

Laravel

Урок 10. Добавление HTML-редактора. Хранение файлов. Очереди в Laravel

Изменение полученных агрегатором данных и создание пользовательских. Добавление интерфейса для редактирования html-страниц с возможностью добавления изображений. Очереди для выполнения параллельных и последовательных задач.

Оглавление

[Практика](#)

[Установка HTML-редактора](#)

[Очереди.](#)

[Сохранение файлов.](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Практика

Установка HTML-редактора

Чтобы получить информацию со стороннего сервиса или ввести ее на сайте, иногда требуется добавлять в нее html-разметку и изображения, менять шрифт или отступы, вводить нумерованные списки. Конечно, всегда можно внести разметку в обычное текстовое поле, а изображения загрузить, используя **ftp**, но это неудобно.

Чтобы пользователю было легко работать с вашим сервисом, можно установить html-редактор и специальный файловый менеджер. Он позволит загружать файлы на сервис и добавлять их в указанную часть редактируемого материала в виде ссылок.

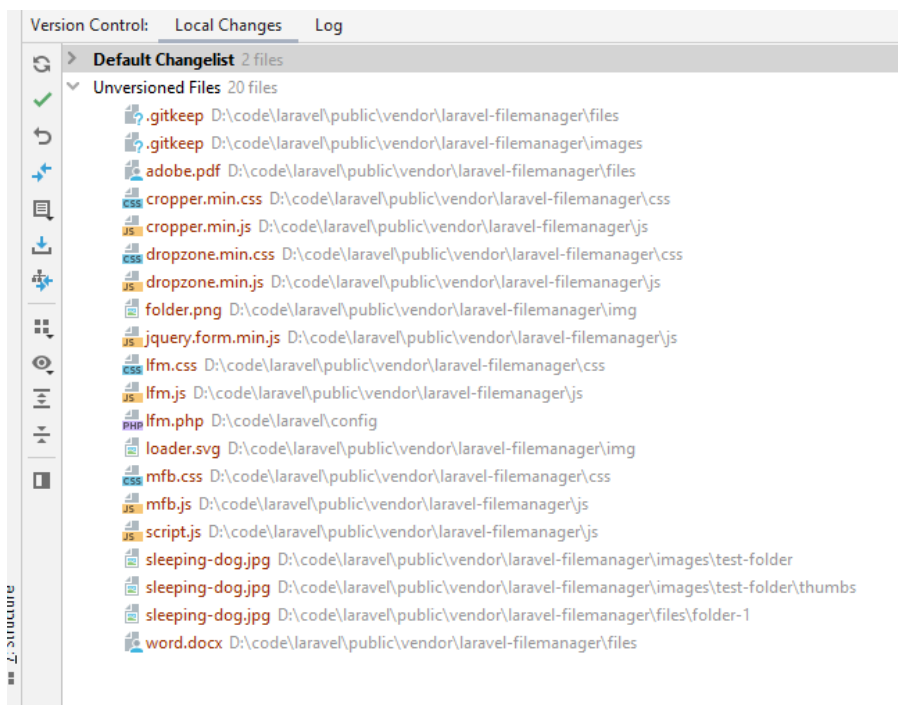
В качестве менеджера файлов рассмотрим пакет **laravel-filemanager**. Подробная документация по нему доступна по адресу <https://unisharp.github.io/laravel-filemanager/installation>.

Для начала установим его в наше приложение, выполнив команду **composer require unisharp/laravel-filemanager:~1.8**

Добавим необходимые зависимости в папку **vendor**. Затем надо перенести из нее некоторые данные в основное приложение. Для этого выполним следующие команды:

- **php artisan vendor:publish --tag=lfm_config**
- **php artisan vendor:publish --tag=lfm_public**

Обратим внимание на локальные изменения в проекте:



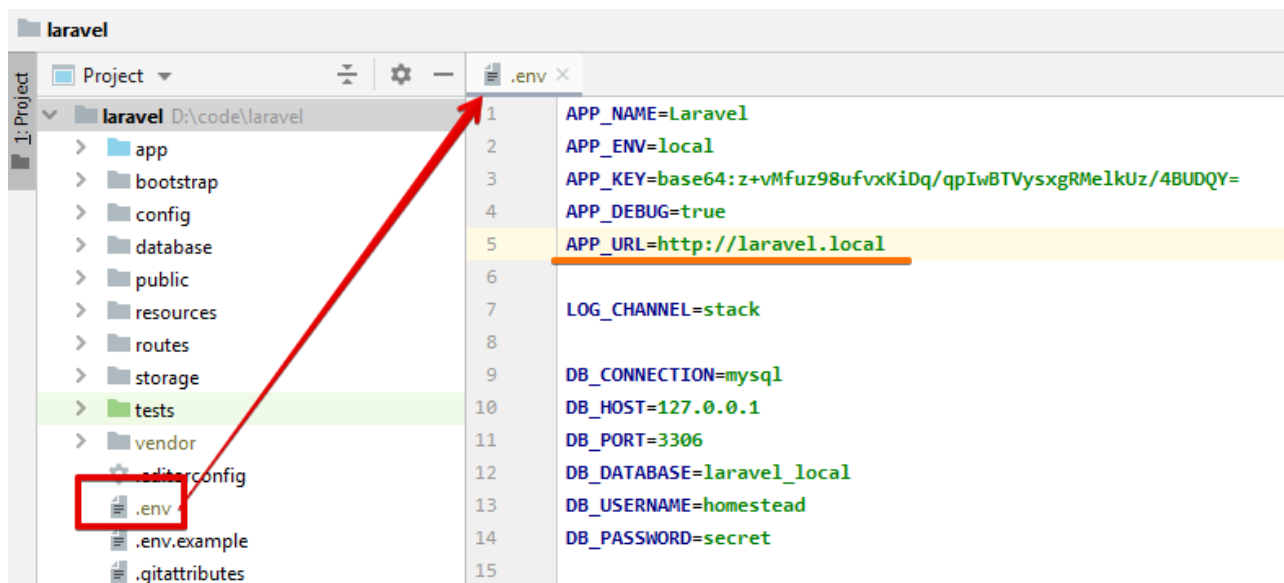
Теперь сбросим кеш нашего приложения, выполнив команды:

- **php artisan route:clear**

- `php artisan config:clear`

Также создадим симлинк из `public/storage` в `storage/app/public`, выполнив команду `php artisan storage:link`

Еще одно условие работы менеджера — правильное указание параметра `APP_URL`. В его значении должен стоять верный url вашего приложения.



Все приготовления выполнены. Теперь установим html-редактор `ckeditor` на страницу добавления и редактирования новостей. В качестве настроек укажем маршруты нашего файлового менеджера.

```

@laravel /resources/ views/ admin/ addNews.blade.php
Project
laravel D:\code\laravel
  app
  bootstrap
  config
  database
  public
  resources
  js
  lang
  sass
  views
    admin
      addNews.blade.php
      allNews.blade.php
      index.blade.php
      main.blade.php
      profile.blade.php
    auth
    layouts
      home.blade.php
      main.blade.php
      news.blade.php
      newsOne.blade.php
    routes
    storage
    tests
    vendor
      editorconfig
      .env
      .env.example
      .gitattributes
      .gitignore
      .styleci.yml
      artisan
      composer.json
      composer.lock
      package.json
      phpunit.xml
      readme.md
      server.php
      webpack.mix.js
  External Libraries
  Scratches and Consoles

@if($errors->has('title'))
  <div class="alert alert-danger">
    @foreach($errors->get('title') as $error)
      <p style="...">{{ $error }}</p>
    @endforeach
  </div>
@endif
<input class="form-control" name="title" placeholder="Название новости" value="{{ $news->title }}" > <br>
@if($errors->has('inform'))
  <div class="alert alert-danger">
    @foreach($errors->get('inform') as $error)
      <p style="...">{{ $error }}</p>
    @endforeach
  </div>
@endif
<textarea class="form-control" name="inform" placeholder="Информация" id="inform">{{ $news->inform }}</textarea> <br>

@if($errors->has('category_id'))...@endif
<select name="category_id" class="form-control" ...>
  <option value="1">Новость приватна?</option>
  <option value="2">...</option>
  <option value="3">...</option>
</select>
<button class="form-control" type="submit" ...>
</button>
</form>

<script src="//cdn.ckeditor.com/4.6.2/standard/ckeditor.js"></script>
<script>
  var options = {
    filebrowserImageBrowseUrl: '/laravel-filemanager?type=Images',
    filebrowserImageUploadUrl: '/laravel-filemanager/upload?type=Images&token=',
    filebrowserBrowseUrl: '/laravel-filemanager?type=Files',
    filebrowserUploadUrl: '/laravel-filemanager/upload?type=Files&token='
  };
</script>
<script>
  CKEDITOR.replace('inform', options);
</script>
</endsection>

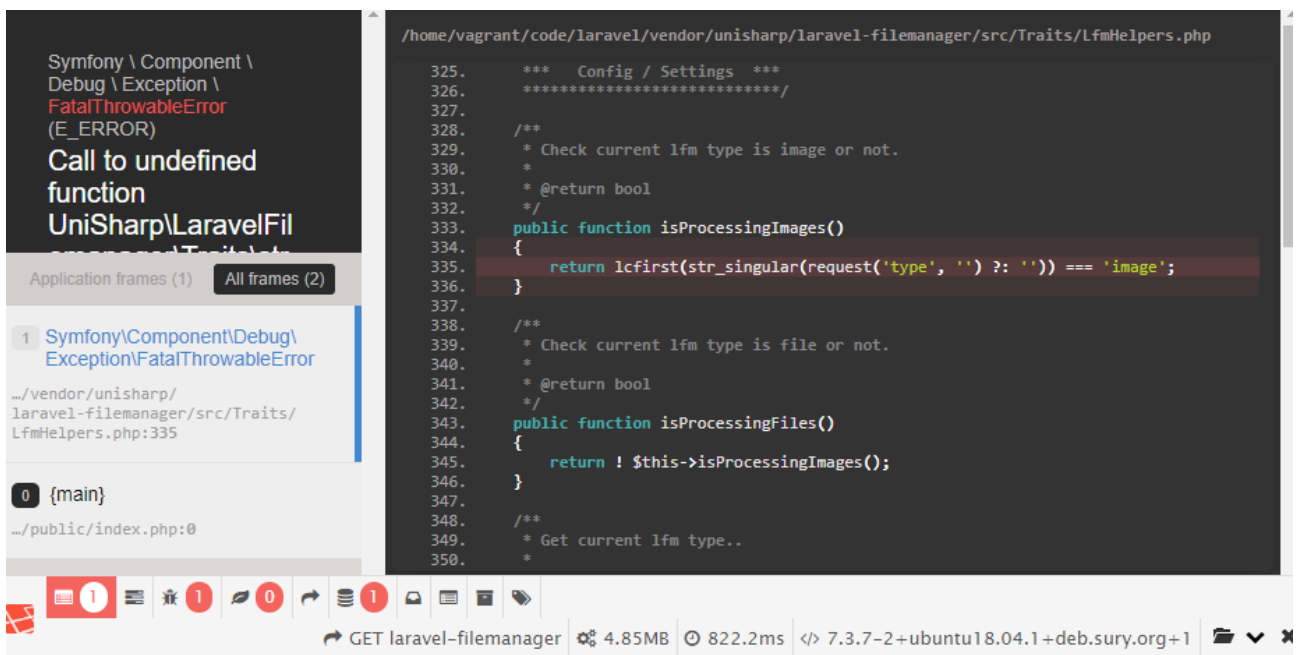
```

Это не самый удачный вариант, так как данные html-редактора будут тянуться из стороннего сервиса. «Портянка» из наборов параметров, которые мы передаем при вызове метода **replace**, тоже выглядит не очень.

Как быть, подскажет документация: <https://ckeditor.com/docs/ckeditor5/latest/index.html>.

А мы движемся дальше.

Перейдем в админ-панель и попробуем создать новую новость.

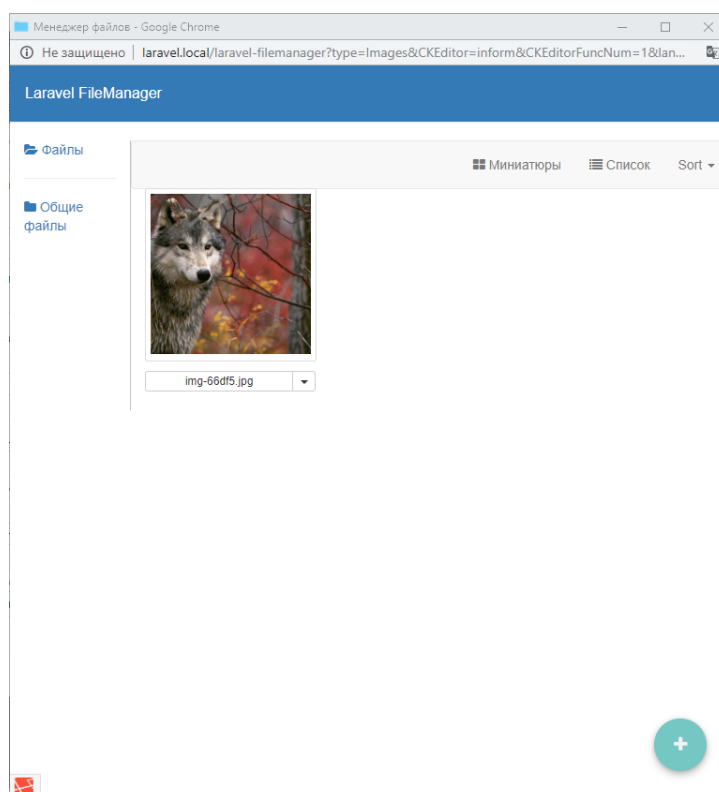


```
error Log
[18:50:31] LOG.error: Call to undefined function UniSharp\LaravelFilemanager\Traits\str_singular() {"userId":1,"exception":{}}
```

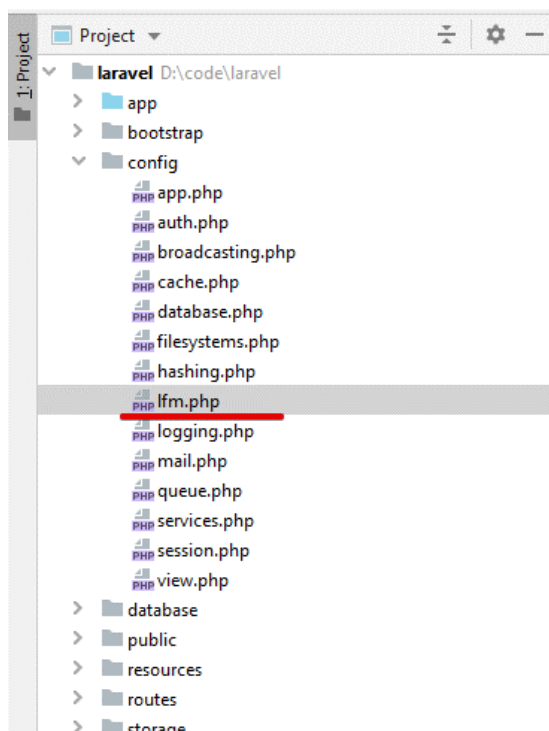
Из сообщения понятно, что чего-то не хватает. Добавим недостающий пакет командой **composer require laravel/helpers**

Затем нажмем еще раз на кнопку «Выбор на сервере».

Теперь менеджер файлов открылся.



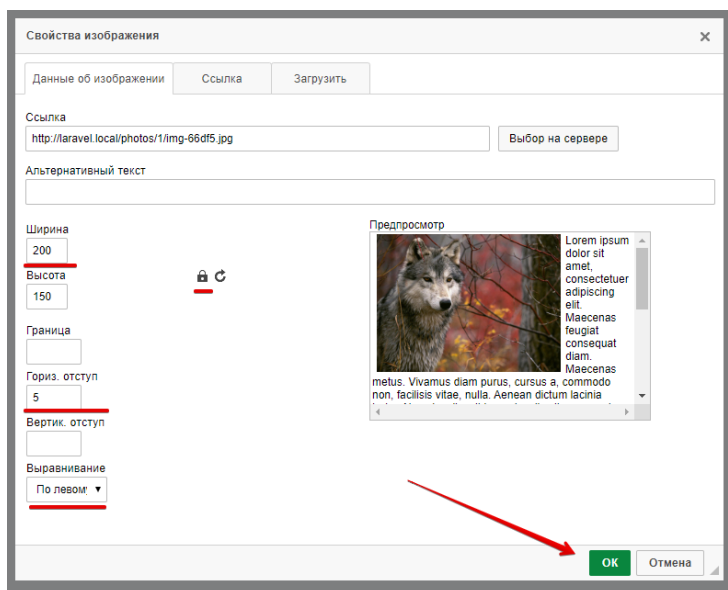
Настройки работы этого файлового менеджера формируются в файле **lfm.php**. Он появился у нас в проекте после выполнения команды публикации данных пакета. К нему вернемся позже.



Добавим какие-нибудь файлы в нашу систему, используя файловый менеджер. После двойного щелчка по загруженной картинке окно с менеджером закроется — и пользователь вновь увидит модальное окно «Свойства изображения». В нем уже появилась выбранная нами картинка.

Это окно позволяет настроить параметры отображения изображения: ширину, высоту, отступы и другое. Меняя значения параметров здесь, мы корректируем значения атрибутов в html-разметке.

Указав необходимые значения, жмем на ОК.



Далее в редакторе можно изменить текст, установить картинку в нужное место. Впишем название новости и укажем нужную категорию. Затем нажмем «Добавить».

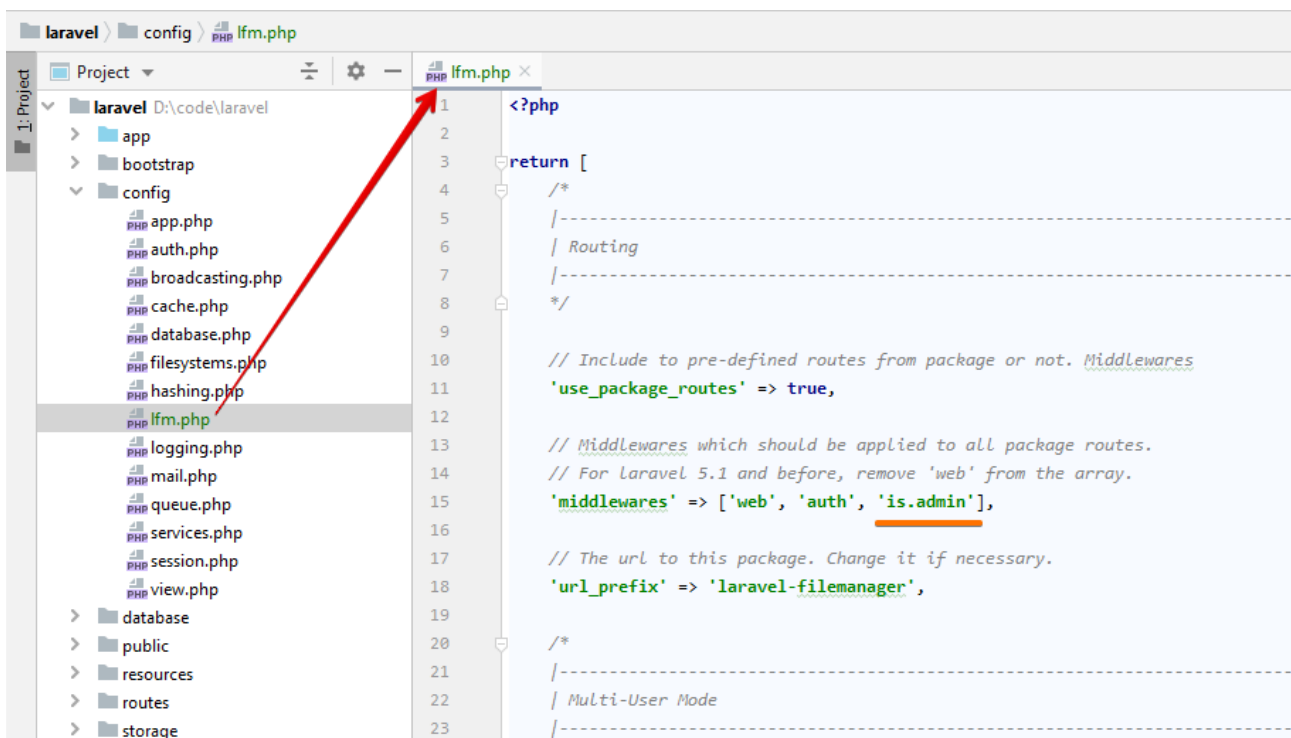

```
news.blade.php
1 @extends('main')
2
3 @section('title')
4     @parent Name
5 @endsection
6
7 @section('content')
8     <h1>Новости</h1>
9     @foreach($news as $item)
10        <h2>
11            {{ $item->title }} @if($item->is_private) <b>Private</b> @endif
12        </h2>
13        <div>{!! $item->inform !!</div>
14
15        @if (!$item->is_private)
16            <a href="{{ route('newsOne', $item->id) }}">
17                Подробнее...
18            </a>
19        @endif
20    <hr>
21    @empty
22        <p>Нет новостей</p>
23    @endforeach
24    {{ $news->links() }}
25 @endsection

newsOne.blade.php
1 @extends('main')
2
3 @section('title')
4     {{ $news->title }}
5 @endsection
6
7 @section('content')
8     <h1>{{ $news->title }}</h1>
9     <div>{!! $news->inform !!</div>
10    <hr>
11    <a href="{{ route('news') }}">К списку новостей</a>
12 @endsection
```

Есть еще одна проблема. Если вы как администратор, находясь в системе, скопируете ссылку, которая указывается в окне с файловым менеджером, а затем залогинитесь через социальную сеть — то обнаружите, что эта ссылка по-прежнему открывает доступ к менеджеру файлов. Это происходит потому, что для роутов менеджера не установлен посредник **is.admin**.

Как его установить?

В файле **lfm.php** хранятся настройки файлового менеджера. Откроем этот файл — и сразу поймем, куда и что надо добавить.



Очереди

Пользователь, открывая страницу сайта, получает информацию, ранее сохраненную для него из базы данных. Эта операция происходит достаточно быстро, и все остаются довольны.

Но данные могут очень быстро меняться — например, данные о свободных местах в самолете. Сохранить заранее и поддерживать их актуальность трудно: авиакомпаний, рейсов и самолетов много.

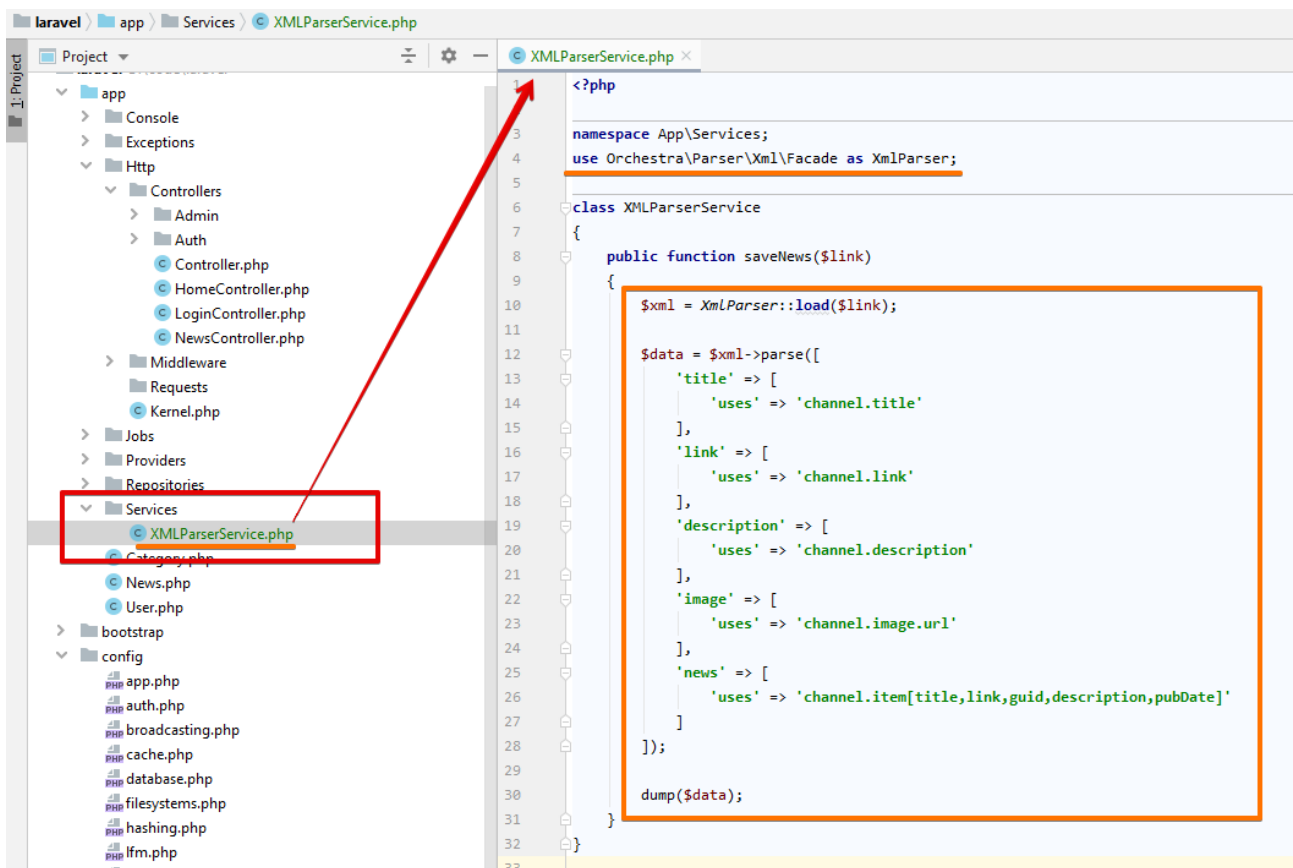
Такие данные обычно получают из сервисов — агрегаторов, которые тоже опрашивают другие ресурсы. Все это требует времени. Один такой запрос проходит относительно быстро. А если пользователь захочет получить информацию сразу из нескольких источников?

Один из комфортных для пользователя вариантов может проходить по такому алгоритму:

1. пользователь делает в браузере запрос на поиск билетов, указывая необходимые для него сервисы.
2. Запрос на сервере обрабатывается. В базу (в очередь) записывается информация о ресурсах, с которых надо получить ответ и, возможно, дополнительные параметры.
3. Пользователю выводится страница с информацией о том, что происходит с запросами к сервисам.
4. В это же время фоновые программы (воркеры) видят, что в очереди (в базе) появились новые задачи. Они начинают их разбирать, запуская парсинги с указанными параметрами.
5. По мере получения ответов из сторонних сервисов данные передаются пользователю.

На прошлом уроке мы создали парсер, собирающий информацию из «Яндекс.Новостей». Алгоритм парсера скрыт за фасадом **Orchestra\Parser\XmlFacade**, которому задан псевдоним **XmlParser**. Тем

не менее кода у нас в контроллере много. Вынесем его в отдельный метод сервиса **XMLParserService**.



```
<?php
namespace App\Services;
use Orchestra\Parser\Xml\Facade as XmlParser;

class XMLParserService
{
    public function saveNews($link)
    {
        $xml = XmlParser::load($link);

        $data = $xml->parse([
            'title' => [
                'uses' => 'channel.title'
            ],
            'link' => [
                'uses' => 'channel.link'
            ],
            'description' => [
                'uses' => 'channel.description'
            ],
            'image' => [
                'uses' => 'channel.image.url'
            ],
            'news' => [
                'uses' => 'channel.item[title,link,guid,description,pubDate]'
            ]
        ]);

        dump($data);
    }
}
```

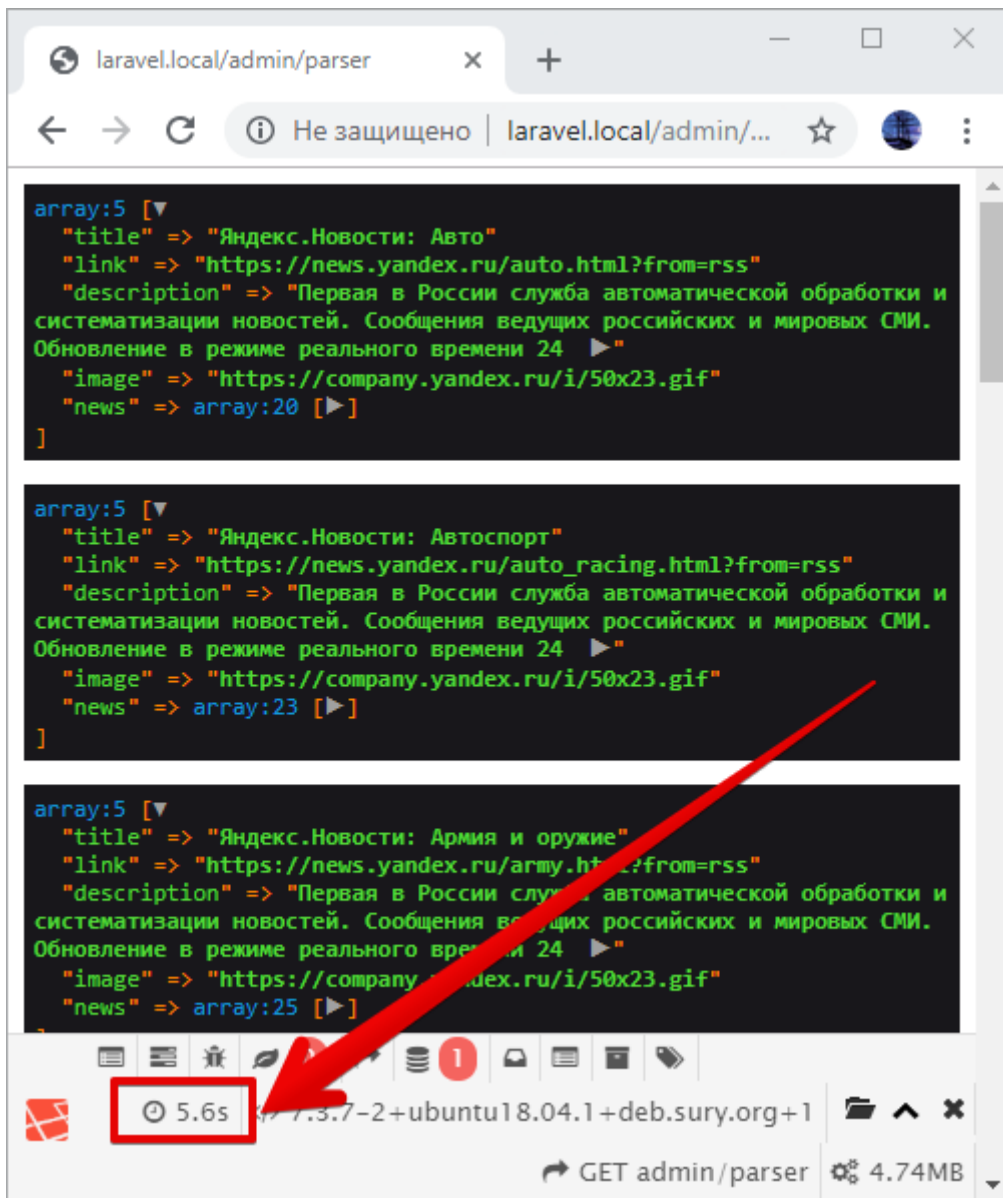
Укажем, что первым параметром в методе **index** класса **ParserController** ожидаем экземпляр данного сервиса — и Laravel сам будет его создавать. В контроллере добавим массив с ссылками на другие категории новостей и при помощи цикла получим данные.

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use App\Services\XMLParserService;
7
8 class ParserController extends Controller
9 {
10     public function index(XMLParserService $parserService)
11     {
12         $rssLinks = [
13             'https://news.yandex.ru/auto.rss',
14             'https://news.yandex.ru/auto_racing.rss',
15             'https://news.yandex.ru/army.rss',
16             'https://news.yandex.ru/gadgets.rss',
17             'https://news.yandex.ru/index.rss',
18             'https://news.yandex.ru/martial_arts.rss',
19             'https://news.yandex.ru/communal.rss',
20             'https://news.yandex.ru/health.rss',
21             'https://news.yandex.ru/games.rss',
22             'https://news.yandex.ru/internet.rss',
23             'https://news.yandex.ru/cyber_sport.rss',
24             'https://news.yandex.ru/movies.rss',
25             'https://news.yandex.ru/cosmos.rss',
26             'https://news.yandex.ru/culture.rss',
27             'https://news.yandex.ru/fire.rss',
28             'https://news.yandex.ru/championsleague.rss',
29             'https://news.yandex.ru/music.rss',
30             'https://news.yandex.ru/nhl.rss',
31         ];
32         foreach ($rssLinks as $link) {
33             $parserService->saveNews($link);
34         }
35     }
36 }
37
```

Вот список адресов:

```
'https://news.yandex.ru/auto.rss',
'https://news.yandex.ru/auto_racing.rss',
'https://news.yandex.ru/army.rss',
'https://news.yandex.ru/gadgets.rss',
'https://news.yandex.ru/index.rss',
'https://news.yandex.ru/martial_arts.rss',
'https://news.yandex.ru/communal.rss',
'https://news.yandex.ru/health.rss',
'https://news.yandex.ru/games.rss',
'https://news.yandex.ru/internet.rss',
'https://news.yandex.ru/cyber_sport.rss',
'https://news.yandex.ru/movies.rss',
'https://news.yandex.ru/cosmos.rss',
'https://news.yandex.ru/culture.rss',
'https://news.yandex.ru/fire.rss',
'https://news.yandex.ru/championsleague.rss',
'https://news.yandex.ru/music.rss',
'https://news.yandex.ru/nhl.rss',
```

Результат выполнения:



Все работает, но есть нюанс.

А если данные с сервиса будут возвращаться также долго, как из сервиса VK?

А если запросов будет больше сотни?

Пользователь уйдет на другой ресурс. Конечно, запускать парсинг можно в ночное время по расписанию. Но насколько новости будут актуальны?

Чтобы ускорить этот процесс, воспользуемся встроенной функциональностью Laravel — очередями. Сначала создадим шаблон задачи, которую будем отправлять в очередь на выполнение. Команда для этого: **php artisan make:job NewsParsing**

Этот шаблон будет получать единственный параметр — ссылку на ресурс, с которого следует получить информацию. После старта выполнения он будет обращаться к сервису **XMLParserService**, его методу **saveNews**, в который и будет передаваться ссылка.

```
1 <?php
2
3 namespace App\Jobs;
4
5 use App\Services\XMLParserService;
6 use Illuminate\Bus\Queueable;
7 use Illuminate\Contracts\Queue\ShouldQueue;
8 use Illuminate\Foundation\Bus\Dispatchable;
9 use Illuminate\Queue\InteractsWithQueue;
10 use Illuminate\Queue\SerializesModels;
11
12 class NewsParsing implements ShouldQueue
13 {
14     use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
15
16     private $link;
17
18     /** Create a new job instance. ...*/
19     private function __construct($link)
20     {
21         $this->link = $link;
22     }
23
24     /** Execute the job. ...*/
25     public function handle(XMLParserService $parserService)
26     {
27         $parserService->saveNews($this->link);
28     }
29 }
```

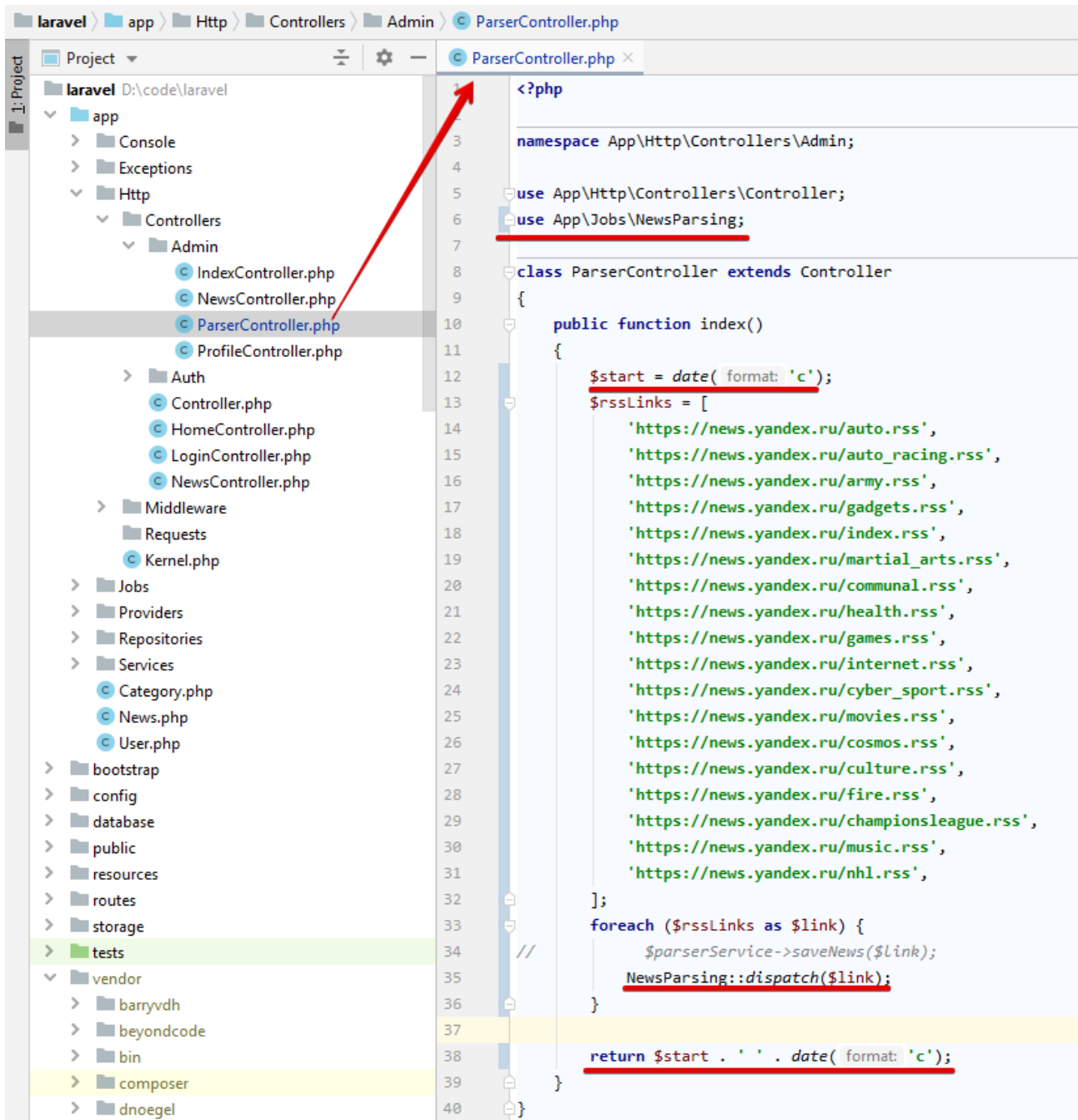
Данные очереди будем хранить в **Redis**. Для этого зайдём в файл **env** и для параметра **QUEUE_CONNECTION** установим значение **redis**.

```
LOG_CHANNEL=stack
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_local
DB_USERNAME=homestead
DB_PASSWORD=secret
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=redis
SESSION_DRIVER=file
SESSION_LIFETIME=120
```

Затем скачаем специальный пакет, который поможет подружить Laravel с redis, установив для фреймворка клиент **redis**. Воспользуемся командой **composer require predis/predis**

Чтобы поставить задачу в очередь, заменим в контроллере обращение к сервису на вызов статического метода **dispatch** у **NewsParsing**. В качестве параметра передадим ссылку на ресурс.

А еще добавим вывод времени в начале и конце выполнения метода **index**, чтобы оценить продолжительность выполнения запроса.



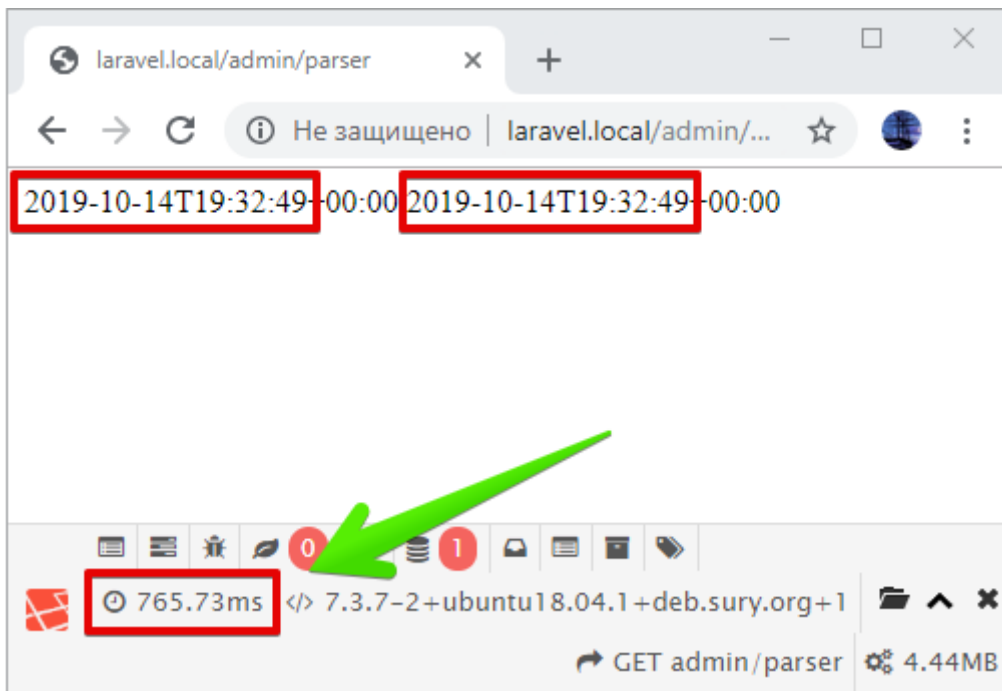
```
<?php
namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Jobs\NewsParsing;

class ParserController extends Controller
{
    public function index()
    {
        $start = date( format: 'c');
        $rssLinks = [
            'https://news.yandex.ru/auto.rss',
            'https://news.yandex.ru/auto_racing.rss',
            'https://news.yandex.ru/army.rss',
            'https://news.yandex.ru/gadgets.rss',
            'https://news.yandex.ru/index.rss',
            'https://news.yandex.ru/martial_arts.rss',
            'https://news.yandex.ru/communal.rss',
            'https://news.yandex.ru/health.rss',
            'https://news.yandex.ru/games.rss',
            'https://news.yandex.ru/internet.rss',
            'https://news.yandex.ru/cyber_sport.rss',
            'https://news.yandex.ru/movies.rss',
            'https://news.yandex.ru/cosmos.rss',
            'https://news.yandex.ru/culture.rss',
            'https://news.yandex.ru/fire.rss',
            'https://news.yandex.ru/championsleague.rss',
            'https://news.yandex.ru/music.rss',
            'https://news.yandex.ru/nhl.rss',
        ];
        foreach ($rssLinks as $link) {
            // $parserService->saveNews($link);
            NewsParsing::dispatch($link);
        }

        return $start . ' ' . date( format: 'c');
    }
}
```

Запускаем:

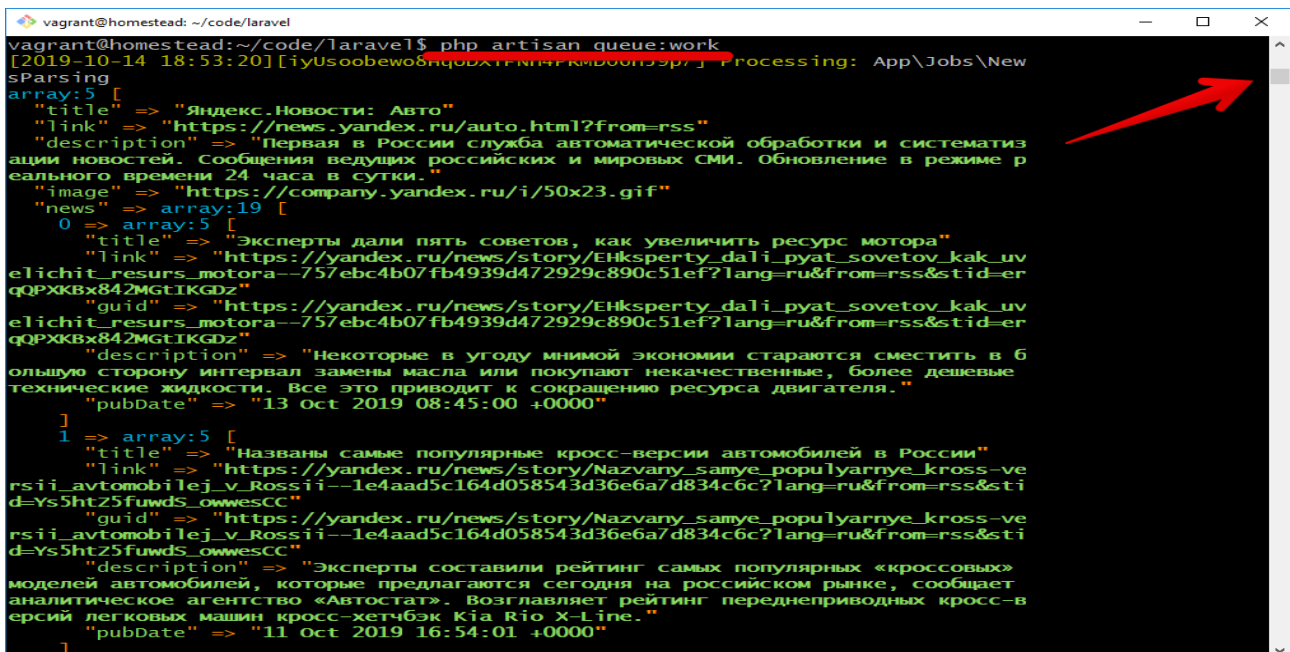


Запрос отработал меньше чем за секунду.

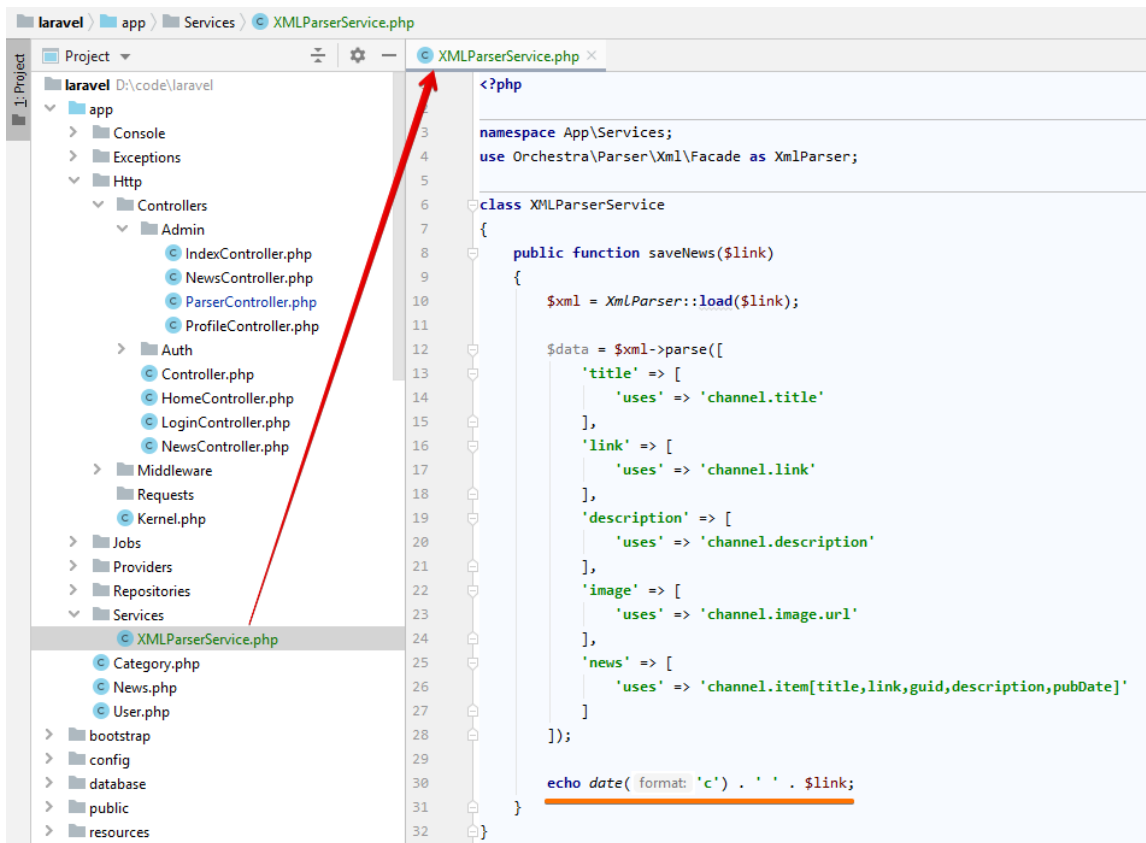
Почему такая разница по сравнению с предыдущим запуском? Все просто — данный запуск не выполнил ни одного парсинга, а только добавил записи в **redis**: название класса задачи, которую необходимо выполнить, и данные для него. То есть каждый раз, выполняя **NewsParsing::dispatch(\$link)**, Laravel добавлял новую запись в **redis**. В ней он указывал класс **NewsParsing** и значение параметра **\$link**.

Как запустить выполнение задач из очереди?

Потребуется демон или воркер — тот скрипт, который будет висеть в фоновом режиме и обрабатывать очередь. Создадим его командой **php artisan queue:work**

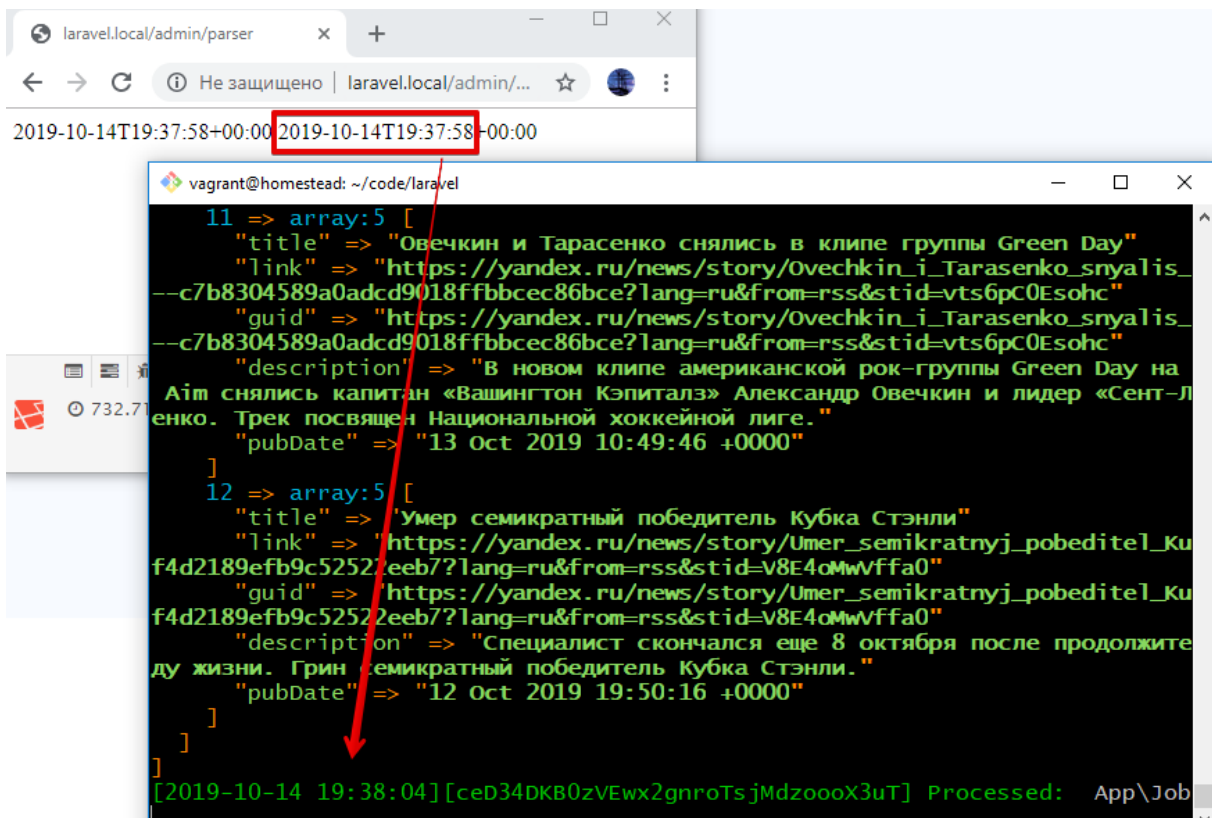


В консоли много информации — сократим ее:



```
<?php
3
4 namespace App\Services;
5 use Orchestra\Parser\Xml\Facade as XmlParser;
6
7 class XMLParserService
8 {
9     public function saveNews($link)
10    {
11        $xml = XmlParser::load($link);
12
13        $data = $xml->parse([
14            'title' => [
15                'uses' => 'channel.title'
16            ],
17            'link' => [
18                'uses' => 'channel.link'
19            ],
20            'description' => [
21                'uses' => 'channel.description'
22            ],
23            'image' => [
24                'uses' => 'channel.image.url'
25            ],
26            'news' => [
27                'uses' => 'channel.item[title,link,guid,description,pubDate]'
28            ]
29        ]));
30
31        echo date('c') . ' ' . $link;
32    }
33 }
```

Еще раз вернемся в браузер и обновим страницу, а после этого откроем консоль.



Воркер сработал — задачи выполнены. Но вывод не изменился.

Дело вот в чем: запуская воркер, программа сохраняется в оперативной памяти — и обращения к файлам не происходит. Чтобы воркер изменился, его надо перезапустить.

Сначала остановим его, нажав на клавиатуре сочетание клавиш Ctrl+C, а затем еще раз запустим.

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ ^C
vagrant@homestead:~/code/laravel$ php artisan queue:work
```

И опять обновим страницу в браузере. Отметим, что вывод сообщений в консоли изменился.

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan queue:work
[2019-10-14 19:41:06] [9TBA361iAfsDffwRNzjU0TnFkR0bTUnT] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:06+00:00 https://news.yandex.ru/auto.rss [2019-10-14 19:41:06] [9TBA361iAfsDffwRNz
[2019-10-14 19:41:06] [uTKA4DSUXo]oEGBZUZQZGynAJTQBWMF] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:06+00:00 https://news.yandex.ru/auto_racing.rss [2019-10-14 19:41:06] [uTKA4DSUXo]
[2019-10-14 19:41:06] [S5GmgkWFZ1LfqOGNqhpXB1zWjT] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:06+00:00 https://news.yandex.ru
[2019-10-14 19:41:06] [LRXiVYi0E781Zo9jcbAyppV7QG] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:07+00:00 https://news.yandex.ru
[2019-10-14 19:41:07] [8RV4B13iEfffqkb2qPX7a9mmwzm] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:07+00:00 https://news.yandex.ru
[2019-10-14 19:41:07] [Gs5QmSYb1lferalJspqQpAPRc2] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:07+00:00 https://news.yandex.ru
[2019-10-14 19:41:07] [Op0ueQ68a8RZAELBD28gmKx5z4] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:07+00:00 https://news.yandex.ru
[2019-10-14 19:41:07] [zdNUW6Ie0cbPiwACr335WRJfXO] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:08+00:00 https://news.yandex.ru
[2019-10-14 19:41:08] [sRLNEEPh1hzWxoiHZMww6Gf2SK] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:08+00:00 https://news.yandex.ru
[2019-10-14 19:41:08] [TocQZZyuVpjtGf0DxangHwwnU7] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:08+00:00 https://news.yandex.ru
[2019-10-14 19:41:08] [C5ACLOdMhYGnuY4rxhtZwUbN3W] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:08+00:00 https://news.yandex.ru
[2019-10-14 19:41:08] [v84tNB9zBfjo1FvzY9FC242uVR] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:09+00:00 https://news.yandex.ru/movies.rss [2019-10-14 19:41:09] [v84tNB9zBfjo1Fvz
[2019-10-14 19:41:09] [hw867Z0ccA2UjtIJBQmB8W64gsU1GAWJ] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:09+00:00 https://news.yandex.ru/cosmos.rss [2019-10-14 19:41:09] [hw867Z0ccA2UjtI
[2019-10-14 19:41:09] [qNpKhKDW2bXfRs7P5JOP42msiiVFPGH9] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:09+00:00 https://news.yandex.ru/culture.rss [2019-10-14 19:41:09] [qNpKhKDW2bXfRs7
[2019-10-14 19:41:09] [XGQz13330gnvmwCqvX2V0qqQE3RNJBBrq] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:09+00:00 https://news.yandex.ru/fire.rss [2019-10-14 19:41:09] [XGQz13330gnvmwCqvX
[2019-10-14 19:41:09] [qNpBDpxzxy3hPFPf1HIy6fSSitw0wEoi] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:10+00:00 https://news.yandex.ru/championsleague.rss [2019-10-14 19:41:10] [qNpBDpx
[2019-10-14 19:41:10] [L9qthREuXKzPabzNIw0yEDrAVNn0FKgz] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:10+00:00 https://news.yandex.ru/music.rss [2019-10-14 19:41:10] [L9qthREuXKzPabzNI
[2019-10-14 19:41:10] [Ap1w0HCqPQdklyYRsZ51Do99I1ZV7YAs] Processing: App\Jobs\NewsParsing
2019-10-14T19:41:10+00:00 https://news.yandex.ru/nhl.rss [2019-10-14 19:41:10] [Ap1w0HCqPQdklyYRsZ5
```

Был запущен только один воркер, поэтому все задачи в очереди выполняются последовательно. Такой вариант работы очереди удобен, например, при загрузке тяжелых файлов на сервер, отправке сообщений.

Для параллельного запуска задач надо запустить несколько воркеров. Их количество воркеров зависит от числа задач, которым предстоит запускаться одновременно. Если задач больше, чем запущенных воркеров, то воркеры, выполнившие задачу, будут забирать следующие, пока не кончатся задачи или их не остановят.

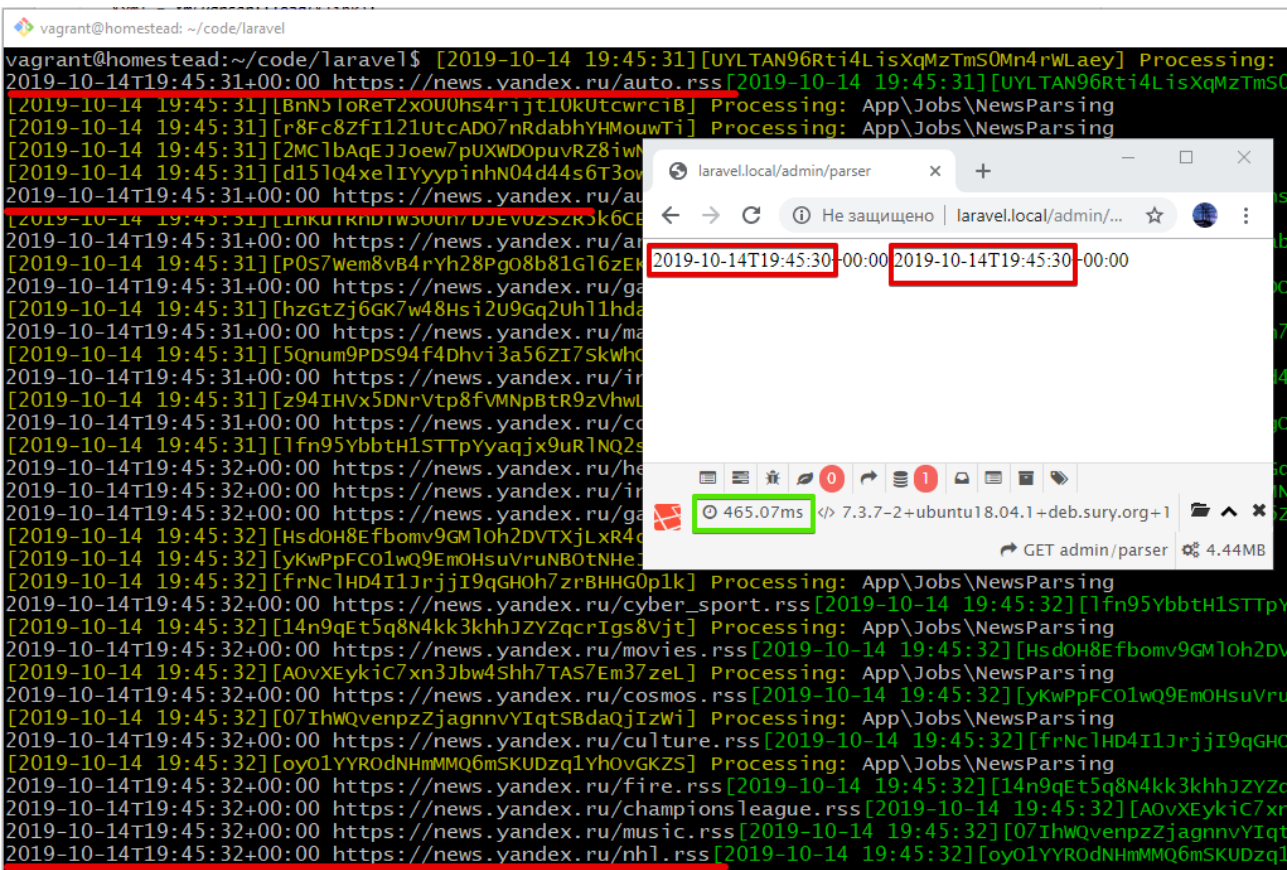
Остановим запущенный воркер.

Команда для запуска нескольких воркеров: `php artisan queue:work &`

```
vagrant@homestead: ~/code/laravel
AC
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[1] 5056
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[2] 5060
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[3] 5064
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[4] 5068
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[5] 5072
vagrant@homestead:~/code/laravel$ |
```

Эта команда отличается от предыдущей одним символом, который запускает воркер в фоновом режиме. Значит, есть возможность запустить и другие. Повторим команду 20 раз. Когда начнется запись задач в очередь, воркеры будут принимать по одной и запустятся параллельно (линков у нас меньше 20).

Перейдем в браузер и обновим страницу.



```
vagrant@homestead:~/code/laravel$ [2019-10-14 19:45:31][UYLTAN96Rti4LisXqMzTmS0Mn4rWLaey] Processing:
2019-10-14T19:45:31+00:00 https://news.yandex.ru/auto.rss [2019-10-14 19:45:31][UYLTAN96Rti4LisXqMzTmS0
[2019-10-14 19:45:31][BnN5IoRet2x0Uhs4r1jt10kUtcwrc1B] Processing: App\Jobs\NewsParsing
[2019-10-14 19:45:31][r8Fc8ZfI121UtcADO7nRdabHYMouwTi] Processing: App\Jobs\NewsParsing
[2019-10-14 19:45:31][2MClbAqEJJJoew7pUXWDOpuvRZ8iw]
[2019-10-14 19:45:31][d15lQ4xe1IYyypinhN04d44s6T3ov]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/au
[2019-10-14 19:45:31][lNKU1RnD1W500h/DJEVUZSZK3k6Ct]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/ai
[2019-10-14 19:45:31][P0S7wem8vB4rYh28Pg08b81G16zEH]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/ga
[2019-10-14 19:45:31][hzGtZj6GK7w48Hsi2U9Gq2Uh1lhd]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/m
[2019-10-14 19:45:31][50num9PDS94f4Dhvi3a56ZI7skwh]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/ir
[2019-10-14 19:45:31][z94IHVx5DNrVtp8fVMNpBtr9zVhw]
2019-10-14T19:45:31+00:00 https://news.yandex.ru/cc
[2019-10-14 19:45:31][1fn95YbbtH1STTpYyaqjx9uR1NQ2s]
2019-10-14T19:45:32+00:00 https://news.yandex.ru/he
2019-10-14T19:45:32+00:00 https://news.yandex.ru/ir
2019-10-14T19:45:32+00:00 https://news.yandex.ru/ga
[2019-10-14 19:45:32][HsdOH8EFbomv9GM1oh2DVTXjLxR4]
[2019-10-14 19:45:32][yKwPpFC01wQ9EmOHsuVruNB0tNHe]
[2019-10-14 19:45:32][frNc1HD4I1Jrjji9qGh0h7zrVhNG0p1k] Processing: App\Jobs\NewsParsing
2019-10-14T19:45:32+00:00 https://news.yandex.ru/cyber_sport.rss [2019-10-14 19:45:32][1fn95YbbtH1STTpY
[2019-10-14 19:45:32][14n9qEt5q8N4kk3khhJZYzqrIgs8Vjt] Processing: App\Jobs\NewsParsing
2019-10-14T19:45:32+00:00 https://news.yandex.ru/movies.rss [2019-10-14 19:45:32][HsdOH8EFbomv9GM1oh2Dv
[2019-10-14 19:45:32][AovXEykic7xn3Jbw4Shh7TAS7Em37zeL] Processing: App\Jobs\NewsParsing
2019-10-14T19:45:32+00:00 https://news.yandex.ru/cosmos.rss [2019-10-14 19:45:32][yKwPpFC01wQ9EmOHsuVru
[2019-10-14 19:45:32][07IhwQvenpzZjagnnvYIqtSBdaQjIzWi] Processing: App\Jobs\NewsParsing
2019-10-14T19:45:32+00:00 https://news.yandex.ru/culture.rss [2019-10-14 19:45:32][frNc1HD4I1Jrjji9qGH
[2019-10-14 19:45:32][oyO1YYRODNHmMMQ6mSKUDzq1YhovGKZS] Processing: App\Jobs\NewsParsing
2019-10-14T19:45:32+00:00 https://news.yandex.ru/fire.rss [2019-10-14 19:45:32][14n9qEt5q8N4kk3khhJZYz
2019-10-14T19:45:32+00:00 https://news.yandex.ru/championsleague.rss [2019-10-14 19:45:32][AovXEykic7xr
2019-10-14T19:45:32+00:00 https://news.yandex.ru/music.rss [2019-10-14 19:45:32][07IhwQvenpzZjagnnvYIqt
2019-10-14T19:45:32+00:00 https://news.yandex.ru/nh1.rss [2019-10-14 19:45:32][oyO1YYRODNHmMMQ6mSKUDzq]
```

Все задачи выполнились примерно за одну секунду — против первоначальных 5,6 секунд.

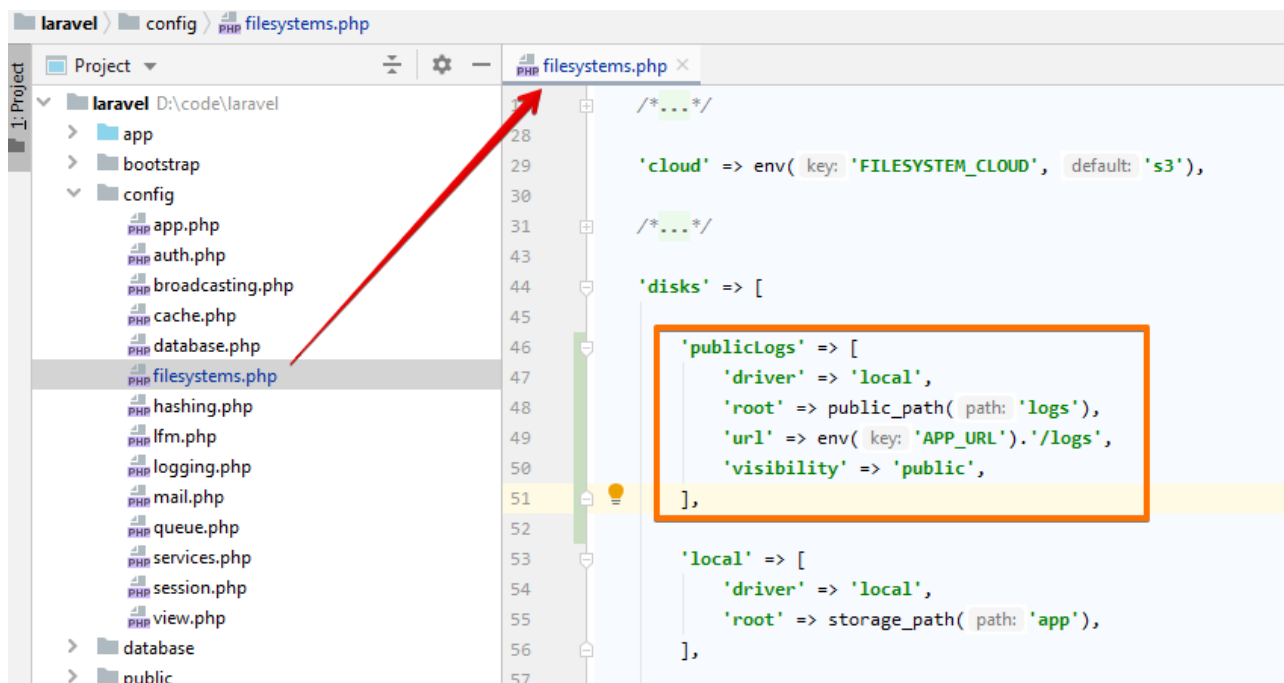
Сохранение файлов

Результат выполнения парсинга правильнее сохранять в базу, но мы сохраним его в файлы. Работая с Laravel, вы можете использовать стандартные функции PHP для работы с файлами, но удобнее пользоваться встроенной функциональностью фреймворка.

В Laravel предустановлен пакет **Flysystem**, который позволяет работать с файловой системой — как локальной, так и удаленной. Настройки файловой системы находятся в файле **filesystems.php**. В нем можно настроить файловые системы — диски. Каждый диск представляет определенный драйвер и

место хранения. В конфигурационном файле есть примеры для каждого поддерживаемого драйвера. Количество дисков ничем не ограничено — есть возможность применять несколько дисков, которые используют один драйвер.

В файле **filesystems.php** хранятся настройки для работы с файловыми системами (подробнее — <https://laravel.ru/docs/v5/filesystem>). Добавим свою настройку.



```
laravel > config > filesystems.php
Project
laravel D:\code\laravel
  app
  bootstrap
  config
    app.php
    auth.php
    broadcasting.php
    cache.php
    database.php
    filesystems.php
    hashing.php
    lfm.php
    logging.php
    mail.php
    queue.php
    services.php
    session.php
    view.php
  database
  public

/*...*/
'cloud' => env( key: 'FILESYSTEM_CLOUD', default: 's3'),
/*...*/

'disks' => [
  'publicLogs' => [
    'driver' => 'local',
    'root' => public_path( path: 'logs'),
    'url' => env( key: 'APP_URL').'/logs',
    'visibility' => 'public',
  ],
  'local' => [
    'driver' => 'local',
    'root' => storage_path( path: 'app'),
  ],
]
```

Эта настройка указывает, что в системе появился новый диск. А также показывает, какой драйвер для него следует использовать, как получить файлы из веб-интерфейса и где они должны физически сохраняться.

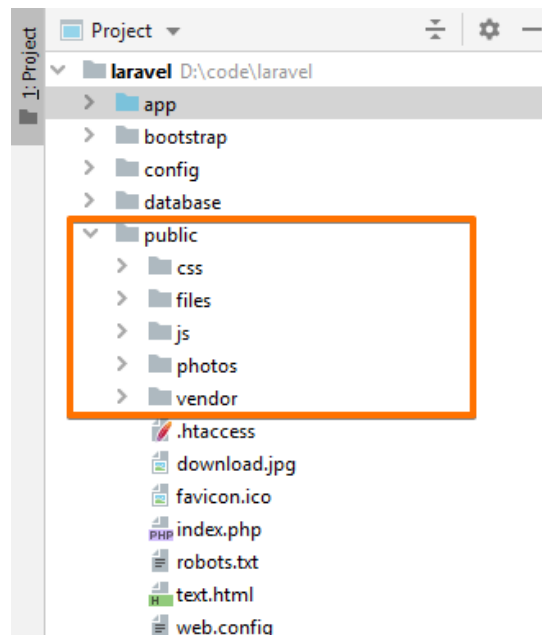
Для работы с файлами используется фасад **Illuminate\Support\Facades\Storage**. Статический метод **disk** данного фасада принимает необязательный параметр — название диска, с которым будет работать. Если он не указан — используется тот, который назначен по умолчанию в файле **filesystems.php**. В результате выполнения этого метода возвращается экземпляр класса **Illuminate\Filesystem\FilesystemAdapter**. Он содержит методы для работы с файлами: удаление, сохранение, запись, проверка на существование и другие.

Реализуем сохранение ссылки, переданной в сервис, в файле **log.txt** внутри созданного «диска».

```
<?php
namespace App\Services;
use Illuminate\Support\Facades\Storage;
use Orchestra\Parser\Xml\Facade as XmlParser;

class XMLParserService
{
    public function saveNews($link)
    {
        $xml = XmlParser::load($link);
        $data = $xml->parse([
            'title' => [
                'uses' => 'channel.title'
            ],
            'link' => [
                'uses' => 'channel.link'
            ],
            'description' => [
                'uses' => 'channel.description'
            ],
            'image' => [
                'uses' => 'channel.image.url'
            ],
            'news' => [
                'uses' => 'channel.item[title,link,guid,description,pubDate]'
            ]
        ]);
        Storage::disk('publicLogs')->append('path: log.txt', data: date('c') . ' ' . $link);
    }
}
```

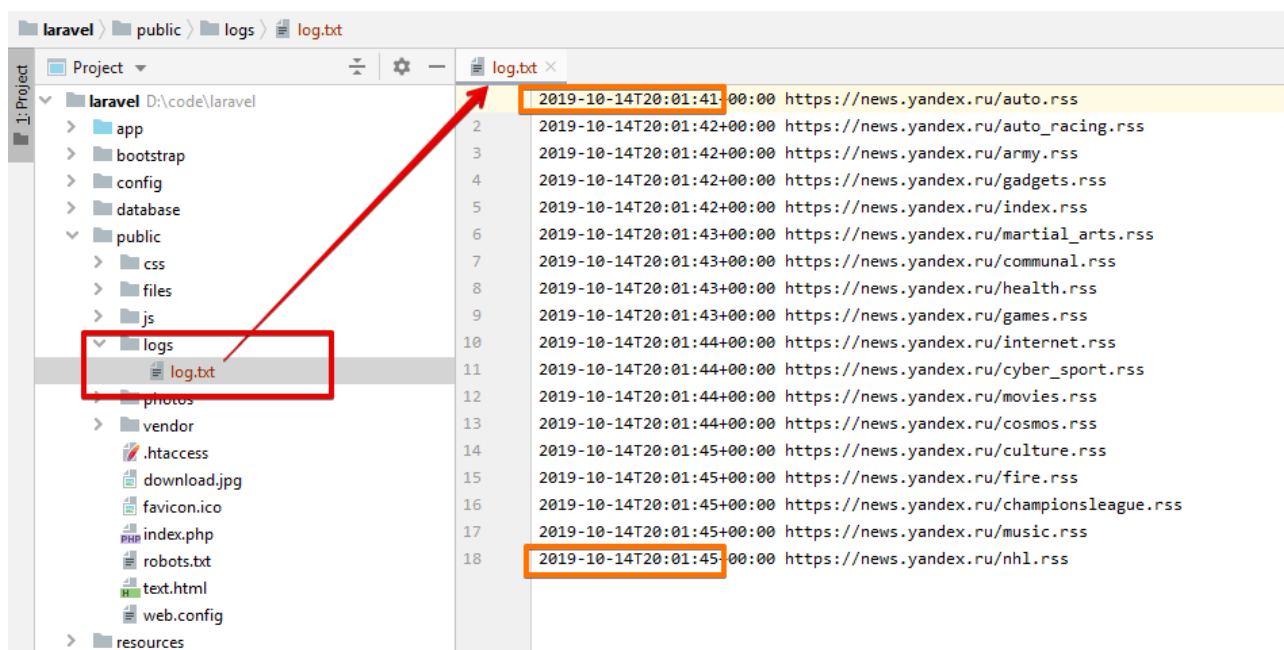
Перед запуском воркеров отметим структуру директории в папке **public**:



Остановим все воркеры и запустим только один.

- **php artisan queue:restart**
- **php artisan queue:work**

Обновим страницу в браузере. Дождемся момента, когда воркер обработает все задачи, и еще раз заглянем в папку **public**.



Отлично! Папка **logs** была создана, а в ней — добавлен необходимый файл, который содержит список ссылок.

Теперь попробуем сохранять файлы с именем, содержащим в себе временную метку.

```

<?php
3 namespace App\Services;
4 use Illuminate\Support\Facades\Storage;
5 use Orchestra\Parser\Xml\Facade as XmlParser;
6
7 class XMLParserService
8 {
9     public function saveNews($link)
10    {
11        $xml = XmlParser::load($link);
12
13        $data = $xml->parse([
14            'title' => [
15                'uses' => 'channel.title'
16            ],
17            'link' => [
18                'uses' => 'channel.link'
19            ],
20            'description' => [
21                'uses' => 'channel.description'
22            ],
23            'image' => [
24                'uses' => 'channel.image.url'
25            ],
26            'news' => [
27                'uses' => 'channel.item[title,link,guid,description,pubDate]'
28            ]
29        ]);
30
31        $fileName = sprintf( format: 'Logs%s.txt', time());
32        Storage::disk( name: 'publicLogs')->put($fileName, $link);
33    }
34 }

```

Перезапустим воркеры и запустим их в количестве 20 штук. Обновим страницу браузера.

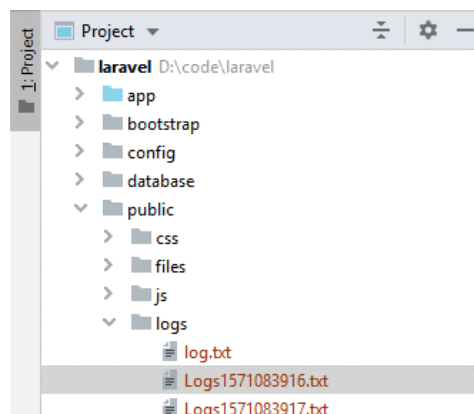
✿ vagrant@homestead: ~/code/laravel

```

^C
vagrant@homestead:~/code/laravel$ php artisan queue:work &
[1] 5203
vagrant@homestead:~/code/laravel$

```

Поскольку воркеры работали практически параллельно, у нас создалось всего два файла.



Внесем небольшой рандом в имена файлов и перезапустим воркеры. Обновим страницу браузера.

```

ParserController.php
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use App\Jobs\NewsParsing;
7
8 class ParserController extends Controller
9 {
10     public function index()
11     {
12         $start = date('c');
13         $rssLinks = [
14             'https://news.yandex.ru/auto.rss',
15             'https://news.yandex.ru/auto_racing.rss',
16             'https://news.yandex.ru/army.rss',
17             'https://news.yandex.ru/gadgets.rss',
18             'https://news.yandex.ru/index.rss',
19             'https://news.yandex.ru/martial_arts.rss',
20             'https://news.yandex.ru/communal.rss',
21             'https://news.yandex.ru/health.rss',
22             'https://news.yandex.ru/games.rss',
23             'https://news.yandex.ru/internet.rss',
24             'https://news.yandex.ru/cyber_sport.rss',
25             'https://news.yandex.ru/movies.rss',
26             'https://news.yandex.ru/cosmos.rss',
27             'https://news.yandex.ru/culture.rss',
28             'https://news.yandex.ru/fire.rss',
29             'https://news.yandex.ru/championsleague.rss',
30             'https://news.yandex.ru/music.rss',
31             'https://news.yandex.ru/nhl.rss',
32         ];
33         foreach ($rssLinks as $link) {
34             // $parserService->saveNews($link);
35             NewsParsing::dispatch($link);
36         }
37     }
38 }
39
XMLParserService.php
1 <?php
2
3 namespace App\Services;
4 use Illuminate\Support\Facades\Storage;
5 use Orchestra\Parser\Xml\Facade as XmlParser;
6
7 class XMLParserService
8 {
9     public function saveNews($link)
10     {
11         $xml = XmlParser::load($link);
12
13         $data = $xml->parse([
14             'title' => [
15                 'uses' => 'channel.title'
16             ],
17             'link' => [
18                 'uses' => 'channel.link'
19             ],
20             'description' => [
21                 'uses' => 'channel.description'
22             ],
23             'image' => [
24                 'uses' => 'channel.image.url'
25             ],
26             'news' => [
27                 'uses' => 'channel.item[title,link,guid,description,pubDate]'
28             ]
29         ]);
30
31         $fileName = sprintf('Logs%s.txt', time() . rand(0, 10000));
32         Storage::disk('publicLogs')->put($fileName, $link);
33     }
34 }
35
File Explorer:
- Default Changelist: 1 file
- Unversioned Files: 18 files
  - Logs1571161730385.txt
  - Logs15711617301115.txt
  - Logs15711617301433.txt
  - Logs15711617301450.txt
  - Logs15711617301586.txt
  - Logs15711617302893.txt
  - Logs15711617303103.txt
  - Logs15711617303252.txt
  - Logs15711617304052.txt
  - Logs15711617305212.txt
  - Logs15711617305349.txt
  - Logs15711617307053.txt
  - Logs15711617308189.txt
  - Logs15711617308787.txt
  - Logs15711617308948.txt
  - Logs15711617309382.txt
  - Logs15711617309453.txt
  - Logs15711617317933.txt
  
```

Практическое задание

1. Оптимизировать использование html-редактора, чтобы его исходники хранились в системе.
2. Создать миграцию для добавления в базу новой таблицы **resources**. Она будет хранить информацию о ресурсах, с которых необходимо забирать новости. Добавить интерфейс для редактирования и добавления данных в эту таблицу.

3. Реализовать алгоритм получения новых новостей из ресурсов, сохраненных в таблице **resources**, с добавлением нужной информации в таблицу **news**.
4. Используя очереди, реализовать алгоритм параллельного запроса информации из сторонних сервисов с выводом ее пользователю в браузер.

Дополнительные материалы

1. <https://laravel.com/docs/5.8/queues>
2. <https://laravel.com/docs/5.8/horizon>
3. <https://laravel.com/docs/5.8/filesystem>
4. <https://ckeditor.com/ckeditor-4/>

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://laravel.su/>
2. <https://laravel.com/docs/6.x/queues/>