**ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА  
СЕРВИСА СИНХРОНИЗАЦИИ КАРТИНОК «PICSYNC»**

***КУРСОВОЙ ПРОЕКТ***

*СПЕЦИАЛЬНОСТЬ 09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ*

Выполнил:

Студент группы 1912с

\_\_\_\_\_\_\_\_\_\_\_ / Заикин В.А

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

Проверил:

Преподаватель

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

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

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Оглавление

[Введение 4](#_Toc190557257)

[Информация о проекте 4](#_Toc190557258)

[Назначение документа 4](#_Toc190557259)

[Актуальность 4](#_Toc190557260)

[Глоссарий 5](#_Toc190557261)

[Глава 1. Техническое задание 6](#_Toc190557262)

[Общее описание 6](#_Toc190557263)

[Требования к функциональности 7](#_Toc190557264)

[Ролевая модель 9](#_Toc190557265)

[Диаграмма прецедентов 10](#_Toc190557266)

[Диаграммы последовательностей 12](#_Toc190557267)

[Технологический стек 23](#_Toc190557268)

[Хранимые данные 23](#_Toc190557269)

[Требования к REST API 24](#_Toc190557270)

[Требования к мобильному приложению клиента 37](#_Toc190557271)

[Требования к мобильному приложению администратора 40](#_Toc190557272)

[Глава 2. Проектирование и разработка базы данных 43](#_Toc190557273)

[Концептуальная модель данных 43](#_Toc190557274)

[Логическая модель данных 44](#_Toc190557275)

[Физическая модель данных 45](#_Toc190557276)

[Глава 3. Разработка серверной части 47](#_Toc190557277)

[Маршрутизация 47](#_Toc190557278)

[Миграции 49](#_Toc190557279)

[Заполнители 51](#_Toc190557280)

[Конфиги 54](#_Toc190557281)

[Модели 54](#_Toc190557282)

[Контроллеры 61](#_Toc190557283)

[Посредники 81](#_Toc190557284)

[Глава 4. Автотестирование API 85](#_Toc190557285)

[401 Unauthorized 85](#_Toc190557286)

[422 Unprocessable Content 86](#_Toc190557287)

[403 Forbidden 87](#_Toc190557288)

[204 No Content 88](#_Toc190557289)

[200 OK 89](#_Toc190557290)

[201 Created 90](#_Toc190557291)

[200 Admin 91](#_Toc190557292)

[200 User 92](#_Toc190557293)

[Глава 5. Разработка мобильного приложения администратора 93](#_Toc190557294)

[Введение 93](#_Toc190557295)

[Структура проекта 93](#_Toc190557296)

[Навигация 94](#_Toc190557297)

[Компоненты 95](#_Toc190557298)

[Конвертеры 99](#_Toc190557299)

[Помощники 101](#_Toc190557300)

[Модели 108](#_Toc190557301)

[Реализация страниц 109](#_Toc190557302)

[Заключение 140](#_Toc190557303)

[Список используемых источников 141](#_Toc190557304)

# Введение

## Информация о проекте

|  |  |
| --- | --- |
| **Название** | Сервис синхронизации картинок «PicSync» |
| **Цель** | Спроектировать и разработать сервис синхронизации картинок «PicSync» |
| **Задачи** | 1. Разработать базу данных сервиса на MySQL. 2. Спроектировать структуру API сервиса. 3. Разработать сервер API сервиса на фреймворке Laravel. 4. Разработать мобильное приложение на MAUI для клиента. 5. Разработать мобильное приложение на MAUI для администратора. 6. Разработать автоматизированные тесты и провести тестирование. |

## Назначение документа

Настоящий документ рассчитан на использование в проектной команде в качестве ключевого документа, определяющего требование по реализации задачи по разработке сервиса синхронизации картинок «PicSync».

## Актуальность

С ростом популярности смартфонов и увеличения качества съемки (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 удаленном 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.

## Глоссарий

|  |  |
| --- | --- |
| **Понятие** | **Определение понятия** |
| API | (интерфейс программирования приложений) – 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 | Инструмент для тестирования и разработки API. Он предоставляет удобный интерфейс для отправки HTTP запросов к API, а также для анализа ответов. |
| Draw.io | Бесплатный онлайн и оффлайн 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 | Акроним, обозначающий четыре базовые функции, используемые при работе с данными: создание, чтение, модификация, удаление. |
| Laravel | Бесплатный, открытый фреймворк для создания веб-приложений на языке программирования PHP. Он предоставляет структуру и базовые инструменты для быстрого создания веб-приложений, используя принципы модели-представления-контроллер (MVC) и другие современные подходы к разработке. |
| MIME | (Multipurpose Internet Mail Extensions) — стандарт, описывающий передачу различных типов данных через интернет, таких как текст, изображения, аудио, видео и другие файлы, в электронной почте и других протоколах. |
| CRON | Планировщик задач Unix-подобных ОС, который позволяет автоматически запускать программы или скрипты в заданное время или через определённые интервалы. |
| CORS | Cross-origin resource sharing — технология современных браузеров, которая позволяет предоставить веб-страницам доступ к ресурсам другого домена. |

# Глава 1. Техническое задание

## Общее описание

Разрабатываемый сервис предназначен для удобного хранения, управления и получения пользователями своих фотографий и картинок с удаленного сервера. Основная цель сервиса – освободить память на мобильных устройствах пользователей, предлагая им безопасное и надежное облачное хранилище для их медиафайлов. Пользователи смогут загружать фотографии и картинки с телефона на сервер через удобный и интуитивно понятный интерфейс, организовывать их в альбомы, добавлять описания и теги для упрощения поиска. Сервис также предоставит возможность делиться фотографиями с другими пользователями посредством приглашения через ссылки. Важной особенностью сервиса будет возможность мгновенного получения фотографий и картинок с сервера на телефон по запросу пользователя, что позволит им всегда иметь доступ к своим медиафайлам, независимо от места нахождения.

Функционал сервиса разделен на две основные категории:

* **Функционал администратора** включает в себя настройку параметров сервера, работу с жалобами, а также выдачей предупреждений, удаление контента. Более подробно функционал администратора представлен на Рисунок 1.
* **Функционал пользователя** охватывает загрузку, организацию, обмен картинками. Пользователи могут создавать альбомы, добавлять теги, а также делиться своими фотографиями с другими. Детали функционала пользователя показаны на Рисунок 2.

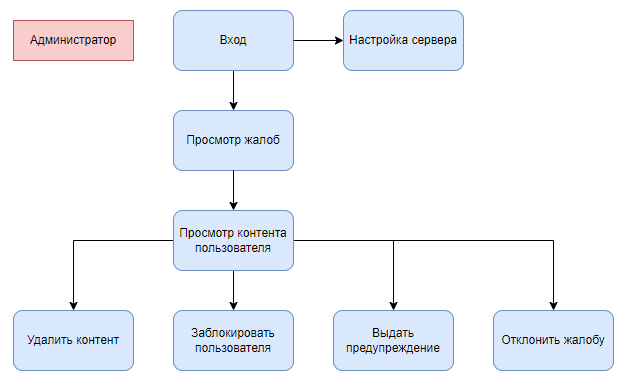


Рисунок 1 – Функционал администратора

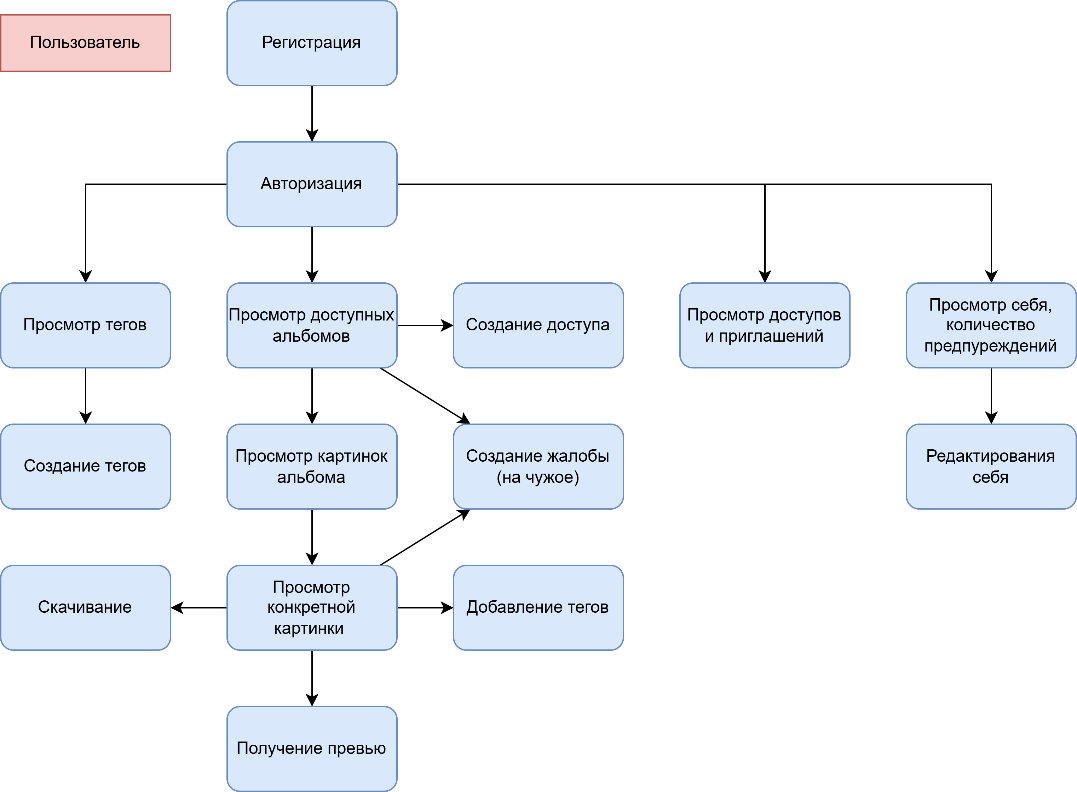


Рисунок 2 – Функционал пользователя

## Требования к функциональности

### Гости

1. Аутентификация и регистрация.
   * Авторизация производится по токену, который генерируется при входе и отзывается при выходе.
   * Вход на другом устройстве не должна вызывать выход из аккаунта на всех устройствах одного пользователя.
2. Публичная информация о сервере:
   * Разрешённые типы файлов (MIME).
   * Список размеров превью.
   * Лимит предупреждений до блокировки.
   * Квота на хранилище для каждого пользователя.
   * Статус разрешения загрузки (меняется при достижении лимита места на диске).
   * Список типов жалоб.

### Пользователи

1. Уметь всё что гость.
2. Управление своими альбомами и картинками:
   * Создать альбом, загружать в него картинки и просматривать их.
   * Установлена максимальная квота (для обычных пользователей) на размер всех загруженных пользователем картинок (регулируется настройками).
   * Удаление и изменение картинок своего альбома, а также и самого альбома.
3. Управление своими тегами:
   * Создавать лично себе теги, изменять и удалять только свои теги
   * Привязывать и отвязывать множество своих тегов на множестве своих картинках
4. Доступ к альбомам:
   * Создать код приглашения на свой альбом с опциональным выставлением лимита вступлений и даты просрочки / временем «жизни» кода.
   * Получить доступ к чужому альбому, перейдя по коду (ссылке) приглашения на доступ от автора альбома.
   * Отозвать выданные доступы и коды приглашения, если он владелец альбома.
   * Отказаться от установленного доступа (отписаться).
5. Жалобы:
   * Пожаловаться на доступный альбом чужой или картинку в нём.
   * Пользователь, который пожаловался, может:
   * Просмотреть список своих созданных жалоб.
   * Отклонить жалобу.
   * Просмотреть статус жалобы (не рассмотрена, исполнена или отклонена).

### Администраторы

1. Уметь всё что пользователь
2. Управление жалобами:
   * Администратор может просматривать все жалобы и содержимое, на которое создана жалоба.
   * Администратор может с чужими жалобами:
   * Отклонить и опционально выдать предупреждение тому, кто пожаловался, за ложную жалобу.
   * Исполнить путём:
     + Выдачи предупреждения владельцу контента.
     + Удаления материала.
     + Блокировки владельца контента.
3. Изменение настроек:
   * Разрешённые типы файлов (MIME).
   * Список размеров превью.
   * Лимит предупреждений до блокировки.
   * Квота на хранилище для каждого пользователя, не имеющих административные права.
   * Список типов жалоб.
   * Лимит занятого места на диске (в процентах), при котором загрузка на сервер будет отключена.

### Планировщик задач

1. Проверка диска:
   * Каждые 10 минут проверяет, кеширует размер диска и размер свободного места.
   * Обновлять кэшированный ответ публичной информации о сервере при достижении заданного настройками лимита занятого места на диске.

## Ролевая модель

Администратор — отвечает за управление системой, настройки, обработку жалоб и контроль пользователей.

Клиент — основной авторизированный пользователь сервиса, который загружает, организует и делится картинками.

Гость — неавторизированный пользователь, которому доступна только регистрация, вход и публичная информация.

Более структурированная и подробная информация представлена в Таблица 1.

Таблица 1 – Ролевая модель системы

| **Функционал** | | **Администратор** | **Клиент** | **Гость** |
| --- | --- | --- | --- | --- |
| Пользователи | Регистрация | да | | |
| Вход | да | | |
| Выход | да | | нет |
| Просмотр | да | да (себя) |
| Изменение |
| Альбомы | Создание | да (себе) | | нет |
| Просмотр | да (свои; доступные; по чужой жалобе) | да (свои; доступные) |
| Изменение | да (свои;  по чужой жалобе) | да (свои) |
| Удаление |
| Приглашения | Вступление в альбом | да (на чужие альбомы) | | нет |
| Создание | да (на свои альбомы) | |
| Удаление |
| Картинки | Создание | да (в своих альбомах) | | нет |
| Просмотр | да (свои; доступные;  по чужой жалобе) | да (свои; доступные) |
| Изменение | да (свои;  по чужой жалобе) | да (свои) |
| Удаление |
| Жалобы | Изменение статуса | да (чужие) | нет | нет |
| Создание | да (на доступные альбомы/картинки) | |
| Просмотр | да | да (свои) |
| Удаление | да (свои) |
| Предупреждения | Создание | да (с чужой жалобой) | нет | |
| Удаление | да |
| Теги | CRUD | да (свои) | | нет |
| Привязка к картинке | да (к своим картинкам) | |
| Отвязка от картинки |
| Доступы | Удаление | да (на свои альбомы; на чужие к себе) | | нет |
| Настройки | Получение публичных | да | | |
| Получение приватных | да | нет | |
| Изменение |

## 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: роли пользователей или внешних систем, взаимодействующих с системой
* Прецеденты: функциональные возможности или сценарии, которые система предоставляет
* Связи: взаимодействия между акторами и прецедентами
* Включение и расширение: отношения между прецедентами для повторного использования или добавления дополнительного поведения

На диаграмме отражены ключевые функции, такие как управление альбомами, обработка жалоб, работа с изображениями, тегами и доступом, а также взаимодействие администратора и пользователей с системой.

В системе выделено 3 актора: гость, клиент и администратор; 7 подсистем прецедентов как общие объекты случаев использования (управление альбомами, картинками, доступами, жалобами, предупреждениями, тегами и пользователями) и была разработана диаграмма прецедентов (см. Рисунок 3).

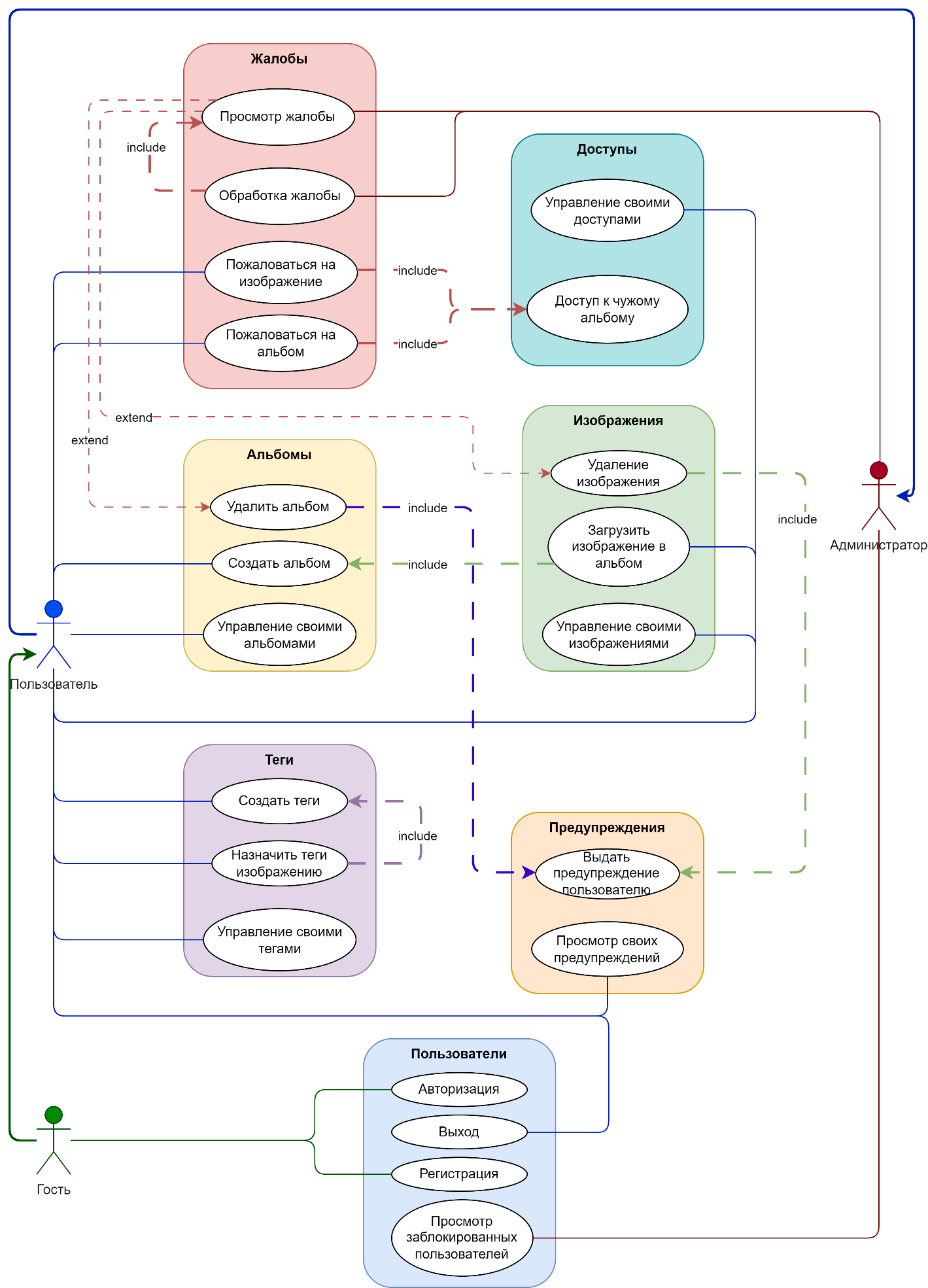


Рисунок 3 – Диаграмма прецедентов

## 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: участники взаимодействия
* Сообщения: вызовы методов или передача данных между объектами
* Временная ось: вертикальная линия, показывающая порядок событий
* Активация: период активности объекта при выполнении операции
* Фреймы: 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>)».

Валидация тела POST запроса осуществляется посредником, проверяя заданные правила на ключах (см. Рисунок 4). Имя ссылки: «валид».

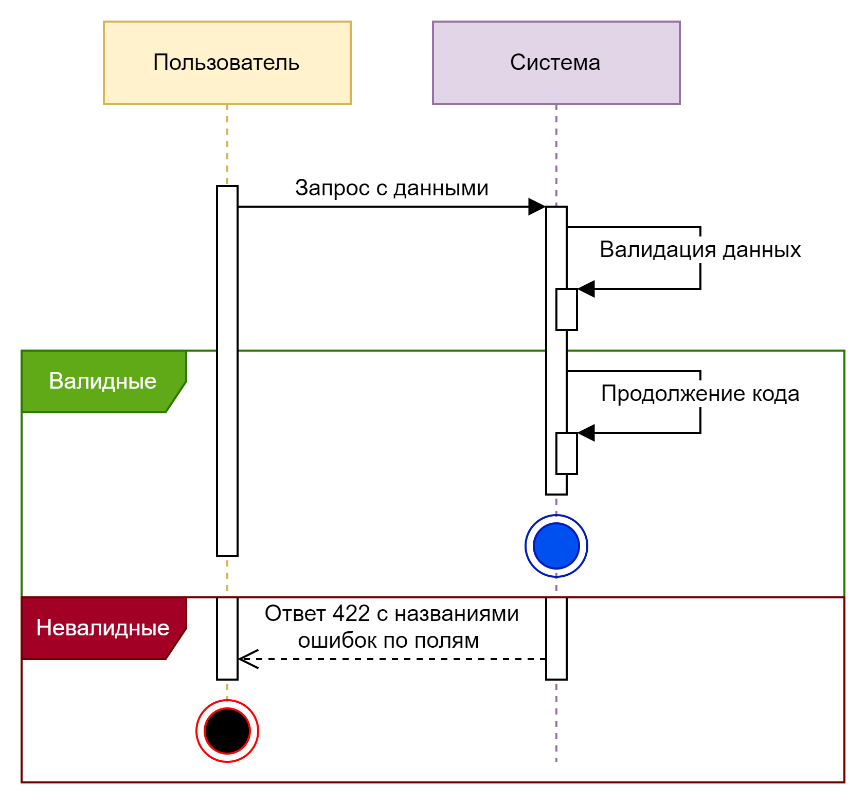


Рисунок 4 – Валидация тела запроса

Внесения в контекст кода модель пользователя и отброс неавторизированных осуществляется посредником, проверяя токен в заголовках запроса и запрашивая пользователя с этим ИД (см. Рисунок 5). Имя ссылки: «токен».

Внесения в контекст кода модель объекта по ИД и отброс если не модель не найдена осуществляется посредником, беря из маршрута ИД и запрашивая из БД её модель (см. Рисунок 6). Имя ссылки: «сущ».

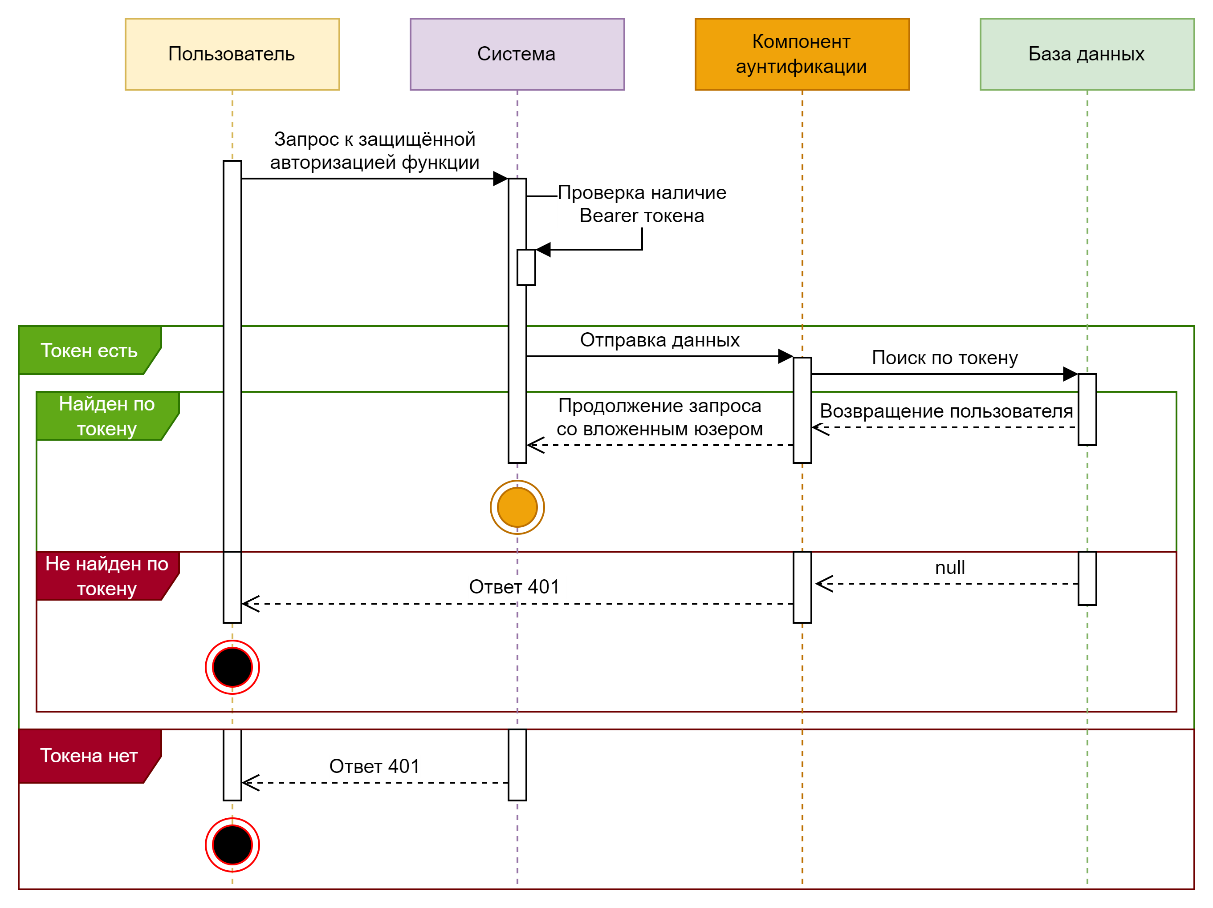


Рисунок 5 – Проверка токена и внесение в контекст пользователя

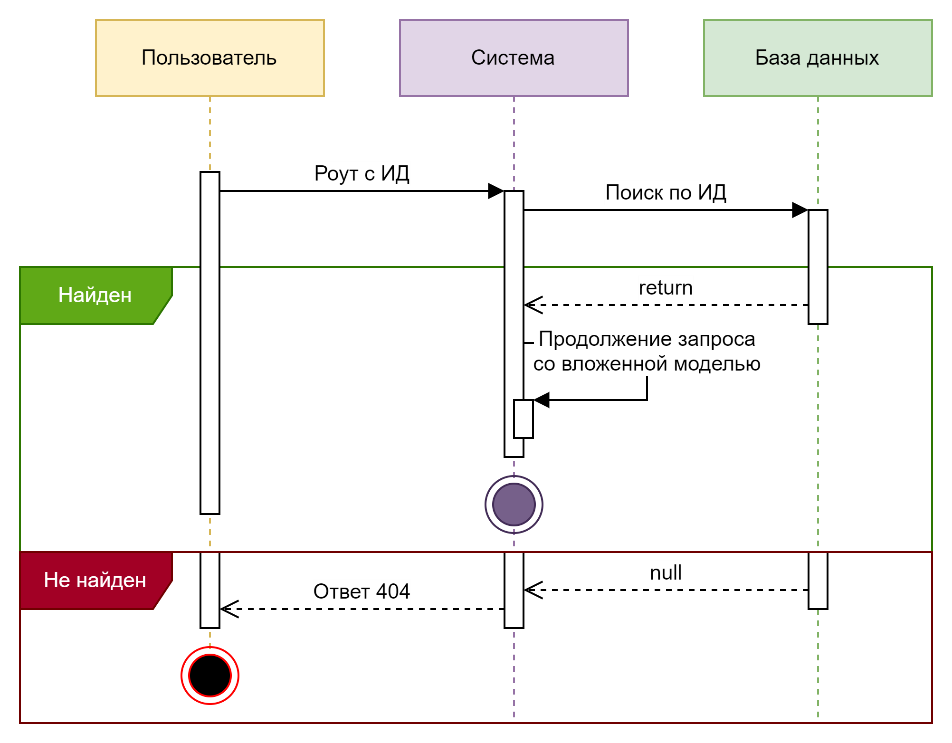


Рисунок 6 – Проверка существования объекта БД

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. Рисунок 7). Имя ссылки: «доступ».

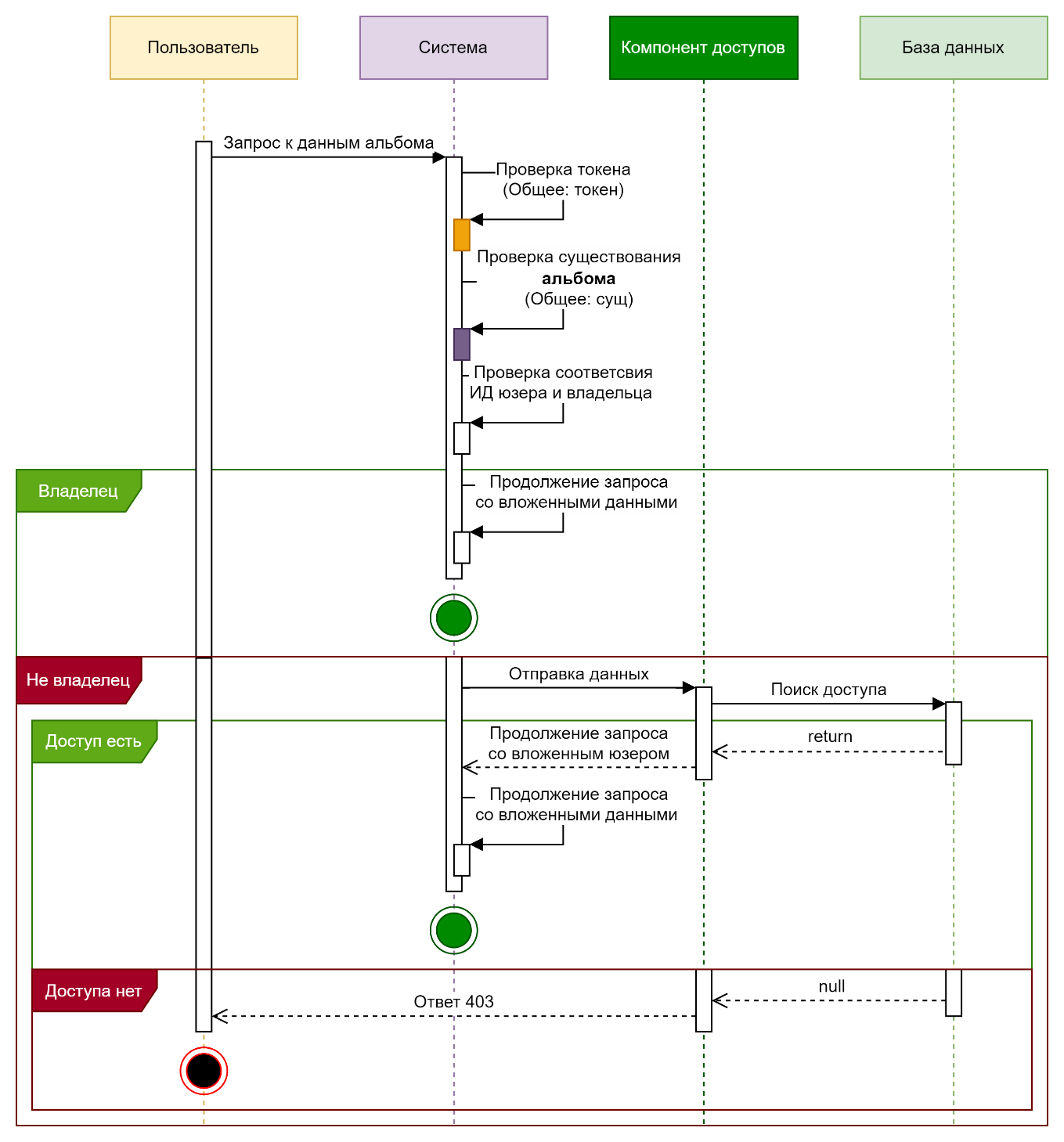


Рисунок 7 – Проверка доступа к альбому

### Настройки сервера

Изменение настроек сервера администратором на Рисунок 8.

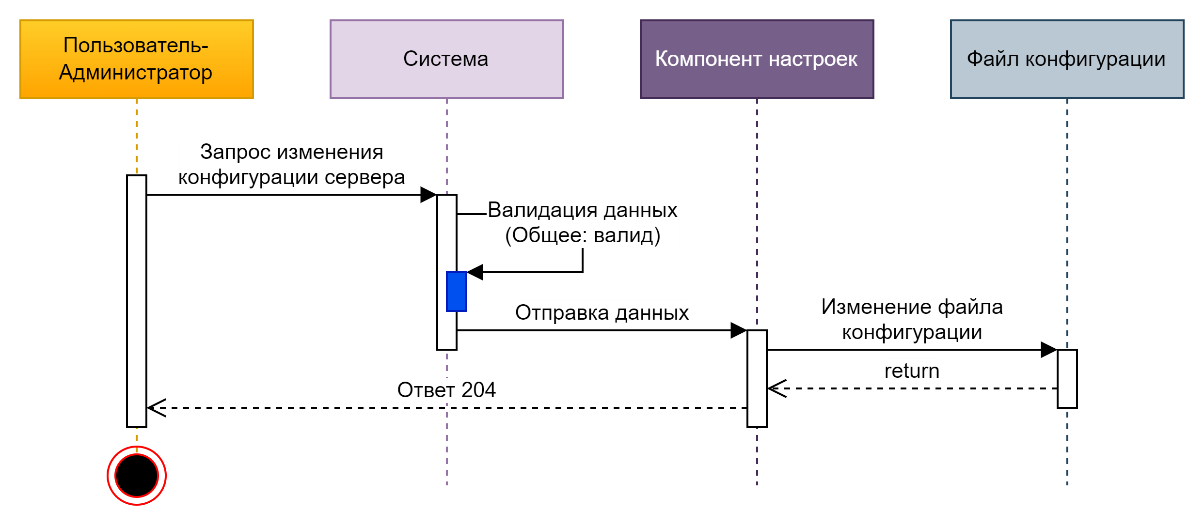


Рисунок 8 - Изменение настроек

### Авторизационные функции

Регистрация новой учётной записи и последующая авторизация представлена на Рисунок 9.

Вход по реквизитам представлена на Рисунок 10.

Выход и удаление токена представлено на Рисунок 11.

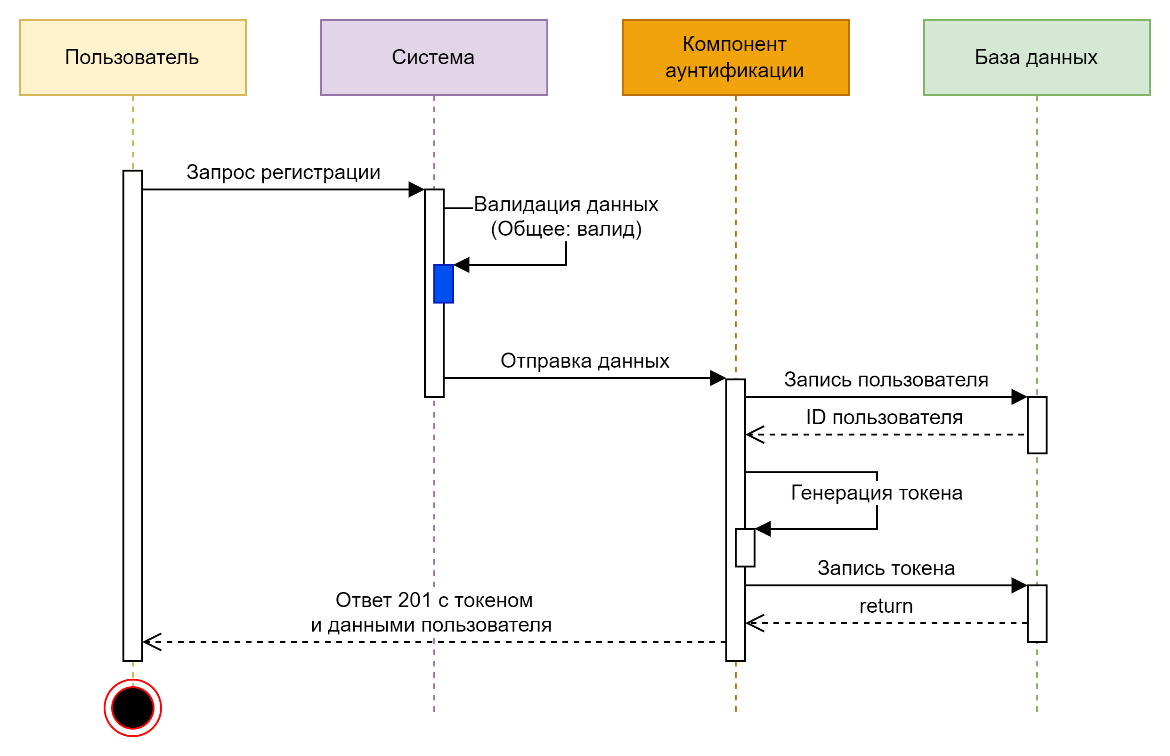


Рисунок 9 – Регистрация

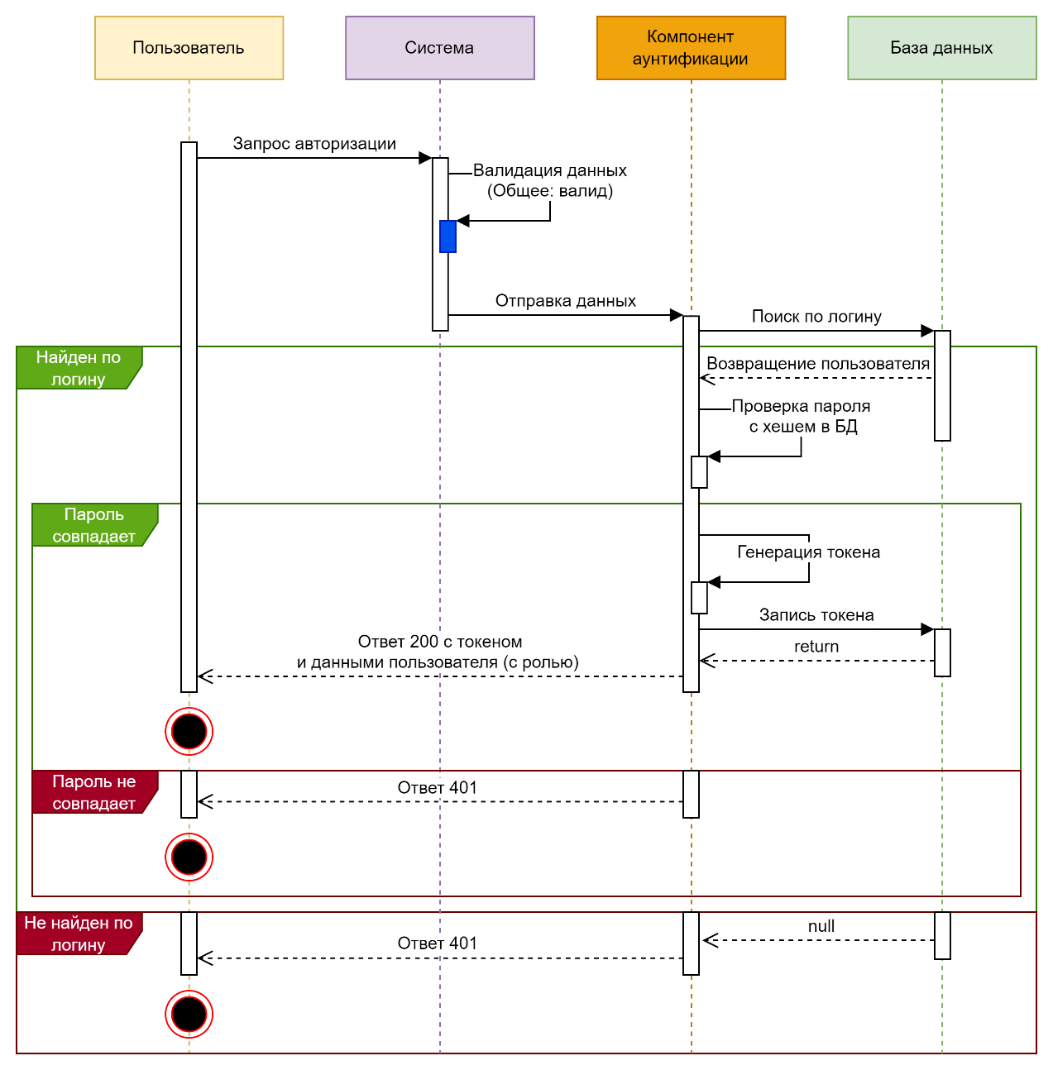


Рисунок 10 – Вход

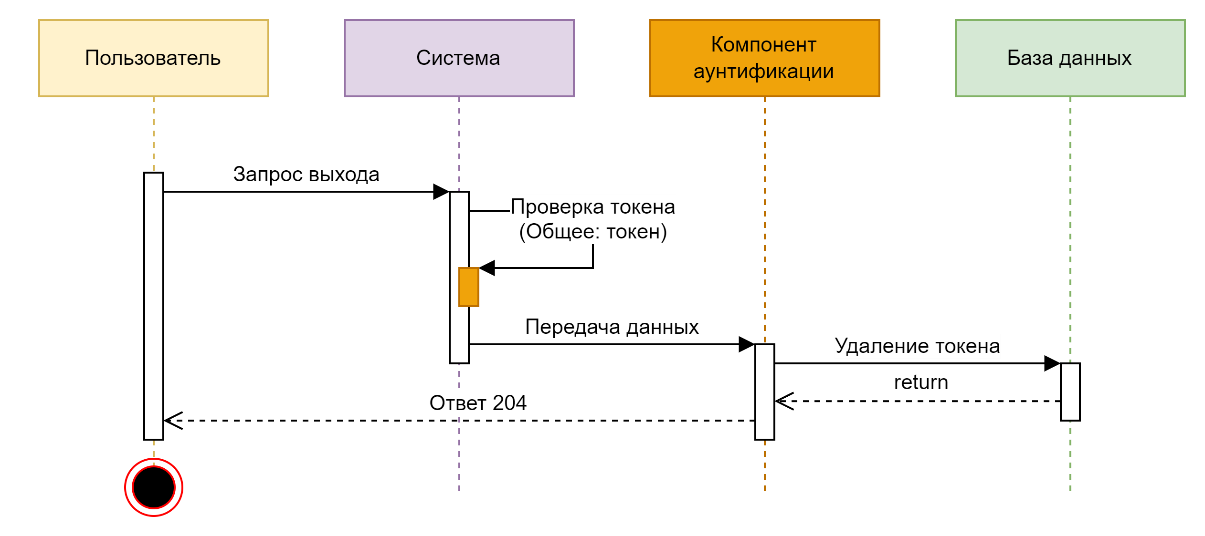


Рисунок 11 – Выход

### Работа с альбомами

Создание нового личного альбома представлена на Рисунок 12.

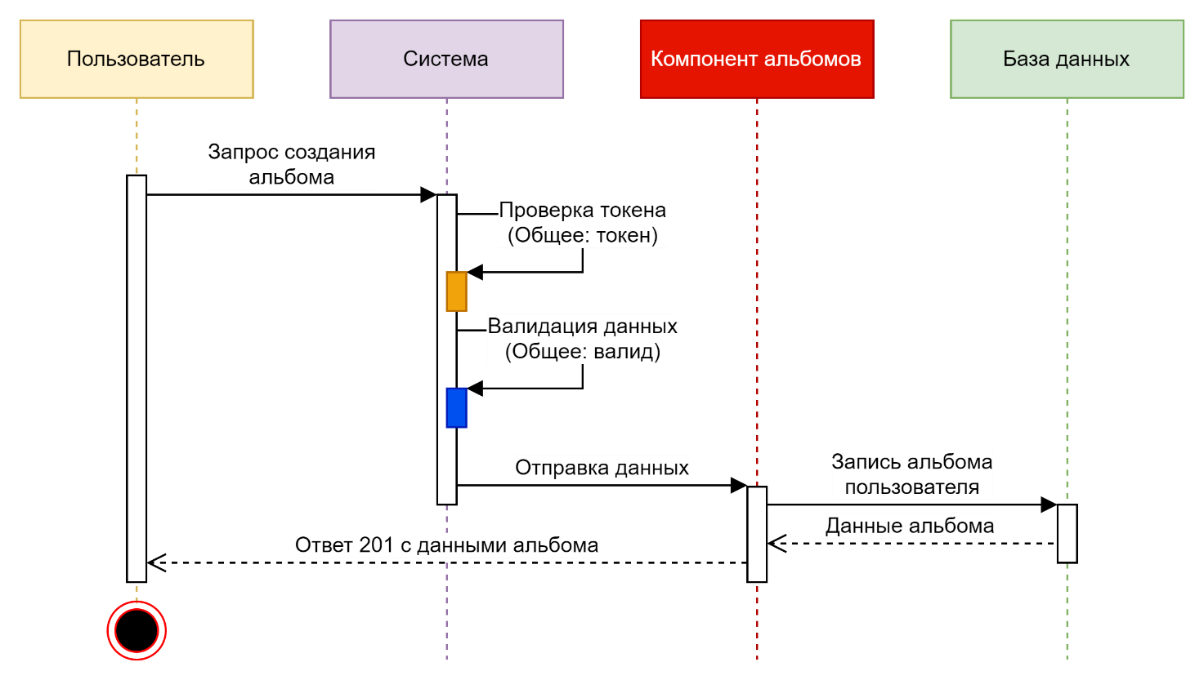


Рисунок 12 – Создание альбома

Получение списка личных и доступных альбомов представлена на Рисунок 13.

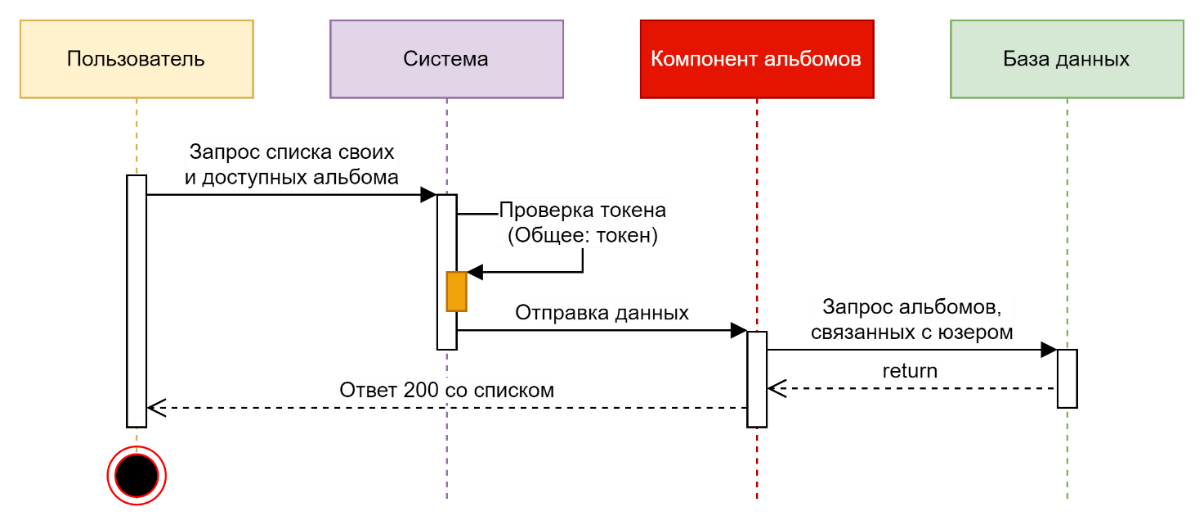


Рисунок 13 – Получение списка альбомов

Обновление данных об личном альбоме представлено на Рисунок 14.

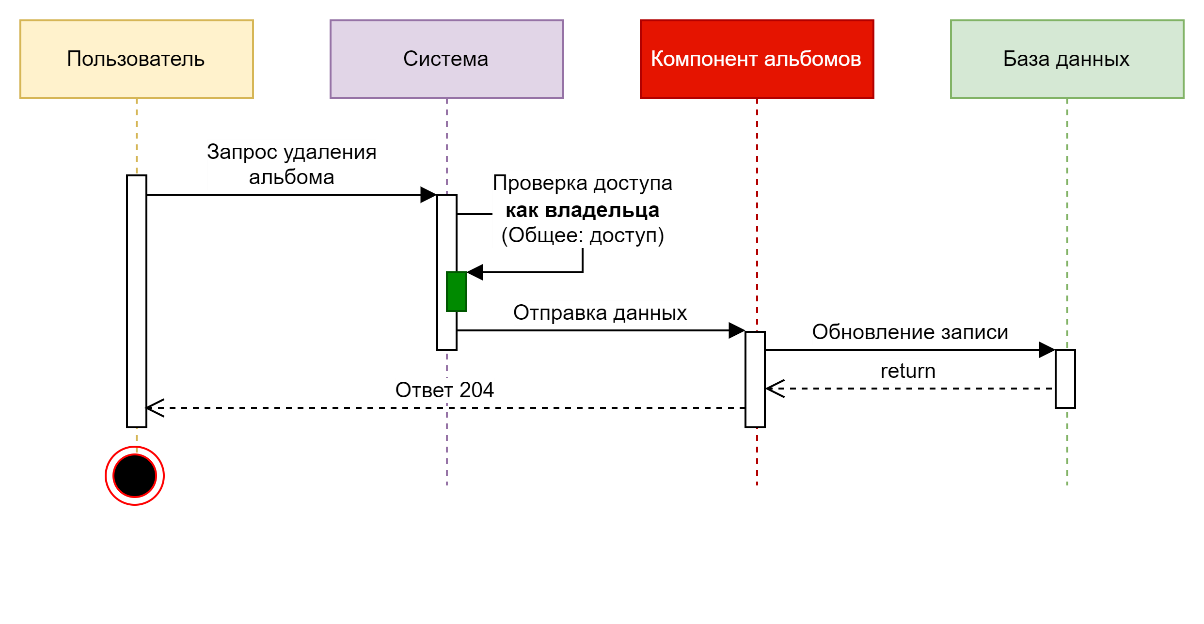


Рисунок 14 – Изменение альбома

Удаление личного альбома в БД и всех связанных файлов представлена на Рисунок 15.

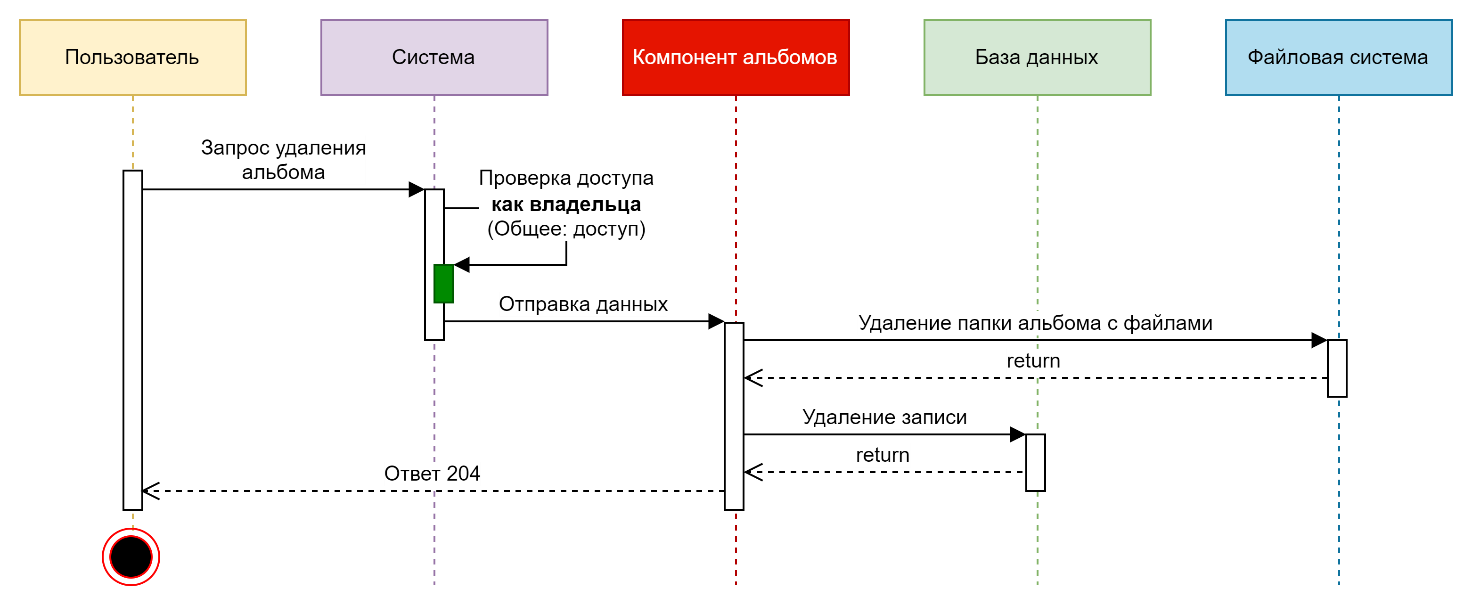


Рисунок 15 – Удаление альбома

### Приглашения на доступ к альбому

Создание приглашения на свой альбом представлено на Рисунок 16.

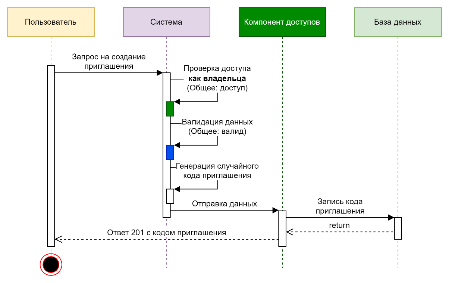


Рисунок 16 – Создание приглашения

Получение доступа к чужому альбому через код доступа представлено на Рисунок 17.

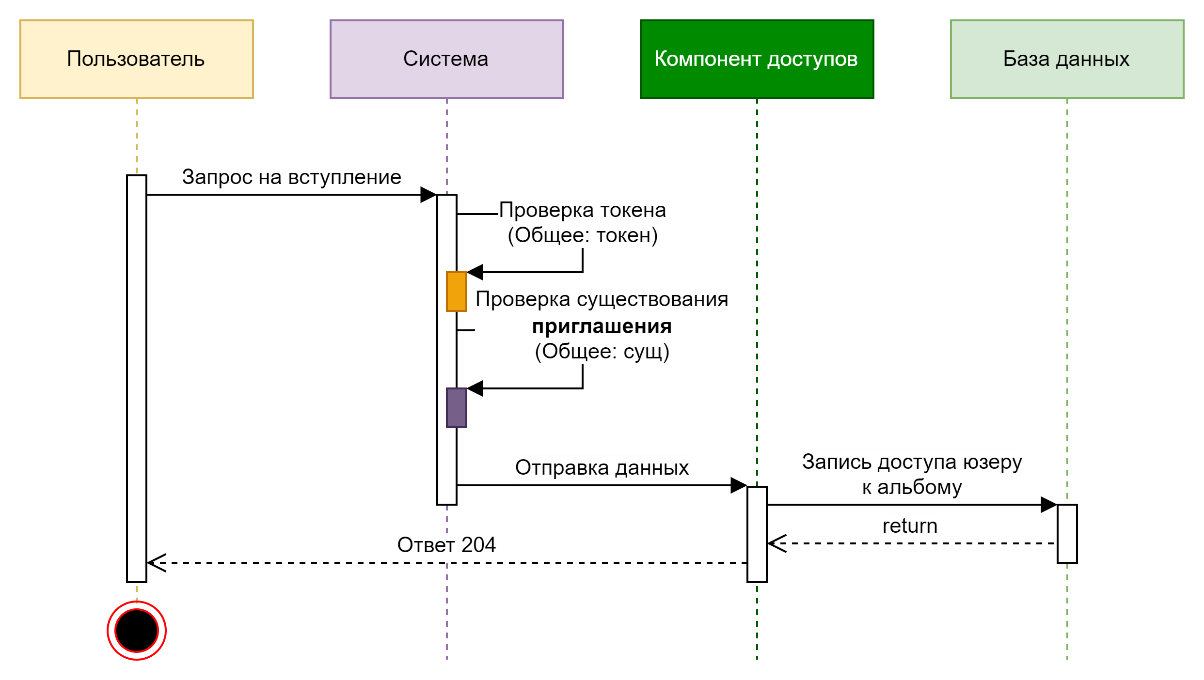


Рисунок 17 – Вступление на чужой альбом

### Работа с картинками

Загрузка на сервер в свой альбом картинок представлена на Рисунок 18.

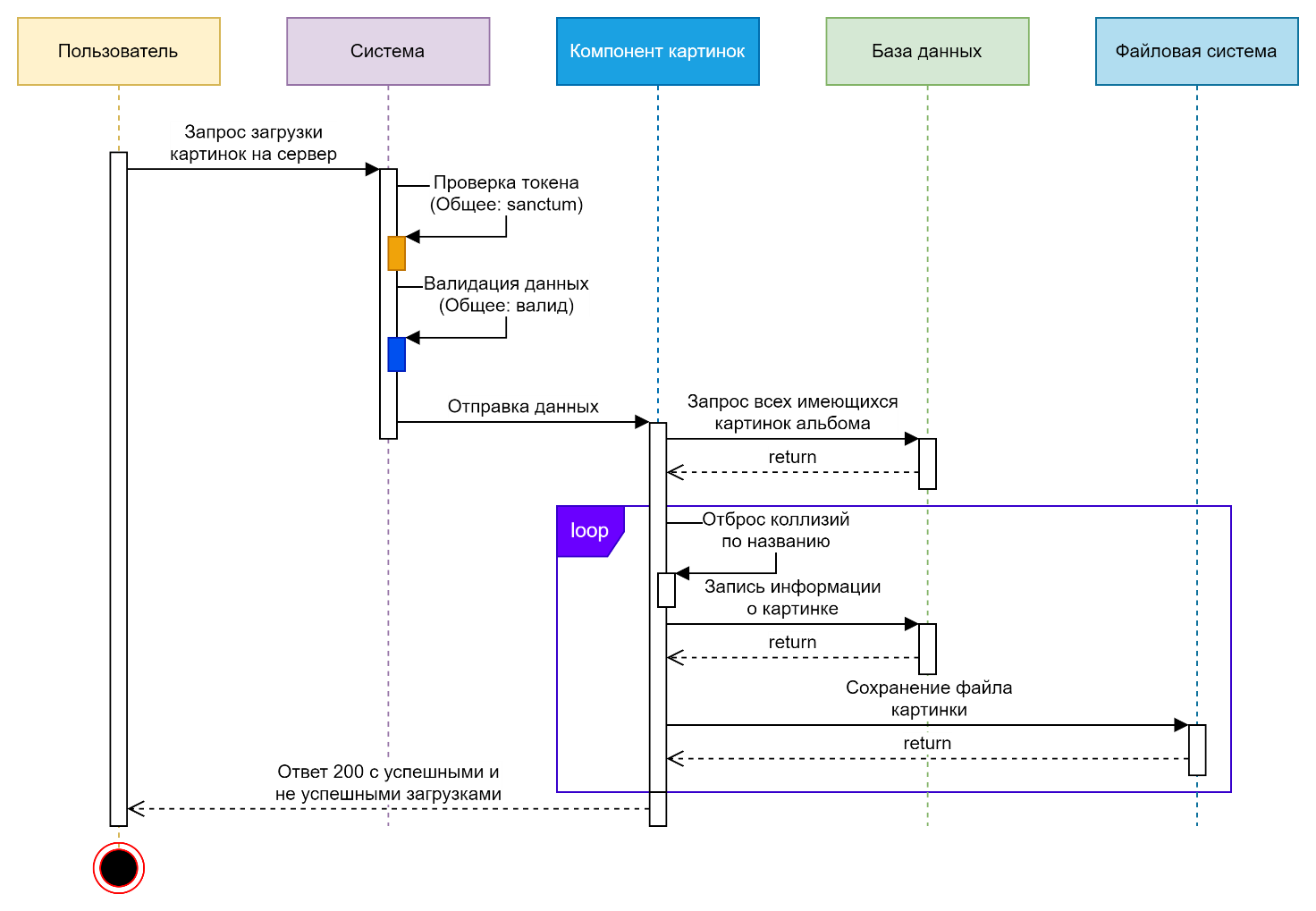


Рисунок 18 – Загрузка картинок в альбом

Получение списка картинок в альбоме представлена на Рисунок 19.

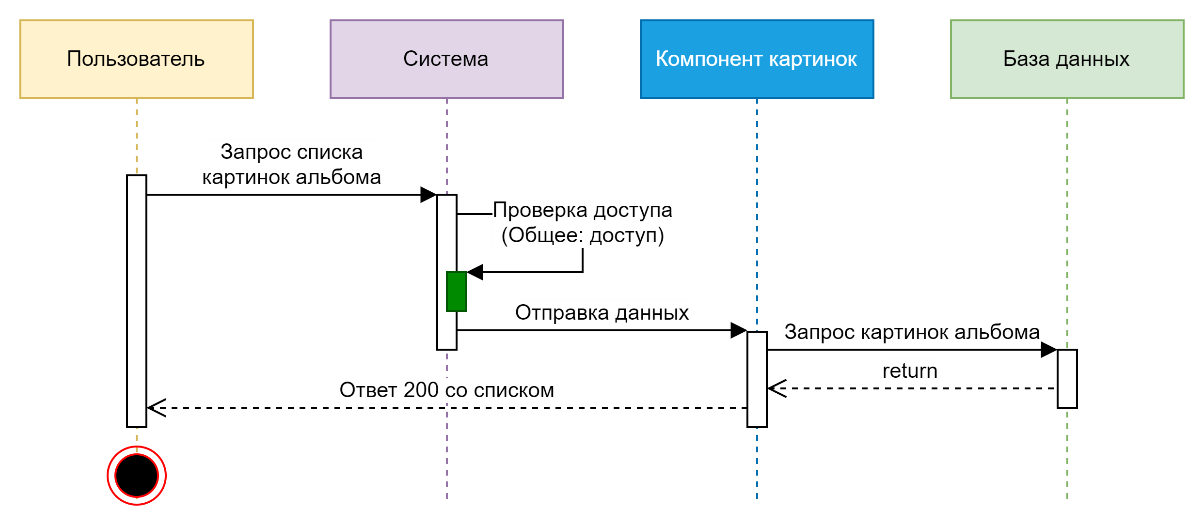


Рисунок 19 – Получение картинок альбома

Скачивание с сервера картинки представлено на Рисунок 20.

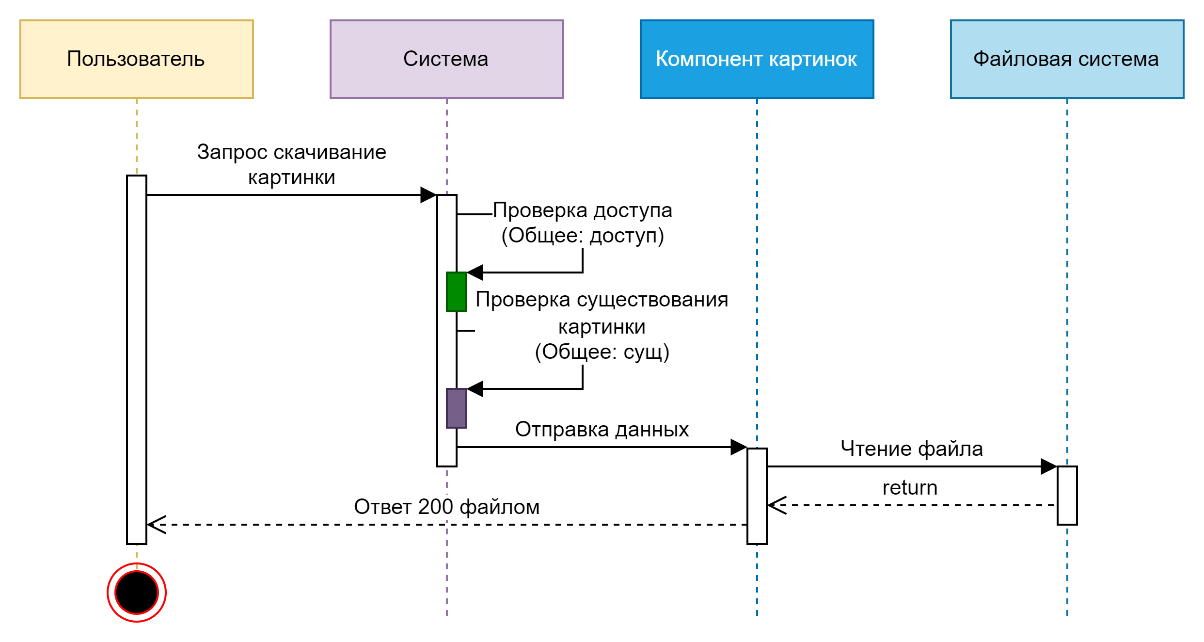


Рисунок 20 – Скачивание картинки

Удаление с сервера картинки представлено на Рисунок 21.

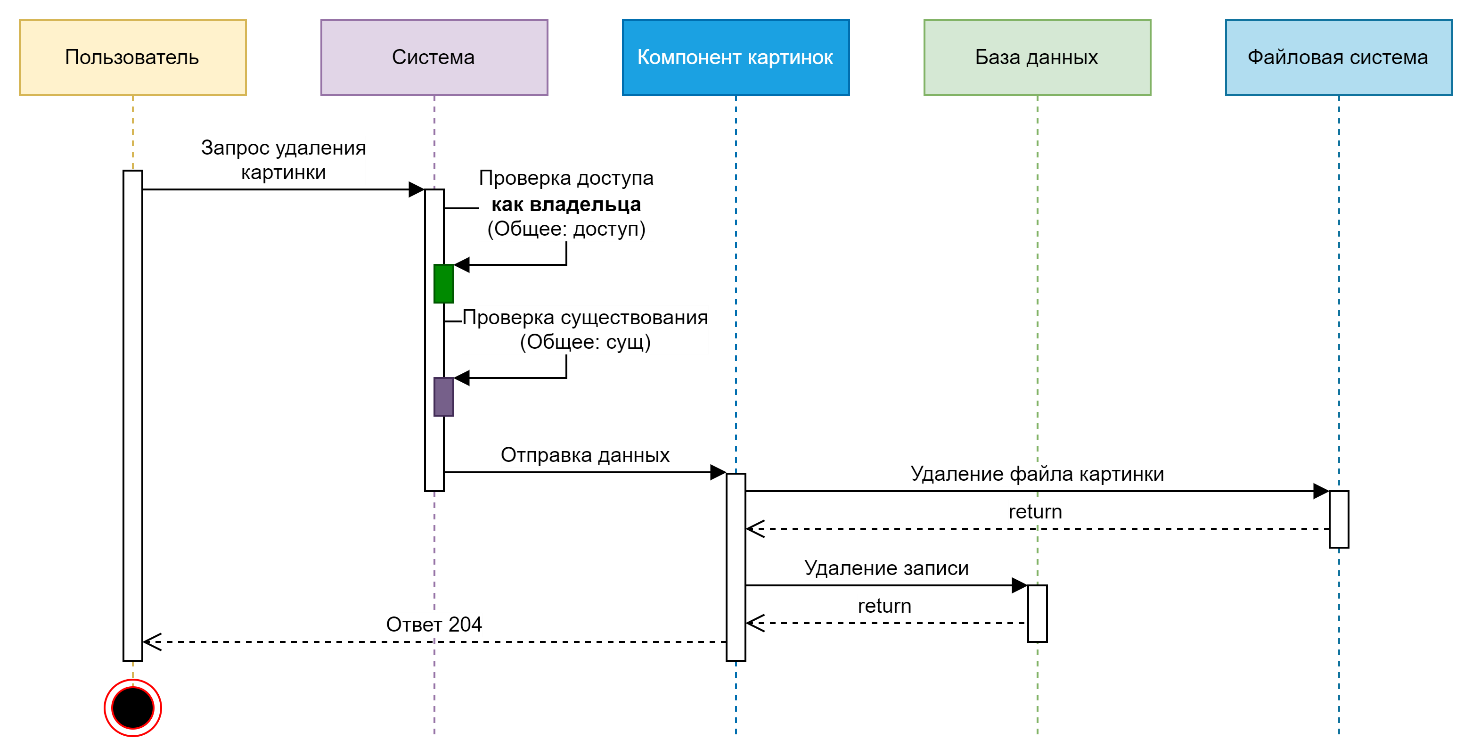


Рисунок 21 – Удаление картинки

### Работа с жалобами

Создание жалобы на доступный чужую картинку/альбом представлена на Рисунок 22.

Изменение статуса всех жалоб на картинке/альбоме представлена на Рисунок 23.

Отзыв владельцем жалобы представлена на Рисунок 24.

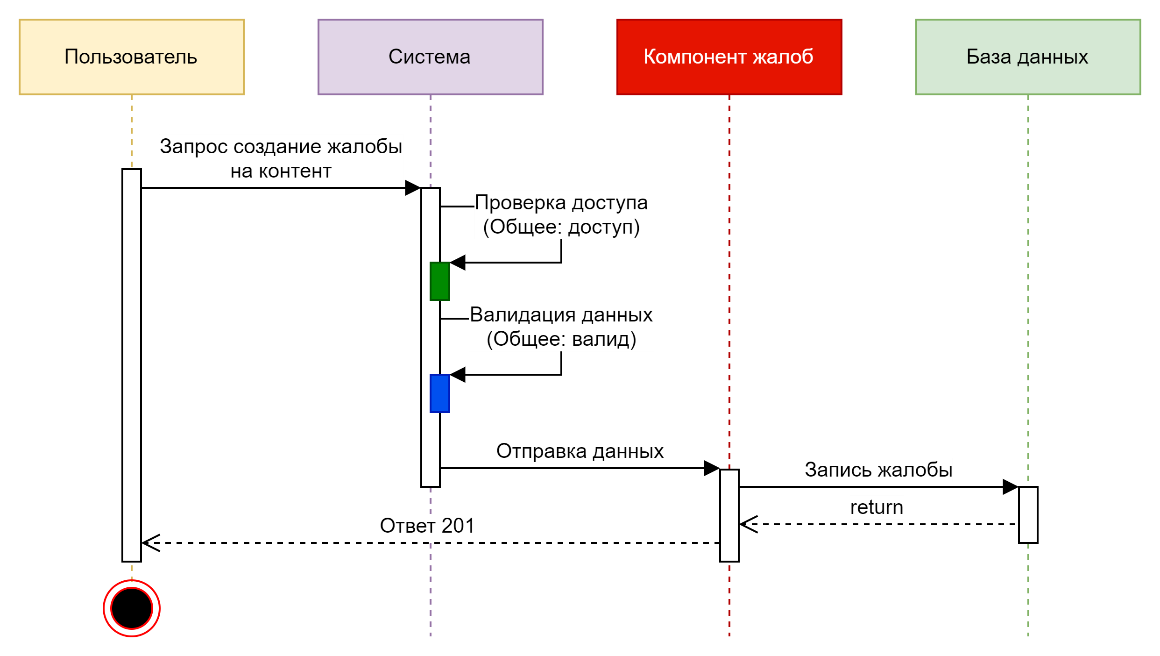


Рисунок 22 – Создание жалобы

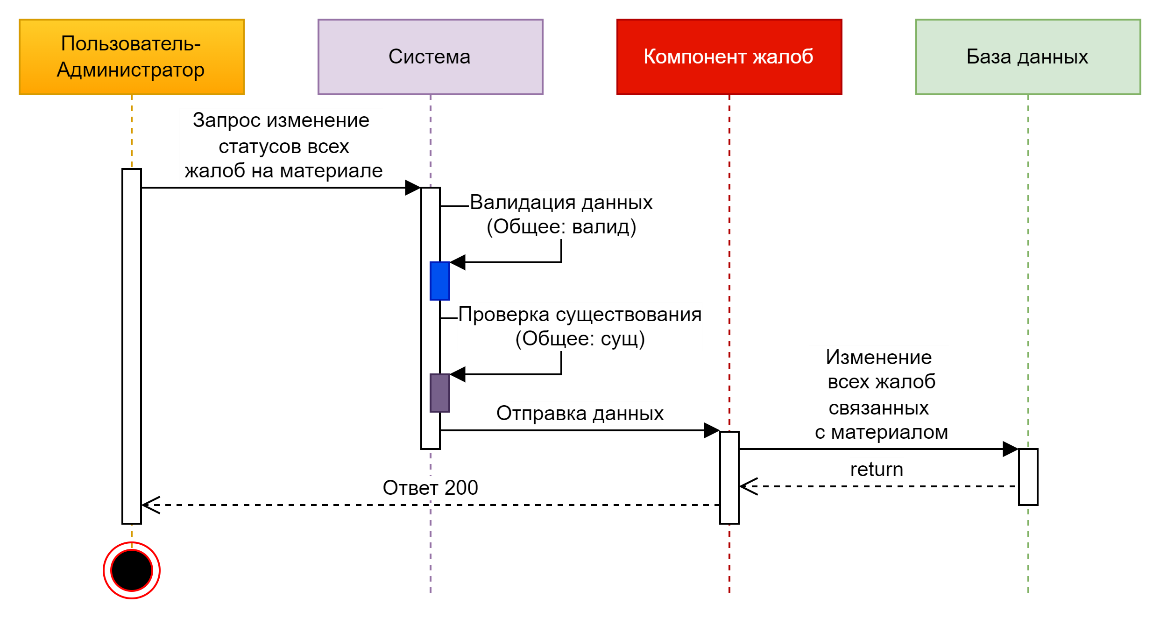


Рисунок 23 – Изменение жалобы

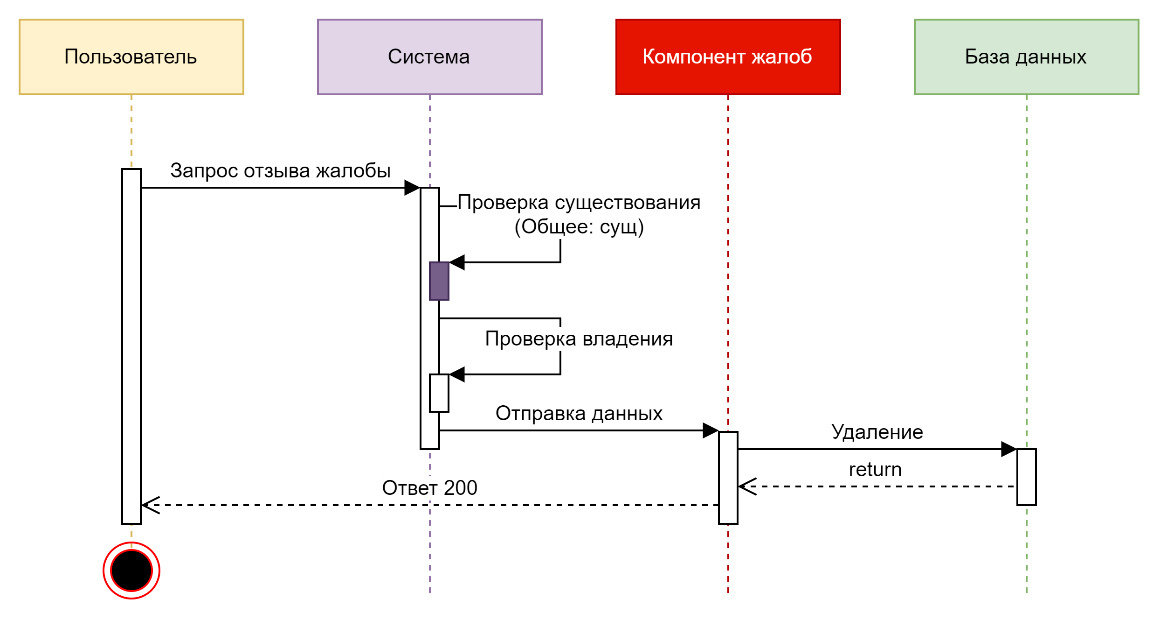


Рисунок 24 – Отзыв жалобы

### Работа с предупреждениями

Создание предупреждения пользователю администратором представлена на Рисунок 25.

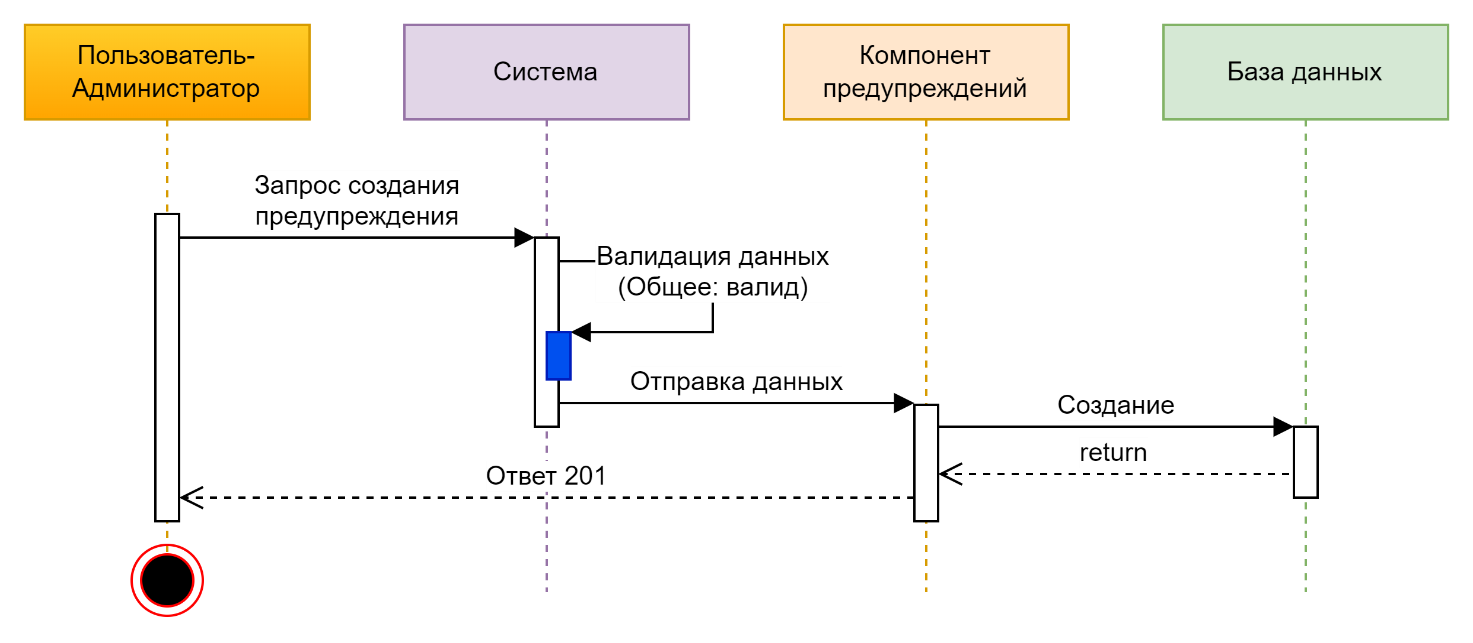


Рисунок 25 – Создание предупреждения

Отзыв предупреждения с пользователя администратором представлена на Рисунок 25.

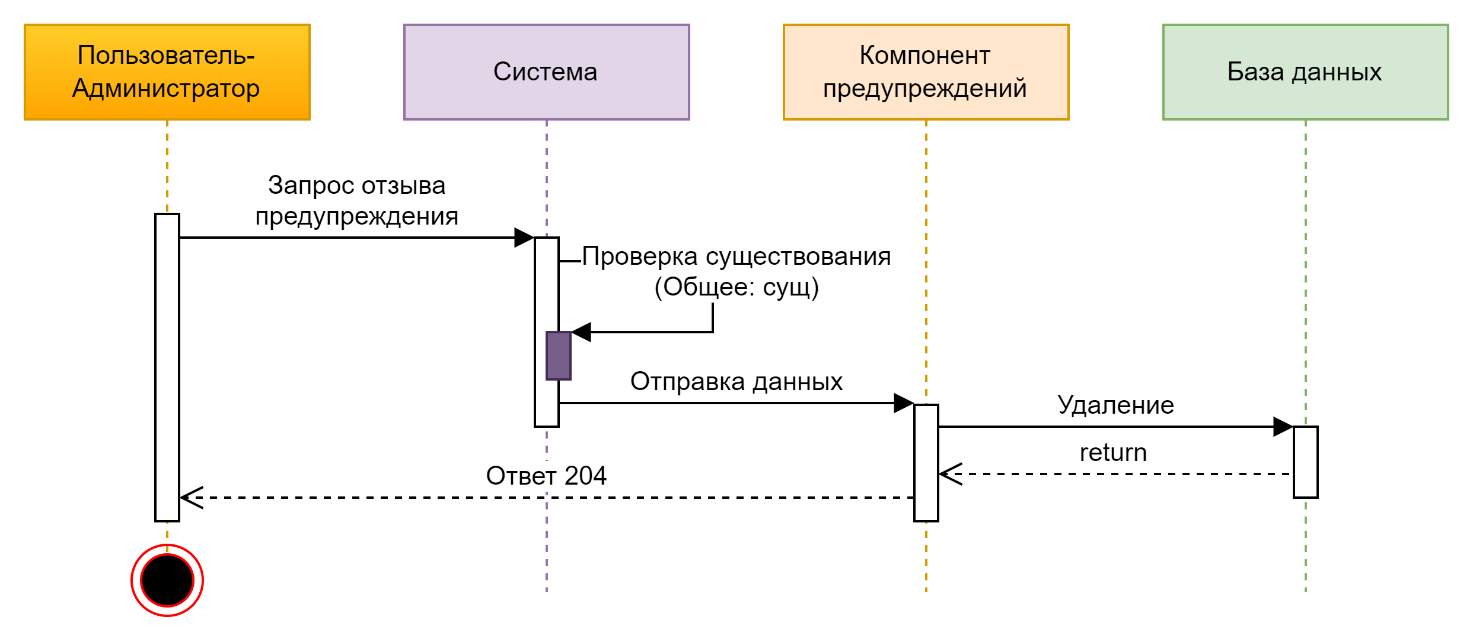


Рисунок 26 – Отзыв предупреждения

## Технологический стек

**Серверная часть**

Язык программирования: PHP 8.3

Фреймворк для разработки: Laravel 11

Система управление базами данными: MySQL

**Мобильные приложение**

Язык программирования и разметки: C#, XAML

Фреймворк для разработки: .NET MAUI

## Хранимые данные

1. **Токены:** значение и связь с пользователем
2. **Пользователи:** никнейм, роль, логин, хеш пароля и состояние бана
3. **Альбомы:** название, связанный пользователь и путь на клиентском устройстве
4. **Доступы к альбомам:** пользователь и альбом
5. **Приглашения доступа:** связанный альбом, ссылка, лимит вступлений и дата просрочки
6. **Картинки:** связанныйальбом, название, размер файла, ширина, высота, дата, хеш и связанные теги
7. **Теги:** название, связанный пользователь
8. **Жалобы:** пояснение, тип, пользователь-отправитель, пользователь на кого жалоба, альбомом и картинка
9. **Предупреждения:** пользователь и пояснение

## Требования к REST 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 к защищенным авторизацией функциям системы во всех запросах необходимо возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 401 Unauthorized |

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

|  |
| --- |
| **Response** |
| **Status:** 403 Forbidden |

При попытке получить несуществующий ресурс возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 404 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:** {  "errors": [  <ключ>: [  <ошибка>,  <ошибка>,  ...  ],  <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.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/register  **Method:** POST  **Content-Type:** application/json  **Body:** {  "name": **{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://api.pics.com |
| 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! |

#### Авторизация

Запрос позволяет авторизировать пользователя и получить токен доступа.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/login  **Method:** POST  **Content-Type:** application/json  **Body:** {  "login": **{login}**  "password": **{password}**  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "token":<**токен**>  "user": {  <**"role":** **"admin"**>  "name": "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}**/logout  **Method:** POST  **Authorization:** Bearer **{token}** | **Status:** 204 No Content  **Возможные статус-коды:** 200, 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 личного альбома пользователя. Возвращает объект альбома с его ID в БД и создаёт папку с выведенным названием в ФС.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "**name**": **{name}**,  "path": **{path}**  } | **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "id": <**id\_альбома**>,  "name": **{name}**,  "path": **{path}**  }  **Возможные статус-коды:** 201, 401, 409, 422 |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | name | string | да | макс: 255, уник(для юзера) | Название | Камера |
| 2 | path | string | нет | макс: 255, уник(для юзера) | Путь на клиенте для синхронизации | DCIM/Camera |

#### Получение альбома

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

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**  **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": "Чел"  }, { ... }, { ... }  ],  "picturesCount": 31,  "picturesInfo": {  "sign": "2\_JDJ5JDEyJG11RGc24aWRX",  "ids": [84, 30, 69, 59]  }  }  }  Чужой доступный  {  "album": {  "id": 1,  "name": "Скаченное",  "owner": {  "id": 215,  "name": "Чел"  },  "picturesCount": 29,  "picturesInfo": {  "sign": "2\_JDJ5JDEyJHoMVpWaEJjVzNSV2",  "ids": [49, 41, 30, 87]  }  }  }  **Возможные статус-коды:** 200, 401, 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 все картинки в ФС.

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

#### Получение списка альбомов

Запрос для получения списка альбомов. Возвращает список всех альбомов пользователя и доступных из БД.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:**  {  "own": [  {  "id": 354,  "name": "Камера",  "path": "DCIM/Camera",  "createdAt": "2024-12-16T10:13:23",  "grantAccessesCount": 3,  "picturesCount": 31,  "picturesInfo": { ... }  }, { ... }, { ... }  ],  "accessible": [  {  "id": 5372,  "name": "Скаченное",  "path": "Pictures/Downloads",  "createdAt": "2024-12-16T10:13:23",  "owner": {  "id": 215,  "name": "Чел"  },  "picturesCount": 31,  "picturesInfo": { ... }  }, { ... }, { ... }  ]  }  **Возможные статус-коды:** 200, 401 |

### 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.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**/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/**{code}**/album  **Method:** GET | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "album": {  "id": 3,  "name": "Огурец",  },  "pictures": [  {...}, {...}, {...}  ]  }  **Возможные статус-коды:** 200, 404 |

Входные параметры:

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

#### Вступление по приглашении

Запрос для получения доступа к альбому чужого пользователя по коду приглашению.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/invitation/**{code}**/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 | **Status:** 204 No Content  **Возможные статус-коды:** 204, 403, 404 |

#### Отзыв доступа для пользователя

Запрос для удаления доступа к своему альбому другого пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{album}**/accesses/**{user}**  **Method:** DELETE | **Status:** 204 No Content  **Возможные статус-коды:** 204, 403, 404 |

#### Отзыв доступа на себе

Запрос для удаления доступа к чужому альбому от себя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{album}**/accesses  **Method:** DELETE | **Status:** 204 No Content  **Возможные статус-коды:** 204, 403, 404 |

### Работа с картинками

#### Загрузка картинок в альбом

Запрос для загрузки на ФС сервера картинок и информации об каждой в БД

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**/pictures  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** multipart/form-data  **Body:** {  "pictures[]": **{pictures}**,  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "successful": [  {  "id": 1,  "name": "test.jpg",  "date": "2024-10-29 19:56:10",  "size": 1696738,  "width": 1360,  "height": 2128,  “hash”: "df59b7d94c03b59a”  }, { ... }  ],  "errored": [  {  "name": "image.png",  "message": “already exist"  }, { ... }  ]  }  **Возможные статус-коды:** 200, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** |
| 1 | pictures | array | да | мин: 1 | Массив картинок |
| 1 | pictures.\* | file | нет | mimes: jpeg,png,gif | Картинка |

#### Получение списка картинок

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

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**/pictures  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "sign": <**сигнатура**>  "pictures": [  {  "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 |

#### 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/**{albumId}**  /pictures/**{pictureId}**  /thumb/**{orient}{size}**  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** image/jpeg  **Возможные статус-коды:** 200, 301, 403, 404 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | pictureId | integer | да | Код картинки | 529 |
|  | orient | "w"|"h"|"q" |  | Ось указания размера (по ширине, высоте или обрезка в квадрат) |  |
| 2 | size | integer | да | Размер по оси (пиксели) | 720 |
| 3 | 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/**{albumId}**  /pictures/**{pictureId}**  /original  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** image/<"jpeg"|"png"|...>  **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/**{albumId}**  /pictures/**{pictureId}**  /download  ?sign=**{sign}**  **Method:** GET | **Status:** 200 OK  **Content-Type:** image/<"jpeg"|"png"|...>  **Content-Disposition:**  attachment; filename=<имя\_файла>  **Возможные статус-коды:** 200, 403, 404 |

#### Удаление картинки

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

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/**{albumId}**  /pictures/**{pictureId}**  **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:** {  "value": **{value}**  } | **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 | value | 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 картинке

Запрос для добавления связи тега с картинкой.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/{**albumId**}  /pictures/**{pictureId}**  /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 картинки

Запрос для удаления связи тега с картинкой.

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

### Работа с жалобами

#### Создание жалобы на альбом

Запрос для создания жалобы на альбом.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/{**albumId**}  /complaint  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "description": **{description}**,  "typeId": **{typeId}**  } | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404, 409 |

#### Создание жалобы на картинку

Запрос для добавления жалобы на картинку.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/albums/{**albumId**}  /pictures/**{pictureId}**  /complaint  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "description": **{description}**,  "typeId": **{typeId}**  } | **Status:** 204 No Content  **Возможные статус-коды:** 204, 401, 403, 404, 409 |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Условия** | **Описание** |
| 1 | description | string | да | макс: 255 | Пояснение |
| 2 | typeId | integer | да | существовать | Тип жалобы |

#### Получение списка жалоб

Запрос для получения списка жалоб. Администратору выведутся все, пользователю — жалобы который он создал.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/complaints  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** [  {  "id": <**id\_жалобы**>,  "description": <**пояснение**>,  "from\_user": <**данные\_жалующегося**>,  "about\_user": <**данные\_на\_кого\_жалоба**>,  <"album"|"picture">: {  <**данные\_предмета\_жалобы**>  }  }, { ... }  ]  **Возможные статус-коды:** 200, 401, 403 |

### Работа с предупреждениями

#### Создание предупреждения

Запрос для добавления предупреждения на пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/**{user}**/warnings  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "comment": **{description}**  } | **Status:** 201 Created  **Возможные статус-коды:** 201, 401, 403, 404 |

Входные параметры:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Обяз** | **Описание** |
| 1 | comment | string | нет | Пояснение |

#### Удаление предупреждения

Запрос для удаления предупреждения с пользователя.

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

### 

### 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:** POST  **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.

|  |  |
| --- | --- |
| **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">  "complaintsAboutCount": 21,  "complaintsAboutAcceptedCount": 0,  "complaintsFromCount": 18,  "complaintsFromAcceptedCount": 9,  "albumsCount": 1,  "albumsViaAccessCount": 2,  "picturesCount": 31  }, { ... }, { ... }  ]  }  **Возможные статус-коды:** 200, 401, 403 |

#### Просмотр пользователя

Запрос просмотра пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/**{userId}**  **Method:** GET  **Authorization:** Bearer **{token}** | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 2,  "name": "Чел",  "isBanned": false,  "login": "test1",  <"role": "admin">  "complaintsAbout": [  {  "id": 1,  "status": null,  "description": "Плохие картинки"  }, { ... }, { ... }  ],  "complaintsFrom": [  {  "id": 1,  "status": null,  "description": "Мне не нравится"  }, { ... }, { ... }  ],  "albumsCount": 1,  "albumsViaAccessCount": 2,  "picturesCount": 31,  "warnings": [  {  "id": 1,  "comment": "Больше так не делай"  }, { ... }, { ... }  ],  }  }  **Возможные статус-коды:** 200, 401, 403, 404 |

#### Редактирование пользователя

Запрос редактирование пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/users/**{userId}**  **Method:** POST  **Authorization:** Bearer **{token}**  **Content-Type:** application/json  **Body:** {  "name": **{nickname}**,  "login": **{login}**,  "password": **{password},**  "role\_id": **{roleId}**,  "is\_banned": **{**isBanned**},**  } | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "user": {  "id": 2,  "name": "Чел",  "isBanned": false,  "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 |
| 2 | isBanned | boolean | нет |  | Статус блокировки | true |

### Работа с настройками

#### Просмотр публичных настроек

Запрос публичных настроек.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL: {api}**/  **Method:** GET | **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "settings": {  "allowed\_upload\_mimes": [  "jpeg", "jpg", "png", "gif"  ],  "allowed\_preview\_sizes": [  144, 240, 360, 480, 720, 1080  ],  "warning\_limit\_for\_ban": 3,  "free\_storage\_limit": 5368709120,  "complaint\_types": [  {  "id": 5,  "name": "Нарушение авторских прав"  },  {  "id": 6,  "name": "Шок-контент"  }  ],  "is\_upload\_disabled": true  }  }  **Возможные статус-коды:** 200 |

## Требования к мобильному приложению клиента

При входе неавторизированного пользователя выводить страницу с переключением регистрации (см. Рисунок 27) и входа (см. Рисунок 28).

|  |  |
| --- | --- |
| Рисунок 27 – Макет регистрации | Рисунок 28 – 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. Рисунок 29) авторизированного пользователя выводить сетку недавних картинок синхронизирующийся альбомов. На каждой картинке выводить статус синхронизации с сервером. Каждая картинка при нажатии открывает полноэкранный просмотрщик (см. Рисунок 30). Также на главной необходима кнопка, которая открывает управление синхронизацией (см. Рисунок 31Рисунок 37).

На странице альбомов (см. Рисунок 32) 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. Рисунок 29).

|  |  |
| --- | --- |
| Рисунок 29 – Макет главной страницы | +  Рисунок 30 – Макет просмотрщика |
| Рисунок 31 – Макет управления синхронизацией | Рисунок 32 – Макет страницы альбомов |

На странице настроек (см. Рисунок 33) отображать информацию о себе с возможностью редактирования, список альбомов, которые открыты пользователям и возможность выйти со всех устройств.



Рисунок 33 – Макет страницы настроек

## Требования к мобильному приложению администратора

При первом запуске приложения предоставляется выбор сервера и добавление нового (см. Рисунок 34).

После выбора сервера страница меняется на авторизацию, на которой пишется выбранный сервер, а также поле для ввода логина и пароля (см. Рисунок 35).

|  |  |  |
| --- | --- | --- |
| Рисунок 34 – Выбор URL |  | Рисунок 35 – Авторизация |

При попытке войти, будучи простым пользователем будет всплывать окно о том, что доступ запрещен. После успешной авторизации становится доступно навигационное меню снизу экрана с названиями: главная, профиль, настройки.

Главная страница выбрана по умолчанию и на ней располагается информация о использованном пространстве, кнопкой обновить жалобы, а также нерассмотренные жалобы сортировкой по альбомам, внутри которых располагаются жалобы на картинки, если они имеются с возможностью перемещать по горизонтали и кнопка просмотреть полностью. Если альбомов много, то добавляется кнопка загрузить еще, по нажатию на которую загружаются альбомы, в противном случае эта кнопка исчезает (см. Рисунок 36).

После нажатия на кнопку просмотреть полностью открывается новая страница с содержимым альбома, располагаются 2 кнопки - создать предупреждение и забанить. Ниже расположена информация о владельце альбома, информация альбома с количеством жалоб и изображений. Далее располагается массив картинок, отсортированный по количеству жалоб на картинку, на самой картинке справа сверху пишется числом кол-во жалоб на неё (см. Рисунок 37).

|  |  |  |
| --- | --- | --- |
| Рисунок 36 – Главная |  | Рисунок 37 – Просмотреть полностью |

По нажатию на картинку можно войти в режим просмотра и рассматривать картинки с функциями приближения (через щипок или двойное нажатие) и перемещения. Сверху картинки имеется кнопка назад, которая ведет обратно в альбом, также пишется название картинки, а чуть правее кнопка удаления. В центре по бокам картинки располагаются кнопки назад и вперед, загружающие картинки (см. Рисунок 38).

Страница с настройками. На данной странице располагается полная информация о том, сколько использовано места на сервере с последним обновлением. Так же там представлена информация о количестве жалоб для блокировки, блокировки загрузки при заполнении хранилища на определенной процент, разрешенные форматы картинок, разрешенные размеры превью, всю эту информацию можно менять, а потом сохранить (см. Рисунок 39).

Страница профиля. На ней располагается форма с 4 полями: никнеймом, логином, паролем и подтверждением пароля. Далее снизу располагаются 2 кнопки сохранить и выход (см. Рисунок 40). По нажатию на кнопку выход пользователь будет перенаправлен на страницу с выбором сервера (см. Рисунок 34 на стр. 40).

|  |  |  |
| --- | --- | --- |
| Рисунок 38 – Просмотр картинки |  | Рисунок 39 – Настройки |
| Рисунок 40 – Профиль | | |

# 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, можно выделить следующие сущности с их атрибутами (см. Рисунок 41):

1. Сущность «**Роль**» с атрибутом «Название».
2. Сущность «**Предупреждение**» с атрибутом «Пояснение».
3. Сущность «**Пользователь**» с атрибутами «Логин», «Никнейм», «Хеш пароля», «Количество жалоб», «Состояние бана».
4. Сущность «**Тег**» с атрибутом «Название».
5. Сущность «**Доступ к альбому**», связывающая сущности «Альбом» и «Пользователь».
6. Сущность «**Токен**», c атрибутом «Содержимое».
7. Сущность «**Альбом**» с атрибутами «Название», «Путь на устройстве».
8. Сущность «**Код** приглашения» с атрибутами «Значение», «Время жизни».
9. Сущность «**Картинка**» с атрибутами «Название», «Хеш», «Размер», «Ширина», «Высота», «Дата».
10. Сущность «**Жалоба**» с атрибутом «Пояснение».
11. Сущность «**Тип жалобы**», c атрибутом «Содержимое».
12. Сущность «**Тег-Картинка**», связывающая сущности «Тег» и «Картинка».

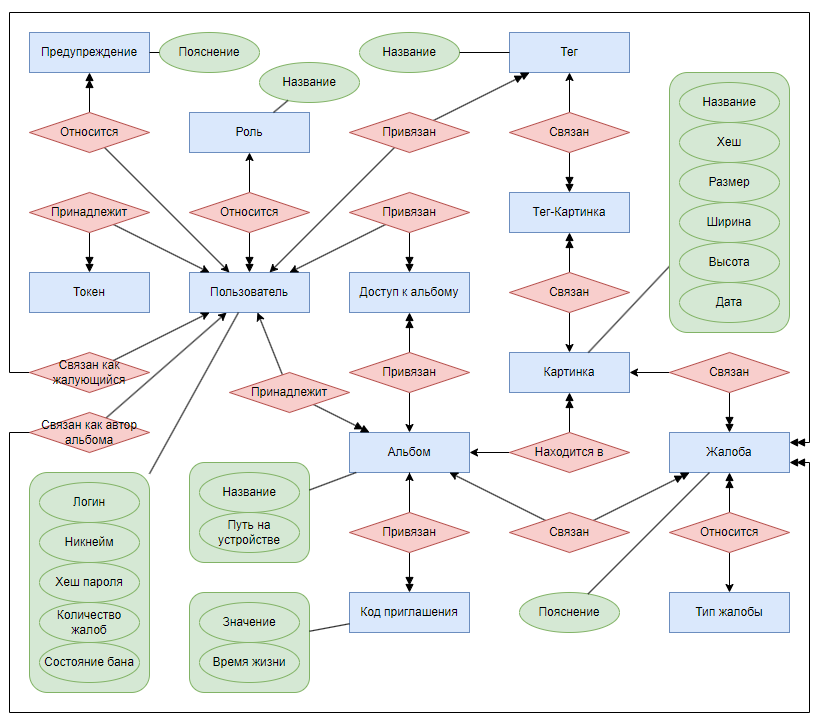


Рисунок 41 – Концептуальная модель данных

## 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 нашу концептуальную модель данных и выполнив нормализацию данных до 3 нормальной формы включительно, построим логическую модель данных (см. Рисунок 42):

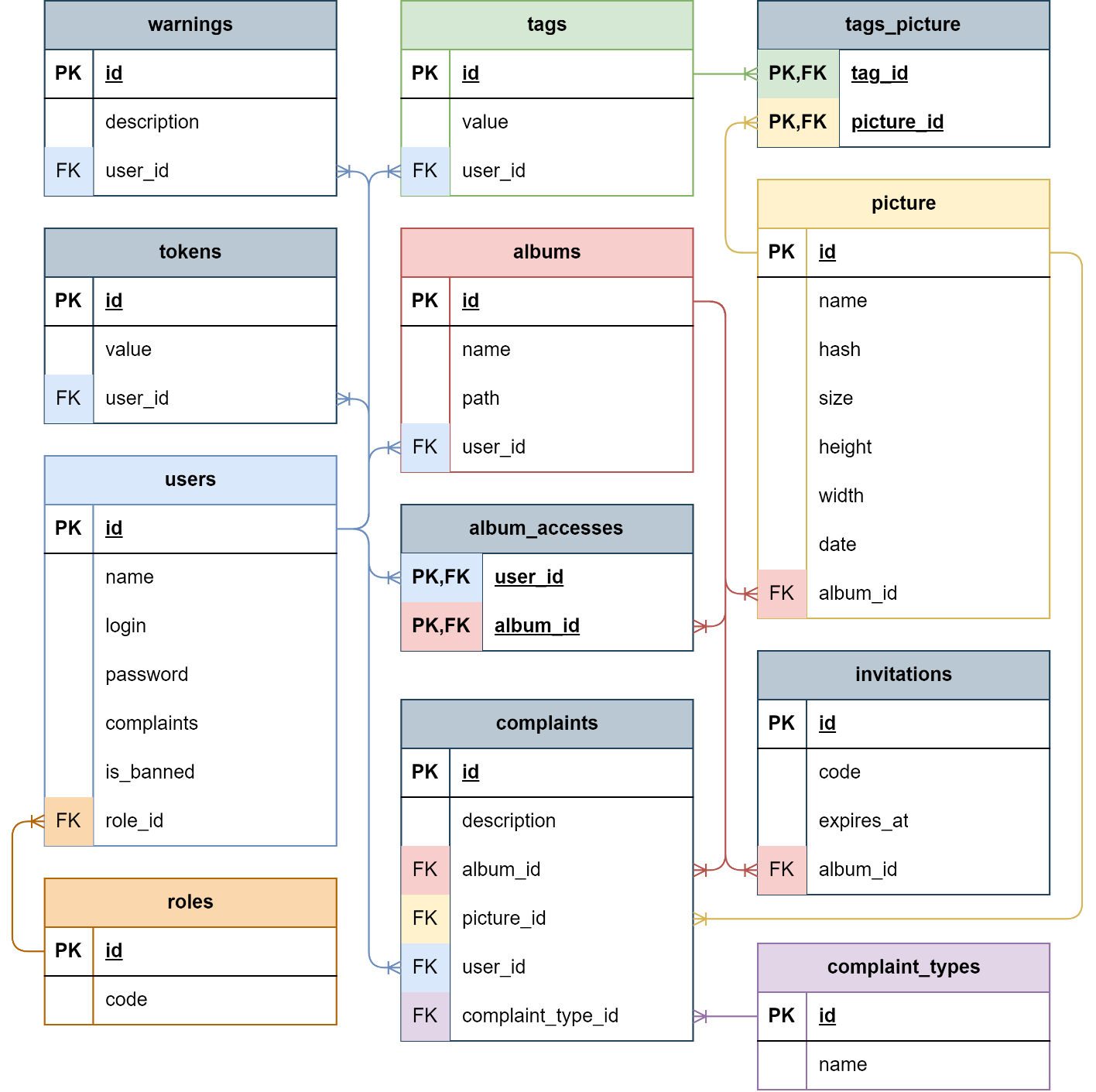


Рисунок 42 – Логическая модель данных

## 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 путем добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имен 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, построим физическую модель данных (см. Рисунок 43):

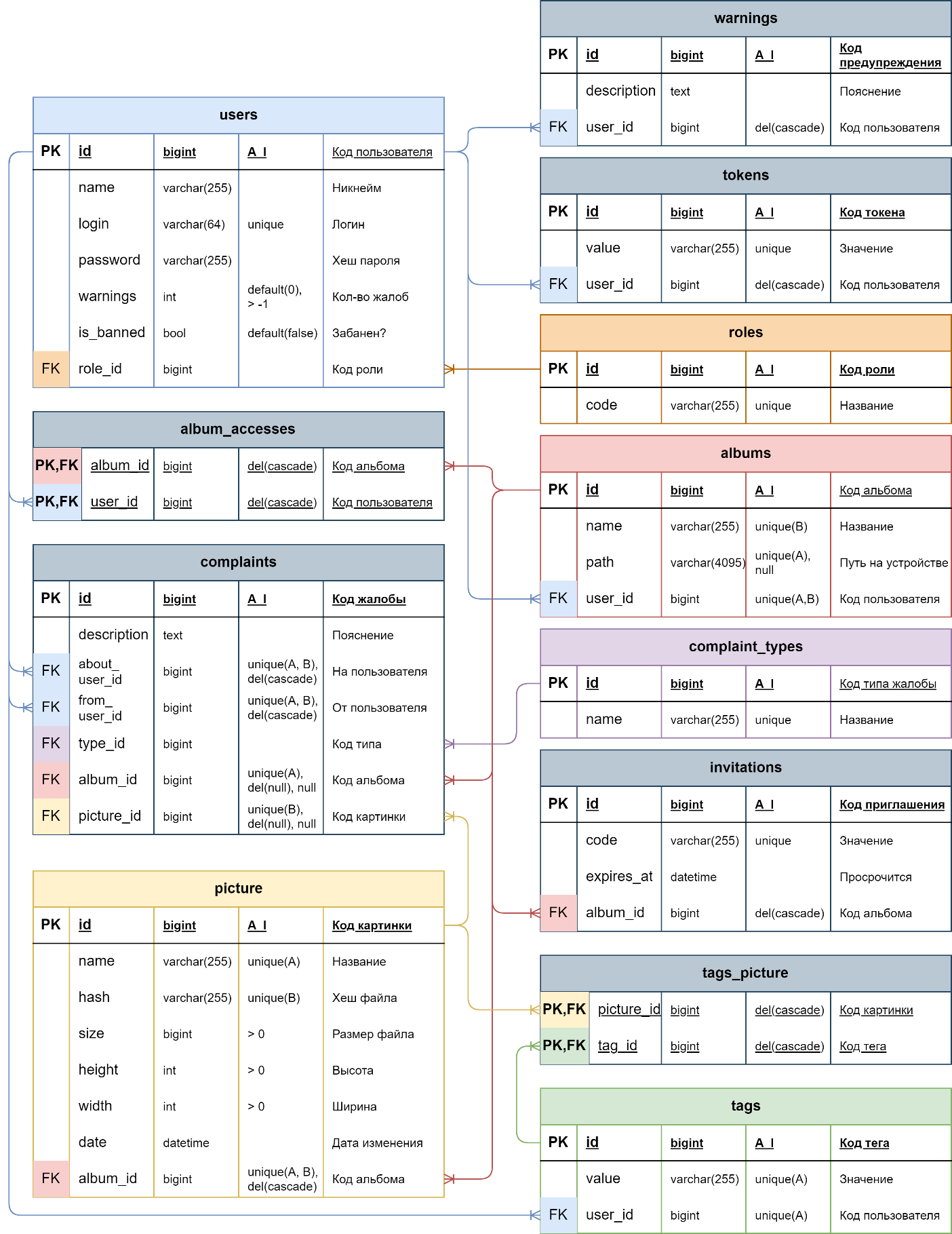


Рисунок 43 – Физическая модель данных

# Глава 3. Разработка серверной части

## Маршрутизация

Все маршруты Laravel должны быть определены в файлах маршрутов, находящихся в нашем каталоге routes. Эти файлы автоматически загружаются нашим поставщиком App\Providers\RouteServiceProvider приложения. Файл routes/web.php определяет маршруты веб-интерфейса. Этим маршрутам назначается группа посредников web, которая обеспечивает такие функции, как состояние сессии и защита от CSRF (Межсайтовая подделка запроса Cross-site request forgery). Маршруты в routes/api.php не сохраняют состояния и им назначается группа посредников API.

Маршруты, определенные в файле routes/api.php, вложены в группу маршрутов в RouteServiceProvider. Внутри этой группы автоматически применяется префикс URI /api, поэтому вам не нужно вручную добавлять его к каждому маршруту в файле маршрутов.

Исходя из технического задания были разработаны следующие маршруты, которые записаны в файле routes\api.php (см. Листинг 1)

Листинг 1 – Маршрутизация по API /routes/api.php

Route  
::controller(AuthController::class)  
->group(function ($auth) { *// [АВТОРИЗАЦИЯ]* $auth->post('register', 'register'); *// Регистрация* $auth->post('login' , 'login'); *// Авторизация* $auth->post('logout' , 'logout') *// Выход (удаление токена)* ->middleware(SanctumAuth::class);  
});  
  
Route  
::middleware(SanctumAuth::class)  
->group(function ($authorized) { *// [АВТОРИЗОВАННЫЕ ФУНКЦИИ]* $authorized  
 ->controller(AlbumController::class)  
 ->prefix('albums')  
 ->group(function ($albums) { *// [АЛЬБОМЫ]* $albums->post('', 'store'); *// Создание ЛИЧНОГО альбома* $albums->get ('', 'index'); *// Просмотр всех ЛИЧНЫХ и ДОСТУПНЫХ ЧУЖИХ альбомов* $albums  
 ->prefix('{album}')  
 ->middleware(CheckAlbumAccess::class)  
 ->group(function ($album) { *// [АЛЬБОМ]* $album->get ('', 'show'); *// Просмотр информации об альбоме* $album->post ('', 'update'); *// Изменение информации об СВОЁМ альбоме* $album->delete('', 'destroy'); *// Удаления СВОЕГО альбома и всё связанное с ним (БД&ФС)* $album->delete('accesses',   
 [AccessController::class, 'destroy']) *// Убрать доступ у СЕБЯ с ЧУЖОГО альбома* ->withoutMiddleware(CheckAlbumAccess::class);  
 $album->delete('accesses/{user}',   
 [AccessController::class, 'destroy']); *// Убрать доступ со СВОЕГО альбома* $album->post ('invite',   
 [InvitationController::class, 'store']); *// Генерировать приглашение на СВОЙ альбом* $album->post ('complaint',   
 [ComplaintController::class, 'storeToAlbum']) *// Создание жалобы на ЧУЖОЙ альбом* ->withoutMiddleware(CheckAlbumAccess::class);  
 $album  
 ->prefix('pictures')  
 ->controller(PictureController::class)  
 ->group(function ($albumPictures) { *// [КАРТИНКИ В АЛЬБОМЕ]* $albumPictures->get ('', 'index'); *// Список картинок в альбоме* $albumPictures->post('', 'store'); *// Загрузка картинок на сервер* $albumPictures  
 ->prefix('{picture}')  
 ->group(function ($picture) { *// [КАРТИНКА]* $picture->get ('', 'info'); *// Получение информации об картинке* $picture->delete('', 'destroy'); *// Удаление СВОЕЙ картинки* $picture->post('complaint',   
 [ComplaintController::class, 'storeToPicture']) *// Жалоба на ЧУЖУЮ картинку* ->withoutMiddleware(CheckAlbumAccess::class);  
 $picture  
 ->withoutMiddleware([SanctumAuth::class, CheckAlbumAccess::class])  
 ->middleware(SignCheck::class)  
 ->group(function ($pictureBySign) { *// [ФАЙЛЫ КАРТИНКИ]* $pictureBySign->get('download' , 'download'); *// Скачивание* $pictureBySign->get('original' , 'original'); *// Отображение* $pictureBySign->get('thumb/{orient}{px}', 'thumbnail') *// Превью* ->name('thumbnail')  
 ->where('orient' , '[qwhQWH]')  
 ->where('px', '[0-9]+')  
 ->withoutMiddleware("throttle:api");  
 });  
 $picture  
 ->prefix('tags')  
 ->controller(TagController::class)  
 ->group(function ($pictureTags) { *// [ТЕГИ НА КАРТИНКЕ]* $pictureTags->post ('{tag}', 'attachToPicture'); *// Прикрепление тега* $pictureTags->delete('{tag}', 'detachToPicture'); *// Открепление тега* });  
 });  
 });  
 });  
 });  
 $authorized  
 ->controller(InvitationController::class)  
 ->prefix('invitation/{invitation}')  
 ->group(function ($invitations) { *// [ПРИГЛАШЕНИЕ]* $invitations->get('album', 'album'); *// Просмотр содержимого альбома по приглашению* $invitations->get('join' , 'join'); *// Присоединиться к альбому (добавление доступа)* $invitations->delete('' , 'destroy'); *// Удалить СВОЙ код приглашения* });  
 $authorized  
 ->controller(AccessController::class)  
 ->prefix('accesses')  
 ->group(function ($accesses) { *// [ДОСТУПЫ]* $accesses->get('', 'index'); *// Список выданных доступов и приглашений* });  
 $authorized  
 ->prefix('tags')  
 ->controller(TagController::class)  
 ->group(function ($tags) { *// [ТЕГИ]* $tags->post('', 'store'); *// Создание ЛИЧНОГО тега* $tags->get ('', 'index'); *// Просмотр ЛИЧНЫХ тегов* $tags  
 ->prefix('{tag}')  
 ->group(function ($tag) { *// [ТЕГ]* $tag->get ('', 'show'); *// Просмотр информации о теге (прикреплённые картинки)* $tag->post ('', 'update'); *// Изменение ЛИЧНОГО тега* $tag->delete('', 'destroy'); *// Удаление ЛИЧНОГО тега* });  
 });  
 $authorized  
 ->prefix('complaints')  
 ->controller(ComplaintController::class)  
 ->group(function ($complaints) { *// [ЖАЛОБЫ]* $complaints  
 ->prefix('types')  
 ->controller(ComplaintTypeController::class)  
 ->middleware(CheckRole::class . ':admin')  
 ->group(function ($complaintTypes) { *// [ТИПЫ ЖАЛОБЫ]* $complaintTypes->post('', 'store'); *// Создание новых типов жалоб* $complaintTypes->get ('', 'index') *// Просмотр всех типов жалоб* ->withoutMiddleware(CheckRole::class . ':admin');  
 $complaintTypes  
 ->prefix('{complaintType}')  
 ->group(function ($complaintType) { *// [ТИП ЖАЛОБЫ]* $complaintType->post ('', 'update'); *// Редактирование типа жалоб* $complaintType->delete('', 'destroy'); *// Удаление типа жалоб* });  
 });  
 $complaints->get('', 'index'); *// Просмотр ВСЕХ жалоб (админ) / СВОИХ жалоб* $complaints  
 ->prefix('{complaint}')  
 ->group(function ($complaint) { *// [ЖАЛОБА]* $complaint->delete('', 'destroy'); *// Удаление СВОЕЙ жалобы* $complaint->post ('', 'updateBatch') *// Изменение жалобы* ->middleware(CheckRole::class . ':admin');  
 });  
 });  
 $authorized  
 ->prefix('users')  
 ->controller(UserController::class)  
 ->group(function ($users) { *// [ПОЛЬЗОВАТЕЛИ]* $users->get ('me', 'showSelf'); *// Получение СЕБЯ* $users->post('me', 'updateSelf'); *// Редактирование СЕБЯ* $users->get ('', 'index') *// Список пользователей* ->middleware(CheckRole::class . ':admin');  
 $users  
 ->prefix('{user}')  
 ->middleware(CheckRole::class . ':admin')  
 ->group(function ($user) { *// [ПОЛЬЗОВАТЕЛЬ]* $user->get ('', 'show'); *// Получение пользователя* $user->post('', 'update'); *// Редактирование пользователя* $user  
 ->prefix('warnings')  
 ->controller(WarningController::class)  
 ->group(function ($warnings) { *// [ПРЕДУПРЕЖДЕНИЯ]* $warnings->post ('' , 'store'); *// Создание предупреждения* $warnings->delete('{warning}', 'destroy'); *// Удаление предупреждения* });  
 });  
 });  
 $authorized  
 ->controller(SettingsController::class)  
 ->prefix('settings')  
 ->middleware(CheckRole::class . ':admin')  
 ->group(function ($settings) { *// [НАСТРОЙКИ]* $settings->get ('', 'index'); *// Получение всех настроек* $settings->post('', 'edit'); *// Изменение настроек* });  
});  
Route::*get*('', [SettingsController::class, 'public'])  
 ->middleware('cache.headers:public;max\_age=2628000;etag'); *// Публичные настройки*

## Миграции

Миграции похожи на контроль версий для вашей базы данных, позволяют вашей команде определять схемы базы данных приложения и совместно использовать их определение.

Фасад Schema обеспечивает независимую от базы данных поддержку для создания и управления таблицами во всех поддерживаемых Laravel системах баз данных. В обычной ситуации, этот фасад используется для создания и изменения таблиц / столбцов базы данных во время миграции. Итоговые миграции представлены в Листинг 2.

Листинг 2 – Миграции базы данных database/migrations/\*.php

Schema::*create*('roles', function (Blueprint $table) {  
 $table->id();  
 $table->string('code')->unique();  
 $table->timestamps();  
});

Schema::*create*('personal\_access\_tokens', function (Blueprint $table) {  
 $table->id();  
 $table->morphs ('tokenable');  
 $table->string ('name');  
 $table->string ('token', 255)->unique();  
 $table->text ('abilities')->nullable();  
 $table->timestamp('last\_used\_at')->nullable();  
 $table->timestamp('expires\_at')->nullable();  
 $table->timestamps();  
});

Schema::*create*('users', function (Blueprint $table) {  
 $table->id();  
 $table->string ('name');  
 $table->string ('login', 64)->unique();  
 $table->string ('password');  
 $table->boolean ('is\_banned')->default(false);  
 $table->foreignId('role\_id')->constrained();  
 $table->rememberToken()->unique();  
 $table->timestamps();  
});

Schema::*create*('albums', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->string('path')->nullable();  
 $table->foreignId('user\_id')->constrained();  
 $table->timestamps();  
  
 $table->unique(['user\_id', 'name']);  
 $table->unique(['user\_id', 'path']);  
});

Schema::*create*('album\_accesses', function (Blueprint $table) {  
 $table->primary(['album\_id', 'user\_id']);  
 $table->foreignId('album\_id')->constrained()->cascadeOnDelete();  
 $table->foreignId('user\_id' )->constrained()->cascadeOnDelete();  
 $table->timestamps();  
});

Schema::*create*('pictures', function (Blueprint $table) {  
 $table->id();  
 $table->string ('name');  
 $table->string ('hash');  
 $table->dateTime ('date');  
 $table->bigInteger('size');  
 $table->integer ('width');  
 $table->integer ('height');  
 $table->foreignId ('album\_id')->constrained()->cascadeOnDelete();  
 $table->timestamps();  
  
 $table->unique(['album\_id', 'name']);  
 $table->unique(['album\_id', 'hash']);  
});

Schema::*create*('tags', function (Blueprint $table) {  
 $table->id();  
 $table->string ('value');  
 $table->foreignId('user\_id')->constrained();  
 $table->timestamps();  
  
 $table->unique(['value', 'user\_id']);  
});  
Schema::*create*('tag\_pictures', function (Blueprint $table) {  
 $table->primary(['tag\_id', 'picture\_id']);  
 $table->foreignId('tag\_id' )->constrained()->cascadeOnDelete();  
 $table->foreignId('picture\_id')->constrained()->cascadeOnDelete();  
 $table->timestamps();  
});

Schema::*create*('invitations', function (Blueprint $table) {  
 $table->id();  
 $table->foreignId('album\_id')->constrained()->cascadeOnDelete();  
 $table->string ('link')->unique();  
 $table->integer ('join\_limit')->nullable();  
 $table->dateTime ('expires\_at')->nullable();  
 $table->timestamps();  
});

Schema::*create*('complaint\_types', function (Blueprint $table) {  
 $table->id();  
 $table->string('name')->unique();  
 $table->timestamps();  
});

Schema::*create*('complaints', function (Blueprint $table) {  
 $table->id();  
 $table->string ('description')->nullable();  
 $table->boolean ('status')->nullable()->default(null);  
 $table->foreignId('album\_id' )->nullable()->constrained()->nullOnDelete();  
 $table->foreignId('picture\_id')->nullable()->constrained()->nullOnDelete();  
 $table->foreignId('from\_user\_id' )->constrained('users')->cascadeOnDelete();  
 $table->foreignId('about\_user\_id')->constrained('users')->cascadeOnDelete();  
 $table->foreignId('complaint\_type\_id')->constrained('complaint\_types');  
 $table->timestamps();  
});

Schema::*create*('warnings', function (Blueprint $table) {  
 $table->id();  
 $table->foreignId('user\_id')->constrained();  
 $table->text ('comment');  
 $table->timestamps();  
});

## Заполнители

Заполнители (Seeders) в Laravel используются для заполнения базы данных начальными данными. Они позволяют автоматически добавлять тестовые или стандартные данные в таблицы, что упрощает процесс разработки и тестирования. Заполнители особенно полезны при работе с командами, так как обеспечивают согласованность данных на всех этапах разработки.

Фасад DB и модель Eloquent предоставляют удобные методы для вставки данных в таблицы. Заполнители обычно создаются в папке database/seeders, а основной класс для запуска всех заполнителей находится в файле DatabaseSeeder.php.

В листинге 3 представлен общий обязательный заполнитель, который находится в файле database/seeders/DatabaseSeeder.php. Этот файл используется для вызова других заполнителей и управления процессом заполнения базы данных.

Листинг 3 – Общий обязательный заполнитель database/seeders/DatabaseSeeder.php

class DatabaseSeeder extends Seeder  
{  
 public function run(): void  
 {  
 $roleAdminId = Role::*firstOrCreate*(['code' => 'admin'])->id;  
 $roleUserId = Role::*firstOrCreate*(['code' => 'user' ])->id;  
 $admin = User  
 ::*where*('login', 'admin')  
 ->first();  
 if ($admin) {  
 $changePass = $this->command->confirm('Do you want change password for admin?', true);  
 if ($changePass) $admin->update([  
 'password' => $this->askOrGenPassword(),  
 'role\_id' => $roleAdminId  
 ]);  
 }  
 else User::*create*([  
 'name' => 'Administrator',  
 'login' => 'admin',  
 'password' => $this->askOrGenPassword(),  
 'role\_id' => $roleAdminId,  
 ]);  
  
 ComplaintType::*firstOrCreate*(['name' => 'Нарушение авторских прав']);  
 ComplaintType::*firstOrCreate*(['name' => 'Шок-контент']);  
 }  
  
 private function askOrGenPassword(): string  
 {  
 $password = $this->command->secret('Enter the admin password (leave blank to generate a random password)');  
  
 if (empty($password)) {  
 $password = Str::*password*(16);  
  
 $choice = $this->command->choice(  
 'How do you want to handle the generated password?',  
 ['Output to screen', 'Save to log', 'Both'],  
 0  
 );  
  
 if ($choice === 'Save to log' || $choice === 'Both')  
 Log::*info*("Admin user created with password: $password");  
  
 if ($choice === 'Output to screen' || $choice === 'Both')  
 $this->command->info("Admin password: $password");  
 }  
 return $password;  
 }  
}

В листинге 4 представлен заполнитель для тестирования, который находится в файлеdatabase/seeders/TestSeeder.php. Этот заполнитель создаёт тестовых пользователей, альбомы, картинки и жалобы для тестирования функционала приложения.

Листинг 4 – Заполнитель для тестирования database/seeders/TestSeeder.php

class TestSeeder extends Seeder  
{public function run(): void  
 {  
 $roleAdminId = Role::*firstOrCreate*(['code' => 'admin'])->id;  
 $roleUserId = Role::*firstOrCreate*(['code' => 'user' ])->id;  
 User::*firstOrCreate*(['login' => 'admin'],['name' => 'Админ', 'login' => 'admin', 'password' => 'Admin123!', 'is\_banned' => 0, 'role\_id' => $roleAdminId]);  
 User::*firstOrCreate*(['login' => 'test1'],['name' => 'Чел 1', 'login' => 'test1', 'password' => 'Test123!', 'is\_banned' => 0, 'role\_id' => $roleUserId]);  
 User::*firstOrCreate*(['login' => 'test2'],['name' => 'Чел 2', 'login' => 'test2', 'password' => 'Test123!', 'is\_banned' => 0, 'role\_id' => $roleUserId]);  
 User::*firstOrCreate*(['login' => 'test3'],['name' => 'Чел 3', 'login' => 'test3', 'password' => 'Test123!', 'is\_banned' => 0, 'role\_id' => $roleUserId]);  
 Album::*firstOrCreate*(['name' => 'Яблоко'],['name' =>"Яблоко", 'path' => fake()->slug(), 'user\_id' => User::*inRandomOrder*()->whereNot('role\_id', $roleAdminId)->first()->id]);  
 Album::*firstOrCreate*(['name' => 'Помидор'],['name'=>"Помидор", 'path' => fake()->slug(), 'user\_id' => User::*inRandomOrder*()->whereNot('role\_id', $roleAdminId)->first()->id]);  
 Album::*firstOrCreate*(['name' => 'Огурец'],['name' =>"Огурец", 'path' => fake()->slug(), 'user\_id' => User::*inRandomOrder*()->whereNot('role\_id', $roleAdminId)->first()->id]);  
 AlbumAccess::*firstOrCreate*(['album\_id' => 1, 'user\_id' => 3]);  
 AlbumAccess::*firstOrCreate*(['album\_id' => 2, 'user\_id' => 4]);  
 AlbumAccess::*firstOrCreate*(['album\_id' => 1, 'user\_id' => 4]);  
 AlbumAccess::*firstOrCreate*(['album\_id' => 3, 'user\_id' => 2]);  
 ComplaintType::*firstOrCreate*(['name' => 'Террористическая пропаганда']);  
 ComplaintType::*firstOrCreate*(['name' => 'Сцены насилия']);  
 ComplaintType::*firstOrCreate*(['name' => 'Разжигание ненависти']);  
 ComplaintType::*firstOrCreate*(['name' => 'Пропаганда насилия']);  
 ComplaintType::*firstOrCreate*(['name' => 'Нарушение авторских прав']);  
 ComplaintType::*firstOrCreate*(['name' => 'Шок-контент']);  
 *// Путь к существующим картинкам* $parameter = $this->command->ask('Введите абсолютный путь до папки с картинками', Storage::*path*('sample'));  
 $sourceDirectory = $parameter;  
 *// Получение всех файлов из директории* $files = File::*files*($sourceDirectory);  
 foreach ($files as $file) {  
 *// Ищем случайного пользователя и альбом* $album = Album::*inRandomOrder*()->first();  
 $user = $album->user;  
 *// Путь назначения* $destinationDirectory = $album->getPath();  
 *// Создаем директорию назначения, если её нет* if (!Storage::*exists*($destinationDirectory)) {  
 Storage::*makeDirectory*($destinationDirectory);  
 }  
 *// Имя файла* $fileName = $file->getFilename();  
 $filePath = Storage::*path*("$destinationDirectory/$fileName");  
 *// Копируем файл* File::*copy*($file->getRealPath(), $filePath);  
 *// Добавляем запись в базу данных* DB::*table*('pictures')->insertOrIgnore([  
 'name' => $fileName,  
 'hash' => md5\_file($filePath),  
 'date' => now(),  
 'size' => File::*size*($filePath),  
 'width' => getimagesize($filePath)[0] ?? 0, *// Проверка на null для ширины* 'height' => getimagesize($filePath)[1] ?? 0, *// Проверка на null для высоты* 'album\_id' => $album->id,  
 'created\_at' => now(),  
 'updated\_at' => now(),  
 ]);  
 }  
  
 for($i=0;$i<=60;$i++)  
 {  
 *// Получение случайного альбома пользователя* $album = Album::*inRandomOrder*()->first();  
 $user = $album->user;  
 *// Получение случайной картинки из альбома* $picture = Picture::*where*('album\_id', $album->id)->inRandomOrder()->first();  
 if($picture)  
 {  
 *// Создание жалобы* DB::*table*('complaints')->insertOrIgnore([  
 'description' => 'This is a test complaint.',  
 'complaint\_type\_id' => ComplaintType::*inRandomOrder*()->first()->id,  
 'picture\_id' => $picture->id,  
 'album\_id' => $album->id,  
 'about\_user\_id' => $user->id,  
 'from\_user\_id' => User::*where*('role\_id', $roleUserId)->inRandomOrder()->first()->id, *// случайный пользователь, кроме создателя,* 'status' => null,  
 ]);  
 }  
 }  
  
 }  
}

## Конфиги

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/settings.php, который содержит настройки сервера. В нем определены допустимые форматы загружаемых файлов (allowed\_upload\_mimes), размеры превью (allowed\_preview\_sizes), лимиты хранения (free\_storage\_limit), а также параметры, регулирующие ограничения загрузки (upload\_disable\_percentage). Все значения загружаются из переменных окружения и могут быть изменены без редактирования кода приложения.

Листинг 5 – Конфиг настроек сервера config/settings.php

return [  
 'allowed\_upload\_mimes' => explode(',', env('ALLOWED\_UPLOAD\_MIMES', 'jpeg,jpg,png,gif')),  
 'allowed\_preview\_sizes' => array\_map('intval',  
 explode(',', env('ALLOWED\_PREVIEW\_SIZES', '144,240,360,480,720,1080'))  
 ),  
 'warning\_limit\_for\_ban' => (int) env('WARNING\_LIMIT\_FOR\_BAN', 3),  
 'free\_storage\_limit' => (int) env('FREE\_STORAGE\_LIMIT', 1024 \* 1024 \* 1024 \* 5), *// 5 GiB* 'upload\_disable\_percentage' => (int) env('UPLOAD\_DISABLE\_PERCENTAGE', 90),  
];

## 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.

В листинге 6 представлена модель Album, расположенная в файле app/Models/Album.php. Она управляет альбомами пользователей и включает:

1. Определение полей (fillable), которые можно заполнять массово (name, path, user\_id).
2. Методы для работы с путями (getPathStatic, getPath), которые позволяют получать путь к альбому.
3. Методы для работы с подписанными токенами (signNonHash, signCacheKey, getSign), обеспечивающие безопасность доступа к альбомам через кэширование и шифрование.
4. Связи с другими моделями, такие как:

* user() — связь «многие к одному» с пользователем (User).
* usersViaAccess() — связь «многие ко многим» с пользователями, имеющими доступ к альбому (album\_accesses).
* pictures() — связь «один ко многим» с картинками (Picture).
* albumAccesses(), invitations(), complaints() — связи с другими сущностями, относящимися к альбому.

Благодаря Eloquent, модель Album позволяет легко взаимодействовать с базой данных, управлять правами доступа и организовывать хранение файлов в альбомах.

Листинг 6 – Модель альбома app/Models/Album.php

class Album extends Model  
{  
 *// Поля*  
 protected $fillable = [  
 'name',  
 'path',  
 'user\_id',  
 ];  
  
 *// Функции*  
 public static function getPathStatic($userId, $albumId): string {  
 return "albums/$userId/$albumId";  
 }  
 public function getPath(): string {  
 return $this->getPathStatic($this->user\_id, $this->id);  
 }  
  
 public static function signNonHash(User $user, $albumId): string {  
 $currentDay = date("Y-m-d");  
 $userToken = $user->tokens[0]->token;  
 return $userToken . $currentDay . $albumId;  
 }  
 public static function signCacheKey($albumId, $userId): string {  
 return "signAccess:to=$albumId;for=$userId";  
 }  
 public function getSign(User $user): string {  
 $cacheKey = static::*signCacheKey*($this->id, $user->id);  
 $cachedSign = Cache::*get*($cacheKey);  
 if ($cachedSign)  
 return $user->id .'\_'. $cachedSign;  
  
 $string = static::*signNonHash*($user, $this->id);  
 $signCode = base64\_encode(Hash::*make*($string));  
  
 Cache::*put*($cacheKey, $signCode . '\_' . $this->user\_id, 3600);  
 return $user->id .'\_'. $signCode;  
 }  
  
 *// Связи* public function user() {  
 return $this->belongsTo(User::class);  
 }  
 public function usersViaAccess() {  
 return $this->belongsToMany(User::class, 'album\_accesses')->using(AlbumAccess::class);  
 }  
 public function pictures() {  
 return $this->hasMany(Picture::class);  
 }  
 public function albumAccesses() {  
 return $this->hasMany(AlbumAccess::class);  
 }  
 public function invitations() {  
 return $this->hasMany(Invitation::class);  
 }  
 public function complaints() {  
 return $this->hasMany(Complaint::class);  
 }  
}

В листинге 7 представлена модель AlbumAccess, расположенная в app/Models/AlbumAccess.php. Она управляет таблицей album\_accesses, отвечая за права доступа к альбомам. Модель содержит заполняемые поля album\_id, user\_id и связи: album() — принадлежит альбому Album, user() — принадлежит пользователю User. Эта модель используется как промежуточная таблица для управления доступом пользователей к альбомам.

Листинг 7 – Модель грантов доступов app/Models/AlbumAccess.php

class AlbumAccess extends Pivot  
{  
 protected $table = 'album\_accesses';  
 *// Поля* protected $fillable = [  
 'album\_id',  
 'user\_id',  
 ];  
 *// Связи* public function album() {  
 return $this->belongsTo(Album::class);  
 }  
 public function user() {  
 return $this->belongsTo(User::class);  
 }  
}

В листинге 8 представлена модель Complaint, расположенная в app/Models/Complaint.php. Она управляет жалобами и содержит заполняемые поля description, complaint\_type\_id, picture\_id, album\_id, about\_user\_id, from\_user\_id, status, а также связи: type() — тип жалобы (ComplaintType), picture() — изображение (Picture), album() — альбом (Album), fromUser() — отправитель (User), aboutUser() — объект жалобы (User).

Листинг 8 – Модель жалобы app/Models/Complaint.php

class Complaint extends Model  
{  
 *// Поля* protected $fillable = [  
 'description',  
 'complaint\_type\_id',  
 'picture\_id',  
 'album\_id',  
 'about\_user\_id',  
 'from\_user\_id',  
 'status',  
 ];  
 *// Связи* public function type() {  
 return $this->belongsTo(ComplaintType::class, 'complaint\_type\_id');  
 }  
 public function picture() {  
 return $this->belongsTo(Picture::class);  
 }  
 public function album() {  
 return $this->belongsTo(Album::class);  
 }  
 public function fromUser() {  
 return $this->belongsTo(User::class, 'from\_user\_id');  
 }  
 public function aboutUser() {  
 return $this->belongsTo(User::class, 'about\_user\_id');  
 }  
}

В листинге 9 представлена модель ComplaintType, расположенная в app/Models/ComplaintType.php. Она управляет типами жалоб, содержит заполняемое поле name и связь complaints() для получения всех связанных жалоб (Complaint).

Листинг 9 – Модель типа жалобы app/Models/ComplaintType.php

class ComplaintType extends Model  
{  
 protected $fillable = ['name'];  
  
 public function complaints() {  
 return $this->hasMany(Complaint::class);  
 }  
}

В листинге 10 представлена модель Invitation, расположенная в app/Models/Invitation.php. Она управляет приглашениями и содержит заполняемые поля code, expires\_at, album\_id, join\_limit, а также методы getRouteKeyName() для задания ключа маршрута и failOnExpires() для проверки срока действия приглашения. Модель также включает связь album(), которая связывает приглашение с альбомом (Album).

Листинг 10 – Модель приглашения app/Models/Invitation.php

class Invitation extends Model  
{  
 *// Поля* protected $fillable = [  
 'code',  
 'expires\_at',  
 'album\_id',  
 'join\_limit',  
 ];  
 *// Функции* public function getRouteKeyName() {  
 return 'link';  
 }  
  
 public function failOnExpires(): void  
 {  
 if ((  
 $this->expires\_at === null ||  
 !now()->greaterThan($this->expires\_at)  
 ) && (  
 $this->join\_limit === null ||  
 $this->join\_limit > 0  
 )) return;  
  
 $this->delete();  
 throw new ApiException('Invitation expired', 409);  
 }  
 *// Связи* public function album() {  
 return $this->belongsTo(Album::class);  
 }  
}

В листинге 11 представлена модель Picture, расположенная в app/Models/Picture.php. Она управляет изображениями и содержит заполняемые поля name, hash, date, size, width, height, album\_id, а также методы для получения путей к картинке и миниатюрам (getPathStatic, getPath, getPathThumbStatic, getPathThumb), а также метод для разрешения маршрута resolveRouteBinding. Модель включает связи с альбомом (album()), тегами через промежуточную таблицу (tagPictures(), tags()), и жалобами (complaints()).

Листинг 11 – Модель картинки app/Models/Picture.php

class Picture extends Model  
{  
 *// Поля*  
 protected $fillable = [  
 'name',  
 'hash',  
 'date',  
 'size',  
 'width',  
 'height',  
 'album\_id',  
 ];  
 *// Функции*  
 public static function getPathStatic($userId, $albumId, $name = null): string {  
 return Album::*getPathStatic*($userId, $albumId) . "/" . $name ?? "";  
 }  
 public function getPath($userId): string {  
 return $this->getPathStatic($userId, $this->album\_id, $this->name);  
 }  
 public static function getPathThumbStatic($userId, $albumId, $pictureId = null, $orientation = '', $size = ''): string {  
 return Album::*getPathStatic*($userId, $albumId) . "/thumbs/" . ($pictureId ? "$pictureId-$orientation$size.jpg" : '');  
 }  
 public function getPathThumb($userId, $orientation, $size): string {  
 return $this->getPathThumbStatic($userId, $this->album\_id, $this->id, $orientation, $size);  
 }  
 public function resolveRouteBinding($value, $field = null) {  
 $album = request()->route('album') ?? 'NULL';  
 if ($album instanceof Album) $album = $album->id;  
 return $this  
 ->where('album\_id', $album)  
 ->where($field ?? 'id', $value)  
 ->firstOrFailCustom();  
 }  
 *// Связи*  
 public function album() {  
 return $this->belongsTo(Album::class);  
 }  
 public function tagPictures() {  
 return $this->hasMany(TagPicture::class);  
 }  
 public function tags() {  
 return $this->belongsToMany(Tag::class, TagPicture::class)->using(TagPicture::class);  
 }  
 public function complaints() {  
 return $this->hasMany(Complaint::class);  
 }  
}

В листинге 12 представлена модель Role, расположенная в app/Models/Role.php. Она управляет ролями пользователей и содержит заполняемое поле code, а также связь users(), которая возвращает всех пользователей, связанных с данной ролью.

Листинг 12 – Модель роли app/Models/Role.php

class Role extends Model  
{  
 protected $fillable = ['code'];  
  
 public function users() {  
 return $this->hasMany(User::class);  
 }  
}

В листинге 13 представлена модель Tag, расположенная в app/Models/Tag.php. Она управляет тегами и содержит заполняемые поля value, user\_id, а также связи с пользователем (user()), изображениями через промежуточную таблицу (tagPictures(), pictures()).

Листинг 13 – Модель тега app/Models/Tag.php

class Tag extends Model  
{  
 *// Поля* protected $fillable = [  
 'value',  
 'user\_id',  
 ];  
 *// Связи* public function user() {  
 return $this->belongsTo(User::class);  
 }  
 public function tagPictures() {  
 return $this->hasMany(TagPicture::class);  
 }  
 public function pictures() {  
 return $this->belongsToMany(Picture::class, TagPicture::class)->using(TagPicture::class);  
 }  
}

В листинге 14 представлена модель TagPicture, расположенная в app/Models/TagPicture.php. Она управляет связью между тегами и картинками в таблице tag\_pictures и содержит заполняемые поля tag\_id, picture\_id. Модель включает связи с тегом (tag()) и картинкой (picture()).

Листинг 14 – Модель связи тега и картинки app/Models/TagPicture.php

class TagPicture extends Pivot  
{  
 protected $table = 'tag\_pictures';  
  
 *// Поля* protected $fillable = [  
 'tag\_id',  
 'picture\_id',  
 ];  
 *// Связи* public function tag() {  
 return $this->belongsTo(Tag::class);  
 }  
 public function picture() {  
 return $this->belongsTo(Picture::class);  
 }  
}

В листинге 15 представлена модель User, расположенная в app/Models/User.php. Эта модель управляет пользователями и реализует интерфейсы аутентификации, авторизации и сброса пароля. Она содержит заполняемые поля name, login, password, role\_id, is\_banned, а также скрытые поля password, remember\_token. Модель включает функции для блокировки пользователя (ban()), кастомизацию полей и их типов через метод casts(). Связи пользователя включают роль (role()), альбомы (albums()), картинки через альбомы (pictures()), жалобы (complaintsFrom, complaintsAbout), доступы к альбомам (albumAccesses, albumsViaAccess), теги (tags) и предупреждения (warnings).

Листинг 15 – Модель пользователя app/Models/User.php

class User extends Model implements  
 *AuthenticatableContract*,  
 *AuthorizableContract*,  
 *CanResetPasswordContract*{  
 use HasFactory, Notifiable, HasApiTokens, Authenticatable, Authorizable, CanResetPassword, MustVerifyEmail;  
  
 *// Поля* protected $fillable = [  
 'name',  
 'login',  
 'password',  
 'role\_id',  
 'is\_banned',  
 ];  
  
 *// Скрытые* protected $hidden = [  
 'password',  
 'remember\_token',  
 ];  
  
 *// Установка по умолчанию* protected $attributes = [  
 'is\_banned' => false,  
 ];  
  
 *// Модификация полей* protected function casts(): array  
 {  
 return [  
 'password' => 'hashed',  
 'is\_banned' => 'boolean'  
 ];  
 }  
  
 *// Функции* public function ban()  
 {  
 AlbumAccess::*whereIn*('album\_id', $this->albums()->pluck('id'))->delete();  
 Complaint::*where*('about\_user\_id', $this->id)->update(['status' => 1]);  
 $this->tokens()->delete();  
 }  
   
 *// Связи* public function role() {  
 return $this->belongsTo(Role::class);  
 }  
 public function albums() {  
 return $this->hasMany(Album::class);  
 }  
 public function pictures() {  
 return $this->hasManyThrough(  
 Picture::class,  
 Album::class,  
 'user\_id',  
 'album\_id'  
 );  
 }  
 public function complaintsFrom() {  
 return $this->hasMany(Complaint::class, 'from\_user\_id');  
 }  
 public function complaintsAbout() {  
 return $this->hasMany(Complaint::class, 'about\_user\_id');  
 }  
 public function albumAccesses() {  
 return $this->hasMany(AlbumAccess::class);  
 }  
 public function albumsViaAccess() {  
 return $this->belongsToMany(Album::class, AlbumAccess::class)  
 ->using(AlbumAccess::class);  
 }  
 public function tags() {  
 return $this->hasMany(Tag::class);  
 }  
 public function warnings() {  
 return $this->hasMany(Warning::class);  
 }  
}

В листинге 16 представлена модель Warning, расположенная в app/Models/Warning.php. Эта модель управляет предупреждениями пользователей и содержит заполняемые поля comment и user\_id. Связь модели с пользователем реализована через метод user(), который возвращает связь "многие к одному" с моделью User.

Листинг 16 – Модель предупреждения app/Models/Warning.php

class Warning extends Model  
{  
 protected $fillable = [  
 'comment',  
 'user\_id',  
 ];  
  
 public function user() {  
 return $this->belongsTo(User::class);  
 }  
}

## 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, DELETE.

В листинге 17 представлен контроллер AccessController, расположенный в app/Http/Controllers/Api/AccessController.php. Этот контроллер предоставляет два метода:

1. index() получает все альбомы пользователя, к которым он имеет доступ, и возвращает их в виде JSON-ответа. Для этого используется метод with() для подгрузки связанных моделей invitations и usersViaAccess, а затем выполняется фильтрация альбомов, которые имеют связанные приглашения или пользователей с доступом. Ответ возвращается в формате JSON с помощью AlbumResource::collection().
2. destroy(Album $album, User $user = null) удаляет доступ пользователя к альбому. Если пользователь не указан, то используется текущий аутентифицированный пользователь. Если пользователь является владельцем альбома, то выбрасывается исключение ApiException. В противном случае происходит удаление связи пользователя с альбомом через метод detach(), и возвращается ответ с кодом 204, который означает успешное удаление без возвращаемого содержимого.

Листинг 17 – Контроллер доступа app/Http/Controllers/Api/AccessController.php

class AccessController extends Controller  
{  
 public function index(): JsonResponse  
 {  
 $user = Auth::*user*();  
 $albumsOwned = $user->albums()->with(['invitations', 'usersViaAccess'])->get();  
 $albums = $albumsOwned->filter(fn($album) =>  
 $album->invitations ->isNotEmpty() ||  
 $album->usersViaAccess->isNotEmpty()  
 );  
 return response()->json(['albums' => AlbumResource::*collection*($albums)]);  
 }  
  
 public function destroy(Album $album, User $user = null): JsonResponse  
 {  
 if ($user === null)  
 $user = Auth::*user*();  
  
 if ($user->id === $album->user\_id)  
 throw new ApiException('You are owner', 409);$album->usersViaAccess()->detach($user->id);  
 return response()->json(null, 204);  
 }  
}

В листинге 18 представлен контроллер AlbumController, который отвечает за обработку запросов, связанных с альбомами. В нем реализованы методы для различных операций с альбомами, таких как создание, получение, обновление и удаление. Рассмотрим подробнее каждый из методов контроллера:

1. index(): Этот метод получает все альбомы пользователя, как его собственные, так и те, к которым он имеет доступ. Он включает подгрузку количества картинок, приглашений и пользователей с доступом для каждого альбома. Также для каждого альбома подгружаются первые четыре картинки, отсортированные по дате. Возвращается JSON-ответ с коллекцией альбомов, разделённых на два типа: собственные и доступные.
2. store(AlbumCreateRequest $request): Метод для создания нового альбома. Он проверяет, существует ли альбом с таким же путем и именем. Если альбом с таким путем уже существует, возвращается ошибка с сообщением. Если альбом с таким именем уже есть, то имя альбома модифицируется, добавляя счётчик. Далее создается новый альбом и соответствующая директория. После этого возвращается информация о созданном альбоме.
3. show(Album $album): Этот метод показывает подробную информацию об альбоме. Для обычного пользователя подгружаются связанные данные, такие как картины и информация о пользователе. Для администратора добавляется информация о жалобах на альбом, а для владельца альбома — информация о пользователях с доступом и приглашениях.
4. update(AlbumUpdateRequest $request, Album $album): Метод для обновления альбома. Он принимает данные запроса и обновляет альбом с новыми значениями, возвращая обновлённый альбом в JSON-формате.
5. destroy(Album $album): Метод для удаления альбома. Он удаляет директорию, связанную с альбомом, и сам альбом из базы данных, после чего возвращается сообщение об успешном удалении.

Листинг 18 – Контроллер альбомов app/Http/Controllers/Api/AlbumController.php

class AlbumController extends Controller  
{  
 public function index(): JsonResponse  
 {  
 $user = Auth::*user*();  
 return response()->json([  
 'own' => AlbumResource::*collection*(  
 $user  
 ->albums()  
 ->withCount([  
 'pictures',  
 'invitations',  
 'usersViaAccess',  
 ])  
 ->with([  
 'pictures' => fn ($query) => $query->orderBy('date', 'DESC')->limit(4),  
 ])  
 ->get()  
 ),  
 'accessible' => AlbumResource::*collection*(  
 $user  
 ->albumsViaAccess()  
 ->withCount([  
 'pictures',  
 ])  
 ->with([  
 'pictures' => fn ($query) => $query->orderBy('date', 'DESC')->limit(4),  
 'user',  
 ])  
 ->get()  
 ),  
 ]);  
 }  
  
 public function store(AlbumCreateRequest $request): JsonResponse  
 {  
 $user = Auth::*user*();  
 $name = $request->input('name');  
 $inputPath = $request->input('path');  
  
 *// Проверка, существует ли альбом с таким путём и возвращаем его если да* if ($inputPath) {  
 $existAlbum = Album  
 ::*where*('path', $inputPath)  
 ->where('user\_id', $user->id)  
 ->first();  
 if ($existAlbum)  
 return response()->json([  
 'message' => 'Album with this path already exists',  
 'album' => AlbumResource::*make*($existAlbum)  
 ], 409);  
 }  
  
 *// Проверка, существует ли альбом с таким именем и модифицируем имя счётчиком если да* $nameCounter = 1;  
 $originalName = $name;  
 while (Album  
 ::*where*('name', $name)  
 ->where('user\_id', $user->id)  
 ->exists()  
 ) $name = "$originalName (". ++$nameCounter .')';  
  
 *// Создание нового альбома с уникальным именем* $newAlbum = Album::*create*([  
 'name' => $name,  
 'path' => $inputPath,  
 'user\_id' => $user->id,  
 ]);  
  
 *// Создание пустой директории* $path = $newAlbum->getPath();  
 Storage::*deleteDirectory*($path);  
 Storage:: *makeDirectory*($path);  
  
 *// Формирование ответа* return response()->json(['album' => AlbumResource::*make*($newAlbum)], 201);  
 }  
  
 public function show(Album $album): JsonResponse  
 {  
 $user = Auth::*user*();  
 $counts[] = 'pictures';  
 if($user->role->code === 'admin')  
 {  
 $relations = ['user'];  
 $counts[] = 'complaints';  
 }  
 else if ($album->user\_id === $user?->id)  
 $relations = ['invitations', 'usersViaAccess'];  
 else  
 $relations = ['user'];  
  
 $album->load([  
 'pictures' => fn ($query) => $query->limit(4),  
 ...$relations,  
 ])->loadCount($counts);  
 return response()->json(['album' => AlbumResource::*make*($album)]);  
 }  
  
 public function update(AlbumUpdateRequest $request, Album $album): JsonResponse  
 {  
 $album->update($request->validated());  
 return response()->json(['album' => AlbumResource::*make*($album)]);  
 }  
  
 public function destroy(Album $album): JsonResponse  
 {  
 Storage::*deleteDirectory*($album->getPath());  
 $album->delete();  
 return response()->json(['message' => 'Album deleted']);  
 }  
}

В Листинге 19 представлен контроллер авторизации (AuthController), который включает три метода для работы с аутентификацией пользователей:

1. login() — выполняет вход пользователя, проверяя его данные (логин и пароль). В случае успеха возвращается токен авторизации и данные пользователя. Если пользователь заблокирован, выбрасывается ошибка.
2. register() — выполняет регистрацию нового пользователя с присваиванием ему роли "user" по умолчанию. После создания пользователя возвращается токен и данные пользователя.
3. logout() — удаляет токен текущего пользователя, тем самым завершив его сессию.

Листинг 19 – Контроллер авторизации app/Http/Controllers/Api/AuthController.php

class AuthController extends Controller  
{  
 public function login(LoginRequest $request): JsonResponse  
 {  
 if (!Auth::*attempt*($request->only('login', 'password')))  
 throw new ApiException('Invalid credentials', 401);  
 $user = Auth::*user*();  
 if($request->is\_banned == 1)  
 {  
 throw new ApiException('Banned', 403);  
 }  
 $user->load('role');  
  
 $token = $user->createToken('api\_token')->plainTextToken;  
 return response()->json([  
 'token' => $token,  
 'user' => UserResource::*make*($user),  
 ]);  
 }  
  
 public function register(RegisterRequest $request): JsonResponse  
 {  
 $roleId = Role::*firstOrCreate*(['code' =>'user'])->id;  
 $user = User::*create*([  
 ...$request->validated(),  
 'role\_id' => $roleId  
 ]);  
  
 $token = $user->createToken('api\_token')->plainTextToken;  
 return response()->json([  
 'token' => $token,  
 'user' => UserResource::*make*($user),  
 ], 201);  
 }  
  
 public function logout(Request $request): JsonResponse  
 {  
 $request->user()->currentAccessToken()->delete();  
 return response()->json(null, 204);  
 }  
}

В Листинге 20 представлен контроллер жалоб (ComplaintController), который реализует методы для работы с жалобами на альбомы и картинки:

1. index() — метод для получения списка альбомов с жалобами. Пользователи могут фильтровать и сортировать жалобы по различным полям (например, статусу или дате создания). Также применяется пагинация и ограничение количества жалоб на каждый альбом.
2. storeToPicture() — метод для создания жалобы на конкретную картинку в альбоме. Он проверяет доступность альбома для пользователя и наличие уже существующей жалобы. Если жалоба уже существует, возвращается ошибка.
3. storeToAlbum() — метод для создания жалобы на весь альбом. Работает аналогично предыдущему методу, проверяя доступ и существование жалобы на альбом.
4. updateBatch() — метод для массового обновления статуса жалоб. Он применяет новый статус ко всем жалобам, связанным с указанным альбомом или картинкой.
5. destroy() — метод для удаления жалобы. Удалить жалобу может только её автор.

Листинг 20 – Контроллер жалоб app/Http/Controllers/Api/ComplaintController.php

class ComplaintController extends Controller  
{  
 public function index(Request $request): JsonResponse  
 {  
 $user = Auth::*user*();  
 $status = $request->query('status'); *// Получаем параметр status из запроса* $sortBy = $request->query('sort', 'created'); *// Сортировка по полю (по умолчанию дата)* $orderBy = $request->has('reverse') ? 'desc' : 'asc'; *// Направление сортировки* $limit = intval($request->query('limit', 30)); *// Лимит записей (по умолчанию 30)* $limit\_per\_album = intval($request->query('limit\_per\_album', 10)); *// Лимит записей жалоб внутри альбома(по умолчанию 10)  
  
 // Проверка валидации сортировки* $allowedSortFields = [  
 'description' => 'description',  
 'status' => 'status',  
 'album' => 'id',  
 'picture' => 'picture\_id',  
 'fromUser' => 'from\_user\_id',  
 'type' => 'complaint\_type\_id',  
 'created' => 'created\_at',  
 'updated' => 'updated\_at',  
 ];  
  
 $keys = array\_keys($allowedSortFields);  
 if (!in\_array($sortBy, $keys)) {  
 throw new ApiException(  
 'Sort must be of the following types: ' . join(', ', $keys),  
 400  
 );  
 }  
  
 *// Запрос для альбомов* $query = Album::*query*();  
  
 if ($user->role->code !== 'admin') {  
 *// Пользователь видит только альбомы, на которые он пожаловался, исключая свои альбомы* $query->whereHas('complaints', function ($q) use ($user) {  
 $q->where('from\_user\_id', $user->id);  
 });  
 }  
 *// Фильтрация по статусу* $query->whereHas('complaints', function ($q) use ($status) {  
 if ($status === "null") {  
 $q->whereNull('status'); *// Не рассмотренные жалобы* } else if($status == null) {  
 return;  
 }  
 else {  
 $q->where('status', $status); *//Конкретный статус* }  
 });  
  
 $query->with(['complaints' => function ($q) use (  
 $limit,  
 $user,  
 $allowedSortFields,  
 $sortBy,  
 $orderBy,  
 $request,  
 $status,  
 $limit\_per\_album,  
 ) {  
  
 if ($user->role->code !== 'admin') {  
 $q->where('from\_user\_id', $user->id);  
 }  
 $q->with(['type', 'fromUser', 'picture'])  
 ->orderBy($allowedSortFields[$sortBy], $orderBy)->limit($limit\_per\_album);  
  
 }])->orderBy($allowedSortFields[$sortBy], $orderBy)  
 ->withCount('complaints');  
  
 *// Пагинация* $albumsPage = $query->paginate($limit);  
  
 return response()->json([  
 'page' => $albumsPage->currentPage(),  
 'limit' => $albumsPage->perPage(),  
 'total' => $albumsPage->total(),  
 'albums' => AlbumResource::*collection*($albumsPage->items()),  
 ]);  
 }  
  
  
 public function storeToPicture(ComplaintCreateRequest $request, Album $album, Picture $picture): JsonResponse  
 {  
 $user = Auth::*user*();  
 $isAccessible = $album->usersViaAccess()->where('user\_id', $user->id)->exists();  
 if(!$isAccessible)  
 throw new ForbiddenException();  
  
 $isExisted = Complaint  
 ::*where*('from\_user\_id', $user->id)  
 ->where('picture\_id', $picture->id)  
 ->exists();  
 if ($isExisted)  
 throw new ApiException('You are already complain to this picture', 409);  
  
 Complaint::*create*([  
 'picture\_id' => $picture->id,  
 'album\_id' => $album->id,  
 'description' => $request->input('description'),  
 'complaint\_type\_id' => $request->input('typeId'),  
 'from\_user\_id' => $user->id,  
 'about\_user\_id' => $album->user\_id,  
 ]);  
 return response()->json(null, 204);  
 }  
  
 public function storeToAlbum(ComplaintCreateRequest $request, Album $album): JsonResponse  
 {  
 $user = Auth::*user*();  
 $isAccessible = $album->usersViaAccess()->where('user\_id', $user->id)->exists();  
 if(!$isAccessible)  
 throw new ForbiddenException();  
  
 $isExisted = Complaint  
 ::*where*('from\_user\_id', $user->id)  
 ->where('album\_id', $album->id)  
 ->exists();  
 if($isExisted)  
 throw new ApiException('You are already complain to this album', 409);  
  
 Complaint::*create*([  
 'album\_id' => $album->id,  
 'description' => $request->input('description'),  
 'complaint\_type\_id' => $request->input('typeId'),  
 'from\_user\_id' => $user->id,  
 'about\_user\_id' => $album->user\_id,  
 ]);  
 return response()->json(null, 204);  
 }  
  
 public function updateBatch(Complaint $complaint, ComplaintUpdateRequest $request): JsonResponse  
 {  
 $complaints = Complaint::*with*(['type', 'aboutUser', 'fromUser', 'picture', 'album'])  
 ->where('about\_user\_id', $complaint->about\_user\_id)  
 ->where('album\_id' , $complaint->album\_id)  
 ->orWhere('picture\_id' , $complaint->picture\_id)  
 ->get();  
 foreach ($complaints as $complaint) {  
 $complaint->status = $request->input('status');  
 $complaint->save();  
 }  
 return response()->json(['complaints' => ComplaintResource::*collection*($complaints)]);  
 }  
  
 public function destroy(Complaint $complaint): JsonResponse  
 {  
 $user = Auth::*user*();  
 if ($complaint->from\_user\_id !== $user->id)  
 throw new ForbiddenException();  
  
 $complaint->delete();  
 return response()->json(null, 204);  
 }  
}

В Листинге 21 представлен контроллер типов жалоб (ComplaintTypeController), который управляет типами жалоб в системе. Он включает следующие методы:

1. index() — метод для получения списка всех типов жалоб, с загруженными жалобами, связанными с каждым типом.
2. store() — метод для создания нового типа жалобы. После создания типа, очищается кэш для публичных настроек.
3. update() — метод для обновления существующего типа жалобы. Также очищается кэш для публичных настроек.
4. destroy() — метод для удаления типа жалобы. После удаления очищается кэш для публичных настроек.

Листинг 21 – Контроллер типов жалоб app/Http/Controllers/Api/ComplaintTypeController.php

class ComplaintTypeController extends Controller  
{  
 public function index(): JsonResponse  
 {  
 $complaints = ComplaintType::*with*('complaints')->get();  
 return response()->json($complaints);  
 }  
  
 public function store(ComplaintTypeCreateRequest $request): JsonResponse  
 {  
 Cache::*forget*('public\_settings');  
 $complaintType = ComplaintType::*create*($request->validated());  
 return response()->json($complaintType, 201);  
 }  
  
 public function update(ComplaintType $complaintType, ComplaintTypeUpdateRequest $request): JsonResponse  
 {  
 Cache::*forget*('public\_settings');  
 $complaintType->update($request->validated());  
 return response()->json($complaintType);  
 }  
  
 public function destroy(ComplaintType $complaintType): JsonResponse  
 {  
 Cache::*forget*('public\_settings');  
 $complaintType->delete();  
 return response()->json(null, 204);  
 }  
}

В Листинге 22 представлен контроллер приглашений (InvitationController), который управляет процессом создания и использования приглашений для доступа к альбомам. Контроллер включает следующие методы:

1. store() — метод для создания нового приглашения для доступа к альбому. Приглашение может быть с ограничением по времени или без него (через expiresAt или timeLimit). Также генерируется случайная ссылка для приглашения.
2. album() — метод для получения информации о приглашении, альбоме и связанных с ним картинках. Если текущий пользователь является владельцем альбома, он также получит данные о приглашении.
3. join() — метод для присоединения пользователя к альбому через приглашение. Проверяется, не является ли пользователь владельцем альбома и не имеет ли он уже доступа. Если количество присоединившихся пользователей ограничено, оно уменьшается, и приглашение может быть удалено.
4. destroy() — метод для удаления приглашения, доступный только владельцу альбома.

Листинг 22 – Контроллер приглашений app/Http/Controllers/Api/InvitationController.php

class InvitationController extends Controller  
{  
 public function store(Album $album, InvitationCreateRequest $request): JsonResponse  
 {  
 $expiresAt = $request->input('expiresAt');  
 $timeLimit = $request->input('timeLimit');  
  
 $expiresDate = null;  
  
 if ($expiresAt)  
 $expiresDate = $expiresAt;  
  
 else if ($timeLimit)  
 $expiresDate = now()->addMinutes((int)$timeLimit);  
  
 $invitation = Invitation::*create*([  
 'expires\_at' => $expiresDate ?? null,  
 'album\_id' => $album->id,  
 'join\_limit' => $request->joinLimit ?? null,  
 'link' => Str::*random*(8)  
 ]);  
  
 return response()->json(['invitation' => InvitationResource::*make*($invitation)], 201);  
 }  
  
 public function album(Invitation $invitation): JsonResponse  
 {  
 $invitation->failOnExpires();  
 $invitation->load(['album', 'album.user']);  
 $pictures = Picture  
 ::*where*('album\_id', $invitation->album\_id)  
 ->orderBy('date', 'DESC')  
 ->limit(30)  
 ->get();  
  
 $album = Album  
 ::*where*('id', $invitation->album\_id)  
 ->withCount([  
 'pictures',  
 ])  
 ->with([  
 'pictures' => fn ($query) => $query->orderBy('date', 'DESC')->limit(4),  
 'user',  
 ])  
 ->first();  
  
 $res = [  
 'invitation' => $invitation,  
 'album' => AlbumResource::*make*($album),  
 'pictures' => PictureResource::*collection*($pictures),  
 ];  
  
  
 if (Auth::*id*() != null && $invitation?->album?->user?->id == Auth::*id*())  
 $res['invite'] = InvitationResource::*make*($invitation);  
  
 return response()->json($res);  
 }  
  
 public function join(Invitation $invitation): JsonResponse  
 {  
 $invitation->failOnExpires();  
 $user = Auth::*user*();  
  
 if ($user->id === $invitation->album->user\_id)  
 throw new ApiException('You are owner', 409);  
  
 if (AlbumAccess  
 ::*where*('user\_id', $user->id)  
 ->where('album\_id', $invitation->album\_id)  
 ->first()  
 ) throw new ApiException('Already accessible', 409);  
  
 AlbumAccess::*create*([  
 'user\_id' => $user->id,  
 'album\_id' => $invitation->album\_id  
 ]);  
  
 if ($invitation->join\_limit !== null) {  
 $invitation->join\_limit -= 1;  
  
 if ($invitation->join\_limit > 0)  
 $invitation->save();  
 else  
 $invitation->delete();  
 }  
 return response()->json(null, 204);  
 }  
  
 public function destroy(Invitation $invitation): JsonResponse  
 {  
 $user = Auth::*user*();  
 if ($user->id !== $invitation->album->user\_id)  
 throw new ForbiddenException();  
  
 $invitation->delete();  
 return response()->json(null, 204);  
 }  
}

В Листинге 23 представлен контроллер картинок (PictureController), который управляет процессом загрузки, отображения и удаления картинок в альбомах. Контроллер включает следующие методы:

1. index() — метод для получения списка картинок в альбоме с возможностью фильтрации по тегам и сортировки по различным полям (дата, имя, размер и т.д.). Также позволяет загружать страницы с ограничением на количество картинок.
2. store() — метод для загрузки новых картинок в альбом. Включает проверку на размер файла, разрешенные форматы, а также проверку на существование картинок с одинаковым хешем или именем. Если все проверки пройдены, картинка сохраняется на сервере и в базе данных.
3. info() — метод для получения информации о конкретной картинке. Включает данные о картинке и её теги, а также подпись (sign) альбома, если пользователь авторизован.
4. thumbnail() — метод для получения миниатюры картинки в зависимости от запрашиваемого ориентиования (по ширине, высоте или квадратное). Если миниатюра не существует, она генерируется динамически.
5. original() — метод для скачивания оригинального изображения из хранилища, доступный для авторизованных пользователей.
6. download() — метод для скачивания изображения в оригинальном формате. Поддерживает прямое скачивание файла.
7. destroy() — метод для удаления картинки из альбома. Картинка удаляется с сервера и из базы данных.

Листинг 23 – Контроллер картинок app/Http/Controllers/Api/PictureController.php

class PictureController extends Controller  
{  
 public function index(Request $request, Album $album): JsonResponse  
 {  
 *// Получаем ID тегов* $tagIds = null;  
 $tagsString = $request->input('tags');  
 if ($tagsString)  
 $tagIds = explode(',', $tagsString);  
  
 $sortBy = $request->query('sort', 'date'); *// Сортировка по полю (по умолчанию дата)* $orderBy = $request->has('reverse') ? 'desc' : 'asc'; *// Направление сортировки* $user = Auth::*user*();  
  
 *// Проверка валидации сортировки* $allowedSortFields = ['date', 'name', 'width', 'height', 'size'];  
  
 *// Админ может делать сортировку по жалобам* if ($user->role->code === 'admin') {  
 if($sortBy === 'complaints') {  
 $allowedSortFields[] = 'complaints';  
 }  
 }  
  
 if (!in\_array($sortBy, $allowedSortFields))  
 throw new ApiException('Sort must be of the following types: ' . join(', ', $allowedSortFields), 400);  
  
  
 *// Фильтрация по заданным параметрам* $query = Picture::*with*('tags')  
 ->withCount('complaints')  
 ->where('album\_id', $album->id);  
  
 *// Сортировка* if ($sortBy === 'complaints') {  
 $query->orderBy('complaints\_count', $orderBy); *// Сортировка по количеству жалоб* } else {  
 $query->orderBy($sortBy, $orderBy); *// Сортировка по другим полям* }  
  
 if ($tagIds) {  
 $query->whereHas('tags', function ($query) use ($tagIds) {  
 $query->whereIn('id', $tagIds); *// Фильтруем по любому из указанных тегов* }, '=', count($tagIds));  
 }  
  
 $limit = intval($request->limit);  
 if (!$limit)  
 $limit = 30;  
  
 $picturesPage = $query->paginate($limit);  
  
 $sign = $album->getSign($user);  
  
 return response()->json([  
 'sign' => $sign,  
 'page' => $picturesPage->currentPage(),  
 'limit' => $picturesPage->perPage(),  
 'total' => $picturesPage->total(),  
 'pictures' => PictureResource::*collection*($picturesPage->items()),  
 ]);  
 }  
  
 public function store(PictureCreateRequest $request, Album $album): JsonResponse  
 {  
 $user = Auth::*user*();  
 $pictures = $request->pictures;  
 $pathToSave = Picture::*getPathStatic*($user->id, $album->id);  
  
 $spaceInfo = SpaceInfo::*getCached*();  
 if ($spaceInfo->usedPercent >= config('settings.upload\_disable\_percentage'))  
 throw new ApiException('Server in read-only mode', 400);  
  
 *// Получаем сколько сейчас весят пользовательские картинки и какой лимит сервера по загрузкам* $currentStorageSize = $user->pictures()->sum('size');  
 $maxStorageSize = config('settings.free\_storage\_limit');  
  
 *// Массивы для ответа* $errored = [];  
 $successful = [];  
  
 *// Обрабатываем файлы по одному* foreach ($pictures as $key => $pictureInRequests) {  
 $file = $request->file("pictures.$key.file");  
 $date = $pictureInRequests['date'] ?? now();  
 $filename = $file->getClientOriginalName();  
 try {  
 *// Валидация mimes файла и пропускаем если не в разрешённых* $validator = Validator::*make*($pictureInRequests, [  
 'file' => 'mimes:' . implode(',', config('settings.allowed\_upload\_mimes')),  
 ]);  
 if ($validator->fails()) {  
 $errored[] = [  
 'name' => $filename,  
 'message' => $validator->errors()->toArray()['file'],  
 ];  
 continue;  
 }  
  
 *// Проверяем, превышает ли размер лимит* $filesize = $file->getSize();  
 if ($currentStorageSize + $filesize >= $maxStorageSize) {  
 $errored[] = [  
 'name' => $filename,  
 'message' => 'Storage limit reached',  
 ];  
 continue;  
 }  
  
 *// Проверка, существует ли картинка с таким хешем и пропускаем если да* $tmpPath = $file->getRealPath();  
 $hash = hash\_file('xxh3', $tmpPath);  
 $existedPictureByHash = Picture  
 ::*where*('album\_id', $album->id)  
 ->where('hash', $hash)  
 ->first();  
 if ($existedPictureByHash) {  
 $errored[] = [  
 'name' => $filename,  
 'message' => 'Already exist with this hash',  
 'picture' => PictureResource::*make*($existedPictureByHash),  
 ];  
 continue;  
 }  
 *// Проверка, существует ли картинка с таким именем и модифицируем имя счётчиком если да* $counter = 1;  
 $extension = pathinfo($filename, *PATHINFO\_EXTENSION*);  
 $nameWithoutExt = pathinfo($filename, *PATHINFO\_FILENAME*);  
 $filenameValid = $filename;  
 while (Picture  
 ::*where*('album\_id', $album->id)  
 ->where('name', $filenameValid)  
 ->exists()  
 ) {  
 $counter++;  
 $filenameValid = "$nameWithoutExt ($counter).$extension";  
 }  
  
 *// Получение размеров, пропускаем если не получилось* try {  
 $imagesizes = getimagesize($tmpPath);  
 }  
 catch (Exception) {  
 $errored[] = [  
 'name' => $filename,  
 'message' => 'No sizes',  
 ];  
 continue;  
 }  
  
 *// Создаём запись в БД* $pictureDB = Picture::*create*([  
 'name' => $filenameValid,  
 'hash' => $hash,  
 'date' => $date,  
 'size' => $filesize,  
 'width' => $imagesizes[0],  
 'height' => $imagesizes[1],  
 'album\_id' => $album->id,  
 ]);  
  
 *// Сохраняем в ФС* $file->storeAs($pathToSave, $filenameValid);  
  
 *// Добавление текущего занятого пространства* $currentStorageSize += $filesize;  
  
 *// Запись как успешного в ответ* $successful[] = $pictureDB;  
 } catch (Exception) {  
 *// Что-то пошло не так* $errored[] = [  
 'name' => $filename,  
 'message' => 'Something going wrong',  
 ];  
 }  
 }  
 return response()->json([  
 'sign' => $album->getSign($user),  
 'successful' => $successful,  
 'errored' => $errored,  
 ]);  
 }  
  
 public function info(Album $album, Picture $picture): JsonResponse  
 {  
 $picture->load('tags');  
 $user = Auth::*user*();  
 $sign = $album->getSign($user);  
 return response()->json([  
 'sign' => $sign,  
 'picture' => PictureResource::*make*($picture),  
 ]);  
 }  
  
 public function thumbnail(Request $request, $albumId, $pictureId, $orientation, $size): BinaryFileResponse|JsonResponse|RedirectResponse  
 {  
 $allowedSizes = config('settings.allowed\_preview\_sizes');  
  
 $ownerId = $request->attributes->get('ownerId');  
 $orientation = strtolower($orientation);  
  
 *// Проверка существования в ФС с исходными данными* $thumbPath = Picture::*getPathThumbStatic*($ownerId, $albumId, $pictureId, $orientation, $size);  
  
 if (!Storage::*exists*($thumbPath)) {  
 *// Проверка запрашиваемого размера и редирект, если не прошло* $askedSize = $size;  
 $allowSize = false;  
 foreach ($allowedSizes as $allowedSize) {  
 if ($size <= $allowedSize) {  
 $size = $allowedSize;  
 $allowSize = true;  
 break;  
 }  
 }  
 if (!$allowSize) $size = $allowedSizes[count($allowedSizes)-1];  
 if ($askedSize != $size) return redirect()  
 ->route('thumbnail',  
 [$albumId, $pictureId, $orientation, $size, 'sign' => $request->query('sign')])  
 ->header('Cache-Control', ['max-age=86400', 'private']);  
  
 try {  
 $picture = Picture  
 ::*where*('album\_id', $albumId)  
 ->where('id', $pictureId)  
 ->firstOrFail();  
 }  
 catch (Exception) {  
 return response()->json(['message' => 'Picture not found'], 404)  
 ->header('Cache-Control', ['max-age=86400', 'private']);  
 }  
  
 *// Редирект если изображение квадратное* if ($picture->width === $picture->height && $orientation !== 'q') return redirect()  
 ->route('thumbnail',  
 [$albumId, $pictureId, 'q', $size, 'sign' => $request->query('sign')])  
 ->header('Cache-Control', ['max-age=86400', 'private']);  
  
 *// Создание превью* $picturePath = $picture->getPath($ownerId);  
 $thumb = *Intervention*::*read*(Storage::*get*($picturePath));  
  
 switch ($orientation) {  
 case 'w':  
 $thumb->scaleDown(width: $size);  
 break;  
 case 'h':  
 $thumb->scaleDown(height: $size);  
 break;  
 default:  
 $thumb->coverDown($size, $size);  
 break;  
 }  
 $dirname = pathinfo($thumbPath, *PATHINFO\_DIRNAME*);  
 if (!Storage::*exists*($dirname))  
 Storage::*makeDirectory*($dirname);  
  
 $thumb  
 ->toJpeg(90)  
 ->save(Storage::*path*($thumbPath));  
 }  
 return response()->file(Storage::*path*($thumbPath), ['Cache-Control' => 'max-age=86400, private']);  
 }  
  
 public function original($albumId, Picture $picture, Request $request): BinaryFileResponse  
 {  
 $ownerId = $request->attributes->get('ownerId');  
 $path = Storage::*path*($picture->getPath($ownerId));  
 return response()->file($path);  
 }  
  
 public function download($albumId, Picture $picture, Request $request): BinaryFileResponse  
 {  
 $ownerId = $request->attributes->get('ownerId');  
 $path = Storage::*path*($picture->getPath($ownerId));  
 return response()->download($path, $picture->name);  
 }  
  
 public function destroy(Album $album, Picture $picture): JsonResponse  
 {  
 Storage::*delete*($picture->getPath($album->user\_id));  
 $picture->delete();  
 return response()->json(null, 204);  
 }  
}

В листинге 24 представлен контроллер настроек app/Http/Controllers/Api/SettingsController.php

Контроллер настроек управляет публичными и приватными настройками системы. Он включает следующие методы:

1. public() — метод для получения публичных настроек. Он использует кэширование для хранения значений настроек, таких как разрешенные типы файлов для загрузки, размеры превью, лимит для предупреждений о блокировке, лимит бесплатного хранилища, а также типы жалоб. Если настройки не находятся в кэше, они загружаются заново. Также проверяется, не превышает ли используемое пространство установленный порог для отключения загрузок.
2. index() — метод для получения всех настроек системы и информации о текущем состоянии хранилища. Он возвращает как стандартные настройки из конфигурации, так и текущее использование пространства, полученное из кэша.
3. edit(SettingsUpdateRequest $request) — метод для обновления настроек, хранящихся в .env файле. Принимает строку с ключом и значением, находит соответствующую строку в .env файле и заменяет её, либо добавляет новую строку в конец файла. После этого кэш настроек удаляется, а конфигурация обновляется с помощью команды Artisan. Этот метод защищён запросом, проверяющим правильность входных данных.

Листинг 24 – Контроллер настроек app/Http/Controllers/Api/SettingsController.php

class SettingsController extends Controller  
{  
 public function public()  
 {  
 $settings = Cache::*remember*('public\_settings', (7 \* 24 \* 60 \* 60), fn () => [  
 'allowed\_upload\_mimes' => config('settings.allowed\_upload\_mimes'),  
 'allowed\_preview\_sizes' => config('settings.allowed\_preview\_sizes'),  
 'warning\_limit\_for\_ban' => config('settings.warning\_limit\_for\_ban'),  
 'free\_storage\_limit' => config('settings.free\_storage\_limit'),  
 'complaint\_types' => ComplaintTypeResource::*collection*(ComplaintType::*all*()),  
 ]);  
  
 $spaceInfo = SpaceInfo::*getCached*();  
 return response()->json([  
 'settings' => [  
 ...$settings,  
 'is\_upload\_disabled' => $spaceInfo->usedPercent >= config('settings.upload\_disable\_percentage'),  
 ],  
 ]);  
 }  
  
 public function index()  
 {  
 $settings = config('settings');  
 $space = SpaceInfo::*getCached*();  
  
 return response()->json([  
 'settings' => $settings,  
 'space' => $space  
 ]);  
 }  
  
 public function edit(SettingsUpdateRequest $request)  
 {  
 Cache::*forget*('public\_settings');  
  
 $key = $request->key;  
 $value = $request->value; *// Принимаем строку  
  
 // Путь к .env файлу* $envPath = base\_path('.env');  
  
 *// Проверяем, существует ли .env файл* if (!File::*exists*($envPath))  
 throw new ApiException('Unable to update .env file', 500);  
  
 *// Читаем содержимое .env файла* $envContent = File::*get*($envPath);  
  
 *// Ключ для поиска* $key = strtoupper($key);  
  
 *// Проверяем, существует ли уже эта настройка в .env* $pattern = "/^{$key}=(.\*)$/m";  
  
 *// Если строка найдена, заменяем её* if (preg\_match($pattern, $envContent))  
 $envContent = preg\_replace($pattern, "{$key}={$value}", $envContent);  
  
 *// Если строки нет, добавляем её в конец* else  
 $envContent .= "\n{$key}={$value}";  
  
 *// Записываем обновленное содержимое в .env файл* File::*put*($envPath, $envContent);  
  
 *// Очистка кэша конфигурации, чтобы новые значения вступили в силу* Artisan::*call*('config:clear');  
  
 return response()->json(['message' => 'Settings updated successfully']);  
 }  
}

В Листинге 25 представлен контроллер тегов app/Http/Controllers/Api/TagController.php, который управляет созданием, обновлением, удалением и привязкой тегов к картинкам. Он включает следующие методы:

1. attachToPicture($albumId, Picture $picture, Tag $tag) — метод для привязки тега к картинке. Если тег принадлежит текущему пользователю, метод проверяет, не прикреплён ли уже этот тег к картинке, и, если нет, привязывает его. В случае успеха возвращается пустой ответ с кодом состояния 204.
2. detachToPicture($albumId, Picture $picture, Tag $tag) — метод для отвязывания тега от картинки. Также проверяется, принадлежит ли тег текущему пользователю. Если тег прикреплён к картинке, он удаляется. Ответ возвращается с кодом состояния 204.
3. index() — метод для получения всех тегов текущего пользователя. Возвращается коллекция тегов, привязанных к пользователю, в формате JSON.
4. store(TagCreateRequest $request) — метод для создания нового тега. Проверяется, не существует ли уже тег с таким значением у текущего пользователя. Если тег уже существует, выбрасывается исключение AlreadyExistsException. Если тег уникален, он создается и возвращается в формате JSON с кодом состояния 201.
5. show(Tag $tag) — метод для получения информации о конкретном теге. Если тег принадлежит текущему пользователю, загружаются связанные с ним картинки, и возвращается подробная информация о теге в формате JSON.
6. update(Tag $tag, TagUpdateRequest $request) — метод для обновления данных тега. Если тег принадлежит текущему пользователю, его данные обновляются и возвращается обновлённая информация о теге в формате JSON.
7. destroy(Tag $tag) — метод для удаления тега. Проверяется, принадлежит ли тег текущему пользователю. Если да, то тег удаляется, и возвращается пустой ответ с кодом состояния 204.

Листинг 25 – Контроллер тегов app/Http/Controllers/Api/TagController.php

class TagController extends Controller  
{  
 public function attachToPicture($albumId, Picture $picture, Tag $tag): JsonResponse  
 {  
 if ($tag->user\_id !== Auth::*id*())  
 throw new ForbiddenException();  
  
 $tagIds = collect([$tag->id]);  
 $existingIds = $picture->tags()  
 ->whereIn('tags.id', $tagIds)  
 ->pluck('tags.id');  
  
 $picture->tags()->attach($tagIds->diff($existingIds));  
 return response()->json(null, 204);  
 }  
 public function detachToPicture($albumId, Picture $picture, Tag $tag): JsonResponse  
 {  
 if ($tag->user\_id !== Auth::*id*())  
 throw new ForbiddenException();  
  
 $tagIds = collect([$tag->id]);  
 $existingIds = $picture->tags()  
 ->whereIn('tags.id', $tagIds)  
 ->pluck('tags.id');  
  
 $picture->tags()->detach($tagIds->intersect($existingIds));  
 return response()->json(null, 204);  
 }  
  
 public function index(): JsonResponse  
 {  
 $tags = Tag::*where*('user\_id', Auth::*id*())->get();  
 return response()->json(['tags' => TagResource::*collection*($tags)]);  
 }  
  
 public function store(TagCreateRequest $request): JsonResponse  
 {  
 $value = $request->input('value');  
 $user = Auth::*user*();  
 $existedTag = Tag  
 ::*where*('user\_id', $user->id)  
 ->where('value', $value)  
 ->first();  
 if($existedTag)  
 throw new AlreadyExistsException($existedTag);  
  
 $tag = Tag::*create*([  
 'value' => $value,  
 'user\_id' => $user->id  
 ]);  
 return response()->json(['tag' => TagResource::*make*($tag)], 201);  
 }  
 public function show(Tag $tag): JsonResponse  
 {  
 if ($tag->user\_id !== Auth::*id*())  
 throw new ForbiddenException();  
  
 $tag->load('pictures');  
 return response()->json(['tag' => TagResource::*make*($tag)]);  
 }  
 public function update(Tag $tag, TagUpdateRequest $request): JsonResponse  
 {  
 if ($tag->user\_id !== Auth::*id*())  
 throw new ForbiddenException();  
  
 $tag->update($request->all());  
 return response()->json(['tag' => TagResource::*make*($tag)]);  
 }  
 public function destroy(Tag $tag): JsonResponse  
 {  
 if ($tag->user\_id !== Auth::*id*())  
 throw new ForbiddenException();  
  
 $tag->delete();  
 return response()->json(null, 204);  
 }  
}

В листинге 26 представлен контроллер пользователей app/Http/Controllers/Api/UserController.php, который управляет информацией о пользователях и их действиями, такими как обновление данных и просмотр профиля. Контроллер включает следующие методы:

1. index() — метод для получения списка всех пользователей. Для каждого пользователя загружаются связанные роли, предупреждения, а также статистика по количеству тегов, альбомов, картинок и жалоб. Возвращает список пользователей в формате JSON.
2. show(User $user) — метод для получения подробной информации о конкретном пользователе. Загружаются данные о роли, предупреждениях, жалобах и статистика по количеству тегов, альбомов и картинок. Возвращает информацию о пользователе в формате JSON.
3. showSelf() — метод для получения информации о текущем авторизованном пользователе. Загружаются данные о его роли, предупреждениях, жалобах, а также статистика по количеству тегов, альбомов и картинок. Возвращает информацию о текущем пользователе в формате JSON.
4. update(UserUpdateRequest $request, User $user) — метод для обновления данных конкретного пользователя. Если в запросе указано, что пользователь должен быть заблокирован (через is\_banned), вызывается метод ban(). После обновления данных пользователя возвращается обновлённая информация о нём в формате JSON.
5. updateSelf(UserSelfUpdateRequest $request) — метод для обновления данных текущего авторизованного пользователя. Применяются только те данные, которые указаны в запросе, и после этого возвращается обновлённая информация о текущем пользователе в формате JSON.

Листинг 26 – Контроллер пользователей app/Http/Controllers/Api/UserController.php

class UserController extends Controller  
{  
 public function index(): JsonResponse  
 {  
 $users = User::*with*([  
 'role', 'warnings',  
 ])->withCount([  
 'tags', 'albums', 'pictures', 'albumsViaAccess', 'complaintsFrom', 'complaintsAbout'  
 ])->get();  
  
 return response()->json(['users' => UserResource::*collection*($users)]);  
 }  
  
 public function show(User $user): JsonResponse  
 {  
 $user->load([  
 'role', 'warnings', 'complaintsAbout', 'complaintsFrom',  
 ])->loadCount([  
 'tags', 'albums', 'pictures', 'albumsViaAccess',  
 ]);  
 return response()->json(['user' => UserResource::*make*($user)]);  
 }  
  
 public function showSelf(): JsonResponse  
 {  
 $user = Auth::*user*();  
 $user->load([  
 'role', 'warnings', 'complaintsFrom',  
 ])->loadCount([  
 'tags', 'albums', 'pictures', 'albumsViaAccess',  
 ]);  
 return response()->json(['user' => UserResource::*make*($user)]);  
 }  
  
 public function update(UserUpdateRequest $request, User $user): JsonResponse  
 {  
 if($request->is\_banned == 1)  
 {  
 $user->ban();  
 }  
 $user->update($request->validated());  
 return response()->json(['user' => UserResource::*make*($user)]);  
 }  
  
 public function updateSelf(UserSelfUpdateRequest $request): JsonResponse  
 {  
 $user = Auth::*user*();  
 $user->update($request->validated());  
 return response()->json(['user' => UserResource::*make*($user)]);  
 }  
}

В листинге 27 представлен контроллер предупреждений app/Http/Controllers/Api/WarningController.php, который управляет процессом создания и удаления предупреждений для пользователей. Он включает следующие методы:

1. store(Request $request, User $user) — метод для создания нового предупреждения для пользователя. Перед созданием предупреждения проверяется, не достиг ли пользователь лимита предупреждений, установленного в конфигурации (warning\_limit\_for\_ban). Если лимит превышен, вызывается метод ban() для блокировки пользователя, и выбрасывается исключение с сообщением о блокировке. Если лимит не превышен, создаётся новое предупреждение с указанным комментарием. Возвращает информацию о созданном предупреждении в формате JSON.
2. destroy(User $user, Warning $warning) — метод для удаления предупреждения. При успешном удалении предупреждения возвращается статус 204 No Content.

Листинг 27 – Контроллер предупреждений app/Http/Controllers/Api/WarningController.php

class WarningController extends Controller  
{  
 public function store(Request $request, User $user): JsonResponse  
 {  
 if (Warning::*where*('user\_id', $user->id)->count() >= config('settings.warning\_limit\_for\_ban'))  
 {  
 $user->ban();  
 throw new ApiException("User $user->username already banned", 409);  
 }  
 $warning = Warning::*create*([  
 'user\_id' => $user->id,  
 'comment' => $request->comment  
 ]);  
 return response()->json(['warning' => WarningResource::*make*($warning)], 201);  
 }  
  
 public function destroy(User $user, Warning $warning): JsonResponse  
 {  
 $warning->delete();  
 return response()->json(null, 204);  
 }  
}

## Посредники

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 28 показан посредник CheckAlbumAccess, который проверяет, имеет ли пользователь доступ к альбому. Владелец альбома пропускается сразу, пользователи с правами чтения – только на GET-запросах. Администраторы могут получить доступ, если на владельца альбома есть нерассмотренная жалоба. Если ни одно из условий не выполнено, выбрасывается ForbiddenException.

Листинг 28 – Посредник проверки доступа к альбому app/Http/Middleware/CheckAccess.php

class CheckAlbumAccess  
{  
 public function handle(Request $request, Closure $next)  
 {  
 $user = $request->user();  
 $album = $request->route('album');  
 if (!($album instanceof Album))  
 $album = Album::*findOrFailCustom*($album);  
  
 *// Впускаем владельца* if ($album->user\_id === $user->id)  
 return $next($request);  
  
 *// Впускаем по выданному владельцем доступу на чтение* if (  
 $request->isMethod('GET') &&  
 AlbumAccess  
 ::*where*('album\_id', $album->id)  
 ->where('user\_id' , $user->id)  
 ->exists()  
 ) return $next($request);  
  
 *// Впускаем админа если есть жалоба* if ($user->role->code === 'admin') {  
 *// Проверяем, есть ли жалоба на этого пользователя, помимо отклоненных* $complaintExists = Complaint  
 ::*where*('about\_user\_id', $album->user\_id)  
 ->where('status',null)  
 ->where('album\_id', $album->id) *// Все нерассмотренные жалобы* ->exists();  
  
 *// Если жалоба существует, разрешаем просмотр альбома* if ($complaintExists)  
 return $next($request);  
 }  
  
 *// Незваных отбрасываем* throw new ForbiddenException();  
 }  
}

В Листинге 29 реализован посредник CheckRole, который проверяет, соответствует ли роль пользователя списку разрешённых. Если пользователь не авторизован – выбрасывается UnauthorizedException, если его роль не входит в список – ForbiddenException. Если всё в порядке, роль добавляется в атрибуты запроса, и выполнение продолжается.

Листинг 29 – Посредник проверки роли app/Http/Middleware/CheckRole.php

class CheckRole  
{  
 public function handle(Request $request, Closure $next, ...$allowedRoles)  
 {  
 $user = $request->user();  
  
 if (!$user)  
 throw new UnauthorizedException();  
  
 $currentRole = $user->role->code;  
 if (!in\_array($currentRole, $allowedRoles))  
 throw new ForbiddenException();  
  
 $request->attributes->add(['role' => $currentRole]);  
 return $next($request);  
 }  
}

В Листинге 30 представлен посредник LogRequest, который логирует запросы и SQL-запросы, выполняемые в их рамках. В методе handle включается логирование SQL-запросов для методов GET, PUT, PATCH, POST, DELETE. В terminate после обработки запроса извлекаются выполненные SQL-запросы, статус ответа, данные о пользователе и источнике запроса, после чего информация записывается в лог http-request.

Листинг 30 – Посредник логирования запросов app/Http/Middleware/LogRequest.php

class LogRequest  
{  
 public function handle(Request $request, Closure $next)  
 {  
 try {  
 $method = strtoupper($request->getMethod());  
 if (in\_array($method, ['GET', 'PUT', 'PATCH', 'POST', 'DELETE']))  
 DB::*enableQueryLog*();  
 }  
 catch (\Exception $e) {  
 Log::*error*($e);  
 }  
 return $next($request);  
 }  
 public function terminate(Request $request, $response)  
 {  
 try {  
 $method = strtoupper($request->getMethod());  
 if (!in\_array($method, ['GET', 'PUT', 'PATCH', 'POST', 'DELETE']))  
 return;  
 $method = str\_pad($method, 6, ' ');  
  
 $dbQueries = DB::*getQueryLog*();  
 $dbQueryStrings = [];  
 foreach ($dbQueries as $dbQuery) {  
 $query = $dbQuery['query'];  
 $bindings = $dbQuery['bindings'];  
  
 *// Заменяем вопросительные знаки на значения параметров* foreach ($bindings as $binding)  
 $query = preg\_replace('/\?/', "'" . addslashes($binding) . "'", $query, 1);  
  
 $dbQueryStrings[] = $query . ";\n";  
 }  
  
 $code = $response->getStatusCode();  
 $sign = $response->isSuccessful() ? "🟢" : "🔴";  
  
 $uri = $request->getPathInfo();  
 $origin = $request->headers->get('origin') ?? "NO\_ORIGIN";  
 $userId = $request->user()?->id ??  
 ($request->has('sign')  
 ? explode('\_', $request->sign)[0]  
 : "GUEST ");  
  
 if (is\_numeric($userId))  
 $userId = str\_pad($userId, 7, '0', *STR\_PAD\_LEFT*);  
  
 $reqQuery = $request->getQueryString();  
 $message = "$sign $code $method 👤$userId $origin $uri"  
 . ($reqQuery ? "?$reqQuery" : '')  
 . (!empty($dbQueryStrings) ? "\n" : '')  
 . implode('', $dbQueryStrings);  
  
 Log::*channel*('http-request')->log('info', $message);  
 }  
 catch (\Exception $e) {  
 Log::*error*($e);  
 }  
 }  
}

В Листинге 31 представлен посредник SanctumAuth, который проверяет аутентификацию пользователя через Sanctum и его статус бана. Если пользователь не аутентифицирован, выбрасывается UnauthorizedException. Если у него установлен флаг is\_banned, выбрасывается ForbiddenException. В противном случае пользователь устанавливается в контексте Auth, и запрос передаётся дальше.

Листинг 31 – Посредник проверки бана app/Http/Middleware/SanctumAuth.php

class SanctumAuth  
{  
 public function handle(Request $request, Closure $next): Response  
 {  
 $user = Auth::*guard*('sanctum')->user();  
 if (!$user)  
 throw new UnauthorizedException();  
 if ($user->is\_banned)  
 throw new ForbiddenException('Your account is banned');  
 Auth::*setUser*($user);  
 return $next($request);  
 }  
}

В Листинге 32 представлен посредник SignCheck, который проверяет сигнатуру (sign) в URL-запросе для доступа к альбому. Он разбирает переданный параметр sign, извлекает userId и signCode, затем сверяет его с закэшированным значением. Если кэш отсутствует, он вычисляет допустимую сигнатуру с помощью Hash::check(). При успешной проверке сигнатура кэшируется на 1 час, иначе выбрасывается ForbiddenException. В запрос добавляется атрибут ownerId, после чего передаётся дальше.

Листинг 32 – Посредник проверки сигнатуры в адресе app/Http/Middleware/SignCheck.php

class SignCheck  
{  
 public function handle(Request $request, Closure $next)  
 {  
 $sign = $request->query('sign');  
 if (!$sign)  
 throw new ForbiddenException();  
  
 $albumId = $request->route('album');  
 $album = null;  
 if ($albumId instanceof Album) {  
 $album = $albumId;  
 $albumId = $album->id;  
 }  
 try {  
 $signExploded = explode('\_', $sign);  
 $userId = $signExploded[0];  
 $signCode = $signExploded[1];  
 }  
 catch (Exception) {  
 throw new ForbiddenException();  
 }  
 $cacheKey = Album::*signCacheKey*($albumId, $userId);  
 $cachedSignRaw = Cache::*get*($cacheKey);  
  
 $cachedSign = null;  
 if ($cachedSignRaw) {  
 $cachedSignExploded = explode('\_', $cachedSignRaw);  
 $cachedSign = $cachedSignExploded[0];  
 $ownerId = $cachedSignExploded[1];  
 }  
 if ($cachedSign !== $signCode) {  
 $user = User::*findOrFailCustom*($userId);  
 if (!$album)  
 $album = Album::*findOrFailCustom*($albumId);  
  
 $ownerId = $album->user\_id;  
  
 $string = Album::*signNonHash*($user, $albumId);  
  
 try {  
 $allow = Hash::*check*($string, base64\_decode($signCode));  
 }  
 catch (Exception) {  
 throw new ForbiddenException();  
 }  
  
 if ($allow)  
 Cache::*put*($cacheKey, $signCode . '\_' . $ownerId, 3600);  
 else  
 throw new ForbiddenException();  
 }  
 $request->attributes->add(['ownerId' => $ownerId]);  
 return $next($request);  
 }  
}

# Глава 4. Автотестирование API

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

Тестирование включало в себя:

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

Разработанные тесты позволили 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 (см. листинг 33)

Листинг 33 – Тест 401 Unauthorized

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(401);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'message'", function () {

    pm.expect(pm.response.json()).to.have.property("message");

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| POST | {{api}}/login |
| POST | {{api}}/logout |
| POST | {{api}}/albums |
| GET | {{api}}/albums |
| GET | {{api}}/albums/{{idalbumtest}} |
| POST | {{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 |
| GET | {{api}}/accesses |
| DELETE | {{api}}/albums/{{idalbumtest}}/accesses/{{idusertest}} |
| POST | {{api}}/albums/{{idalbumtest}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| POST | {{api}}/tags |
| GET | {{api}}/tags |
| POST | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/tags/{{idtagtest}} |
| DELETE | {{api}}/tags/{{idtagtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| GET | {{api}}/complaints/types |
| POST | {{api}}/complaints/types |
| POST | {{api}}/complaints/types/{{idtypetest}} |
| DELETE | {{api}}/complaints/types/{{idtypetest}} |
| POST | {{api}}/albums/{{idalbumtest}}/complaint |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/complaint |
| GET | {{api}}/complaints |
| POST | {{api}}/complaints/{{idcomplainttest}} |
| DELETE | {{api}}/complaints/{{idcomplainttest}} |
| POST | {{api}}/users/{{idusertest}}/warnings |
| DELETE | {{api}}/users/{{idusertest}}/warnings/{{idwarningtest}} |
| GET | {{api}}/users/me |
| POST | {{api}}/users/me |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| POST | {{api}}/users/{{idusertest}} |
| GET | {{api}}/settings |
| POST | {{api}}/settings |

## 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 34.

Листинг 34 – Тест 422 Unproccessable Content

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(422);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'message'", function () {

    pm.expect(pm.response.json()).to.have.property("message");

});

pm.test("Проверка наличия name в errors", function () {

    pm.expect(pm.response.json().errors).to.have.property("name");

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| POST | {{api}}/login |
| POST | {{api}}/register |
| POST | {{api}}/login |
| POST | {{api}}/albums |
| POST | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| POST | {{api}}/albums/{{idalbumtest}}/pictures |
| POST | {{api}}/tags |
| POST | {{api}}/tags/{{idtagtest}} |
| POST | {{api}}/complaints/types |
| POST | {{api}}/complaints/types/{{idtypetest}} |
| POST | {{api}}/albums/{{idalbumtest}}/complaint |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/complaint |
| POST | {{api}}/complaints/{{idcomplainttest}} |
| POST | {{api}}/users/{{idusertest}}/warnings |
| POST | {{api}}/users/me |
| POST | {{api}}/users/{{idusertest}} |
| POST | {{api}}/settings |

## 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 тест представлен в листинге 35.

Листинг 35 – Тест 403 Forbidden

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(403);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'message'", function () {

    pm.expect(pm.response.json()).to.have.property("message");

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| POST | {{api}}/login |
| GET | {{api}}/albums/{{idalbumtest}} |
| POST | {{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}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/thumb/{{orientation}}{{size}}?sign={{$random}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/original?sign={{$randomBitcoin}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/download?sign={{$randomBitcoin}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| POST | {{api}}/complaints/types |
| POST | {{api}}/complaints/types/{{idtypetest}} |
| DELETE | {{api}}/complaints/types/{{idtypetest}} |
| POST | {{api}}/albums/{{idalbumtest}}/complaint |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/complaint |
| POST | {{api}}/complaints/{{idcomplainttest}} |
| DELETE | {{api}}/complaints/{{idcomplainttest}} |
| POST | {{api}}/users/{{idusertest}}/warnings |
| DELETE | {{api}}/users/{{idusertest}}/warnings/{{idwarningtest}} |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| POST | {{api}}/users/{{idusertest}} |
| GET | {{api}}/settings |
| POST | {{api}}/settings |

## 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 36.

Листинг 36 – Тест 204 No Content

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(204);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка, что тело ответа пустое", function () {

    pm.expect(pm.response.text()).to.equal("");

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| POST | {{api}}/logout |
| DELETE | {{api}}/albums/{{idalbumtest}} |
| DELETE | {{api}}/invitation/{{codeinvitationtest}} |
| POST | {{api}}/invitation/{{codeinvitationtest}}/join |
| DELETE | {{api}}/albums/{{idalbumtest}}/accesses/{{idusertest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| DELETE | {{api}}/tags/{{idtagtest}} |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/tags/{{idtagtest}} |
| DELETE | {{api}}/complaints/types/{{idtypetest}} |
| DELETE | {{api}}/complaints/{{idcomplainttest}} |
| DELETE | {{api}}/users/{{idusertest}}/warnings/{{idwarningtest}} |

## 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 37.

Листинг 37 – Тест 200 OK

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(200);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'key'", function () {

    pm.expect(pm.response.json()).to.have.property("key");

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| GET | {{api}}/albums |
| GET | {{api}}/albums/{{idalbumtest}} |
| POST | {{api}}/albums/{{idalbumtest}} |
| GET | {{api}}/invitation/{{codeinvitationtest}}/album |
| GET | {{api}}/accesses |
| POST | {{api}}/albums/{{idalbumtest}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/thumb/{{orientation}}{{size}}?sign={{signtest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/original?sign={{signtest}} |
| GET | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/download?sign={{signtest}} |
| GET | {{api}}/tags |
| POST | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/tags/{{idtagtest}} |
| GET | {{api}}/complaints/types |
| POST | {{api}}/complaints/types/{{idtypetest}} |
| GET | {{api}}/complaints |
| POST | {{api}}/complaints/{{idcomplainttest}} |
| GET | {{api}}/users/me |
| POST | {{api}}/users/me |
| GET | {{api}}/users |
| GET | {{api}}/users/{{idusertest}} |
| POST | {{api}}/users/{{idusertest}} |
| GET | {{api}}/ |
| GET | {{api}}/settings |
| POST | {{api}}/settings |

## 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 38.

Листинг 38 – Тест 201 Created

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(201);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

**Ресурсы:**

|  |  |
| --- | --- |
| **Метод** | **Эндпоинт** |
| POST | {{api}}/register |
| POST | {{api}}/albums |
| POST | {{api}}/albums/{{idalbumtest}}/invite |
| POST | {{api}}/tags |
| POST | {{api}}/complaints/types |
| POST | {{api}}/albums/{{idalbumtest}}/complaint |
| POST | {{api}}/albums/{{idalbumtest}}/pictures/{{idpicturetest}}/complaint |
| POST | {{api}}/users/{{idusertest}}/warnings |

## 200 Admin

Этот тест проверяет успешную авторизацию за администратора. Он отправляет POST запрос на эндпоинт /login с данными логина и пароля администратора, ожидая получить ответ с кодом состояния 200, временем ответа до 500мс и наличием ключей token, user, name, а также role со значением admin. Код теста представлен в листинге 39.

Листинг 39 – Тест 200 Admin

pm.test("Проверка кода ответа", function () {

    pm.response.to.have.status(200);

});

pm.test("Проверка времени ответа", function () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'token'", function () {

    pm.expect(pm.response.json()).to.have.property("token");

});

pm.test("Проверка наличия ключа 'user'", function () {

    pm.expect(pm.response.json()).to.have.property("user");

});

pm.test("Проверка наличия ключа 'name'", function () {

    pm.expect(pm.response.json().user).to.have.property("name");

});

pm.test("Проверка наличия ключа 'role' и значения admin", function () {

    pm.expect(pm.response.json().user).to.have.property("role", "admin");

});

pm.environment.set('tokenadmintest', pm.response.json().token)

pm.environment.set('idadmintest', pm.response.json().user.id)

## 200 User

Этот тест проверяет успешную авторизацию за обычного пользователя. Он отправляет POST-запрос на эндпоинт /login с данными логина и пароля пользователя, ожидая получить ответ с кодом состояния 200, временем ответа до 500 мс и наличием ключей token, user, name. Код теста представлен в листинге 40.

Листинг 40 – Тест 200 User

pm.test("Проверка кода ответа", function () {

pm.response.to.have.status(200);

});

pm.test("Проверка времени ответа", function () {

pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.test("Проверка наличия ключа 'token'", function () {

pm.expect(pm.response.json()).to.have.property("token");

});

pm.test("Проверка наличия ключа 'user'", function () {

pm.expect(pm.response.json()).to.have.property("user");

});

pm.test("Проверка наличия ключа 'name'", function () {

pm.expect(pm.response.json().user).to.have.property("name");

});

pm.environment.set('tokenusertest', pm.response.json().token)

pm.environment.set('idusertest', pm.response.json().user.id)

# Глава 5. Разработка мобильного приложения администратора

## Введение

Мобильное приложение PicSyncAdmin разработано для администраторов системы PicSync, предоставляя удобный интерфейс для управления пользовательским контентом, обработки жалоб на изображения и контроля параметров сервиса. Основная цель приложения — предоставить администраторам быстрый и интуитивно понятный доступ к функционалу модерации, минимизируя время на обработку жалоб и настройку системы.

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

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.

## Структура проекта

Мобильное приложение PicSyncAdmin построено на платформе .NET MAUI с применением архитектурного паттерна MVVM (Model-View-ViewModel).

Архитектура MVVM

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 файлы:

* Components/ - содержит дополнительные компоненты
  + PinchZoom.cs – компонент для реализации жестов масштабирования и перемещения изображений.
* Converters/ - содержит конвертеры данных
  + CustomDateTimeConverter.cs – преобразует дату и время разных форматов в единый формат.
  + InverseBooleanConverter.cs – инвертирует булевское значение.
  + StatusIsNullConverter.cs – проверяет равно ли значение статуса null.
* Helpers/ - вспомогательные классы для работы с API, сессией и настройками
  + API\_URL.cs – класс для управления URL-адресами API
  + ApiHelper.cs – вспомогательный класс для работы с API, хранит в себе различные исключения.
  + AppSettings – хранит информация о настройках сервера.
  + AuthSession – хранит информацию об аутентификации.
  + Fetch – обрабатывает сетевые запросы
* Models/ - модели данных, используемые в приложении.
  + Response/ - содержит модели для обработки ответов от API
    - AlbumResponse.cs – модель ответа с альбомом.
    - AuthResponse.cs – модель ответа на авторизацию.
    - ComplaintResponse.cs – модель ответа с жалобами.
    - PicturesResponse.cs – модель ответа с изображениями.
    - SettingsResponse.cs – модель ответа с настройками.
    - ValidationErrorResponse – модель для обработки ошибок валидации.
  + Album.cs – модель альбома.
  + Complaints.cs – модель жалобы.
  + Picture.cs – модель изображения.
  + Settings.cs – модель настроек.
  + User.cs – модель пользователя.
* Resources/ - содержит файл ресурсов, такие как изображения и стили
* ViewModels/ - содержит ViewModel'и, реализующие логику взаимодействия с данными.
  + ApiUrlSelectionViewModel - логика выбора URL сервера.
  + HomeViewModel - логика главной страницы.
  + LoginViewModel - логика страницы авторизации.
  + ProfileViewModel - логика страницы профиля.
  + SettingsViewModel - логика страницы настроек.
  + UserContentViewModel - логика работы с контентом пользователя.
* Views/ - содержит XAML-страницы интерфейса.
  + Auth/Login.xaml - страница авторизации.
  + ApiUrlSelectionPage.xaml - страница выбора URL API.
  + Home.xaml - главная страница.
  + Profile.xaml - страница профиля.
  + SettingsPage.xaml - страница настроек.
  + UserContentPage.xaml - страница управления контентом пользователя.
* App.xaml - глобальные настройки приложения, стили.
* AppShell.xaml - конфигурация навигации между страницами.

## Навигация

AppShell определяет основную навигацию в приложении, используя Shell — встроенный механизм навигации в .NET MAUI. В данном случае навигация реализована в виде нижнего меню с помощью TabBar.

Каждый элемент ShellContent отвечает за конкретную страницу приложения:

* ContentTemplate указывает, какая страница будет загружена.
* Title задает название страницы, которое по умолчанию отображается в верхнем левом углу.
* Route определяет маршрут, по которому можно выполнить навигацию программно.

Вся навигация реализована в AppShell.xaml, код которого представлен в Листинг 41.

Листинг 41 – Код навигации AppShell.xaml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <Shell  x:Class="PicsyncAdmin.AppShell"  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  xmlns:local="clr-namespace:PicsyncAdmin.Views"  Shell.FlyoutBehavior="Disabled"  Title="PicsyncAdmin">  <ShellContent  Title="Выбор URL"  ContentTemplate="{DataTemplate local:ApiUrlSelectionPage}"  Route="ApiUrlSelectionPage" />  <ShellContent  Title="Авторизация"  ContentTemplate="{DataTemplate local:Auth.Login}"  Route="LoginPage" />  <ShellContent  Title="Контент пользователя"  ContentTemplate="{DataTemplate local:UserContentPage}"  Route="UserContentPage" />  <TabBar>  <Tab Title="Главная" Icon="house.png">  <ShellContent ContentTemplate="{DataTemplate local:Home}" Route="MainPage"/>  </Tab>  <Tab Title="Профиль" Icon="user.png">  <ShellContent ContentTemplate="{DataTemplate local:Profile}" Route="Profile"/>  </Tab>  <Tab Title="Настройки" Icon="settings.png">  <ShellContent ContentTemplate="{DataTemplate local:SettingsPage}" Route="SettingsPage"/>  </Tab>  </TabBar>  </Shell> |

## Компоненты

Компоненты в PicSyncAdmin представляют собой пользовательские элементы управления, расширяющие стандартные возможности .NET MAUI. Они позволяют добавлять сложное поведение, упрощая работу с интерфейсом и улучшая пользовательский опыт. Компоненты могут включать поддержку мультитач-жестов, кастомные элементы отображения данных и взаимодействия с пользователем. Например, PinchZoom реализует масштабирование и перемещение изображения, обеспечивая удобную навигацию при просмотре контента, код которого представлен в Листинг 42. Использование компонентов помогает организовать код, повышает повторное использование и упрощает поддержку проекта.

Листинг 42 – Код компонента Components/PicnhZoom.cs

using System.Diagnostics;

using PicsyncAdmin.ViewModels;

namespace PicsyncAdmin.Components

{

public class PinchZoom : ContentView

{

private double \_currentScale = 1;

private double \_startScale = 1;

private double \_xOffset = 0;

private double \_yOffset = 0;

public PinchZoom()

{

var pinchGesture = new PinchGestureRecognizer();

pinchGesture.PinchUpdated += PinchUpdated;

GestureRecognizers.Add(pinchGesture);

var panGesture = new PanGestureRecognizer();

panGesture.PanUpdated += OnPanUpdated;

GestureRecognizers.Add(panGesture);

var doubleTapGesture = new TapGestureRecognizer { NumberOfTapsRequired = 2 };

doubleTapGesture.Tapped += DoubleTapped;

GestureRecognizers.Add(doubleTapGesture);

var tapGesture = new TapGestureRecognizer();

tapGesture.Tapped += Tapped;

GestureRecognizers.Add(tapGesture);

}

private async void Tapped(object sender, EventArgs e)

{

UserContentViewModel.Instance.ToggleControlsVisibilityCommand.Execute(null);

}

private void PinchUpdated(object sender, PinchGestureUpdatedEventArgs e)

{

switch (e.Status)

{

case GestureStatus.Started:

\_startScale = Content.Scale;

Content.AnchorX = 0;

Content.AnchorY = 0;

break;

case GestureStatus.Running:

{

\_currentScale += (e.Scale - 1) \* \_startScale;

\_currentScale = Math.Max(1, \_currentScale);

var renderedX = Content.X + \_xOffset;

var deltaX = renderedX / Width;

var deltaWidth = Width / (Content.Width \* \_startScale);

var originX = (e.ScaleOrigin.X - deltaX) \* deltaWidth;

var renderedY = Content.Y + \_yOffset;

var deltaY = renderedY / Height;

var deltaHeight = Height / (Content.Height \* \_startScale);

var originY = (e.ScaleOrigin.Y - deltaY) \* deltaHeight;

var targetX = \_xOffset - originX \* Content.Width \* (\_currentScale - \_startScale);

var targetY = \_yOffset - originY \* Content.Height \* (\_currentScale - \_startScale);

Content.TranslationX = Math.Min(0, Math.Max(targetX, -Content.Width \* (\_currentScale - 1)));

Content.TranslationY = Math.Min(0, Math.Max(targetY, -Content.Height \* (\_currentScale - 1)));

Content.Scale = \_currentScale;

Debug.WriteLine(\_currentScale);

break;

}

case GestureStatus.Completed:

\_xOffset = Content.TranslationX;

\_yOffset = Content.TranslationY;

break;

}

}

public void OnPanUpdated(object sender, PanUpdatedEventArgs e)

{

if (Content.Scale == 1)

{

return;

}

switch (e.StatusType)

{

case GestureStatus.Running:

var newX = e.TotalX \* Scale + \_xOffset;

var newY = e.TotalY \* Scale + \_yOffset;

// Получаем размеры видимой области (вашего ContentView или страницы)

var imageWidth = Width;

var imageHeight = Height;

var DisplayMaxHeight = DeviceDisplay.Current.MainDisplayInfo.Height / DeviceDisplay.Current.MainDisplayInfo.Density;

var DisplayMaxWidth = DeviceDisplay.Current.MainDisplayInfo.Width / DeviceDisplay.Current.MainDisplayInfo.Density;

Debug.WriteLine("DisplayMaxHeight: " + DisplayMaxHeight);

Debug.WriteLine("DisplayMaxWidth: " + DisplayMaxWidth);

// Получаем размеры изображения с учетом масштаба

var scaledWidth = Content.Width \* Content.Scale;

var scaledHeight = Content.Height \* Content.Scale;

// Логируем ключевые переменные

Debug.WriteLine($"imageWidth: {imageWidth}, imageHeight: {imageHeight}");

Debug.WriteLine($"scaledWidth: {scaledWidth}, scaledHeight: {scaledHeight}");

Debug.WriteLine($"newX: {newX}, newY: {newY}");

// Проверяем, можно ли перемещать изображение по горизонтали и вертикали

var canMoveX = scaledWidth > imageWidth && DisplayMaxWidth <= scaledWidth;

var canMoveY = scaledHeight > imageHeight && DisplayMaxHeight <= scaledHeight;

Debug.WriteLine($"canMoveX: {canMoveX}, canMoveY: {canMoveY}");

var emptyWidth = (DisplayMaxWidth - imageWidth) / 2;

var emptyHeight = (DisplayMaxHeight - imageHeight) / 2;

// Ограничение по горизонтали (X)

if (canMoveX)

{

var minX = imageWidth - scaledWidth + emptyWidth; // Левая граница

var maxX = 0 - emptyWidth; // Правая граница

if (newX < minX)

{

newX = minX;

}

if (newX > maxX)

{

newX = maxX;

}

}

else

{

// Если изображение меньше видимой области, центрируем его

newX = (imageWidth - scaledWidth) / 2;

}

// Ограничение по вертикали (Y)

if (canMoveY)

{

// Если изображение больше видимой области, разрешаем перемещение с ограничениями

var minY = imageHeight - scaledHeight + emptyHeight; // Верхняя граница

var maxY = 0 - emptyHeight; // Нижняя граница

Debug.WriteLine($"minY: {minY}, maxY: {maxY}");

if (newY < minY)

{

newY = minY;

}

if (newY > maxY)

{

newY = maxY;

}

}

else

{

// Если перемещение по вертикали запрещено, центрируем изображение по вертикали

Debug.WriteLine("Перемещение по вертикали запрещено. Центрируем изображение по вертикали.");

newY = (imageHeight - scaledHeight) / 2; // Центрируем изображение

}

Debug.WriteLine($"Итоговые значения: newX: {newX}, newY: {newY}");

// Применяем новые координаты

Content.TranslationX = newX;

Content.TranslationY = newY;

break;

case GestureStatus.Completed:

// Сохраняем текущее смещение для следующего перемещения

\_xOffset = Content.TranslationX;

\_yOffset = Content.TranslationY;

break;

case GestureStatus.Started:

break;

case GestureStatus.Canceled:

break;

default:

throw new ArgumentOutOfRangeException();

}

}

public async void DoubleTapped(object sender, EventArgs e)

{

var multiplicator = 2;

\_startScale = Content.Scale;

Content.AnchorX = 0;

Content.AnchorY = 0;

if (\_currentScale == 1) // если это не второй двойной тап, увеличиваем масштаб

{

\_currentScale \*= multiplicator;

}

else // если это второй двойной тап, возвращаем масштаб к 1

{

\_currentScale = 1; // Устанавливаем масштаб в 1

}

var renderedX = Content.X + \_xOffset;

var deltaX = renderedX / Width;

var deltaWidth = Width / (Content.Width \* \_startScale);

var originX = (0.5 - deltaX) \* deltaWidth;

var renderedY = Content.Y + \_yOffset;

var deltaY = renderedY / Height;

var deltaHeight = Height / (Content.Height \* \_startScale);

var originY = (0.5 - deltaY) \* deltaHeight;

var targetX = \_xOffset - originX \* Content.Width \* (\_currentScale - \_startScale);

var targetY = \_yOffset - originY \* Content.Height \* (\_currentScale - \_startScale);

Content.TranslationX = Math.Min(0, Math.Max(targetX, -Content.Width \* (\_currentScale - 1)));

Content.TranslationY = Math.Min(0, Math.Max(targetY, -Content.Height \* (\_currentScale - 1)));

Content.Scale = \_currentScale;

await Task.Delay(10);

\_xOffset = Content.TranslationX;

\_yOffset = Content.TranslationY;

}

}

}

## Конвертеры

Конвертеры в PicSyncAdmin используются для преобразования данных между различными форматами при сериализации и десериализации JSON. Они помогают корректно обрабатывать специфические структуры данных, обеспечивая совместимость с серверными API и удобство работы в клиентском коде. Например, CustomDateTimeConverter предназначен для преобразования дат в несколько возможных форматов, таких как ISO 8601 и форматы с миллисекундами, что гарантирует правильное считывание и запись временных меток, код которого представлен в Листинг 43. Использование конвертеров позволяет гибко настраивать обработку данных и исключать потенциальные ошибки, связанные с различиями в форматах.

Листинг 43 – Код конвертера CustomDateTimeConverter.cs

using System.Globalization;

using System.Text.Json;

using System.Text.Json.Serialization;

namespace PicsyncAdmin.Converters

{

public class CustomDateTimeConverter : JsonConverter<DateTime>

{

private readonly string[] \_formats = new[]

{

"yyyy-MM-ddTHH:mm:ssZ", // ISO 8601 формат с Z (UTC)

"yyyy-MM-ddTHH:mm:ss.fffZ", // Формат с миллисекундами и Z

"yyyy-MM-ddTHH:mm:ss.ffffffZ", // Формат с микросекундами и Z

"yyyy-MM-dd HH:mm:ss", // Формат с пробелом между датой и временем

};

public override DateTime Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)

{

string dateString = reader.GetString();

foreach (var format in \_formats)

{

if (DateTime.TryParseExact(dateString, format, CultureInfo.InvariantCulture, DateTimeStyles.AssumeUniversal, out var date))

{

return date;

}

}

// Если ни один формат не подошел, выбрасываем исключение с дополнительной информацией

throw new JsonException($"Unable to parse date: {dateString}");

}

public override void Write(Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)

{

writer.WriteStringValue(value.ToString("yyyy-MM-ddTHH:mm:ssZ"));

}

}

}

Конвертер InverseBooleanConverter используется для инверсии логических значений (true → false, false → true). Код представлен в Листинг 44. Он применяется в случаях, когда требуется обратное отображение булевских значений в пользовательском интерфейсе. Например, если кнопка должна быть видима при false, но скрыта при true, этот конвертер позволяет автоматически изменять поведение привязки данных. Метод Convert проверяет, является ли входное значение bool, и возвращает его инверсированное значение. Если входные данные не являются булевыми или равны null, по умолчанию возвращается false. Метод ConvertBack выполняет ту же операцию для обратного преобразования, что полезно при двустороннем связывании данных.

Листинг 44 – Код конвертера InverseBooleanConverter.cs

using System.Globalization;

namespace PicsyncAdmin.Converters

{

public class InverseBooleanConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is bool booleanValue)

{

return !booleanValue;

}

return false; // Если значение null или не boolean, возвращаем false

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is bool booleanValue)

{

return !booleanValue;

}

return false;

}

}

}

Конвертер StatusIsNullConverter используется для проверки, является ли переданное значение null. Он возвращает true, если значение отсутствует, и false, если оно установлено (см. Листинг 45). Это удобно, например, для управления отображением элементов интерфейса, которые должны быть скрыты, если статус задан.

Листинг 45 – Код конвертера StatusIsNullConverter.cs

namespace PicsyncAdmin.Converters

{

public class StatusIsNullConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

// Проверяем, что значение статуса равно null

if (value == null)

return true; // Статус равен null

return false; // Статус не равен null

}

public object? ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

return null;

}

}

}

## Помощники

Помощники (Helpers) в PicsyncAdmin представляют собой вспомогательные классы, упрощающие выполнение часто используемых операций. Они обеспечивают удобные методы для работы с данными, взаимодействия с API, обработки изображений и других задач, которые могут потребоваться в разных частях приложения.

Использование помощников позволяет: избежать дублирования кода, разделить логику на отдельные модули, улучшая читаемость и поддержку кода, обеспечить удобный и централизованный доступ к часто используемым функциям.

Класс Fetch представляет собой вспомогательный инструмент для выполнения HTTP-запросов к API. Он включает метод DoAsync, который принимает HTTP-метод, путь запроса, тело запроса (опционально) и управляет обработкой ошибок, сериализацией и десериализацией JSON-данных. Запрос отправляется на URL, который формируется с помощью класса API\_URL, код которого представлен в Листинг 46. Перед отправкой запроса, если имеется AuthSession.Token, он добавляется в заголовки авторизации. В случае ошибки ответ сервера логируется с помощью Debug.WriteLine, а сообщение ошибки передается через setError. Также реализован перегруженный метод DoAsync<T>, который дополнительно десериализует JSON-ответ в указанный тип T. Код данного класса представлен в Листинг 47.

Запрос отправляется на URL, который записан в классе API\_URL.

Листинг 46 – Код класса Helpers/API\_URL.cs

using System;

using System.Diagnostics;

namespace PicsyncAdmin.Helpers

{

public class API\_URL

{

private readonly string \_path;

public API\_URL(string path)

{

\_path = path.TrimStart('/');

}

public static implicit operator string(API\_URL apiUrl)

{

// Всегда берём актуальное значение SelectedUrl

var baseValue = AuthSession.SelectedUrl;

if (string.IsNullOrWhiteSpace(baseValue))

{

throw new InvalidOperationException("Base URL не задан. Проверьте AuthSession.SelectedUrl.");

}

var fullUrl = $"{baseValue}/api/{apiUrl.\_path}";

Debug.WriteLine($"Сформированный URL: {fullUrl}");

return fullUrl;

}

}

}

Листинг 47 – Код класса Helpers/Fetch.cs

using System.Diagnostics;

using System.Text.Json;

using PicsyncAdmin.Models.Response;

namespace PicsyncAdmin.Helpers

{

public static class Fetch

{

private readonly static HttpClient \_httpClient = new();

public static async Task<HttpResponseMessage> DoAsync(

HttpMethod method,

string path,

Action<bool>? setIsFetch = null,

Action<string>? setError = null,

dynamic? body = null,

bool serialize = false

)

{

setIsFetch?.Invoke(true);

setError?.Invoke(null);

try

{

// Формируем запрос

var request = new HttpRequestMessage(method, new API\_URL(path));

if (AuthSession.Token != null)

request.Headers.Add("Authorization", $"Bearer {AuthSession.Token}");

if (body != null)

{

if (serialize)

{

string json = JsonSerializer.Serialize(body);

Debug.WriteLine($"FETCH: REQUEST: JSON: {json}");

request.Content = new StringContent(json, System.Text.Encoding.UTF8, "application/json");

}

else if (body is HttpContent)

{

request.Content = body;

}

else

{

throw new ArgumentException("Body should be of the type HttpContent if serialize is false");

}

}

// Выполняем запрос

var response = await \_httpClient.SendAsync(request);

if (!response.IsSuccessStatusCode)

{

var responseJsonErr = await response.Content.ReadAsStringAsync();

Debug.WriteLine($"FETCH: ERROR: Json: {responseJsonErr}");

try

{

var responseBodyErr = JsonSerializer.Deserialize<ValidationErrorResponse>(responseJsonErr);

setError?.Invoke(responseBodyErr?.Message ?? "Произошла ошибка.");

}

catch (JsonException)

{

setError?.Invoke("Ошибка обработки ответа от сервера.");

}

}

return response;

}

catch (Exception ex)

{

Debug.WriteLine($"FETCH: EXCEPTION: {ex.Message}");

setError?.Invoke(ex.Message);

return new HttpResponseMessage(System.Net.HttpStatusCode.InternalServerError);

}

finally

{

setIsFetch?.Invoke(false);

}

}

public static async Task<(HttpResponseMessage, T?)> DoAsync<T>(

HttpMethod method,

string path,

Action<bool>? setIsFetch = null,

Action<string>? setError = null,

dynamic? body = null,

bool serialize = false

)

{

var response = await DoAsync(method, path, setIsFetch, setError, body, serialize);

try

{

var responseJson = await response.Content.ReadAsStringAsync();

var responseBody = JsonSerializer.Deserialize<T>(responseJson);

return (response, responseBody);

}

catch (JsonException ex)

{

Debug.WriteLine($"FETCH: JSON DESERIALIZATION ERROR: {ex.Message}");

setError?.Invoke("Ошибка обработки данных от сервера.");

return (response, default);

}

}

}

}

ApiHelper — вспомогательный класс для выполнения сетевых запросов с обработкой ошибок, код которого представлен в Листинг 48. Он оборачивает выполнение запроса в try-catch, перехватывает исключения (HttpRequestException, TaskCanceledException и другие), выводит уведомления пользователю при сбоях и возвращает default в случае ошибки. Это позволяет централизовать обработку исключений и избежать дублирования кода в приложении.

Листинг 48 – Код класса Helpers/ApiHelper.cs

namespace PicsyncAdmin.Helpers

{

public static class ApiHelper

{

public static async Task<TResult?> ExecuteRequestAsync<TResult>(Func<Task<TResult>> requestFunc)

{

try

{

return await requestFunc();

}

catch (HttpRequestException httpEx)

{

await Shell.Current.DisplayAlert("Ошибка сети", $"Ошибка HTTP: {httpEx.Message}", "OK");

}

catch (TaskCanceledException)

{

await Shell.Current.DisplayAlert("Ошибка сети", "Превышено время ожидания запроса.", "OK");

}

catch (Exception ex)

{

await Shell.Current.DisplayAlert("Ошибка", $"Ошибка выполнения запроса: {ex.Message}", "OK");

}

return default; // Возвращаем null или значение по умолчанию

}

}

}

AppSettings — вспомогательный класс для хранения и управления настройками, связанными с дисковым пространством и ограничениями загрузки, код которого вы можете увидеть в Листинг 49.Он содержит свойства (UploadDisablePercentage, TotalSpace, FreeSpace, UsedSpace, UsedPercent), которые автоматически вызывают событие SettingsUpdated при изменении, уведомляя подписчиков об обновлениях. Также реализован метод BytesToHuman(), преобразующий байты в человекочитаемый формат (KiB, MiB, GiB и т. д.), что удобно для отображения информации о хранилище в интерфейсе.

Листинг 49 – Код класса Helpers/AppSettings.cs

namespace PicsyncAdmin.Helpers

{

public static class AppSettings

{

public static event Action? SettingsUpdated; // Событие для обновлений

private static int \_uploadDisablePercentage;

public static int UploadDisablePercentage

{

get => \_uploadDisablePercentage;

set

{

if (\_uploadDisablePercentage != value)

{

\_uploadDisablePercentage = value;

OnSettingsUpdated(); // Вызываем событие обновления

}

}

}

private static long \_totalSpace;

public static long TotalSpace

{

get => \_totalSpace;

set

{

if (\_totalSpace != value)

{

\_totalSpace = value;

OnSettingsUpdated(); // Вызываем событие обновления

}

}

}

private static long \_freeSpace;

public static long FreeSpace

{

get => \_freeSpace;

set

{

if (\_freeSpace != value)

{

\_freeSpace = value;

OnSettingsUpdated(); // Вызываем событие обновления

}

}

}

private static long \_usedSpace;

public static long UsedSpace

{

get => \_usedSpace;

set

{

if (\_usedSpace != value)

{

\_usedSpace = value;

OnSettingsUpdated(); // Вызываем событие обновления

}

}

}

private static long \_usedPercent;

public static long UsedPercent

{

get => \_usedPercent;

set

{

if (\_usedPercent != value)

{

\_usedPercent = value;

OnSettingsUpdated(); // Вызываем событие обновления

}

}

}

// Метод для вызова события

private static void OnSettingsUpdated()

{

SettingsUpdated?.Invoke(); // Уведомляем подписчиков

}

// Метод для преобразования байтов в человекочитаемый формат

public static string BytesToHuman(long bytes)

{

string[] units = { "B", "KiB", "MiB", "GiB", "TiB", "PiB" };

if (bytes < 0)

{

return "0 B";

}

int pow = (int)Math.Floor(bytes > 0 ? Math.Log(bytes) / Math.Log(1024) : 0);

pow = Math.Min(pow, units.Length - 1);

double size = bytes / Math.Pow(1024, pow);

return $"{size:0.###} {units[pow]}";

}

}

}

AuthSession — вспомогательный класс для управления сессией пользователя, включая хранение токена, URL API и информации о пользователе, код которого представлен в Листинг 50. Реализует свойства Token и SelectedUrl, которые автоматически сохраняются и загружаются через Preferences. Метод SaveUrl() позволяет сохранять URL API, ClearSession() — очищает данные при выходе, а LoadSession() восстанавливает сессию из Preferences и перенаправляет на нужную страницу (LoginPage или ApiUrlSelectionPage). CheckAuthFromServer() проверяет действительность токена через API, перенаправляя пользователя в зависимости от результата.

Листинг 50 – Код класса Helpers/AuthSession.cs

using PicsyncAdmin.Models;

using System.Diagnostics;

using System.Text.Json;

namespace PicsyncAdmin.Helpers

{

public static class AuthSession

{

public static event Action<string?>? OnUrlChanged;

public static User? User { get; set; }

private static string? \_token = null;

public static string? Token

{

get

{

var token = \_token;

if (token == null)

{

token = Preferences.Get("token", string.Empty);

}

if (token == string.Empty)

{

return null;

}

\_token = token;

return token;

}

set

{

\_token = value;

if (value == null)

Preferences.Remove("token");

else

Preferences.Set("token", value);

}

}

private static string? \_selectedUrl = null;

public static string? SelectedUrl

{

get => \_selectedUrl ??= Preferences.Get("SelectedUrl", string.Empty);

set

{

\_selectedUrl = value;

if (value == null)

Preferences.Remove("SelectedUrl");

else

Preferences.Set("SelectedUrl", value);

}

}

public static void SaveUrl(string url)

{

if (string.IsNullOrWhiteSpace(url))

throw new ArgumentException("URL не может быть пустым", nameof(url));

SelectedUrl = url;

Preferences.Set("SelectedUrl", url);

Debug.WriteLine($"URL сохранён в AuthSession: {SelectedUrl}");

}

// Метод для очистки данных при выходе

public static void ClearSession()

{

User = null;

Token = null;

SelectedUrl = null;

}

// Метод для восстановления данных из Preferences

public static async Task LoadSession()

{

// Восстанавливаем SelectedUrl

SelectedUrl = Preferences.Get("SelectedUrl", string.Empty);

Debug.WriteLine($"SelectedUrl из Preferences: {SelectedUrl}");

if (string.IsNullOrWhiteSpace(SelectedUrl))

{

await Shell.Current.GoToAsync("///ApiUrlSelectionPage");

return;

}

// Восстанавливаем токен

Token = Preferences.Get("token", null);

// Восстанавливаем пользователя только если в Preferences есть данные

var userJson = Preferences.Get("User", null);

if (!string.IsNullOrEmpty(userJson))

{

try

{

User = JsonSerializer.Deserialize<User>(userJson);

await CheckAuthFromServer();

}

catch (Exception)

{

ClearSession();

await Shell.Current.GoToAsync("///LoginPage");

}

}

else

{

await Shell.Current.GoToAsync("///LoginPage");

}

}

//Проверка работоспособности сервера и ещё чето слово умное ну ти пон

public static async Task CheckAuthFromServer()

{

try

{

var response = await Fetch.DoAsync(HttpMethod.Get, "/users/me");

if (response.IsSuccessStatusCode)

{

Debug.WriteLine("AUTH: Пользователь авторизован");

await Shell.Current.GoToAsync("//MainPage");

}

else if ((int)response.StatusCode >= 500)

{

Debug.WriteLine("AUTH: Ошибка сервера");

ClearSession();

await Shell.Current.GoToAsync("///ApiUrlSelectionPage");

}

else

{

Debug.WriteLine("AUTH: Токен недействителен или истёк");

ClearSession();

await Shell.Current.GoToAsync("///LoginPage");

}

}

catch (Exception ex)

{

Debug.WriteLine($"AUTH: Ошибка проверки авторизации: {ex.Message}");

ClearSession();

await Shell.Current.GoToAsync("///ApiUrlSelectionPage");

}

}

}

}

## Модели

Модели в проекте представляют собой классы, описывающие структуру данных, получаемых и отправляемых через API. В качестве примера приведён класс User, код которого представлен в Листинг 51, содержащий свойства, соответствующие полям таблицы users, такие как Id, Name, Login, Password, Role, а также временные метки CreatedAt и UpdatedAt. Для сериализации и десериализации используется JsonPropertyName. Другой пример — AuthResponse, который описывает ответ сервера при авторизации, включая объект пользователя и токен, код которого можно увидеть в Листинг 52. Остальные модели устроены аналогично и содержат необходимые для работы свойства.

Листинг 51 – Код класса Models/User.cs

using System.Text.Json.Serialization;

namespace PicsyncAdmin.Models

{

// Класс описывающий таблицу users

public class User

{

[JsonPropertyName("id")] public ulong? Id { get; set; }

[JsonPropertyName("name")] public string? Name { get; set; }

[JsonPropertyName("login")] public string? Login { get; set; }

[JsonPropertyName("password")] public string? Password { get; set; }

[JsonPropertyName("role")] public string? Role { get; set; }

[JsonPropertyName("is\_banned")] public bool? IsBanned { get; set; }

[JsonPropertyName("remember\_token")] public string? RememberToken { get; set; }

[JsonPropertyName("created\_at")] public DateTime? CreatedAt { get; set; }

[JsonPropertyName("updated\_at")] public DateTime? UpdatedAt { get; set; }

}

}

Листинг 52 – Код класса Models/Response/AuthResponse.cs

using System.Text.Json.Serialization;

namespace PicsyncAdmin.Models.Response

{

// Класс описывающий ответ модуля Auth

public class AuthResponse

{

[JsonPropertyName("user")] public User User { get; set; }

[JsonPropertyName("token")] public string Token { get; set; }

}

}

## Реализация страниц

Везде привязка данных к View выполняется в файле xaml.cs. В качестве примера рассмотрим привязку ViewModel к странице авторизации (см. Листинг 53).

Листинг 53 – Привязка ViewModel к View

using PicsyncAdmin.ViewModels;

namespace PicsyncAdmin.Views.Auth;

public partial class Login : ContentPage

{

public Login()

{

InitializeComponent();

BindingContext = new LoginViewModel(); // Привязываем ViewModel

}

}

### Страница выбора URL

Данная страница позволяет пользователю выбрать API URL, с которым будет работать приложение. Можно ввести новый URL или выбрать сохранённый ранее. Если пользователь не авторизован, то она является главной. Код разметки можно увидеть в Листинг 54.

Листинг 54 – Разметка страницы Views/ApiUrlSelectionPage.xaml

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:vm="clr-namespace:PicsyncAdmin.ViewModels"

x:Class="PicsyncAdmin.Views.ApiUrlSelectionPage"

Title="Выбор API URL">

<ContentPage.BindingContext>

<vm:ApiUrlSelectionViewModel />

</ContentPage.BindingContext>

<VerticalStackLayout Padding="20" Spacing="25">

<!-- Кнопки управления -->

<VerticalStackLayout Spacing="15">

<Button Text="Добавить новый URL"

Command="{Binding SaveApiUrlCommand}"

HorizontalOptions="FillAndExpand" />

<Button Text="Выбрать существующий URL"

Command="{Binding ShowApiUrlSelectionCommand}"

HorizontalOptions="FillAndExpand" />

</VerticalStackLayout>

</VerticalStackLayout>

</ContentPage>

Итоговый интерфейс страницы выбора API URL представлен на Рисунок 44. Он содержит две основные кнопки: "Добавить новый URL" и "Выбрать существующий URL", позволяя пользователю сохранить новый адрес или выбрать ранее сохраненный. Этот экран служит важным этапом настройки приложения перед началом работы.

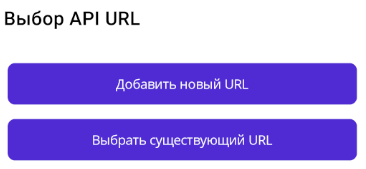


Рисунок 44 – Страница выбора API URL

Класс ApiUrlSelectionViewModel, представленный в Листинг 55, реализует логику управления выбором API URL, позволяя пользователю сохранять, выбирать и проверять работоспособность серверов. Он использует Preferences для хранения списка URL и предоставляет команды для их обработки. Валидация URL предотвращает сохранение некорректных адресов, а механизм проверки доступности сервера (TestUriAp) позволяет убедиться, что выбранный URL активен перед переходом к авторизации.

Листинг 55 – Код ViewModels/ApiUrlSelectionViewModel.cs

using System.Collections.ObjectModel;

using System.Diagnostics;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using PicsyncAdmin.Helpers;

namespace PicsyncAdmin.ViewModels

{

public partial class ApiUrlSelectionViewModel : ObservableObject

{

// Список сохранённых URL

[ObservableProperty]

private ObservableCollection<string> savedApiUrls = new ObservableCollection<string>();

// Поле для ввода URL

[ObservableProperty] public string apiUrlEntry;

// Выбранный URL

[ObservableProperty] public string selectedApiUrl;

// Конструктор

public ApiUrlSelectionViewModel()

{

LoadSavedApiUrls();

}

[RelayCommand]

public async Task ShowApiUrlSelection()

{

SelectedApiUrl = await Shell.Current.DisplayActionSheet("Выберите API URL", "Отмена", null, SavedApiUrls.ToArray());

if (!string.IsNullOrEmpty(SelectedApiUrl) && SelectedApiUrl != "Отмена")

{

if (IsValidUrl(SelectedApiUrl))

{

AuthSession.SaveUrl(SelectedApiUrl);

Debug.Write(SelectedApiUrl ,"Во время выбора и сохранения токена на странице с выбором апи");

await TestUriAp();

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Некорректный URL. Выберите другой.", "OK");

}

}

}

private void LoadSavedApiUrls()

{

var savedUrls = Preferences.Get("savedApiUrls", string.Empty);

if (!string.IsNullOrEmpty(savedUrls))

{

var urls = savedUrls.Split(';').ToList();

SavedApiUrls = new ObservableCollection<string>(urls);

}

}

[RelayCommand]

public async Task SaveApiUrl()

{

ApiUrlEntry = await Shell.Current.DisplayPromptAsync("Сервер", "Введите URL", "OK", "Отмена", "https://example.com");

if (!string.IsNullOrEmpty(ApiUrlEntry))

{

if (IsValidUrl(ApiUrlEntry))

{

var result = await Shell.Current.DisplayActionSheet(

"Выберите действие", // Заголовок

"Отмена", // Кнопка "Отмена"

null, // Нет текста для cancel кнопки

"Сохранить и выбрать", // Первый вариант

"Сохранить" // Второй вариант

);

if (result == "Сохранить и выбрать")

{

await SaveUrlAndSelect();

}

else if (result == "Сохранить")

{

SaveUrlOnly();

}

}

else

{

// Если URL некорректный, показываем сообщение об ошибке

await Shell.Current.DisplayAlert("Ошибка", "URL должен иметь стандартный формат и не оканчиваться на /", "OK");

}

}

}

// Функция для проверки правильности URL

private static bool IsValidUrl(string url)

{

// Регулярное выражение для проверки URL без api и up

var regex = @"^https?://(?!.\*(/$|/api$|/up$)).\*$";

return System.Text.RegularExpressions.Regex.IsMatch(url, regex);

}

// Функция для сохранения и одновременного выбора Url

[RelayCommand]

private async Task SaveUrlAndSelect()

{

SavedApiUrls.Add(ApiUrlEntry);

var urls = SavedApiUrls.ToList();

Preferences.Set("savedApiUrls", string.Join(";", urls));

AuthSession.SaveUrl(ApiUrlEntry);

await TestUriAp();

}

// Функция для сохранения Url

private void SaveUrlOnly()

{

SavedApiUrls.Add(ApiUrlEntry);

var urls = SavedApiUrls.ToList();

Preferences.Set("savedApiUrls", string.Join(";", urls));

}

// Функция для проверки работоспособности Url

[RelayCommand]

public static async Task TestUriAp()

{

var url = AuthSession.SelectedUrl;

if (string.IsNullOrEmpty(url))

{

Debug.WriteLine("SelectedUrl пустой или не задан.");

await Shell.Current.DisplayAlert("Ошибка", "Базовый URL не задан. Выберите сервер снова.", "OK");

await Shell.Current.GoToAsync("//ApiUrlSelectionPage");

return;

}

var timeout = TimeSpan.FromSeconds(5);

var isUp = await ApiHelper.ExecuteRequestAsync(async () =>

{

using var cts = new CancellationTokenSource(timeout);

var client = new HttpClient();

var response = await client.GetAsync($"{url}/up", cts.Token);

if (response.IsSuccessStatusCode)

{

return true;

}

if ((int)response.StatusCode == 502 || (int)response.StatusCode == 504)

{

throw new Exception("Сервер временно недоступен (502/504).");

}

throw new Exception($"Ошибка: {response.StatusCode}");

});

if (isUp)

{

await Shell.Current.GoToAsync("///LoginPage");

}

else

{

var retryChoice = await Shell.Current.DisplayActionSheet(

"Сервер недоступен. Попробовать снова или выбрать другой?",

"Отмена",

null,

"Попробовать снова",

"Выбрать другой сервер"

);

if (retryChoice == "Попробовать снова")

{

await TestUriAp(); // Рекурсия для повторной проверки

}

else if (retryChoice == "Выбрать другой сервер")

{

AuthSession.SelectedUrl = null;

await Shell.Current.GoToAsync("//ApiUrlSelectionPage");

}

}

}

}

}

### Страница авторизации

Данная страница позволяет пользователю авторизоваться в системе, введя логин и пароль. Также отображается выбранный сервер, к которому будет выполняться запрос на вход, с возможностью его смены. При нажатии на кнопку входа происходит валидация введённых данных и отправка запроса к API. Код разметки представлен в Листинг 56.

Листинг 56 – Разметка страницы Views/Auth/Login.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:vm="clr-namespace:PicsyncAdmin.ViewModels"

x:Class="PicsyncAdmin.Views.Auth.Login"

Title="Авторизация">

<VerticalStackLayout Padding="20" Spacing="25">

<!-- Информация о сервере -->

<VerticalStackLayout Spacing="5">

<Label FontSize="Medium" Text="Выбранный сервер: "/>

<Label Text="{Binding SelectedServer}"

FontAttributes="Bold"

Padding="0, 0, 10, 0"/>

<Border Padding="10" HorizontalOptions="End" StrokeThickness="2">

<Label Text="Сменить сервер"

FontAttributes="Bold"

FontSize="Subtitle"

HorizontalOptions="End">

<Label.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectServerCommand}" />

</Label.GestureRecognizers>

</Label>

</Border>

</VerticalStackLayout>

<!-- Поля ввода -->

<VerticalStackLayout Spacing="15">

<Entry Placeholder="Введите логин"

Text="{Binding Login}" />

<Entry Placeholder="Введите пароль"

Text="{Binding Password}"

IsPassword="True" />

</VerticalStackLayout>

<!-- Кнопка входа -->

<Button Text="Войти"

Command="{Binding TryLoginCommand}"

HorizontalOptions="FillAndExpand" />

</VerticalStackLayout>

</ContentPage>

Итоговый интерфейс страницы авторизации включает в себя поля для ввода логина и пароля, текстовое отображение текущего сервера, кнопку смены сервера и кнопку входа (см. Рисунок 45). Все элементы адаптированы для удобного взаимодействия и автоматически обновляются в зависимости от состояния приложения.

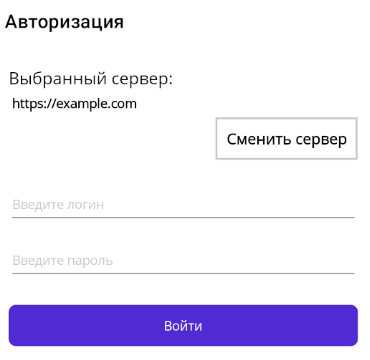


Рисунок 45 – Страница авторизации

Класс LoginViewModel отвечает за обработку пользовательского ввода и выполнение логики авторизации. В нём реализованы свойства для хранения данных входа, механизм отправки запроса к API и обработки ответа, а также команда смены сервера (см. Листинг 57). Использование CommunityToolkit.Mvvm позволяет автоматически обновлять интерфейс при изменении данных.

Листинг 57 – Код ViewModels/LoginViewModel.cs

using System.Diagnostics;

using PicsyncAdmin.Helpers;

using PicsyncAdmin.Models.Response;

using CommunityToolkit.Mvvm.Input;

using CommunityToolkit.Mvvm.ComponentModel;

using ObservableObject = CommunityToolkit.Mvvm.ComponentModel.ObservableObject;

using System.Text.Json;

namespace PicsyncAdmin.ViewModels

{

public partial class LoginViewModel : ObservableObject

{

[ObservableProperty] private string? login;

[ObservableProperty] private string? password;

[ObservableProperty] private string? error;

[ObservableProperty] private string? selectedServer = AuthSession.SelectedUrl;

[ObservableProperty]

[NotifyCanExecuteChangedFor(nameof(TryLoginCommand))]

private bool isFetch = false;

public LoginViewModel()

{

AuthSession.OnUrlChanged += url => SelectedServer = url;

SelectedServer = AuthSession.SelectedUrl; // Инициализация текущим значением

}

[RelayCommand]

public static async Task SelectServer()

{

AuthSession.ClearSession();

await Shell.Current.GoToAsync("//ApiUrlSelectionPage");

}

[RelayCommand]

private async Task TryLogin()

{

if (string.IsNullOrWhiteSpace(Login) || string.IsNullOrWhiteSpace(Password))

{

await Shell.Current.DisplayAlert("Ошибка", "Введите логин и пароль", "OK");

return;

}

IsFetch = true;

try

{

var loginData = new { login = Login, password = Password };

var (response, authResponse) = await Fetch.DoAsync<AuthResponse>(

HttpMethod.Post,

"login",

setIsFetch: isFetching => IsFetch = isFetching,

setError: message => Error = message,

body: loginData,

serialize: true

);

if (Error != null)

{

await Shell.Current.DisplayAlert("Ошибка", Error, "OK");

return;

}

if (authResponse == null || !response.IsSuccessStatusCode)

{

await Shell.Current.DisplayAlert("Ошибка", "Неправильный логин или пароль", "OK");

return;

}

if(authResponse.User.Role == "admin")

{

// Сохраняем пользователя и токен в AuthSession

AuthSession.User = authResponse.User;

AuthSession.Token = authResponse.Token;

// Сохраняем в Preferences

Preferences.Set("User", JsonSerializer.Serialize(authResponse.User));

Preferences.Set("Token", authResponse.Token);

await Shell.Current.GoToAsync("//MainPage");

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Доступ запрещен", "OK");

}

}

catch (Exception ex)

{

await Shell.Current.DisplayAlert("Ошибка", $"Что-то пошло не так: {ex.Message}", "OK");

}

}

}

}

### Главная страница

Данная страница позволяет администратору просматривать текущие жалобы пользователей, следить за загрузкой дискового пространства и загружать дополнительные данные при необходимости. На главной странице отображается прогресс-бар, демонстрирующий использование хранилища, а также список альбомов с жалобами, включающий миниатюры изображений, информацию о типе жалобы и пользователе, отправившем её. Для удобства предусмотрены кнопки обновления списка жалоб и загрузки дополнительных данных. Код разметки представлен в Листинг 58.

Листинг 58 – Разметка страницы Views/Home.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:PicsyncAdmin.ViewModels"

x:Class="PicsyncAdmin.Views.Home"

Title="Главная">

<ScrollView>

<VerticalStackLayout Padding="10">

<!-- Прогресс бар, отображающий использование пространства -->

<StackLayout Grid.Row="2" Spacing="10" Padding="20">

<Label Text="Использовано пространства"

HorizontalOptions="Center"

FontSize="16"

FontAttributes="Bold" />

<ProgressBar Progress="{Binding UsedPercent}"

HorizontalOptions="FillAndExpand" />

<StackLayout Orientation="Horizontal" Spacing="5" HorizontalOptions="Center">

<Label Text="{Binding FreeSpaceHumanReadable}"

VerticalOptions="Center"

FontSize="12" />

<Label Text="/"

VerticalOptions="Center"

FontSize="12" />

<Label Text="{Binding TotalSpaceHumanReadable}"

VerticalOptions="Center"

FontSize="12" />

<Label Text="{Binding UsedPercentDisplay, StringFormat='{0:F0}%'}"

HorizontalOptions="Center"

FontSize="14"

VerticalOptions="Center" />

</StackLayout>

</StackLayout>

<!-- Заголовок -->

<Label Text="Нерассмотренные жалобы"

HorizontalOptions="Center"

FontSize="20"

Padding="20" />

<!-- Кнопка обновления -->

<ImageButton HeightRequest="50"

WidthRequest="5000"

Source="refresh.png"

Command="{Binding ResetComplaintsCommand}" />

<Label Text="{Binding StatusMessage}"

FontSize="Medium"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="{Binding IsFetch}" />

<!-- Список жалоб -->

<CollectionView ItemsSource="{Binding Albums}">

<CollectionView.ItemTemplate>

<DataTemplate>

<Frame Padding="10" Margin="5">

<VerticalStackLayout>

<Grid ColumnDefinitions="\*, Auto" RowDefinitions="Auto, Auto">

<!-- Информация об альбоме -->

<Label Text="{Binding Name}"

FontSize="16"

FontAttributes="Bold"

Grid.Row="0"

Grid.Column="0" />

<Label Text="{Binding ComplaintsCount, StringFormat='Жалоб: {0}'}"

FontSize="12"

Grid.Row="1"

Grid.Column="0" />

<!-- Кнопка с параметром -->

<Button Text="Просмотреть полностью"

HorizontalOptions="End"

Grid.Row="0"

Grid.Column="1"

Grid.RowSpan="2"

Command="{Binding Source={RelativeSource AncestorType={x:Type local:HomeViewModel}}, Path=NavigateToUserContentPageCommand}"

CommandParameter="{Binding Album}" />

</Grid>

<!-- Список жалоб -->

<CollectionView ItemsSource="{Binding Complaints}">

<CollectionView.ItemsLayout>

<LinearItemsLayout Orientation="Horizontal" />

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<VerticalStackLayout Padding="5" MaximumWidthRequest="200">

<!-- Картинка жалобы -->

<Image MaximumWidthRequest="200" MaximumHeightRequest="200">

<Image.Source>

<UriImageSource Uri="{Binding Picture.Path}" CacheValidity="10:00:00:00" />

</Image.Source>

</Image>

<!-- Информация о жалобе -->

<StackLayout Spacing="2">

<Label Text="{Binding FromUser.Name, StringFormat='Пожаловался: {0}'}" FontSize="12" />

<Label Text="{Binding Type, StringFormat='Тип жалобы: {0}'}" FontSize="12" />

<Label Text="{Binding Description, StringFormat='Описание: {0}'}" FontSize="12"

LineBreakMode="WordWrap" MaxLines="2" />

<Label Text="{Binding StatusDescription}" FontSize="12" />

</StackLayout>

</VerticalStackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</VerticalStackLayout>

</Frame>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

<!-- Кнопка загрузки дополнительных данных -->

<Button Text="Загрузить ещё"

IsVisible="{Binding CanLoadMore}"

Command="{Binding LoadComplaintsCommand}" />

</VerticalStackLayout>

</ScrollView>

</ContentPage>

Итоговый интерфейс содержит удобную навигацию, обеспечивая возможность просматривать альбомы с жалобами и переходить к их детальному рассмотрению. Каждая жалоба сопровождается изображением, информацией о пользователе и описанием проблемы. Визуально страница выполнена в минималистичном стиле с использованием прогресс-бара, кнопок и списка карточек с жалобами (см. Рисунок 46).



Рисунок 46 – Главная страница

Класс HomeViewModel управляет логикой работы страницы, включая загрузку данных, обработку ошибок и навигацию. Он содержит свойства, отвечающие за отображение прогресса использования хранилища, список жалоб, статус загрузки и команды для обновления данных. При помощи RelayCommand реализованы команды для загрузки жалоб, обновления списка и перехода на страницу детального просмотра альбомов. Запросы к API выполняются через Fetch, а данные обрабатываются и отображаются в привязанных элементах интерфейса.

Листинг 59 – Код ViewModels/HomeViewModel.cs

using System.Collections.ObjectModel;

using System.Diagnostics;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using Newtonsoft.Json;

using PicsyncAdmin.Helpers;

using PicsyncAdmin.Models;

using PicsyncAdmin.Models.Response;

using PicsyncAdmin.Models.Response.PicsyncAdmin.Models.Response;

using PicsyncAdmin.Views;

using ObservableObject = CommunityToolkit.Mvvm.ComponentModel.ObservableObject;

namespace PicsyncAdmin.ViewModels

{

public partial class HomeViewModel : ObservableObject

{

public static HomeViewModel? Instance { get; private set; }

private readonly User? \_user = AuthSession.User;

private readonly string? \_token = AuthSession.Token;

private readonly HttpClient \_httpClient;

[ObservableProperty]

[NotifyCanExecuteChangedFor(nameof(LoadComplaintsCommand))]

private bool isFetch = false;

[ObservableProperty] private string statusMessage;

[ObservableProperty] private double usedPercent;

[ObservableProperty] private string usedSpaceHumanReadable;

[ObservableProperty] private string totalSpaceHumanReadable;

[ObservableProperty] private string freeSpaceHumanReadable;

[ObservableProperty] private double usedPercentDisplay;

[ObservableProperty] public bool canLoadMore = false;

[ObservableProperty] private int currentPage = 1;

public ObservableCollection<AlbumComplaintData> Albums { get; set; } = new ObservableCollection<AlbumComplaintData>();

public HomeViewModel()

{

Instance = this;

\_httpClient = new HttpClient();

// Подписка на событие обновления настроек

AppSettings.SettingsUpdated += OnSettingsUpdated;

// Загрузка настроек

Task.Run(() => LoadSettings());

Task.Run(() => LoadComplaints());

}

[RelayCommand(CanExecute = nameof(CanLoadComplaints))]

public async Task LoadComplaints()

{

try

{

StatusMessage = "Загрузка жалоб...";

IsFetch = true;

// Отправляем запрос на сервер

var response = await Fetch.DoAsync(

HttpMethod.Get,

$"/complaints?status=null&limit\_per\_album=3&page={CurrentPage}&limit=3",

setError: msg => Debug.WriteLine($"Error: {msg}")

);

IsFetch = false;

if (!response.IsSuccessStatusCode)

{

StatusMessage = "Ошибка загрузки жалоб. Попробуйте снова.";

return;

}

var responseString = await response.Content.ReadAsStringAsync();

var responseObject = JsonConvert.DeserializeObject<ComplaintResponse>(responseString);

if (responseObject == null || responseObject.Albums == null)

{

StatusMessage = "Жалоб нет.";

return;

}

StatusMessage = ""; // Скрываем статус, если жалобы есть

// Устанавливаем значение CanLoadMore для определения возможности подгрузки данных

CanLoadMore = responseObject.Total > responseObject.Page \* responseObject.Limit;

CurrentPage++;

// Обрабатываем данные из API

foreach (var albumData in responseObject.Albums)

{

if (albumData == null || albumData.Complaints == null)

{

continue; // Пропускаем null значения

}

// Создаем объект Album, если он отсутствует

albumData.Album ??= new Album

{

Id = albumData.Id,

Name = albumData.Name,

User = albumData.Complaints?.FirstOrDefault()?.FromUser ?? new User

{

Id = 0,

Name = "Неизвестный пользователь"

}

};

// Обрабатываем пути для картинок в жалобах, если они существуют

foreach (var complaint in albumData.Complaints?.Where(c => c.Picture != null) ?? Enumerable.Empty<Complaint>())

{

// Убедитесь, что albumData.Album не равен null

if (albumData.Album != null && complaint.Picture != null)

{

complaint.Picture.Path = new API\_URL($"/albums/{albumData.Album.Id}/pictures/{complaint.Picture.Id}/thumb/q480?sign={complaint.Sign}");

}

}

// Добавляем объект albumData в коллекцию Albums

Albums.Add(albumData);

}

Debug.WriteLine($"Successfully loaded page {responseObject.Page} with {responseObject.Albums.Count} albums.");

}

catch (System.Text.Json.JsonException ex)

{

StatusMessage = "Ошибка обработки данных.";

}

catch (Exception)

{

StatusMessage = "Произошла ошибка. Попробуйте снова.";

}

}

private bool CanLoadComplaints() =>

!IsFetch;

[RelayCommand]

private async Task NavigateToUserContentPage(Album album)

{

try

{

await Shell.Current.Navigation.PushAsync(new UserContentPage(album));

}

catch (Exception e)

{

await Shell.Current.DisplayAlert("Error", e.Message, "OK");

throw;

}

}

[RelayCommand]

public async Task ResetComplaints()

{

Albums.Clear(); // Очистка списка жалоб

CurrentPage = 1; // Сброс текущей страницы

CanLoadMore = false; // Сброс состояния загрузки дополнительных данных

await LoadComplaints();

}

// Обработчик события обновления

private void OnSettingsUpdated()

{

UsedPercentDisplay = AppSettings.UsedPercent;

UsedPercent = (double)AppSettings.UsedPercent / 100;

UsedSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.UsedSpace);

TotalSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.TotalSpace);

FreeSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.FreeSpace);

}

public async Task LoadSettings()

{

try

{

// Отправляем запрос через Fetch.DoAsync

var response = await Fetch.DoAsync(

HttpMethod.Get,

"/settings", // Путь запроса

setError: msg => Debug.WriteLine($"Error: {msg}") // Обработчик ошибок

);

var responseString = await response.Content.ReadAsStringAsync();

var settingsResponse = System.Text.Json.JsonSerializer.Deserialize<SettingsResponse>(responseString);

if (settingsResponse == null)

{

bool accept = await Shell.Current.DisplayAlert("Ошибка загрузки настроек", "Желаете попробовать снова?", "Да", "Нет");

if (accept)

{

await LoadSettings();

}

return;

}

// Обновление значений настроек

AppSettings.UploadDisablePercentage = settingsResponse.Settings.UploadDisablePercentage;

AppSettings.TotalSpace = settingsResponse.Space.Total;

AppSettings.FreeSpace = settingsResponse.Space.Free;

AppSettings.UsedSpace = settingsResponse.Space.Used;

AppSettings.UsedPercent = settingsResponse.Space.UsedPercent;

// Вызываем событие обновления настроек

AppSettings.SettingsUpdated += OnSettingsUpdated;

}

catch (Exception ex)

{

await Shell.Current.DisplayAlert($"Error loading settings", ex.Message, "OK");

}

}

}

}

### Страница контента пользователя

Этот экран позволяет администраторам управлять контентом пользователей, включая просмотр изображений альбома, удаление изображений, выдачу предупреждений, блокировку пользователей и отклонение жалоб. Пользователь может просматривать изображения в полноэкранном режиме, а также перемещаться между ними. Отображается информация о количестве жалоб, владельце альбома и общем количестве изображений. Код разметки представлен в Листинг 60.

Листинг 60 – Разметка страницы Views/UserContentPage.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:vm="clr-namespace:PicsyncAdmin.ViewModels"

xmlns:local="clr-namespace:PicsyncAdmin.Converters"

xmlns:pinch="clr-namespace:PicsyncAdmin.Components"

x:Class="PicsyncAdmin.Views.UserContentPage"

xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

Title="Контент пользователя">

<ContentPage.Resources>

<ResourceDictionary>

<local:InverseBooleanConverter x:Key="InverseBooleanConverter" />

</ResourceDictionary>

</ContentPage.Resources>

<Grid>

<!-- Основной контент -->

<ScrollView IsVisible="{Binding IsFullScreenVisible, Converter={StaticResource InverseBooleanConverter}}">

<VerticalStackLayout Spacing="10" Padding="20">

<Button Text="Создать предупреждение" Command="{Binding IssueWarningCommand}" />

<Button Text="Забанить" Command="{Binding BlockUserCommand}" />

<Label Text="{Binding AlbumName, StringFormat='Изображения альбома: {0}'}" FontSize="Subtitle" IsVisible="{Binding IsFetch, Converter={StaticResource InverseBooleanConverter}}" />

<Label Text="{Binding UserName, StringFormat='Владелец альбома: {0}'}" FontSize="Subtitle" IsVisible="{Binding IsFetch, Converter={StaticResource InverseBooleanConverter}}" />

<Label Text="{Binding ComplaintsCount, StringFormat='Жалоб на альбом: {0}'}" FontSize="Subtitle" IsVisible="{Binding IsFetch, Converter={StaticResource InverseBooleanConverter}}" />

<Label Text="{Binding PicturesCount, StringFormat='Изображений в альбоме: {0}'}" FontSize="Subtitle" IsVisible="{Binding IsFetch, Converter={StaticResource InverseBooleanConverter}}" />

<Label Text="{Binding StatusMessage}" FontSize="Medium" HorizontalOptions="Center" VerticalOptions="Center" IsVisible="{Binding IsFetch}" />

<!-- Коллекция изображений -->

<CollectionView ItemsSource="{Binding AlbumPictures}" x:Name="ImageCollection">

<CollectionView.ItemsLayout>

<GridItemsLayout Orientation="Vertical" HorizontalItemSpacing="5" x:Name="AdaptiveGridLayout"/>

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid>

<!-- Картинка -->

<Image Source="{Binding Path}" Aspect="AspectFit" Margin="5">

<Image.GestureRecognizers>

<TapGestureRecognizer Command="{Binding Source={RelativeSource AncestorType={x:Type vm:UserContentViewModel}}, Path=ViewImageCommand}" CommandParameter="{Binding .}" />

</Image.GestureRecognizers>

</Image>

<!-- Оверлей с счетчиком жалоб -->

<Label Text="{Binding ComplaintCount}"

TextColor="White"

HorizontalOptions="End"

FontSize="Medium"

FontAttributes="Bold"

VerticalOptions="Start" Margin="5"

BackgroundColor="#96000000">

<Label.Shadow>

<Shadow Brush="Black" Offset="1,0" Opacity="20" Radius="5" />

</Label.Shadow>

<Label.Triggers>

<DataTrigger TargetType="Label" Binding="{Binding ComplaintCount}" Value="0">

<Setter Property="IsVisible" Value="False" />

</DataTrigger>

</Label.Triggers>

</Label>

</Grid>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

<Button Text="Загрузить ещё" IsVisible="{Binding CanLoadMore}" Command="{Binding LoadDataCommand}" />

</VerticalStackLayout>

</ScrollView>

<!-- Полноэкранный режим -->

<Grid IsVisible="{Binding IsFullScreenVisible}" BackgroundColor="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Black}}">

<Grid ZIndex="2" IsVisible="{Binding AreControlsVisible}">

<!-- Определение строк и столбцов -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<!-- Верхняя панель -->

<StackLayout ZIndex="1" Orientation="Horizontal" Grid.Row="0" Grid.ColumnSpan="3" Spacing="10" BackgroundColor="{AppThemeBinding Light='#96FFFFFF', Dark='#96000000'}">

<ImageButton Source="arrow\_left.png"

Command="{Binding CloseFullScreenCommand}"

HorizontalOptions="Start"

Padding="5"

BackgroundColor="Transparent"

Margin="10"/>

<Label Text="{Binding CurrentPicture.Name}"

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center"

FontSize="Medium"

LineBreakMode="TailTruncation"/>

<ImageButton Source="trash.png"

Command="{Binding DeleteImageCommand}"

HorizontalOptions="End"

Padding="5"

BackgroundColor="Transparent"

Margin="10"/>

</StackLayout>

<!-- Кнопка "Влево" -->

<ImageButton Source="chevron\_left.png"

Command="{Binding PreviousImageCommand}"

Grid.Row="1"

Grid.Column="0"

VerticalOptions="Center"

Padding="5"

BackgroundColor="{AppThemeBinding Light='#96FFFFFF', Dark='#96000000'}"

CornerRadius="50"/>

<!-- Кнопка "Вправо" -->

<ImageButton Source="chevron\_right.png"

Command="{Binding NextImageCommand}"

Grid.Row="1"

Grid.Column="2"

VerticalOptions="Center"

Padding="5"

BackgroundColor="{AppThemeBinding Light='#96FFFFFF', Dark='#96000000'}"

CornerRadius="50"/>

</Grid>

<!-- Отображение изображения -->

<Grid HorizontalOptions="Center" VerticalOptions="Center">

<pinch:PinchZoom>

<Image Source="{Binding CurrentPicture.OriginalPath}" Aspect = "AspectFit"/>

</pinch:PinchZoom>

</Grid>

</Grid>

</Grid>

</ContentPage>

Итоговый интерфейс включает список изображений, который можно прокручивать, кнопки управления (заблокировать, удалить, выдать предупреждение) и полноэкранный просмотр с возможностью навигации между изображениями. Если изображение имеет жалобы, их количество отображается поверх изображения (см. Рисунок 47). В полноэкранном режиме есть кнопки навигации, скрытия элементов управления и удаления изображения (см. Рисунок 48).

|  |  |
| --- | --- |
| Рисунок 47 – Список изображений альбома пользователя | Рисунок 48 – Полноэкранный просмотр изображений |

Класс UserContentViewModel управляет загрузкой данных альбома, обработкой команд (удаление, блокировка, отклонение жалобы и пр.), а также реализацией полноэкранного режима просмотра изображений. Используются команды RelayCommand для обработки действий, а ObservableProperty обеспечивает обновление данных в реальном времени (см. Листинг 61).

Листинг 61 – Код ViewModels/UserContentViewModel.cs

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using Newtonsoft.Json;

using System.Collections.ObjectModel;

using PicsyncAdmin.Helpers;

using PicsyncAdmin.Models;

using PicsyncAdmin.Models.Response;

using System.Diagnostics;

namespace PicsyncAdmin.ViewModels

{

public partial class UserContentViewModel : ObservableObject

{

public static UserContentViewModel? Instance { get; private set; }

private readonly string? \_token = AuthSession.Token;

public Album Album { get; set; }

public UserContentViewModel(Album album)

{

Instance = this;

Album = album;

\_ = LoadData();

}

[ObservableProperty] private ObservableCollection<Picture> albumPictures = new();

[ObservableProperty] private bool isFullScreenVisible;

[ObservableProperty] private bool areControlsVisible = true;

[ObservableProperty] private Picture? currentPicture;

[ObservableProperty]

[NotifyCanExecuteChangedFor(nameof(NextImageCommand))]

[NotifyCanExecuteChangedFor(nameof(PreviousImageCommand))]

private int currentIndex;

private bool CanLoadData() => !IsFetch;

public bool CanGoToPrevious() => CurrentIndex > 0;

public bool CanGoToNext() => CurrentIndex < PicturesCount - 1;

[ObservableProperty]

[NotifyCanExecuteChangedFor(nameof(LoadDataCommand))]

private bool isFetch = false;

[ObservableProperty] private string statusMessage;

[ObservableProperty] private bool canLoadMore = false;

[ObservableProperty] private int currentPage = 1;

[ObservableProperty] private int complaintCount;

[ObservableProperty] private string? mainImagePath;

[ObservableProperty] private bool isMainImageVisible;

[ObservableProperty]

[NotifyCanExecuteChangedFor(nameof(NextImageCommand))]

private int picturesCount;

[ObservableProperty] private int complaintsCount;

[ObservableProperty] private string albumName;

[ObservableProperty] private string userName;

[RelayCommand]

public async Task IssueWarning()

{

string comment = await Shell.Current.DisplayPromptAsync("Создание предупреждения", "Комментарий");

var warningResponse = await Fetch.DoAsync(

HttpMethod.Post,

$"/users/{Album.User.Id}/warnings",

setError: msg => Debug.WriteLine(msg),

body: new { comment = comment },

serialize: true);

var complaintResponse = await Fetch.DoAsync(HttpMethod.Post,

$"/complaints/{Album.User.Id}",

setError: msg => Debug.WriteLine(msg),

body: new { status = 1 },

serialize: true);

if (warningResponse.IsSuccessStatusCode && complaintResponse.IsSuccessStatusCode)

{

await Shell.Current.DisplayAlert("Успех", "Предупреждение выдано", "OK");

await HomeViewModel.Instance.ResetComplaints();

await Shell.Current.Navigation.PopModalAsync();

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Не удалось выдать предупреждение", "OK");

}

}

[RelayCommand]

public async Task DeleteAlbum()

{

var result = await Shell.Current.DisplayAlert(

"Подтверждение удаления",

"Вы уверены, что хотите удалить этот альбом?",

"Да", "Нет");

if (!result) return;

var response = await Fetch.DoAsync(

HttpMethod.Delete,

$"/albums/{Album.Id}",

setError: msg => Debug.WriteLine(msg));

if (response.IsSuccessStatusCode)

{

await LoadData();

await Shell.Current.GoToAsync("//HomePage");

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Не удалось удалить альбом", "OK");

}

}

[RelayCommand]

public async Task BlockUser()

{

var result = await Shell.Current.DisplayAlert(

"Подтверждение блокировки",

"Вы уверены, что хотите заблокировать этого пользователя?",

"Да", "Нет");

if (!result) return;

var response = await Fetch.DoAsync(

HttpMethod.Post,

$"/users/{Album.User.Id}",

setError: msg => Debug.WriteLine(msg),

body: new { is\_banned = true },

serialize: true);

if (response.IsSuccessStatusCode)

{

await Shell.Current.DisplayAlert("Успех", "Пользователь заблокирован", "OK");

await HomeViewModel.Instance.ResetComplaints();

await Shell.Current.Navigation.PopModalAsync();

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Не удалось заблокировать пользователя", "OK");

}

}

[RelayCommand]

public async Task RejectComplaint()

{

var response = await Fetch.DoAsync(

HttpMethod.Delete,

$"/complaints/{Album.User.Id}",

setError: msg => Debug.WriteLine(msg));

if (response.IsSuccessStatusCode)

{

await Shell.Current.GoToAsync("//HomePage");

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Не удалось отклонить жалобу", "OK");

}

}

[RelayCommand]

private void ViewImage(Picture picture)

{

CurrentPicture = picture;

CurrentIndex = AlbumPictures.IndexOf(picture);

IsFullScreenVisible = true;

Debug.WriteLine(picture.ComplaintCount);

Shell.SetNavBarIsVisible(Shell.Current, false);

Shell.SetTabBarIsVisible(Shell.Current.CurrentPage, false);

}

[RelayCommand]

private void CloseFullScreen()

{

IsFullScreenVisible = false;

CurrentPicture = null;

Shell.SetNavBarIsVisible(Shell.Current, true);

Shell.SetTabBarIsVisible(Shell.Current, true);

}

[RelayCommand]

public void ToggleControlsVisibility()

{

AreControlsVisible = !AreControlsVisible;

// Платформо-специфичное управление статус-баром и навигационной панелью

#if ANDROID

var activity = Platform.CurrentActivity;

if (activity != null)

{

if (AreControlsVisible)

{

// Показать статус-бар и навигационную панель

activity.Window.DecorView.SystemUiVisibility = Android.Views.StatusBarVisibility.Visible;

}

else

{

// Скрыть статус-бар и навигационную панель

activity.Window.DecorView.SystemUiVisibility =

(Android.Views.StatusBarVisibility)(

Android.Views.SystemUiFlags.HideNavigation |

Android.Views.SystemUiFlags.Fullscreen |

Android.Views.SystemUiFlags.ImmersiveSticky

);

}

}

#elif IOS

UIKit.UIApplication.SharedApplication.SetStatusBarHidden(!AreControlsVisible, UIKit.UIStatusBarAnimation.Fade);

#endif

}

[RelayCommand]

public async Task DeleteImage()

{

var confirmDelete = await Shell.Current.DisplayAlert("Подтверждение", "Вы уверены, что хотите удалить это изображение?", "Да", "Нет");

if (!confirmDelete) return;

var response = await Fetch.DoAsync(

HttpMethod.Delete,

$"/albums/{Album.Id}/pictures/{CurrentPicture.Id}",

setError: msg => Debug.WriteLine(msg)

);

if (response.IsSuccessStatusCode)

{

await Shell.Current.DisplayAlert("Удаление", "Изображение удалено.", "Ок");

AlbumPictures.Remove(CurrentPicture);

// Обновляем текущую картинку

if (CanGoToNext()) // Если можно перейти к следующему изображению

{

\_ = NextImage();

}

else if (CanGoToPrevious()) // Если нет следующего, переходим к предыдущему

{

PreviousImage();

}

else

{

// Если не осталось изображений, возвращаем на главную страницу или выполняем другие действия

await Shell.Current.GoToAsync("//HomePage");

}

}

else

{

await Shell.Current.DisplayAlert("Ошибка", "Не удалось удалить изображение.", "Ок");

}

}

[RelayCommand(CanExecute = nameof(CanGoToPrevious))]

private void PreviousImage()

{

CurrentIndex--;

CurrentPicture = AlbumPictures[CurrentIndex];

}

[RelayCommand(CanExecute = nameof(CanGoToNext))]

private async Task NextImage()

{

try

{

// Проверяем, можно ли загрузить больше картинок

if (CanGoToNext() && CanLoadMore && CurrentIndex == AlbumPictures.Count-1)

{

await LoadData(); // Загружаем больше данных, если это возможно

}

// После загрузки новых картинок (или если их не было), переходим к следующей картинке

if (CanGoToNext())

{

CurrentIndex++;

CurrentPicture = AlbumPictures[CurrentIndex];

}

}

catch (Exception ex)

{

await Shell.Current.DisplayAlert("Ошибка", "Произошла ошибка при загрузке следующего изображения.", "OK");

}

}

[RelayCommand(CanExecute = nameof(CanLoadData))]

public async Task LoadData()

{

if (Album.Id == 0) return;

try

{

StatusMessage = "Загрузка данных...";

IsFetch = true;

// Получение списка картинок

var response = await Fetch.DoAsync(

HttpMethod.Get,

$"/albums/{Album.Id}/pictures?reverse&sort=complaints&page={CurrentPage}",

setError: msg => Debug.WriteLine(msg));

// Получение информации об альбоме

var responseAlbum = await Fetch.DoAsync(

HttpMethod.Get,

$"/albums/{Album.Id}",

setError: msg => Debug.WriteLine(msg));

IsFetch = false;

if (!response.IsSuccessStatusCode || !responseAlbum.IsSuccessStatusCode)

{

StatusMessage = "Ошибка загрузки данных. Попробуйте снова.";

return;

}

// Разбор ответа с картинками

var responseObject = JsonConvert.DeserializeObject<PicturesResponse>(await response.Content.ReadAsStringAsync());

if (responseObject?.Pictures == null || !responseObject.Pictures.Any())

{

StatusMessage = "Картинки отсутствуют.";

return;

}

// Разбор ответа с данными альбома

var albumData = JsonConvert.DeserializeObject<AlbumResponse>(await responseAlbum.Content.ReadAsStringAsync());

if (albumData?.Album == null)

{

StatusMessage = "Не удалось загрузить данные альбома.";

return;

}

// Запись данных в свойства

PicturesCount = albumData.Album.PicturesCount;

ComplaintsCount = albumData.Album.ComplaintsCount;

AlbumName = albumData.Album.Name;

UserName = albumData.Album.User?.Name ?? "";

StatusMessage = ""; // Скрываем статус, если данные загружены

// Логика работы с картинками

CanLoadMore = responseObject.Total > responseObject.Page \* responseObject.Limit;

CurrentPage++;

foreach (var picture in responseObject.Pictures)

{

picture.OriginalPath ??= new API\_URL($"/albums/{Album.Id}/pictures/{picture.Id}/original?sign={responseObject.Sign}");

picture.Path ??= new API\_URL($"/albums/{Album.Id}/pictures/{picture.Id}/thumb/q480?sign={responseObject.Sign}");

}

foreach (var picture in responseObject.Pictures)

{

AlbumPictures.Add(picture);

}

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка: {ex.Message}");

StatusMessage = "Произошла ошибка при загрузке данных.";

}

}

}

}

### Страница профиля

Данная страница позволяет пользователю изменить своё имя, логин и пароль, а также выйти из системы. При сохранении данных выполняется проверка на совпадение паролей и отправляется запрос на обновление профиля. В случае ошибки или успешного изменения отображается соответствующее сообщение. Код разметки представлен в Листинг 62.

Листинг 62 – Разметка страницы Views/Profile.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="PicsyncAdmin.Views.Profile"

Title="Профиль">

<VerticalStackLayout Padding="20" Spacing="15">

<Entry Placeholder="Имя" Text="{Binding Name}" />

<Entry Placeholder="Логин" Text="{Binding Login}" />

<Entry Placeholder="Новый пароль" Text="{Binding Password}" IsPassword="True" />

<Entry Placeholder="Повторите пароль" Text="{Binding ConfirmPassword}" IsPassword="True" />

<Label Text="{Binding ValidationMessage}"

TextColor="{Binding ValidationMessageColor}"

HorizontalOptions="Center" />

<HorizontalStackLayout Spacing="10">

<Button Text="Сохранить" Command="{Binding SaveChangesCommand}" />

<Button Text="Выход" Command="{Binding LogoutCommand}" />

</HorizontalStackLayout>

</VerticalStackLayout>

</ContentPage>

Итоговый интерфейс включает в себя поля ввода для имени, логина и пароля, а также кнопку сохранения изменений и кнопку выхода. При вводе некорректных данных под полями отображается сообщение с указанием ошибки. Кнопки управления размещены в горизонтальном ряду для удобства (см. Рисунок 49).

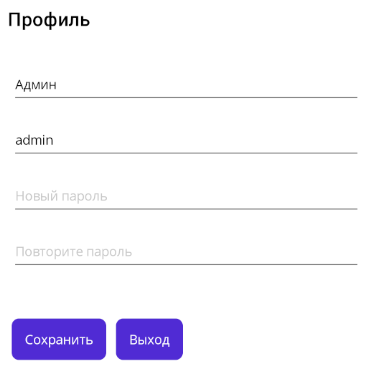


Рисунок 49 – Страница профиля

Класс ProfileViewModel управляет данными профиля, загружая их из AuthSession.User. Команда SaveChangesCommand отправляет обновлённые данные на сервер и при успехе обновляет локальные данные пользователя. Команда LogoutCommand выполняет выход из системы, очищая сессию и перенаправляя пользователя на страницу выбора API (см. Листинг 63).

Листинг 63 – Код ViewModels/ProfileViewModel.cs

using System.Diagnostics;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using PicsyncAdmin.Helpers;

namespace PicsyncAdmin.ViewModels

{

public partial class ProfileViewModel : ObservableObject

{

[ObservableProperty] private string? name;

[ObservableProperty] private string? login;

[ObservableProperty] private string? password;

[ObservableProperty] private string? confirmPassword;

[ObservableProperty] private string? validationMessage;

[ObservableProperty] private Color? validationMessageColor;

public ProfileViewModel()

{

// Инициализация свойств данными из текущей сессии

Name = AuthSession.User?.Name;

Login = AuthSession.User?.Login;

}

[RelayCommand]

private async Task SaveChangesAsync()

{

ValidationMessage = null;

ValidationMessageColor = Colors.Red; // По умолчанию ошибки будут красными

try

{

if (Password != ConfirmPassword)

{

ValidationMessage = "Пароли не совпадают";

return;

}

var payload = new { Name, Login, Password };

var response = await Fetch.DoAsync(HttpMethod.Post, "/users/me", setError: msg => ValidationMessage = msg, body: payload, serialize: true);

if (response.IsSuccessStatusCode)

{

// Обновляем данные пользователя в сессии

AuthSession.User!.Name = Name;

AuthSession.User!.Login = Login;

ValidationMessage = "Изменения сохранены успешно.";

ValidationMessageColor = Colors.Green; // Успешное сообщение зелёным

// Очищаем сообщение через 3 секунды

await Task.Delay(3000);

ValidationMessage = null;

}

else

{

ValidationMessage ??= "Не удалось сохранить изменения.";

}

}

catch (Exception ex)

{

ValidationMessage = $"Ошибка: {ex.Message}";

}

}

[RelayCommand]

private async Task LogoutAsync()

{

bool isExit = false;

try

{

var response = await Fetch.DoAsync(HttpMethod.Post, "/logout", setError: msg => Debug.WriteLine(msg));

if (!response.IsSuccessStatusCode)

{

throw new Exception($"Ошибка {response.StatusCode}");

}

isExit = true;

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка при выходе: {ex.Message}");

isExit = await Shell.Current.DisplayAlert(

"Не получилось выйти",

$"Вы желаете насильно выйти? \n{ex.Message}",

"Да",

"Нет"

);

}

if (!isExit) return;

AuthSession.ClearSession();

await Shell.Current.GoToAsync("//ApiUrlSelectionPage");

}

}

}

### Страница настроек

Данная страница позволяет администратору управлять параметрами хранилища и загрузки файлов. Пользователь может изменять порог блокировки загрузок, задавать форматы допустимых изображений и размеры превью, а также контролировать использование дискового пространства. Страница также предоставляет возможность просмотра доступного и использованного пространства с визуальным прогресс-баром. Код разметки представлен в Листинг 64.

Листинг 64 – Разметка страницы Views/SettingsPage.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="PicsyncAdmin.Views.SettingsPage"

xmlns:vm="clr-namespace:PicsyncAdmin.ViewModels"

Title="Настройки">

<ScrollView>

<StackLayout Padding="20" Spacing="15">

<!-- Пространство -->

<StackLayout Grid.Row="2" Spacing="10" Padding="20">

<Label Text="Использовано пространства"

HorizontalOptions="Center"

FontSize="16"

FontAttributes="Bold" />

<ProgressBar Progress="{Binding UsedPercent}"

HorizontalOptions="FillAndExpand" />

<StackLayout Orientation="Horizontal" Spacing="5" HorizontalOptions="Center">

<Label Text="{Binding FreeSpaceHumanReadable}"

VerticalOptions="Center"

FontSize="12" />

<Label Text="/"

VerticalOptions="Center"

FontSize="12" />

<Label Text="{Binding TotalSpaceHumanReadable}"

VerticalOptions="Center"

FontSize="12" />

<Label Text="{Binding UsedPercentDisplay, StringFormat='{0:F0}%'}"

HorizontalOptions="Center"

FontSize="14"

VerticalOptions="Center" />

</StackLayout>

<Label HorizontalOptions="Center"

Text="{Binding Space.GotAt, StringFormat='Последнее обновление: {0}'}" />

</StackLayout>

<!-- Индикатор загрузки -->

<ActivityIndicator IsRunning="{Binding IsLoading}"

IsVisible="{Binding IsLoading}" />

<Label Text="{Binding ValidationMessage}"

TextColor="{Binding ValidationMessageColor}"

HorizontalOptions="Center"

VerticalOptions="Center"

FontSize="Large" />

<!-- Поле для редактирования WarningLimitForBan -->

<Label Text="Количество жалоб для блокировки: " />

<Entry Text="{Binding WarningLimitForBan, Mode=TwoWay}"

Keyboard="Numeric"

TextChanged="OnEntryTextChanged" />

<!-- Поле для редактирования UploadDisablePercentage -->

<VerticalStackLayout>

<Label Text="{Binding UploadDisablePercentage, StringFormat='Блокировка загрузки при заполнении хранилища на: {0:F0}%'}"

LineBreakMode="WordWrap" />

<Slider Minimum="0"

Maximum="100"

Value="{Binding UploadDisablePercentage, Mode=TwoWay}" />

</VerticalStackLayout>

<!-- Список AllowedUploadMimes -->

<Grid ColumnDefinitions="Auto, \*" RowDefinitions="Auto, Auto" ColumnSpacing="10">

<!-- Заголовок и кнопка -->

<Label Text="Разрешенные форматы картинок:"

Grid.Column="0"

Grid.Row="0"

VerticalOptions="Center" />

<Button Text="Добавить"

Grid.Column="1"

Grid.Row="0"

HorizontalOptions="Start"

Command="{Binding AddMimeCommand}" />

<!-- CollectionView с горизонтальным расположением -->

<CollectionView ItemsSource="{Binding Settings.AllowedUploadMimes, Mode=TwoWay}"

Grid.Row="1"

Grid.ColumnSpan="2">

<CollectionView.ItemsLayout>

<LinearItemsLayout Orientation="Horizontal" />

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Label Text="{Binding .}"

Padding="10"

FontAttributes="Bold">

<Label.GestureRecognizers>

<TapGestureRecognizer

Command="{Binding Source={RelativeSource AncestorType={x:Type vm:SettingsViewModel}}, Path=TapCommand}"

CommandParameter="{Binding .}" />

</Label.GestureRecognizers>

</Label>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</Grid>

<!-- Список AllowedPreviewSizes -->

<Grid ColumnDefinitions="Auto, \*" RowDefinitions="Auto, Auto" ColumnSpacing="10">

<!-- Заголовок и кнопка -->

<Label Text="Разрешенные размеры превью:"

Grid.Column="0"

Grid.Row="0"

VerticalOptions="Center" />

<Button Text="Добавить"

Grid.Column="1"

Grid.Row="0"

HorizontalOptions="Start"

Command="{Binding AddSizeCommand}" />

<!-- CollectionView с горизонтальным расположением -->

<CollectionView ItemsSource="{Binding Settings.AllowedPreviewSizes, Mode=TwoWay}"

Grid.Row="1"

Grid.ColumnSpan="2">

<CollectionView.ItemsLayout>

<LinearItemsLayout Orientation="Horizontal" />

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Label Text="{Binding .}"

Padding="10"

FontAttributes="Bold">

<Label.GestureRecognizers>

<TapGestureRecognizer

Command="{Binding Source={RelativeSource AncestorType={x:Type vm:SettingsViewModel}}, Path=TapCommand}"

CommandParameter="{Binding .}" />

</Label.GestureRecognizers>

</Label>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</Grid>

<!-- Кнопка сохранения -->

<Button Text="Сохранить"

Command="{Binding Source={RelativeSource AncestorType={x:Type vm:SettingsViewModel}}, Path=SaveBanSpaceCommand}" />

</StackLayout>

</ScrollView>

</ContentPage>

Итоговый интерфейс (см. Рисунок 50) включает в себя отображение информации о хранилище, индикатор загрузки, интерактивные поля для изменения настроек, слайдер для регулировки предела загрузки, а также списки допустимых MIME-типов и размеров превью, которые можно редактировать и удалять. Завершающий элемент — кнопка сохранения изменений.



Рисунок 50 – Страница настроек

Класс SettingsViewModel отвечает за загрузку и сохранение настроек через API, обработку изменений параметров и валидацию данных. Он реализует команды для редактирования и удаления MIME-типов и размеров превью, а также управляет состоянием страницы, включая загрузку данных и отображение сообщений о результатах изменений (см. Листинг 65).

Листинг 65 – Код ViewModels/SettingsViewModel.cs

using System.Collections.ObjectModel;

using System.Diagnostics;

using System.Text.Json;

using System.Text.RegularExpressions;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using PicsyncAdmin.Helpers;

using PicsyncAdmin.Models.Response.PicsyncAdmin.Models.Response;

using Settings = PicsyncAdmin.Models.Settings;

using Space = PicsyncAdmin.Models.Space;

namespace PicsyncAdmin.ViewModels

{

public partial class SettingsViewModel : ObservableObject

{

[ObservableProperty] private Settings settings;

[ObservableProperty] private Space space;

[ObservableProperty] private bool isLoading;

[ObservableProperty] private int uploadDisablePercentage;

[ObservableProperty] private int warningLimitForBan;

[ObservableProperty] private double usedPercent;

[ObservableProperty] private string usedSpaceHumanReadable;

[ObservableProperty] private string totalSpaceHumanReadable;

[ObservableProperty] private string freeSpaceHumanReadable;

[ObservableProperty] private double usedPercentDisplay;

[ObservableProperty] private string? validationMessage;

[ObservableProperty] private Color? validationMessageColor;

public SettingsViewModel()

{

\_ = LoadSettingsAsync();

// Инициализация по умолчанию

}

[RelayCommand]

public async Task Tap(object item)

{

var result = await Shell.Current.DisplayActionSheet(

"Выберите действие", // Заголовок

"Отмена", // Кнопка "Отмена"

null, // Нет текста для cancel кнопки

"Редактирование", // Первый вариант

"Удаление" // Второй вариант

);

if (result == "Редактирование")

{

await Edit(item);

}

else if (result == "Удаление")

{

await Del(item);

}

}

[RelayCommand]

public async Task Del(object item)

{

string value;

string key;

if (item is string)

{

// Логика для AllowedUploadMimes

Settings.AllowedUploadMimes.Remove((string)item);

value = string.Join(",", Settings.AllowedUploadMimes);

key = "allowed\_upload\_mimes";

}

else if (item is int)

{

// Логика для AllowedPreviewSizes

Settings.AllowedPreviewSizes.Remove((int)item);

value = string.Join(",", Settings.AllowedPreviewSizes);

key = "allowed\_preview\_sizes";

}

else

{

// Обработка случая, если item не является string или int

throw new ArgumentException("Неподдерживаемый тип элемента", nameof(item));

}

await Save(key, value);

}

[RelayCommand]

public async Task Edit(object item)

{

string prompt = await Shell.Current.DisplayPromptAsync("Редактирование", "Введите новое значение", initialValue: item.ToString());

if (!string.IsNullOrEmpty(prompt))

{

if (item is string)

{

// Редактирование элемента в AllowedUploadMimes

if(Regex.IsMatch(prompt, @"^[a-z]{3,5}$"))

{

int index = Settings.AllowedUploadMimes.IndexOf((string)item);

if (index != -1)

{

Settings.AllowedUploadMimes[index] = prompt;

await Save("allowed\_upload\_mimes", prompt);

}

}

else

{

ValidationMessage = "MIME должен быть из 5 букв нижнего регистра American Language";

ValidationMessageColor = Colors.Red;

}

}

else if (item is int)

{

// Редактирование элемента в AllowedPreviewSizes

if (Regex.IsMatch(prompt, @"^(10000|[0-9]{1,4})$"))

{

int index = Settings.AllowedPreviewSizes.IndexOf((int)item);

if (index != -1)

{

if (int.TryParse(prompt, out int newIntValue))

{

Settings.AllowedPreviewSizes[index] = newIntValue;

await Save("allowed\_preview\_sizes", string.Join(",", Settings.AllowedPreviewSizes));

}

}

}

else

{

ValidationMessage = "Размер превью должен быть числом и не превышать 10000";

ValidationMessageColor = Colors.Red;

}

}

}

}

[RelayCommand]

public async Task SaveBanSpace()

{

if(WarningLimitForBan != Settings.WarningLimitForBan)

{

await Save("warning\_limit\_for\_ban", Convert.ToString(WarningLimitForBan));

}

if(UploadDisablePercentage != Settings.UploadDisablePercentage)

{

await Save("upload\_disable\_percentage", Convert.ToString(UploadDisablePercentage));

}

}

public async Task Save(string key, object value)

{

var payload = new { value, key };

try

{

var response = await Fetch.DoAsync(HttpMethod.Post, "/settings", setError: msg => ValidationMessage = msg, body: payload, serialize: true);

if (response.IsSuccessStatusCode)

{

ValidationMessage = "Изменения сохранены успешно.";

ValidationMessageColor = Colors.Green; // Успешное сообщение зелёным

// Очищаем сообщение через 3 секунды

await Task.Delay(2000);

ValidationMessage = null;

}

else

{

ValidationMessageColor = Colors.Red;

ValidationMessage ??= "Не удалось сохранить изменения.";

}

}

catch (Exception ex)

{

ValidationMessage = $"Ошибка: {ex.Message}";

}

}

[RelayCommand]

public async Task AddMime()

{

string prompt = await Shell.Current.DisplayPromptAsync("MIME", "Введите новый MIME", "OK", "Отмена");

if (!string.IsNullOrEmpty(prompt) && Regex.IsMatch(prompt, @"^[a-z]{3,5}$"))

{

Settings.AllowedUploadMimes.Add(prompt);

var value = string.Join(",", Settings.AllowedUploadMimes);

await Save("allowed\_upload\_mimes", value);

}

else

{

ValidationMessage = "MIME должен быть из 5 букв нижнего регистра American Language";

ValidationMessageColor = Colors.Red;

}

}

[RelayCommand]

public async Task AddSize()

{

string prompt = await Shell.Current.DisplayPromptAsync("MIME", "Введите новый MIME", "OK", "Отмена");

if (!string.IsNullOrEmpty(prompt) && Regex.IsMatch(prompt, @"^(10000|[0-9]{1,4})$"))

{

if (int.TryParse(prompt, out int newIntValue))

{

Settings.AllowedPreviewSizes.Add(newIntValue);

await Save("allowed\_preview\_sizes", string.Join(",", Settings.AllowedPreviewSizes));

}

}

else

{

ValidationMessage = "Размер превью должен быть числом и не превышать 10000";

ValidationMessageColor = Colors.Red;

}

}

[RelayCommand]

public async Task LoadSettingsAsync()

{

IsLoading = true;

try

{

var response = await Fetch.DoAsync(HttpMethod.Get, "/settings", setError: msg => Debug.WriteLine(msg));

var responseString = await response.Content.ReadAsStringAsync();

if (!string.IsNullOrEmpty(responseString))

{

var settingsResponse = JsonSerializer.Deserialize<SettingsResponse>(responseString);

if (settingsResponse != null)

{

Settings = new Settings

{

AllowedUploadMimes = new ObservableCollection<string>(settingsResponse.Settings.AllowedUploadMimes),

AllowedPreviewSizes = new ObservableCollection<int>(settingsResponse.Settings.AllowedPreviewSizes),

WarningLimitForBan = settingsResponse.Settings.WarningLimitForBan,

FreeStorageLimit = settingsResponse.Settings.FreeStorageLimit,

UploadDisablePercentage = settingsResponse.Settings.UploadDisablePercentage

};

UploadDisablePercentage = settingsResponse.Settings.UploadDisablePercentage;

WarningLimitForBan = settingsResponse.Settings.WarningLimitForBan;

Space = new Space

{

Total = settingsResponse.Space.Total,

Free = settingsResponse.Space.Free,

Used = settingsResponse.Space.Used,

UsedPercent = settingsResponse.Space.UsedPercent,

GotAt = settingsResponse.Space.GotAt

};

UsedPercentDisplay = AppSettings.UsedPercent;

UsedPercent = (double)AppSettings.UsedPercent / 100;

UsedSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.UsedSpace);

TotalSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.TotalSpace);

FreeSpaceHumanReadable = AppSettings.BytesToHuman(AppSettings.FreeSpace);

}

}

}

catch (Exception ex)

{

Debug.WriteLine($"Error loading settings: {ex.Message}");

}

finally

{

IsLoading = false;

}

}

}

}

# Заключение

Разработка мобильного приложения администратора PicSyncAdmin продемонстрировала эффективное использование .NET MAUI, паттерна MVVM и различных инструментов для создания удобного и функционального интерфейса. В ходе работы было реализовано взаимодействие с API, а также механизмы управления пользователями и их контентом, включая выдачу предупреждений, блокировку пользователей и работу с жалобами. Кроме того, в приложении предусмотрена удобная система просмотра изображений с возможностью перехода в полноэкранный режим, переключения между картинками и их удаления, что облегчает модерирование пользовательского контента.

Важным этапом стало внедрение архитектуры MVVM, которая позволила четко разделить логику представления и обработки данных, обеспечив модульность и расширяемость приложения. Использование CommunityToolkit.Mvvm упростило работу с командами и привязками данных.

Одним из ключевых вызовов при разработке стало обеспечение удобной навигации и оптимизация загрузки данных. Для этого была реализована поддержка постраничной подгрузки контента, что позволило избежать перегрузки интерфейса и ускорить работу с большими объемами данных. Функция полноэкранного просмотра изображений также была доработана, включая управление жестами и скрытие элементов управления, что улучшило пользовательский опыт. Кроме того, были внедрены механизмы работы с API, позволяющие отправлять запросы для модерирования контента и обработки жалоб, что повысило общую эффективность взаимодействия администратора с системой.

# Список используемых источников

* Официальная документация Laravel / Режим доступа: <https://laravel.su/docs/11.x>, открытый.
* Руководство по .NET MAUI и C# / Режим доступа: <https://metanit.com/sharp/maui/>, открытый.
* Официальная документация по .NET MAUI / Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/maui/?view=net-maui-9.0>, открытый.
* Руководство по PHP / Режим доступа: <https://metanit.com/php/tutorial/>, открытый.
* Руководство по MySql / Режим доступа: <https://metanit.com/sql/mysql/>, открытый.
* Список кодов состояний HTTP / Режим доступа: <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>, открытый.
* Гайды на YouTube по Postman / Режим доступа: <https://youtube.com/playlist?list=PLzf74dZga1fQwc2fKWemxdwrzsx3wlh-v&si=Li-SKn23ah7OY4t0>, открытый.