ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

К ЗАЩИТЕ ДОПУЩЕН

Директор ОГБПОУ «ТЭПК»

\_\_\_\_\_\_\_\_\_\_\_\_ / О.Н. Пояркова

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА  
МЕДИА-СЕРВИСА «WEPICSYNC»**

ДИПЛОМНЫЙ ПРОЕКТ

*Специальность 09.02.07 Информационные системы и программирование*

Выполнил:

Студент IV курса группы 1912с

\_\_\_\_\_\_\_\_\_\_\_\_\_ / Матохнюк А.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Проверил:

Руководитель ДП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Уляхин В.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**Оглавление**

[Введение 4](#_Toc201233283)

[Глава 1. Техническое задание 5](#_Toc201233284)

[Предметная область 5](#_Toc201233285)

[Глоссарий 5](#_Toc201233286)

[Требования к функциональности 5](#_Toc201233287)

[Ролевая модель 6](#_Toc201233288)

[Диаграмма прецедентов 7](#_Toc201233289)

[Диаграммы деятельностей 9](#_Toc201233290)

[Диаграммы последовательностей 11](#_Toc201233291)

[Требования к API 20](#_Toc201233292)

[Требования к сайту 31](#_Toc201233293)

[Требования к мобильному приложению 33](#_Toc201233294)

[Глава 2. Проектирование и разработка базы данных 36](#_Toc201233295)

[Концептуальная модель данных 36](#_Toc201233296)

[Логическая модель данных 36](#_Toc201233297)

[Физическая модель данных 37](#_Toc201233298)

[Глава 3. Проектирование и разработка API сервиса 39](#_Toc201233299)

[Зависимости проекта 39](#_Toc201233300)

[Структура проекта 40](#_Toc201233301)

[Маршруты 40](#_Toc201233302)

[Конфигурационный файл 43](#_Toc201233303)

[Модели 43](#_Toc201233304)

[Контроллеры 47](#_Toc201233305)

[Посредник 51](#_Toc201233306)

[Глава 4. Проектирование и разработка веб-сайта 67](#_Toc201233307)

[Зависимости проекта 67](#_Toc201233308)

[Структура проекта 67](#_Toc201233309)

[Базовые элементы 67](#_Toc201233310)

[Компоненты 69](#_Toc201233311)

[Глобальные хранилища 80](#_Toc201233312)

[Функции-помощники 83](#_Toc201233313)

[Маршрутизатор 85](#_Toc201233314)

[Конфигурационный файл 86](#_Toc201233315)

[Представления 86](#_Toc201233316)

[Глава 5. Проектирование и разработка мобильного приложения 90](#_Toc201233317)

[Зависимости проекта 90](#_Toc201233318)

[Структура проекта 90](#_Toc201233319)

[Реализация моделей 91](#_Toc201233320)

[Реализация утилит 96](#_Toc201233321)

[Реализация представлений 106](#_Toc201233322)

[Глава 6. Тестирование API 124](#_Toc201233323)

[401 Unauthorized 124](#_Toc201233324)

[422 Unprocessable Content 125](#_Toc201233325)

[403 Forbidden 126](#_Toc201233326)

[204 No Content 126](#_Toc201233327)

[200 OK 127](#_Toc201233328)

[201 Created 128](#_Toc201233329)

[Вывод 128](#_Toc201233330)

[Заключение 129](#_Toc201233331)

[Список использованных источников 130](#_Toc201233332)

# Введение

С ростом популярности смартфонов и увеличения качества съёмки (например, снимков в высоком разрешении), объем данных, которые пользователи, храня на своих устройствах, постоянно растет. Это приводит к необходимости освобождать место на устройствах, что часто сопровождается удалением важных или ценных файлов.

Разрабатываемый сервис предлагает удобное и безопасное облачное хранилище, которое позволяет пользователям освободить память на своих устройствах, сохраняя при этом доступ к своим медиафайлам в любое время и в любом месте.

Хранение данных на удалённом сервере защищает файлы от потери в случае повреждения или утери устройства.

Возможность делиться фотографиями и создавать альбомы с ограниченным доступом делает сервис полезным не только для личного использования, но и для совместной работы или обмена медиа с семьёй, друзьями и коллегами.

Пользователи могут ставить теги на своё медиа, позволяя категорировать и упрощая поиск в своих коллекциях нужное фото или видео.

Цель дипломного проекта является проектирование и разработка медиа-сервис «WepicSync».

Для достижения заданной цели необходимо выполнить следующие задачи:

1. Спроектировать и разработать базу данных сервиса для MySQL.
2. Спроектировать и разработать сервер часть сервиса на фреймворке Laravel.
3. Спроектировать и разработать веб-сайт сервиса на фреймворке Vue.js.
4. Спроектировать и разработать мобильное приложение синхронизации медиа на MAUI.
5. Разработать автоматизированные тесты API и провести тестирование.

# Глава 1. Техническое задание

## Предметная область

Разрабатываемый сервис предназначен для удобного хранения, управления и получения пользователями своих картинок и видео на удалённом сервере. Основная цель сервиса – обеспечить надёжное хранение медиафайлов путём повышения числа копий. Пользователи смогут загружать на сервер фотографии и снятые видео с телефона, скаченные коллекции медиафайлов через удобный и интуитивно понятный интерфейс, организовывать их в альбомы, добавлять теги для упрощения поиска. Сервис также предоставит возможность делиться фотографиями с другими пользователями или выставлять на публику посредством приглашения через ссылки.

Сервис ориентирован:

1. Для лица, который хочет хранить медиафайлы на своём сервере, не платя подписки и не делясь со сторонними компаниями своими личными фотографиями и видео.
2. Для команды/организации, которая хочет поделится большим количеством медиафайлов для публики или конкретным лицам в удобном виде.

## Глоссарий

|  |  |
| --- | --- |
| **Понятие** | **Определение понятия** |
| API | API (интерфейс программирования приложений) – это набор определенных правил и соглашений, который используется для взаимодействия между различными программными компонентами. API определяет, как различные части программного обеспечения могут взаимодействовать друг с другом, обычно путем обмена данными и выполнения определенных операций. |
| Postman | Postman – это инструмент для тестирования и разработки API. Он предоставляет удобный интерфейс для отправки HTTP запросов к API, а также для анализа ответов. |
| Draw.io | Это бесплатный онлайн инструмент для создания диаграмм и схем. Он предоставляет широкий набор инструментов и элементов, которые позволяют пользователям создавать различные типы диаграмм, такие как организационные диаграммы, блок-схемы, ER-диаграммы, сетевые диаграммы, диаграммы потока процессов и многое другое. |
| CRUD | CRUD – акроним, обозначающий четыре базовые функции, используемые при работе с данными: создание, чтение, модификация, удаление. |

## Требования к функциональности

В системе есть следующие роли:

1. Гость — вход, регистрация, получение публичных настроек.
2. Пользователь — управление над личными альбомами.
3. Администратор — управление учётными записями, управление серверными альбома.

Гость может:

1. Входить и регистрироваться. Аутентификация производится по токеном, генерируемым по входу и регистрации и отзываемых при выходе. Вход на другом устройстве не должна вызывать выход из аккаунта на всех устройствах.
2. Просматривать публичные альбомы и их медиа.
3. Просматривать часть альбома в приглашении.
4. Получить следующую информацию о сервере: разрешённые типы файлов (MIME), размеры превью, квота на хранилище для новых пользователя, статус разрешения загрузки (который меняется при достижении лимита занятого места на диске).

Пользователь может:

1. Создать личный альбом.
2. Загрузить в личные альбомы медиа и просмотреть их. У каждого пользователя есть квота на размер всех загруженных медиа (регулирующаяся администратором и общая для всех).
3. Создать и назначать несколько тегов на каждую свою медиа.
4. Установить несколько реакций на каждую доступную медиа.
5. Создать код приглашения на свой альбом. Он может сделать альбом публичным или установить лимит вступлений и дату просрочки или минут жизни.
6. Получить доступ к чужому альбому путём перехода по коду приглашения (ссылке) от владельца альбома.
7. Владелец альбома может отозвать выданные доступы и коды приглашения. Пользователь, который имеет доступ, может отказаться от доступа.
8. Владелец может переиндексировать альбом на поиск нового содержимого.

Администратор может:

1. Создать серверные альбомы и управлять всеми альбомами и медиа.
2. Администратор может создать учётные записи и задавать квоты для каждого.
3. Администратор может менять следующие предустановки: разрешённые типы файлов (MIME), список разрешённых размеров превью, квота на хранилище для регистрируемого пользователя и лимит занятого на диске место (в процентах) при котором будет отключена загрузка на сервер.

При настроенном планировщике сервер каждые 10 минут проверяет и кеширует размер диска и размер свободного места, а также каждые день индексирует все серверные альбомы.

## Ролевая модель

Исходя из основных требований функциональности системы можно построить таблицу, разделяя роли и функциональность (см. таблица 1):

Таблица 1 – Ролевая модель

| **Функционал** | | **Администратор** | **Пользователь** | **Гость** |
| --- | --- | --- | --- | --- |
| Пользователи | Регистрация | нет | нет | да |
| Вход | нет | нет | да |
| Выход | да | | нет |
| Просмотр | да | да (себя) |
| Изменение |
| Альбомы | CRUD | да | да (свои) | нет |
| Просмотр публичного | да | | |
| Приглашения | Вступление в альбом | да (на чужие альбомы) | | нет |
| Создание | да (на свои альбомы) | |
| Удаление |
| Медиа | CRUD | да | да (свои) | нет |
| Просмотр информации | да (доступные) | |
| Скачивание |
| Теги | Просмотр и создание | да | | нет |
| Добавление к медиа | да (к своим медиа) | |
| Удаление с медиа |
| Доступы на альбомы | Создание | да (на свои альбомы) | | нет |
| Удаление | да (на свои альбомы; от себя на чужих) | |
| Предустановки | Получение публичных | да | | |
| Реакции | CRUD | да | только чтение | |
| Добавление к медиа | да (к доступным медиа) | | нет |
| Удаление с медиа |

## Диаграмма прецедентов

Диаграмма прецедентов используется для моделирования функциональных требований системы и взаимодействия между пользователями и системой. Она помогает определить основные сценарии использования системы и её ключевые функции. Состоит из следующих ключевых элементов:

* *Акторы (человек)*: роли пользователей или внешних систем, взаимодействующих с системой.
* *Прецеденты (овалы)*: функциональные возможности или сценарии, которые система предоставляет.
* *Связи (непрерывные стрелки)*:взаимодействия между акторами и прецедентами.
* *Включение* и *расширение (пунктирные стрелки с inc и ext)*: отношения между прецедентами для повторного использования или добавления дополнительного поведения.

В системе выделено 4 актора (из них 3 как роли пользователей) и 9 подсистем прецедентов как общие объекты случаев использования и была разработана диаграмма прецедентов (см. рис. 1).

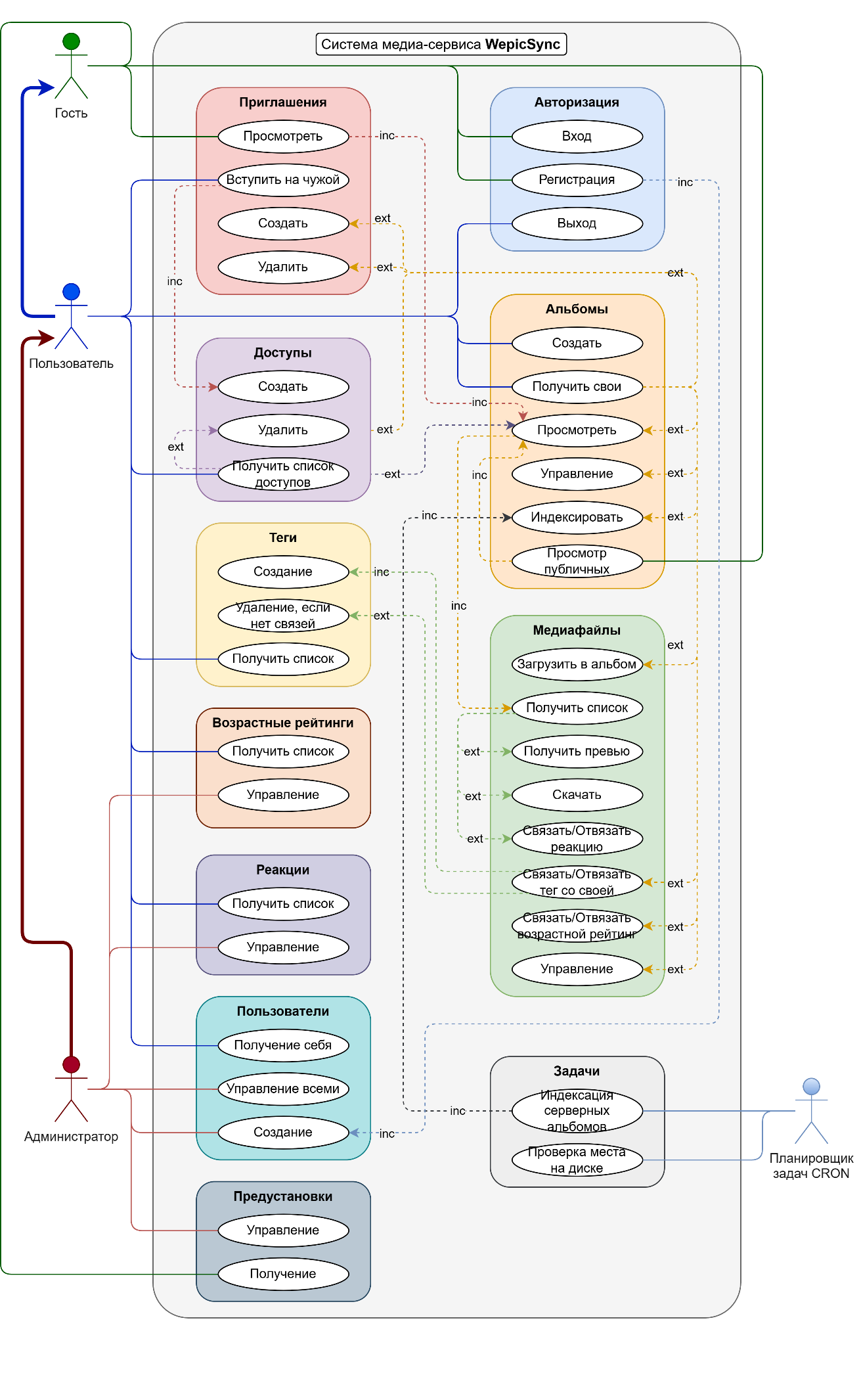


Рисунок 1 – Диаграмма прецедентов

## Диаграммы деятельностей

Диаграмма деятельностей используется для моделирования бизнес-процессов, рабочих потоков или алгоритмов. Она фокусируется на последовательности действий и логике их выполнения. Состоит из следующих ключевых элементов:

* *Действия (стянутый овал)*: шаги или задачи, выполняемые в процессе.
* *Потоки управления (стрелки)*: указывают порядок выполнения действий.
* *Ветвления* и *условия (ромбы)*: точки принятия решений.
* *Параллельные потоки (параллельные линии)*: разделение и синхронизация процессов.
* *Начало* и *завершение (круги)*: маркеры начала и конца процесса.

### Авторизационные функции

При входе неавторизированным должна быть доступна кнопка для входа в учётную запись или создание новой.

При входе с неправильным логином и/или паролём выводится что пользователь не найден. Если пользователь нашёлся — авторизовать пользователя. Подробнее на рисунке 2.

При регистрации, не заполоняя все требуемые поля или вводя не верные данные, выводятся все допущенные ошибки. Если прошли валидацию — сохраняем данные в БД и авторизируем пользователя. Подробнее на Рисунок 3.

|  |  |
| --- | --- |
| Рисунок 2 – Вход | Рисунок 3 – Регистрация |

После авторизации появляется опция выхода, которая удаляет токен из базы и переводит на страницу с формой входа (см. рис. 4Рисунок 4).

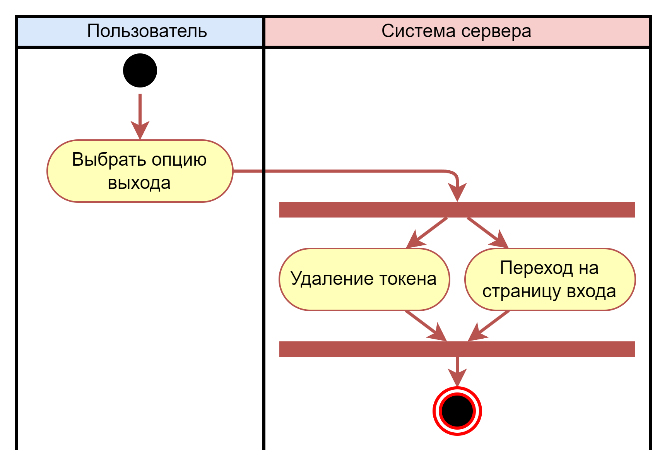


Рисунок 4 – Выход

### Альбомы и медиа

При входе авторизированным пользователя встречает вкладка со всеми медиа и вкладка альбомов. На вкладке альбомов, помимо открытия его медиа, можно создать приглашение на свой альбом — выводится форма, в которой можно сделать альбом публичным и получить ссылку или создать приглашение с определённым сроком и лимитом вступлений. Подробнее на рисунке 5.

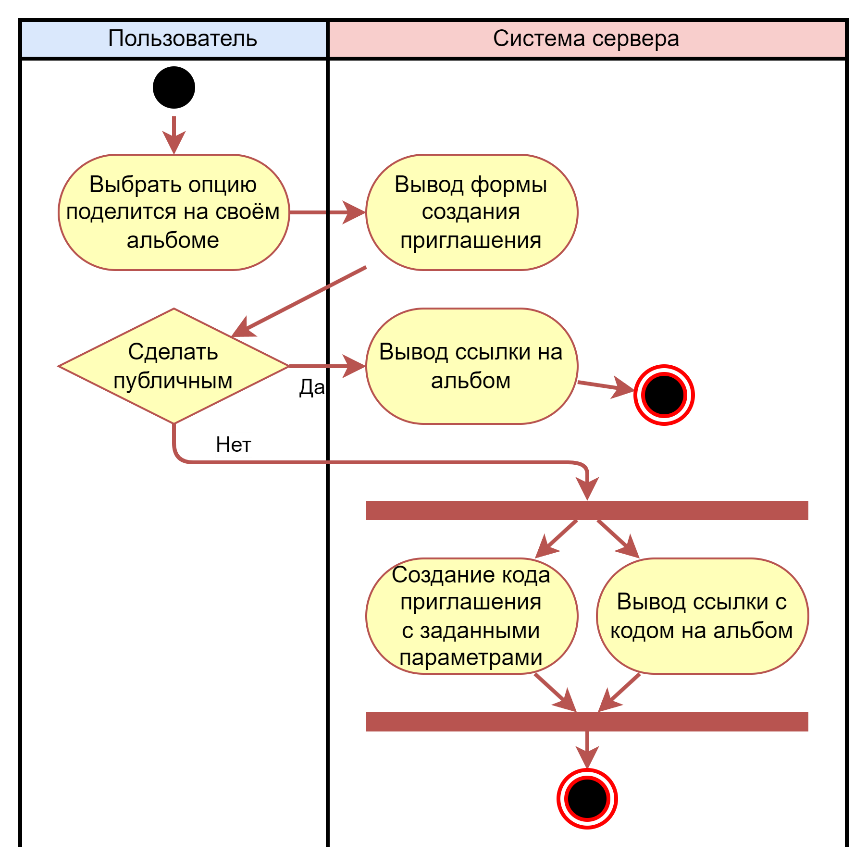
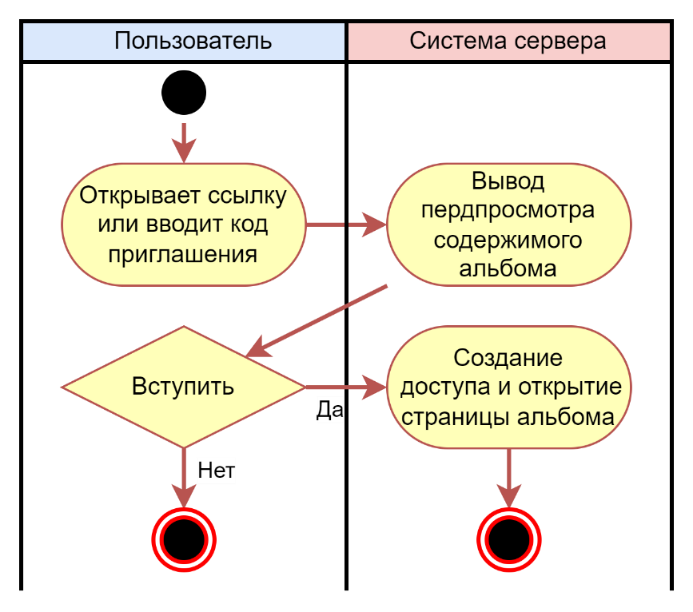


Рисунок 5 –Создание приглашения

Кодом приглашения можно вставить в поле код приглашения, который выводит превью-содержимое и имеются действия по вступлению (сохранение постоянного доступа). Подробнее на рисунке 6.



*Рисунок 6 – Получение постоянного доступа к чужому альбому*

## Диаграммы последовательностей

Диаграмма последовательностей визуализирует взаимодействие объектов в системе в рамках конкретного сценария. Она отражает последовательность сообщений, передаваемых между объектами, и их временную упорядоченность. Состоит из следующих ключевых элементов:

* *Объекты* и *акторы (прямоугольники сверху)*: участники взаимодействия.
* *Сообщения (стрелки)*: вызовы методов или передача данных между объектами.
* *Временная* *ось (вертикальная линия)*: показывает порядок событий под актором.
* *Активация (вертикальный прямоугольник на временной оси)*: период активности объекта при выполнении операции.
* *Фреймы (кадр с подписью)*: обёртки над элементами для описания конкретного случая или цикла.

### Общие диаграммы-дополнения

Общие диаграммы последовательности не являются самостоятельными и предназначены для дополнения к другим последовательностям для разгрузки повторяющихся элементов. Обычные диаграммы ссылаются на дополнительны посредством цветной активации по цвету выхода и припиской «(Общее: *<название ссылки>*)».

Валидация тела POST запроса осуществляется посредником, проверяя заданные правила на ключах (см. рис. 7). Эта диаграмма используется под названием «валид» для расширения других диаграмм.

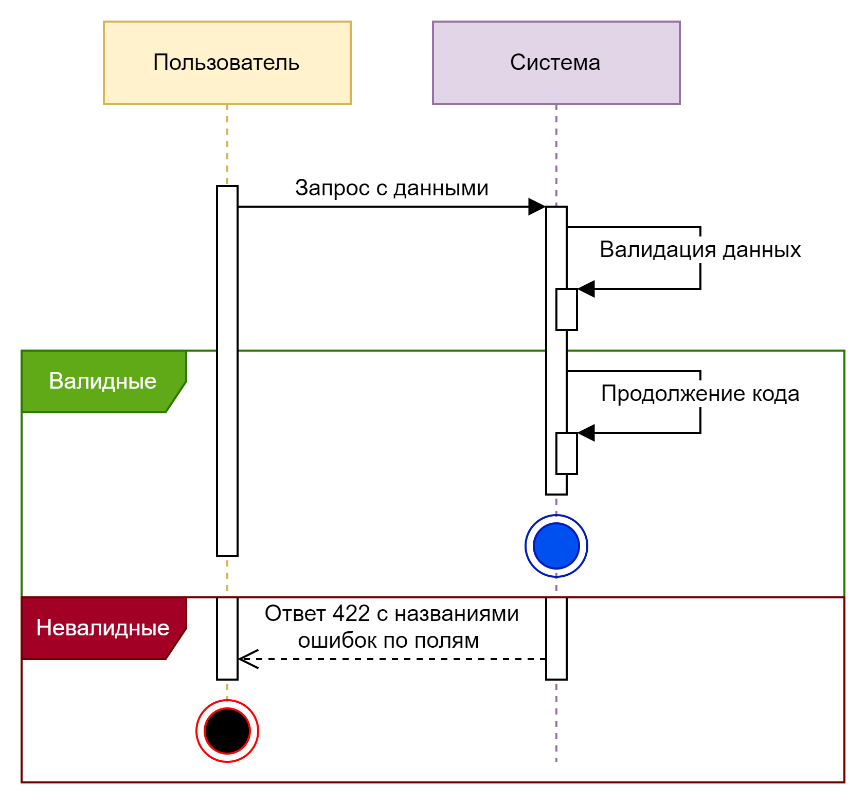


Рисунок 7 – Валидация тела запроса

Внесения в контекст кода модель пользователя и отброс неавторизированных осуществляется посредником, проверяя токен в заголовках запроса и запрашивая пользователя с этим ИД (см. рис. 8). Эта диаграмма используется под названием «токен» для расширения других диаграмм.

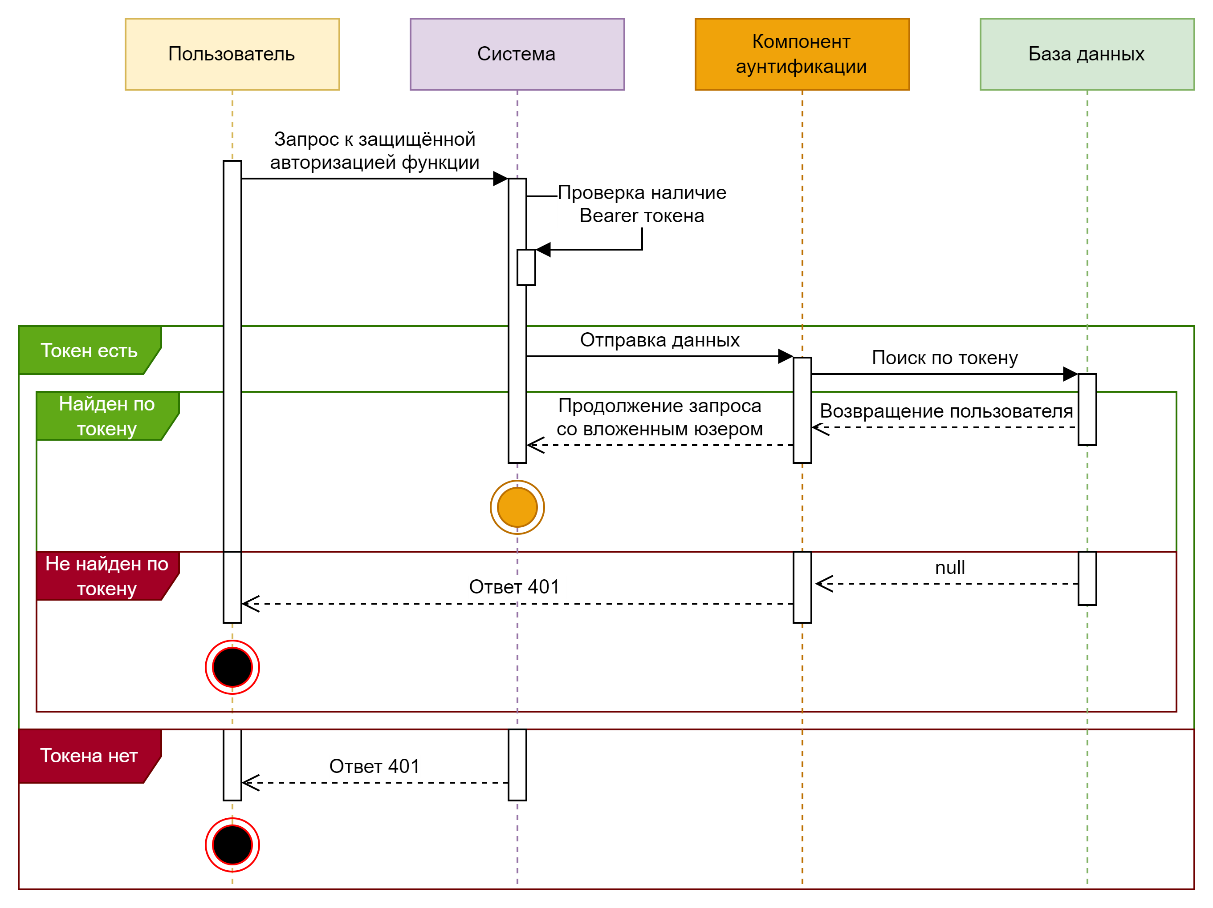


Рисунок 8 – Проверка токена и внесение в контекст пользователя

Внесения в контекст кода модель объекта по ИД и отброс если не модель не найдена осуществляется посредником, беря из роута ИД и запрашивая из БД её модель (см. рис. 9Рисунок 9). Эта диаграмма используется под названием «сущ» для расширения других диаграмм.

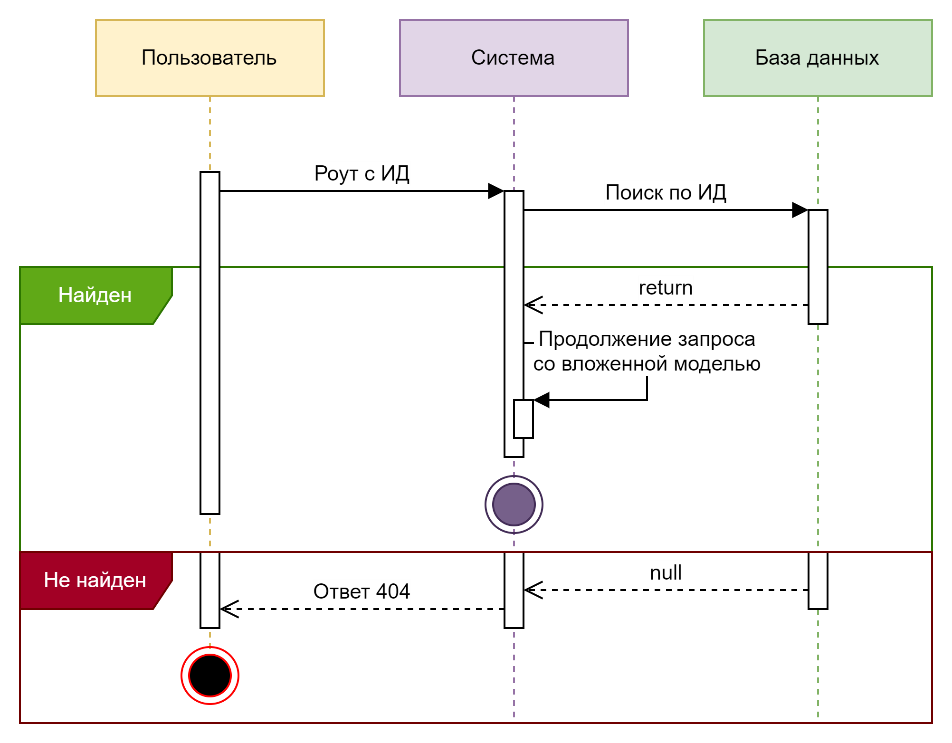


Рисунок 9 – Проверка существования объекта БД

Пропуск пользователя к данным альбома и к функциям над альбомом осуществляется посредником, проверяя соответствие ИД владельца альбома и ИД текущего пользователя, и если он не владелец, то проверяем в таблице доступов запись с этими ИД (см. рис. 10Рисунок 10). Эта диаграмма используется под названием «доступ» для расширения других диаграмм.

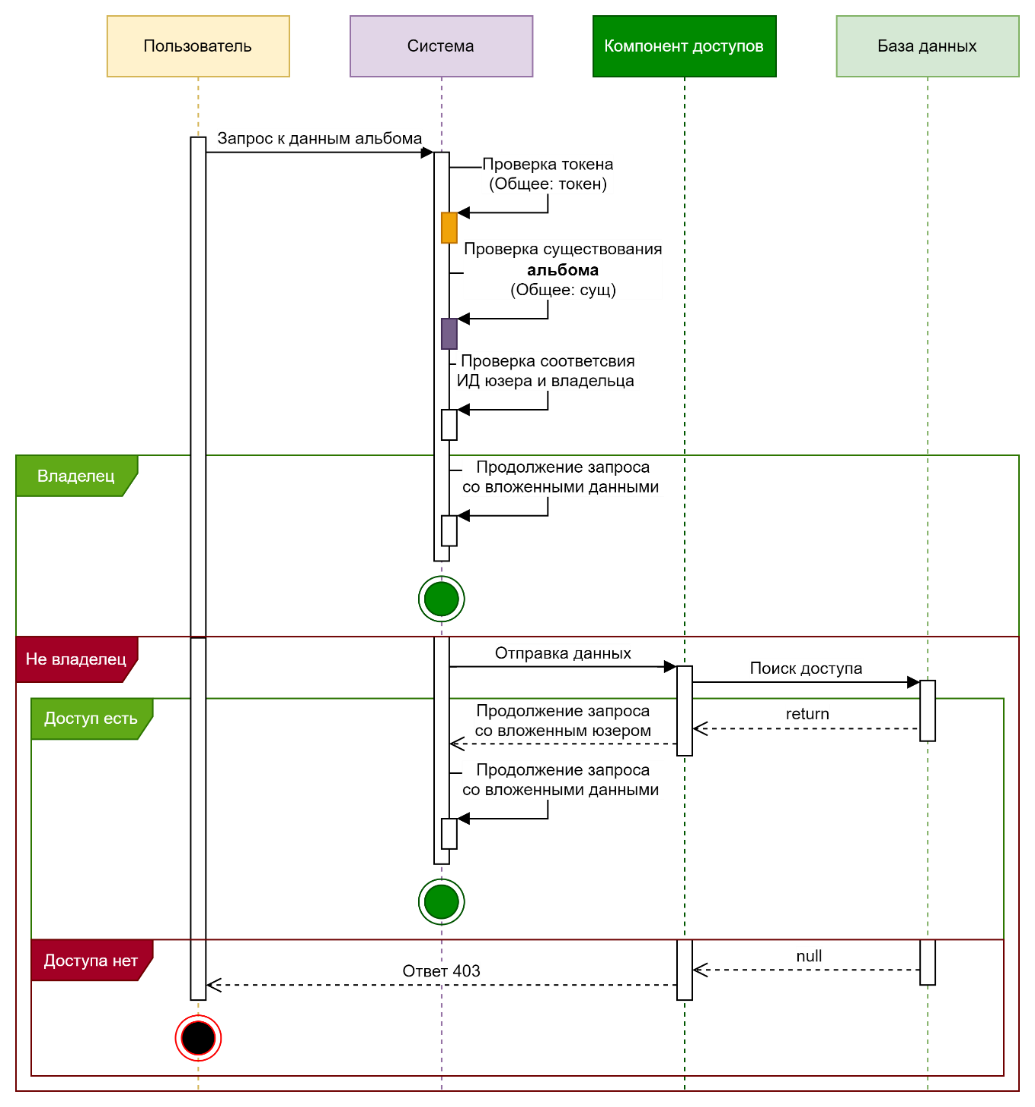


Рисунок 10 – Проверка доступа к альбому

### Авторизационные функции

Алгоритм регистрации новой учётной записи и последующая авторизация представлен на рисунке 11Рисунок 11. Производит валидацию тела запроса (вместе с проверкой существования в БД), которое выводит ошибки при неверных данных. В случае успешной валидации производится сохранение учётных данных в БД, генерация токена, сохранение токена и возвращается токен с другими данными пользователя.

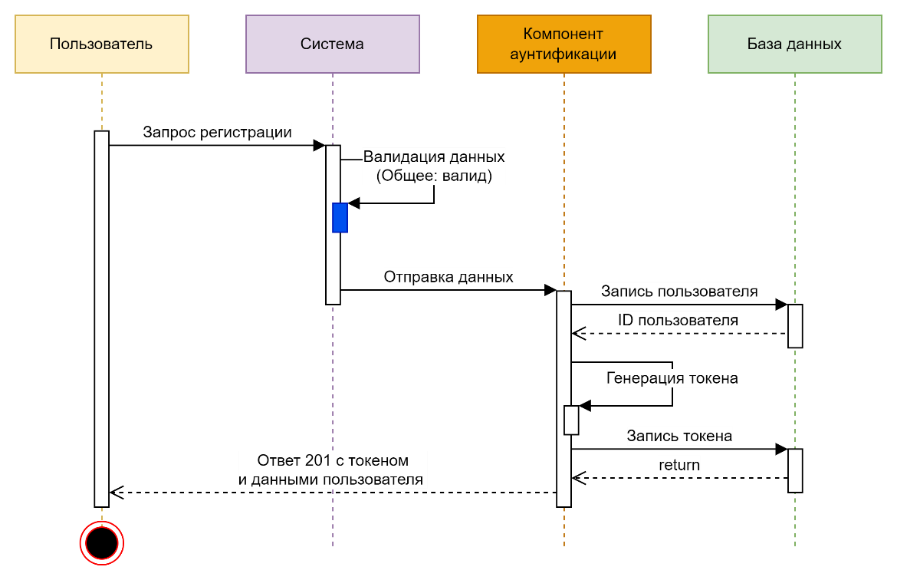


Рисунок 11 – Регистрация

Алгоритм входа по реквизитам представлен на рисунке 12. Производится валидация на то, что поля были переданы. При валидных данных производится поиск по логину в БД. Если найден по логину, то проверяем совпадение переданного пароля и хешированного пароля из БД. Если совпадает, то генерирует и сохраняет токен, выводит учётные данные и новый токен. Если не найден по логину, или пароли не совпадают — возвращает 401.

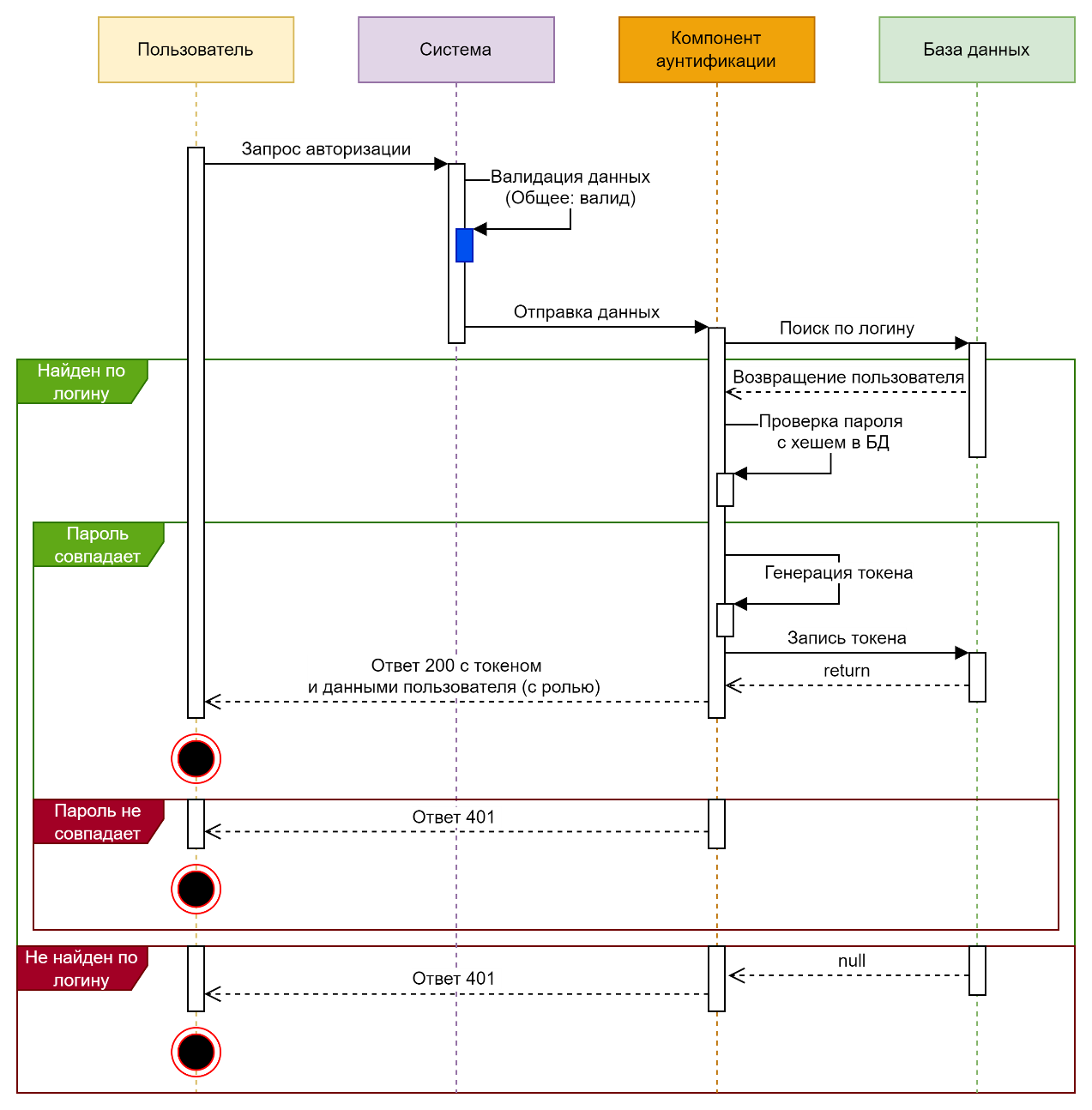


Рисунок 12 – Вход

Алгоритм выхода и удаление токена представлен на рисунке 13. Производится проверка что токен действителен, и в случае успеха — удаляет из БД и возвращает 204

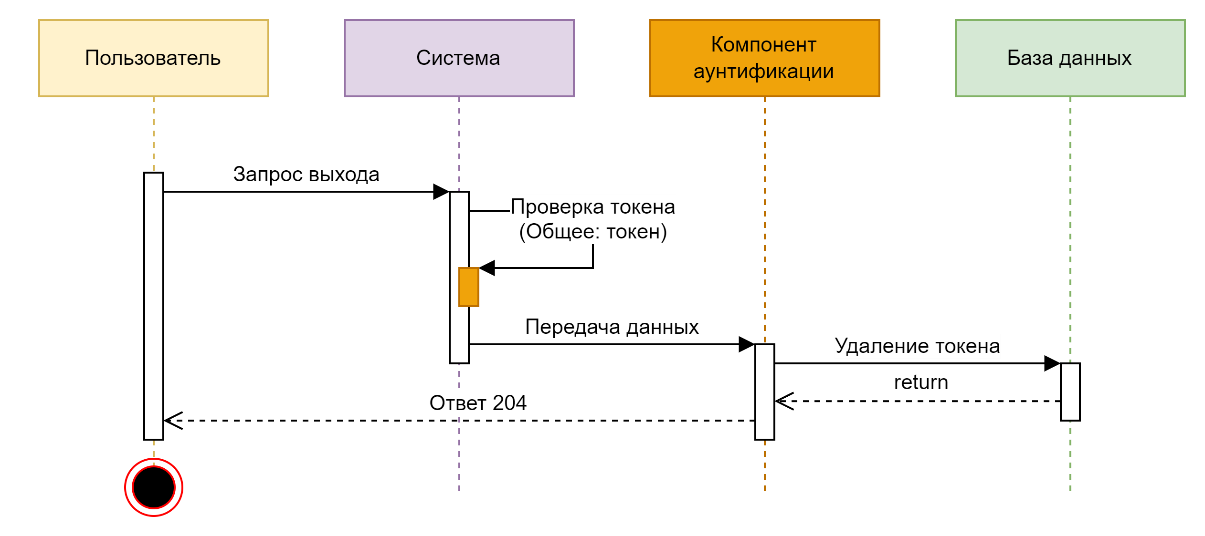


Рисунок 13 – Выход

### Работа с альбомами

Алгоритм создания нового личного альбома представлен на рисунке 14. Для создания альбома необходимо быть авторизированным, чтобы привязать его к владельцу-пользователю. Производится валидация данных и в случае успеха альбом создаётся в БД, возвращая его идентификационные данные для последующих действий пользователем.



Рисунок 14 – Создание альбома

Получение списка личных и доступных альбомов представлена на рисунке 15. Необходимо быть авторизованным. Запрашивается и выводится из БД личные альбомы и альбомы, в которых пользователь записан как имеющий доступ.

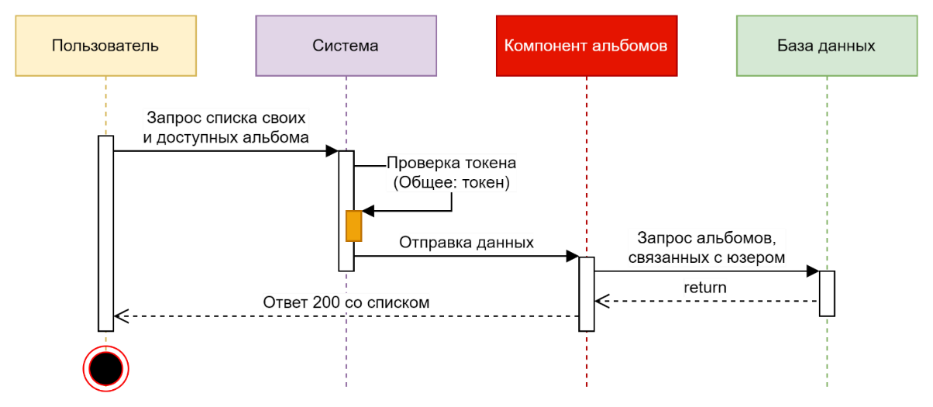


Рисунок 15 – Получение списка альбомов

Алгоритм обновления данных об альбоме представлен на рисунке 16. Проверятся доступ как владельца или как администратора. Изменение может включать себя переключение публичности альбома. При успешном обновление приходит 204.



Рисунок 16 – Изменение альбома

Алгоритм удаления альбома представлен на рисунке 17. Проверятся доступ как владельца или как администратора. Удаление производится в БД, каскадно удаляя все связанные данные, и в файловой системе, рекурсивно удаляя и вложенные файлы.

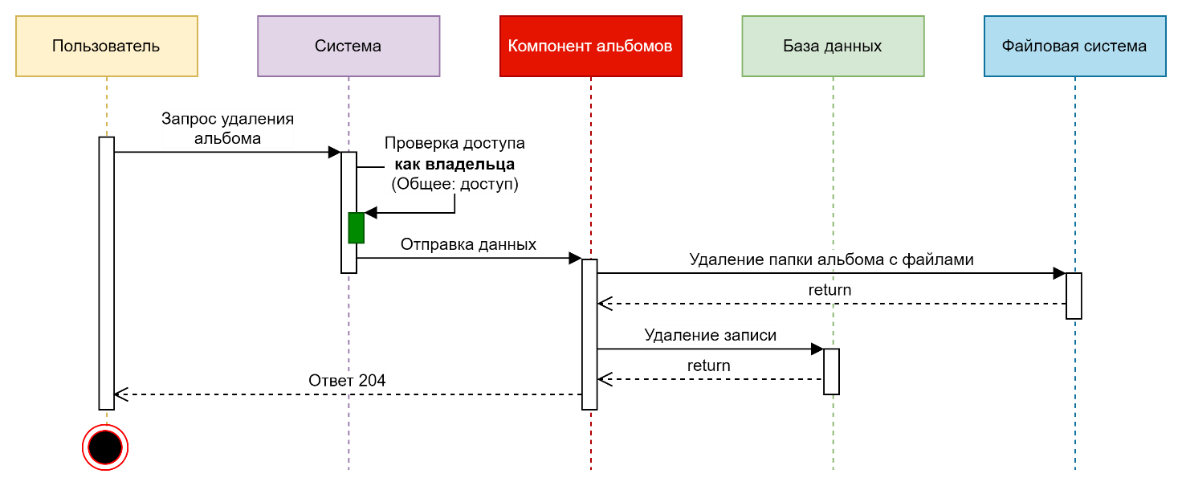


Рисунок 17 – Удаление альбома

### Приглашения на доступ к альбому

Алгоритм создания приглашения на свой альбом представлен на рисунке 18. Проверяется доступ как владельца и валидность отправленных данных. Генерирует и сохраняет в БД случайную строку как код приглашения, который может использоваться самостоятельно или быть в ссылке-приглашении. Возвращает сгенерированный код приглашения.

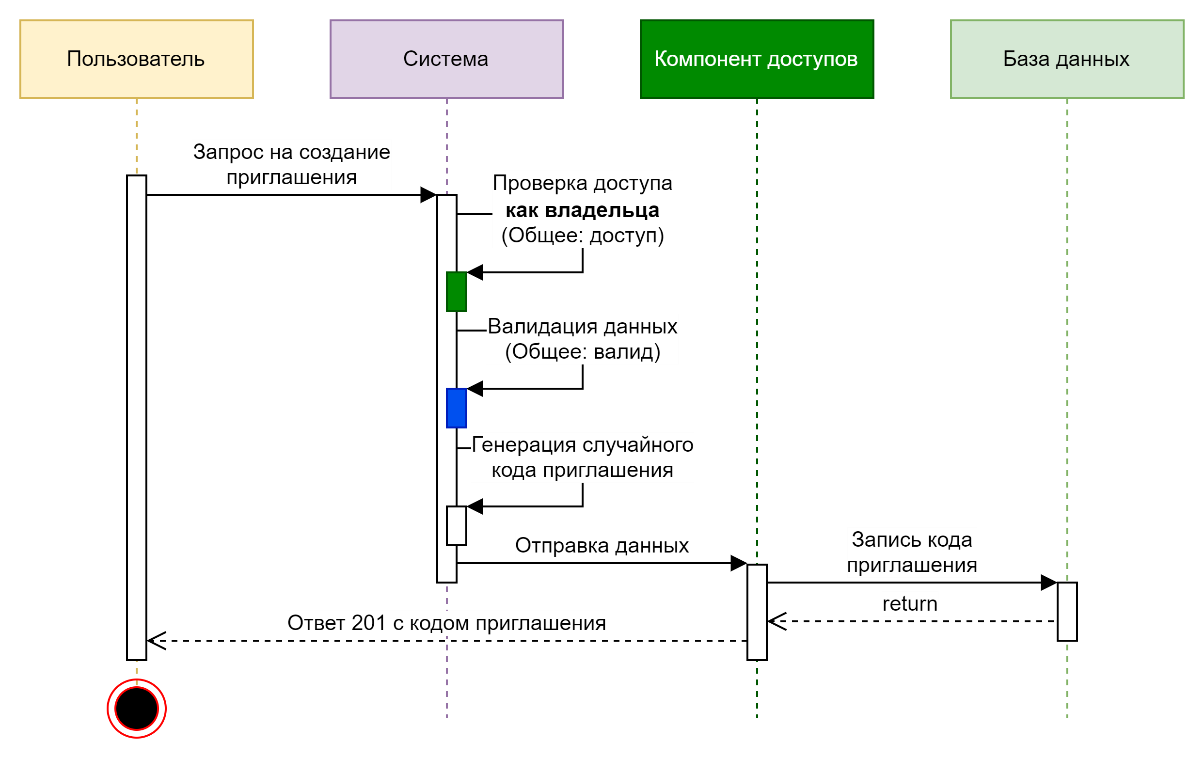


Рисунок 18 – Создание приглашения

Алгоритм получения доступа к чужому альбому представлен на рисунке 19. Пользователь передаёт код доступа, который проверяется в БД что он ещё не просрочен и лимит вступлений не достигнут. В случае успеха создаётся связь, которая даёт бессрочный доступ на чужой альбом и возвращается 204.

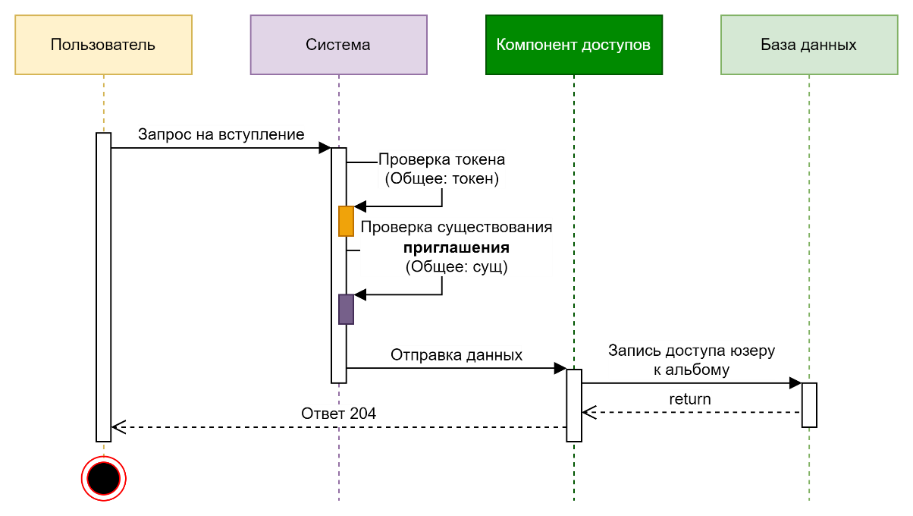


Рисунок 19 – Вступление на чужой альбом

### Работа с медиа

Алгоритм загрузки на сервер в свой альбом медиафайлы представлен на рисунке 20. Валидируется структура передаваемого массива. В случае валидности запрашиваются записи медиа из БД для сверки названий и хешей файлов. Если есть с похожим названием — переименовываем с счётчиком. Если есть с подобным хешем — пропускаем. Возвращает массив успешных и провальных/уже существующих (с идентификаторами) вставок.

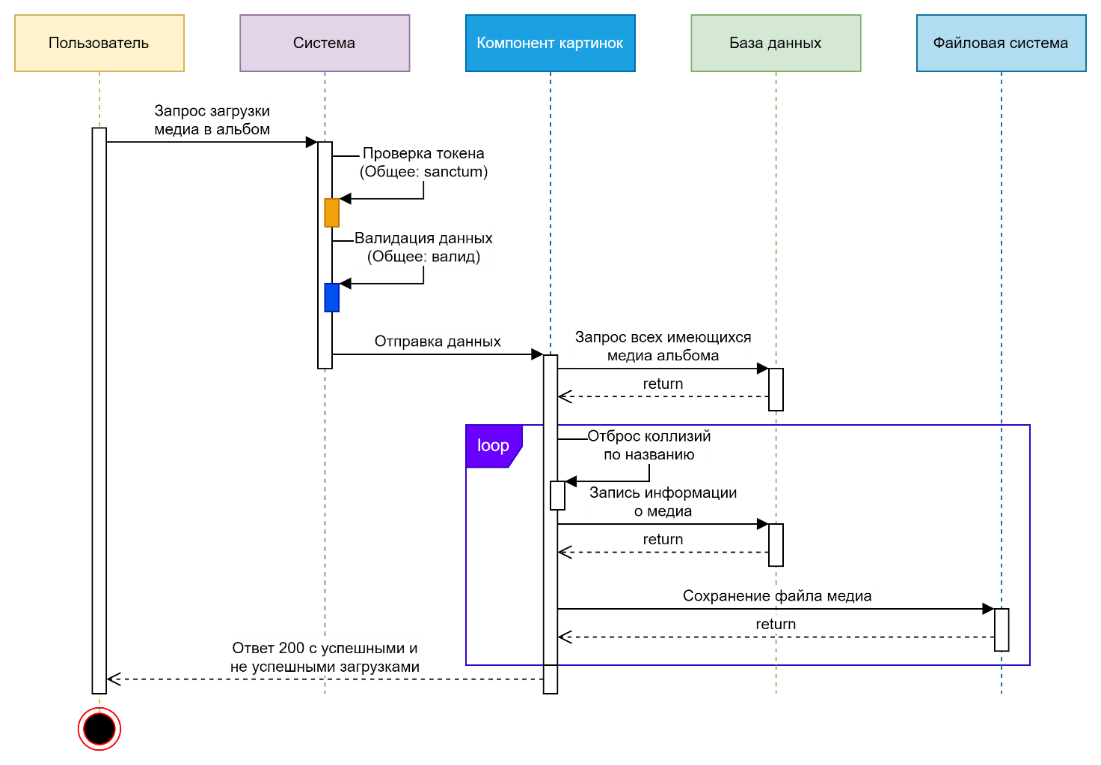


Рисунок 20 – Загрузка медиа в альбом

Алгоритм получение списка медиафайлов в альбоме представлен на рисунке 21. Проверяется доступ как читателя и возвращает из БД список медиа с идентификаторами, по которым уже можно получить файлы. Генерирует и добавляет в ответ сигнатуру доступа, если альбом не публичный. Сигнатура нужна т.к. передать в запросе к медиафайлу, находящийся в разметке, нельзя передать токен.

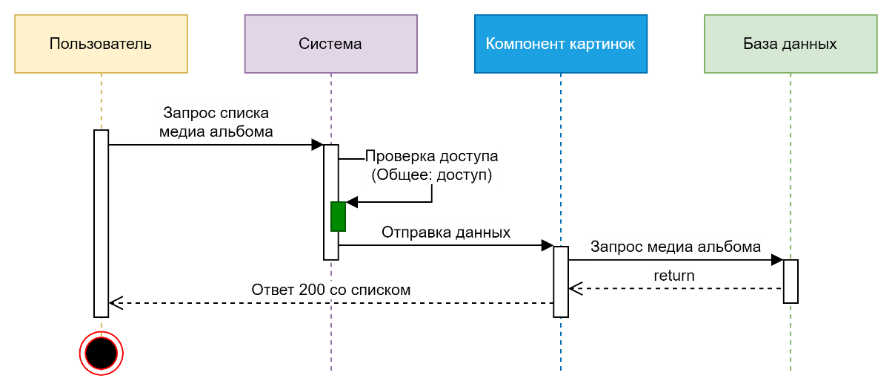


Рисунок 21 – Получение медиа альбома

Алгоритм скачивания с сервера медиафайла представлен на рисунке 22. Производится проверка что медиа существует в БД (получив таким образом его полный путь) и возвращается из файловой системы через полный путь.

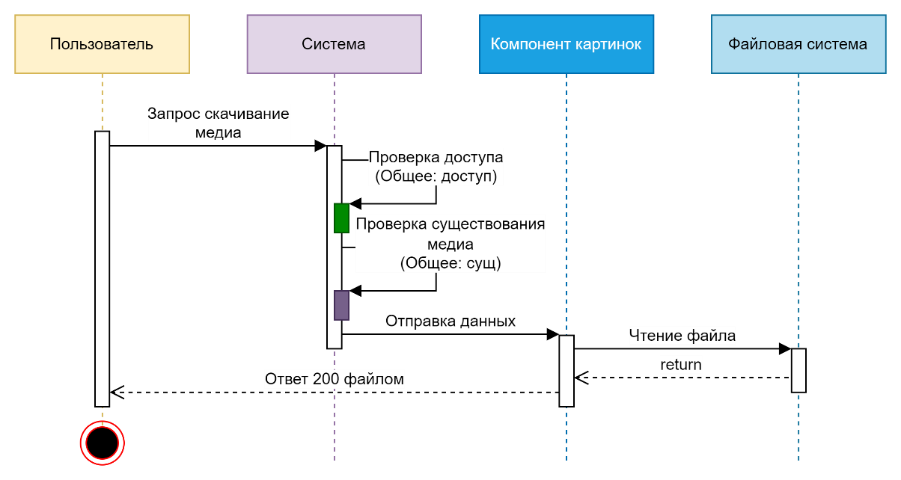


Рисунок 22 – Скачивание медиа

Удаление с сервера медиафайла представлено на рисунке 23. Производится проверка что медиа существует в БД (получив таким образом его полный путь) и удаляется из файловой системы через полный путь, возвращая 204.

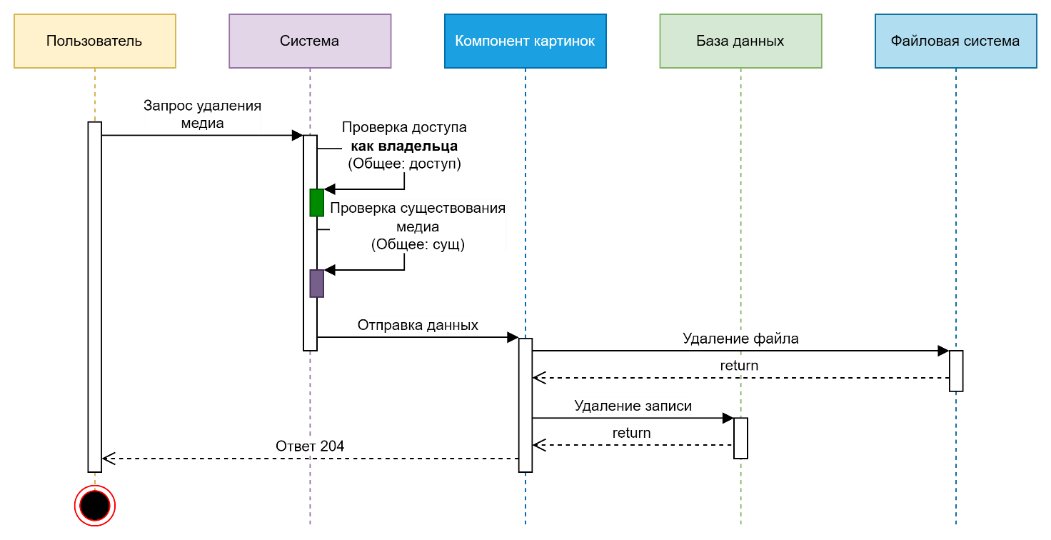


Рисунок 23 – Удаление медиа

## Требования к API

### Общие требования

Должно быть открытое АPI — правила CORS должны разрешать доступ с других доменов.

Идентификацию пользователя для не гостевых функции должно быть организовано посредством Bearer Token.

При попытке доступа гостем к защищённым авторизацией функциям системы во всех запросах необходимо возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 401 Unauthorized  **Content-Type:** application/json  **Body:** { "message": "Unauthorized" } |

При попытке доступа авторизованным пользователем к функциям недоступным для своей роли во всех запросах необходимо возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 403 Forbidden  **Content-Type:** application/json  **Body:** { "message": "Forbidden" } |

При попытке получить несуществующий ресурс возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 404 Not Found  **Content-Type:** application/json  **Body:** { "message": "<**название\_модели**> not found" } |

В случае ошибок связанных с валидацией данных в теле запроса необходимо возвращать следующий ответ:

|  |
| --- |
| **Response** |
| **Status:** 422 Unprocessable Content  **Content-Type:** application/json  **Body:** {  "message": "Validation fails"  "errors": [  <ключ>: [ <ошибка1>, <ошибка2>, ... ],  <ключ>: [ ... ],  ...  ]  } |

Вместо поля <ключ> должно быть название ключа запроса, в котором произошла ошибка валидации, а вместо <ошибка> — описание допущенной ошибки в этом ключе. Ошибок в одном ключе может быть множество, как и в одном запросе — множество ключей с ошибками.

### Авторизация

#### Регистрация

Запрос позволяет пользователю зарегистрироваться и вернуть его токен. Регистрация может быть отключена администратором и возвращать 403 код.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/reg  **Method:** POST  **Content-Type:** application/json  **Body:** {  "nickname": **{nickname}**,  "login": **{login}**,  "password": **{password}**  } | **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "token": **<токен>**  "user": <**данные\_пользователя**>  }  **Возможные статус-коды:** 201, 422 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | api | string | да |  | URL до API | https://kpch.su/api |
| 2 | nickname | string | да | мин: 2, макс: 255 | Имя юзера | Testello |
| 3 | login | string | да | мин: 2, макс: 64, уник,  regex:^[a-zA-Z0-9\_-]+$ | Логин | test |
| 4 | password | string | да | мин: 8, макс: 255 | Пароль | Test123! |

#### ВХОД

Запрос проверяет реквизиты пользователя и возвращает токен доступа при успехе.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/login  **Method:** POST  **Content-Type:** application/json  **Body:** {  "login": **{login}**  "password": **{password}**  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "token":<**токен**>  "user": {  <**"isAdmin":** **true**>  "nickname": "Testello",  "login": "test"  }  }  **Возможные статус-коды:** 200, 401, 422 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | login | string | да | Логин | mylogin |
| 2 | password | string | да | Пароль | Pass123! |

#### Выход

Запрос предназначен для удаления текущего токена пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/logout  **Method:** POST  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | token | string | да | API токен | 27|fxR7b3H3o5nPYAdtWhbk61do |

### Работа с альбомами

#### Создание альбома

Запрос для создания нового альбома (под указанным в адресной строке альбоме) пользователя. Возвращает объект альбома с его кодом в БД и создаёт папку в ФС.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "**name**": **{name}**  } | **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "hash": <**публичный\_код\_альбома**>,  "name": **{name}**,  }  **Возможные статус-коды:** 201, 401, 422 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | name | string | да | макс: 255, уник(между дочерними альбомами) | Название | Камера |
| 2 | albumCode | integer | да | существовать | Код альбома | 87 |

#### Получение альбома

Запрос для получения альбома и его дочерних альбомов. Каждый не пустой альбом имеет поле с 4-мя id медиа (mediaPreview) для получения их превью.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:**  Свой  {  "album": {  "id": 1,  "name": "Камера",  "createdAt": "2024-12-16T10:13:23",  "grantAccesses": [  {  "id": 215,  "name": "Какой-то пользователь"  }, { ... }, { ... }  ],  "mediaCount": 31,  "mediaPreview": {  "sign": "2\_JDJ5JDEyJG11RGc24aWRX",  "ids": [84, 30, 69, 59]  },  "children": [ <**дочерние\_альбомы**> ]  }  }  Чужой доступный  {  "album": {  "id": 1,  "name": "Скаченное",  "owner": {  "id": 326,  "name": "От какого-то пользователя"  },  "mediaCount": 29,  <**...**>  }  }  **Возможные статус-коды:** 200, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | albumId | integer | да | существовать | Код альбома | 87 |

#### Удаление альбома

Запрос для удаления альбома. Удаляет все связанные данные в БД и всё медиа в ФС.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**  **Method:** DELETE  **Authorization:** Bearer **{token}** | **Status:** 204 Deleted  **Возможные статус-коды:** 204, 401, 403, 404 |

### Работа с альбомными доступами

#### Создание кода приглашения

Запрос для создания кода приглашения для доступа к альбому. Если лимиты не указаны, то приглашение бессрочное и вступить может любое количество человек. Если задать expiresAt и timeLimit, то сработает только timeLimit.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**/invite  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "expiresAt": **{expiresAt},**  "timeLimit": **{timeLimit}**,  "joinLimit": **{joinLimit}**  } | **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "invitation": {  "code": "928Lm8e4",  "expiresAt": "2025-12-30 20:50:00",  "joinLimit": 1  }  }  **Возможные статус-коды:** 201, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | expiresAt | datetime | нет | > текущего | Дата просрочки | 2024-12-30 |
| 2 | timeLimit | integer | нет | мин: 1 | Лимит времени (минуты) | 60 |
| 3 | joinLimit | integer | нет | мин: 1 | Лимит вступлений | 1 |

#### Получение альбома в приглашении

Запрос для получения информации об альбоме по коду приглашения. Показывает 30 последних картинок.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/invitation/**{invationCode}**/album  **Method:** GET | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "album": {  "id": 3,  "name": "Чей-то чужой альбом",  },  "media": [  {...}, {...}, {...}  ]  }  **Возможные статус-коды:** 200, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | invationCode | string | да | Код приглашения | Bc7ELcIf |

#### Вступление по приглашениию

Запрос для получения доступа к альбому чужого пользователя по коду приглашению. При попытке повторно вступить или попытке вступить в свой альбом — 409 код.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/invitation/**{invationCode}**/join  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 404, 409 |

#### Удаление кода приглашения

Запрос для удаления кода приглашения.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/invitation/**{code}**  **Method:** DELETE  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404 |

#### Отзыв доступа для пользователя

Запрос для удаления доступа к своему альбому другого пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**/access/**{userId}**  **Method:** DELETE  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404 |

#### Отзыв доступа на себе

Запрос для удаления доступа к чужому альбому от себя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**/access  **Method:** DELETE  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 404 |

### Работа с медиа

#### Загрузка медиа в альбом

Запрос для загрузки на ФС сервера медиа и информации об каждой в БД. Разрешённые типы проверяются по указанным в настройках сервера. Конфликт имён разрешается автоматическим переименованием, дубликаты обнаруживаются по вычисленному хешу файла и пропускаются.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**/media  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** multipart/form-data  **Body:** {  "medias[]": **{medias}**,  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "successful": [  {  "id": 1,  "name": "test.jpg",  "date": "2024-10-29 19:56:10",  "size": 1696738,  "width": 1080,  "height": 2340,  "hash": "df59b7d94c03b59a”  }, { ... }  ],  "errored": [  {  "name": "image.png",  "message": “already exist"  }, { ... }  ]  }  **Возможные статус-коды:** 200, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** |
| 1 | medias | array | да | мин: 1 | Массив файлов |
| 1 | medias.\* | file | нет | mimes: *<определено настройками>* | Картинка |

#### Получение списка МЕДИА

Запрос для получения списка всех медиа в указанном альбоме. Так же выводится временная (на сутки) сигнатура доступа для получения превью/оригинала/загрузки медиа из этого альбома.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**/media  ?page=**{page}**  &limit=**{limit}**  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "sign": <**сигнатура**>  "media": [  {  "id": 1,  "name": "test.jpg",  "date": "2024-10-29 19:56:10",  "size": 1696738,  "width": 1360,  "height": 2128,  "hash": "df59b7d94c03b59a”  },  { ... }, { ... },  ],  }  **Возможные статус-коды:** 200, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | page | integer | да | Номер страницы | 2 |
| 2 | limit | integer | нет | Лимит на каждую страницу | 100 |

#### Отображение превью картинки

Запрос для получения файла-превью указанного размера (разрешённый размер из публичной информации о сервере). Для аутентификация используется сигнатура доступа, которую можно получить из запроса списка картинок. Если указать запрещённый размер, то будет перенаправление на приближённый размер.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  /medias/**{mediaHash}**  /thumb/**{orient}{size}**  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** image/webp  **Возможные статус-коды:** 200, 301, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | pictureId | integer | да | Код картинки | 529 |
| 2 | orient | "w"|"h"|"q" |  | Ось указания размера (по ширине, высоте или обрезка в квадрат) |  |
| 3 | size | integer | да | Размер по оси (пиксели) | 720 |
| 4 | sign | string | да | Сигнатура доступа | 2\_5EyJHc3aBoY1Ju |

#### Отображение картинки (оригинал)

Запрос для отображения файла. Для аутентификация используется сигнатура доступа, которую можно получить из запроса списка картинок.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  /medias/**{mediaHash}**  /original  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** <"image/jpeg"|"video/mp4"|...>  **Возможные статус-коды:** 200, 403, 404 |

#### Скачивание картинки

Запрос для скачивания файла. Для аутентификация используется сигнатура доступа, которую можно получить из запроса списка картинок.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  /medias/**{mediaHash}**  /download  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** <"image/jpeg"|"video/mp4"|...>  **Content-Disposition:**  attachment; filename=<имя\_файла>  **Возможные статус-коды:** 200, 403, 404 |

#### Удаление медиа

Запрос для удаления конкретного медиа. Удаляет связанные данные в БД и картинку в ФС.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumCode}**  /medias/**{mediaHash}**  **Method:** DELETE  **Authorization:** Bearer **{token}** | Успех  **Status:** 204 Deleted  **Возможные статус-коды:** 204, 401, 403, 404 |

### Работа с тегами

#### Создание личного тега

Запрос для создания личного тега пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "name": **{name}**  } | **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "id": <**id\_тега**>,  "name": <**название**>  }  **Возможные статус-коды:** 201, 401, 409 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | name | string | да | макс: 255, уник(для юзера) | Название | Семья |

#### Получение списка личных тегов

Запрос на получение списка личных тегов.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** [  {  "id": <**id\_тега**>,  "name": <**название**>  },  { ... },  ]  **Возможные статус-коды:** 204, 401 |

#### Добавление тега к МЕДИА

Запрос для добавления связи тега с конкретным медиа.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/{**albumCode**}  /medias/**{mediaHash}**  /tags/**{tagId}**  **Method:** POST  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404, 409 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | tagId | string | да | существовать | Код тега | 5 |

#### Удаление тега с МЕДИА

Запрос для удаления связи тега с конкретным медиа.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/{**mediaHash**}  /medias/**{mediaHash}**  /tags/**{tagId}**  **Method:** DELETE  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404, 409 |

### Работа с пользователями

#### Получение себя

Запрос на данных о себе.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/me  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 1,  "name": "Testello",  "login": "test",  <"role": "admin">  }  }  **Возможные статус-коды:** 200, 401 |

#### Редактирование себя

Запрос на изменение данных о себе.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/me  **Method:** PATCH  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "name": **{nickname}**,  "login": **{login}**,  "password": **{password}**  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 1,  "name": "Testello",  "login": "test",  <"role": "admin">  }  }  **Возможные статус-коды:** 200, 401, 422 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | nickname | string | нет | мин: 2, макс: 255 | Имя юзера | Testello2 |
| 2 | login | string | нет | мин: 2, макс: 64, уник,  regex: ^[a-zA-Z0-9\_-]+$ | Логин | test2 |
| 3 | password | string | нет | мин: 8, макс: 255 | Пароль | NewPass! |

#### Просмотр списка пользователей

Запрос на просмотр списка пользователей администратором.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "users": [  {  "id": 2,  "name": "Чел",  "isBanned": false,  "login": "test1",  <"role": "admin">  "albumsCount": 1,  "mediaCount": 31,  "usedQuota": 10630044057,  "quota": 10737418240  }, { ... }, { ... }  ]  }  **Возможные статус-коды:** 200, 401, 403 |

#### Просмотр пользователя

Запрос просмотра пользователя администратором.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/**{userId}**  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 2,  "name": "Никнейм пользователя",  "login": "test1",  <"role": "admin">  "albumsCount": 1,  "accessableCount": 2,  "sharesCount": 2,  "mediaCount": 31,  "usedQuota": 10630044057,  "quota": 10737418240  }  }  **Возможные статус-коды:** 200, 401, 403, 404 |

#### Редактирование пользователя

Запрос редактирование пользователя администратором.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/**{userId}**  **Method:** PATCH  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "name": **{nickname}**,  "login": **{login}**,  "password": **{password},**  "role\_id": **{roleId}**  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 2,  "name": "Никнейм пользователя",  "login": "test1",  <"role": "admin">  }  }  **Возможные статус-коды:** 200, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | roleId | integer | нет | существовать | ИД роль | 2 |

### Работа с настройками

#### Просмотр публичных настроек

Запрос публичных настроек. Возвращает разрешённые расширения, разрешённые запрашиваемые размеры превью, размер бесплатной квоты и статус разрешённый загрузки.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/setups  **Method:** GET | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "setups": {  "allowed\_upload\_mimes": [  "mp4", "jpg", "png", "gif"  ],  "allowed\_preview\_sizes": [  144, 240, 360, 480, 720, 1080  ],  "free\_quota": 5368709120,  "is\_upload\_disabled": false  }  }  **Возможные статус-коды:** 200 |

## Требования к сайту

При входе неавторизированного пользователя он может раскрыть форму с переключением регистрации (см. рис. 24) и входа (см. рис. 25).

|  |  |
| --- | --- |
| Рисунок 24 – Макет панели регистрации | Рисунок 25 – Макет панели входа |

После регистрации администратор попадает на главную — страницу коревого альбома. Пока там пусто и предлагается создать новый альбом или загрузить на сайт картинки. Заполненная страница альбома представлена на рисунке 26.

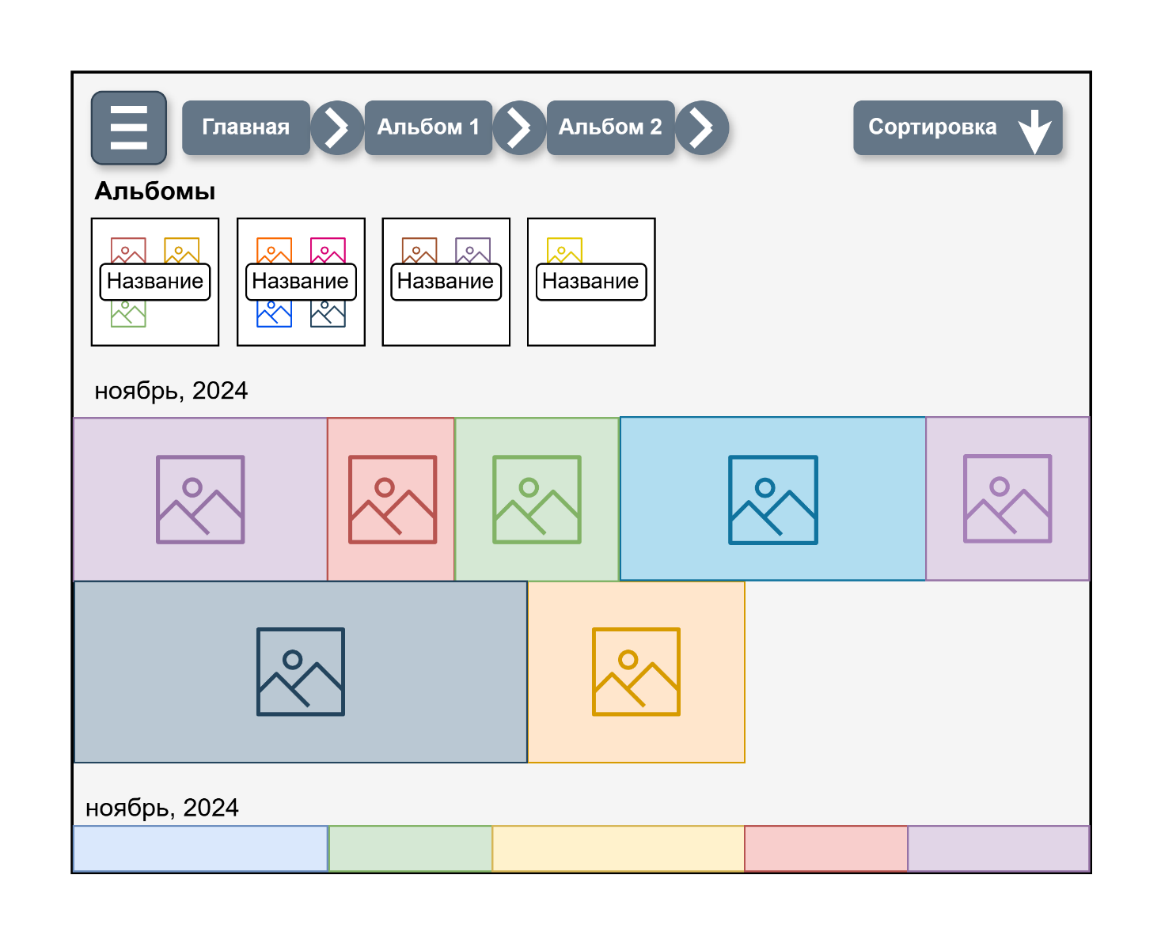


Рисунок 26 – Макет страницы альбома

В заголовке страницы можно увидеть кнопку меню, цепочку родителей до открытого альбома (через который можно из любой части пути просмотреть дочерние альбомы) и выпадающий список сортировки.

При клике на картинку открывается просмотрщик, позволяющий детально посмотреть медиа (см. рис. 27).

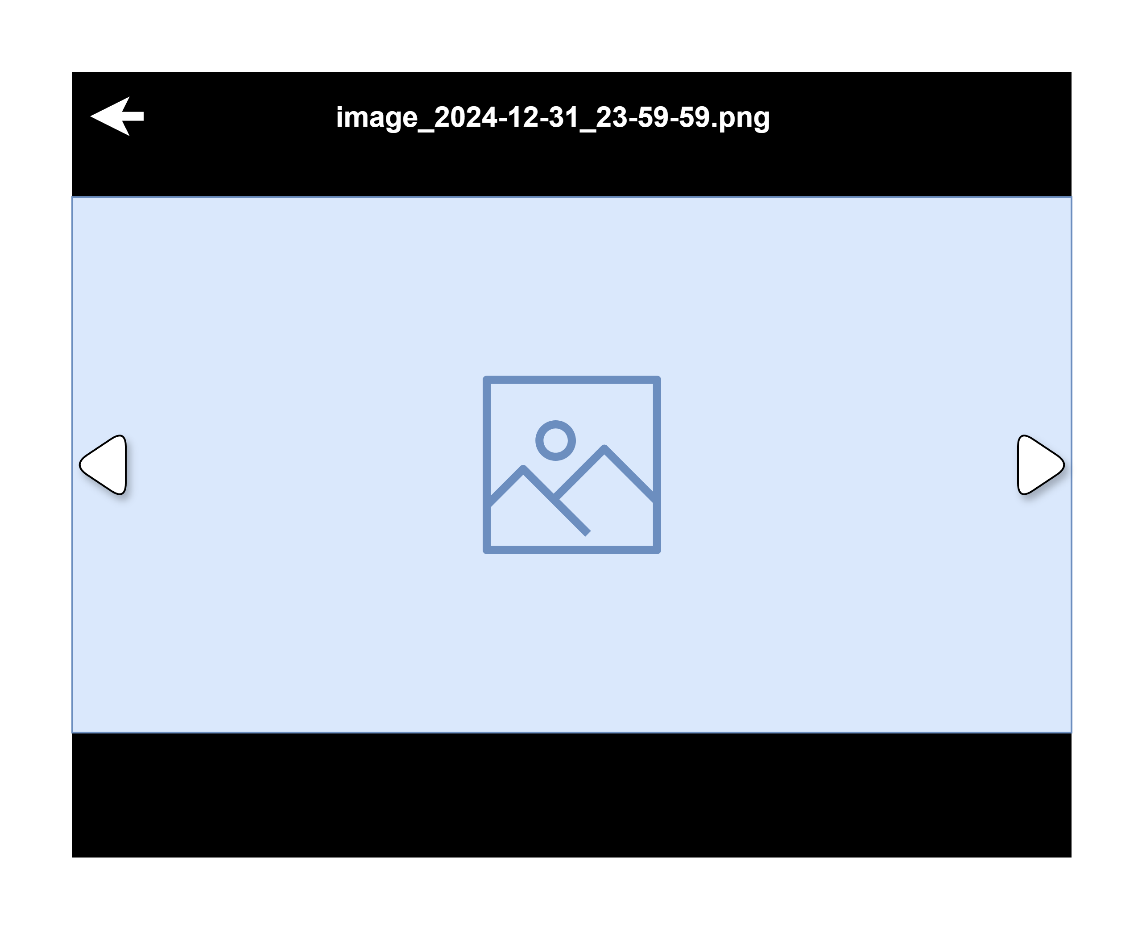


Рисунок 27 – Макет просмотрщика

## Требования к мобильному приложению

При входе неавторизированного пользователя выводить страницу с переключением регистрации (см. рис. 28) и входа (см. рис. 29).

|  |  |
| --- | --- |
| Рисунок 28 – Макет регистрации | Рисунок 29 – Макет входа |

При входе авторизированного пользователю доступны страницы, которые можно переключать (через таббар):

* *Недавние*: недавние фотографии и картинки
* *Альбомы*: список всех доступных альбомов-папок
* *Настройки*: обзор профиля, информация о квоте и сервере

На главной (см. рис. 30) авторизированного пользователя выводить сетку недавних картинок синхронизирующийся альбомов. На каждой картинке выводить статус синхронизации с сервером. Каждая картинка при нажатии открывает полноэкранный просмотрщик (см. рис. 31). Также на главной необходима кнопка, которая открывает управление синхронизацией (см. рис. 32).

На странице альбомов (см. рис. 33) показывать списки доступных альбомов других пользователей, список альбомов, которые есть на сервере, список, синхронизирующийся альбомов и список локальных не синхронизирующийся альбомов. При нажатии на альбом открывать сетку картинок, похожую на сетку главной (см. рис. 30).

|  |  |
| --- | --- |
| Рисунок 30 – Макет главной страницы | +  Рисунок 31 – Макет просмотрщика |
| Рисунок 32 – Макет управления синхронизацией | Рисунок 33 – Макет страницы альбомов |

На странице настроек (см. рис. 34) отображать информацию о себе с возможностью редактирования, список альбомов, которые открыты пользователям и возможность выйти со всех устройств.



Рисунок 34 – Макет страницы настроек

# Глава 2. Проектирование и разработка базы данных

## Концептуальная модель данных

Концептуальная модель — это формализованное описание предметной области, не связанное с какими-либо компьютерными средствами. Сущности (таблицы) обозначаются прямоугольниками; Атрибуты (поля таблиц) — овалы, которые могут быть объедены в скруглённые прямоугольники; Связи с обозначением мощности — стрелками (в направлении, где мощность равна многим — двойная стрелка, а со стороны, где она равна единице — одинарная).

Анализируя предметную область, были выделены следующие сущности с их атрибутами (см. рис. 35):

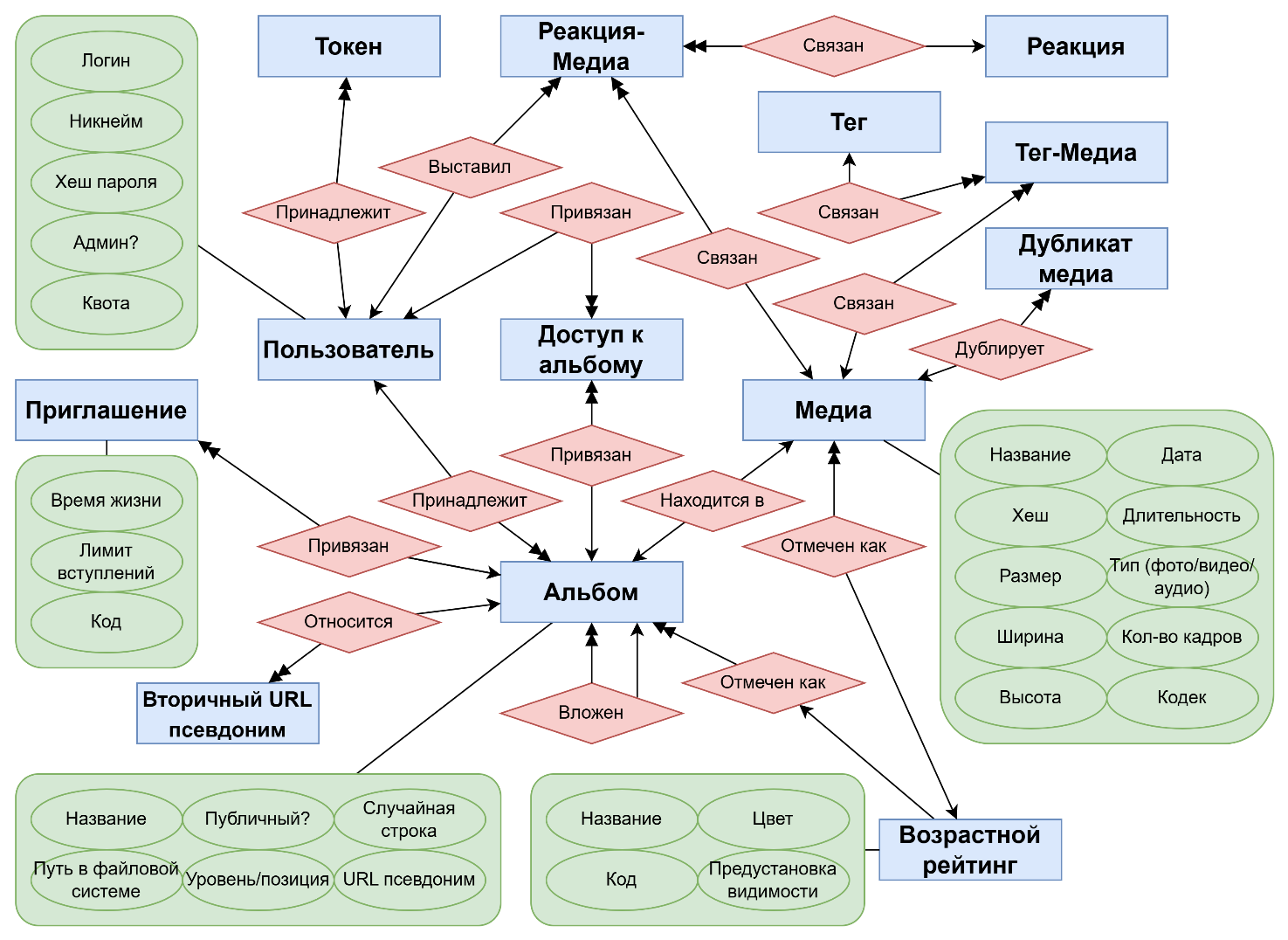


Рисунок 35 – Концептуальная модель данных

## Логическая модель данных

Логическая модель базы данных — схема базы данных, выраженная в понятиях модели данных. Этим отличается от концептуальной модели, описывающей семантику предметной области без указания технологии (конкретных методов реализации), и от физической модели, которая описывает конкретные физические механизмы, применяемые для хранения данных в накопителях.

Используя концептуальную модель данных и выполнив нормализацию данных до 3 нормальной формы, была разработана логическая модель данных (см. рис. 36):

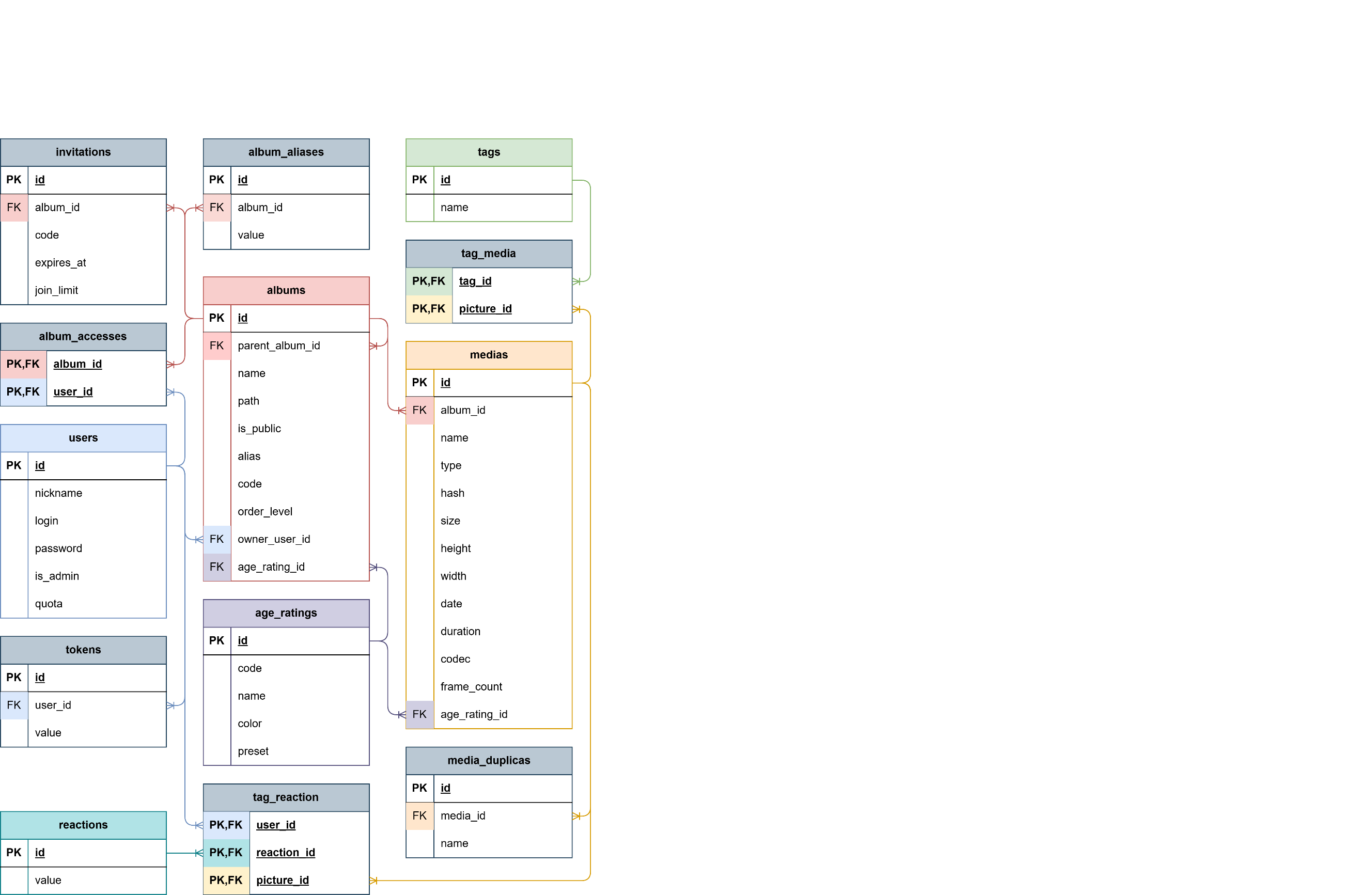


Рисунок 36 – Логическая модель данных

## Физическая модель данных

Физическая модель данных — это модель данных, описанная с помощью средств конкретной системы управления базами данных (СУБД). Физическая модель данных строится на базе логической путём добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имён таблицам, атрибутам и т.д. Физическая модель данных фактически является готовым заданием на создание БД, имея которое можно реализовать БД в выбранной СУБД.

Выбранной СУБД является MySQL. Используя логическую модель, была разработана физическая модель данных (см. рис. 37):

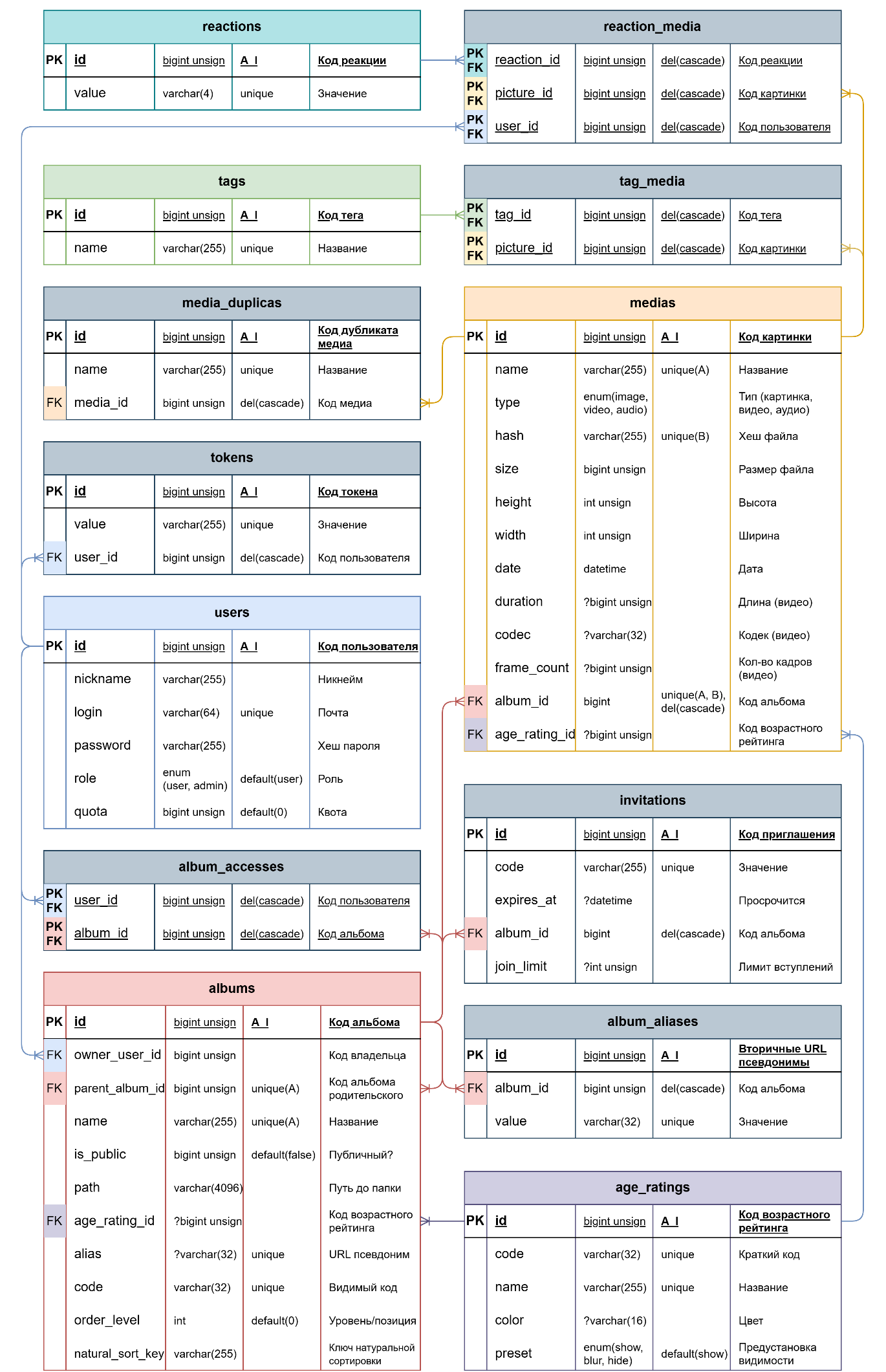


Рисунок 37 – Физическая модель данных

# Глава 3. Проектирование и разработка API сервиса

Backend-часть проекта WepicSync разработана как RESTful API на Laravel 12 и предназначена для управления альбомами медиа, тегами, пользователями и правами доступа.

API реализовано с применением архитектурного паттерна MVC (Model-View-Controller), что обеспечивает четкое разделение логики, данных и слоя взаимодействия с клиентом. Для работы с изображениями и видео используется Intervention Image (обёртка GD, Imagick и libvips) и FFMpeg PHP (обёртка FFMpeg и FFProbe), позволяющие обрабатывать медиафайлы для отображения в веб-сайте создавая нужные превью для экономии к скорости соединения и возвращая совместимые форматы.

Разработка велась в ИСР PHPStorm с использованием локального WAPM сервера Laragon, содержащий сборку из веб-сервера Apache, системы управления базами данных MySQL и интерпретатор скриптов PHP. Для контроля версий использовался Git.

## Зависимости проекта

1. *PHP 8.3* – серверный язык программирования с улучшенной производительностью и синтаксисом.
2. *Laravel 12* – фреймворк для создания веб-приложений и API, предоставляющий ORM (Eloquent), маршрутизацию, миграции и другие инструменты.
3. *Intervention Image 3.11* – интеграция с библиотеками для работы над изображениями такими как GD, Imagisk и libvips.
   1. *GD Library 2.1* –библиотека для создания и обработки изображений на лету. Поддерживает форматы JPEG, PNG, GIF, WebP и другие.
   2. *Intervention Image Laravel 1.3* –интеграция Intervention Image в Laravel
4. *PHP FFMpeg 1.3*–интеграция FFmpeg для обработки видео (конвертация, извлечение кадров) и FFProbe (получение информации о контейнере, видео и аудио дорожках).
   1. *FFmpeg 7.1* – набор инструментов для работы с аудио и видео (кодирование, декодирование, конвертация, потоковая передача).
   2. *Laravel FFmpeg 8.7*–интеграция PHP FFmpeg в Laravel
5. *Kalnoy/NestedSet 6* –расширение Eloquent, реализация иерархических структур (деревьев) в базе данных и удобные методы для извлечения цепочки родителей и ветвей из дочерних элементов.
6. *Spatie Laravel Tags 4* – расширение Eloquent, система тегирования с поддержкой отношений и гибкой фильтрации.
7. *Puppeteer 24* – библиотека с удобным API для управления Chrome/Firefox headless (безголовым браузером, в фоновом процессе)
   1. *Chrome Headless Shell 136* – безоконный браузер.
   2. *Spatie Browsershot 5* – интеграция генерации скриншотов веб-страниц через Puppeteer/Chrome в Laravel.
8. *g4t/swagger 4*–автоматическая генерация OpenAPI-документации для API.

## Структура проекта

Backend-часть проекта построена на паттерне MVC (Model-View-Controller):

1. Модели (Models) — отвечают за структуру данных и бизнес-логику, связанную с БД.
2. Контроллеры (Controllers) — обрабатывают HTTP-запросы, работают с моделями и возвращают представления/ресурсы.
3. Представления (Views) — отвечает за отображение информации пользователю.

Структура проекта включает в себя следующие основные каталоги и основные файлы:

1. app/ – Ядро приложения
   1. Console/ – Artisan-команды.
   2. Exceptions/ – Исключения, возвращающие Http ответ с JSON.
   3. Helpers/ – Общие помощники.
   4. Enums/ – Типы строк.
   5. Http/ –Логика обработки HTTP-запросов.
      1. Controllers/ – Контроллеры.
      2. Middleware/ – Промежуточное ПО (аутентификация, проверка доступа).
      3. Requests/ – Валидация входящих запросов.
      4. Resources/ – API-ресурсы (JSON-трансформеры).
   6. Models/ – Eloquent-модели.
2. config/ – Конфигурации (Intervention, FFMpeg).
3. database/ – Работа с БД.
   1. migrations/ – Миграции.
   2. seeders/ – Начальные данные (сидеры).
4. public/ – Точка входа (index.php), статические файлы.
5. routes/ – Маршруты.
   1. api.php – API-маршруты.
   2. web.php – Веб-маршурты.
6. storage/ – Логи, загруженные файлы, кеш.

## Маршруты

### Сервисные маршруты

Публичный маршрут /setups возвращает разрешённые расширение файлов, разрешённые размеры превью, разрешённые реакций и возрастные рейтинги. Через cache.headers, добавляющий заголовок в поле, запросы будут кешироваться на стороне клиента и прокси-серверов на 1 месяц (max\_age=2628000) для снижения нагрузки на сервер.

Публичный маршрут /who возвращает название проекта и установленный администратором название сервиса (который отображается в заголовке вкладки).

Оба описанных выше маршрута представлены на листинге 1.

Листинг 1 – Часть кода с сервисными маршрутами из /routes/api.php

1. *// Публичные предустановки*Route::get('/setups', [SettingsController::class, 'public'])
2. ->middleware('cache.headers:public;max\_age=2628000;etag');
3. *// Возвращение название проекта и установленного название сервиса*
4. Route::any('who', fn () => [
5. 'service' => 'WepicSync',
6. 'appname' => config('app.name'),
7. ]);

### Маршруты пользователей (UserController)

Группа маршрутов с префиксом /users, представленный на листинге 2, разделена на три части:

* *Публичные методы*: login (вход) и reg (регистрация) доступны без авторизации.
* *Личный кабинет*: маршруты me, editSelf, logout защищены middleware token.auth и требуют валидного токена. Здесь пользователь может управлять своим профилем.
* *CRUD пользователей*: Доступна только с ролью admin (проверка через token.auth:admin). Включает CRUD-операции над пользователями (create, showAll, show, edit, delete)

Листинг 2 – Часть кода с маршрутами пользователей из /routes/api.php

1. Route
2. ::controller(UserController::class)
3. ->prefix('users')
4. ->group(function ($users) { *// [ПОЛЬЗОВАТЕЛИ]*
5. $users->post ('login' , 'login'); *// Вход*
6. $users->post ('reg' , 'reg' ); *// Регистрация*
7. $users->middleware('token.auth')->group(function ($authorized) { *// [АВТОРИЗИРОВАННЫЙ]*
8. $authorized->get ('me', 'showSelf'); *// Получение личных данных*
9. $authorized->patch('', 'editSelf'); *// Редактирования себя*
10. $authorized->post ('logout', 'logout' ); *// Выход (удаление токена)*
11. });
12. $users->middleware('token.auth:admin')->group(function ($usersManage) { *// [УПРАВЛЕНИЕ]*
13. $usersManage->post('', 'create' ); *// Создание*
14. $usersManage->get ('', 'showAll'); *// Список*
15. $usersManage->prefix('{id}')->group(function ($userManage) {
16. $userManage->get ('', 'show' )->where('id', '[0-9]+'); *// Отображение*
17. $userManage->patch ('', 'edit' )->where('id', '[0-9]+'); *// Редактирование*
18. $userManage->delete('', 'delete')->where('id', '[0-9]+'); *// Удаление*
19. });
20. });
21. });

### Маршруты альбомов, доступы (AlbumController & AccessController)

Группа маршрутов с префиксом /albums/{album\_hash}, представленный на листинге 3, имеет свои и вложенные другими контроллерами частей:

* *Свои маршруты:* получению информации о текущем, вложенных и родительских альбомов — get, генерация и вывод превью-картинки для мессенджеров — ogImage.
* *Для владельца:* CRUD альбомов и reindex для переиндексации содержимого.
* *Управление доступами:* вложенный префикс /access, контроллер AccessController.
* *Управление медиа-содержимым:* вложенный префикс /medias, контроллер MediaController.

Листинг 3 – Часть кода с маршрутами альбомов из /routes/api.php

1. Route
2. ::middleware('token.auth:guest')
3. ->controller(AlbumController::class)
4. ->prefix('albums/{album\_hash}')
5. ->group(function ($album) { *// [АЛЬБОМ]*
6. $album->get('', 'get'); *// Получение общей информации, дочерние альбомы и цепочку предков*
7. $album->get('og', 'ogImage')->name('get.album.og'); *// Генерация и/или возвращение превью*
8. $album->middleware('token.auth:owner')->group(function ($albumManage) { *// [УПРАВЛЕНИЕ]*
9. $albumManage->get ('reindex', 'reindex'); *// Переиндексация*
10. $albumManage->post ('', 'create'); *// Ручное создание нового альбома*
11. $albumManage->patch ('', 'update'); *// Обновление информации*
12. $albumManage->delete('', 'delete'); *// Удаление всего альбома с содержимым*
13. });
14. $album->post('invite', [InvitationController::class, 'store'])
15. ->middleware('token.auth:owner'); *// Генерировать код приглашения*
16. $album
17. ->controller(AccessController::class)
18. ->middleware('token.auth:owner)
19. ->prefix('access')
20. ->group(function ($albumRights) {  *// [ДОСТУПЫ НА АЛЬБОМЕ]*
21. $albumRights->get ('', 'showAll'); *// Показать всех, кому доступен альбом*
22. $albumRights->post ('', 'create' ); *// Создание доступа*
23. $albumRights->delete('', 'delete'); *// Отзыв доступа*
24. });
25. $album->controller(MediaController::class)->prefix('medias')->group( *...* );
26. });

### Маршруты медиа, установки тегов и реакций (MediaController)

* Группа маршрутов с префиксом /images, представленный на листинге 4, имеет следующие маршруты:
* *Свои маршруты:* получение постраничного списка медиа в альбоме index.
* *Конкретное медиа:* префикс {media\_hash}, получение информации о конкретном медиа show, получение оригинального медиафайла orig, скачивание медиафайла как attachment download, получение/генерация превью медиа.
* *Для владельца:* CRUD медиа, загрузка в свой альбом медиа.
* *Управление тегами:* вложенный префикс /tags, контроллер TagController, только для владельца.
* *Управление реакциями:* вложенный префикс /reactions, контроллер ReactionController.

Листинг 4 – Часть кода с медиа альбомов из /routes/api.php

1. $album
2. ->controller(MediaController::class)
3. ->prefix('medias')
4. ->group(function ($albumImages) { *// [МЕДИА В АЛЬБОМЕ]*
5. $albumMedias->get('', 'index') *// Страница с информацией о каждом медиа*
6. ->withoutMiddleware('throttle:api'); *// Отключение rate лимита*
7. $albumMedias->middleware('token.auth:owner')->post('', 'upload');*// Загрузка в альбом*
8. $albumMedias->prefix('{media\_hash}')->group(function ($media) { *// [КОНКРЕТНОЕ МЕДИА]*
9. $media->middleware('token.auth:owner')->delete('', 'delete'); *// Удаление*
10. $media->middleware('token.auth:owner')->patch ('', 'update'); *// Обновление информации*
11. $media->get('', 'show'); *// Получение информации*
12. $media->get('orig', 'orig') *// Получение оригинального файла*
13. ->withoutMiddleware('throttle:api')
14. ->name('get.media.orig');
15. $media->any('download', 'download'); *// Получение файла как attachment*
16. $media->get('thumb/{orient}{px}{ani?}', 'thumb') *// Получение превью медиа*
17. ->where('orient', '[whqWHQ]')
18. ->where('px' , '[0-9]+')
19. ->where('ani' , '[a]')
20. ->withoutMiddleware('throttle:api')
21. ->name('get.media.thumb');
22. $media
23. ->controller(TagController::class)
24. ->middleware('token.auth:owner')
25. ->prefix('tags')
26. ->group(function ($mediaTags) { *// [ТЕГИ НА МЕДИА]*
27. $mediaTags->post ('', 'set'); *// Установить тег на медиа*
28. $mediaTags->delete('', 'unset'); *// Убрать тег с медиа*
29. });
30. $media
31. ->controller(ReactionController::class)
32. ->middleware('token.auth:user')
33. ->prefix('reactions')
34. ->group(function ($mediaReactions) { *// [РЕАКЦИИ НА МЕДИА]*
35. $mediaReactions->post ('', 'set'); *// Установить реакцию на медиа*
36. $mediaReactions->delete('', 'unset'); *// Убрать реакцию с медиа*
37. });
38. });
39. });

## Конфигурационный файл

Конфигурационные файлы (Config) в Laravel используются для управления настройками приложения. Они позволяют централизованно хранить параметры, которые могут изменяться в зависимости от среды выполнения, обеспечивая гибкость и удобство при развертывании проекта.

Конфигурационные файлы располагаются в папке config/, а значения в них могут быть заданы через переменные окружения, хранящиеся в файле .env. Это позволяет легко адаптировать приложение под различные серверные окружения без изменения исходного кода.

В листинге 5 представлен файл конфигурации config/setups.php, который содержит настройки сервера. В нем определены допустимые форматы загружаемых файлов (allowed\_\*\*\*\_extensions), размеры превью (allowed\_preview\_sizes), лимиты квоты для регистрируемых пользователей (default\_quota\_bytes), а также параметры, регулирующие ограничения загрузки (upload\_disable\_percentage).

Листинг 5 – Конфигурация настроек сервера config/setups.php

1. return [
2. 'upload\_disable\_percentage' => (int) env('UPLOAD\_DISABLE\_PERCENTAGE', 90),
3. 'default\_quota\_bytes' => (int) env('DEFAULT\_QUOTA\_BYTES', 0),
4. 'allowed\_image\_extensions' => explode(',', env('ALLOWED\_EXT\_IMAGE', 'jpg,png,gif,webp')),
5. 'allowed\_video\_extensions' => explode(',', env('ALLOWED\_EXT\_VIDEO', 'mp4,webm,mkv')),
6. 'allowed\_audio\_extensions' => explode(',', env('ALLOWED\_EXT\_AUDIO', 'mp3,ogg,flac')),
7. 'allowed\_preview\_sizes' => array\_map('intval',
8. explode(',', env('ALLOWED\_PREVIEW\_SIZES', '144,240,360,480,720,1080'))
9. ),
10. ];

## Модели

Модели в Laravel используются для работы с базой данных через Eloquent ORM. Они позволяют взаимодействовать с таблицами, определять связи между данными и управлять бизнес-логикой. Каждая модель представляет собой класс, связанный с определенной таблицей, а свойства и методы модели помогают упрощать работу с данными.

Каждая модель может включать следующие части:

1. Определение полей (fillable), которые можно массово заполнять;
2. Преобразователь атрибутов (casts) при обращении к ним;
3. Переопределение название таблицы (table);
4. Инициализация обработчиков событий (booted) чтобы выполнять свои действия (например, при сохранении модели);
5. Cвои вспомогательные методы для работы с моделью;
6. Связи с другими моделями.

В проекте 11 моделей: Album, Media, User, Tag, Reaction, AgeRating, AccessRight, MediaDuplica, ReactionMedia, Token — у всех fillable повторяет обычные поля из схемы БД. Описан будет только Album (частично).

### Album

Модель альбома включает обработчик события сохранения, который автоматически обновляет поле натуральной сортировки (natural\_sort\_key). Это позволяет корректно сортировать альбомы с учётом числовых значений в их названиях (например, "Альбом 2" будет следовать после "Альбом 1", а не "Альбом 10"). Код представлен на листинге 6.

Листинг 6 – Ключ натуральной сортировки альбома из app/Models/Album.php

1. /\*\* Инициализация обработчиков событий модели \*/
2. protected static function booted(): void {
3. static::saving(function ($item) {
4. // Автоматически обновляем natural\_sort\_name при сохранении модели
5. $item->natural\_sort\_key = self::normalizeName($item?->name ?? '');
6. });
7. }
8. /\*\* Константа максимальной длины сортировочного имени \*/
9. const MAX\_NATURAL\_SORT\_KEY\_LENGTH = 255;
10. /\*\* Константа количества дополнительных "0" перед числами в названии \*/
11. const NATURAL\_SORT\_NUM\_PADDING = 12;
12. */\*\* Генерация имени для натуральной сортировки.*
13. *\* Преобразует строку так, чтобы числа в названиях сортировались числовым порядком*
14. *\* @param* string $originalName *исходное название для обработки*
15. *\* @return string название, по которому можно совершать натуральную сортировку*
16. *\* @uses* MAX\_NATURAL\_SORT\_KEY\_LENGTH *ограничение длины итоговой строки*
17. *\* @uses* NATURAL\_SORT\_NUM\_PADDING *дополнение чисел нулями до определённого кол-во знаков*
18. *\*/*
19. public static function normalizeName(string $originalName): string
20. {
21. // Каждое число в названии расширяется нулями
22. // При 12 число 100 преобразуется в 000 000 000 100 без пробелов
23. $normalizedName = preg\_replace\_callback('/\d+/', function ($matches) {
24. return str\_pad($matches[0], self::NATURAL\_SORT\_NUM\_PADDING, '0', STR\_PAD\_LEFT);
25. }, $originalName);
26. // Обрезка строки до определённого количества символов
27. $truncatedName = mb\_substr($normalizedName, 0, self::MAX\_NATURAL\_SORT\_KEY\_LENGTH);
28. // Удаление битых/невалидных Unicode символов
29. return iconv('UTF-8', 'UTF-8//IGNORE', $truncatedName);
30. }

Для получения альбома по таким идентификаторам альбома как: внешний код hash, первичный алиас alias и вторичные алиасы в таблице album\_aliases — используется метод getByHashOrAlias (см. лист. 7), который может быть дополнен дополнительными запросами, передав 2-м параметром callback функцию. Так же если в БД ещё не существует запрашиваемый корневой модели пользователя или сервера — автоматически создаются и возвращаются.

Листинг 7 – Метод поиска альбома из app/Models/Album.php

1. */\*\* Получение альбома по его уникальному хешу или алиасам.*
2. *\* Может создавать корневой альбом пользователя или сервера*
3. *\* @param string $hashOrAlias хеш или алиас*
4. *\* @param Closure $modifyQuery*
5. *\* @return Album найденная модель альбома с выполненным modifyQuery*
6. *\* @throws NotFoundException*
7. *\*/*
8. static public function getByHashOrAlias(string $hashOrAlias, Closure $modifyQuery = null): Album
9. {
10. $user = request()->user();
11. $query = Album::*query*();
12. if ($modifyQuery)
13. $modifyQuery($query);
14. *// Если запрошен корневой альбом сервера, то генерируем и возвращаем, если нет*
15. if ($hashOrAlias === 'root') {
16. $album = $query->firstOrCreate([
17. 'path' => '/'
18. ], [
19. 'name' => '',
20. 'hash' => 'root',
21. ]);
22. }
23. *// Если запрошен корневой альбом пользователя, то генерируем и возвращаем, если нет*
24. else if ($hashOrAlias === 'my' && $user !== null) {
25. $album = $query->firstOrCreate([
26. 'path' => "/../users/$user->id/"
27. ], [
28. 'name' => "User #{$user->id} root album",
29. 'hash' => Str::*random*(25),
30. 'owner\_user\_id' => $user->id,
31. ]);
32. }
33. *// Ищем альбом по хешу, первичному и вторичному алисасу*
34. else {
35. $album = $query
36. ->where('hash', $hashOrAlias)
37. ->orWhere('alias', $hashOrAlias)
38. ->first();
39. if (!$album) {
40. *// Ищем по вторичные алиасам*
41. $album = $query
42. ->join((new AlbumAlias())->getTable() .' as aliases', 'albums.id', '=', 'aliases.album\_id')
43. ->where('aliases.name', $hashOrAlias)
44. ->first();
45. }
46. }
47. if (!$album)
48. throw new NotFoundException(Album::class);
50. return $album;
51. }

Чтобы получать медиафайлы из не публичного альбома требуется проверять доступ и пользователя. При запросе из разметки фронтенд-частей медиафайлов — токен не отправить, поэтому используется сигнатура, которая является хешем токена пользователя вместе с текущей датой. Метод создания и проверки находятся в коде модели альбома (см. лист. 8).

Листинг 8 – Методы генерации и проверки сигнатуры из app/Models/Album.php

1. */\*\* Константа времени жизни (секунд) кеша доступа по сигнатуре к медиа-файлам \*/*
2. const *SIGN\_CACHE\_TTL* = 3600;
3. */\*\* Проверка на доступ к статичным файлам альбома*
4. *\* @param string $albumHash*
5. *\* @param int|null $userId*
6. *\* @return string*
7. *\*/*
8. static public function buildSignCacheKey(string $albumHash, int $userId = null): string {
9. return "signAccess:to=$albumHash;for=$userId";
10. }
11. */\*\* Генерация или возвращения из кеша сигнатуры доступа к прямому файлу альбома*
12. *\* @param User $user пользователь, которому точно доступен альбом*
13. *\* @return string сигнатура*
14. *\*/*
15. public function getSign(User $user) {
16. return static::*getSignStatic*($this->hash, $user);
17. }
18. */\*\* Генерация или возвращения из кеша сигнатуры доступа к прямому файлу альбома*
19. *\* @param string $albumHash хеш закрытого альбома, к которому выдаётся доступ*
20. *\* @param User $user пользователь, которому точно доступен альбом*
21. *\* @return string сигнатура*
22. *\*/*
23. public static function getSignStatic(string $albumHash, User $user): string
24. {
25. *// Построение ключа кеша и попытка получения из кеша сигнатуры*
26. $cacheKey = static::*buildSignCacheKey*($albumHash, $user->id);
27. $cachedSign = Cache::get($cacheKey);
28. *// Если в кеше оказалось - возвращаем, иначе создаём*
29. if ($cachedSign) return $user->id .'\_'. $cachedSign;
30. *// Текущая дата в формате год-месяц-день (сигнатура перестанет быть действительной если ройдут сутки)*
31. $currentDay = date('Y-m-d');
32. *// Токен пользователя (сигнатура перестанет быть действительной если пользователь выйдет со всех устройств)*
33. $userToken = $user->tokens[0]->value;
34. *// Сырая строка для сигнатуры*
35. $string = $userToken . $currentDay . $albumHash;
36. *// Сигнатура из захешированной строки, и закодированной из бинарного в base64 для передачи*
37. $signCode = base64\_encode(Hash::make($string));
38. *// Сохранение в кеш*
39. Cache::put($cacheKey, $signCode, static::*SIGN\_CACHE\_TTL*);
40. return $user->id .'\_'. $signCode;
41. }
42. */\*\* Проверка в кеше, что такая сигнатура валидная*
43. *\* @param string $sign полученная от пользователя сигнатура*
44. *\* @return bool 1 - доступ разрешён, 2 – запрещён*
45. *\*/*
46. public function checkSign(string $sign): bool {
47. return static::*checkSignStatic*($this->hash, $sign);
48. }
49. */\*\* Проверка в кеше, что такая сигнатура валидная*
50. *\* @param string $albumHash хеш альбома, в котором находится запрашиваемый медиа-файл*
51. *\* @param string $sign полученная от пользователя сигнатура*
52. *\* @return bool 1 - доступ разрешён, 2 – запрещён*
53. *\*/*
54. public static function checkSignStatic(string $albumHash, string $sign): bool
55. {
56. *// Попытка получить ИД пользователя отдельно от сигнатуры*
57. try {
58. $signExploded = explode('\_', $sign);
59. $userId = $signExploded[0];
60. $signCode = $signExploded[1];
61. }
62. catch (\Exception $e) {
63. return false;
64. }
65. *// Сборка ключа кеша*
66. $cacheKey = static::*buildSignCacheKey*($albumHash, $userId);
67. *// Если в кеше такая сигнатуры есть, то разрешаем доступ*
68. $cachedSign = Cache::get($cacheKey);
69. if ($cachedSign === $signCode) return true;
70. *// Если пользователя с таким ИД нет, то доступ запрещён*
71. $user = User::*find*($signExploded[0]);
72. if (!$user)
73. return false;
74. *// Сырые данные (текущий день и токен пользователя) для получения сырой строки сигнатуры*
75. $currentDay = date("Y-m-d");
76. $string = $user->tokens[0]->value . $currentDay . $albumHash;
77. *// Если хеши сигнатур совпадают, то доступ разершён, иначе запрещён*
78. $allow = Hash::check($string, base64\_decode($signExploded[1]));
79. Cache::put($cacheKey, $signCode, static::*SIGN\_CACHE\_TTL*); *// Сохранение в кеш*
80. return $allow;
81. }

## Контроллеры

Вместо того чтобы определять всю логику обработки запросов как замыкания в файлах маршрутов, мы организуем это поведение с помощью классов «контроллеров». Контроллеры могут сгруппировать связанную логику обработки запросов в один класс.

Контроллеры в Laravel отвечают за обработку входящих запросов, выполнение бизнес-логики и возврат соответствующих ответов. Они часто используются для управления данными, взаимодействия с моделями и отображения информации через API или веб-страницы. Контроллеры могут включать методы для обработки различных HTTP-запросов, таких как GET, POST, PUT, PATCH, DELETE.

Примерно такие базовые методы могут иметь контроллеры:

1. *index():* Получает все или страницу элеметов. Возвращается коллекцию, форматированную ресурсным JSON-генератором (Resource).
2. *store(CustomRequest $request):* Cоздания нового элемента и возвращение как форматированного ресурса. Обычно используется правила валидация (Request), которые возвращают 422 с ошибками для каждого поля.
3. *show(MyModel $myModel):* Показывает подробную информацию о элементе в виде ресурса. Если параметр типизирован, название параметра и поля в маршруте совпадают — автоматически производится поиск модели и возвращается 404 если не найдено.
4. *update(CustomRequest $request, MyModel $myModel):* Обновления полей элемента. Возвращает получившийся ресурс.
5. *destroy(MyModel $myModel):* Удаления элемента.

В проекте 9 контроллеров: AlbumController, MediaController, UserController, AccessController, InvitationController, ReactionController, TagController, ViewController. Описан будет только два метода из MediaController.

### MediaController

Контроллер имеет следующие методы:

1. *upload(UploadRequest $request, $albumHash):* Массовое сохранение медиафайлов альбома в файловую систему и создание записей в базе данных. Код представлен на листинге 9.

Листинг 9 – Метод upload из app/Http/Controllers/MediaController.php

1. public function upload(UploadRequest $request, $albumHash)
2. {
3. $user = $request->user();
4. $album = Album::*getByHash*($albumHash);
5. $files = $request->file('medias');
6. *// Получаем сколько сейчас занято на диске, если достигли предела - выводим ошибку*
7. $spaceInfo = SpaceInfoCacheable::*getCached*();
8. if ($spaceInfo->isUploadDisabled)
9. throw new ApiException(400, 'Server in read-only mode');
10. *// Получаем сколько сейчас весят пользовательское медиа и какой лимит по загрузкам*
11. $currentStorageSize = $user->quotaUsed();
12. $maxStorageSize = $user->quotaTotal();
13. *// Путь альбома для сохранения*
14. $pathToSave = "users/$user->id/$album->path";
15. *// Разрешённые расширения*
16. $allowedImageExtensions = config('setups.allowed\_image\_extensions');
17. $allowedVideoExtensions = config('setups.allowed\_video\_extensions');
18. $allowedAudioExtensions = config('setups.allowed\_audio\_extensions');
19. *// Массивы для ответа*
20. $errored = [];
21. $successful = [];
22. *// Драйвер FFProbe для получения информации из видео и аудио*
23. $probe = FFProbe::*create*();
24. foreach ($files as $key => $pictureInRequests) {
25. $file = $request->file("medias.$key.file");
26. $date = $pictureInRequests['date'] ?? now();
27. $filename = $pictureInRequests['name'] ?? $file->getClientOriginalName();
28. $extReq = pathinfo($filename, *PATHINFO\_EXTENSION*);
29. $extReal = $file->guessExtension() ?? $file->extension();
30. if ($extReal != null && $extReal != $extReq)
31. $filename .= ".$extReal";
32. *// Пропуск файла с не найденного в разрешённых расширению, иначе определяется тип*
33. $extDown = strtolower($extReal);
34. if (in\_array($extDown, $allowedImageExtensions))
35. $type = 'image';
36. else
37. if (in\_array($extDown, $allowedVideoExtensions))
38. $type = 'video';
39. Else
40. if (in\_array($extDown, $allowedAudioExtensions))
41. $type = 'audio';
42. else {
43. $errored[] = [
44. 'name' => $filename,
45. 'code' => 1409,
46. 'message' => "\".$extDown\" not support/allowed",
47. ];
48. continue;
49. }
50. *// Проверяем, превышает ли размер лимит*
51. $filesize = $file->getSize();
52. if ($currentStorageSize + $filesize >= $maxStorageSize) {
53. $errored[] = [
54. 'name' => $filename,
55. 'code' => 1410,
56. 'message' => 'Storage limit reached',
57. 'debug1' => $currentStorageSize + $filesize >= $maxStorageSize,
58. 'debug2' => $currentStorageSize,
59. 'debug3' => $filesize,
60. 'debug4' => $maxStorageSize,
61. ];
62. continue;
63. }
64. *// Проверка, существует ли картинка с таким хешем и пропускаем если да*
65. $tmpPath = $file->getRealPath();
66. $hash = hash\_file('xxh3', $tmpPath);
67. $existedImageByHash = Media
68. ::*where*('album\_id', $album->id)
69. ->where('hash', $hash)
70. ->first();
71. if ($existedImageByHash) {
72. $errored[] = [
73. 'name' => $filename,
74. 'code' => 1203,
75. 'message' => 'Already exist with this hash',
76. 'media' => ImageResource::*make*($existedImageByHash),
77. ];
78. continue;
79. }
80. *// Сохранение в другом названии, если есть какой-то запрещённый символ в названии*
81. if (!preg\_match('~^[^/\\\\:\*?"<>|]+$~u', $filename))
82. $filename = $hash . ".$extReal";
83. *// Проверка, существует ли картинка с таким именем и модифицируем имя счётчиком если да*
84. $counter = 1;
85. $extension = pathinfo($filename, *PATHINFO\_EXTENSION*);
86. $nameWithoutExt = pathinfo($filename, *PATHINFO\_FILENAME*);
87. $filenameValid = $filename;
88. while (Media
89. ::*where*('album\_id', $album->id)
90. ->where('name', $filenameValid)
91. ->exists()
92. ) {
93. $counter++;
94. $filenameValid = "$nameWithoutExt ($counter).$extension";
95. }
96. *// Сохранение файла в хранилище*
97. $file->storeAs($pathToSave, $filenameValid);
98. *// Получение размеров картинки, если нет размеров (не получили) — пропуск*
99. if ($type === 'image') {
100. $sizes = getimagesize($file); *// TODO: на перевёрнутых JPG даёт те же размеры*
101. if (!$sizes) {
102. $errored[] = [
103. 'name' => $filename,
104. 'code' => 1413,
105. 'message' => 'Cannot grab size of image',
106. 'media' => MediaResource::*make*($existedImageByHash),
107. ];
108. continue;
109. }
110. }
111. if ($type !== 'audio')
112. $probeInfo = $probe->streams($file)->videos()->first();
113. Else
114. $probeInfo = $probe->streams($file)->audios()->first();
115. $steamContentFields = [];
116. if ($type === 'image' && ($probeInfo?->get('duration\_ts') ?? 0) > 1)
117. $type = 'imageAnimated';
118. if ($type !== 'image') {
119. if (!($probeInfo?->get('duration\_ts'))) {
120. $errored[] = [
121. 'name' => $filename,
122. 'code' => 1413,
123. 'message' => 'Cannot grab duration\_ts',
124. 'media' => MediaResource::*make*($existedImageByHash),
125. ];
126. continue;
127. }
128. $steamContentFields['codec\_name'] = $probeInfo->get('codec\_name');
129. $number = $probeInfo->get('duration');
130. if (!str\_contains($number, '.')) $number .= '.000';
131. [$intPart, $decimalPart] = explode('.', $number, 2);
132. $decimalPart = substr($decimalPart . '000', 0, 3);
133. $steamContentFields['duration\_ms'] = (int)($intPart . $decimalPart);
134. if ($type !== 'audio') {
135. $sizes = [
136. $probeInfo->get('width'),
137. $probeInfo->get('height')
138. ];
139. $framerate = array\_map('intval',
140. explode('/', $probeInfo->get('avg\_frame\_rate'))
141. );
142. $steamContentFields['avg\_frame\_rate\_num'] = $framerate[0];
143. $steamContentFields['avg\_frame\_rate\_den'] = $framerate[1];
144. $steamContentFields['frame\_count'] = (int)$probeInfo->get('nb\_frames');
145. }
146. *// Сохранение записи в БД*
147. $mediaDB = Media::*create*([
148. 'album\_id' => $album->id,
149. 'name' => $filenameValid,
150. 'type' => $type,
151. 'hash' => $hash,
152. 'date' => $date,
153. 'size' => $filesize,
154. 'width' => $sizes[0],
155. 'height' => $sizes[1],
156. ...$steamContentFields,
157. ]);
158. *// Сохранение успешного ответа API*
159. $successful[] = MediaResource::*make*($mediaDB);
160. }
161. return response([
162. 'sign' => $album->getSign($user),
163. 'successful' => $successful,
164. 'errored' => $errored,
165. ]);
166. }

1. index(AlbumMediasRequest $request, $albumHash): Страничный список медиафайлов в альбоме с всевозможной фильтрацией, сортировкой и переключением видимости вложенных медиа.
2. *show($albumHash, $mediaHash):* Информация о медиа.
3. *thumb($albumHash, $mediaHash, $orientation = null, $size = null, $animated = null):* Возвращение (или генерация, если не существует) превью медиа. Использует Intervention Image для уменьшения или/и обрезки, FFProbe для получения вложенной обложки видео и FFMpeg для получение обложки или кадра. Требует сигнатуры. Код представлен на листинге 10.

Листинг 10 – Метод thumb из app/Http/Controllers/MediaController.php

1. */\*\* Создание превью-файла (в виде уменьшенной картинки или сжатого видео)*
2. *\* @param string $albumHash Хеш альбома*
3. *\* @param string $mediaHash Хеш медиа*
4. *\* @param ?string $dimension Ось (размера)*
5. *\* @param ?int $size Размер*
6. *\* @param ?string $animated Анимированное?*
7. *\* @return RedirectResponse|BinaryFileResponse Редирект на новый путь / Сгенерированный превью-*
8. *Айл*
9. *\*/*
10. public function thumb(
11. string $albumHash,
12. string $mediaHash,
13. string $dimension = null,
14. int $size = null,
15. string $animated = null
16. ) {
17. *// Проверка доступа по токену в ссылке*
18. $sign = request()->sign;
19. if (
20. (Album::*getAccessLevelCachedByHash*($albumHash, null) === AccessLevel::*None*) &&
21. !($sign && Album::*checkSignStatic*($albumHash, $sign))
22. ) {
23. *// Проверка доступа по токену в заголовках*
24. $media = Media::*getByHashOrAlias*($albumHash, $mediaHash);
25. if (Album::*getAccessLevelCachedByHash*($albumHash, request()->user()) === AccessLevel::*None*)
26. throw new ApiException(403, 'Forbidden for you');
27. }
28. *// Проверка наличия превью в файлах*
29. $dirname = "thumbs/{$dimension}{$size}{$animated}";
30. $thumbPath = "$dirname/$mediaHash" . (!$animated ? '.webp' : '.mp4');
31. if (!Storage::exists($thumbPath)) {
32. *// Проверка запрашиваемого размера и редирект, если не прошло*
33. $askedSize = $size;
34. $allowedSizes = [144, 240, 360, 480, 720, 1080];
35. $allowSize = false;
36. foreach ($allowedSizes as $allowedSize) {
37. if ($size <= $allowedSize) {
38. $size = $allowedSize;
39. $allowSize = true;
40. break;
41. }
42. }
43. if (!$allowSize) $size = $allowedSizes[count($allowedSizes)-1];
44. if ($askedSize != $size)
45. return redirect()->route('get.media.thumb', [
46. $albumHash, $mediaHash, $dimension, $size, $animated
47. ])->header('Cache-Control', ['max-age=86400', 'private']);
48. *// Создание превью*
49. $media = $media ?? Media::*getByHashOrAlias*($albumHash, $mediaHash);
50. $type = $media->type;
51. if ($animated && ($type !== 'imageAnimated' && $type !== 'video'))
52. return redirect()->route('get.media.thumb', [
53. $albumHash, $mediaHash, $dimension, $size
54. ])->header('Cache-Control', ['max-age=86400', 'private']);
55. $mediaPath = Storage::path('medias'. $media->album->path . $media->name);
56. if ($type === 'image')
57. *// Создание превью как mp4 видео*
58. $imagePath = $mediaPath;
59. else if ($type === 'video' || $type === 'imageAnimated') {
60. if (!$animated) {
61. $framePath = Storage::path("thumbs/frames/$media->hash.png");
62. *// Использование оригинального превью/кадра видео*
63. if (file\_exists($framePath))
64. $imagePath = $framePath;
65. else
66. $imagePath = StreamHelper::*extractPreview*($mediaPath, $media);
67. }
68. }
69. else
70. throw new ApiException(500, "Media type \"$type\" not supported");
71. if ($animated) {
72. *// Создание превью как mp4 видео*
73. GeneratePreviewVideo::*dispatch*($media, $mediaPath, $thumbPath, $dimension, $size);
74. }
75. else {
76. *// Создание превью как webp картинку*
77. $thumb = *Intervention*::read($imagePath);
78. *// Ось для уменьшения размера (по умолчанию по двум осям, т.е. в квадрат)*
79. switch ($dimension) {
80. case 'w':
81. $thumb->scaleDown(width: $size);
82. break;
83. case 'h':
84. $thumb->scaleDown(height: $size);
85. break;
86. default:
87. $thumb->coverDown($size, $size);
88. break;
89. }
90. if (!Storage::exists($dirname))
91. Storage::makeDirectory($dirname);
92. *// Сохранение файла*
93. $thumb->toWebp(90)->save(Storage::path($thumbPath));
94. unset($thumb);
95. }
96. }
97. else
98. if ($animated && Storage::fileSize($thumbPath) < 1) {
99. *// Ожидание создания превью*
100. return response([
101. 'message' => 'Pending create video preview'
102. ], 202);
103. }
104. *// Возвращаем превью-файл с заголовком для кеширования на прокси/клиенте*
105. return response()->file(Storage::path($thumbPath),
106. ['Cache-Control' => ['max-age=86400', 'private']]);
107. }
108. *orig($albumHash, $mediaHash):* Возвращение медиафайла. Требует сигнатуры.
109. *download ($albumHash, $mediaHash):* Возвращение медиафайла как attachment. Требует сигнатуры.
110. *update($albumHash, $mediaHash):* Обновление информации о медиа.
111. *destroy($albumHash, $mediaHash):*Удаление медиафайла из ФС и записи из БД.

## Посредник

Посредники (Middleware) в Laravel — это механизм обработки HTTP-запросов перед их передачей в контроллер. Они выполняют роль фильтров, которые могут изменять запрос или отклонять его, если он не соответствует определённым условиям.

Посредники полезны для:

* Аутентификации (проверка, авторизован ли пользователь).
* Авторизации (разрешён ли доступ к определённому ресурсу).
* Логирования и мониторинга запросов.
* Фильтрации и изменения запроса перед обработкой контроллером.

Каждый посредник в Laravel — это обычный PHP-класс, реализующий метод handle($request, Closure $next), где $next($request) передаёт запрос дальше. Если next() не вызывается, запрос останавливается.

В листинге 11 показан посредник TokenAuth, который проверяет, имеет ли пользователь доступ к альбому. Также вкладывает пользователя в $request. Для не обязательной авторизации может применяться как token.auth:guest, при этом вкладывая пользователя если токен был получен.

Листинг 11 – Посредник проверки доступа к альбому app/Http/Middleware/TokenAuth.php

1. public function handle(Request $request, Closure $next, $allowedLevel = 'user')
2. {
3. *// Получаем значение токена из запроса*
4. $tokenValue = $request->bearerToken();
5. if (!$tokenValue) {
6. if (!($allowedLevel === 'guest'))
7. *// Если токена нет, но заданный уровень доступа НЕ "гость" —— выводить 401 ошибку*
8. throw new ApiException(401, 'Token not provided');
9. }
10. else {
11. *// Получение пользователя*
12. $user = User::*getByToken*($tokenValue);
13. if (!$user)
14. *// Если пользователь не настоящий, но заданный уровень доступа не "гость" —— выводить 401 ошибку*
15. throw new ApiException(401, 'Token corrupted');
16. if (!$user->is\_admin) {
17. if ($allowedLevel === 'admin')
18. *// Если пользователь не админ, но заданный уровень доступа "администратор" —— выводить 403 ошибку*
19. throw new ApiException(403, 'Admin rights required');
20. if ($allowedLevel === 'owner') {
21. $albumHash = $request->route('album\_hash');
22. $album = Album::*getByHash*($albumHash);
23. if ($album->owner\_user\_id !== $user->id)
24. *// Если пользователь не владелец, но заданный уровень доступа "владелец" —— выводить 403 ошибку*
25. throw new ApiException(403, 'You not a owner of this album');
26. }
27. }
28. *// Запись пользователя в запрос для последующих обработок в контроллерах*
29. $request->setUserResolver(function () use ($user) {
30. return $user;
31. });
32. }
33. return $next($request);
34. }

## Помощники

Помощники (Helpers) в Laravel — это набор глобальных функций, предоставляющих удобные и быстрые способы выполнения распространённых задач, таких как работа с массивами, строками, маршрутами, файлами и другими аспектами приложения. Они упрощают разработку, сокращают количество повторяющегося кода и позволяют выполнять операции в более лаконичной форме.

Помощник StreamHelper помогает работать со потоковыми файлами для генерации статичного и анимированного превью.

Получение статичного превью производится проверкой через FFProbe на наличие потока с attached\_pic (вложенной обложи). Если нет такого потока — берётся кадр с 10-й секунды видео (с первого кадра, если видео короче 20 секунд). В обоих случаях используется FFMpeg для извлечения картинки на превью. Код представлен на листинге 12.

Листинг 12 – Часть кода про получение статичного превью из app/Helpers/StreamHelper.php

1. */\*\* Главный метод: сохраняет обложку или кадр в Storage::path("thumbs/frames/{hash}")*
2. *\* @param string $videoPath Путь до медиа*
3. *\* @param Media $video Модель медиа*
4. *\* @return string Выходной путь*
5. *\*/*
6. public static function extractPreview(string $videoPath, Media $video): ?string
7. {
8. $outputDir = Storage::path('thumbs/frames');
9. Storage::makeDirectory('thumbs/frames');
10. $ffprobe = FFProbe::*create*();
11. *// Все потоки в медиа*
12. $streams = $ffprobe->streams($videoPath);
13. *// Поток с attached\_pic (обложкой)*
14. $coverStream = null;
15. *// Ищем поток с обложкой*
16. foreach ($streams as $stream) {
17. if ($stream->get('codec\_type') === 'video' &&
18. ($stream->get('disposition')['attached\_pic'] ?? false)) {
19. $coverStream = $stream;
20. break;
21. }
22. }
23. if ($coverStream)
24. *// Если на видео уже есть обложка, то извлечь*
25. return self::*extractAttachedCover*($videoPath, $outputDir, $coverStream, $video->hash);
26. else
27. *// Иначе взять кадр с 10-й секунды (или первый кадр, если видео меньше 20 секунд)*
28. return self::*extractVideoFrame*($videoPath, $outputDir, $video->duration, $video->hash);
29. }
30. */\*\* Извлекает встроенную обложку с оригинальным расширением*
31. *\* @param string $videoPath Путь до медиа*
32. *\* @param string $outputDir Выходной путь*
33. *\* @param Stream $stream Поток*
34. *\* @param string $hash Хеш медиа*
35. *\* @return string Выходной путь*
36. *\*/*
37. public static function extractAttachedCover(string $videoPath, string $outputDir, Stream
38. stream, string $hash): string
39. {
40. $outputFile = "{$outputDir}/{$hash}.png";
41. FFMpeg::fromDisk('local')
42. ->open(self::*relativePath*($videoPath))
43. ->addFilter([
44. '-map', '0:v',
45. '-map', '-0:V',
46. ])
47. ->export()
48. ->save(self::*relativePath*($outputFile));
49. return $outputFile;
50. }
51. */\*\* Извлекает кадр с 10-й секунды (если длительность ≥ 20 сек), иначе — первый кадр. PNG.*
52. *\* @param string $videoPath Путь до медиа*
53. *\* @param string $outputDir Выходной путь*
54. *\* @param int $durationMs Длительность медиа*
55. *\* @param string $hash Хеш медиа*
56. *\* @return string Выходной путь*
57. *\*/*
58. public static function extractVideoFrame(string $videoPath, string $outputDir, int
59. durationMs, string $hash): string
60. {
61. $seconds = ($durationMs / 1000) >= 20 ? 10 : 0;
62. $outputFile = "{$outputDir}/{$hash}.png";
63. FFMpeg::fromDisk('local')
64. ->open(self::*relativePath*($videoPath))
65. ->getFrameFromSeconds($seconds)
66. ->export()
67. ->save(self::*relativePath*($outputFile));
68. return $outputFile;
69. }

Получение анимированного превью производится через перекодирование видео с меньшим разрешением, битрейтом (бит/с) и фреймрейтом (кадров в секунду). Для ускорения используется кодировщик из видеокарты NVIDIA: NVENC. Так же есть резервный кодировщик на мощностях центрального процессора. Код представлен на листинге 13.

Листинг 13 – Часть кода про получение анимированного превью из app/Helpers/StreamHelper

1. */\*\* Формула оптимального битрейта*
2. *\* @param int $width Ширина*
3. *\* @param int $height Высота*
4. *\* @param float $framerate FPS*
5. *\* @param float $coef Коэффицент качества*
6. *\* @return int Битрейт*
7. *\*/*
8. public static function calcKiloBitrate(int $width, int $height, float $framerate, float
9. coef = 0.1): int {
10. return $width \* $height \* $framerate \* $coef / 1000;
11. }
12. */\*\* Создание видео-превью для видео или анимированной картинки*
13. *\* (без аудио, уменьшение до конкретного размера, битрейта и фреймрейта)*
14. *\* @param Media $media Модель медиа*
15. *\* @param string $mediaPath Путь до оригинала медиа*
16. *\* @param string $outputPath Выходной путь*
17. *\* @param string $dimension Ось (размера)*
18. *\* @param int $size Размер*
19. *\* @param float $duration Длительность*
20. *\* @param float $maxFramerate Максимально допустимый FPS если соблюдён битрейт*
21. *\* @param float $ecoFramerate Если заданного битрейта оказалось мало, то попытка пересчитать на экономный FPS*
22. *\* @param float $bitrateCoef Коэффициент качества*
23. *\* @param float $maxBitrate Максимально допустимый битрейт*
24. *\* @param float $minBitrate Минимально допустимый битрейт*
25. *\* @return string Выходной путь*
26. *\*/*
27. public static function genPreviewVideo(
28. Media $media,
29. string $mediaPath,
30. string $outputPath,
31. string $dimension = 'q',
32. int $size = 240,
33. float $duration = 60,
34. float $maxFramerate = 60,
35. float $ecoFramerate = 30,
36. float $bitrateCoef = 0.1,
37. float $maxBitrate = 8000,
38. float $minBitrate = 300,
39. ): string {
40. *// Путь до временного файла*
41. $tempFile = "temp/thumb\_{$media->hash}\_{$dimension}{$size}a.mp4";
42. *// Размеры превью*
43. [$width, $height] = ThumbHelper::*calcDimensions*($media->width, $media->height, $dimension, $size);
44. *// Фильтры масштабирования*
45. $filter = match ($dimension) {
46. 'q' => implode(',', [
47. "scale='if(gt(a,1),{$size},-2)':'if(gt(a,1),-2,{$size})'",
48. "crop='trunc(in\_w/2)\*2':'trunc(in\_h/2)\*2'"
49. ]),
50. default => "scale=$width:$height",
51. };
52. *// Кадров в секунду оригинала и требуемое*
53. $origFramerate = $media?->avg\_frame\_rate\_den
54. ? $media->avg\_frame\_rate\_num / $media->avg\_frame\_rate\_den
55. : $maxFramerate;
56. $targetFramerate = min($origFramerate, $maxFramerate);
57. *// Параметры кодировщика*
58. $params = [
59. '-an', *// Убрать аудио*
60. '-t', $duration, *// Длительность*
61. '-pix\_fmt', 'yuv420p', *// Кодировка цвета и пикселей*
62. ];
63. if ($origFramerate > $maxFramerate) {
64. $params[] = '-r'; *// Урезание кадров в секунду*
65. $params[] = $maxFramerate;
66. }
67. *// Скорость потока данных*
68. $targetBitrate = self::*calcKiloBitrate*($width, $height, $targetFramerate, $bitrateCoef);
69. if ($targetBitrate > $maxBitrate)
70. $targetBitrate = self::*calcKiloBitrate*($width, $height, $ecoFramerate, $bitrateCoef);
71. if ($targetBitrate < $minBitrate)
72. $targetBitrate = $minBitrate;
73. else
74. if ($targetBitrate > $maxBitrate)
75. $targetBitrate = $maxBitrate;
76. if ($media?->frame\_count && $media->frame\_count < 30)
77. $targetBitrate = $maxBitrate;
78. *// Полный быстрый NVIDIA кодировщик*
79. $format = new NVENCFormat();
80. $format
81. ->setKiloBitrate($targetBitrate)
82. ->setInitialParameters([
83. '-hwaccel', 'cuda',
84. ])
85. ->setAdditionalParameters($params);
86. *// Преобразование*
87. try {
88. FFMpeg::fromDisk('local')
89. ->open(self::*relativePath*($mediaPath))
90. ->addFilter(fn($filters) => $filters->custom($filter))
91. ->export()
92. ->inFormat($format)
93. ->save($tempFile);
94. }
95. catch (\Exception $e) {
96. Log::error('NVENC: '. $e->getMessage());
97. *// Резервный кодировщик*
98. $formatSecond = new X264();
99. $formatSecond
100. ->setKiloBitrate($targetBitrate)
101. ->setAdditionalParameters($params);
102. *// Повторное преобразование*
103. FFMpeg::fromDisk('local')
104. ->open(self::*relativePath*($mediaPath))
105. ->addFilter(fn($filters) => $filters->custom($filter))
106. ->export()
107. ->inFormat($formatSecond)
108. ->save($tempFile);
109. }
110. *// Больше не временный файл, перенос в папку с превью*
111. Storage::move($tempFile, $outputPath);
112. return $outputPath;
113. }
114. }

## Консольные команды

Консольные команды в Laravel — это мощный инструмент для выполнения фоновых задач, автоматизации рутинных операций и управления приложением через командную строку (CLI). В отличие от HTTP-запросов, которые обрабатываются через маршруты и контроллеры, консольные команды запускаются вручную или по расписанию (например, с помощью Cron) и выполняют длительные или периодические задачи, такие как:

* Обработка данных (импорт/экспорт, генерация отчётов).
* Очистка кеша, логов и устаревших записей.
* Запуск фоновых задач (парсинг, рассылка писем, уведомления).
* Администрирование приложения (миграции, сидирование базы данных).

Команда app:index предназначена для сканирования файловой структуры корневого серверного медиа-альбомов, сравнения её с записями в базе данных и автоматического исправления расхождений.

Команда поддерживает опции:

* --start-from <id|hash|alias> (-s): начать индексацию с определённого альбома (по ИД, хешу или алиасу).
* --auto-destroy (-d): автоматически удалять не найденные записи без подтверждения.

В начале команда исправляет иерархию альбомов с помощью Album::fixTree() для корректного учёта вложенности. Затем получает все альбомы из БД, сортируя их по глубине вложенности (path) для последовательной обработки. После получает допустимые расширения файлов из файла конфигурации (setups.allowed\_\*\_extensions). Также если был передан --start-from — ищет альбом и начинает индексацию с него.

Для каждого альбома команда (см. лист. 14):

* Проверяет существование папки в хранилище. Если папка удалена: предлагает удалить альбом из БД (если не активирован --auto-destroy).
* Сравнивает подпапки с записями в БД: Новые папки добавляются как дочерние альбомы (с генерацией хеша). Отсутствующие папки отмечаются для удаления (с подтверждением).
* Фиксирует изменения в иерархии, обновляя связи parent\_album\_id.

Листинг 14 – Часть кода индексации про альбомы из app\Console\Commands\StoreIndex.php

1. *// Проход по альбомам*
2. while ($albums->count() > $currentAlbumKey) {
3. $currentAlbum = $albums[$currentAlbumKey];
4. $path = Storage::path("medias$currentAlbum->path");
5. $currentAlbumKey++;
6. $this->line('<fg=gray;options=bold>['.static::*counter*($currentAlbumKey, $albums->count())
7. .'] #' . static::*formatNumber*($currentAlbum->id)
8. ." <fg=yellow;options=bold>$currentAlbum->name</> "
9. ." <bg=black;fg=white;href=". url('../album/'. $currentAlbum->hash) ."> 🌐
10. .($currentAlbum->alias ?? $currentAlbum->hash)." </> "
11. ." <bg=gray;fg=black;href=file:///$path> 📁 $currentAlbum->path </></> "
12. );
13. *// Попытка получить все дочерние директории в альбоме*
14. try {
15. $folders = File::directories($path);
16. }
17. catch (DirectoryNotFoundException $e)
18. {
19. *// Альбом не найден, спрашиваем "удалить ли", если не было передано опции авто-удаления*
20. $this->error(' DELETED ');
21. if (!Album::*find*($currentAlbum->id)) continue;
22. if ($this->option('auto-destroy') ||
23. $this->confirm("Do you wish remove not founded albums from DB? ["
24. .$currentAlbum->children->count()." subalbums & ". $currentAlbum->medias-
25. count() ." medias known]")
26. ) Album::*destroy*($currentAlbum->id);
27. continue;
28. }
29. *// Дочерние альбомы из БД*
30. $albumChildren = $albums->where('parent\_album\_id', $currentAlbum->id);
31. *// Отображаем сколько в файловой системе и в базе данных альбомов*
32. $this->line('Checking folders in album ['
33. . count($folders) .' in FS / '
34. . $albumChildren->count() .' in DB]'
35. );
36. $newAlbums = [];
37. *// Проход по папкам альбома (дочерние альбомы)*
38. foreach ($folders as $folder) {
39. $childPath = $currentAlbum->path . basename($folder) .'/';
40. $basename = basename($childPath);
41. *// Проверка наличия в БД вложенного альбома, создание если нет*
42. $key = $albumChildren->search(fn ($a) => $a['path'] === $childPath);
43. if ($key !== false) {
44. $albumChild = $albumChildren[$key];
45. $this->line(
46. " <fg=gray;href=". url("../album/$albumChild->hash") .">$albumChild->hash</> "
47. ."<fg=gray;href=file:///". Storage::path("medias$childPath") .">$basename/</> "
48. );
49. $albumChildren->forget($key);
50. }
51. else {
52. *// Создание, прикрепление как дочернего (appendToNode) и сохранение в БД*
53. $hash = Str::*random*(25);
54. $childAlbum = Album::*create*([
55. 'name' => $basename,
56. 'path' => $childPath,
57. 'hash' => $hash,
58. ]);
59. $childAlbum->appendToNode($currentAlbum);
60. $childAlbum->save();
61. $newAlbums[] = $childAlbum;
62. $this->info('<fg=green>+ '
63. ."<fg=green;href=". url("../album/$hash") .">$hash</> "
64. ."<fg=green;href=file:///". Storage::path("medias$childPath")
65. ">$basename/</></> "
66. );
67. }
68. }
69. *// Отображение всех не найденных альбомов*
70. foreach ($albumChildren as $key => $notFoundedAlbum) {
71. $this->line('<fg=red>- '
72. ."<fg=red;href=". url("../album/$notFoundedAlbum->hash") .">$notFoundedAlbum-
73. hash</> "
74. ."<fg=red;href=file:///". Storage::path('medias'). $notFoundedAlbum->path
75. '>'.basename($notFoundedAlbum->path).'/</></> '
76. );
77. $albums->forget($key);
78. }
79. $albums->splice($currentAlbumKey, 0, $newAlbums);
80. *// Спрашиваем "удалить ли не найденные альбомы", если не было передано опции авто-удаления*
81. $notFoundedCount = $albumChildren->count();
82. if ($this->option('auto-destroy') || (
83. $notFoundedCount &&
84. $this->confirm("Do you wish remove not founded albums from DB? [$notFoundedCount]")
85. )) {
86. Album::*destroy*($albumChildren->pluck('id')->toArray());
87. }
88. *...*

Для каждого файла в альбоме команда (см. лист. 15):

* Проверяет расширение (избегая недопустимых типов).
* Сравнивает с записями в БД:
  + Если файл существует – проверяет хеш (XXH3) и метаданные.
  + Если файл новый – создаёт запись в БД с данными (размер, длительность, кодек и т. д.).
  + Если файл дубликат – связывает его с оригиналом через MediaDuplica.
* Удаляет "потерянные" медиа:
  + Файлы, отсутствующие в хранилище, но сохранённые в БД.
  + Дубликаты с некорректными ссылкам

Листинг 15 – Часть кода индексации про медиа из app\Console\Commands\StoreIndex.php

1. *// Получение файлов альбома*
2. $start = now();
3. $glob = glob("$path\*", *GLOB\_MARK*);
4. $timeGlob = $start->diffInMilliseconds();
5. $files = array\_filter($glob, fn ($path) => !in\_array($path[-1], ['/', '\\']));
6. *// Получение имеющихся медиа в БД*
7. $mediasInDB = $currentAlbum->medias()->with('duplicas')->get();
8. *// Объединение медиа и их дубликатов в единый массив*
9. $medias = $mediasInDB->flatMap(function ($media) {
10. $origMedia = $media->toArray();
11. return array\_merge(
12. [$origMedia],
13. $media->duplicas->map(fn ($duplica) =>
14. array\_merge($origMedia, [
15. 'name' => $duplica->name,
16. 'origId' => $media->Id,
17. 'origName' => $media->name,
18. ])
19. )->toArray()
20. );
21. })->toArray();
22. $filesCount = count($files);
23. $this->line('Checking medias in album ['
24. . $filesCount .' files in FS / '
25. . count($medias) ." in DB] [glob $timeGlob ms]"
26. );
27. $notFoundedMedias = $medias;
28. *// Массивы для поиска*
29. $mediasNames = array\_column($medias, 'name');
30. $mediasHashes = array\_column($medias, 'hash');
31. *// Проход по файлам*
32. foreach ($files as $i => $file) {
33. $name = basename($file);
34. $counter = '['.static::*counter*($i+1, $filesCount).']';
35. $this->output->write(" $counter <href=file:///$file>$name</>");
36. try {
37. *// Проверка наличия в БД медиа по названию, если есть — пропуск*
38. $key = array\_search($name, $mediasNames);
39. if ($key !== false) {
40. $existMedia = $medias[$key];
41. $isDuplica = array\_key\_exists('origName', $existMedia);
42. $this->line("<fg=gray>\r"
43. .($isDuplica ? '↩ ' : ' ')
44. . $counter
45. .' '
46. .($isDuplica ? '' : '<fg=gray;href='
47. . url("api/albums/$currentAlbum->hash/medias/$existMedia[hash]/orig")
48. .">$existMedia[hash]</> "
49. )
50. ."<fg=gray;href=file:///$file>$name</>"
51. .($isDuplica ? '<fg=white> duplica of </>'
52. .'<fg=gray;href='. $path . $existMedia['origName'] .">$existMedia[origName]</>" : ''
53. )
54. .'</>'
55. );
56. unset($notFoundedMedias[$key]);
57. continue;
58. }
59. *// Отсекание не-медиа по расширению файла, определение типа*
60. $extension = pathinfo($file, *PATHINFO\_EXTENSION*);
61. $extension = strtolower($extension);
62. if (in\_array($extension, $allowedImageExtensions))
63. $type = 'image';
64. else
65. if (in\_array($extension, $allowedVideoExtensions))
66. $type = 'video';
67. else
68. if (in\_array($extension, $allowedAudioExtensions))
69. $type = 'audio';
70. else {
71. $this->line("<fg=blue>\r× "
72. . $counter
73. ." <fg=blue;href=file:///$file>$name</>"
74. ." \".$extension\" not support/allowed"
75. .'</>'
76. );
77. continue;
78. }
79. *// Отсекаем не совпадающие c определителем MIME по заголовкам файла*
80. $guessExtension = File::guessExtension($file);
81. if ($extension !== $guessExtension) {
82. $this->line("<fg=red>\r× "
83. . $counter
84. ." <fg=blue;href=file:///$file>$name</>"
85. ." \".$guessExtension\" is actual format"
86. .'</>'
87. );
88. continue;
89. }
90. *// Получение хеша медиа*
91. $hash = base64url\_encode(hash\_file('xxh3', $file, true));
92. *// Проверка наличия в БД медиа по хешу, если есть — проверяем в ФС*
93. $key = array\_search($hash, $mediasHashes);
94. if ($key !== false) {
95. *// Полное имя медиа-оригинала (из БД)*
96. $mediaFullName = $path . $medias[$key]['name'];
97. *// Проверка наличие медиа-оригинала в ФС ...*
98. $filesKey = array\_search($mediaFullName, $files);
99. if ($filesKey === false) {
100. *// Если нет — переименовываем в БД*
101. Media
102. ::*where*('id', $medias[$key]['id'])
103. ->update(['name' => $name]);
104. $this->line("<fg=yellow>\r→ "
105. . $counter
106. .' <fg=yellow;href='. url("api/albums/$currentAlbum->hash/medias/$hash/orig") .">$hash</>"
107. ." <fg=yellow;href=file:///$file>". $medias[$key]['name'] ."<fg=white> renamed to </>$name</>"
108. .'</>'
109. );
110. $medias[$key]['name'] = $name;
111. $mediasNames[$key] = $name;
112. unset($notFoundedMedias[$key]);
113. } else {
114. *// Если есть — создаём дубликат в БД (чтобы в следующий раз не хешировать)*
115. MediaDuplica::*create*([
116. 'media\_id' => $medias[$key]['id'],
117. 'name' => $name,
118. ]);
119. *// Записываем в массивы чтобы не наткнутся на повторы снова*
120. $medias[$key]['duplicas'][] = ['name' => $name];
121. $medias[] = array\_merge($medias[$key], [
122. 'name' => $name,
123. 'origId' => $medias[$key]['id'],
124. 'origName' => $medias[$key]['name']
125. ]);
126. $this->line("<fg=yellow>\r↩ "
127. . $counter
128. .' <fg=yellow;href='. url("api/albums/$currentAlbum->hash/medias/$hash/orig") .">$hash</>"
129. ." <fg=yellow;href=file:///$file>$name<fg=white> linked to </>". $medias[$key]['name'] .'</>'
130. .'</>'
131. );
132. }
133. *// И пропуск создания медиа в БД*
134. unset($notFoundedMedias[$key]);
135. continue;
136. }
137. *// Получение размеров медиа, если нет размеров (не получили) — пропуск*
138. if ($type === 'media') {
139. $sizes = getmediasize($file);
140. if (!$sizes) {
141. $this->line("<fg=red>\r× "
142. . $counter
143. ." <fg=blue;href=file:///$file>$name</>"
144. ." cannot get media sizes"
145. .'</>'
146. );
147. continue;
148. }
149. }
150. *// Получение информации об основном потоку*
151. if ($type !== 'audio')
152. $probeInfo = $probe->streams($file)->videos()->first();
153. else
154. $probeInfo = $probe->streams($file)->audios()->first();
155. $steamContentFields = [];
156. *// Если это картинка с длительностью, то это анимированная картинка*
157. if ($type === 'image' && ($probeInfo?->get('duration\_ts') ?? 0) > 1)
158. $type = 'imageAnimated';
159. *// Получение информации о главном потоке*
160. if ($type !== 'image') {
161. if (!($probeInfo?->get('duration\_ts'))) {
162. $this->line("<fg=red>\r× "
163. . $counter
164. ." <fg=blue;href=file:///$file>$name</>"
165. ." cannot get duration\_ts from $type"
166. .'</>'
167. );
168. continue;
169. }
170. $number = $probeInfo->get('duration');
171. if (!str\_contains($number, '.')) $number .= '.000';
172. [$intPart, $decimalPart] = explode('.', $number, 2);
173. $decimalPart = substr($decimalPart . '000', 0, 3);
174. $steamContentFields['duration\_ms'] = (int)($intPart . $decimalPart);
175. }
176. *// Создание в БД записи*
177. $mediaModel = Media::*create*([
178. 'album\_id' => $currentAlbum->id,
179. 'name' => $name,
180. 'type' => $type,
181. 'hash' => $hash,
182. 'date' => Carbon::*createFromTimestamp*(File::lastModified($file)),
183. 'size' => File::size($file),
184. 'width' => $sizes[0],
185. 'height' => $sizes[1],
186. ...$steamContentFields,
187. ]);
188. $this->line("<fg=green>\r+ "
189. . $counter
190. ." <fg=green;href="
191. . url("api/albums/$currentAlbum->hash/medias/$hash/orig")
192. .">$hash</> "
193. ."<fg=green;href=file:///$file>$name</></>"
194. );
195. *// Обновление массивов медиа альбома*
196. $medias[] = $mediaModel->toArray();
197. $mediasNames[] = $name;
198. $mediasHashes[] = $hash;
199. }
200. catch (\Exception $ex) {
201. $this->error("\r/ "
202. . $counter
203. ." <bg=red;fg=white;href=file:///$file>$name</> "
204. .$ex->getMessage()
205. );
206. continue;
207. }
208. }

При запуске команды php artisan app:index можно увидеть как отображается контент (см. рис. 38): известные (серые). новые (зелёные), дубликаты (оранжевые), не совместимые (синие) и с ошибками (красные). Каждый медиафайл и альбом имеет ссылку для проводника (file:///) и если успешно удалось проиндексировать — ссылку на сайт.

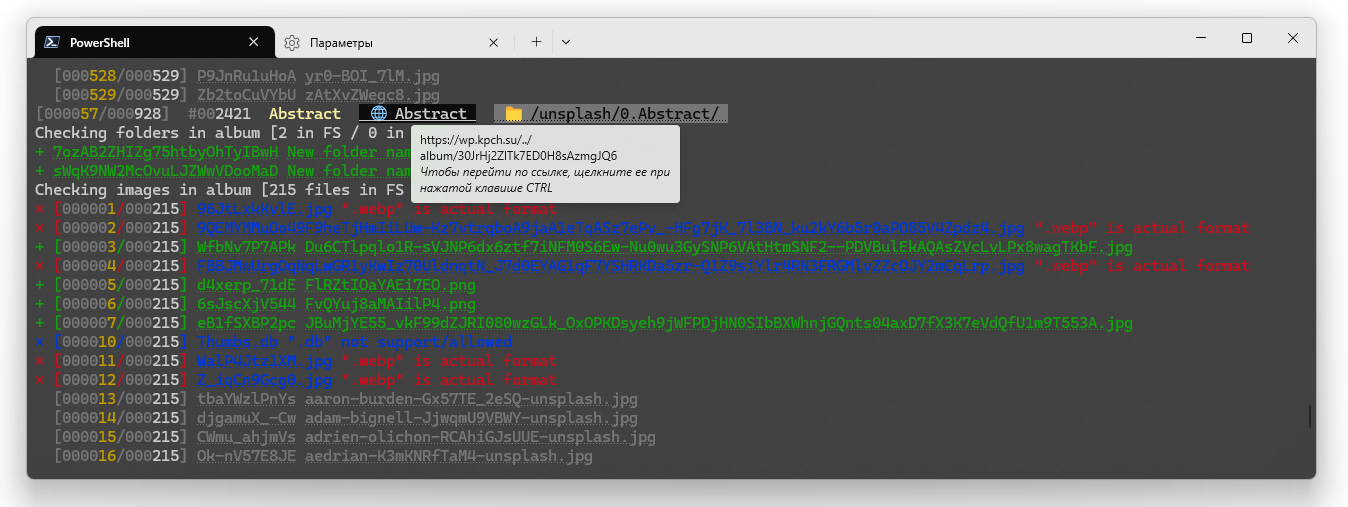


Рисунок 38 – Результат разработки команды индексации

## Представления

Представления (Views) в Laravel отвечают за отображение данных пользователю и формирование HTML-структуры веб-страниц. Они позволяют отделить логику приложения от визуальной части, что улучшает читаемость и поддерживаемость кода.

В Laravel представления хранятся в директории resources/views и могут быть написаны на HTML, PHP, и на шаблонизатора Blade. Blade предоставляет удобный синтаксис для работы с данными, условиями, циклами и шаблонами, компилируясь в оптимизированный PHP-код без потери производительности.

В текущем проекте Blade используется для внутренней генерации OG (Open Graph) изображения содержимого альбома и для вывода OG тегов. За главное представление отвечает Vue проект.

Шаблон OG тегов (cм. лист. 16) получат данные из ViewController. Он может отобразить информацию и изображение содержимого альбома (генерируемый через Browsershot, через внутренний браузер), может отобразить картинку и видео. Результат можно увидеть на рисунке 39.

Листинг 16 – Шаблон Open Graph тегов из resources\views\index.blade.php

1. <title>{{ config('app.name') }}</title>
2. <meta property="og:site\_name" content="{{ config('app.name') }}" />
3. @if(isset($album))
4. @if(!isset($image))
5. @php
6. *// Пункты содержимого альбома*
7. $parts = [];
8. if ($album->albums\_count) $parts[] = "{$album->albums\_count} sub-album". ($album->albums\_count > 1 ? 's' : '');
9. if ($album->audios\_count) $parts[] = "{$album->audios\_count} audio" . ($album->audios\_count > 1 ? 's' : '');
10. if ($album->videos\_count) $parts[] = "{$album->videos\_count} video" . ($album->videos\_count > 1 ? 's' : '');
11. if ($album->images\_count) $parts[] = "{$album->images\_count} image" . ($album->images\_count > 1 ? 's' : '');
12. *// Длительность всех видео и размер всех медиа*
13. $duration = $album->duration ? durationToHuman($album->duration / 1000).' in length' : null;
14. $size = $album->size ? bytesToHuman ($album->size) .' in size' : null;
15. $hasCounters = !!count($parts);
16. $hasContent = $hasCounters || $duration || $size;
17. if (!$hasContent)
18. $description = 'Empty album';
19. else {
20. $description = 'Explore an album';
21. if (count($parts)) {
22. $last = array\_pop($parts);
23. $description .= ' with '.(count($parts)
24. ? implode(', ', $parts) .' and '. $last
25. : $last
26. );
27. }
28. if ($duration || $size) {
29. $tail = ($duration && $size)
30. ? "$duration and $size"
31. : ($duration ?? $size);
32. $description .= ($hasCounters ? ', totaling ' : ' with totaling '). $tail;
33. }
34. }
35. @endphp
36. <meta property="og:title" content="{{ $album->name }}" />
37. <meta property="og:description" content="{{ $description }}" />
38. <meta property="og:image:type" content="image/png" />
39. <meta property="og:image:width" content="1200" />
40. <meta property="og:image:height" content="1200" />
41. <meta property="og:image" content="{{ route('get.album.og', $album->hash) }}" />
42. <meta name="twitter:card" content="summary\_large\_image">
43. <meta name="twitter:image:type" content="image/png" />
44. <meta name="twitter:image:width" content="1200" />
45. <meta name="twitter:image:height" content="1200" />
46. <meta name="twitter:image" content="{{ route('get.album.og', $album->hash) }}" />
47. @else
48. <meta property="og:title" content="{{ $image->name }}" />
49. <meta property="og:image:width" content="{{ $image->widthThumb }}" />
50. <meta property="og:image:height" content="{{ $image->heightThumb }}" />
51. <meta property="og:image" content="{{ $image->urlThumbRoute }}" />
52. <meta name="twitter:image:width" content="{{ $image->widthThumb }}" />
53. <meta name="twitter:image:height" content="{{ $image->heightThumb }}" />
54. <meta name="twitter:image" content="{{ $image->urlThumbRoute }}" />
55. @if($image->type === 'video' || $image->type === 'imageAnimated')
56. <meta property="og:description" content="Explore {{
57. (($album?->videos\_count ?? 0) > 1
58. ? ($album->videos\_count - 1)." more videos in "
59. : (($album?->albums\_count ?? 0) > 1
60. ? "$album->albums\_count sub-albums in "
61. : ''
62. ))
63. }}{{ $album->name }}" />
64. <meta property="og:type" content="video.other" />
65. <meta property="og:video:width" content="{{ $image->width }}" />
66. <meta property="og:video:height" content="{{ $image->height }}" />
67. <meta property="og:video:duration" content="{{ (int)($image->duration\_ms / 1000) }}" />
68. <meta property="og:video" content="{{ $image->urlOrigRoute }}" />
69. <meta name="twitter:card" content="player" />
70. <meta name="twitter:player:width" content="{{ $image->width }}" />
71. <meta name="twitter:player:height" content="{{ $image->height }}" />
72. <meta name="twitter:player" content="{{ $image->urlOrigRoute }}" />
73. @else
74. <meta property="og:description" content="Explore {{
75. (($album?->images\_count ?? 0) > 1
76. ? ($album->images\_count - 1)." more images in "
77. : (($album?->albums\_count ?? 0) > 1
78. ? "$album->albums\_count sub-albums in "
79. : ''
80. ))
81. }}{{ $album->name }}" />
82. <meta name="twitter:card" content="summary\_large\_image">
83. @endif
84. @endif
85. @else
86. @if(Request::*is*('/'))
87. <meta property="og:title" content="Homepage" />
88. @else
89. <meta property="og:title" content="Wepics" />
90. @endif <meta property="og:image" content="/favicon/maskable\_icon\_x512.png" />
91. @endif

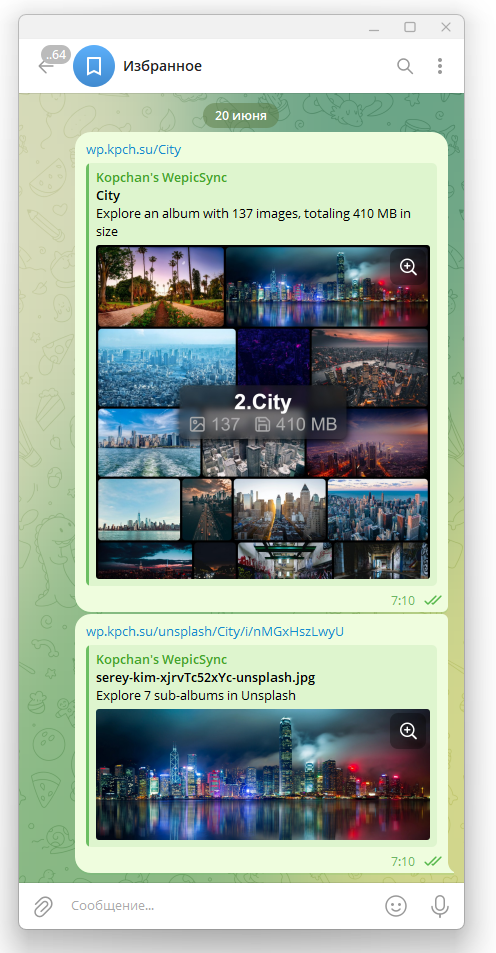


Рисунок 39 – Результат разработки Open Graph тегов.   
Демонстрация в Telegram мессенджере

# Глава 4. Проектирование и разработка веб-сайта

Веб-сайт проектаWepicSync представляет собой одностраничное приложение (SPA), разработанное на Vue 3 с использованием Composition API. Для управления состоянием приложения применена библиотека Pinia, эффективно работая с глобальными данными.

Веб-сайт взаимодействует с серверной частью через стандартный Fetch API, выполняя аутентификацию по токенам и обмен данными в формате JSON.

Разработка велась в Visual Studio Codeс поддержкой Vue-экосистемы в виде расширения. Для контроля версий использовался Git. Сборка проекта выполнялась с помощью Vite – современного инструмента для быстрой разработки, обеспечивающего мгновенный горячий перезапуск во время изменения кода (HMR) и оптимизированную сборку для продакшена. Для запуска Vite требуется Node.js.

## Зависимости проекта

1. *Vue 3* – Прогрессивный JavaScript-фреймворк для создания пользовательских интерфейсов
2. *Pinia 2* – Официальное хранилище (state management) для Vue 3.
3. *Lucide 0.511* – Библиотека svg иконок с открытым исходным кодом.
4. *Vue-router 4.3* – Официальная библиотека маршрутизации для Vue 3.

## Структура проекта

Структура проекта включает в себя следующие основные каталоги и основные файлы:

1. public/ – Общие ресурсы, иконки.
   1. manifest.json – Манифест для установки сайта как приложение (PWA).
2. src/ – Исходный код.
   1. api/ – Помощники с API.
      1. urls.js – все маршруты API.
   2. assets/ – Стили, шрифт.
   3. components/ – Компоненты.
   4. config/ – Конфигурационные значение из .env.
   5. helpers/ – Помощники общие.
   6. router/ – Маршруты для представлений.
   7. stores/ – Pinia хранилища.
   8. views/ – Представления.
   9. App.vue – Главный компонент.
   10. main.js – Точка входа в приложение.

## Базовые элементы

### Точка входа

Точка входа приложения (см. лист. 17) находится в корневой папке проекта и называется main.js. В этом файле импортируется Vue и все основные компоненты, директивы, плагины и т.д.

Листинг 17 – Код точки входа из /src/main.js

1. import './assets/main.scss'
2. import { createApp } from 'vue'
3. import { createPinia } from 'pinia'
4. import PrimeVue from 'primevue/config'
5. import App from './App.vue'
6. import router from './router'
7. const app = createApp(App)
8. app.use(createPinia())
9. app.use(router)
10. app.use(PrimeVue)
11. app.mount('#app')

**Главный компонент**

Главный компонент Vue.js (см. лист. 18) приложения называется App.vue. Он обычно находится в корневой папке вашего проекта и является основным компонентом, который включает в себя другие компоненты и представляет собой пользовательский интерфейс приложения. Здесь описана логика переключения темы.

Листинг 18 – Код главного компонента из /src/App.vue

1. <script setup>
2. // Установленная пользователем тема
3. const { theme } = storeToRefs(useSettingsStore())
4. // Функция получения HTML Node'ов по обычной строке
5. const getNodesFromString = (html) => {
6. const template = document.createElement('div')
7. template.innerHTML = html
8. return template.childNodes
9. }
10. // Функция  установки темы
11. const setupTheme = () => {
12. // Удаляем старую тему
13. document.documentElement.classList.remove('light', 'dark')
14. const metaTags = document.head.querySelectorAll('[name="theme-color"]')
15. for(const metaTag of metaTags) metaTag.remove()
16. // Устанавливаем новую тему
17. switch (theme.value) {
18. case 'light':
19. case 'dark':
20. // Если тёмная или светлая
21. document.documentElement.classList.add(theme.value)
22. getNodesFromString(
23. `<meta name="theme-color" content="#${theme.value == 'light' ? 'fff' : '000'}">`
24. ).forEach(node => {
25. document.head.appendChild(node)
26. })
27. return
28. case 'auto':
29. // Если автоматическая
30. getNodesFromString(
31. '<meta name="theme-color" content="#fff" media="(prefers-color-scheme: light)">' +
32. '<meta name="theme-color" content="#000" media="(prefers-color-scheme:  dark)">'
33. ).forEach(node => {
34. document.head.appendChild(node)
35. })
36. return
37. }
38. }
39. onMounted(() => {
40. // Убираем авто-тему, если пользователь установил конкретную
41. if (theme.value == 'light' ||
42. theme.value == 'dark')
43. setupTheme()
44. })
45. // Устанавливаем новую тему, если пользователь сменил её
46. watch(() => theme.value, setupTheme)
47. </script>
48. <template>
49. <TheHeader/>
50. <RouterView/>
51. </template>

## Компоненты

В Vue.js компоненты — это самое основное строительное блоки при создании приложений. Они представляют собой независимые, повторно используемые куски кода, которые могут быть легко составлены вместе для создания сложных пользовательских интерфейсов.

Компоненты в Vue.js имеют следующие основные элементы:

1. Шаблон (Template): это HTML-шаблон, который определяет структуру компонента. В шаблоне можно использовать директивы Vue, такие как v-if, v-for, v-bind и другие.
2. Скрипт (Script): это JavaScript-код, который определяет логику компонента. В скрипте объявляются свойства, методы и другие функции, которые могут быть использованы внутри компонента.
3. Стили (Styles): это CSS-код, который определяет внешний вид компонента. В стилях можно использовать селекторы, которые будут применяться только к этому компоненту.

### Основные компоненты

1. Заголовок сайта (см. лист. 19)

Компонент, обеспечивающий функционал навигации и управления альбомами. Компонент содержит навигационную цепочку, создания объектов (альбома/картинок), управлять доступом к альбому и может переключать сортировку. Компонент также поддерживает функционал всплывающих окон для отображения списка дочерних альбомов, создания объектов и поделиться альбомом. Внутри компонента используется fetchWrapper для запроса данных о текущем альбоме. Результат разработки изображён на рисунке 40.

Листинг 19 – Часть кода заголовка сайта из /src/components/TheHeader.vue

1. <script setup>
2. // Данные об текущем открытом альбоме
3. const {
4. targetAlbum, sort, isReverse, albumData
5. } = storeToRefs(useAlbumParamsStore())
6. // Данные об текущем пользователе
7. const { user } = storeToRefs(useAuthStore())
8. // Всплывающие окна
9. const authCard = ref()
10. const userCard = ref()
11. const customizCard = ref()
12. const subAlbumsCard = ref()
13. const albumShareCard = ref()
14. const objCreateCard = ref()
15. // Логика переключение всплывающих окон со списком дочерних альбомов
16. const albumChildren = ref(null)
17. const toggleSubAlbumsCard = (e, hash) => {
18. albumChildren.value = hash
19. subAlbumsCard.value.toggle(e)
20. }
21. // Заполнение данных об альбоме
22. onMounted(() => {
23. debouncedWatch(
24. () => targetAlbum.value,
25. () => {
26. fetchWrapper.get(
27. '/albums/' + targetAlbum.value
28. ).then(data => {
29. console.log({albumData: data})
30. albumData.value = data
31. })
32. },
33. { debounce: 250, immediate: true }
34. )
35. })
36. </script>

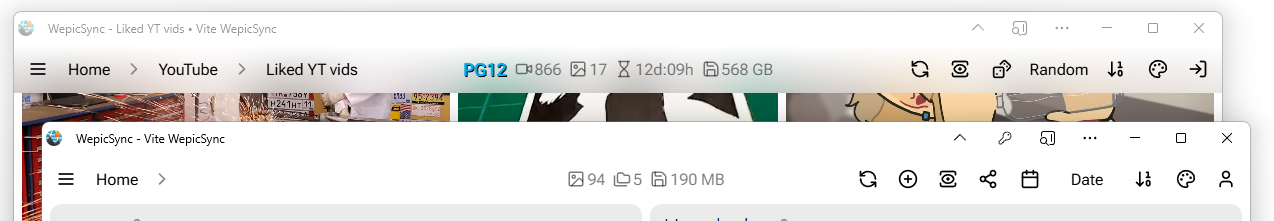


Рисунок 40 – Результат разработки TheHeader

1. Радио-кнопки (см. лист. 20)

Компонент, обеспечивающий функционал радио-кнопок. Компонент принимает три параметра: "name" (имя группы радио-кнопок), "options" (массив опций) и "disabled" (флаг, который отключает все радио-кнопки). Компонент также поддерживает v-model для двусторонней привязки значения. Внутри компонента используется цикл v-for для создания радио-кнопок для каждой опции. Результат разработки изображён на рисунке 41.

Листинг 20 – Часть код радио-кнопок из /src/components/ui/BtnRadios.vue

1. <script setup>
2. // Параметры компонента
3. const props = defineProps({
4. name: String,
5. options: Object,
6. disabled: {
7. required: false,
8. default: false
9. }
10. })
11. // v-model компонента по умолчанию
12. const value = defineModel()
13. // Параметры компонента в переменные
14. const { name, options } = toRefs(props)
15. // Установка по умолчанию
16. value.value ??= Object.keys(options.value)[0]
17. </script>
18. <template>
19. <div class="line">
20. <template v-for="(option, index) in options" :key="index">
21. <input
22. type="radio"
23. v-model="value"
24. :name  ="name"
25. :value ="index"
26. :id    ="name +'\_'+ index"
27. :disabled="disabled"
28. >
29. <label :for="name +'\_'+ index" class="btn">{{ option }}</label>
30. </template >
31. </div>
32. </template>



Рисунок 41 – Результат разработки BtnRadios

### Компоненты-окна

1. Панель авторизации (см. лист. 21)

Компонент, обеспечивающий форму авторизации и регистрации пользователей. Компонент содержит два режима: "login" и "signup", которые можно выбрать с помощью радио-кнопок. Форма входа и регистрации содержат поля для ввода логина, пароля и никнейма (для режима регистрации). Компонент также следит за статусом загрузки, ошибками и очищает ошибки при изменении режима или данных формы. При нажатии на кнопку "Enter" или "Create" вызывается функция "authorize", которая отправляет данные формы на сервер для авторизации или регистрации. В случае ошибки, сообщение об ошибке отображается на странице. Результат разработки изображён на рисунке 42.

Листинг 21 – Часть кода панели авторизации /src/components/AuthPanel.vue

1. <script setup>
2. const isFetching = ref(false) // Статус "загружается"
3. const mode = ref('login')    // Режим
4. const form = ref({})        // Отпровляемые данные
5. const error = ref(null)    // Ошибка загрузки
6. // Стор с функциями входа/регистрации
7. const auth = useAuthStore()
8. // Функция входа/регистрации
9. const authorize = () => {
10. isFetching.value = true
11. error.value = null
12. // Выбор входа/регистрации
13. auth[mode.value](form.value)
14. .catch(err => error.value = err)
15. isFetching.value = false
16. }
17. // Следим за режимом и данными формы, при изменение очищаем ошибку
18. watch(
19. () => [mode.value, form.value],
20. () => error.value = null,
21. { deep: true }
22. )
23. </script>
24. <template>
25. <div class="outer">
26. <!-- Селектор режима -->
27. <BtnRadios name="mode" v-model="mode" :options="{login: 'Log In', signup: 'Sign Up'}"/>
28. <!-- Форма входа -->
29. <form @submit.prevent="authorize" v-if="mode == 'login'">
30. <input class="text-box" required v-model="form.login"    placeholder="Login">
31. <input class="text-box" required v-model="form.password" placeholder="Password" type="password">
32. <button class="btn" :disable="isFetching">Enter</button>
33. <!-- Окно ошибки -->
34. <div class="error" v-if="error">
35. <p v-if="error.status == 401">Login and/or password not correct</p>
36. <p v-else>Authorization failed</p>
37. </div>
38. </form>
39. <!-- Форма регистрации -->
40. <form @submit.prevent="authorize" v-else>
41. <input class="text-box" required v-model="form.nickname" placeholder="Nickname">
42. <input class="text-box" required v-model="form.login"    placeholder="Login">
43. <input class="text-box" required v-model="form.password" placeholder="Password" type="password">
44. <button class="btn" :disable="isFetching">Create</button>
45. <!-- Окно ошибки -->
46. <div class="error" v-if="error">
47. <template v-if="error.status == 422">
48. <template v-for="errorArr in error.message" :key="errorArr">
49. <template v-for="error in errorArr" :key="error">
50. <p>{{ error }}</p>
51. </template>
52. </template>
53. </template>
54. <p v-else>Registration failed</p>
55. </div>
56. </form>
57. </div>
58. </template>

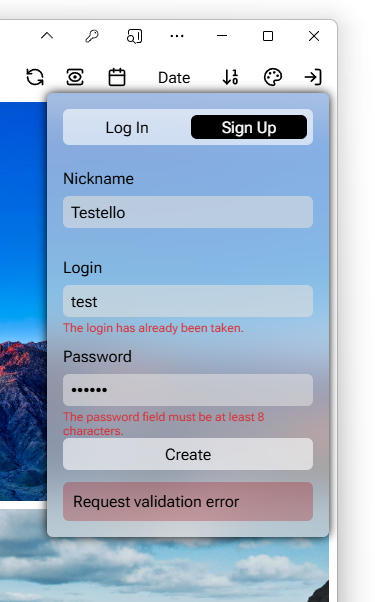


Рисунок 42 – Результат разработки AuthPanel

1. Панель пользователя (см. лист. 22)

Компонент, обеспечивающий функционал выхода из системы. Компонент принимает данные о текущем пользователе из хранилища и отображает его никнейм и статус администратора. Внутри компонента используется кнопка для выхода из системы. Результат разработки изображён на рисунке 43.

Листинг 22 – Часть кода панели пользователя /src/components/panels/UserPanel.vue

1. <script setup>
2. const auth = useAuthStore()
3. const { user } = storeToRefs(useAuthStore())
4. </script>
5. <template>
6. <div class="outer">
7. <div>
8. <p class="gray">
9. You are logged in as
10. <span v-if="user.isAdmin" class="badge">admin</span>
11. </p>
12. <p class="name">
13. {{ user.nickname }}
14. </p>
15. </div>
16. <button class="btn" @click="auth.logout">Log Out</button>
17. </div>
18. </template>

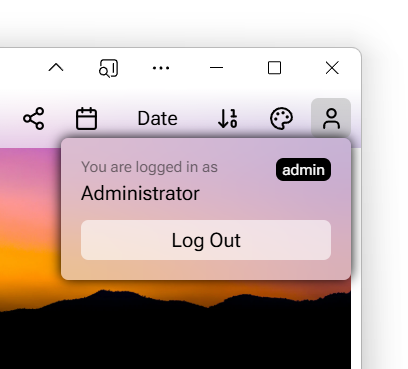


Рисунок 43. Результат разработки UserPanel

1. Панель настроек вида (см. лист. 23)

Компонент, обеспечивающий функционал настройки кастомизации. Компонент принимает данные о текущих настройках кастомизации из хранилища и отображает их в виде радио-кнопок, полей ввода и слайдеров. Внутри компонента используется кнопка для сброса настроек кастомизации. Результат разработки изображён на рисунке 44.

Листинг 23 – Часть кода панели настроек вида /src/components/panels/SettingsPanel.vue

1. <script setup>
2. // Стор с настроками кастомизации
3. const settings = useSettingsStore()
4. const {
5. size, isStrictSize, isRealSize, lines, gap, radius, orientation, theme
6. } = storeToRefs(settings)
7. </script>
8. <template>
9. <div class="outer">
10. <h4>Customization</h4>
11. <!-- Выбор темы -->
12. <div class="label-group">
13. <label>Theme</label>
14. </div>
15. <BtnRadios name="theme" :options="{auto: 'Auto', light: 'Light', dark: 'Dark'}" v-model="theme"/>
16. <!-- Выбор кол-во столбцов/строк -->
17. <div class="label-group">
18. <label>{{ orientation == 'w' ? 'Columns' : 'Rows' }}</label>
19. <input type="number" min="0" max="12" v-model.number="lines" class="text-box">
20. </div>
21. <input type="range" min="0" max="12" v-model.number="lines">
22. <!-- Выбор размера карточек -->
23. <div class="label-group">
24. <label>{{ orientation == 'w' ? 'Width' : 'Height' }}</label>
25. <input type="number" min="144" max="1080" v-model.number="size" class="text-box" :disabled="lines && !isStrictSize">
26. </div>
27. <input type="range" step="2" min="144" max="1080" v-model.number="size" :disabled="lines && !isStrictSize">
28. <BtnRadios name="isRealSize" :options="['Relative', 'Absolute']" v-model="isRealSize" :disabled="lines && !isStrictSize"/>
29. <BtnRadios name="isStrictSize" :options="['Fill', 'Strict']" v-model="isStrictSize"/>
30. <!-- Выбор размера отступов -->
31. <div class="label-group">
32. <label>Gap</label>
33. <input type="number" min="0" max="32" v-model.number="gap" class="text-box">
34. </div>
35. <input type="range" min="0" max="32" v-model.number="gap">
36. <!-- Выбор размера закругления -->
37. <div class="label-group">
38. <label>Radius</label>
39. <input type="number" min="0" max="32" v-model.number="radius" class="text-box">
40. </div>
41. <input type="range" min="0" max="32" v-model.number="radius">
42. <button class="btn" @click="settings.reset">Reset</button>
43. </div>
44. </template>

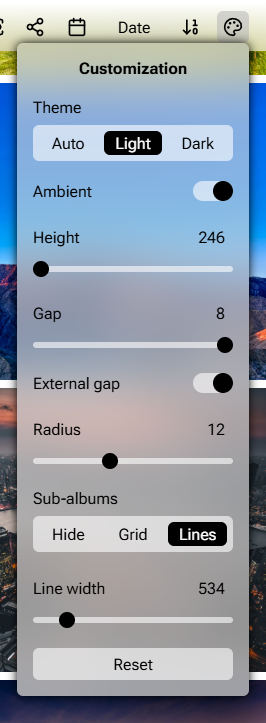


Рисунок 44. Результат разработки SettingsPanel

1. Панель списка дочерних альбомов (см. лист. 24)

Компонент, обеспечивающий функционал отображения списка дочерних альбомов. Компонент принимает хеш альбома и отображает список дочерних альбомов. Результат разработки изображён на рисунке 45.

Листинг 24 – Часть кода панели дочерних альбомов /src/components/panels/SubAlbumsPanel

1. <script setup>
2. import { toRefs, ref } from 'vue'
3. import { storeToRefs } from 'pinia'
4. import { fetchWrapper } from '@/helpers'
5. import { useAlbumParamsStore } from '@/stores'
6. // Заданные данные компоненту
7. const props = defineProps({
8. hash: String
9. })
10. // Хеш тыкнутого альбома
11. const { hash } = toRefs(props)
12. // Данные по текущему открытому альбому
13. const { targetAlbum, albumData } = storeToRefs(useAlbumParamsStore())
14. const subAlbumData = ref(null) // Данные по тыкнутому альбому
15. const isErrored = ref(false)  // Статус "произошла ошибка"
16. const isLoading = ref(true)  // Статус "загружаюсь"
17. if (hash.value === targetAlbum.value) {
18. // Если тыкнутый альбом = текущий альбом, то данные уже есть
19. isLoading.value = false
20. subAlbumData.value = albumData.value
21. }
22. else fetchWrapper.get('/albums/' + hash.value)
23. // Иначе загрузить по хешу
24. .then(data => {
25. isLoading.value = false
26. subAlbumData.value = data
27. }).catch(() => {
28. isLoading.value = false
29. isErrored.value = true
30. })
31. </script>
32. <template>
33. <div class="outer">
34. <template v-if="subAlbumData?.children">
35. <button
36. v-for="(childParams, child) in subAlbumData?.children"
37. :key="child"
38. class="btn"
39. @click="targetAlbum = childParams.hash">
40. {{ child }}
41. </button>
42. </template>
43. <template v-else>
44. <p v-if="isLoading">Loading...</p>
45. <p v-else-if="isErrored">Error download</p>
46. <p v-else-if="!subAlbumData?.children">No childs</p>
47. </template>
48. </div>
49. </template>

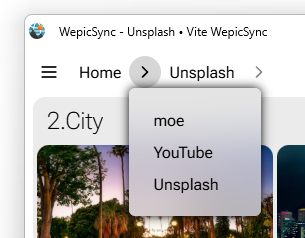


Рисунок 45. Результат разработки SubAlbumsPanel

1. Панель сортировки (см. лист. 25)

Компонент, обеспечивающий функционал сортировки медиа и альбомов. Также при случайной сортировке выводится поле seed с случайным числом, чтобы между страницами списка медиа не было повторов. При включении сортировка по содержимому альбомов — отображает на каждом альбоме дополнительное результирующие поле (например дата первой медиа в альбоме). Результат разработки изображён на рисунке 46.

Листинг 25 – Разметка панели сортировки /src/components/panels/SortPanel.vue

1. <template>
2. <div class="outer">
3. <h4>Sort & Filter</h4>
4. <!-- Медиа -->
5. <div class="label-group">
6. <label>Media sort</label>
7. <button
8. @click="isReverse = !isReverse"
9. :disabled="sort === 'random'"
10. :class="{
11. 'btn': true,
12. 'reverse': true,
13. 'btn--inverse': isReverse,
14. }">
15. Reverse
16. <template>
17. <ArrowUpAZIcon size="20" v-if="isReverse"/>
18. <ArrowDownAZIcon size="20" v-else/>
19. </template>
20. </button>
21. </div>
22. <SelectorRadios
23. :options="imagesSortTypes"
24. v-model="sort"
25. name="mediasSort"
26. noCenter
27. grid
28. />
29. <div class="label-group">
30. <label>Show nested media</label>
31. <label class="switch"><input type="checkbox" v-model.number="nested"/><div></div></label>
32. </div>
33. <!-- Альбомы -->
34. <div class="label-group">
35. <label>Albums sort</label>
36. <button
37. @click="isReverseAlbums = !isReverseAlbums"
38. :disabled="sortAlbums === 'random'"
39. :class="{
40. 'btn': true,
41. 'reverse': true,
42. 'btn--inverse': isReverseAlbums,
43. }">
44. Reverse
45. <template v-if="sortAlbums == 'name'">
46. <ArrowUpAZIcon size="20" v-if="isReverseAlbums"/>
47. <ArrowDownAZIcon size="20" v-else/>
48. </template>
49. <template v-else>
50. <ArrowUp10Icon size="20" v-if="isReverseAlbums"/>
51. <ArrowDown10Icon size="20" v-else/>
52. </template>
53. </button>
54. </div>
55. <SelectorRadios2
56. :options="albumsSortTypes"
57. v-model="sortAlbums"
58. name="albumSort"
59. noCenter
60. grid
61. />
62. <div class="label-group">
63. <label>Ignore custom order</label>
64. <label class="switch"><input type="checkbox" v-model.number="disord"/><div></div></label>
65. </div>
66. <div class="label-group" v-if="sortAlbums === 'random' || sort === 'random'">
67. <label>Random seed</label>
68. <input class="text-box small" type="number" v-model="randomSeed" maxlength="10">
69. </div>
70. </div>
71. </template>

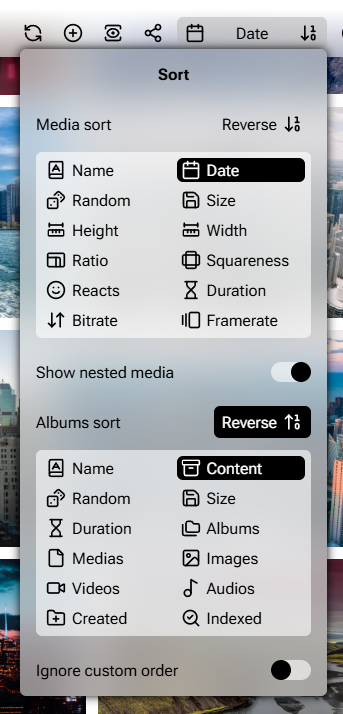


Рисунок 46. Результат разработки SortPanel

1. Панель управления доступом (см. лист. 26)

Компонент, обеспечивающий функционал управления доступом к альбому. Компонент принимает хеш выбранного альбома и отображает список допущенных и не допущенных пользователей. Внутри компонента запрашивается список прав на выбранном альбоме и отправки данных о создании правила доступа. Компонент также поддерживает функционал удаления правила доступа. Результат разработки изображён на рисунке 47.

Листинг 26 –Часть кода панели доступов /src/components/panels/AlbumSharePanel.vue

1. <script setup>
2. // Параметры компонента
3. const props = defineProps({
4. hash: String
5. })
6. // Запрос списка прав на выбранном альбоме
7. fetchWrapper.get(
8. '/albums/' + hash.value + '/access'
9. ).then(data => {
10. isLoading.value = false
11. isForGuest.value = data.isGuestAllowed ?? null
12. allowedRights   .value = data.listAllowed    ?? []
13. disallowedRights.value = data.listDisallowed ?? []
14. if (isForGuest.value === true )
15. allowedRights.value.push(guestObj)
16. if (isForGuest.value === false)
17. disallowedRights.value.push(guestObj)
18. }).catch(() => {
19. isLoading.value = false
20. isErrored.value = true
21. })
22. // Удаление правила
23. const removeAccess = (user\_id = null) => {
24. console.log(user\_id)
25. fetchWrapper.delete(
26. '/albums/' + hash.value +
27. '/access' +
28. (user\_id ? ('?user\_id=' + user\_id) : '')
29. )
30. if (user\_id) {
31. allowedRights.value = allowedRights.value
32. .filter(el => !(el.user\_id === user\_id))
33. disallowedRights.value = disallowedRights.value
34. .filter(el => !(el.user\_id === user\_id))
35. }
36. }
37. const mode = ref('show') // Режим отображения
38. const users = ref([]) // Список пользователей системы
39. // Переключение на страницу выбора пользователя для создания правила
40. const toUserSelect = () => {
41. mode.value = 'addRule'
42. isLoading.value = true
43. // Получение всех пользователей
44. fetchWrapper.get('/users').then(gettedUsers => {
45. // Все правила на выбранном альбоме
46. const rules = [
47. ...(   allowedRights.value),
48. ...(disallowedRights.value)
49. ]
50. // Добавление гостя в список пользователей
51. gettedUsers.push({id: null, nickname: 'guest'})
53. // Запись допущенных для выборки создания правила
54. // т.е. пользователи, которые не указаны в правилах выбранного альбома
55. users.value = gettedUsers.filter(user => {
56. for (const rule of rules) {
57. if (rule.user\_id === user.id)
58. return false
59. }
60. return true
61. })
62. isLoading.value = false
63. }).catch(() => {
64. isLoading.value = false
65. isErrored.value = true
66. })
67. }
68. // Создания правила
69. const addAccess = (user = null, isAllow = true) => {
70. fetchWrapper.post(
71. '/albums/' + hash.value +
72. '/access' +
73. '?allow=' + (isAllow ? 1 : 0) +
74. (user.id ? ('&user\_id=' + user.id) : '')
75. ).then(() => {
76. if (isAllow) allowedRights.value.push({user\_id: user.id, nickname: user.nickname})
77. else      disallowedRights.value.push({user\_id: user.id, nickname: user.nickname})
78. mode.value = 'show'
79. }).catch(() => {
80. isLoading.value = false
81. isErrored.value = true
82. })
83. }
84. </script>

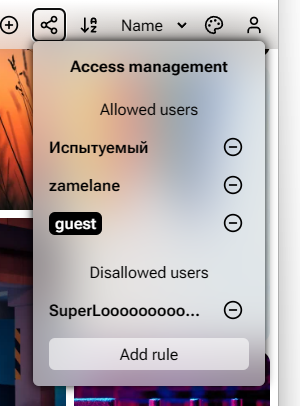


Рисунок 47. Результат разработки AlbumSharePanel

## Глобальные хранилища

Для глобальных хранилищ состояний была выбрана Pinia. Pinia — это современный и простой стейт-менеджер для Vue.js, который позволяет вам управлять состоянием приложения с помощью простых и интуитивно понятного API. Она была создана как замена Vuex, но с более простым API и меньшим количеством кода, что делает его идеальным выбором для проекта.

1. Параметры текущего альбома (см. лист. 27)

Хранит текущий открытый альбом, сортировку и другие параметры фильтрации и доп-сортировок из адресной строки. Для того чтобы брать из адресной строки различные параметры реализована функция createProp.

Листинг 27 – Параметры текущего альбома /src/stores/albumParams.js

1. import { computed, ref } from 'vue'
2. import { defineStore } from 'pinia'
3. import { router } from '@/router'
4. export const useAlbumParamsStore = defineStore('albumParams', () => {
5. // Функция создания get-set переменной для параметра в адресной строке
6. const createProp = (
7. name,
8. defaultValue = null,
9. routeName = false,
10. isArray = false,
11. ) => computed({
12. get: () => {
13. // Получение параметра из адресной строки
14. const prop = router.currentRoute.value[routeName ? 'params' : 'query'][name]
15. // Если переменная была с предустноновкой "это массив",
16. // то элементы массива считываются через запятую
17. if (isArray)
18. return prop?.split(',')?.map(elem => decodeURIComponent(elem)) ?? []
19. // Если null, то это true
20. else if (prop === null)
21. return true
23. // Если undefined, то возвращается стандартное значение,
24. // иначе полученное из адресной строки
25. return prop ?? defaultValue
26. },
27. set: (value) => {
28. // Полученние query-параметров из адресной строки
29. const query = router.currentRoute.value.query
30. if (routeName) {
31. if (value === undefined || value === defaultValue) {
32. // Если получено значение "ничего" или равняющееся по умолчанию,
33. // то перейти на главную страницу
34. router.push({
35. name: 'home',
36. query
37. })
38. return
39. }
41. // Переход на новое заданное значение
42. const params = router.currentRoute.value.params
43. router.push({
44. name: routeName,
45. params: { ...params, [name]: value ?? defaultValue },
46. query
47. })
48. return
49. }
51. // Обработка нового параметра
52. let newQuery = {...query}
53. if (value === undefined || value === false || value.length === 0)
54. delete newQuery[name]
56. else if (value === true)
57. newQuery[name] = null
58. else if (value instanceof Array)
59. newQuery[name] = value.map(elem => encodeURIComponent(elem))?.join(',')
61. else newQuery[name] = value
62. // Замена параметров в адресной строке
63. router.replace({ query: newQuery })
64. }
65. })
66. const targetAlbum = createProp('albumHash', 'root', 'album')
67. const limit       = createProp('limit', 30 )
68. const sort        = createProp('sort', 'name')
69. const isReverse   = createProp('reverse', false)
70. const tags        = createProp('tags', [], undefined, true)
71. const albumData = ref({})
73. return { targetAlbum, limit, sort, isReverse, tags, albumData }
74. })
75. Пользователь и аутентификация (см. лист. 28)

Хранит данные пользователя, токен и имеет функции входа, регистрации и выхода.

Листинг 28 – Пользователь и аутентификация /src/stores/auth.js

1. import { defineStore } from 'pinia'
2. import { fetchWrapper, storageWrapper } from '@/helpers'
3. export const useAuthStore = defineStore('auth', () => {
4. // Инфа об юзере в localStorage
5. const user = storageWrapper('user')
6. // Регистрация
7. async function signup({ nickname, login, password }) {
8. const response = await fetchWrapper
9. .post('/users/reg', { nickname, login, password })
11. user.token = response.token
12. user.isAdmin = false
13. user.nickname = nickname
14. }
15. // Вход
16. async function login({ login, password }) {
17. const res = await fetchWrapper
18. .post('/users/login', { login, password })
20. user.token = res.token
21. user.isAdmin = res.isAdmin ?? false
22. user.nickname = res.nickname
23. }
25. // Выход
26. async function logout() {
27. await fetch('/users/logout', {
28. headers: { Authorization: `Bearer ${user.token}` }
29. }).catch()
31. for (const key in user)
32. delete user[key]
33. }
34. return { user, signup, login, logout }
35. })
36. Настройки внешнего вида (см. лист. 29)

Хранит различные параметры для внешнего вида такие как размер медиа, отступы, закругление, тему и другие.

Листинг 29 – Настройки внешнего вида /src/stores/settings.js

1. import { computed } from 'vue'
2. import { defineStore } from 'pinia'
3. import { storageWrapper } from '@/helpers'
4. export const useSettingsStore = defineStore('settings', () => {
5. // Инфа об настройках в localStorage
6. const settings = storageWrapper('settings')
7. // Сброс настроек
8. const reset = () => {
9. for (const key in settings) {
10. delete settings[key]
11. }
12. }
13. // Создание параметра в localStorage со значением по умолчанию
14. const createProp = (name, defaultValue) => {
15. return computed({
16. get: ()      => settings[name] ?? defaultValue,
17. set: (value) => settings[name] = value
18. })
19. }
20. const size         = createProp('size', 360)
21. const lines        = createProp('lines', 0)
22. const gap          = createProp('gap', 8)
23. const radius       = createProp('radius', 12)
24. const theme        = createProp('theme', 'auto')
26. return {
27. settings, reset, size, isStrictSize, isRealSize,
28. lines, gap, radius, orientation, theme
29. }
30. })

## Функции-помощники

1. Обёртка над fetch (см. лист. 30)

Этот код представляет собой функциональный обертку для работы с HTTP-запросами в JavaScript, используя fetch API. Он предоставляет удобный интерфейс для выполнения GET, POST, PATCH и DELETE-запросов к API, которое находится по пути API\_PATH из файла конфигурации. Функция request генерирует методы для каждого типа запроса, устанавливая необходимые заголовки и, если необходимо, сериализуя тело запроса в JSON. Функция authHeader добавляет заголовок авторизации с токеном, если он доступен в хранилище авторизации. Функция handleResponse обрабатывает ответы от API, пытается распарсить их в JSON, и в случае ошибок вызывает логаут, если это 401-ошибка, или задержку перед повторным запросом, если это 429-ошибка. В случае успешного запроса, функция возвращает данные в JSON формате, полученные от API.

Листинг 30 – Обёртка над fetch /src/helpers/fetchWrapper.js

1. export const fetchWrapper = {
2. get:    request('GET'),
3. post:   request('POST'),
4. patch:  request('PATCH'),
5. delete: request('DELETE')
6. }
7. function request(method) {
8. return (path, body = null) => {
9. const requestOptions = {
10. method,
11. headers: authHeader()
12. }
13. if (body) {
14. requestOptions.headers['Content-Type'] = 'application/json'
15. requestOptions.body = JSON.stringify(body)
16. }
18. return fetch(API\_PATH + path, requestOptions).then(handleResponse)
19. }
20. }
21. function authHeader() {
22. const auth = useAuthStore()
23. if (auth.user.token)
24. return { Authorization: `Bearer ${auth.user.token}` }
26. return {}
27. }
28. function handleResponse(response) {
29. return response.text().then(async text => {
30. const data = tryParceJSON(text)
32. if (!response.ok) {
33. const auth = useAuthStore()
34. if ([401].includes(response.status) && auth.user.token)
35. auth.logout()
37. if (response.status == 429) {
38. await sleep(5000)
39. }
40. const message = (data?.message || data) || response.statusText
41. console.error(`fetchWrapper: ${response.status}: ${message}`)
42. return Promise.reject({
43. response,
44. status: response.status,
45. message,
46. })
47. }
48. return data
49. })
50. }
51. function tryParceJSON(text) {
52. try {
53. return JSON.parse(text)
54. }
55. catch (e) {
56. return text
57. }
58. }
59. export default fetchWrapper
60. Обёртка над localStorage (см. лист. 31)

Этот код представляет собой функцию, которая обеспечивает функционал хранения данных в localStorage. Функция storageWrapper принимает ключ и возвращает объект, который содержит данные из localStorage с указанным ключом. Функция storageWrapper также обеспечивает функционал отслеживания изменений в объекте и сохранения данных в localStorage с указанным ключом.

Листинг 31 – Обёртка над fetch /src/helpers/storageWrapper.js

1. import { watchDebounced } from "@vueuse/core"
2. import { reactive } from "vue"
3. const readStorage = (key) => {
4. try {
5. return JSON.parse(localStorage.getItem(key)) || {}
6. } catch (e) {
7. return {}
8. }
9. }
10. export const storageWrapper = (key = 'data') => {
11. const dataStorage = reactive(readStorage(key))
12. watchDebounced(
13. () => dataStorage,
14. () => {
15. localStorage.setItem(key, JSON.stringify(dataStorage))
16. },
17. { deep: true, debounce: 250 }
18. )
20. return dataStorage
21. }
22. export default storageWrapper
23. Функция с задержками при множественном вызове (см. лист. 32)

Этот код представляет собой функцию, которая обеспечивает функционал debounce с немедленным вызовом. Функция debounceImmediate принимает функцию и время задержки в миллисекундах и возвращает обёртку вокруг функции, которая вызывает функцию немедленно, а затем задерживает вызов функции на указанное время.

Листинг 32 – Функция с задержками при мульти-вызове /src/helpers/debounceImmediate.js

1. export const debounceImmediate = (func, ms) => {
2. let timeout
3. let isAsked = false
4. let isDoubleAsked = false
5. const createTimeout = (...args) => {
6. timeout = setTimeout(() => {
7. isAsked = false
8. if (isDoubleAsked) {
9. func(...args)
10. isDoubleAsked = false
11. }
12. }, ms)
13. }
14. return (...args) => {
15. if (!isAsked) {
16. func(...args)
17. createTimeout(...args)
18. }
19. if (isAsked) {
20. isDoubleAsked = true
21. clearTimeout(timeout)
22. createTimeout(...args)
23. }
24. isAsked = true
25. }
26. }
27. export default debounceImmediate

## Маршрутизатор

Vue Router — это официальный маршрутизатор для Vue.js, который позволяет создавать одностраничные приложения (SPA) с поддержкой истории браузера. Он предоставляет мощный API для навигации между представлениями и манипуляции URL.

Маршрутизатор проекта представлен на листинге 33.

Листинг 33 – Код маршрутизатора /src/router/index.js

1. export default createRouter({
2. history: BASE\_URL
3. ? createWebHistory    (import.meta.env.VITE\_BASE\_URL)
4. : createWebHashHistory(import.meta.env.VITE\_BASE\_URL),
5. routes: [
6. {
7. path: '/', // Главная
8. name: 'home',
9. component: HomeView
10. },
11. {
12. path: '/album/:albumHash?', // Поддержка старого маршрута
13. name: 'albumLegacy',
14. redirect: to => ({
15. name: 'openAlbum',
16. params: { album: to.params.albumHash },
17. query: to.query,
18. })
19. },
20. {
21. path: '/@:user', // Корневой альбом пользователя
22. name: 'userRoot',
23. component: HomeView
24. },
25. {
26. path: '/:album', // Альбом по алиасу или хешу
27. name: 'openAlbum',
28. component: HomeView
29. },
30. {
31. path: '/:album/:type/:media', // Просмотр медиа
32. name: 'openAlbumMedia',
33. component: HomeView
34. },
35. {
36. path: '/:album/:trueAlbum/:type/:media', // Просмотр медиа во вложенном альбоме
37. name: 'openAlbumNestedMedia',
38. component: HomeView
39. },
40. {
41. path: '/:catchAll(.\*)', // Неизвестный путь
42. name: 'notFound',
43. component: PageNotExist
44. },
45. ]
46. })

## Конфигурационный файл

Конфигурационный файл (см. лист. 34) предназначен для установки значение по умолчанию, если отсутствует поле в файле окружения (.env). В данном случае если не будет задан путь до API сервере — автоматически будет запрашивать на том же домене, что и фронт, по пути «api/».

Листинг 34 – Часть конфигурационного файла /src/config/index.js

1. export const API\_PATH = import.meta.env.VITE\_API\_URL || 'api'

## Представления

Папка «views» используется для хранения основных компонентов, которые представляют разные страницы в приложении. Эти компоненты могут содержать логику и стили и они должны быть использованы в качестве роутов в маршрутизаторе Vue Router.

1. «Страницы не существует» (см. лист. 35)

Страница с кнопками назад и домой, которая возникает если ввести неизвестный маршрут. Результат изображён на рисунке 48.

Листинг 35 – Код страницы 404 /src/views/PageNotExist.vue

1. <template>
2. <div class="outer">
3. <div class="card">
4. <h2>404</h2>
5. <p>Page not found</p>
6. <a v-if="$router.options.history.state.position" class="btn" @click="$router.back()">
7. Back
8. </a>
9. <RouterLink class="btn" to="/">To Home</RouterLink>
10. </div>
11. </div>
12. </template>
13. <style>
14. .outer {
15. height: 100vh;
16. display: flex;
17. justify-content: center;
18. align-items: center;
19. }
20. .card {
21. p {
22. text-align: center;
23. font-size: 48px;
24. }
25. }
26. </style>

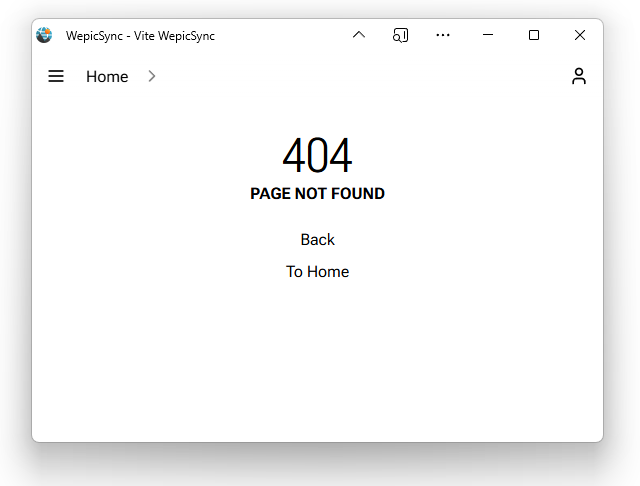


Рисунок 48 – Результат разработки PageNotExist

1. Главная страница (см. лист. 36)

Компонент, который используется для отображения альбомов и медиа. Компонент использует функциональность Vue Router для слежения за изменениями в роуте и обновления соответствующих картинок. Реализован бесконечный скролл. Результат работы представлен на рисунке 49 и рисунке 50.

Листинг 36 – Часть логики главной страницы /src/views/HomeView.vue

1. // Параметры в адресной строке
2. const  {
3. limit, sort, trueReverse, tags, nested, fullImage, ratings, types,
4. targetUser, targetAlbum, albumData, imageData, randomSeed,
5. } = storeToRefs(useAlbumParamsStore())
7. // Функция очистки всех медиа
8. const cleanUpWall = () => {
9. //console.log('clean')
10. canLoadMore.value = false
11. isLoading  .value = false
12. images     .value = []
13. retries = 0
14. imgRetryCounts.value = {}
15. currentPage = 1
16. canLoadMore.value = true
17. }
18. // Наблюдение за параметрами в адресной строке, при изменении — перезапрашиваем медиа
19. watch(
20. [targetAlbum, limit, sort, trueReverse, nested, tags, randomSeed],
21. () => {
22. //console.log(val1, val2)
23. debounceImmediate(() => {
24. //console.log('imgs', route, isLoading.value)
25. if (isLoading.value) watch(
26. isLoading,
27. cleanUpWall,
28. { once: true }
29. )
30. else
31. cleanUpWall()
32. }, 500)()
33. },
34. )
35. // Косметические параметры и URL на превью
36. const {
37. size, gap, extGap, radius, orientation, scroll,
38. ambient, albumsLayout, albumPreviewSize, imagePreviewSize,
39. } = storeToRefs(useSettingsStore())
40. const imgSign = ref(null)
41. // Порционный запрос картинок
42. let currentPage = 1
43. let retries = 0
44. const isLoading   = ref(null)
45. const canLoadMore = ref(true)
46. const images      = ref([])
47. const loadMore = async () => {
48. if (!canLoadMore.value || isLoading.value) return
49. isLoading.value = true
50. await fetchWrapper.get(
51. urls.albumImages(targetAlbum.value, {
52. page: currentPage,
53. limit: limit.value,
54. tags: tags.value,
55. sort: sort.value,
56. isReverse: trueReverse.value,
57. nested: nested.value,
58. seed: sort.value === 'random' ? randomSeed.value : null,
59. ratings: ratings.value,
60. types: types.value,
61. })
62. ).then((data) => {
63. albumData.value.nestedImagesCount = nested.value ? data.total : null
64. if (!canLoadMore.value) return
65. canLoadMore.value = !(data.total < currentPage \* limit.value)
66. albumData.value.sign = imgSign.value = data?.sign
67. const newImages = data.pictures
68. newImages.forEach(element => {
69. const parts = element.name.split('.')
70. element.ext = parts.length === 1 ? 'no ext' : parts.at(-1)
71. element.name = parts.slice(0, -1).join('.')
72. element.ratio = element.width / element.height
73. }
74. images.value.push(...newImages)
75. currentPage++
76. isLoading.value = false
77. }).catch(async (error) => {})
78. }

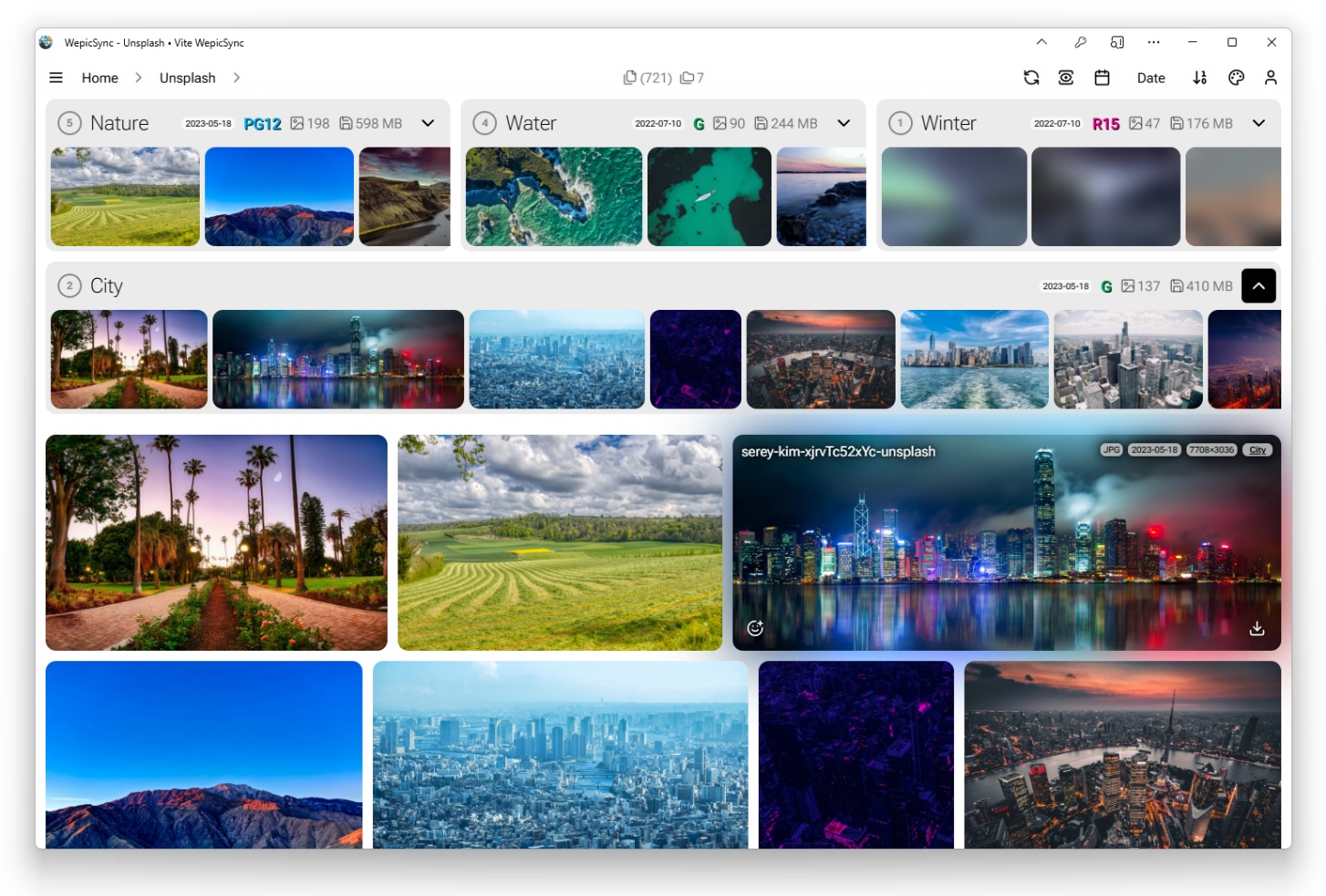


Рисунок 49 – Результат разработки HomeView



Рисунок 50 – Карточка медиа из HomeView

# Глава 5. Проектирование и разработка мобильного приложения

Мобильное приложение WepicSync разработано для синхронизирования локальных картинок пользователя на альбомы в экземплярах сервера проекта, а также возможность назначать теги, делиться альбомами и просматривать свои и доступные чужие альбомы. Разработка велась в Microsoft Visual Studio

Приложение разработано на .NET MAUI с применением паттерна MVVM (Model-View-ViewModel), что позволяет разделить логику приложения, представление и данные. Это обеспечивает лёгкость в расширении функционала и тестировании. В качестве основного инструмента управления состоянием используется CommunityToolkit.Mvvm.

Приложение взаимодействует с серверной частью, реализованной на Laravel, через REST API. Все запросы проходят аутентификацию и авторизацию с использованием токенов, что обеспечивает безопасность работы с системой.

Приложение нацелено на Android 5 платформу, но с помощью кроссплатформенности MAUI приложение можно запустить на iOS 11, Windows 10, macOS 10. Полный функционал только на Android, на остальных платформах доступен только просмотр альбомов на сервере.

## Зависимости проекта

1. *.NET 8.0* – модульная платформа для разработки программного обеспечения с открытым исходным кодом. Совместима с такими операционными системами как Windows, Linux и macOS. Была выпущена компанией Microsoft.
2. *С# 12.0*–объектно-ориентированный язык программирования общего назначения.
3. *Microsoft.Maui.Controls 8.0* – основная библиотека .NET MAUI, предоставляющая UI-элементы и функциональность для создания кроссплатформенных приложений.
4. *CommunityToolkit.Maui 9.1* – набор инструментов для .NET MAUI, предоставляющий дополнительные контролы, расширения и утилиты для упрощения разработки. Включает элементы UI, анимации, конвертеры и другие полезные компоненты.
5. *CommunityToolkit.Mvvm 8.4* – библиотека для реализации шаблона MVVM (Model-View-ViewModel). Упрощает создание привязок данных, команд и других элементов MVVM.
6. *sqlite-net-pcl 1.9* – легковесная ORM (Object-Relational Mapping) для работы с SQLite. Упрощает работу с локальной базой данных SQLite в приложении.
7. *Humanizer 2.14* – библиотека для работы с текстом, датами, числами и другими типами данных. Позволяет легко форматировать строки, например, преобразовывать числа в слова или склонять слова.

## Структура проекта

Мобильное приложение построено на паттерне MVVM (Model-View-ViewModel):

1. View отвечает только за отображение данных.
2. ViewModel содержит бизнес-логику и взаимодействует с API.
3. Model представляет структуру данных, получаемых с сервера.

Структура проекта включает в себя следующие основные каталоги и основные файлы:

1. Api/ – типизация маршрутов, тел запросов и ответов серверной части
   1. Request/ – схемы отправляемых тел запросов
   2. Response/ – схемы приходящих тел ответов
   3. UrlOptions/ – схемы для параметров в строке маршрута
   4. URLs.cs – список всех используемых маршрутов
2. Popups/ – всплывающие окна
3. Converters/ – конверторы JSON и Binding’ов
4. Models/ – общие модели данных
5. Resources/ – ресурсы стилей и иконок
6. Utils/ – вынесенные часто используемые утилиты
   1. AuthData.cs – статичные данные пользователя
   2. LocalData.cs – статичные данные локальных картинок
   3. RemoteAlbumsData.cs – статичные данные картинок на сервере
   4. ServerData.cs – статичные общие данные сервера
   5. Fetcher.cs – обёртка над HttpClient
   6. LocalDB.cs – обёртка над SQLite-NET
   7. PictureSender.cs – фоновый отправитель картинок
   8. Helpers.cs – не категорированные простые утилиты
7. ViewModels – бизнес-логика страниц
8. Views/ – разметка страниц интерфейса
9. App.xaml – глобальные настройки приложения, стили
10. AppShell.xaml – конфигурация навигации между страницами

## Реализация моделей

Модели представляют собой абстрактные представления данных, структуры или поведения системы, которые используются для организации и обработки информации. Они определяют, как данные хранятся, обрабатываются и связаны друг с другом. Модели часто используются в различных парадигмах программирования, таких как объектно-ориентированное программирование (ООП) или архитектурные подходы, например, MVC (Model-View-Controller) или в текущем проекте MVVM (Model-View-ViewModel).

Модель User представляет сущность пользователя в системе. Она включает такие атрибуты, как уникальный идентификатор, логин, пароль, имя (никнейм) и роль. Эта модель используется для хранения и обработки данных о пользователях, а также для взаимодействия с другими компонентами системы. Код представлен на листинге 37.

Листинг 37 – Модель пользователя

1. public class User
2. {
3. [JsonPropertyName("id" )] public required ulong Id { get; init; }
4. [JsonPropertyName("login" )] public string? Login { get; set; }
5. [JsonPropertyName("password")] public string? Password { get; set; }
6. [JsonPropertyName("name" )] public required string Nickname { get; set; }
7. [JsonPropertyName("role" )] public string? Role { get; set; } = "user";
8. }

Модель ServerSettings представляет настройки сервера, которые доступны для клиентов. Она включает такие параметры, как разрешенные MIME-типы для загрузки, допустимые размеры превью, ограничение на бесплатное хранилище и флаг, отключающий загрузку файлов. Эта модель используется для передачи клиентам информации о текущих настройках сервера. Код представлен на листинге 38.

Листинг 38 – Модель публичных серверных настроек

1. public class ServerSettings
2. {
3. [JsonPropertyName("allowed\_upload\_mimes")]
4. public List<string> AllowedUploadMimes { get; set; } = ["png", "jpg", "jpeg"];
5. [JsonPropertyName("allowed\_preview\_sizes")]
6. public List<int> AllowedPreviewSizes { get; set; } = [144, 240, 360, 480, 20, 1080];
7. [JsonPropertyName("free\_storage\_limit")]
8. public ulong? FreeStorageLimit { get; set; }
9. [JsonPropertyName("is\_upload\_disabled")]
10. public bool IsUploadDisabled { get; set; } = false;
11. public DateTime GotAt { get; set; }
12. }

Модель UploadsAlbum отвечает за управление процессом загрузки изображений в альбом. Она включает очередь загрузок, коллекцию загружаемых элементов, а также свойства для отслеживания прогресса, скорости загрузки и оставшегося времени (ETA). Модель автоматически обновляет свои свойства при изменении состояния загрузок благодаря использованию атрибута ObservableProperty. Код представлен на листинге 39.

Листинг 39 – Модель загрузок альбома

1. public partial class UploadsAlbum : ObservableObject
2. {
3. public readonly ConcurrentQueue<UploadItem<PictureLocal>> UploadQueue = [];
4. public required AlbumSynced Album { get; set; }
5. public ObservableCollection<UploadItem<PictureLocal>> Uploads { get; } = [];
6. [ObservableProperty]
7. [NotifyPropertyChangedFor(nameof(Progress))]
8. [NotifyPropertyChangedFor(nameof(ETA))]
9. public int uploadedCount = 0;
10. [ObservableProperty]
11. public ulong averageSpeed = 0;
12. [SetsRequiredMembers]
13. public UploadsAlbum(AlbumSynced album, IEnumerable<UploadItem<PictureLocal>>?
14. ploadPictures = null) {
15. Album = album;
16. if (uploadPictures != null)
17. Uploads = new(uploadPictures);
18. Uploads.CollectionChanged += OnUploadsChanged;
19. }
20. [SetsRequiredMembers]
21. public UploadsAlbum(AlbumSynced album, bool uploadFromLocal) {
22. Album = album;
23. if (uploadFromLocal) {
24. var pictures = album.LocalPictures.OfType<PictureLocal>().ToList();
25. foreach (var picture in pictures) {
26. UploadItem<PictureLocal> uploadItem = new(picture);
27. Uploads.Add(uploadItem);
28. UploadQueue.Enqueue(uploadItem);
29. }
30. }
31. Uploads.CollectionChanged += OnUploadsChanged;
32. }
33. public int TotalCount => Uploads.Count;
34. public ulong TotalBytes => Uploads.Aggregate(0UL, (sum, u) => sum + u.Item.Size);
35. public double Progress => (double)UploadedCount / TotalCount;
36. public TimeSpan ETA
37. {
38. get {
39. if (UploadedCount == 0)
40. return TimeSpan.Zero;
41. var completedUploads = Uploads.Where(u => u.TimeSpent != default).ToList();
42. if (completedUploads.Count == 0)
43. return TimeSpan.Zero;
44. ulong totalBytesUploaded = completedUploads
45. .Aggregate(0UL, (sum, u) => sum + u.Item.Size);
46. TimeSpan totalTimeSpent = completedUploads
47. .Where(u => u.TimeSpent != default)
48. .Aggregate(TimeSpan.Zero, (sum, u) => {
49. if (u.TimeSpent is not TimeSpan spent)
50. return sum;
51. return sum + spent;
52. });
53. if (totalTimeSpent == TimeSpan.Zero)
54. return TimeSpan.Zero;
55. AverageSpeed =
56. *// bytes per second*
57. (ulong)(totalBytesUploaded / totalTimeSpent.TotalSeconds);
58. ulong remainingBytes = Uploads
59. .Where(u => u.TimeSpent == default)
60. .Aggregate(0UL, (sum, u) => sum + u.Item.Size);
61. return TimeSpan.FromSeconds(remainingBytes / AverageSpeed);
62. }
63. }
64. }

### Альбомы

Интерфейсы IAlbum и IAlbumLocal, а также абстрактный класс AlbumBase определяют базовую структуру для работы с альбомами. Они включают такие свойства, как название альбома, количество изображений и пути к миниатюрам. Интерфейс IAlbumLocal расширяет функциональность, добавляя поддержку локальных путей и изображений. Код представлен на листинге 40.

Листинг 40 – Вспомогательные интерфейсы и абстрактный класс альбома

1. public interface IAlbum
2. {
3. *// Свойства*
4. public string Name { get; set; }
5. *// Геттеры*
6. public int PicturesCount { get; }
7. public List<string> ThumbnailPaths { get; }
8. }
9. public abstract class AlbumBase : IAlbum
10. {
11. *// Свойства*
12. public virtual string Name { get; set; }
13. *// Геттеры*
14. public abstract int PicturesCount { get; }
15. public abstract List<string> ThumbnailPaths { get; }
16. }
17. public interface IAlbumLocal : IAlbum
18. {
19. *// Свойства*
20. [Ignore] public int? NameDuplicaIndex { get; set; }
21. public List<IPictureLocal> LocalPictures { get; set; }
22. public string LocalPath { get; set; }
23. }
24. public interface IAlbumLocal : IAlbum
25. {
26. *// Свойства*
27. [Ignore] public int? NameDuplicaIndex { get; set; }
28. public List<IPictureLocal> LocalPictures { get; set; }
29. public string LocalPath { get; set; }
30. }

Модель AlbumLocal представляет локальный альбом, который хранится на устройстве пользователя. Она наследует функциональность от AlbumBase и реализует интерфейс IAlbumLocal. Модель включает свойства для пути к альбому, списка локальных изображений и индекса дубликатов имени. Также реализовано ленивое получение путей к миниатюрам для оптимизации производительности.

Модель AlbumRemote представляет альбом, который хранится на удалённом сервере. Она включает такие свойства, как идентификатор альбома, название, дата создания, количество изображений, информация о доступе и приглашениях, а также данные о владельце и превью изображений. Модель также поддерживает обновление своих свойств при изменении данных.

Модель AlbumSynced объединяет функциональность локальных и удалённых альбомов, наследуясь от AlbumRemote и реализуя интерфейс IAlbumLocal. Он содержит свойства для хранения пути к локальному альбому, списка локальных изображений и индекса дубликатов имени. Геттеры позволяют определить количество синхронизированных, локальных и удалённых изображений.

### Картинки

Интерфейс IPicture определяет общую структуру для работы с изображениями, включая свойства имени, хеша, размера, разрешения и даты. Также он содержит геттеры для путей к оригиналу и миниатюре, а также флаги, указывающие на тип изображения (локальное, синхронизированное или удалённое). Абстрактный класс PictureBase реализует IPicture, предоставляя базовую логику для наследуемых классов. Он содержит виртуальные свойства и реализует флаги принадлежности к локальным или удалённым альбомам. Код представлен на листинге 41.

Листинг 41 – Вспомогательные интерфейсы и абстрактный класс картинки

1. public interface IPicture
2. {
3. *// Свойства*
4. public string Name { get; set; }
5. public string Hash { get; set; }
6. public ulong Size { get; set; }
7. public int Width { get; set; }
8. public int Height { get; set; }
9. public DateTime Date { get; set; }
10. public IAlbum Album { get; set; }
11. *// Геттеры*
12. public string OriginalPath { get; }
13. public string ThumbnailPath { get; }
14. public bool IsLocal { get; }
15. public bool IsSynced { get; }
16. public bool IsRemote { get; }
17. public bool IsRemoteNonOwned { get; }
18. public bool IsRemoteOwned { get; }
19. }
20. public abstract class PictureBase : IPicture
21. {
22. *// Свойства*
23. public virtual string Name { get; set; }
24. public virtual string Hash { get; set; }
25. public virtual ulong Size { get; set; }
26. public virtual int Width { get; set; }
27. public virtual int Height { get; set; }
28. public virtual DateTime Date { get; set; }
29. [Ignore]
30. public virtual IAlbum Album { get; set; }
32. *// Геттеры*
33. public abstract string OriginalPath { get; }
34. public abstract string ThumbnailPath { get; }
35. }
36. public interface IPictureLocal : IPicture
37. *// Свойства*
38. public string LocalPath { get; set; }
39. [JsonIgnore]
40. public AlbumLocal SpecificAlbum
41. {
42. get => (AlbumLocal)Album;
43. set => Album = value;
44. }
45. }

Модель PictureLocal представляет изображение, хранящееся на устройстве пользователя. Он наследуется от PictureBase и реализует IPictureLocal, добавляя свойство LocalPath. Название файла определяется автоматически на основе пути. Геттеры возвращают оригинальный путь изображения и его миниатюру, совпадающие с LocalPath. Флаги указывают, что изображение является локальным, а также определяют его принадлежность к синхронизированному альбому.

Модель PictureRemote описывает изображение, загруженное на удалённый сервер. Он содержит уникальный Id, хеш, размер, разрешение и дату загрузки. Геттеры формируют ссылки на оригинал и миниатюру с сервера. Также предусмотрены флаги, указывающие статус изображения: является ли оно удалённым, принадлежит ли владельцу или другому пользователю.

Модель PictureSynced объединяет данные локального и удалённого изображений, наследуясь от PictureRemote и реализуя IPictureLocal. Включает свойства локального пути и синхронизированного альбома. Геттеры возвращают локальные пути, а методы Update позволяют обновлять данные как с локального, так и с удалённого источника. Флаги отражают смешанный статус изображения — оно одновременно локальное и удалённое, но не является строго удалённым или строго локальным.

## Реализация утилит

Утилита AuthData управляет авторизационной информацией, включая токен пользователя, данные аккаунта и статистику. Данные хранятся в SecureStorage и Preferences для сохранности между сессиями. Свойства автоматически загружают данные при первом обращении, обеспечивая ленивую инициализацию.

Методы SaveAndNavigate и TryExitAndNavigate управляют входом и выходом пользователя, сохраняя токен и данные при успешной авторизации или удаляя их при выходе. Метод Update обновляет информацию о пользователе и его квоте, синхронизируя данные с сервером. Код представлен на листинге 42.

Листинг 42 – Авторизационные данные пользователя

1. public static class AuthData
2. {
3. public static void Save(string? token, User? user) {
4. Token = token;
5. User = user;
6. }
7. private static string? \_token = null;
8. public static string? Token
9. {
10. get => \_token ??= SecureStorage.GetAsync("token").Result;
11. set {
12. \_token = value;
13. if (value == null)
14. SecureStorage.Remove("token");
15. else
16. SecureStorage.SetAsync("token", value);
17. }
18. }
19. private static User? \_user = null;
20. public static User? User {
21. get {
22. if (\_user == null) {
23. var setting = Preferences.Get("user", null);
24. if (setting != null)
25. \_user = JsonSerializer.Deserialize<User>(setting);
26. }
27. return \_user;
28. }
29. set {
30. \_user = value;
31. if (value == null)
32. Preferences.Remove("user");
33. else
34. Preferences.Set("user", JsonSerializer.Serialize(value));
35. }
36. }
37. private static UserStats? \_stats = null;
38. public static UserStats? Stats {
39. get {
40. if (\_stats == null)
41. {
42. var setting = Preferences.Get("stats", null);
43. if (setting != null)
44. \_stats = JsonSerializer.Deserialize<UserStats>(setting);
45. }
46. return \_stats;
47. }
48. set {
49. \_stats = value;
50. if (value == null)
51. Preferences.Remove("stats");
52. else
53. Preferences.Set("stats", JsonSerializer.Serialize(value));
54. }
55. }
56. }

Утилита ServerData управляет путём подключения к серверу и данные о нём, сохраняя URL, настройки и историю подключений в Preferences. Он автоматически загружает сохранённые данные и обновляет их при успешном соединении.

Методы TrySaveAndNavigate запрашивают настройки сервера и сохраняет серверные данные (путь подключения, правила), а ForgetAndNavigate сбрасывает данные и отправляет пользователя на выбор нового сервера. PastUrls хранит историю подключений, добавляя новые адреса при подключении. Код представлен на листинге 43.

Листинг 43 – Серверные данные

1. public static class ServerData
2. {
3. private static Uri? \_url = null;
4. public static Uri? Url {
5. get {
6. if (\_url == null) {
7. var url = Preferences.Get("url", null);
8. if (url != null)
9. \_url = new Uri(url);
10. }
11. return \_url;
12. }
13. set {
14. \_url = value;
15. if (value == null)
16. Preferences.Remove("url");
17. else
18. Preferences.Set("url", value.ToString());
19. }
20. }
21. private static ServerSettings? \_settings = null;
22. public static ServerSettings? Settings {
23. get {
24. if (\_settings == null) {
25. var setting = Preferences.Get("server\_settings", null);
26. if (setting != null)
27. \_settings = JsonSerializer.Deserialize<ServerSettings>(setting);
28. }
29. return \_settings;
30. }
31. set {
32. \_settings = value;
33. if (value == null)
34. Preferences.Remove("server\_settings");
35. else
36. Preferences.Set("server\_settings", JsonSerializer.Serialize(value));
37. }
38. }
39. private static ObservableCollection<string>? \_pastUrls = null;
40. public static ObservableCollection<string> PastUrls {
41. get {
42. if (\_pastUrls == null) {
43. var pastUrls = Preferences.Get("past\_urls", null);
44. Debug.WriteLine("pastUrls: " + pastUrls);
45. \_pastUrls = new(pastUrls?.Split(';') ?? []);
46. \_pastUrls.CollectionChanged += OnPastUrlsChanged;
47. }
48. return \_pastUrls;
49. }
50. }
51. private static void OnPastUrlsChanged(object? sender, NotifyCollectionChangedEventArgs e) {
52. Debug.WriteLine("OnPastUrlsChanged: " + JsonSerializer.Serialize(PastUrls));
53. Preferences.Set("past\_urls", string.Join(";", PastUrls));
54. }
55. }

Утилита RemoteAlbumsData управляет списком удалённых альбомов, разделяя их на AlbumsOwn (собственные) и AlbumsAccessible (доступные). Метод FillAlbums запрашивает данные с сервера, обновляет время последнего запроса (GotAt) и перезаписывает коллекции. Код представлен на листинге 44.

Листинг 44 – Данные удалённых альбомов

1. public static class RemoteAlbumsData
2. {
3. public static DateTime? GotAt { get; set; } = null;
4. public static ObservableCollection<AlbumRemote> AlbumsOwn { get; } = [];
5. public static ObservableCollection<AlbumRemote> AlbumsAccessible { get; } = [];
6. public static async Task FillAlbums(
7. Action<bool>? setIsFetch = null,
8. Action<string>? setError = null,
9. CancellationToken cancellationToken = default
10. ) {
11. (var res, var body) = await FetchAsync<AlbumsResponse>(
12. HttpMethod.Get, URLs.Albums,
13. setIsFetch, setError,
14. cancellationToken: cancellationToken
15. );
16. if (body == null) {
17. return;
18. }
19. GotAt = DateTime.Now;
20. AlbumsOwn.Clear();
21. AlbumsAccessible.Clear();
22. foreach (var album in body.Accessible)
23. AlbumsAccessible.Add(album);
24. foreach (var album in body.Own)
25. AlbumsOwn.Add(album);
26. }
27. }

Утилита LocalData управляет локальными изображениями и альбомами на устройстве, используя локальную базу данных для хранения информации о синхронизированных файлах. Метод FillPictures загружает данные о картинках, извлекая их через ContentResolver с помощью курсора, который собирает путь к файлу, размер, ширину, высоту и дату изменения. Эти данные используются для создания объектов изображений и организации их в альбомы. Если альбом или картинка уже есть в базе, они помечаются как синхронизированные, а новые добавляются в коллекции Albums и Pictures. Код представлен на листинге 45.

Листинг 45 – Данные картинок и альбомов на устройстве

1. public static class LocalData {
2. public static async Task<bool> CheckPermissions() {
3. #if ANDROID
4. return ContextCompat.CheckSelfPermission(
5. Platform.CurrentActivity,
6. (int)Build.VERSION.SdkInt >= 33
7. ? Manifest.Permission.**ReadMediaImages**
8. : Manifest.Permission.**ReadExternalStorage**
9. ) == Permission.**Granted**;
10. #else
11. throw new PlatformNotSupportedException("К сожалению доступно только для Android");
12. #endif
13. }
14. public static async Task<bool> RequestPermissions() {
15. #if ANDROID
16. bool granted = await CheckPermissions();
17. if (granted) return true;
18. ActivityCompat.RequestPermissions(
19. Platform.CurrentActivity,
20. (int)Build.VERSION.SdkInt >= 33
21. ? [Manifest.Permission.**ReadMediaImages**]
22. : [Manifest.Permission.**ReadExternalStorage**],
23. requestCode: 0
24. );
25. DateTime startTime = DateTime.UtcNow;
26. while (DateTime.UtcNow - startTime < TimeSpan.FromSeconds(10)) {
27. granted = await CheckPermissions();
28. if (granted) return true;
29. await Task.Delay(500);
30. }
31. return false;
32. #else
33. throw new PlatformNotSupportedException("К сожалению доступно только для Android");
34. #endif
35. }
36. public static LocalLoadStatus Status { get; set; } = LocalLoadStatus.**NotLoad**;
37. public static ObservableCollection<IPictureLocal> Pictures { get; } = [];
38. public static ObservableCollection<IAlbumLocal> Albums { get; } = [];
39. public static async Task FillPictures()
40. {
41. Pictures.Clear();
42. Albums.Clear();
43. #if ANDROID
44. Status = LocalLoadStatus.**InLoad**;
45. bool granted = await CheckPermissions();
46. if (!granted)
47. Status = LocalLoadStatus.**NoPermissions**;
48. var contentResolver = Android.App.Application.Context.ContentResolver;
49. string[] projection = [
50. Android.Provider.MediaStore.Images.Media.InterfaceConsts.**Data**,
51. Android.Provider.MediaStore.Images.Media.InterfaceConsts.**Size**,
52. Android.Provider.MediaStore.Images.Media.InterfaceConsts.**Width**,
53. Android.Provider.MediaStore.Images.Media.InterfaceConsts.**Height**,
54. Android.Provider.MediaStore.Images.Media.InterfaceConsts.**DateModified**,
55. ];
56. var sortOrder = $"{projection[4]} DESC";
57. var uri = Android.Provider.MediaStore.Images.Media.ExternalContentUri;
58. if (uri == null) {
59. #if DEBUG
60. \_ = Shell.Current.DisplayAlert("DEBUG ERROR", $"Ошибка получение uri", OK");
61. #endif
62. return;
63. }
64. System.Diagnostics.Debug.WriteLine($"URI: " + uri.ToString());
65. var cursor = contentResolver?.Query(uri, projection, null, null, sortOrder);
66. if (cursor == null) {
67. #if DEBUG
68. \_ = Shell.Current.DisplayAlert("DEBUG ERROR", $"Ошибка получение cursor", "OK");
69. #endif
70. return;
71. }
72. try
73. {
74. int dataColumn = cursor.GetColumnIndexOrThrow(projection[0]);
75. int sizeColumn = cursor.GetColumnIndexOrThrow(projection[1]);
76. int widthColumn = cursor.GetColumnIndexOrThrow(projection[2]);
77. int heightColumn = cursor.GetColumnIndexOrThrow(projection[3]);
78. int dateColumn = cursor.GetColumnIndexOrThrow(projection[4]);
79. await Task.Run(() => {
80. *// Проход по картинкам*
81. while (cursor.MoveToNext()) {
82. var picturePath = cursor.GetString(dataColumn);
83. if (picturePath == null) continue;
84. var albumPath = Path.GetDirectoryName(picturePath);
85. if (albumPath == null) continue;
86. *// Альбом*
87. IAlbumLocal? album;
88. bool isInSync = false;
89. *// Ищем в списке-переменной есть ли альбом уже*
90. album = Albums.FirstOrDefault(a => a.LocalPath == albumPath);
91. if (album == null) {
92. *// Ищем в локальной БД есть ли альбом как синхронизирующийся*
93. album = DB.Table<AlbumSynced>().FirstOrDefault(a => a.LocalPath == albumPath);
94. isInSync = album != null;
95. *// Создаём новый объект альбома если нигде нет ещё*
96. if (!isInSync)
97. album = new AlbumLocal(albumPath);
98. *// Ищем дубликат по имени*
99. var duplica = Albums
100. .Where(d => d.Name == album.Name)
101. .OrderByDescending(d => d.NameDuplicaIndex)
102. .FirstOrDefault();
103. *// Назначаем индекс если новый альбом это дубликат*
104. if (duplica != null)
105. album.NameDuplicaIndex = (duplica.NameDuplicaIndex ??= 1) + 1;
106. *// Добавляем альбом к общему списку локальных альбомов*
107. Albums.Add(album);
108. }
109. else
110. isInSync = album is AlbumSynced;
111. *// Картинка*
112. IPictureLocal? picture = null;
113. *// Ищем в локальной БД есть ли картинка как синхронизированная*
114. if (isInSync) {
115. picture = DB.Table<PictureSynced>().FirstOrDefault(p => p.LocalPath == picturePath);
116. picture.Album = album;
117. }
118. *// Создаём новый объект альбома если в БД нет, т.е. ещё не синхронизирована*
119. picture ??= new PictureLocal() {
120. LocalPath = picturePath,
121. Size = (ulong)cursor.GetLong(sizeColumn),
122. Width = cursor.GetInt(widthColumn),
123. Height = cursor.GetInt(heightColumn),
124. Date = DateTimeOffset.FromUnixTimeSeconds(cursor.GetLong(dateColumn)).DateTime,
125. Album = album,
126. };
127. *// Добавляем картинку в список локальных картинок альбома*
128. album.LocalPictures.Add(picture);
129. *// Добавляем картинку к общему списку локальных картинок*
130. Pictures.Add(picture);
131. }
132. });
133. }
134. catch (Exception ex) {
135. #if DEBUG
136. \_ = Shell.Current.DisplayAlert("DEBUG ERROR", $"FillPictures:\n{ex.Message}", "OK");
137. throw ex;
138. #endif
139. }
140. finally {
141. cursor.Close();
142. }
143. Status = LocalLoadStatus.**Loaded**;
144. #else
145. throw new PlatformNotSupportedException("К сожалению доступно только для Android");
146. #endif
147. }
148. }

Утилита LocalDB представляет собой обёртку над локальной базой данных, используя SQLite для хранения информации о синхронизированных альбомах и картинках. База данных инициализируется с уникальным именем, зависящим от URL сервера и логина пользователя, а затем создаются таблицы AlbumSynced, PictureSynced и PictureDuplica. Метод Reset позволяет сбросить соединение с базой — используется при смене пользователя или сервера. Код представлен на листинге 46.

Листинг 46 – Обёртка над локальной базой данных

1. public static class LocalDB {
2. private static SQLiteConnection? \_db = null;
3. public static SQLiteConnection DB {
4. get {
5. if (\_db != null)
6. return \_db;
7. *// Инициализация*
8. string hashname;
9. try {
10. string host = ServerData.Url?.Host.ToLower() ?? "nohost";
11. string login = AuthData.User?.Login?.ToLower() ?? "nologin";
12. string data = host + "\_" + login;
13. #if DEBUG
14. Debug.WriteLine("LocalDB: Init: data: " + data);
15. #endif
16. hashname = HashUTF8toMD5HEX(data);
17. }
18. catch (Exception ex) {
19. Debug.WriteLine("LocalDB: Init: Exception: " + ex.Message);
20. hashname = "default";
21. }
22. \_db = new(Path.Combine(
23. Environment.GetFolderPath(Environment.SpecialFolder.**LocalApplicationData**),
24. $"local-{hashname}.db"
25. ));
26. \_db.CreateTable<AlbumSynced>();
27. \_db.CreateTable<PictureSynced>();
28. \_db.CreateTable<PictureDuplica>();
29. return \_db;
30. }
31. }
32. public static void Reset() {
33. \_db = null;
34. }
35. }

Утилита Fetcher представляет собой обёртку над HttpClient, упрощая выполнение HTTP-запросов. Он поддерживает отправку запросов с авторизацией (Bearer-токен), возможность сериализации тела запроса в JSON и обработку различных ошибок, таких как тайм-ауты и проблемы соединения. Реализован перегруженный метод FetchAsync<T>, который дополнительно десериализует JSON-ответ в объект указанного типа. Код представлен на листинге 47.

Листинг 47 – Обёртка над HttpClient

1. public static class Fetcher {
2. private readonly static HttpClient \_httpClient = new();
3. public static async Task<HttpResponseMessage?> FetchAsync(
4. HttpMethod method,
5. dynamic path,
6. Action<bool>? setIsFetch = null,
7. Action<string>? setError = null,
8. dynamic? body = null,
9. bool serialize = false,
10. CancellationToken cancellationToken = default
11. ) {
12. Debug.WriteLine("FETCH: Start");
13. setIsFetch?.Invoke(true);
14. setError?.Invoke(null);
15. *// Готовим запрос*
16. Uri fullUrl;
17. if (path is Uri uri) fullUrl = uri;
18. else if (path is string urlEnd) {
19. Try {
20. fullUrl = new Uri($"{URLs.API\_URL}/{urlEnd}");
21. }
22. catch (Exception ex) {
23. setIsFetch?.Invoke(false);
24. setError?.Invoke($"Ошибка URL: {ex.Message}");
25. return new();
26. }
27. }
28. else throw new ArgumentException("path should be of the type: Uri, string");
29. Debug.WriteLine("FETCH: FullUrl: " + fullUrl);
30. var request = new HttpRequestMessage(method, fullUrl);
31. if (AuthData.Token != null)
32. request.Headers.Add("Authorization", $"Bearer {AuthData.Token}");
33. if (body != null) {
34. if (serialize) {
35. string json = JsonSerializer.Serialize(body);
36. Debug.WriteLine("FETCH: REQUEST: JSON: " + json);
37. request.Content = new StringContent(json, System.Text.Encoding.UTF8);
38. }
39. else if (body is HttpContent)
40. request.Content = body;
42. else throw new ArgumentException("body should be of the type HttpContent");
43. }
44. try {
45. *// Запрашиваем*
46. var response = await \_httpClient.SendAsync(request, cancellationToken);
47. *// Обрабатываем ответ*
48. if (!response.IsSuccessStatusCode) {
49. var responseJsonErr = await response.Content.ReadAsStringAsync();
50. try {
51. Debug.WriteLine("FETCH: ERROR: Json: " + responseJsonErr);
52. var responseBodyErr =
53. sonSerializer.Deserialize<ErrorResponse>(responseJsonErr);
54. setError?.Invoke(responseBodyErr?.Message);
55. }
56. catch (Exception ex) {
57. Debug.WriteLine("FETCH: responseJsonErr: " + ex.Message);
58. setError?.Invoke($"Пришёл плохой ответ
59. {(int)response.StatusCode} ({response.ReasonPhrase})");
60. }
61. }
62. setIsFetch?.Invoke(false);
63. return response;
64. }
65. catch (TaskCanceledException ex) {
66. setIsFetch?.Invoke(false);
67. setError?.Invoke("Время ожидания вышло");
68. Debug.WriteLine($"FETCH: TaskCanceledException: {ex.Message}");
69. return null;
70. }
71. catch (HttpRequestException ex) {
72. setIsFetch?.Invoke(false);
73. string message = ex.Message == "Connection failure" ? "Нет соединения" : ex.Message;
74. Debug.WriteLine($"FETCH: HttpRequestException: {message}");
75. setError?.Invoke(message);
76. return null;
77. }
78. }
79. }

Утилита PictureSender управляет фоновой загрузкой изображений, организуя очередь отправки и отслеживая процесс передачи файлов на сервер. Он рассчитывает прогресс загрузки, общий объем переданных данных, среднюю скорость и оставшееся время. При успешной отправке в локальную БД добавляется запись новой синхронизированной картинки.

При успешной отправке изображение обновляется в базе данных как синхронизированное, а в случае ошибки выполняется обработка — либо дубликат добавляется в специальный список, либо изображение корректно привязывается к альбому. Это позволяет избежать повторной загрузки уже существующих файлов и обеспечивает надежную синхронизацию данных между локальным хранилищем и сервером. Код представлен на листинге 48.

Листинг 48 – Фоновый отправитель картинок

1. public partial class PictureSender : ObservableObject {  
    public double Progress => (double)UploadedCount / TotalCount;
2. public int TotalCount => UploadsAlbums.Aggregate(0, (sum, uplAlb) => sum +
3. plAlb.Uploads.Count);
4. public ulong TotalBytes => UploadsAlbums
5. .SelectMany(a => a.Uploads)
6. .Aggregate(0UL, (sum, u) => sum + u.Item.Size);
7. public ulong RemainingBytes => TotalBytes - UploadedBytes;
8. public ulong AverageSpeed => (ulong)(UploadedBytes / TimeSpent.TotalSeconds);
9. public TimeSpan ETA => AverageSpeed == 0
10. ? TimeSpan.Zero
11. : TimeSpan.FromSeconds((double)RemainingBytes / AverageSpeed);
12. private void OnUploadsChanged() {
13. OnPropertyChanged(nameof(TotalCount));
14. OnPropertyChanged(nameof(TotalBytes));
15. OnPropertyChanged(nameof(Progress));
16. OnPropertyChanged(nameof(RemainingBytes));
17. OnPropertyChanged(nameof(ETA));
18. }
19. public void AllManualAlbumUpload() {
20. var albums = LocalData.Albums.OfType<AlbumSynced>().ToList();
21. foreach (var album in albums) {
22. if (album.LocalPictures.OfType<PictureLocal>().Any())
23. ManualAlbumUpload(album);
24. }
25. }
26. public UploadsAlbum ManualAlbumUpload(AlbumSynced album) {
27. var uploadsAlbum = UploadsAlbums.Where(uplAlb => uplAlb.Album == album).FirstOrDefault();
28. if (uploadsAlbum == null) {
29. uploadsAlbum = new(album, true);
30. UploadsAlbums.Add(uploadsAlbum);
31. OnUploadsChanged();
32. return uploadsAlbum;
33. }
34. var pictures = album.LocalPictures.OfType<PictureLocal>().ToList();
35. foreach (var picture in pictures) {
36. if (uploadsAlbum.UploadQueue.Where(uplPic => uplPic.Item == picture).Any())
37. continue;
38. UploadItem<PictureLocal> uploadItem = new(picture);
39. uploadsAlbum.Uploads.Add(uploadItem);
40. uploadsAlbum.UploadQueue.Enqueue(uploadItem);
41. }
42. OnUploadsChanged();
43. return uploadsAlbum;
44. }
45. public void StartUploadIfNotActive() {
46. if (IsUploading) return;
47. \_ = ProcessUploadQueueAsync();
48. }
49. private async Task ProcessUploadQueueAsync() {
50. IsUploading = true;
51. \_cts = new CancellationTokenSource();
52. foreach (var uploadsAlbum in UploadsAlbums) {
53. if (\_cts.Token.IsCancellationRequested)
54. break;
55. while (uploadsAlbum.UploadQueue.TryDequeue(out var uploadItem)) {
56. await UploadPictureAsync(uploadItem, uploadsAlbum);
57. if (\_cts.Token.IsCancellationRequested)
58. break;
59. }
60. }
61. IsUploading = false;
62. \_cts.Dispose();
63. \_cts = null;
64. }
65. public void StopUpload() {
66. \_cts?.Cancel();
67. IsUploading = false;
68. }
69. public async Task<bool?> UploadPictureAsync(
70. UploadItem<PictureLocal> uploadedPicture,
71. UploadsAlbum uploadsAlbum
72. ) {
73. var localPicture = uploadedPicture.Item;
74. AlbumSynced syncedAlbum = uploadsAlbum.Album;
75. ProgressStreamContent streamContent = new(File.OpenRead(localPicture.LocalPath));
76. streamContent.ProgressChanged += (bytes, currBytes, totalBytes) =>
77. uploadedPicture.Progress = (double)currBytes / totalBytes; *//\* 100;*
78. MultipartFormDataContent content = new() {
79. {
80. streamContent, "pictures[0][file]", localPicture.Name
81. }, {
82. new StringContent(localPicture.Date.ToString("yyyy-MM-ddTHH:mm:ss.fffffffzzz")),
83. "pictures[0][date]"
84. }, {
85. new StringContent(localPicture.Name),
86. "pictures[0][name]"
87. }
88. };
89. string? error = null;
90. var startedAt = DateTime.Now;
91. uploadedPicture.StartedAt = DateTime.Now;
92. (var res, var body) = await FetchAsync<PicturesSendResponse>(
93. HttpMethod.Post,
94. URLs.AlbumPictures(syncedAlbum.Id),
95. setError: e => error = e,
96. body: content,
97. cancellationToken: \_cts?.Token ?? CancellationToken.None
98. );
99. var currentTimeSpent = DateTime.Now - startedAt;
100. TimeSpent += currentTimeSpent;
101. UploadedCount++;
102. UploadedBytes += uploadedPicture.Item.Size;
103. uploadsAlbum.UploadedCount++;
104. uploadedPicture.Progress = 1;
105. uploadedPicture.TimeSpent = currentTimeSpent;
106. if (body?.Successful?.Count > 0) {
107. var remotePicture = body.Successful[0];
108. PictureSynced syncedPicture = new(localPicture, remotePicture, syncedAlbum);
109. LocalData. Pictures.ReplaceOrAdd(localPicture, syncedPicture);
110. syncedAlbum.LocalPictures.ReplaceOrAdd(localPicture, syncedPicture);
111. DB.Insert(syncedPicture);
112. return true;
113. }
114. }
115. }

Helpers содержит утилиты для преобразования данных, включая хеширование строк в MD5, форматирование байтов и чисел в удобочитаемый вид. BytesToHuman конвертирует объем данных в КБ, МБ и выше, а CountToHuman сокращает большие числа в тысячи, миллионы и т.д. Код представлен в листинге 49.

Листинг 49 – Не категорированные простые утилиты

1. public static class Helpers {
2. public static string HashUTF8toMD5HEX(string input) {
3. byte[] inputBytes = System.Text.Encoding.UTF8.GetBytes(input);
4. byte[] hashBytes = MD5.HashData(inputBytes);
5. return Convert.ToHexString(hashBytes);
6. }
7. public static readonly string[] BytesSuffixes = ["Б", "КБ", "МБ", "ГБ", "ТБ", "ПБ", "ЭБ"];
8. public static string BytesToHuman(ulong bytes) {
9. int suffixIndex = 0;
10. double count = bytes;
11. while (count >= 1000 && suffixIndex < BytesSuffixes.Length - 1) {
12. count /= 1024;
13. suffixIndex++;
14. }
15. string format = (
16. count >= 100 ||
17. suffixIndex == 0
18. )
19. ? "0"
20. : count >= 10
21. ? "0.0"
22. : "0.00";
23. return count.ToString(format) + " " + BytesSuffixes[suffixIndex];
24. }
26. public static readonly string[] CountSuffixes = ["", "K", "M", "B", "T", "Q", "Qt", "Sx"];
27. public static string CountToHuman(ulong number) {
28. int pow = (int)Floor((number > 0 ? Log(number) : 0) / Log(1000));
29. pow = Min(pow, CountSuffixes.Length - 1);
30. double value = number / Pow(1000, pow);
31. string formatted = value % 1 == 0
32. ? value.ToString("F0")
33. : value.ToString("F" +
34. (3 - Floor(Log10(value) + 1)).ToString()
35. );
36. return formatted + CountSuffixes[pow];
37. }
38. }

## Реализация представлений

### Страница выбора сервера

Разметка ServerSelectorPage описывает страницу выбора сервера. В основе страницы лежит StackLayout, содержащий заголовок, поле для ввода URL, кнопку подключения и список ранее использованных серверов. Возникающие ошибки будут отображены в красной рамке. Также предусмотрен механизм отмены подключения. Раздел с историей серверов представлен с помощью CollectionView, где каждый элемент можно нажать для повторного подключения или удалить. Вся разметка связана с ServerSelectorViewModel (см. лист. 51), который обрабатывает ввод пользователя и управляет логикой подключения. Разметка представлена в листинге 50. Итоговый вид страницы изображён на рисунке 51.

Листинг 50 – Разметка страницы выбора сервера

1. <ContentPage>
2. <ScrollView>
3. <StackLayout Spacing="10" Padding="10" VerticalOptions="Center" MaximumWidth="500">
4. <Label Text="Выбор сервера" HorizontalTextAlignment="**Center**" FontSize="24"/>
5. <Border Padding="10" Stroke="Red"
6. IsVisible="{Binding Error, Converter={StaticResource IsNotNullConverter}}"
7. Background="{AppThemeBinding>
8. <Border.StrokeShape>
9. <RoundRectangle CornerRadius="6"/>
10. </Border.StrokeShape>
11. <StackLayout Spacing="5">
12. <HorizontalStackLayout Spacing="5">
13. <Image Source="triangle\_alert.png" HeightRequest="24" WidthRequest="24">
14. <Image.Behaviors>
15. <tk:IconTintColorBehavior
16. TintColor="{AppThemeBinding Dark=White, Light=Black}"/>
17. </Image.Behaviors>
18. </Image>
19. <Label Text="Ошибка" FontAttributes="**Bold**" FontSize="16"/>
20. </HorizontalStackLayout>
21. <Label Text="{Binding Error}"/>
22. </StackLayout>
23. </Border>
24. <Entry Text="{Binding Url}" Placeholder="Введите URL"
25. Completed="TryNewConnect" Keyboard="Url"/>
26. <Button Text="Подключиться" Command="{Binding TryNewConnectCommand}"/>
27. <StackLayout HeightRequest="30">
28. <Label Text="Отмена"
29. IsVisible="{Binding IsFetch}"
30. TextColor="{AppThemeBinding Dark={StaticResource PrimaryDark},
31. Light={StaticResource Primary}}"
32. TextDecorations="**Underline**" FontSize="16" Padding="5"
33. HorizontalTextAlignment="**Center**">
34. <Label.GestureRecognizers>
35. <TapGestureRecognizer Command="{Binding CancelConnectCommand}"/>
36. </Label.GestureRecognizers>
37. </Label>
38. </StackLayout>
39. </StackLayout>
40. </ScrollView>
41. </ContentPage>

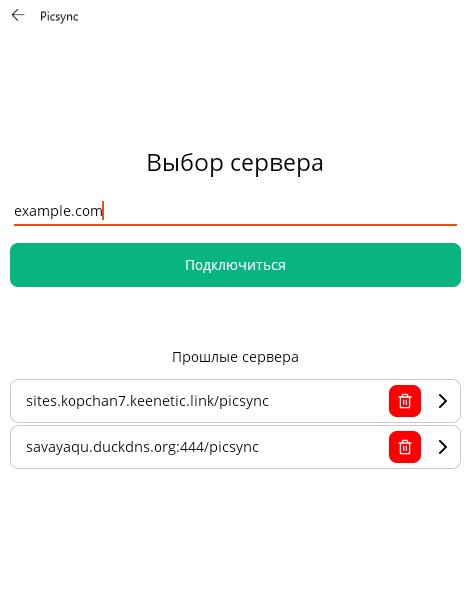


Рисунок 51 – Вид страницы выбора сервера

Листинг 51 – Часть логики страницы выбора сервера

1. public partial class ServerSelectorViewModel : ObservableObject
2. {
3. public ServerSelectorViewModel() {
4. PastUrls.CollectionChanged += (sender, e) =>
5. OnPropertyChanged(nameof(PastUrlsIsVisible));
6. }
7. public bool CanTryNewConnect() => Url != "" && !IsFetch;
8. [RelayCommand(CanExecute = nameof(CanTryNewConnect))]
9. private async Task TryNewConnect() {
10. \_cancellationTokenSource = new();
11. bool isSuccess = await ServerData.TrySaveAndNavigate(
12. Url,
13. isFetch => IsFetch = isFetch,
14. error => Error = error,
15. \_cancellationTokenSource.Token
16. );
17. if (isSuccess)
18. Url = "";
19. }
20. private bool CanTryPastConnect() => !IsFetch;
21. [RelayCommand(CanExecute = nameof(CanTryPastConnect))]
22. private async Task TryPastConnect(string url) {
23. \_cancellationTokenSource = new();
24. await ServerData.TrySaveAndNavigate(
25. url,
26. isFetch => IsFetch = isFetch,
27. error => Error = error,
28. \_cancellationTokenSource.Token
29. );
30. }
31. public bool CanCancelConnect() => IsFetch;
32. [RelayCommand(CanExecute = nameof(CanCancelConnect))]
33. private void CancelConnect() {
34. \_cancellationTokenSource?.Cancel();
35. }
36. [RelayCommand]
37. private void DeletePastUrl(string url) =>
38. PastUrls.Remove(url);
39. }

### Страница входа

Разметка страницы (LoginPage) содержит поля ввода для логина и пароля, кнопку входа и ссылки на выбор сервера и регистрацию. При ошибке отображается предупреждающий блок. Разметка представлена в листинге 52. Итоговый вид изображён на рисунке 52.

Листинг 52 – Разметка страницы входа

1. <ContentPage>
2. <ScrollView>
3. <StackLayout Spacing="10" Padding="10" VerticalOptions="Center" MaximumWidth="400">
4. <Label Text="Вход" HorizontalTextAlignment="**Center**" FontSize="24"/>
5. <Label Text="{Binding Url}" HorizontalTextAlignment="**Center**" Margin="0,10,0,0"/>
6. <Label Text="Выбрать другой сервер" FontSize="16" Padding="5"
7. TextDecorations="**Underline**"
8. HorizontalTextAlignment="**Center**" Margin="0,-5,0,0">
9. <Label.GestureRecognizers>
10. <TapGestureRecognizer Command="{Binding ForgetServerCommand}"/>
11. </Label.GestureRecognizers>
12. </Label>
13. <Border Padding="10" Stroke="Red"
14. Background="{AppThemeBinding Dark={StaticResource RedDark},
15. ight={StaticResource Red}}"
16. IsVisible="{Binding Error, Converter={StaticResource IsNotNullConverter}}">
17. <Border.StrokeShape>
18. <RoundRectangle CornerRadius="6"/>
19. </Border.StrokeShape>
20. <StackLayout Grid.Column="1" Spacing="5">
21. <HorizontalStackLayout Spacing="5">
22. <Image Source="triangle\_alert.png" HeightRequest="24" WidthRequest="24">
23. <Image.Behaviors>
24. <tk:IconTintColorBehavior TintColor="{AppThemeBinding
25. ark=White, Light=Black}"/>
26. </Image.Behaviors>
27. </Image>
28. <Label Text="Ошибка" FontAttributes="**Bold**" FontSize="16"/>
29. </HorizontalStackLayout>
30. <Label Text="{Binding Error}"/>
31. </StackLayout>
32. </Border>
33. <Entry Text="{Binding Login}" Placeholder="Введите логин" Completed="FocusToPassw"/>
34. <Entry Text="{Binding Password}" Placeholder="Введите пароль" Completed="TryLogin" IsPassword="True" x:Name="PasswordEntry"/>
35. <Button Text="Войти" Command="{Binding TryLoginCommand}"/>
37. <Label Text="У вас нет учётной записи?" HorizontalTextAlignment="**Center**"/>
38. <Label Text="Зарегистрироваться"
39. TextDecorations="**Underline**" FontSize="16" Padding="5"
40. HorizontalTextAlignment="**Center**" Margin="0,-5,0,0">
41. <Label.GestureRecognizers>
42. <TapGestureRecognizer Command="{Binding GoToSignupCommand}"/>
43. </Label.GestureRecognizers>
44. </Label>
45. </StackLayout>
46. </ScrollView>
47. </ContentPage>

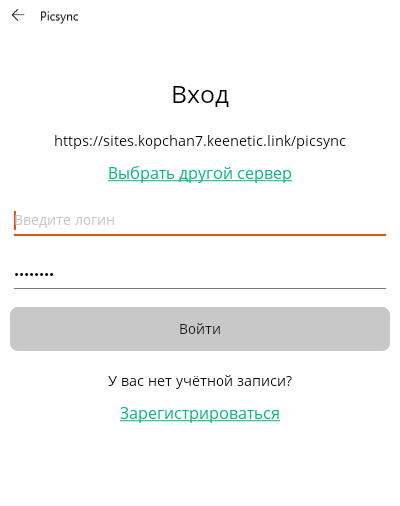


Рисунок 52 – Вид страницы входа

LoginViewModel управляет состоянием страницы: хранит логин, пароль, ошибки, статус загрузки (IsFetch) и URL сервера. Метод TryLogin выполняет асинхронный запрос на сервер, обрабатывает ошибки и, при успешной аутентификации, сохраняет токен и информацию о пользователе. Реализованы команды для смены сервера (ForgetServer) и перехода к регистрации (GoToSignup). Код представлен в листинге 53.

Листинг 53 – Логика страницы входа

1. public partial class LoginViewModel : ObservableObject {
2. public string Url => ServerData.Url?.ToString();
3. [RelayCommand]
4. public void ForgetServer() => ServerData.ForgetAndNavigate();
5. [RelayCommand]
6. private void GoToSignup() => Shell.Current.GoToAsync("//Signup");
7. private bool CanLogin() =>
8. Login != "" &&
9. Password != "" &&
10. !IsFetch;
11. [RelayCommand(CanExecute = nameof(CanLogin))]
12. private async Task TryLogin() {
13. var credentials = new CredentialsRequest(Login, Password);
14. Try
15. {
16. (var res, var body) = await FetchAsync<AuthResponse>(
17. HttpMethod.Post, "login",
18. isFetch => IsFetch = isFetch,
19. error => Error = error,
20. credentials,
21. serialize: true
22. );
23. if (res == null)
24. throw new Exception(Error ?? "Сервер не ответил");
25. if (res.StatusCode == HttpStatusCode.**Unauthorized**)
26. throw new Exception("Неправильный логин и/или пароль");
27. if (!res.IsSuccessStatusCode)
28. throw new Exception(Error ?? $"Пришёл плохой код {(int)res.StatusCode}");
29. if (Error != null)
30. throw new Exception(Error);
31. if (body?.Token == null || body?.User == null)
32. throw new Exception("Ошибка сервера: не пришли нужные данные\n"
33. + await res.Content.ReadAsStringAsync());
34. Password = "";
35. AuthData.SaveAndNavigate(body.Token, body.User);
36. }
37. catch (Exception ex) {
38. Error = ex.Message;
39. }
40. }
41. public void Update() {
42. OnPropertyChanged(nameof(Url));
43. }
44. }

### Страница регистрации

Разметка страницы (SignupPage) содержит поля ввода для логина, никнейма, пароля и его подтверждения, а также кнопки для отправки формы и перехода на страницу входа. Реализована обработка ошибок, включая выделение некорректных полей и отображение сообщений пользователю. Итоговый вид изображён на рисунке 53.

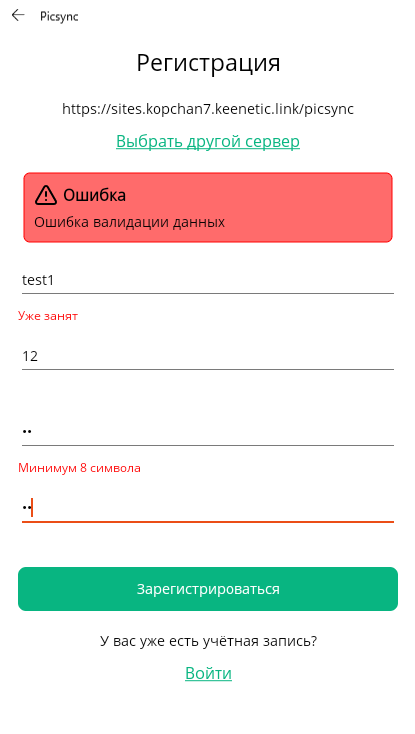


Рисунок 53 – Вид страницы регистрации

Логика страницы реализована в SignupViewModel, где хранятся введенные пользователем данные и выполняется отправка запроса на сервер. В случае ошибок валидации данные выделяются, а серверные ошибки обрабатываются и отображаются. При успешной регистрации пользователя перенаправляют в систему, сохраняя его токен аутентификации. Также есть команды для смены сервера и перехода к странице входа. Код представлен в листинге 54.

Листинг 54 – Часть логики страницы регистрации

1. public partial class SignupViewModel : ObservableObject {
2. [RelayCommand(CanExecute = nameof(CanSignup))]
3. private async Task TrySignup() {
4. Error = null;
5. BadFields = [];
6. if (Password != PasswordConfirm) {
7. Error = "Ошибка валидации данных";
8. BadFields = new()
9. {
10. { "passwordConfirm", ["Пароли не совпадают"] },
11. };
12. return;
13. }
14. var regData = new RegisterRequest(Login, Password, Nickname);
15. try {
16. (var res, var body) = await FetchAsync<AuthResponse>(
17. HttpMethod.Post, "register",
18. isFetch => IsFetch = isFetch,
19. error => Error = error,
20. regData,
21. serialize: true
22. );
23. if (res == null)
24. throw new Exception(Error ?? "Сервер не ответил");
25. if (res.StatusCode == HttpStatusCode.**UnprocessableEntity**) {
26. var errorJson = await res.Content.ReadAsStringAsync();
27. Debug.WriteLine("ERRJSON: " + errorJson);
28. BadFields = JsonSerializer.Deserialize<ErrorResponse>(errorJson)?.Errors ?? [];
29. throw new Exception("Ошибка валидации данных");
30. }
31. if (!res.IsSuccessStatusCode)
32. throw new Exception(Error ?? $"Пришёл плохой код {(int)res.StatusCode}");
33. if (Error != null)
34. throw new Exception(Error);
35. if (body?.Token == null || body?.User == null)
36. throw new Exception("Ошибка сервера: не пришли нужные данные\n"
37. + await res.Content.ReadAsStringAsync());
38. Password = "";
39. PasswordConfirm = "";
40. AuthData.SaveAndNavigate(body.Token, body.User);
41. }
42. catch (Exception ex) {
43. Error = ex.Message;
44. }
45. }
46. }

### Главная страница

Разметка главной страницы (MainPage) включает в себя список синхронизируемых фотографий, блок с кнопками управления синхронизацией и обновлением, а также сообщения об ошибках. Если альбомы для синхронизации отсутствуют, отображается заглушка с кнопкой перехода к выбору альбомов (см. рис. 54). В противном случае показывается коллекция изображений, сгруппированных по месяцам и годам (см.рис. 55). Дизайн включает кнопки для синхронизации и обновления данных, а также индикаторы состояния загрузки.

|  |  |
| --- | --- |
| Рисунок 54 – Вид главной страницы при неимении синхронизированных альбомов | Рисунок 55 – Вид главной страницы с имением синхронизированных альбомов |

Логика главной страницы реализована в MainPageViewModel, который управляет списком синхронизированных альбомов, обработкой команд обновления (RefreshCommand), выбора альбомов (GoToAlbumsCommand) и управления синхронизацией (SyncManageCommand). Он обрабатывает разрешения на доступ к локальным файлам, запрашивает данные о фотографиях и группирует их для удобного отображения. Также в модели реализован механизм динамического вычисления количества колонок в коллекции изображений в зависимости от ширины экрана. Код логики представлен в листинге 55.

Листинг 55 – Часть логики главной страницы

1. public partial class MainPageViewModel : ObservableObject
2. {
3. [RelayCommand]
4. public async Task Refresh() {
5. IsRefreshing = true;
6. CanLoadMore = false;
7. await RequestLocalAlbums();
8. HasSynced = AlbumsSynced.Any();
9. if (HasSynced) {
10. PicturesCursors = null;
11. CanLoadMore = true;
12. }
13. IsRefreshing = false;
14. }
15. [RelayCommand]
16. public async Task SyncManage() {
17. GeneralSyncManagePopup popup = new();
18. await Shell.Current.CurrentPage.ShowPopupAsync(popup);
19. }
20. [ObservableProperty]
21. private bool hasPermissions = false;
22. [RelayCommand]
23. public async Task GoToAlbums() {
24. try {
25. HasPermissions = await LocalData.RequestPermissions();
26. if (HasPermissions)
27. \_ = Shell.Current.GoToAsync("//Albums");
28. else
29. \_ = Shell.Current.DisplayAlert("Ошибка", "Вы должны дать разрешение", "OK");
30. }
31. catch (PlatformNotSupportedException ex) {
32. \_ = Shell.Current.DisplayAlert("Ошибка", ex.Message, "OK");
33. }
34. }
35. public async Task RequestLocalAlbums() {
36. *// Если не разрешено или не поддерживается — произойдёт исключение*
37. try {
38. HasPermissions = await LocalData.CheckPermissions();
39. if (!HasPermissions)
40. return;
41. }
42. catch (PlatformNotSupportedException) {
43. return;
44. }
45. if (LocalData.Status == LocalLoadStatus.**NotLoad**){
46. await LocalData.FillPictures();
47. }
48. else if (LocalData.Status == LocalLoadStatus.**InLoad**) {
49. while (LocalData.Status == LocalLoadStatus.**InLoad**) {
50. await Task.Delay(1000);
51. }
52. }
53. else if (LocalData.Status == LocalLoadStatus.**Loaded**) {
54. await LocalData.FillPictures();
55. }
56. AlbumsSynced = new(LocalData.Albums.OfType<AlbumSynced>().ToList());
57. PicturesFlat = AlbumsSynced
58. .SelectMany(album => album.LocalPictures)
59. .OrderByDescending(picture => picture.Date)
60. .ToList();
61. PicturesGroups = new(PicturesFlat
62. .GroupBy(picture => new DateOnly(
63. picture.Date.Year,
64. picture.Date.Month,
65. 1
66. )).ToList()
67. );
68. }
69. }

### Страница просмотрщика картинки

Разметка страницы просмотрщика изображения (ViewerPage) предназначена для отображения выбранной фотографии с возможностью её увеличения, перелистывания и просмотра информации. В верхней части экрана отображается заголовок с названием изображения, его порядковым номером, размерами и размером файла (см. рис. 56). Также присутствуют индикаторы состояния синхронизации, такие как загруженность в облако или статус локального хранения. Основная область содержит изображение с поддержкой масштабирования жестами (Pinch-to-Zoom) и скрывающийся при касании интерфейс. По бокам размещены кнопки для перехода к предыдущему и следующему файлу.

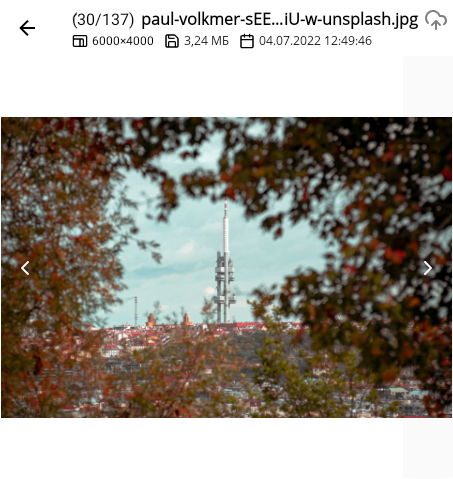


Рисунок 56 – Вид страницы просмотрщика картинки

Логика страницы реализована в ViewerViewModel и отвечает за загрузку изображения, управление отображением управляющих элементов и перелистывание фотографий. В модели хранятся текущая позиция изображения, его общая численность в альбоме, а также информация о файле. Команды MoveNextCommand и MovePreviousCommand позволяют перемещаться между изображениями, а ToggleControlsVisibilityCommand скрывает или показывает интерфейс, в зависимости от пользовательского ввода. Также предусмотрена обработка платформенных особенностей, например, скрытие статус-бара на Android при полноэкранном просмотре. Код логики страницы представлен в листинге 56.

Листинг 56 –Часть логика страницы просмотрщика картинки

1. public partial class ViewerViewModel : ObservableObject {
2. public ViewerViewModel(Models.Pictures.IPicture picture, AlbumViewModel? albumViewModel =
3. ull) {
4. AlbumViewModel = albumViewModel;
5. Picture = picture;
6. if (AlbumViewModel == null || AlbumViewModel.PicturesGroups.Count < 1) return;
7. Total = AlbumViewModel.Album.PicturesCount;
8. Position = 0;
9. foreach (var group in AlbumViewModel.PicturesGroups) {
10. foreach (var pictureInGroup in group) {
11. if (pictureInGroup == Picture || pictureInGroup.Equals(Picture)) return;
12. Position++;
13. }
14. }
15. }
16. [ObservableProperty]
17. private bool \_areControlsVisible = true;
18. [RelayCommand]
19. private void ToggleControlsVisibility() {
20. AreControlsVisible = !AreControlsVisible;
21. #if ANDROID
22. *// Платформо-специфичное управление статус-баром и навигационной панелью*
23. var activity = Platform.CurrentActivity;
24. System.Diagnostics.Debug.WriteLine($"activity: {activity}");
25. if (activity == null) return;
26. if (AreControlsVisible) {
27. if (activity.Window == null) return;
29. *// Показать статус-бар и навигационную панель*
30. activity.Window.DecorView.SystemUiVisibility = StatusBarVisibility.**Visible**;
31. activity.Window.DecorView.SystemUiVisibility = SystemUiFlags.**Visible**;
32. activity.Window.ClearFlags(WindowManagerFlags.**Fullscreen**);
33. activity.Window?.AddFlags(WindowManagerFlags.**ForceNotFullscreen**);
34. activity.Window?.ClearFlags(WindowManagerFlags.**Fullscreen**);
35. var controller = activity.Window?.InsetsController;
36. controller?.Show(WindowInsets.Type.SystemBars());
37. }
38. else {
39. if (activity.Window == null) return;
40. activity.Window.AddFlags(WindowManagerFlags.**Fullscreen**);
41. activity.Window.ClearFlags(WindowManagerFlags.**ForceNotFullscreen**);
42. var controller = activity.Window.InsetsController;
43. controller?.Hide(WindowInsets.Type.SystemBars());
44. activity.Window.AddFlags(WindowManagerFlags.**Fullscreen**);
45. }
46. }
47. [RelayCommand]
48. private async Task OpenInfo() {
49. PictureInfoPopup popup = new(Picture);
50. var result = await Shell.Current.CurrentPage.ShowPopupAsync(popup);
51. if (result is bool and true)
52. {
53. if (Picture is PictureRemote remote)
54. RemoteAlbumsData.AlbumsAccessible.Remove(remote.SpecificAlbum);
55. \_ = Shell.Current.GoToAsync("//Albums");
56. }
57. OnPropertyChanged(nameof(Picture));
58. }
59. [RelayCommand(CanExecute = nameof(CanMoveNext))]
60. private async Task MoveNext() {
61. if (AlbumViewModel == null || Position == null || Total == null)
62. return;
63. IsBusy = true;
64. Position++;
65. if (Position >= AlbumViewModel.PicturesGroups.Sum(g => g.Count)) {
66. await AlbumViewModel.LoadMoreCommand.ExecuteAsync(null);
67. }
68. UpdateCurrentPicture();
69. IsBusy = false;
70. }
71. }

### Страница альбомов

Разметка страницы альбомов (AlbumsPage) включает несколько коллекций: локальные, синхронизируемые и удалённые альбомы. В зависимости от наличия данных отображаются соответствующие секции, каждая из которых имеет кнопку обновления и индикатор загрузки. Синхронизируемые альбомы (AlbumsSynced) представлены в виде коллекции изображений, где каждый альбом отображает миниатюры своих фотографий, название и дополнительные сведения (см. рис. 57). Локальные и удалённые альбомы организованы аналогично, но скрываются при отсутствии данных. Если у пользователя нет разрешений на доступ к локальным файлам, отображается кнопка запроса разрешений. В случае ошибки появляется блок с сообщением об ошибке и иконкой предупреждения.

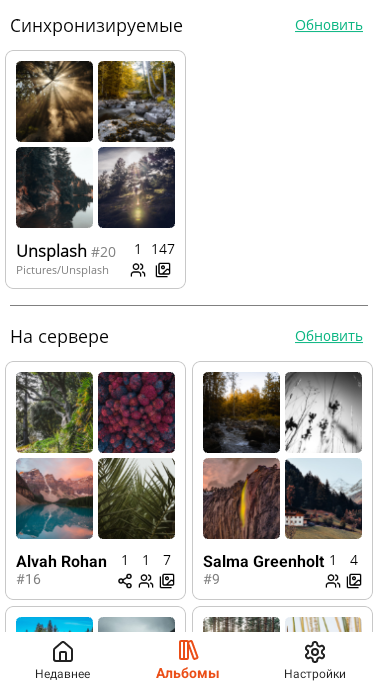


Рисунок 57 – Вид страницы альбомов

Логика управления страницей реализована в модели представления AlbumsViewModel. Она управляет коллекциями альбомов, выполняет команды обновления данных (RequestAlbumsCommand, RequestRemoteAlbumsCommand, RequestLocalAlbumsCommand) и обрабатывает разрешения на доступ к файлам. При обновлении синхронизируемых альбомов модель проверяет их актуальность на сервере и обновляет локальную базу данных. Логика страницы представлена в листинге 57.

Листинг 57 – Часть логики страницы альбомов

1. public partial class AlbumsViewModel : ObservableObject {
2. public AlbumsViewModel() {
3. AlbumsSynced.CollectionChanged += OnAlbumsSyncedCollectionChanged;
4. RequestAlbumsCommand.PropertyChanged += (s, e) => {
5. if (e.PropertyName == nameof(AsyncRelayCommand.IsRunning))
6. OnPropertyChanged(nameof(CanRequestAlbums));
7. };
8. RequestAlbumsCommand.Execute(null);
9. }
10. public bool SyncedIsVisible => AlbumsSynced.Count > 0;
11. private void OnAlbumsSyncedCollectionChanged(object? sender,
12. ystem.Collections.Specialized.NotifyCollectionChangedEventArgs e)
13. => OnPropertyChanged(nameof(SyncedIsVisible));
14. partial void OnAlbumsSyncedChanged(ObservableCollection<AlbumSynced>? oldValue,
15. bservableCollection<AlbumSynced> newValue) {
16. if (oldValue != null)
17. oldValue.CollectionChanged -= OnAlbumsSyncedCollectionChanged;
18. newValue.CollectionChanged += OnAlbumsSyncedCollectionChanged;
19. OnPropertyChanged(nameof(SyncedIsVisible));
20. }
21. public bool CanRequestAlbums => !(RequestAlbumsCommand.IsRunning || IsFetch);
22. [RelayCommand(CanExecute = nameof(CanRequestAlbums), IncludeCancelCommand = true)]
23. private async Task<bool> RequestAlbums(CancellationToken token = default) {
24. var remoteTask = RequestRemoteAlbumsCommand.ExecuteAsync(token);
25. var localTask = RequestLocalAlbumsCommand.ExecuteAsync(null);
26. await Task.WhenAll(localTask, remoteTask);
27. if (!HasPermissions || Error != null) return false;
28. List<AlbumSynced> syncedAlbumsFromLocal = new(AlbumsSynced);
29. AlbumsSynced.Clear();
30. *// Проверка, что синхронизирующиеся альбомы до сих пор на сервере*
31. for (int i = AlbumsRemote.Count - 1; i >= 0; i--) {
32. var remote = AlbumsRemote[i];
33. var synced = syncedAlbumsFromLocal.FirstOrDefault(a => a.Id == remote.Id);
34. if (synced is not { }) continue;
35. Debug.WriteLine($"RequestAlbums: synced.simple: #{synced.Id} {synced.LocalPath}");
36. synced.Update(remote);
37. DB.Update(synced);
38. AlbumsRemote.RemoveAt(i);
39. AlbumsSynced.Insert(0, synced);
40. syncedAlbumsFromLocal.Remove(synced);
41. }
42. *// Оставшиеся в syncedAlbumsFromLocal более не синхронизируются*
43. foreach (var nonSynced in syncedAlbumsFromLocal) {
44. Debug.WriteLine("RequestAlbums: foreach: \n" + JsonSerializer.Serialize(nonSynced));
45. AlbumLocal localAlbum = new(nonSynced);
46. AlbumsLocal.Add(localAlbum);
47. LocalData.Albums.ReplaceOrAdd(nonSynced, localAlbum);
48. DB.Delete(nonSynced);
49. }
50. return true;
51. }
52. }

### Страница конкретного альбома

Разметка страницы конкретного альбома (ContentPage) включает в себя список изображений, сгруппированных по категориям, с возможностью обновления данных с помощью кнопки синхронизации. Страница имеет заголовок с горизонтальной прокруткой, в котором отображаются кнопки управления синхронизацией и доступами (см. рис. 59 и рис. 60). В управлении доступами можно создавать приглашения (см. рис. 61) и отзывать доступы у кого кто присоединился. Изображения группируются по месяцам. Отображаются изображения в виде плитки с возможностью открытия в просмотрщике). Итоговый вид изображён на рисунке 58.

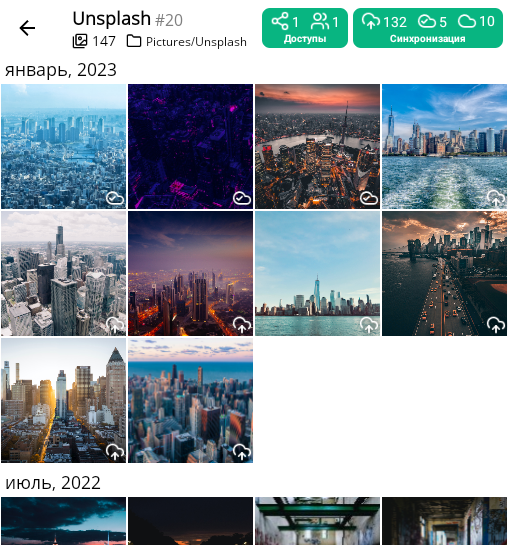


Рисунок 58 – Вид страницы синхронизирующего альбома

Логика страницы конкретного альбома реализована в AlbumViewModel, который управляет состоянием альбома, загрузкой данных и синхронизацией фотографий. Он содержит свойства, такие как PicturesGroups для хранения сгруппированных изображений и методы для обновления данных (например, Refresh() и LoadInfo()). Механизм загрузки дополнительных изображений реализован через команду LoadMore(), которая учитывает локальные и удалённые фотографии, а также их синхронизацию. Для синхронизации и управления доступами используются методы Sync(), SyncManage(), AccessManage(), каждый из которых открывает соответствующие всплывающие окна для взаимодействия с пользователем. Логика выполнения команд зависит от состояния альбома и реализована через проверки на тип альбома (локальный, синхронизированный, удалённый). Код логики представлен в листинге 58.

|  |  |
| --- | --- |
| Рисунок 59 – Вид всплывающего окна синхронизации | Рисунок 60 – Вид всплывающего окна управления доступами    Рисунок 61 – Вид всплывающего окна создания приглашения |

Листинг 58 – Часть логики страницы конкретного альбома

1. public partial class AlbumViewModel : ObservableObject {
2. [RelayCommand(CanExecute = nameof(CanLoadMore))]
3. private async Task LoadMore() {
4. if (!CanLoadMore) return;
5. IsBusy = true;
6. try {
7. List<Models.Pictures.IPicture> piece;
8. if (Album is AlbumSynced albumSynced) {
9. var syncedPictures = albumSynced.LocalPictures
10. .OfType<PictureSynced>();
11. piece = new(PageSize);
12. while (piece.Count < PageSize) {
13. *// Элементы выбора*
14. IPictureLocal? localPicture = null;
15. if (LocalOffset < albumSynced.LocalPictures.Count)
16. localPicture = albumSynced.LocalPictures[LocalOffset];
17. PictureRemote? remotePicture = null;
18. if (LastRemotePicturesPage == null || RemoteOffset < AlbumSynced.RemotePicturesCount) {
19. int requiredPage = RemoteOffset / PageSize + 1;
20. *// Запрашиваем удалённые если надо*
21. if (LastRemotePicturesPage == null || LastRemotePicturesPage.Page != RequiredPage) {
22. var (res, picturesPage) = await FetchAsync<PicturesPageResponse>(
23. HttpMethod.Get,
24. URLs.AlbumPictures(albumSynced.Id, new()
25. {
26. Page = requiredPage,
27. Limit = PageSize,
28. Sort = PicturesSort.**date**,
29. IsReverse = true,
30. }),
31. setError: e => ErrorOnPage = e
32. );
33. if (picturesPage != null)
34. {
35. albumSynced.RemotePicturesCount = picturesPage.Total;
36. if (picturesPage.Signature != null)
37. {
38. if (albumSynced.Preview == null)
39. albumSynced.Preview = new() { Signature = PicturesPage.Signature };
40. else
41. albumSynced.Preview.Signature = picturesPage.Signature;
42. }
43. foreach (var picture in picturesPage.Pictures)
44. picture.Album = Album;
45. LastRemotePicturesPage = picturesPage;
46. }
47. Debug.WriteLine("return from server: " +
48. sonSerializer.Serialize(picturesPage));
49. }
50. if (RemoteOffset < albumSynced.RemotePicturesCount)
51. remotePicture = LastRemotePicturesPage?.Pictures[RemoteOffset –
52. ageSize \* (RemoteOffset / PageSize)];
53. }
54. *// Проверяем, существует ли локальная копия удалённой картинки*
55. if (remotePicture != null) {
56. var syncedPicture = syncedPictures.FirstOrDefault(l => l.Id == RemotePicture.Id);
57. if (syncedPicture != null) {
58. Debug.WriteLine("found synced of remote picture: " + JsonSerializer.Serialize(syncedPicture));
59. syncedPicture.Update(remotePicture);
60. DB.Update(syncedPicture);
61. remotePicture = null;
62. RemoteOffset++;
63. }
64. }
65. }
66. }
67. else if (Album is AlbumLocal albumLocal){
68. piece = albumLocal.LocalPictures
69. .Skip(LocalOffset)
70. .Take(PageSize)
71. .Cast<Models.Pictures.IPicture>()
72. .ToList();
73. LocalOffset += PageSize;
74. }
75. else if (Album is AlbumRemote albumRemote) {
76. var (res, picturesPage) = await FetchAsync<PicturesPageResponse>(
77. HttpMethod.Get,
78. URLs.AlbumPictures(albumRemote.Id, new() {
79. Page = RemoteOffset / PageSize + 1,
80. Limit = PageSize,
81. Sort = PicturesSort.**date**,
82. IsReverse = true,
83. }),
84. setError: e => ErrorOnPage = e
85. );
86. if (picturesPage == null) {
87. IsBusy = false;
88. return;
89. }
90. albumRemote.RemotePicturesCount = picturesPage.Total;
91. if (picturesPage.Signature != null) {
92. if (albumRemote.Preview == null)
93. albumRemote.Preview = new() { Signature = picturesPage.Signature };
94. Else
95. albumRemote.Preview.Signature = picturesPage.Signature;
96. }
97. piece = new(picturesPage.Pictures.Count);
98. foreach (var picture in picturesPage.Pictures) {
99. picture.Album = Album;
100. piece.Add(picture);
101. }
102. RemoteOffset += PageSize;
103. }
104. }
105. }
106. }

### Страница настроек

Разметка страницы настроек (SettingsPage) включает в себя несколько ключевых блоков: информацию о пользователе, статистику, данные о сервере и квоте, а также возможность выхода из системы. В персональном блоке содержится информация о никнейме, ИД и роли; по нажатию открывается окно редактирования данных пользователя (см. рис. 63). В статистике указывается количество картинок, альбомов, жалоб и предупреждений. В блоке о сервере можно при нажатии копировать URL. Итоговый вид изображён на рисунке 62.

|  |  |
| --- | --- |
| Рисунок 62 – Вид страницы настроек | Рисунок 63 – Вид всплывающего окна изменения себя |

Логика страницы настроек реализована в SettingsViewModel, который управляет состоянием страницы и обработкой команд. Он включает команды для обновления данных пользователя и сервера (Refresh), копирования URL в буфер обмена (CopyUrl), редактирования данных пользователя (EditSelf), а также для выхода из системы (TryExit). Модель отслеживает изменения в данных пользователя, статистики и настроек сервера, обновляя соответствующие свойства через биндинг. Также реализована поддержка асинхронных операций с использованием Task.WhenAll для параллельного обновления данных. Логика страницы представлена в листинге 59.

Листинг 59 – Логика страницы настроек

1. public partial class SettingsViewModel : ObservableObject
2. {
3. [RelayCommand(IncludeCancelCommand = true)]
4. private async Task Refresh(CancellationToken token = default) {
5. var userTask = AuthData.Update(e => StatsGetError = e, token);
6. var serverTask = ServerData.Update(e => SettingsGetError = e, token);
7. await Task.WhenAll(userTask, serverTask);
8. }
9. public void SoftUpdate() {
10. OnPropertyChanged(nameof(User));
11. OnPropertyChanged(nameof(Stats));
12. OnPropertyChanged(nameof(Url));
13. OnPropertyChanged(nameof(ServerSettings));
14. }
15. [RelayCommand]
16. private async Task CopyUrl() {
17. if (Url is not { } uri) return;
18. await Clipboard.SetTextAsync(uri.ToString());
19. await Toast.Make("Скопировано в буфер").Show();
20. }
21. [RelayCommand]
22. private async Task EditSelf() {
23. EditSelfPopup popup = new();
24. await Shell.Current.CurrentPage.ShowPopupAsync(popup);
25. OnPropertyChanged(nameof(User));
26. }
27. private bool CanExit() => !IsFetch;
28. [RelayCommand(CanExecute = nameof(CanExit))]
29. private async Task TryExit() {
30. await AuthData.TryExitAndNavigate(isFetch => IsFetch = isFetch);
31. }
32. }

# Глава 6. Тестирование API

В ходе разработки системы было проведено автоматизированное тестирование API с использованием Postman. Для проверки корректности работы маршрутов были созданы тестовые сценарии, охватывающие основные аспекты взаимодействия с сервером.

Тестирование включало в себя:

* Проверку корректности обработки запросов (GET, PATCH, POST, DELETE).
* Оценку соответствия возвращаемых данных ожидаемому формату.
* Тестирование обработки ошибок и валидации данных.
* Измерение времени отклика API.

Разработанные тесты позволяют выявить и устранить возможные ошибки на ранних этапах разработки, обеспечив стабильность и предсказуемость работы сервиса.

## 401 Unauthorized

Этот тест проверяет аутентификацию пользователя, он отправляет запрос без авторизации на ресурс, либо некорректный Bearer Token, ожидая получить ответ с кодом состояния 401, а также наличие ключа «message» и временем отклика до 500мс. Код теста представлен в листинге 60.

Листинг 60 – Тест 401 Unauthorized

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(401);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });
7. pm.**test**("Проверка наличия ключа 'message'", **function** () {
8. pm.expect(pm.response.json()).to.have.property("message");
9. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| POST | {{api}}/users/logout |
| POST | {{api}}/albums |
| GET | {{api}}/albums |
| GET | {{api}}/albums/{{idalbumtest}} |
| PATCH | {{api}}/albums/{{idalbumtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| DELETE | {{api}}/invitation/{{codeinvitationtest}} |
| POST | {{api}}/invitation/{{codeinvitationtest}}/join |
| GET | {{api}}/invitation/{{codeinvitationtest}}/album |
| DELETE | {{api}}/albums/{{idalbumtest}}/access/{{idusertest}} |
| POST | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| PATCH | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| POST | {{api}}/tags |
| GET | {{api}}/tags |
| PATCH | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/tags/{{idtagtest}} |
| DELETE | {{api}}/tags/{{idtagtest}} |
| PATCH | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |
| GET | {{api}}/users/me |
| POST | {{api}}/users/me |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| POST | {{api}}/users/{{idusertest}} |
| GET | {{api}}/setups |

## 422 Unprocessable Content

Этот тест проверяет корректность валидации вводимых данных пользователем. Он отправляет запрос с указанными ключами, но пустыми значениями в полях. Ожидается, что сервер вернет ответ с кодом состояния 422, а также с полем «message», объясняющим ошибку, и массивом «errors», содержащим дополнительные ключи, соответствующие полям, в которых были обнаружены ошибки. Время отклика должно быть не более 500 мс. Код теста представлен в листинге 61.

Листинг 61 – Тест 422 Unproccessable Content

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(422);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });
7. pm.**test**("Проверка наличия ключа 'message'", **function** () {
8. pm.expect(pm.response.json()).to.have.property("message");
9. });
10. pm.**test**("Проверка значений в errors", **function** () {
11. pm.expect(pm.response.json().errors).to.have.property("login");
12. pm.expect(pm.response.json().errors).to.have.property("password");
13. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| POST | {{api}}/users/reg |
| POST | {{api}}/users/login |
| POST | {{api}}/albums |
| PATCH | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| POST | {{api}}/albums/{{idalbumtest}}/medias |
| POST | {{api}}/tags |
| PATCH | {{api}}/tags/{{idtagtest}} |
| PATCH | {{api}}/users/me |
| PATCH | {{api}}/users/{{idusertest}} |

## 403 Forbidden

Этот тест проверяет правильность обработки запроса, когда пользователь не имеет достаточных прав для доступа к ресурсу. В ответе ожидается код состояния 403, который сигнализирует о запрете на выполнение операции, а также проверяется, что в ответе присутствует ключ «message», который обычно содержит описание причины ошибки или запрета. Время отклика должно быть не более 500 мс. Код теста представлен в листинге 62.

Листинг 62 – Тест 403 Forbidden

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(403);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });
7. pm.**test**("Проверка наличия ключа 'message'", **function** () {
8. pm.expect(pm.response.json()).to.have.property("message");
9. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| GET | {{api}}/albums/{{idalbumtest}} |
| PATCH | {{api}}/albums/{{idalbumtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| DELETE | {{api}}/invitation/{{codeinvitationtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/accesses/{{idusertest}} |
| POST | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/thumb/{{orientation}}{{size}} |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/orig |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/download |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| POST | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| PATCH | {{api}}/users/{{idusertest}} |
| GET | {{api}}/setups |

## 204 No Content

Этот тест проверяет, что сервер корректно обрабатывает запрос и возвращает статус 204, указывающий на успешное выполнение без содержимого в ответе. Время отклика должно быть не более 500 мс. Код теста представлен в листинге 63.

Листинг 63 – Тест 204 No Content

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(204);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });
7. pm.**test**("Проверка, что тело ответа пустое", **function** () {
8. pm.expect(pm.response.text()).to.equal("");
9. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| POST | {{api}}/users/logout |
| DELETE | {{api}}/albums/{{idalbumtest}} |
| DELETE | {{api}}/invitation/{{codeinvitationtest}} |
| POST | {{api}}/invitation/{{codeinvitationtest}}/join |
| DELETE | {{api}}/albums/{{idalbumtest}}/accesses/{{idusertest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| DELETE | {{api}}/tags/{{idtagtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/tags/{{idtagtest}} |

## 200 OK

Этот тест проверяет, что сервер корректно обрабатывает запрос и возвращает статус 200, указывающий на успешное выполнение, а также проверяемый наличие нужного поля в ответе. Время отклика должно быть не более 500 мс. Код теста представлен в листинге 64.

Листинг 64 – Тест 200 OK

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(200);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| GET | {{api}}/albums |
| GET | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}} |
| GET | {{api}}/invitation/{{codeinvitationtest}}/album |
| GET | {{api}}/accesses |
| POST | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}} |
| GET | {{api}}/albums/{{idalbumtest}}/medias/{{idpicturetest}}/thumb/{{orientation}}{{size}}?sign={{signtest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/orig?sign={{signtest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/download?sign={{signtest}} |
| GET | {{api}}/tags |
| PATCH | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/users/me |
| POST | {{api}}/users/me |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| POST | {{api}}/users/{{idusertest}} |
| GET | {{api}}/setups |

## 201 Created

Этот тест проверяет, что сервер корректно обрабатывает запрос и возвращает статус 201, указывающий на успешное создание объекта. Время отклика должно быть не более 500 мс. Код теста представлен в листинге 65.

Листинг 65 – Тест 201 Created

1. pm.**test**("Проверка кода ответа", **function** () {
2. pm.response.to.have.status(201);
3. });
4. pm.**test**("Проверка времени ответа", **function** () {
5. pm.expect(pm.response.responseTime).to.be.below(500);
6. });

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Маршрут** |
| POST | {{api}}/users/reg |
| POST | {{api}}/albums |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| POST | {{api}}/tags |

## Вывод

Проведённое тестирование подтвердило корректность работы всех маршрутов, которые проверяет работу всех систем за ним и вывело стабильную работу обработки запросов, что обеспечивает надёжное функционирование всех возложенных на систему задач.

# Заключение

В рамках реализации дипломного проекта был разработан медиа-сервис «WepicSync» — современное облачное решение для хранения, синхронизации и обмена фото- и видеоматериалами. Основной задачей проекта являлось создание удобного и безопасного инструмента, позволяющего пользователям эффективно управлять своим медиаконтентом, организовывать его в альбомы, добавлять теги и делиться файлами с другими пользователями.

Для достижения поставленных целей была спроектирована и реализована структура базы данных на основе MySQL, соответствующая требованиям третьей нормальной формы, что позволило минимизировать избыточность информации и обеспечить высокую степень согласованности хранимых данных. Серверная часть системы была реализована с использованием фреймворка Laravel, который выступает основой для предоставления данных как веб-интерфейсу, так и мобильному приложению в соответствии с заданными API-требованиями.

Клиентская часть представлена веб-интерфейсом, разработанным с использованием Vue.js и интегрированным в Laravel-приложение, что обеспечивает широкие возможности взаимодействия с медиафайлами через браузер. Также была создана мобильная версия сервиса на платформе MAUI, которая позволяет пользователям легко загружать, синхронизировать и просматривать медиафайлы с мобильных устройств, освобождая их от необходимости хранения тяжёлых файлов локально.

Особое внимание было уделено качеству и надёжности API: были разработаны автоматизированные тесты, подтвердившие корректную работу всех маршрутов и соответствие их требованиям. В результате создано устойчивое и отказоустойчивое приложение, готовое к практическому применению.

Сервис «WepicSync» решает актуальную проблему ограниченного объема памяти на устройствах пользователей, предлагая вместо этого надежное облачное хранилище. Благодаря таким возможностям, как общие альбомы, систему тегов и гибкие настройки доступа, сервис может использоваться как для личных нужд, так и в командной работе.

Таким образом, все задачи, поставленные в начале проекта, успешно выполнены, достигнута цель создания комплексного облачного медиа-сервиса, который полностью соответствует современным требованиям к подобным системам и готов к дальнейшему развитию и внедрению в реальную эксплуатацию.

# Список использованных источников

1. Учебные курсы и справочники по HTML и CSS [Электронный ресурс]. — URL: https://webref.ru/ (дата обращения 11.04.2025).
2. Современный учебник JavaScript [Электронный ресурс]. — URL: https://learn.javascript.ru/ (дата обращения 20.04.2025.
3. Официальная документация Laragon [Электронный ресурс]. — URL: https://laragon.org/docs/ (дата обращения 05.06.2025).
4. Официальная документация Vue 3 [Электронный ресурс]. — URL: https://v3.ru.vuejs.org/ (дата обращения 05.06.2025).
5. Руководство по .NET MAUI [Электронный ресурс]. — URL: https://metanit.com/sharp/maui/ (дата обращения 05.06.2025).
6. Документация Laravel [Электронный ресурс]. 12 — URL: https://laravel.su/docs/12.x/documentation/ (дата обращения 05.06.2025).
7. Обучающий центр Postman [Электронный ресурс]. — URL: https://learning.postman.com/ (дата обращения 05.06.2025).
8. Официальная документация Pinia: менеджер глобальных состояний для Vue.js [Электронный ресурс]. — URL: https://pinia.vuejs.org/core-concepts/ (дата обращения 05.06.2025).
9. Официальная документация ffmpeg: фреймворк для кодирования, декодирования видео и аудио [Электронный ресурс]. — URL: https://ffmpeg.org/ffmpeg.html (дата обращения 05.06.2025).
10. Официальная документация Intervention Image 3: обработка картинок в PHP [Электронный ресурс]. — URL: https://image.intervention.io/v3 (дата обращения 05.06.2025).
11. Официальная документация микроразметки Open Graph [Электронный ресурс]. — URL: https://ogp.me/ (дата обращения 05.06.2025).