**Архитектура хранения медиа-файлов для маркетплейсов**

Основываясь на исследовании практик крупных российских маркетплейсов (Avito, Ozon, Юла) и международных best practices, представляю комплексное решение для локального хранения медиа-файлов с учетом ваших требований: Laravel/PHP, максимум 10 фото на мастера, публичный доступ и простота поддержки одним разработчиком.

**Структура папок для хранения файлов**

Крупные маркетплейсы используют **двухуровневую хеш-структуру** для равномерного распределения миллионов файлов. Эта архитектура предотвращает деградацию производительности файловой системы при большом количестве файлов в одной директории. [medium](https://medium.com/eonian-technologies/file-name-hashing-creating-a-hashed-directory-structure-eabb03aa4091)

**Рекомендуемая структура директорий**

/storage/

├── app/

│ └── media/

│ ├── 00/

│ │ ├── 00/

│ │ ├── 01/

│ │ └── ... (256 поддиректорий)

│ ├── 01/

│ └── ... (256 директорий первого уровня)

├── temp/ # Временные файлы при загрузке

├── cache/ # Кешированные версии изображений

└── backups/ # Локальные бекапы

**Ключевой принцип**: Используется хеш от имени файла для определения пути. Это создает **65,536 возможных директорий** (256×256), позволяя хранить миллионы файлов с производительностью O(1) для доступа. [mediumMedium](https://medium.com/eonian-technologies/file-name-hashing-creating-a-hashed-directory-structure-eabb03aa4091)

**PHP реализация генерации пути**

php

class FilePathGenerator

{

public static function generatePath(string $filename): string

{

$hash = md5($filename);

$dir1 = substr($hash, 0, 2);

$dir2 = substr($hash, 2, 2);

return sprintf('/%s/%s/%s', $dir1, $dir2, $filename);

}

}

**Именование файлов**

Российские маркетплейсы избегают предсказуемых имен для защиты от перебора. Оптимальная стратегия для вашего проекта:

**UUID-based naming (рекомендуется)**

php

use Illuminate\Support\Str;

class FileNameService

{

public function generateSecureFilename(UploadedFile $file): string

{

$extension = $file->getClientOriginalExtension();

$uuid = Str::uuid()->toString();

return sprintf('%s\_%s.%s',

time(), *// Timestamp для сортировки*

$uuid, *// UUID для уникальности*

$extension *// Оригинальное расширение*

);

}

}

**Формат**: 1705123456\_abc123def-456-789.jpg

Преимущества:

* **Невозможен перебор** URL
* **Уникальность** гарантирована
* **Временная метка** помогает в отладке и очистке старых файлов

**Оптимизация изображений**

**Размеры и форматы**

Основываясь на анализе производительности маркетплейсов, рекомендую генерировать **4 версии каждого изображения**:

| **Версия** | **Размер** | **Формат** | **Качество** | **Размер файла** | **Использование** |
| --- | --- | --- | --- | --- | --- |
| thumbnail | 300×300px | WebP/JPEG | 80/75 | <50KB | Списки, превью |
| medium | 800×800px | WebP/JPEG | 85/80 | <150KB | Карточка товара |
| large | 1200×1200px | WebP/JPEG | 90/85 | <300KB | Детальный просмотр |
| original | Как загружено | Оригинал | 100 | <10MB | Резерв/zoom |

**Laravel реализация с Intervention Image**

php

use Intervention\Image\Laravel\Facades\Image;

class ImageOptimizationService

{

private array $sizes = [

'thumbnail' => ['width' => 300, 'height' => 300, 'quality' => 80],

'medium' => ['width' => 800, 'height' => 800, 'quality' => 85],

'large' => ['width' => 1200, 'height' => 1200, 'quality' => 90]

];

public function processImage(string $path): array

{

$results = [];

$image = Image::read($path);

foreach ($this->sizes as $name => $config) {

*// Генерируем JPEG версию*

$jpegPath = $this->generateVersionPath($path, $name, 'jpg');

$image->scale($config['width'], $config['height'])

->toJpeg($config['quality'])

->save($jpegPath);

*// Генерируем WebP версию*

$webpPath = $this->generateVersionPath($path, $name, 'webp');

$image->toWebp($config['quality'] + 5) *// WebP quality выше на 5*

->save($webpPath);

$results[$name] = [

'jpeg' => $jpegPath,

'webp' => $webpPath

];

}

return $results;

}

}

[GitHub +2](https://github.com/Intervention/image)

**Lazy Loading стратегия**

html

*<!-- Первые 3-4 изображения загружаем сразу (для LCP) -->*

<picture>

<source srcset="/media/product1.webp" type="image/webp">

<img src="/media/product1.jpg"

alt="Product"

width="800"

height="800"

fetchpriority="high">

</picture>

*<!-- Остальные с lazy loading -->*

<picture>

<source data-srcset="/media/product5.webp" type="image/webp">

<img data-src="/media/product5.jpg"

loading="lazy"

alt="Product"

width="800"

height="800"

class="lazy">

</picture>

[Cloudinary](https://cloudinary.com/blog/lazy_loading_choosing_the_best_option)[Mozilla](https://developer.mozilla.org/en-US/docs/Web/Performance/Guides/Lazy_loading)

**Структура базы данных**

**Основная таблица медиа-файлов**

sql

CREATE TABLE media\_files (

id BIGSERIAL PRIMARY KEY,

uuid UUID UNIQUE NOT NULL DEFAULT gen\_random\_uuid(),

master\_id BIGINT NOT NULL,

original\_name VARCHAR(255) NOT NULL,

stored\_name VARCHAR(255) NOT NULL,

file\_path VARCHAR(500) NOT NULL,

file\_hash VARCHAR(64) NOT NULL,

mime\_type VARCHAR(100) NOT NULL,

file\_size BIGINT NOT NULL,

width INTEGER,

height INTEGER,

sort\_order INTEGER DEFAULT 0,

is\_primary BOOLEAN DEFAULT FALSE,

metadata JSONB,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

INDEX idx\_master\_files (master\_id, sort\_order),

INDEX idx\_file\_hash (file\_hash),

INDEX idx\_uuid (uuid),

CONSTRAINT max\_10\_photos CHECK (

(SELECT COUNT(\*) FROM media\_files WHERE master\_id = NEW.master\_id) <= 10

)

);

**Таблица версий изображений**

sql

CREATE TABLE image\_versions (

id BIGSERIAL PRIMARY KEY,

media\_file\_id BIGINT REFERENCES media\_files(id) ON DELETE CASCADE,

version\_type VARCHAR(20) NOT NULL, *-- 'thumbnail', 'medium', 'large'*

format VARCHAR(10) NOT NULL, *-- 'jpeg', 'webp'*

file\_path VARCHAR(500) NOT NULL,

file\_size BIGINT NOT NULL,

width INTEGER NOT NULL,

height INTEGER NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

INDEX idx\_media\_versions (media\_file\_id, version\_type)

);

**Защита от перебора URL**

**Signed URLs с временными ограничениями**

php

use Illuminate\Support\Facades\URL;

class SecureMediaController extends Controller

{

public function generateSecureUrl(MediaFile $file): string

{

*// Генерируем подписанный URL на 1 час*

return URL::temporarySignedRoute(

'media.download',

now()->addHour(),

[

'uuid' => $file->uuid,

'version' => request('version', 'medium')

]

);

}

public function download(Request $request, string $uuid)

{

*// Проверка подписи*

if (!$request->hasValidSignature()) {

abort(403, 'Invalid or expired link');

}

*// Rate limiting*

if (RateLimiter::tooManyAttempts('download:' . $request->ip(), 60)) {

abort(429, 'Too many requests');

}

RateLimiter::increment('download:' . $request->ip());

$file = MediaFile::where('uuid', $uuid)->firstOrFail();

return Storage::disk('media')->response($file->file\_path);

}

}

**Дополнительные меры защиты**

php

*// Middleware для защиты от hotlinking*

class PreventHotlinking

{

public function handle($request, Closure $next)

{

$referer = $request->headers->get('referer');

$allowedDomains = [config('app.url'), 'https://yandex.ru'];

if ($referer && !Str::startsWith($referer, $allowedDomains)) {

abort(403, 'Hotlinking not allowed');

}

return $next($request);

}

}

**Версионирование и кеширование**

**HTTP заголовки для оптимального кеширования**

php

class MediaCacheService

{

public function serveWithCache(string $path): Response

{

$response = Storage::disk('media')->response($path);

*// Кеширование на 1 год для неизменяемых файлов*

$response->headers->set('Cache-Control', 'public, max-age=31536000, immutable');

$response->headers->set('ETag', md5\_file($path));

*// Защитные заголовки*

$response->headers->set('X-Content-Type-Options', 'nosniff');

$response->headers->set('X-Frame-Options', 'SAMEORIGIN');

return $response;

}

}

**Версионирование через URL параметры**

php

*// Генерация URL с версией*

$imageUrl = route('media.show', [

'uuid' => $file->uuid,

'v' => substr($file->file\_hash, 0, 8) *// Первые 8 символов хеша*

]);

*// Результат: /media/abc-123?v=a1b2c3d4*

**Обработка разных размеров**

**Queue для асинхронной обработки**

php

use Illuminate\Bus\Queueable;

use Illuminate\Contracts\Queue\ShouldQueue;

use Illuminate\Queue\InteractsWithQueue;

class ProcessUploadedImage implements ShouldQueue

{

use Queueable, InteractsWithQueue;

public $timeout = 120; *// 2 минуты*

public $tries = 3;

public function \_\_construct(

private int $mediaFileId

) {}

public function handle(ImageOptimizationService $optimizer): void

{

$mediaFile = MediaFile::findOrFail($this->mediaFileId);

$originalPath = Storage::disk('media')->path($mediaFile->file\_path);

*// Генерируем все размеры*

$versions = $optimizer->processImage($originalPath);

*// Сохраняем информацию о версиях*

foreach ($versions as $type => $paths) {

foreach ($paths as $format => $path) {

ImageVersion::create([

'media\_file\_id' => $mediaFile->id,

'version\_type' => $type,

'format' => $format,

'file\_path' => $path,

'file\_size' => filesize($path),

'width' => getimagesize($path)[0],

'height' => getimagesize($path)[1]

]);

}

}

}

}

*// Dispatch при загрузке*

ProcessUploadedImage::dispatch($mediaFile->id)

->onQueue('media\_processing');

**Ограничения на размер и количество**

**Валидация в контроллере**

php

class MasterMediaController extends Controller

{

public function upload(Request $request, Master $master)

{

*// Проверка количества существующих фото*

$existingCount = $master->mediaFiles()->count();

$uploadCount = count($request->file('photos', []));

if ($existingCount + $uploadCount > 10) {

return response()->json([

'error' => 'Максимум 10 фото на мастера',

'current' => $existingCount,

'limit' => 10

], 422);

}

$request->validate([

'photos' => 'required|array|max:10',

'photos.\*' => [

'required',

'image',

'mimes:jpeg,jpg,png,webp',

'max:10240', *// 10MB*

'dimensions:min\_width=300,min\_height=300,max\_width=5000,max\_height=5000'

]

]);

*// Обработка загрузки...*

}

}

**Очереди для обработки медиа**

**Конфигурация Redis очередей**

php

*// config/queue.php*

'connections' => [

'redis' => [

'driver' => 'redis',

'connection' => 'default',

'queue' => 'default',

'retry\_after' => 90,

],

'media' => [

'driver' => 'redis',

'connection' => 'default',

'queue' => 'media\_processing',

'retry\_after' => 300, *// 5 минут для обработки изображений*

],

],

*// .env*

QUEUE\_CONNECTION=redis

**Supervisor конфигурация**

ini

[program:laravel-media-worker]

process\_name=%(program\_name)s\_%(process\_num)02d

command=php /var/www/artisan queue:work redis --queue=media\_processing --sleep=3 --tries=3 --max-time=3600

autostart=true

autorestart=true

stopasgroup=true

killasgroup=true

user=www-data

numprocs=2

redirect\_stderr=true

stdout\_logfile=/var/log/laravel-media-worker.log

**Backup стратегии**

**Автоматизированный backup скрипт**

bash

#!/bin/bash

*# /usr/local/bin/backup-media.sh*

SOURCE="/var/www/storage/app/media"

BACKUP="/backup/media"

DATE=$(date +%Y%m%d\_%H%M%S)

RETENTION\_DAYS=30

*# Инкрементальный backup с rsync*

rsync -av --link-dest=$BACKUP/latest $SOURCE/ $BACKUP/$DATE/

*# Обновляем симлинк на последний backup*

rm -f $BACKUP/latest

ln -s $BACKUP/$DATE $BACKUP/latest

*# Удаляем старые backup'ы*

find $BACKUP -maxdepth 1 -type d -mtime +$RETENTION\_DAYS -exec rm -rf {} \;

*# Опционально: загрузка на удаленный сервер*

*# rsync -av $BACKUP/$DATE/ backup@remote:/backup/media/*

**Laravel команда для backup**

php

class BackupMediaCommand extends Command

{

protected $signature = 'media:backup';

public function handle(): void

{

$this->info('Starting media backup...');

$timestamp = now()->format('Y-m-d\_H-i-s');

$mediaPath = storage\_path('app/media');

$backupPath = storage\_path("backups/media\_{$timestamp}.tar.gz");

*// Создаем архив*

exec("tar -czf {$backupPath} -C {$mediaPath} .", $output, $result);

if ($result === 0) {

$this->info("Backup created: {$backupPath}");

*// Очищаем старые backup'ы*

$this->cleanOldBackups();

}

}

private function cleanOldBackups(): void

{

$files = Storage::disk('backups')->files();

$oldFiles = collect($files)

->filter(fn($file) =>

Carbon::createFromTimestamp(filemtime($file))->lt(now()->subDays(30))

);

Storage::disk('backups')->delete($oldFiles->toArray());

}

}

**Рекомендации для простой поддержки**

**Минимальный production-ready стек**

1. **Laravel 10+ с Spatie Media Library** - проверенное решение с отличной документацией
2. **Redis для очередей** - простая настройка, высокая производительность
3. **Intervention Image v3** - современная библиотека с поддержкой всех форматов [Laravel News](https://laravel-news.com/intervention-image)[Intervention Image](https://image.intervention.io/v3)
4. **Supervisor** - для управления воркерами очередей
5. **Cron + rsync** - для автоматических backup'ов

**Мониторинг и обслуживание**

php

*// app/Console/Kernel.php*

protected function schedule(Schedule $schedule)

{

*// Ежедневный backup в 3 ночи*

$schedule->command('media:backup')->dailyAt('03:00');

*// Очистка временных файлов*

$schedule->command('media:clean-temp')->hourly();

*// Проверка целостности*

$schedule->command('media:health-check')->twiceDaily();

}

**Контрольный список для запуска**

* Настроить структуру директорий с правами 755/644
* Установить Spatie Media Library [Spatie](https://spatie.be/docs/laravel-medialibrary/v11/installation-setup) и Intervention Image
* Настроить Redis и Supervisor для очередей
* Реализовать signed URLs для защиты
* Настроить cron для backup'ов
* Добавить rate limiting на endpoint'ы [Medium](https://medium.com/@antoine.lame/using-the-rate-limiter-in-laravel-dab58a5040bc)[Hirededicatedlaraveldeveloper](https://www.hirededicatedlaraveldeveloper.com/blog/api-rate-limiting-in-laravel-everything-you-need-to-know/)
* Протестировать загрузку 10 фото одновременно
* Проверить генерацию всех размеров изображений
* Настроить мониторинг дискового пространства
* Документировать API для фронтенда

Это решение обеспечивает баланс между производительностью уровня крупных маркетплейсов и простотой поддержки для одного разработчика, полностью соответствуя вашим требованиям по локальному хранению и ограничению в 10 фото на мастера.