**Детальный пошаговый план реализации хранилища медиа-файлов**

**ЭТАП 1: ПОДГОТОВКА ИНФРАСТРУКТУРЫ (30 минут)**

**Шаг 1.1: Создание структуры директорий**

bash

*# Создай эти команды в корне проекта*

mkdir -p storage/app/media

mkdir -p storage/app/temp

mkdir -p storage/app/cache

mkdir -p storage/backups/media

*# Установи права доступа*

chmod -R 755 storage/app/media

chmod -R 755 storage/app/temp

chmod -R 755 storage/app/cache

**Шаг 1.2: Установка необходимых пакетов**

bash

*# Выполни в терминале*

composer require intervention/image-laravel

composer require intervention/image

composer require ramsey/uuid

**Шаг 1.3: Публикация конфигурации**

bash

php artisan vendor:publish --provider="Intervention\Image\Laravel\ServiceProvider"

**ЭТАП 2: СОЗДАНИЕ БАЗЫ ДАННЫХ (20 минут)**

**Шаг 2.1: Создание миграции для media\_files**

bash

php artisan make:migration create\_media\_files\_table

**Содержимое миграции:**

php

// database/migrations/2024\_xx\_xx\_create\_media\_files\_table.php

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

public function up()

{

Schema::create('media\_files', function (Blueprint $table) {

$table->id();

$table->uuid('uuid')->unique();

$table->foreignId('master\_id')->constrained('masters')->onDelete('cascade');

$table->string('original\_name');

$table->string('stored\_name');

$table->string('file\_path', 500);

$table->string('file\_hash', 64);

$table->string('mime\_type', 100);

$table->bigInteger('file\_size');

$table->integer('width')->nullable();

$table->integer('height')->nullable();

$table->integer('sort\_order')->default(0);

$table->boolean('is\_primary')->default(false);

$table->json('metadata')->nullable();

$table->timestamps();

$table->index(['master\_id', 'sort\_order']);

$table->index('file\_hash');

$table->index('uuid');

});

}

public function down()

{

Schema::dropIfExists('media\_files');

}

};

**Шаг 2.2: Создание миграции для image\_versions**

bash

php artisan make:migration create\_image\_versions\_table

**Содержимое миграции:**

php

// database/migrations/2024\_xx\_xx\_create\_image\_versions\_table.php

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

public function up()

{

Schema::create('image\_versions', function (Blueprint $table) {

$table->id();

$table->foreignId('media\_file\_id')->constrained('media\_files')->onDelete('cascade');

$table->enum('version\_type', ['thumbnail', 'medium', 'large']);

$table->enum('format', ['jpeg', 'webp']);

$table->string('file\_path', 500);

$table->bigInteger('file\_size');

$table->integer('width');

$table->integer('height');

$table->timestamps();

$table->index(['media\_file\_id', 'version\_type']);

});

}

public function down()

{

Schema::dropIfExists('image\_versions');

}

};

**Шаг 2.3: Запуск миграций**

bash

php artisan migrate

**ЭТАП 3: СОЗДАНИЕ СЕРВИСОВ (1 час)**

**Шаг 3.1: Создание сервиса генерации путей**

**Создай файл:** app/Services/Media/FilePathService.php

php

<?php

namespace App\Services\Media;

use Illuminate\Support\Str;

class FilePathService

{

*/\*\**

\* Генерация безопасного имени файла

*\*/*

public function generateSecureFilename(string $originalName): string

{

$extension = pathinfo($originalName, PATHINFO\_EXTENSION);

$timestamp = time();

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

return sprintf('%s\_%s.%s', $timestamp, $uuid, strtolower($extension));

}

*/\*\**

\* Генерация пути на основе хеша (как на больших маркетплейсах)

*\*/*

public function generateHashPath(string $filename): string

{

$hash = md5($filename);

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

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

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

}

*/\*\**

\* Полный путь для сохранения

*\*/*

public function getFullPath(string $filename): string

{

return 'media/' . $this->generateHashPath($filename);

}

*/\*\**

\* Создание директорий если не существуют

*\*/*

public function ensureDirectoryExists(string $path): void

{

$directory = storage\_path('app/' . dirname($path));

if (!file\_exists($directory)) {

mkdir($directory, 0755, true);

}

}

}

**Шаг 3.2: Создание сервиса оптимизации изображений**

**Создай файл:** app/Services/Media/ImageOptimizationService.php

php

<?php

namespace App\Services\Media;

use Intervention\Image\Laravel\Facades\Image;

use Illuminate\Support\Facades\Storage;

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

]

];

*/\*\**

\* Обработка загруженного изображения

*\*/*

public function processImage(string $originalPath): array

{

$results = [];

$pathInfo = pathinfo($originalPath);

$directory = $pathInfo['dirname'];

$filename = $pathInfo['filename'];

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

*// Создаем JPEG версию*

$jpegFilename = sprintf('%s\_%s.jpg', $filename, $sizeName);

$jpegPath = $directory . '/' . $jpegFilename;

$image = Image::read(storage\_path('app/' . $originalPath));

*// Пропорциональное изменение размера*

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

*// Сохраняем JPEG*

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

->save(storage\_path('app/' . $jpegPath));

*// Создаем WebP версию*

$webpFilename = sprintf('%s\_%s.webp', $filename, $sizeName);

$webpPath = $directory . '/' . $webpFilename;

$image->toWebp($config['quality'])

->save(storage\_path('app/' . $webpPath));

$results[$sizeName] = [

'jpeg' => [

'path' => $jpegPath,

'size' => filesize(storage\_path('app/' . $jpegPath)),

'width' => $image->width(),

'height' => $image->height()

],

'webp' => [

'path' => $webpPath,

'size' => filesize(storage\_path('app/' . $webpPath)),

'width' => $image->width(),

'height' => $image->height()

]

];

}

return $results;

}

*/\*\**

\* Удаление всех версий изображения

*\*/*

public function deleteAllVersions(string $originalPath): void

{

$pathInfo = pathinfo($originalPath);

$directory = $pathInfo['dirname'];

$filename = $pathInfo['filename'];

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

Storage::delete($directory . '/' . $filename . '\_' . $sizeName . '.jpg');

Storage::delete($directory . '/' . $filename . '\_' . $sizeName . '.webp');

}

}

}

**Шаг 3.3: Создание основного сервиса управления медиа**

**Создай файл:** app/Services/Media/MediaService.php

php

<?php

namespace App\Services\Media;

use App\Models\MediaFile;

use App\Models\ImageVersion;

use Illuminate\Http\UploadedFile;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Str;

class MediaService

{

public function \_\_construct(

private FilePathService $pathService,

private ImageOptimizationService $imageService

) {}

*/\*\**

\* Загрузка и обработка изображения

*\*/*

public function uploadImage(UploadedFile $file, int $masterId): MediaFile

{

return DB::transaction(function () use ($file, $masterId) {

*// Проверяем количество существующих фото*

$existingCount = MediaFile::where('master\_id', $masterId)->count();

if ($existingCount >= 10) {

throw new \Exception('Максимум 10 фото на мастера');

}

*// Генерируем безопасное имя*

$originalName = $file->getClientOriginalName();

$storedName = $this->pathService->generateSecureFilename($originalName);

$fullPath = $this->pathService->getFullPath($storedName);

*// Создаем директории*

$this->pathService->ensureDirectoryExists($fullPath);

*// Сохраняем оригинал*

$file->storeAs(dirname($fullPath), basename($fullPath));

*// Получаем информацию об изображении*

$imageInfo = getimagesize(storage\_path('app/' . $fullPath));

*// Создаем запись в БД*

$mediaFile = MediaFile::create([

'uuid' => Str::uuid(),

'master\_id' => $masterId,

'original\_name' => $originalName,

'stored\_name' => $storedName,

'file\_path' => $fullPath,

'file\_hash' => md5\_file(storage\_path('app/' . $fullPath)),

'mime\_type' => $file->getMimeType(),

'file\_size' => $file->getSize(),

'width' => $imageInfo[0] ?? null,

'height' => $imageInfo[1] ?? null,

'sort\_order' => $existingCount,

'is\_primary' => $existingCount === 0

]);

*// Обрабатываем изображение асинхронно*

$this->processImageVersions($mediaFile);

return $mediaFile;

});

}

*/\*\**

\* Обработка версий изображения

*\*/*

public function processImageVersions(MediaFile $mediaFile): void

{

$versions = $this->imageService->processImage($mediaFile->file\_path);

foreach ($versions as $versionType => $formats) {

foreach ($formats as $format => $data) {

ImageVersion::create([

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

'version\_type' => $versionType,

'format' => $format === 'jpeg' ? 'jpeg' : 'webp',

'file\_path' => $data['path'],

'file\_size' => $data['size'],

'width' => $data['width'],

'height' => $data['height']

]);

}

}

}

*/\*\**

\* Удаление медиа-файла

*\*/*

public function deleteMedia(MediaFile $mediaFile): bool

{

return DB::transaction(function () use ($mediaFile) {

*// Удаляем все версии из файловой системы*

$this->imageService->deleteAllVersions($mediaFile->file\_path);

*// Удаляем оригинал*

Storage::delete($mediaFile->file\_path);

*// Удаляем из БД (версии удалятся каскадно)*

return $mediaFile->delete();

});

}

*/\*\**

\* Изменение порядка фото

*\*/*

public function reorderPhotos(int $masterId, array $photoIds): void

{

foreach ($photoIds as $order => $photoId) {

MediaFile::where('id', $photoId)

->where('master\_id', $masterId)

->update(['sort\_order' => $order]);

}

}

*/\*\**

\* Установка главного фото

*\*/*

public function setPrimaryPhoto(int $masterId, int $photoId): void

{

DB::transaction(function () use ($masterId, $photoId) {

*// Сбрасываем текущее главное*

MediaFile::where('master\_id', $masterId)

->update(['is\_primary' => false]);

*// Устанавливаем новое главное*

MediaFile::where('id', $photoId)

->where('master\_id', $masterId)

->update(['is\_primary' => true]);

});

}

}

**ЭТАП 4: СОЗДАНИЕ МОДЕЛЕЙ (20 минут)**

**Шаг 4.1: Создание модели MediaFile**

**Создай файл:** app/Models/MediaFile.php

php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Support\Facades\Storage;

class MediaFile extends Model

{

protected $fillable = [

'uuid',

'master\_id',

'original\_name',

'stored\_name',

'file\_path',

'file\_hash',

'mime\_type',

'file\_size',

'width',

'height',

'sort\_order',

'is\_primary',

'metadata'

];

protected $casts = [

'metadata' => 'array',

'is\_primary' => 'boolean',

'file\_size' => 'integer',

'width' => 'integer',

'height' => 'integer',

'sort\_order' => 'integer'

];

*/\*\**

\* Мастер владелец

*\*/*

public function master(): BelongsTo

{

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

}

*/\*\**

\* Версии изображения

*\*/*

public function versions(): HasMany

{

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

}

*/\*\**

\* Получить URL для версии

*\*/*

public function getUrl(string $version = 'medium', string $format = 'jpeg'): string

{

$imageVersion = $this->versions()

->where('version\_type', $version)

->where('format', $format)

->first();

if ($imageVersion) {

return Storage::url($imageVersion->file\_path);

}

return Storage::url($this->file\_path);

}

*/\*\**

\* Получить подписанный URL

*\*/*

public function getSignedUrl(string $version = 'medium'): string

{

return route('media.show', [

'uuid' => $this->uuid,

'version' => $version,

'signature' => $this->generateSignature($version)

]);

}

*/\*\**

\* Генерация подписи для URL

*\*/*

private function generateSignature(string $version): string

{

return hash\_hmac('sha256', $this->uuid . $version, config('app.key'));

}

*/\*\**

\* Размер в человекочитаемом формате

*\*/*

public function getHumanSize(): string

{

$bytes = $this->file\_size;

$units = ['B', 'KB', 'MB', 'GB'];

for ($i = 0; $bytes > 1024 && $i < count($units) - 1; $i++) {

$bytes /= 1024;

}

return round($bytes, 2) . ' ' . $units[$i];

}

}

**Шаг 4.2: Создание модели ImageVersion**

**Создай файл:** app/Models/ImageVersion.php

php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

class ImageVersion extends Model

{

protected $fillable = [

'media\_file\_id',

'version\_type',

'format',

'file\_path',

'file\_size',

'width',

'height'

];

protected $casts = [

'file\_size' => 'integer',

'width' => 'integer',

'height' => 'integer'

];

*/\*\**

\* Основной медиа-файл

*\*/*

public function mediaFile(): BelongsTo

{

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

}

*/\*\**

\* Получить URL версии

*\*/*

public function getUrl(): string

{

return asset('storage/' . $this->file\_path);

}

}

**ЭТАП 5: СОЗДАНИЕ КОНТРОЛЛЕРА (30 минут)**

**Шаг 5.1: Создание контроллера для медиа**

**Создай файл:** app/Http/Controllers/MasterMediaController.php

php

<?php

namespace App\Http\Controllers;

use App\Models\Master;

use App\Models\MediaFile;

use App\Services\Media\MediaService;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\RateLimiter;

use Inertia\Inertia;

class MasterMediaController extends Controller

{

public function \_\_construct(

private MediaService $mediaService

) {}

*/\*\**

\* Показать страницу управления медиа

*\*/*

public function index(Master $master)

{

$this->authorize('update', $master);

$mediaFiles = $master->mediaFiles()

->with('versions')

->orderBy('sort\_order')

->get();

return Inertia::render('Master/Media/Index', [

'master' => $master,

'mediaFiles' => $mediaFiles

]);

}

*/\*\**

\* Загрузка изображений

*\*/*

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

{

$this->authorize('update', $master);

*// Rate limiting: 10 загрузок в минуту*

$key = 'upload:' . $request->user()->id;

if (RateLimiter::tooManyAttempts($key, 10)) {

return response()->json([

'message' => 'Слишком много попыток. Попробуйте позже.'

], 429);

}

RateLimiter::increment($key);

*// Валидация*

$request->validate([

'photo' => [

'required',

'image',

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

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

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

]

]);

try {

$mediaFile = $this->mediaService->uploadImage(

$request->file('photo'),

$master->id

);

return response()->json([

'success' => true,

'media' => $mediaFile->load('versions')

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => $e->getMessage()

], 422);

}

}

*/\*\**

\* Удаление фото

*\*/*

public function destroy(Master $master, MediaFile $mediaFile)

{

$this->authorize('update', $master);

if ($mediaFile->master\_id !== $master->id) {

abort(403, 'Фото не принадлежит этому мастеру');

}

$this->mediaService->deleteMedia($mediaFile);

return response()->json([

'success' => true,

'message' => 'Фото удалено'

]);

}

*/\*\**

\* Изменение порядка фото

*\*/*

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

{

$this->authorize('update', $master);

$request->validate([

'photo\_ids' => 'required|array',

'photo\_ids.\*' => 'exists:media\_files,id'

]);

$this->mediaService->reorderPhotos($master->id, $request->photo\_ids);

return response()->json([

'success' => true,

'message' => 'Порядок обновлен'

]);

}

*/\*\**

\* Установка главного фото

*\*/*

public function setPrimary(Master $master, MediaFile $mediaFile)

{

$this->authorize('update', $master);

if ($mediaFile->master\_id !== $master->id) {

abort(403, 'Фото не принадлежит этому мастеру');

}

$this->mediaService->setPrimaryPhoto($master->id, $mediaFile->id);

return response()->json([

'success' => true,

'message' => 'Главное фото установлено'

]);

}

*/\*\**

\* Отдача медиа-файла (с проверкой подписи)

*\*/*

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

{

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

*// Проверка подписи для приватных файлов*

if ($request->has('signature')) {

$expectedSignature = hash\_hmac(

'sha256',

$uuid . $request->get('version', 'medium'),

config('app.key')

);

if (!hash\_equals($expectedSignature, $request->signature)) {

abort(403, 'Invalid signature');

}

}

$version = $request->get('version', 'medium');

$format = $request->get('format', 'jpeg');

$imageVersion = $mediaFile->versions()

->where('version\_type', $version)

->where('format', $format)

->first();

if (!$imageVersion) {

abort(404, 'Version not found');

}

$path = storage\_path('app/' . $imageVersion->file\_path);

return response()->file($path, [

'Cache-Control' => 'public, max-age=31536000, immutable',

'ETag' => md5\_file($path)

]);

}

}

**ЭТАП 6: СОЗДАНИЕ РОУТОВ (10 минут)**

**Шаг 6.1: Добавление роутов в web.php**

**Добавь в файл:** routes/web.php

php

use App\Http\Controllers\MasterMediaController;

*// Группа роутов для авторизованных пользователей*

Route::middleware(['auth'])->group(function () {

*// Управление медиа мастера*

Route::prefix('masters/{master}/media')->name('masters.media.')->group(function () {

Route::get('/', [MasterMediaController::class, 'index'])->name('index');

Route::post('/upload', [MasterMediaController::class, 'upload'])->name('upload');

Route::delete('/{mediaFile}', [MasterMediaController::class, 'destroy'])->name('destroy');

Route::post('/reorder', [MasterMediaController::class, 'reorder'])->name('reorder');

Route::post('/{mediaFile}/set-primary', [MasterMediaController::class, 'setPrimary'])->name('set-primary');

});

});

*// Публичный роут для отдачи медиа*

Route::get('/media/{uuid}', [MasterMediaController::class, 'show'])->name('media.show');

**ЭТАП 7: СОЗДАНИЕ FRONTEND КОМПОНЕНТА (40 минут)**

**Шаг 7.1: Создание Vue компонента для загрузки**

**Создай файл:** resources/js/Pages/Master/Media/Index.vue

vue

<template>

<div class="master-media-page">

<h1>Управление фотографиями</h1>

<!-- Счетчик фото -->

<div class="photo-counter">

Загружено {{ photos.length }} из 10 фото

</div>

<!-- Зона загрузки -->

<div

v-if="photos.length < 10"

class="upload-zone"

@dragover.prevent="isDragging = true"

@dragleave.prevent="isDragging = false"

@drop.prevent="handleDrop"

:class="{ 'dragging': isDragging }"

>

<input

ref="fileInput"

type="file"

accept="image/\*"

multiple

@change="handleFileSelect"

style="display: none"

>

<div @click="$refs.fileInput.click()" class="upload-content">

<svg class="upload-icon" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke-linecap="round" stroke-linejoin="round" stroke-width="2"

d="M7 16a4 4 0 01-.88-7.903A5 5 0 1115.9 6L16 6a5 5 0 011 9.9M15 13l-3-3m0 0l-3 3m3-3v12" />

</svg>

<p>Перетащите фото сюда или кликните для выбора</p>

<p class="text-sm text-gray-500">

JPG, PNG или WebP. Максимум 10MB

</p>

</div>

</div>

<!-- Прогресс загрузки -->

<div v-if="uploadProgress > 0 && uploadProgress < 100" class="upload-progress">

<div class="progress-bar">

<div class="progress-fill" :style="{ width: uploadProgress + '%' }"></div>

</div>

<p>Загрузка: {{ uploadProgress }}%</p>

</div>

<!-- Галерея фото -->

<div class="photo-gallery">

<draggable

v-model="photos"

@end="handleReorder"

item-key="id"

class="gallery-grid"

>

<template #item="{ element }">

<div class="photo-item" :class="{ 'primary': element.is\_primary }">

<img

:src="getPhotoUrl(element, 'medium')"

:alt="element.original\_name"

loading="lazy"

>

<div class="photo-actions">

<button

v-if="!element.is\_primary"

@click="setPrimary(element.id)"

class="btn-primary"

title="Сделать главным"

>

⭐

</button>

<button

@click="deletePhoto(element.id)"

class="btn-delete"

title="Удалить"

>

🗑️

</button>

</div>

<div v-if="element.is\_primary" class="primary-badge">

Главное фото

</div>

</div>

</template>

</draggable>

</div>

<!-- Сообщения об ошибках -->

<div v-if="errors.length" class="errors">

<div v-for="error in errors" :key="error" class="error-message">

{{ error }}

</div>

</div>

</div>

</template>

<script setup>

import { ref, computed } from 'vue'

import { useForm, router } from '@inertiajs/vue3'

import draggable from 'vuedraggable'

import axios from 'axios'

const props = defineProps({

master: Object,

mediaFiles: Array

})

// Состояние

const photos = ref(props.mediaFiles || [])

const isDragging = ref(false)

const uploadProgress = ref(0)

const errors = ref([])

// Получение URL фото

const getPhotoUrl = (photo, version = 'medium') => {

const imageVersion = photo.versions?.find(v =>

v.version\_type === version && v.format === 'jpeg'

)

return imageVersion ? `/storage/${imageVersion.file\_path}` : '#'

}

// Обработка выбора файлов

const handleFileSelect = (event) => {

const files = Array.from(event.target.files)

uploadFiles(files)

}

// Обработка drag & drop

const handleDrop = (event) => {

isDragging.value = false

const files = Array.from(event.dataTransfer.files)

uploadFiles(files)

}

// Загрузка файлов

const uploadFiles = async (files) => {

errors.value = []

for (const file of files) {

if (photos.value.length >= 10) {

errors.value.push('Достигнут лимит в 10 фото')

break

}

// Проверка типа

if (!file.type.startsWith('image/')) {

errors.value.push(`${file.name} не является изображением`)

continue

}

// Проверка размера

if (file.size > 10 \* 1024 \* 1024) {

errors.value.push(`${file.name} превышает лимит в 10MB`)

continue

}

// Загрузка

const formData = new FormData()

formData.append('photo', file)

try {

const response = await axios.post(

`/masters/${props.master.id}/media/upload`,

formData,

{

headers: {

'Content-Type': 'multipart/form-data'

},

onUploadProgress: (progressEvent) => {

uploadProgress.value = Math.round(

(progressEvent.loaded \* 100) / progressEvent.total

)

}

}

)

if (response.data.success) {

photos.value.push(response.data.media)

}

} catch (error) {

errors.value.push(

error.response?.data?.message || 'Ошибка загрузки'

)

}

}

uploadProgress.value = 0

}

// Изменение порядка

const handleReorder = async () => {

const photoIds = photos.value.map(p => p.id)

try {

await axios.post(`/masters/${props.master.id}/media/reorder`, {

photo\_ids: photoIds

})

} catch (error) {

console.error('Ошибка изменения порядка:', error)

}

}

// Установка главного фото

const setPrimary = async (photoId) => {

try {

const response = await axios.post(

`/masters/${props.master.id}/media/${photoId}/set-primary`

)

if (response.data.success) {

photos.value.forEach(p => {

p.is\_primary = p.id === photoId

})

}

} catch (error) {

errors.value.push('Ошибка установки главного фото')

}

}

// Удаление фото

const deletePhoto = async (photoId) => {

if (!confirm('Удалить это фото?')) return

try {

const response = await axios.delete(

`/masters/${props.master.id}/media/${photoId}`

)

if (response.data.success) {

photos.value = photos.value.filter(p => p.id !== photoId)

}

} catch (error) {

errors.value.push('Ошибка удаления фото')

}

}

</script>

<style scoped>

.master-media-page {

max-width: 1200px;

margin: 0 auto;

padding: 2rem;

}

.photo-counter {

margin-bottom: 1rem;

font-size: 1.1rem;

color: #666;

}

.upload-zone {

border: 2px dashed #cbd5e0;

border-radius: 0.5rem;

padding: 3rem;

text-align: center;

cursor: pointer;

transition: all 0.3s;

margin-bottom: 2rem;

}

.upload-zone:hover,

.upload-zone.dragging {

border-color: #4299e1;

background-color: #f7fafc;

}

.upload-icon {

width: 4rem;

height: 4rem;

margin: 0 auto 1rem;

color: #a0aec0;

}

.upload-progress {

margin: 1rem 0;

}

.progress-bar {

height: 8px;

background: #e2e8f0;

border-radius: 4px;

overflow: hidden;

}

.progress-fill {

height: 100%;

background: #4299e1;

transition: width 0.3s;

}

.gallery-grid {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));

gap: 1rem;

}

.photo-item {

position: relative;

border-radius: 0.5rem;

overflow: hidden;

box-shadow: 0 2px 4px rgba(0,0,0,0.1);

}

.photo-item.primary {

box-shadow: 0 0 0 3px #4299e1;

}

.photo-item img {

width: 100%;

height: 200px;

object-fit: cover;

}

.photo-actions {

position: absolute;

top: 0.5rem;

right: 0.5rem;

display: flex;

gap: 0.5rem;

opacity: 0;

transition: opacity 0.3s;

}

.photo-item:hover .photo-actions {

opacity: 1;

}

.photo-actions button {

padding: 0.25rem 0.5rem;

background: white;

border-radius: 0.25rem;

border: 1px solid #e2e8f0;

cursor: pointer;

}

.primary-badge {

position: absolute;

bottom: 0;

left: 0;

right: 0;

background: #4299e1;

color: white;

padding: 0.25rem;

text-align: center;

font-size: 0.875rem;

}

.errors {

margin-top: 1rem;

}

.error-message {

padding: 0.75rem;

background: #fee;

color: #c53030;

border-radius: 0.25rem;

margin-bottom: 0.5rem;

}

</style>

**ЭТАП 8: НАСТРОЙКА ОЧЕРЕДЕЙ (15 минут)**

**Шаг 8.1: Создание Job для обработки изображений**

**Создай файл:** app/Jobs/ProcessUploadedImage.php

php

<?php

namespace App\Jobs;

use App\Models\MediaFile;

use App\Services\Media\ImageOptimizationService;

use Illuminate\Bus\Queueable;

use Illuminate\Contracts\Queue\ShouldQueue;

use Illuminate\Foundation\Bus\Dispatchable;

use Illuminate\Queue\InteractsWithQueue;

use Illuminate\Queue\SerializesModels;

class ProcessUploadedImage implements ShouldQueue

{

use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

public $timeout = 120;

public $tries = 3;

public function \_\_construct(

private int $mediaFileId

) {}

public function handle(ImageOptimizationService $optimizer): void

{

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

*// Обработка версий*

$versions = $optimizer->processImage($mediaFile->file\_path);

*// Сохранение в БД*

foreach ($versions as $versionType => $formats) {

foreach ($formats as $format => $data) {

$mediaFile->versions()->create([

'version\_type' => $versionType,

'format' => $format === 'jpeg' ? 'jpeg' : 'webp',

'file\_path' => $data['path'],

'file\_size' => $data['size'],

'width' => $data['width'],

'height' => $data['height']

]);

}

}

}

}

**Шаг 8.2: Настройка конфигурации очередей**

**Обнови файл:** config/queue.php

php

'connections' => [

*// ... существующие настройки*

'media' => [

'driver' => 'database',

'table' => 'jobs',

'queue' => 'media',

'retry\_after' => 300,

],

],

**ЭТАП 9: СОЗДАНИЕ КОМАНД ДЛЯ ОБСЛУЖИВАНИЯ (20 минут)**

**Шаг 9.1: Команда очистки временных файлов**

**Создай команду:**

bash

php artisan make:command CleanTempMedia

**Содержимое файла:** app/Console/Commands/CleanTempMedia.php

php

<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

use Illuminate\Support\Facades\Storage;

use Carbon\Carbon;

class CleanTempMedia extends Command

{

protected $signature = 'media:clean-temp';

protected $description = 'Очистка временных медиа-файлов';

public function handle(): int

{

$tempPath = storage\_path('app/temp');

$deletedCount = 0;

if (!is\_dir($tempPath)) {

$this->info('Временная папка не существует');

return 0;

}

$files = glob($tempPath . '/\*');

foreach ($files as $file) {

*// Удаляем файлы старше 24 часов*

if (filemtime($file) < time() - 86400) {

unlink($file);

$deletedCount++;

}

}

$this->info("Удалено временных файлов: {$deletedCount}");

return 0;

}

}

**Шаг 9.2: Команда проверки целостности**

**Создай команду:**

bash

php artisan make:command MediaHealthCheck

**Содержимое файла:** app/Console/Commands/MediaHealthCheck.php

php

<?php

namespace App\Console\Commands;

use App\Models\MediaFile;

use Illuminate\Console\Command;

use Illuminate\Support\Facades\Storage;

class MediaHealthCheck extends Command

{

protected $signature = 'media:health-check';

protected $description = 'Проверка целостности медиа-файлов';

public function handle(): int

{

$this->info('Проверка медиа-файлов...');

$missingFiles = [];

$corruptedFiles = [];

MediaFile::chunk(100, function ($mediaFiles) use (&$missingFiles, &$corruptedFiles) {

foreach ($mediaFiles as $mediaFile) {

$path = storage\_path('app/' . $mediaFile->file\_path);

*// Проверка существования*

if (!file\_exists($path)) {

$missingFiles[] = $mediaFile->id;

continue;

}

*// Проверка хеша*

$currentHash = md5\_file($path);

if ($currentHash !== $mediaFile->file\_hash) {

$corruptedFiles[] = $mediaFile->id;

}

}

});

if (count($missingFiles) > 0) {

$this->error('Отсутствующие файлы: ' . implode(', ', $missingFiles));

}

if (count($corruptedFiles) > 0) {

$this->error('Поврежденные файлы: ' . implode(', ', $corruptedFiles));

}

if (count($missingFiles) === 0 && count($corruptedFiles) === 0) {

$this->info('Все файлы в порядке!');

}

return 0;

}

}

**ЭТАП 10: НАСТРОЙКА CRON И ФИНАЛЬНАЯ ПРОВЕРКА (15 минут)**

**Шаг 10.1: Настройка расписания**

**Обнови файл:** app/Console/Kernel.php

php

protected function schedule(Schedule $schedule)

{

*// Очистка временных файлов каждый час*

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

*// Проверка целостности два раза в день*

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

->twiceDaily(1, 13)

->emailOutputTo('admin@example.com');

*// Обработка очереди медиа*

$schedule->command('queue:work --queue=media --max-jobs=10')

->everyMinute()

->withoutOverlapping();

}

**Шаг 10.2: Финальная проверка**

bash

*# 1. Проверь миграции*

php artisan migrate:status

*# 2. Очисти кеш*

php artisan cache:clear

php artisan config:clear

php artisan route:clear

*# 3. Создай символическую ссылку для storage*

php artisan storage:link

*# 4. Проверь права доступа*

chmod -R 755 storage/app/media

chmod -R 755 storage/app/temp

*# 5. Запусти тесты загрузки*

php artisan tinker

>>> $file = \Illuminate\Http\UploadedFile::fake()->image('test.jpg', 800, 600);

>>> app(\App\Services\Media\MediaService::class)->uploadImage($file, 1);

**ИТОГОВЫЙ ЧЕКЛИСТ**

✅ **База данных:**

* Созданы таблицы media\_files и image\_versions
* Настроены индексы и внешние ключи
* Добавлено ограничение на 10 фото

✅ **Файловая система:**

* Создана структура папок с хеш-директориями
* Настроены права доступа
* Создана символическая ссылка storage

✅ **Backend:**

* Реализован сервис FilePathService
* Реализован сервис ImageOptimizationService
* Реализован сервис MediaService
* Создан контроллер MasterMediaController
* Настроены роуты
* Добавлены Jobs для очередей

✅ **Frontend:**

* Создан Vue компонент для управления медиа
* Реализован drag & drop
* Добавлена сортировка фото
* Реализован выбор главного фото

✅ **Оптимизация:**

* Генерируются 3 размера + оригинал
* Создаются WebP версии
* Настроено кеширование
* Добавлен lazy loading

✅ **Безопасность:**

* Используются UUID для файлов
* Реализована хеш-структура папок
* Добавлен rate limiting
* Настроена валидация файлов

✅ **Обслуживание:**

* Команда очистки временных файлов
* Команда проверки целостности
* Настроен cron для автоматизации
* Добавлено логирование ошибок

**КОМАНДА ДЛЯ СТАРТА**

После выполнения всех шагов выполни:

bash

*# Финальная команда для запуска*

php artisan migrate && \

php artisan storage:link && \

php artisan queue:work --queue=media &