се создадат со помош на командата **php artisan make:controller CartsController --resource**. Со помош на **--resource**, автоматски ги имаме креирано основните CRUD методи во новосозданиот контролер.

4. Рүти

Со рутите се дефинира која URL патека ќе мапирана на кој метод во контролерот. Во `route/web.php` фолдерот, може само да се повика команда која автоматски ќе ги креира патеките со задавање на контролерот и посакувано име кое ќе го претставува тој контролер во URL-то.

```
Route::post('/charge', 'CheckoutController@charge')->name('charge.charge');
Route::get('/charge', 'CheckoutController@index')->name('charge.index');
```

Доколку е потребно вгнезден приказ на URL патеките (на пр. за приказ на коментар поставен на даден продукт, потребна е патеката **products/{product_id}/comments/{comment_id}**), имињата се прилепуваат со точка (products.comments)

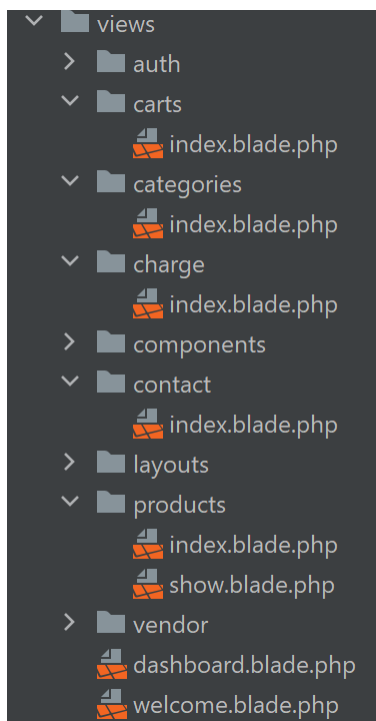
```
Route::resource('products', 'ProductsController');
Route::resource('categories', 'CategoriesController');
Route::resource('products.comments', 'CommentsController');
```

Со командата **php artisan route:list** се излитуваат сите достапни патеки и методите со кои се повикуваат.

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	categories	categories.index	App\Http\Controllers\CategoriesController@index	web
	POST	categories	categories.store	App\Http\Controllers\CategoriesController@store	web
	GET HEAD	categories/create	categories.create	App\Http\Controllers\CategoriesController@create	web
	GET HEAD	categories/{category}	categories.show	App\Http\Controllers\CategoriesController@show	web
	PUT PATCH	categories/{category}	categories.update	App\Http\Controllers\CategoriesController@update	web
	DELETE	categories/{category}	categories.destroy	App\Http\Controllers\CategoriesController@destroy	web
	GET HEAD	categories/{category}/edit	categories.edit	App\Http\Controllers\CategoriesController@edit	web

5. Views

Оваа веб апликација ги содржи слениве страни:



Веб страните т.е views се креирани со помош на Blade template engine, кој овозможува флексибилно креирање на HTML страници. Секоја страна е сместена во blade.php датотека која има исто име како и методата со која се повикувани. Даден views се повикува со view() функцијата. Преку view(), може да се пратат податоци за моделите кои треба да се прикажат во HTML кодот, со помош на compact() функцијата.

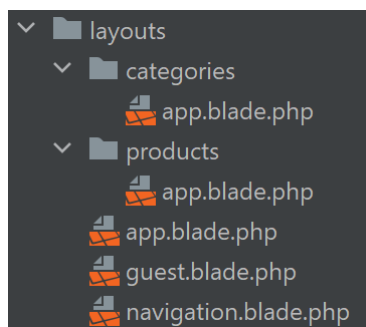
```
public function index()
{
    //
    $allcategories=Category::all();
    return view('categories.index',compact('allcategories'));
}
```

Blade содржи посебни тагови кои овозможуваат изведба на покомплексни операции како if-else (за приказ на даден сегмент на страната според даден услов), foreach loops(), итн.

```
<h3>Comments:</h3>
@if($product->comments==null)
    <h4>There are no comments on this product yet.</h4>
@else
    @foreach($product->comments as $comment)
        <div class="row">
            <div class="col">
                <div class="card border-light mb-3">
                    <div class="card-header">USER NAME HERE <span class="icon novi-
icon icon-md-middle icon-gray-1 mdi mdi-calendar-clock"></span><span>{{ $comment-
>createDate}}</span></div>
                    <div class="card-body">
                        <h5 class="card-title">Rating: {{ $comment->
CommentRating}}</h5>
                        <p class="card-text">{{ $comment->FullComment}}</p>
                    </div>
                </div>
            </div>
        </div>
    @endforeach
@endif
```

Структурата на веб страниците е поделена на два дела: статички и динамички дел.

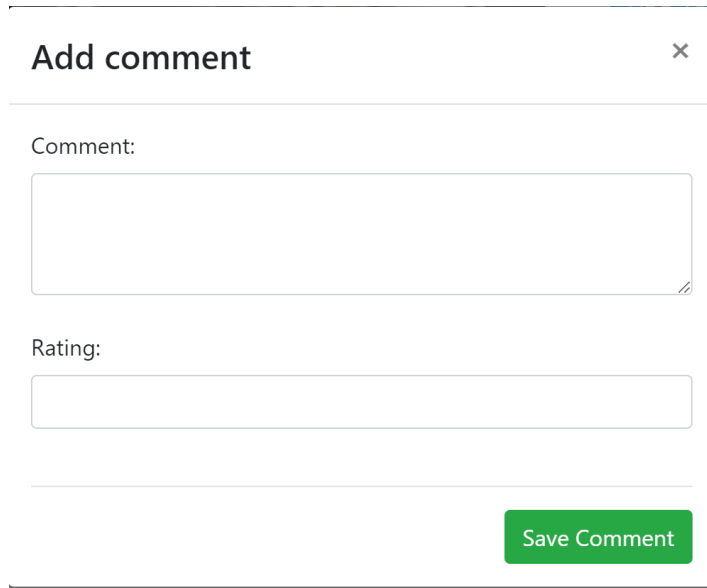
Статичкиот дел т.е layout-тот се состои од header и footer, т.е навигациско мени на врвот на страната и оддел за копчиња кои водат кон различни социјални мрежи на дното на страната. Навигациското мени има копчиња за листање на сите продукти, листање на сите категории, приказ на купувачката кошничка на корисникот, како и копчиња за регистрирање и логирање на корисник.



Статичкиот дел од страните се наоѓаат во `app.blade.php` датотеките за соодветните URL патеки. Динамичкиот дел се поставува на местото на `@yield('content')` тагот. Во главната страна, статичкиот дел се референцира преку тагот `@extends ('layouts.app')`

Во динамичкиот дел на страната се прикажува содржината и каде се главно се случува корисничкото искуство. Ова вклучува листање на продукти, преглед на продуктот, создавање на коментар, итн. Овој дел се дефинира помеѓу `@section ('content')` и `@endsection`.

Освен Blade, искористена е и Bootstrap рамката при изработка на изгледот на страниците, како и создавање на готови форми спремни за употреба.



Пример за таква форма е прозорецот за пишување на коментар. Формата во себе има дефинирано на која метода во кој контролер ќе се врати POST пакетот, како и имињата на полињата од кои ќе се земат новите вредности (Rating, commentBody, ProductId). Тие пристигнуваат до методот во форма на променлива од типот Request. Потоа може да се создаде нова инстанца за коментар која ќе се зачува во базата на податоци.

```
<form action="/products/{{ $product->id }}/comments" method="post" >
  <input type="text" hidden="hidden" name="ProductId" value="{{ $product->id }}" />
  <label>Comment:</label>
  <textarea type="text" class="form-control" rows="3" id="commentBody"
name="commentBody"></textarea>
  <br />
  <label>Rating:</label>
  <br />
  <input class="form-control" type="number" min="1" max="5" id="Rating"
name="Rating" />
  <br />
  <hr />
  <input type="submit" class="btn btn-success float-right" value="Save
Comment" />
</form>
```

```

public function store(Request $request)
{
    if (Auth::check()==false) {
        // The user is not logged in...
        return redirect('login');
    }
    $fullcomment=$request->input('commentBody');
    $rating=$request->input('Rating');
    $product_id=$request->input('ProductId');
    //dd($fullcomment);
    $newcomment=new Comment();
    $newcomment->product_id=$product_id;
    $newcomment->FullComment=$fullcomment;
    $newcomment->CommentRating=$rating;
    $newcomment->createDate=date("Y-m-d H:i:s");
    $newcomment->save();
    $redirect_url='/products/'.$product_id;
    return redirect($redirect_url);
}

```

6. Автентикација

За потребите на оваа апликација, како што е онлајн наплата, користење на администрациски панел и оставање на коментари, имплементирана е автентикација на корисници.

Со артисан командата **php artisan ui vue --auth** се инсталираат сите потребни контролери и views за логирање, одлогирање, регистрирање на корисник и промена на лозинка

Во RouteServiceProvider може да дефинираме каде корисникот би бил редиректиран по успешно логирање или создавање на нов кориснички профил. Во оваа апликација, редиректирањето се врши на /categories .

Со помош на Auth класата, може да се извршуваат различни операции, како проверка дали даден корисник е логиран, и информација за моментално логираниот корисник.

```

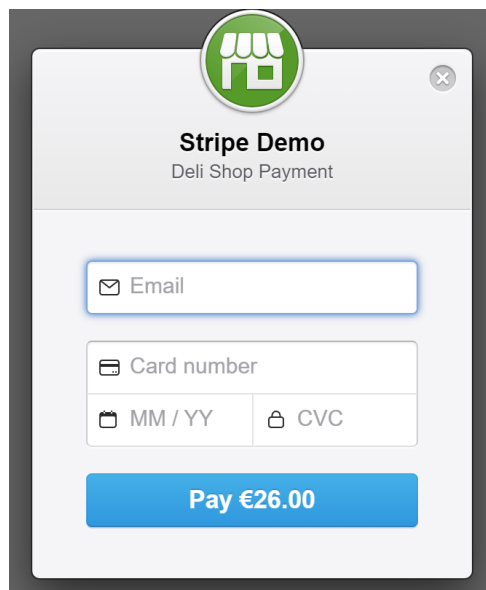
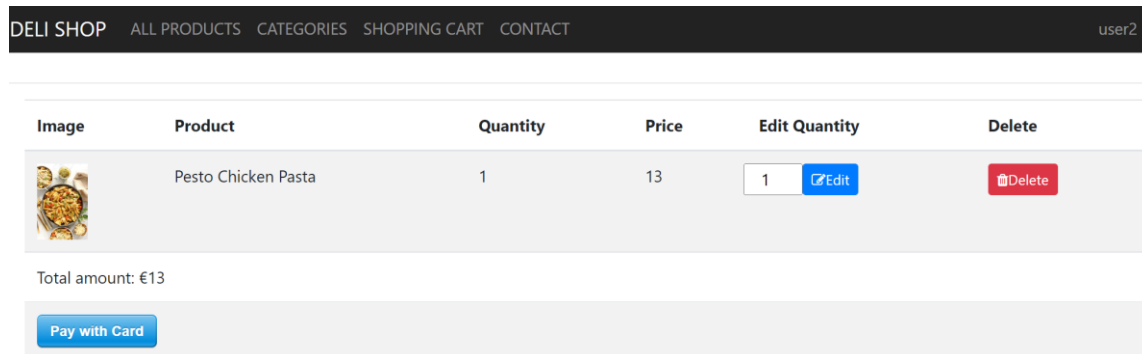
if (Auth::check()==false) {
    // The user is not logged in...
    return redirect('login');
}

```

7. Онлајн плаќање

За оваа функционалност се создадени се претходно споменатите модели Cart, CartItem и User, како и релациите помеѓу нив, нивните табели, контролери, рути соодветните views. Создаден е и нов Checkout контролер.

Онлајн наплаќањето со помош на кредитна картичка е овозможено со Stripe. Овој пакет нуди интерфејс за наплата, како и безбедно тестирање на наплати со псевдо информации. На страната каде се прикажува копче за наплата кое отвара модален прозорец за внес на информации од кредитна картичка. Ова копче се појавува само доколку има барем еден елемент поставен во кошничката.



Откако ќе се наплати дадена кошничка, првично се поминуваат сите елементи во неа и се намалува достапната количина на продуктите во база. Доколку продуктот целосно се продаде, се поминуваат сите постоечки кошнички и количината на продуктот во нив се става на нула. Оваа постапка се извршува со цел да се избегне наплата на продукт кој веќе не се продава.

На секоја продадена кошничка, се креира запис во табелата orders при што имаме увид на секоја продажна, купувачот, вкупната сума и продуктите кои биле купени.

Laravel NOVA Администрациски портал

Laravel NOVA е официјален пакет од Laravel за создавање на администрациски портали. Истиот дава можност за следниве функционалности:

Менаџмент на ресурси

Laravel NOVA имплементира комплетно нов CRUD интерфејс за веќе дефинираните Eloquent модели во базата на податоци. Притоа се поддржани сите типови на релации помеѓу моделите. Ваквите CRUD интерфејси во Laravel NOVA се нарекуваат ресурси.

Акции

Акции се групни PHP инструкции кои може истовремено да се извршуваат врз повеќе записи од базата на податоци. Ни овозможуваат брза и лесна промена на голем број податоци. Акциите однапред се дефинираат и се активираат од страна на администраторот.

Филтри

Филтрите се кориснички дефинирани процедури кои овозможуваат лесна промена на погледите на одредени ресурси според некои критериуми. Овозможуваат прикажување на записи од еден ресурс кои исполнуваат однапред дефинирани услови.

Лупи (Lenses)

Овозможуваат покомплексни филтрирачки операции кои не е можно да се направат со помош на филтри. Лупите даваат комплетна контрола врз Eloquent барањето кое се испраќа до базата на податоци.

Custom алатки

Laravel NOVA овозможува развој на комплетно нови алатки за примена во администрацискиот портал. Овие алатки се користат за изработка на компоненти кои поинаку не би можеле да се направат со помош на постоечките NOVA компоненти. Секоја алатка всушност претставува нова Vue компонента при што комплетната контрола врз функционалноста и изгледот на алатката се во рацете на програмерот.

Метрики

Метриките се графички прикази преку кои може да се агрегираат одредени показатели кои произлегуваат од податоците во базата. Ваквите показатели може да имаат оперативна или бизнис перспектива и истите може да се прикажат на повеќе типови графички прикази.

Авторизација

Laravel NOVA е комплетно интегрирана со системот за авторизација кој е применет при изработка на самата Laravel апликација и ги користи корисниците кои се веќе дефинирани. Постои можност за ограничување на пристап на одреден дел од CRUD операциите во зависност од овластувањата кои ги има корисникот.

Custom полиња

Покрај големиот број предефинирани полиња за приказ на одреден ресурс, NOVA овозможува креирање на поле за приказ по желба на корисникот. Се користи за приказ на комплексни податоци за кои не може да се применат веќе постоечките NOVA полиња.

Пребарување

Во рамки на администрацискиот портал, дадена е можност за локално пребарување низ одреден ресурс, како и глобално пребарување низ повеќе ресурси. За покомплексни пребарувања, можна е интеграција со Laravel Scout.

Нотификации

Администраторот на порталот може да биде известен за одредени настани од значај. Пример за значајни настани се нова продажба, ниско ниво на расположливост на одреден продукт во продавницата, итн.

Релации

NOVA ги поддржува веќе постоечките Eloquent релации во базата на податоци. При креирање и промена на запис постои можност за директен внес на податоците кои се однесуваат на некој од моделите кои се во релација со моделот кој се менува.

Кондиционални полиња

При приказ на полињата на еден ресурс овозможено е селективно прикажување на само одредени полиња во зависност од вредноста на други полиња.

1. Инсталација на NOVA

За оваа апликација, искористена е NOVA ver 3.27.0, која е компатибилна со Laravel ver.8. Се до верзија NOVA 3.x можни се два типа на инсталација на пакетот во рамки на Laravel проектот.

- Инсталација преку zip датотека
- Инсталација преку онлајн репозиториум на <https://nova.laravel.com>

Почнувајќи од верзија 4.x, можна е исклучиво само онлајн инсталација.

За пристап до NOVA пакетот, потребно е да се изврши регистрација на Laravel NOVA порталот и да се набави лиценца. Постојат два типа на лиценци:

- Single – за поединечен проект
- Unlimited – неограничен број на проекти

Со валидна лиценца, симнуваме zip датотека од Releases секцијата на Laravel NOVA страната, истата се распакува и копира во /nova фолдер во Laravel проектот.

За инсталација преку Composer, ги додаваме следниве секции во composer.json.

```
"repositories": [
    {
        "type": "path",
        "url": "../nova"
    }
],
```

Во require делот се додава следнава линија "laravel/nova": "*"

Доколку сакаме да извршиме онлајн инсталација:

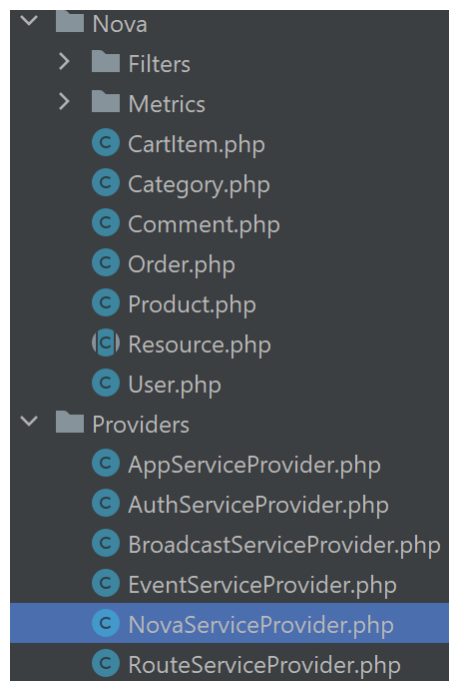
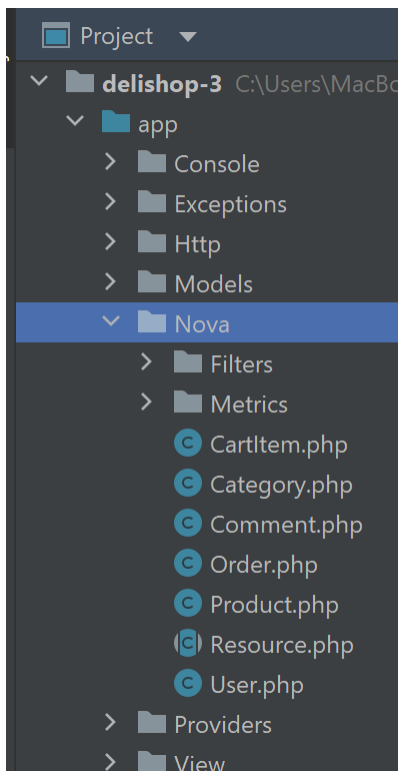
```
"repositories": [
    {
        "type": "path",
        "url": "https://nova.laravel.com"
    }
],
```

Следат командите за инсталација:

- composer update
- php artisan nova:install
- php artisan migrate

Доколку сите чекори се превземени правилно, би требало да се појави

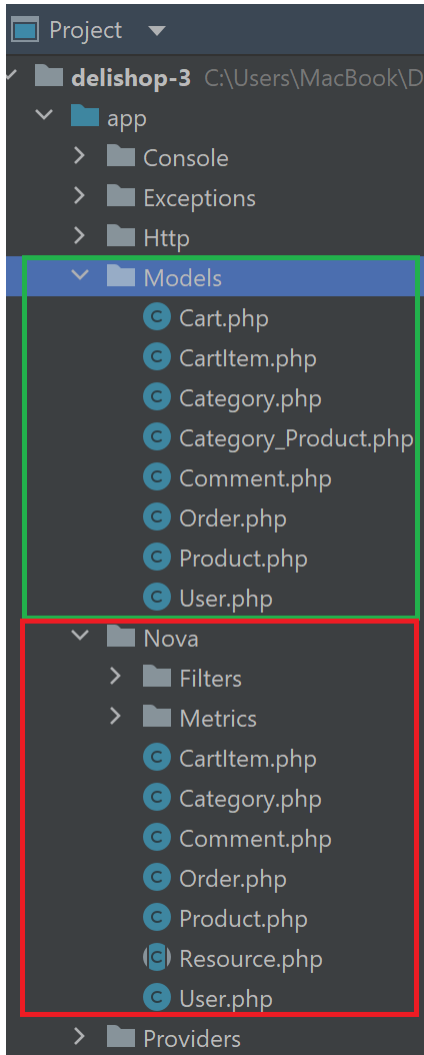
App\Providers\NovaServiceProvider.php и App\Nova датотека во која се сместуваат сите ресурси кои ќе се прикажуваат на администрацискиот панел.



2. Додавање на ресурси(Модели)

Introduction

Моделите сместени во \Nova датотеката се однесуваат на ресурсите од администрацискиот портал, додека моделите во App\Models се Eloquent моделите кои се однесуваат на корисничката апликација.

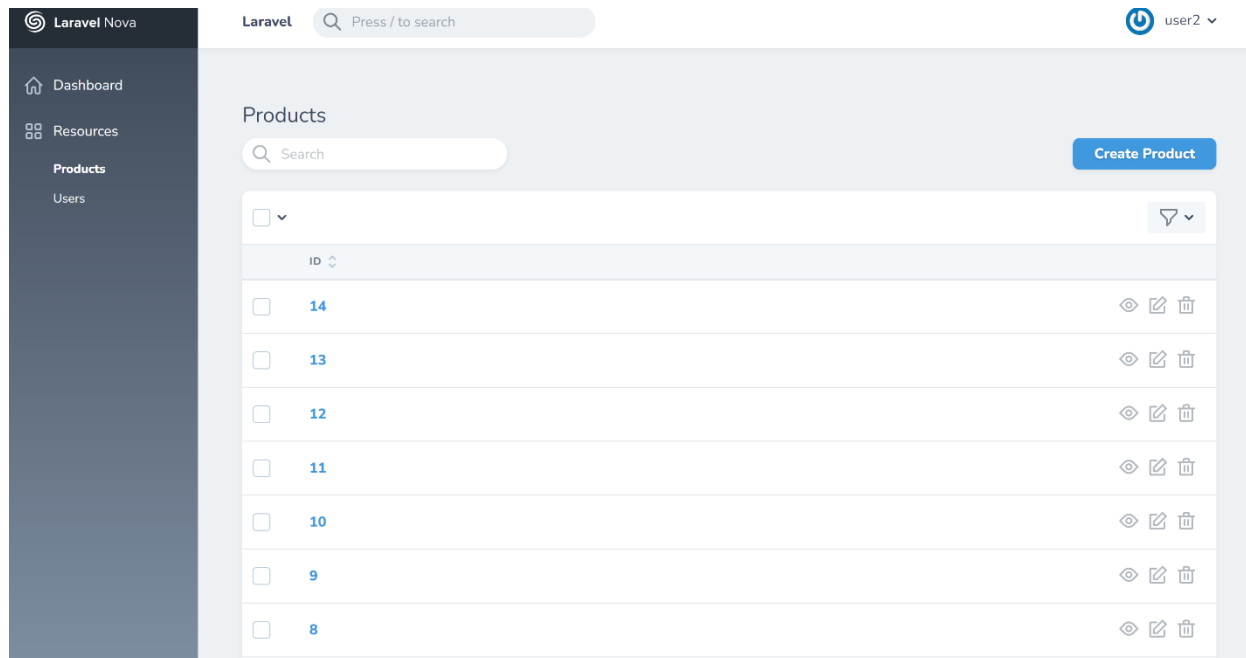


Креираните ресурси се прикажуваат во левиот мени панел од Nova порталот во секцијата Resources.

3. Креирање на ресурс (Модел Product)

Со командата `php artisan nova:resource Product` се создава нов ресурс, кој ќе ги прикажува информациите од базата на податоци во административниот панел. Сите модификации врз него се вршат во `\app\Nova\Product.php` датотеката. Сите CRUD операции потребни за модифицирање на ресурсите се автоматски креирани од NOVA.

Во првиот приказ на еден ресурс, најчесто е прикажано само ID полето од базата. За прикажување на дополнителни полиња од Product моделот, потребна е модификација на fields() методот.



4. Додавање на полиња/Fields на даден ресурс

Даден ресурс и неговите особини се прикажуваат преку полиња/fields. Приказот кои точно информации ќе се прикажат се дефинира во fields() функцијата во самата класа на ресурсот.

```
public function fields(Request $request)
{
    return [
        ID::make('__('ID'), 'id')->sortable(),
        ExternalImage::make('Product Image', 'ProductImageURL')->width(32)-
        >height(32)->rules('required'),
        Text::make('Product Name', 'ProductName')->sortable()-
        >rules('required'),
        Number::make('Product Price', 'ProductPrice')->sortable()-
        >onlyOnForms()->rules('required'),
        Text::make('Product Price', 'ProductPrice')->resolveUsing(function
        ($ProductPrice){
            return '€'.$ProductPrice;
        })->sortable()->exceptOnForms(),
        Number::make('Product Stock', 'ProductStock')->sortable()-
        >rules('required'),

        Textarea::make('Product Description', 'ProductDescription')-
        >hideFromIndex()->rules('required'),
        Textarea::make('Product Ingredients', 'ProductIngredients')-
        >hideFromIndex()->rules('required'),
```



```
} ;  
}
```

Основната синтакса за додавање на поле е следнава:

[Тип на променлива]::make('името на полето','името на полето во базата на податоци')

Постои широк обем на можни типови кои може да се прикажат во полето. Во оваа веб апликација се искористени следниве:

- Text – за текстуални полиња
- Textarea – за текстуални полиња со поголем формат
- Number – за нумерички полиња
- Password – за текстуално поле за лозинка
- ExternalImage – за приказ на слика од URL адреса
 - Ова е овозможено со помош на chaseconey/nova-external-image пакетот.

По конфигурацијата на полињата, се добива следниов изглед на Product ресурсот.

<input type="checkbox"/>	1		Broccoli Cheese Soup	12	20	  
<input type="checkbox"/>	2		Chicken Noodle Soup	12	20	  
<input type="checkbox"/>	4		Tomato Soup	10	20	  
<input type="checkbox"/>	5		Spicy Pumpkin Soup	12	20	  
<input type="checkbox"/>	6		Pesto Chicken Pasta	13	20	  
<input type="checkbox"/>	7		Sweet Potato Spinach Lasagna	12	20	  
<input type="checkbox"/>	8		Caramel Apple Cinnamon Waffles	13	20	  
<input type="checkbox"/>	9		Carrot Cake Pancakes	13	20	  
<input type="checkbox"/>	10		Chocolate Chia Seed Pudding	10	20	  

На прикажаниот изглед со помош на иконите на крајот на секој ред, се повикуваат погледите за различните CRUD методи за секој запис во базата (view, edit/update, delete). Погледот за Create се повикува преку копчето Create Product.

Со помош на посебни методи се специфицира во кои точно погледи/views т.е за кои CRUD операции се видливи дадени полиња. За оваа апликација се искористени следниве:

- onlyOnForms() – полиња кои се појавуваат само во форми за внес.
- exceptOnForms() – полиња кои не се појавуваат на форми за внес.
- hideFromIndex() – полиња кои не се прикажуваат во листањето на ресурсот

Во дадениот пример за Product update, полињата Product Description и Product Ingredients се видливи откако ќе се повика edit/update погледот за одреден продукт.

Dashboard
Resources
Categories
Products
Users

Update Product: Zucchini Bread

Product Image

Product Name

Product Price

Product Stock

Product Description

Make yourself a loaf of healthy zucchini bread made with 100% whole grains and no butter! This zucchini bread is perfect for an easy breakfast or an easy snack in the afternoon!




Product Ingredients

white whole wheat flour, coconut sugar, salt, zucchini, eggs, maple syrup, unsweetened almond milk, banana, coconut oil

По потреба, може да се дефинира форматирање на приказот на одредено поле според желбите на корисникот. На пример, полето за цена на продукт е прилагодено со прикажување на симболот за валута, и истото се користи само за прикажување на погледи.

```
Text::make('Product Price', 'ProductPrice')->resolveUsing(function
($ProductPrice){
    return '€'.$ProductPrice;
})->sortable()->exceptOnForms(),
```

☐
☐

ID	PRODUCT IMAGE	PRODUCT NAME	PRODUCT PRICE	PRODUCT STOCK	
<input type="checkbox"/> 14		Zucchini Bread	€12	20	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/> 13		Chocolate Chocolate Chip Banana Bread	€12	20	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/> 12		Strawberry Banana Bread	€13	19	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

За да се овозможи сортирање на табелата според одредено поле/колона, се додава методот `sortable()` во дефиницијата на полето. Со тоа, покрај името на полето во табелата, се прикажуваат две стрелки, кои овозможуваат сортирање по опаѓачки или растечки редослед.

Можна е валидација при внес на некои од полињата на ресурсот: За ова се користи `rules()` методот кој се додава во дефиницијата на методот и ги следи истите правила за валидација како Laravel. Пример за неколку од нив се:

- `rules('required')` – полето е задолжително и мора да се пополни
- `rules('required', 'max:255')` – полето смее да прими најмногу до 255 карактери
- `'alpha_num'` – во полето дозволени се само алфанумерички карактери
- `'email'` – внесот во полето мора да е во формат на емаил адреса
- `'password'` – внесот во полето мора да одговара на лозинката на корисникот
- `'numeric'` – во полето дозволени се само нумерички карактери
- `'nullable'` – се дозволува во полето да нема влез

Полињата кои се задолжителни во формите за внес, се означени со црвена ѕвездичка покрај името на полето.

5. Релации

Покрај главните полиња на секој ресурс кои ги опишавме, Laravel NOVA овозможува прикажување на податоците кои произлегуваат од релациите помеѓу моделите на ресурсите. Откако ќе се вклучат релациите како дел од полињата кои треба да се прикажуваат за еден ресурс, во погледот за детален приказ на еден запис автоматски се прикажуваат и записите од други ресурси со кои тој е во одредена релација.

Пр. избран продукт и категории на кои припаѓа.

Product Details: Zucchini Bread

ID	14
Product Image	
Product Name	Zucchini Bread
Product Price	€12
Product Stock	20
Product Description	Show Content
Product Ingredients	Show Content

categories

Search

Attach Category

ID	CATEGORY IMAGE	CATEGORY NAME	
4		Breads	<div></div> <div></div> <div></div>
2		Main Courses	<div></div> <div></div> <div></div>

Previous

1-2 of 2

Next

Laravel NOVA ги поддржува сите типови на релации на начин како што се дефинирани преку Eloquent релациониот модел во базата. Тоа се:

- One-to-One
- One-to-Many
- Many-to-Many

За вклучување на податоците од релацијата на еден ресурс, се користат истите типови на методи кои ги користиме за опис на релациите во Eloquent моделот. Соодветно се користат:

- One-to-One
 - hasOne/belongsTo
- One-to-Many
 - hasMany/belongsTo
- Many-to-Many
 - belongsToMany/belongsToMany

Релациите се вклучуваат во ресурсите како релациски типови на полиња во fields() методот во Nova моделот.

Bo Product:

```
BelongsToMany::make('categories'),  
HasMany::make('comments'),
```

Bo Comment:

```
BelongsTo::make('product')->sortable(),
```

Bo Category:

```
BelongsToMany::make('products')
```


Соодветно, при избор на категорија, исто така можеме да ги прикажеме продуктите кои припаѓаат на таа категорија

Category Details: Breads

ID

4

Category Image



Category Name

Breads

products

Search

Attach Product

ID	PRODUCT IMAGE	PRODUCT NAME	PRODUCT PRICE	PRODUCT STOCK	
<input type="checkbox"/> 14		Zucchini Bread	€12	20	<div> <div></div> <div></div> <div></div> </div>
<input type="checkbox"/> 13		Chocolate Chocolate Chip Banana Bread	€12	20	<div> <div></div> <div></div> <div></div> </div>
<input type="checkbox"/> 12		Strawberry Banana Bread	€13	19	<div> <div></div> <div></div> <div></div> </div>
<input type="checkbox"/> 11		Cornbread	€10	19	<div> <div></div> <div></div> <div></div> </div>

Previous

1-4 of 4

Next

Како посебен случај, ќе спомнеме релација помеѓу три табели. Laravel NOVA овозможува еден ресурс да прикаже податоци од третата табела преку релацијата која ја има со втората табела. Во конкретниот случај имаме

- Order (Many-To-Many) Carts (Many-To-Many) CartItems

Со користење на **HasManyThrough::make('cartitems')**, во Order ресурсот ги прикажуваме сите продукти кои биле дел од Cart кој бил купен.

Order Details: 3

ID

3

user

user1

Amount

34

Currency

eur

cartitems

Search

Create Cart Item

ID	PRODUCT	QUANTITY	
<input type="checkbox"/> 4	Chocolate Chia Seed Pudding	1	<div> <div></div> <div></div> <div></div> </div>
<input type="checkbox"/> 3	Sweet Potato Spinach Lasagna	2	<div> <div></div> <div></div> <div></div> </div>

Previous

1-2 of 2

Next

6. Пребарување

Laravel NOVA овозможува два типа на пребарувања

- Глобално пребарување – пребарува низ повеќе ресурси истовремено
- Локално пребарување -- пребарување на ниво на прикажан ресурс.

Локалното пребарување во администрацискиот панел дава можност за пребарување според дадени полиња на еден ресурс. Во секој ресурс може да дефинираме според кои полиња од табелата може да се извршува пребарувањето. Ова се дефинира во `search()` методот во моделот.

```
public static $search = [  
    'id', 'ProductName'  
];
```

На следнава слика е прикажано локално пребарување низ ресурсот за продукти според името на продуктот и пребарување на клучниот збор 'soup'.

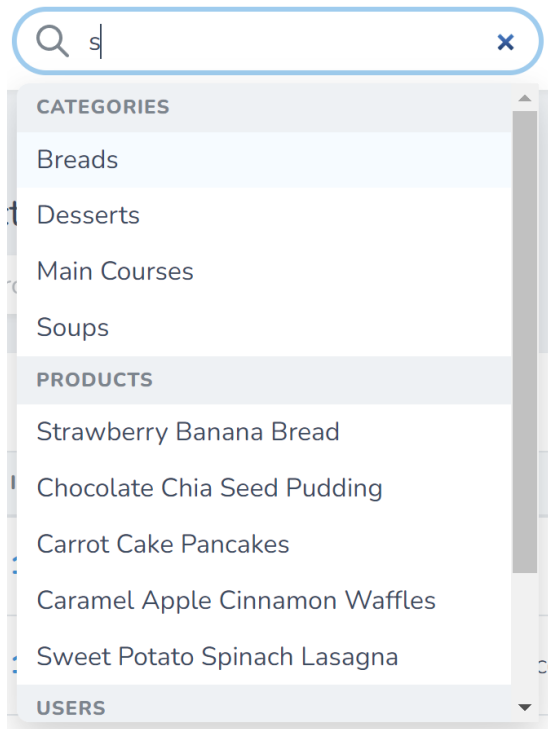
The screenshot shows the 'Products' resource in Laravel Nova. A search bar at the top contains the text 'soup'. Below the search bar is a table with 5 products. The table has columns for ID, Product Image, Product Name, Product Price, and Product Stock. The products listed are: Spicy Pumpkin Soup (ID 5), Tomato Soup (ID 4), Chicken Noodle Soup (ID 3), Chicken Noodle Soup (ID 2), and Broccoli Cheese Soup (ID 1). Each row has a checkbox on the left and three action icons (eye, edit, delete) on the right. At the bottom of the table, there are 'Previous' and 'Next' buttons, and a pagination indicator '1-5 of 5'.

ID	PRODUCT IMAGE	PRODUCT NAME	PRODUCT PRICE	PRODUCT STOCK
5		Spicy Pumpkin Soup	12	20
4		Tomato Soup	10	20
3		Chicken Noodle Soup	12	20
2		Chicken Noodle Soup	12	20
1		Broccoli Cheese Soup	12	20

Глобалното пребарување овозможува пребарување по клучен збор низ сите ресурси. Во секој ресурс треба да се специфицира кој негов атрибут ќе се прикаже во рамките на глобалното пребарување. Ова се дефинира преку статичката променлива `$title` во моделот.

```
public static $title = 'CategoryName';
```

На сликата е прикажано глобално пребарување низ сите ресурси со клучниот збор 's'.

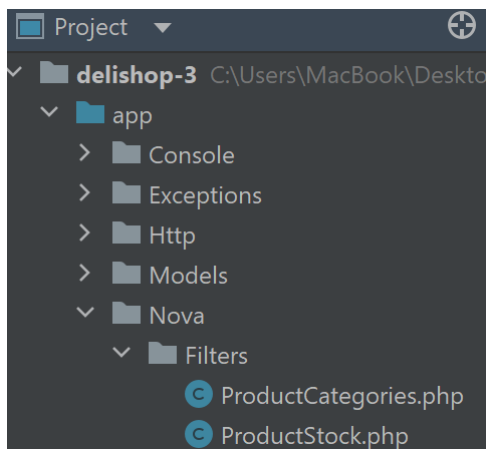


7. Филтрирање

Филтрирањето во Laravel NOVA ни дава можност листата на записи од еден ресурс да ја филтрираме според некој одреден критериум.

Филтрирањето на ресурсите се врши со помош на посебен тип на класа која се создава со **php artisan nova:filter [име на филтер]** командата. Во секој филтер може да се дефинира услов според кој ќе се прикажуваат одредени инстанци од ресурсот.

По креирањето на филтер се генерира фолдерот `App\Nova\Filters` и во него се сместува конфигурациската датотека за одреден филтер.



За потребите на овој администрациски панел, создадени се два филтри: ProductStock и ProductCategories. Однесувањето на филтерот се дефинира со помош на две функции во класата на филтерот.

Во методот option() се дефинираат можните опции според кои се прикажат во филтер менито. Во филтерот ProductCategories, option() методот треба да ни врати низа од сите категории, кои ќе ни овозможат да ја филтрираме листата од продукти според било која категорија.

```
public function options(Request $request)
{
    // return [];
    $categories=Category::all();
    $category_names=[];
    foreach ($categories as $category){
        $category_names[$category->CategoryName]=$category->id;
    }
    return $category_names;
}
```

Методот apply() ја опишува функционалноста на филтерот. Во филтерот ProductCategories треба да ни овозможи да ги прикаже само продуктите кои припаѓаат на дадена категорија.

```
public function apply(Request $request, $query, $value)
{
    return $query->whereHas('categories', function($q) use ($value){
        $q->where('category_id', '=', $value);
    })->get();
}
```

На сличен начин филтерот ProductStock ни овозможува прикажување на листа на продукти според нивната расположливост. Неговите option() и apply() се следни:

```
public function options(Request $request)
{
    return [
        'Low Stock'=>$this->threshold,
        'In Stock'=>$this->threshold+1
    ];
}
public function apply(Request $request, $query, $value)
{
    if($value==$this->threshold)
    {
        return $query->where('ProductStock', '<=', $value);
    }
    else{
        return $query->where('ProductStock', '>=', $value);
    }
    return $query;
}
```


За да станат филтрите употребливи во администрацискиот панел, тие треба да се регистрираат во моделот за соодветниот ресурс, поточно во `filters()` методот.

```
public function filters(Request $request)
{
    return [
        new Filters\ProductStock,
        new Filters\ProductCategories
    ];
}
```

8. Метрики

Laravel NOVA овозможува брз преглед на индикаторите кои се клучни за бизнис и оперативните аспекти на веб апликацијата врз која работи. Како примери за бизнис индикатори се имплементирани следниве:

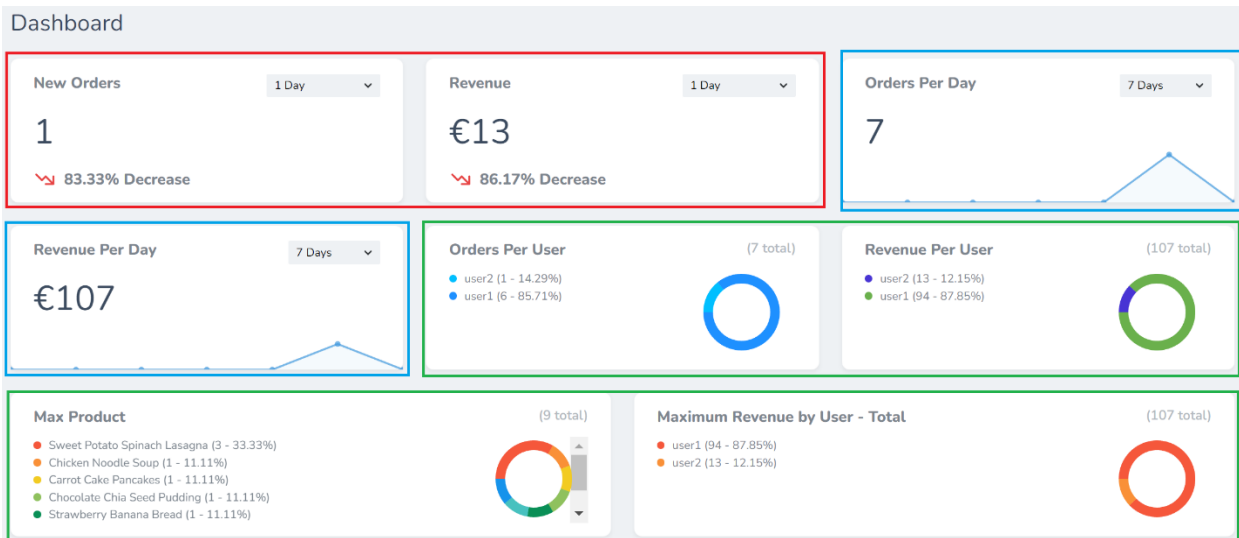
- Дневен промет
- Број на дневни нарачки
- Тренд на промет во даден период
- Тренд на нарачки во даден период
- Преглед на корисници според број на нарачки
- Преглед на корисници според промет
- Продукти според најголема продаваност

Метриките се посебен тип на класи во Laravel NOVA кои прикажуваат дадена пресметка/анализа на сите записи на даден ресурс.

Постојат повеќе типови на метрики:

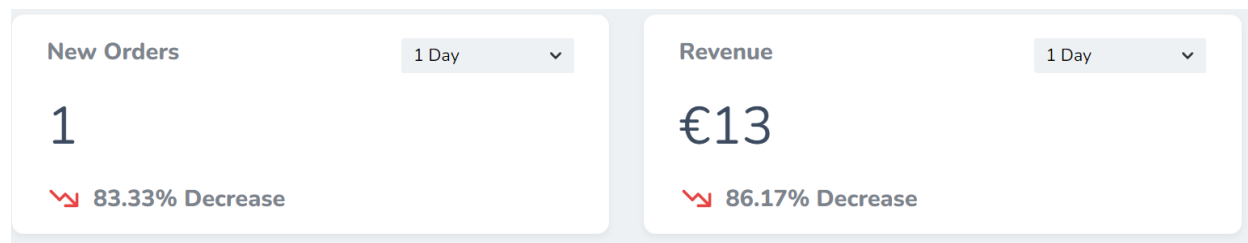
- Value Метрики
- Trend Метрики
- Partition Метрики

Во рамки во проект се креирани сите типови на метрики и истите се поставени на dashboard од администрацискиот панел. Изгледот на dashboard-от со означени типови на метрики следи подолу:

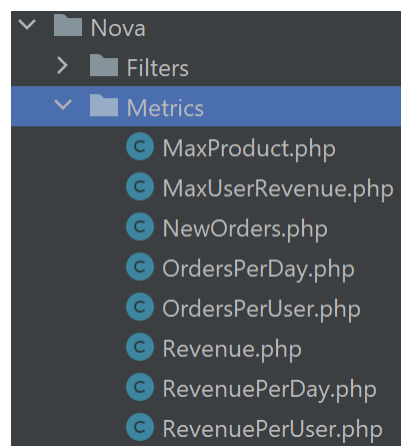


Value метрики

Value метриците прикажуваат единечна вредност на одреден параметар во даден интервал. Исто така, прикажува и компаративна анализа со претходен временски интервал. На пр. број на нови нарачки во денот и споредба со број на нарачки од претходен ден.



Value метриците се создаваат со командата **php artisan nova:value [име на метриката]**. Со оваа команда се создава нова App\Nova\Metrics папка доколку таа не постои претходно, каде се сместува конфигурациската датотека за метриката.



Функционалноста на метриката се дефинира во `calculate()` методата. Притоа се користат различни функции за агрегација:

- `count()`
- `max()`
- `min()`
- `average()`
- `sum()`

Дефинирање на временскиот период на метриката се прави во `ranges()` методот:

```
public function ranges()
{
    return [
        1 => ('1 Day'),
        7 => ('7 Days'),
        30 => ('30 Days'),
        60 => ('60 Days'),
        365 => ('365 Days'),
        'TODAY' => ('Today'),
        'MTD' => ('Month To Date'),
        'QTD' => ('Quarter To Date'),
        'YTD' => ('Year To Date'),
    ];
}
```

Во конкретниот случај за број на нарачки:

```
public function calculate(NovaRequest $request)
{
    return $this->count($request, \App\Models\Order::class);
}
```

Додека за дневен промет:

```
public function calculate(NovaRequest $request)
{
    return $this->sum($request, Order::class, 'amount')->currency('€');
}
```

Направено е форматирање на излезот со користење на `currency()` методата.

Прикажување сите типови на метрики е можно на два начина:

- Приказ на Dashboard
- Приказ во погледот на индивидуален ресурс

За приказ на Dashboard вршме регистрација на метриката во `cards()` методот во `NovaServiceProvider.php`, додека за конкретен ресурс во `cards()` во моделот на тој ресурс.

```
public function cards(Request $request)
{

```

```

return [
    new NewOrders,
    new Revenue,
    new OrdersPerDay,
    new RevenuePerDay,
    new OrdersPerUser,
    new RevenuePerUser
];
}

```

Trend Метрики

Trend метриците ја прикажуваат промената на вредноста на одреден параметар во избран временски интервал. Во конкретниот пример, користиме тренд метрика за приказ на вкупниот дневен промет споредено за секој ден од неделата. Временскиот интервал се избира по желба на корисникот (7 дена, 30 дена, 60 дена, 90 дена).



Trend метриците се создаваат со командата **php artisan nova:trend [име на метриката]**. Со оваа команда се создава нова App\Nova\Metrics папка доколку таа не постои претходно, каде се сместува конфигурациската датотека за метриката. Функционалноста на метриката се дефинира во calculate() методата. Функциите кои се користат за агрегација во calculate() се комбинација од value метриците (min, max, average, count, sum) како главни категории за следење на вредностите за време на одредени временски интервали (Months, Weeks, Days, Hours, Minutes).

- count()
 - countByMonths()
 - countByWeeks()
 - countByDays()
 - countByHours()
 - countByMinutes()
- max()
 - maxByMonths()
 - maxByWeeks()
 - maxByDays()
 - maxByHours()
 - maxByMinutes()
- min()
 - minByMonths()

- minByWeeks()
 - minByDays()
 - minByHours()
 - minByMinutes()
- average()
 - averageByMonths()
 - averageByWeeks()
 - averageByDays()
 - averageByHours()
 - averageByMinutes()
- sum()
 - sumByMonths()
 - sumByWeeks()
 - sumByDays()
 - sumByHours()
 - sumByMinutes()

Во рамки на проектот се креирани две тренд метрики: RevenuePerDay и OrdersPerDay за следење на прометот споредбено по денови и следење на бројот на нарачки споредбено по денови. Calculate() методите за овие метрики се соодветно прикажани:

```
public function calculate(NovaRequest $request)
{
    return $this->countByDays($request, Order::class)->showSumValue();
}
```

```
public function calculate(NovaRequest $request)
{
    return $this->sumByDays($request, Order::class, 'amount')->prefix('€')->showSumValue();
}
```

Стандардно, trend метриките ја прикажуваат само последната вредност на параметарот. За да се прикаже збирот на сите вредности се користи методот showSumValue(). Во метриката за вкупен промет, дополнително користиме prefix('€') за приказ на валута.

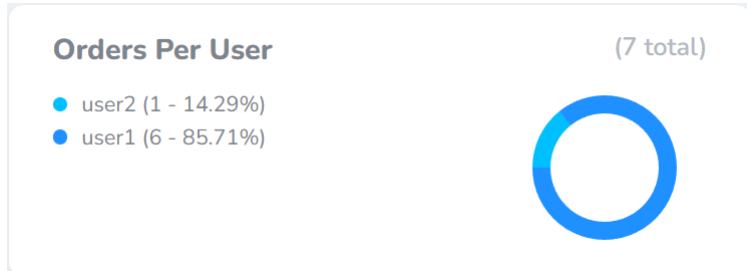
Како и кај value метриките, опциите за избор на временскиот интервал се дефинирани во ranges() методот.

```
public function ranges()
{
    return [
        7 => __ ('7 Days'),
        30 => __ ('30 Days'),
        60 => __ ('60 Days'),
        90 => __ ('90 Days'),
    ];
}
```

Регистрацијата на методите се врши соодветно на dashboard и/или во индивидуалните погледи за ресурси како што е претходно објаснето.

Partition метрики

Partition метриците се состојат од pie chart тип на приказ на кој јасно се гледа распределбата на избраната вредност за метриката според друг избран параметар. Како на пример, преку partition метрика можеме да прикажеме број на нарачки според клиентот:



Value метриците се создаваат со командата **php artisan nova:partition [име на метриката]**. Со оваа команда се создава нова App\Nova\Metrics папка доколку таа не постои претходно, каде се сместува конфигурациската датотека за метриката.

Функционалноста на метриката се дефинира во calculate() методата. Притоа се користат различни функции за агрегација:

- count()
- max()
- min()
- average()
- sum()

Во рамки на проектот се креирани 4 тренд метрики: MaxProduct, MaxUserRevenue, RevenuePerUser и OrdersPerUser, со цел администраторот да има увид на:

- Продукти кои најчесто се продаваат
- Корисници кои најчесто прават нарачки
- Корисници кои направиле најголем промет

Calculate() методите за овие метрики се соодветно прикажани:

OrdersPerUser

```
public function calculate(NovaRequest $request)
{
    return $this->count($request, Order::class, 'user_id')->label(function
($value) {
        $user=User::find($value);
        return $user->name;
    })->colors([
        User::find(1)->name => '#1E90FF',
        User::find(2)->name => '#00BFFF',
    ]);
}
```

RevenuePerUser

```

public function calculate(NovaRequest $request)
{
    return $this->sum($request, Order::class, 'amount', 'user_id')->label(function ($value) {
        $user=User::find($value);
        return $user->name;
    })->colors([
        User::find(1)->name => '#6ab04c',
        User::find(2)->name => 'rgb(72,52,212)',
    ]);
}

```

MaxProduct

Во оваа partition метрика, calculate() методот не содржи едноставна агрегација на полиња од базата, туку резултатот на пресметката се прави преку мануелна селекција на записите кои треба да се прикажат и соодветно форматирање на низата која треба да има предефинирана конструкција од типот.

```

return $this->result([
    'Group 1' => 100,
    'Group 2' => 200,
    'Group 3' => 300,
]);

```

```

public function calculate(NovaRequest $request)
{
    $allorders=Order::all();
    $products_array=[];
    foreach ($allorders as $order)
    {
        $cartitems=$order->cartitems;
        foreach ($cartitems as $cartitem)
        {
            $product_name=Product::find($cartitem->product_id)->ProductName;
            if(array_key_exists($product_name,$products_array)){
                $products_array[$product_name]=$products_array[$product_name]+$cartitem->quantity;
            }
            else{
                $products_array[$product_name]=$cartitem->quantity;
            }
        }
    }
    arsort($products_array);
    return $this->result($products_array);
}

```

MaxUserRevenue

```

public function calculate(NovaRequest $request)
{
    $allorders=Order::all();
    $users_array=[];
    foreach ($allorders as $order)
    {
        $username=User::find($order->user_id)->name;
        if(array_key_exists($username,$users_array)){
            $users_array[$username]=$users_array[$username]+$order->amount;
        }
        else{
            $users_array[$username]=$order->amount;
        }
    }
    arsort($users_array);
    return $this->result($users_array);
}

```

9. Кустомизација

Како што видовме од приложените calculate() методи, можни се прилагодувања на приказот на partition метриката според потребите.

Промената на приказот за името на полето, наместо user_id да се прикаже username, се прави со помош на label() методата која повикува функција преку која се враќа саканиот приказ.

```

return $this->sum($request, Order::class, 'amount','user_id')->label(function
($value) {
    $user=User::find($value);
    return $user->name;
})

```

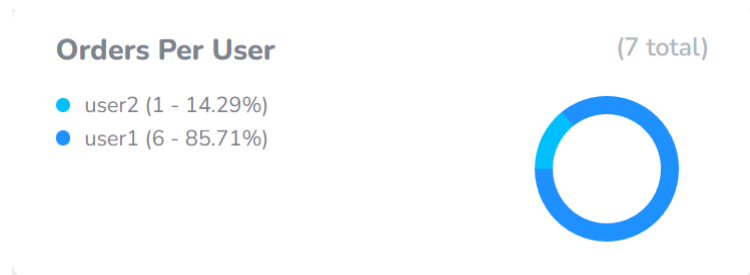
Исто така, во calculate() методата може да се дефинираат боите кои ќе се користат при конструкција на pie chart приказот. Истото се прави со colors() методата.

```

colors([
    User::find(1)->name => '#6ab04c',
    User::find(2)->name => 'rgb(72,52,212)',
]);

```


Прилагодениот изглед на метриката следи:



Сите метрики овозможуваат прикажување на порталот во различни ширини. Истото се прави при регистрација на метриката на соодветниот поглед преку додавање на методата `width()` и наведување на посакуваната ширина.

```
(new MaxProduct)->width('1/2'),  
(new MaxUserRevenue)->width('1/2'),
```

Секој наслов на метриките може да се дефинира по желба со промена на `name()` функцијата во моделот на метриката.

За `MaxUserRevenue`

```
public function name()  
{  
    return 'Maximum Revenue by User - Total';  
}
```

Крајниот изглед на оваа метрика со широчина $\frac{1}{2}$ половина и променет наслов е следен:



10. Заклучок

Со изработка на овој администрациски портал во Laravel NOVA за дадената веб продавница се овозможува комплетно менаџирање на содржините на веб продавницата од страна на администраторот. Тоа е овозможено при примена на CRUD погледите за секој од дефинираните ресурси во администрацискиот портал.

Пример на вакви активности кои администраторот може да ги извршува се:

- Додавање и одземање на количина на расположливи продукти во продавницата
- Увид на состојба и детекција на недостаток на артикли
- Промена на цени на поединечни артикли
- Додавање на нови артикли, нивен опис, состојки и цена.
- Додавање на нови категории, продукти
- Доделување на категорија на одреден продукт
- Следење на коментари од корисниците за секој продукт
- Следење на нарачки и продукти кои биле купени

Преку примена на метрики администраторот има увид и кон одредени бизнис показатели како:

- Следење на промет на дневно ниво
- Следење на тренд на прометот во тек на одреден временски период
- Увид на најактивни корисници по број на нарачки
- Увид на најактивни корисници по висина на промет
- Следење на продаваност и популарност на продуктите

11. Референци

- <https://laravel.com/docs/6.x/eloquent>
- https://www.digitalocean.com/community/tutorial_series/a-practical-introduction-to-laravel-eloquent-orm
- <https://laravel.com/docs/8.x/starter-kits>
- <https://dashboard.stripe.com/test/payments>
- <https://blog.quickadminpanel.com/stripe-payments-in-laravel-the-ultimate-guide/>
- <https://nova.laravel.com/docs/3.0/installation.html#installing-nova>