**ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА  
НОВОСТНОГО СЕРВИСА «MeNews»**

***КУРСОВАЯ РАБОТА***

*СПЕЦИАЛЬНОСТЬ 09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ*

Выполнил:

Студент группы 1912с

\_\_\_\_\_\_\_\_\_\_\_ / Малинин Е.М.

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

Проверил:

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_ / Летягина И.Е.

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

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Оглавление

[Оглавление 2](#_Toc190468904)

[Введение 3](#_Toc190468905)

[Информация о проекте 3](#_Toc190468906)

[Назначение документа 3](#_Toc190468907)

[Актуальность 3](#_Toc190468908)

[Глоссарий 3](#_Toc190468909)

[Техническое задание 5](#_Toc190468910)

[Общее описание 5](#_Toc190468911)

[Требования к функциональности 5](#_Toc190468912)

[Ролевая модель 6](#_Toc190468913)

[Диаграмма прецедентов 7](#_Toc190468914)

[Диаграммы последовательностей 8](#_Toc190468915)

[Технологический стек 23](#_Toc190468916)

[Хранимые данные 23](#_Toc190468917)

[Требования к REST API 23](#_Toc190468918)

[Требования к мобильному приложению клиента 48](#_Toc190468919)

[Проектирование и разработка базы данных 53](#_Toc190468920)

[Концептуальная модель данных 53](#_Toc190468921)

[Логическая модель данных 55](#_Toc190468922)

[Физическая модель данных 56](#_Toc190468923)

[Разработка серверной части 60](#_Toc190468924)

[Миграции 60](#_Toc190468925)

[Модели 62](#_Toc190468926)

[Маршрутизация 70](#_Toc190468927)

[Контроллеры 73](#_Toc190468928)

[Посредники 84](#_Toc190468929)

[Политики 85](#_Toc190468930)

[Конфигурация 88](#_Toc190468931)

[Заполнители 89](#_Toc190468932)

Введение

Информация о проекте

|  |  |
| --- | --- |
| Название проекта | Новостной сервис «MeNews» |
| Цель проекта | Спроектировать и разработать сервис новостного информирования «MeNews» |
| Задачи проекта | 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. 5. Разработать автоматизированные тесты и провести тестирование. |

Назначение документа

Настоящий документ рассчитан на использование в качестве ключевого документа, определяющего требование по реализации задачи по разработке новостного сервиса «MeNews».

Актуальность

С ростом темпа жизни и увеличения объема информации, которую люди потребляют ежедневно, пользователи сталкиваются с необходимостью быстро и удобно получать доступ к актуальным новостям из различных источников. Традиционные способы получения информации, такие как телевидение или печатные издания, уже не всегда соответствуют современным требованиям мобильности и оперативности.

Разрабатываемое новостное приложение предлагает персонализированную ленту новостей, которая агрегирует информацию из проверенных источников и предоставляет только релевантный контент. Это позволяет экономить время и быть в курсе самых важных событий в мире, стране или конкретной сфере.

Возможность добавлять статьи в избранное и делиться материалами с друзьями или коллегами делает приложение незаменимым инструментом для тех, кто ценит свое время и хочет оставаться в информационном потоке. Кроме того, оно предоставляет возможность для обсуждений через комментарии, что позволяет пользователям не только потреблять новости, но и участвовать в их обсуждении, формируя более глубокое понимание происходящего.

Глоссарий

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

Техническое задание

Общее описание

Разрабатываемый сервис предназначен для получения и написания информационных новостей. Его основная цель — информировать пользователей о самых актуальных событиях в мире и предоставлять возможность для их обсуждения с другими пользователями в рамках социального взаимодействия.

Читатели смогут просматривать самые популярные и актуальные новости, оценивать их, ставить реакции, что, в свою очередь, будет способствовать их выходу в топ самых обсуждаемых материалов.

Редакторы смогут размещать актуальные новости в системе и взаимодействовать с читателями через комментарии, что также будет способствовать их попаданию в рейтинг самых популярных.

Кроме того, сервис предоставит возможность классифицировать новости по тегам, искать их по ключевым словам и делиться ими по ссылке. Важной особенностью сервиса станет возможность сохранять избранные новости в памяти устройства, что обеспечит доступ к ним в любое время, независимо от наличия интернет-соединения.

Функционал сервиса разделён на две основные категории:

* **Функционал гостя** включает в себя просмотр каталога новостей, их сортировку, просмотр самих новостей и комментариев к ним.
* **Функционал читателя** включает в себя все возможности гостя, но добавляет возможность учёта истории просмотра, оставления комментариев и добавления и просмотра избранного.

**// Диаграммы было бы неплохо**

Требования к функциональности

**Гость**

1. Аутентификация и регистрация.
   * Авторизация производится по токену, выданному в процессе аутентификации (входе) пользователем в системе и отзывается при выходе.
   * Вход с другого устройства не должен приводить к выходу из аккаунта на всех остальных устройствах, принадлежащих одному пользователю.
2. Просмотр каталога новостей и самих новостей.
   * Пользователь может просматривать каталог новостей, выполнять их сортировку (по времени, по просмотрам или в случайном порядке) и фильтрации (по категориям и тегам).
   * Просмотра самой новости, а именно: полное текстовое содержимое, автора, изображения новости, количество просмотров и комментариев.
   * Комментарии новости, а именно: их автор, содержимое и аватарку.
3. Просмотр категорий.

**Читатель (пользователь)**

1. Умеет всё, что и гость.
2. Просмотр и хранение своей истории чтения.
3. Добавление и просмотр избранного.
4. Написание комментариев к новости.
5. Обновление своего профиля.
   * Обновление основной информации: фамилия, имя, адрес электронной почты.
   * Изменение пароля от аккаунта.
   * Загрузка и изменение аватара профиля.

**Репортёр**

1. Умеет всё, что и читатель (пользователь).
2. Написание и редактирование своих новостей.
   * Загрузка изображений.
   * Редактирование и написание текста.
   * Выбор категории и тегов.
   * Публикация в каталоге.

Ролевая модель

Гость — неавторизованный пользователь, которому доступен только вход, регистрация, просмотр новостей и публичная информация о пользователях.

Читатель (пользователь) — основной авторизированный пользователь сервиса, который просматривает новости, добавляет их в избранное и оставляет комментарии.

Репортёр — отвечает за написание новостей, их организацию и распределение по категориям. Также имеет те же права и возможности, что и читатель.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функционал** | | **Репортёр** | **Читатель** | **Гость** |
| Пользователи | CRUD | Да (себя) | Да (себя) | Нет |
| Авторизация | Нет | Нет | Да |
| Регистрация | Да |
| Выход с одного устройства | Да (себя) | Да (себя) | Нет |
| Выход со всех устройств | Да (себя) | Да (себя) | Нет |
| Загрузить новую аватарку | Да (себя) | Да (себя) | Нет |
| Теги | Создание | Да | Нет | Нет |
| Удаление | Нет | Нет |
| Просмотр | Да | Да | Да |
| Города | Создание | Нет | Нет | Нет |
| Просмотр | Да | Да | Да |
| Редактирование | Нет | Нет | Нет |
| Удаление | Нет |
| Реакции | Создание | Нет | Нет | Нет |
| Удаление |
| Просмотр | Да | Да | Да |
| Реакции к новости | Добавление | Да (только за себя) | Да (только за себя) | Нет |
| Удаление | Да (только за себя) | Да (только за себя) | Нет |
| Новости | Создание | Да | Нет | Нет |
| Редактирование | Да (только свои) |
| Удаление |
| Комментарии | Создание | Да | Да | Нет |
| Просмотр | Да |
| Редактирование | Да (только свои) | Да (только свои) | Нет |
| Удаление |
| Жалобы | Просмотр списка | Нет | Нет | Нет |
| Создание | Да | Да |
| Удаление | Нет | Нет |
| Изображение | Информация об изображении | Да | Да | Да |
| Просмотр изображения |
| Загрузка изображения | Нет |
| Удаление |
| Города | Создание | Нет | Нет | Нет |
| Просмотр | Да | Да | Да |
| Редактирование | Нет | Нет | Нет |
| Удаление |

Диаграмма прецедентов

Для проектирования сервисного функционала были проанализированы ключевые требования, выявленные в процессе анализа взаимодействия будущего приложения с пользователем. Эти требования охватывают как базовые сценарии использования, так и специфические, которые обеспечивают гибкость и удобство работы с системой.

На основе определённых данных была разработана диаграмма прецедентов, которая отражает основные взаимодействия между пользователями и системой, а также их возможные сценарии *(см. Рисунок 1).*

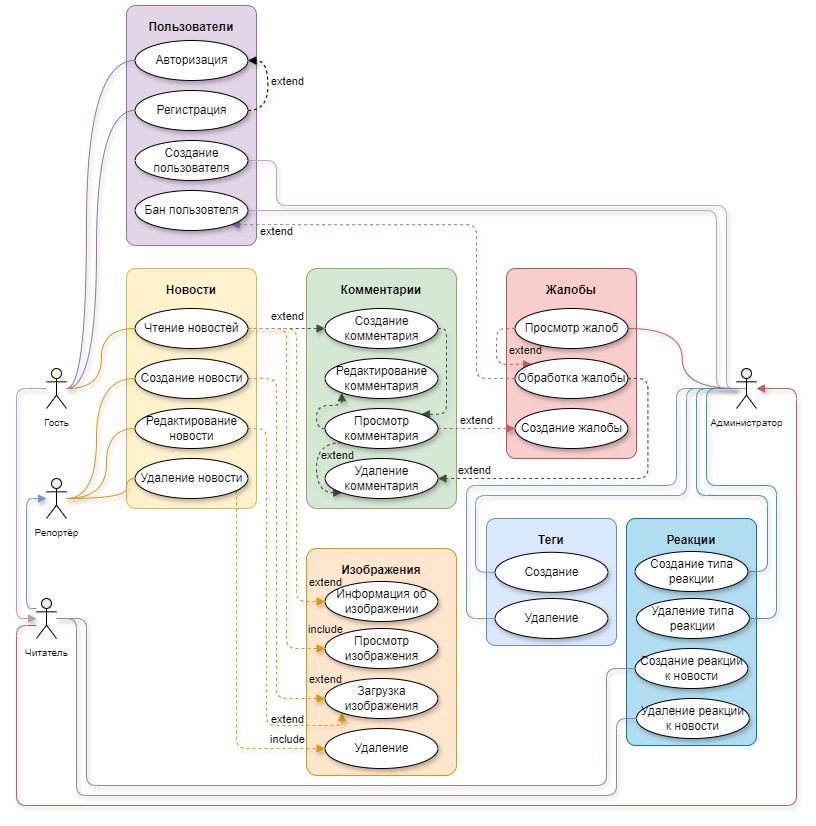


Рисунок 1 — Диаграмма прецедентов

Диаграммы последовательностей

Диаграмма последовательностей представляет собой визуальное отображение взаимодействия объектов в системе в рамках определённого сценария. Она демонстрирует последовательность сообщений, передаваемых между объектами, а также их временную упорядоченность. Диаграмма включает в себя следующие ключевые элементы:

* Объекты и акторы — участники взаимодействия
* Сообщения — вызовы методов или передача данных между объектами.
* Временная ось — вертикальная линия, на которой события расположены в хронологическом порядке.
* Активация — период, когда объект активно выполняет операцию.
* Фреймы — это своеобразные оболочки, которые окружают элементы и позволяют описать конкретный случай или цикл.

Общие диаграммы-дополнения

Диаграммы последовательности не являются независимыми и служат для уточнения других последовательностей, позволяя избежать дублирования элементов. Обычные диаграммы ссылаются на дополнительные посредством цветовой активации по цвету выхода и добавления подписи «Общее: <Название ссылки>».

Валидация тела POST-запроса выполняется посредником, который проверяет соответствие заданным правилам на основе ключей (*см.* *Рисунок 2*). Имя ссылки — «Валидация».

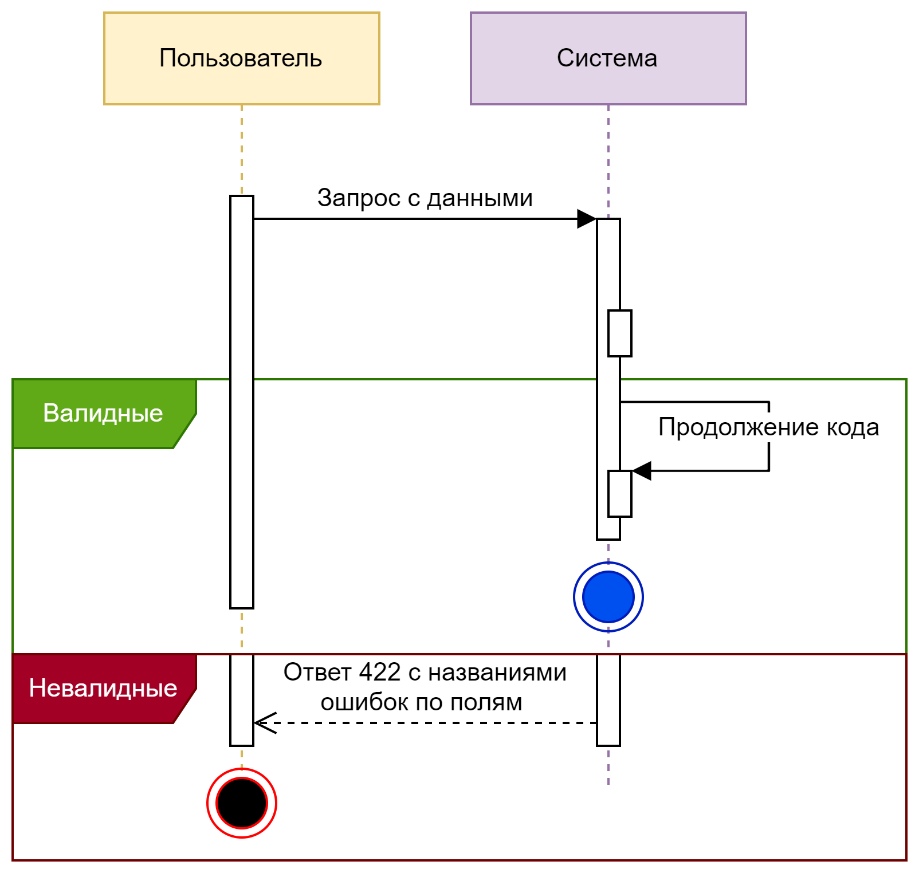


Рисунок 2 — Валидация тела запроса

В контексте кода модель пользователя и отсеивание несанкционированных запросов осуществляется посредником путем проверки токена в заголовках запроса и запроса пользователя с указанным идентификатором (*см. Рисунок 3*). Имя ссылки: «Проверка прав».

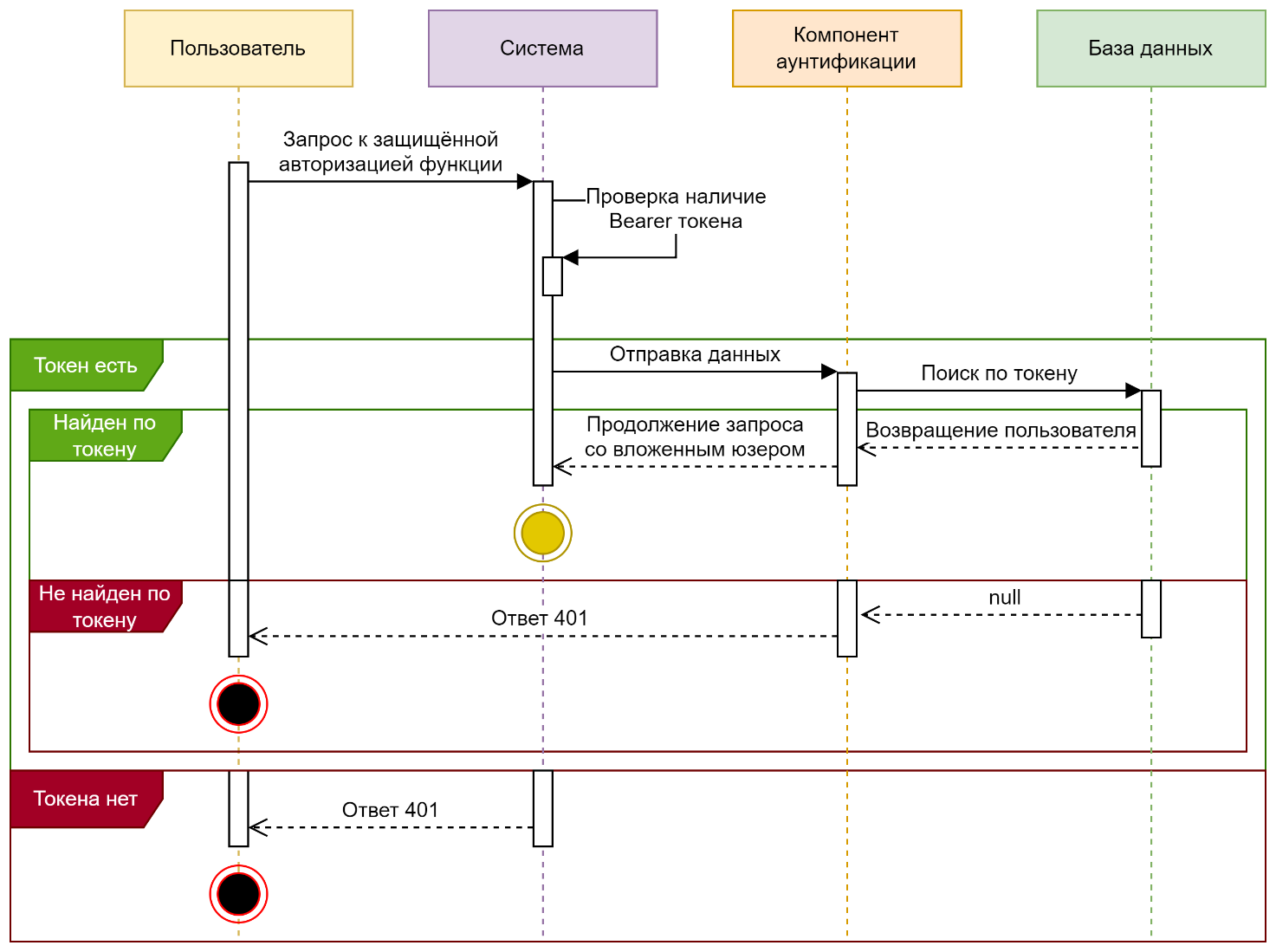


Рисунок 3 — Проверка токена и определение прав доступа

В рамках программного кода происходит поиск модели объекта по идентификатору. Если модель не найдена, то посредник получает идентификатор из маршрута и запрашивает модель из базы данных. (*см. Рисунок 4*). Имя ссылки: «Поиск по ИД».

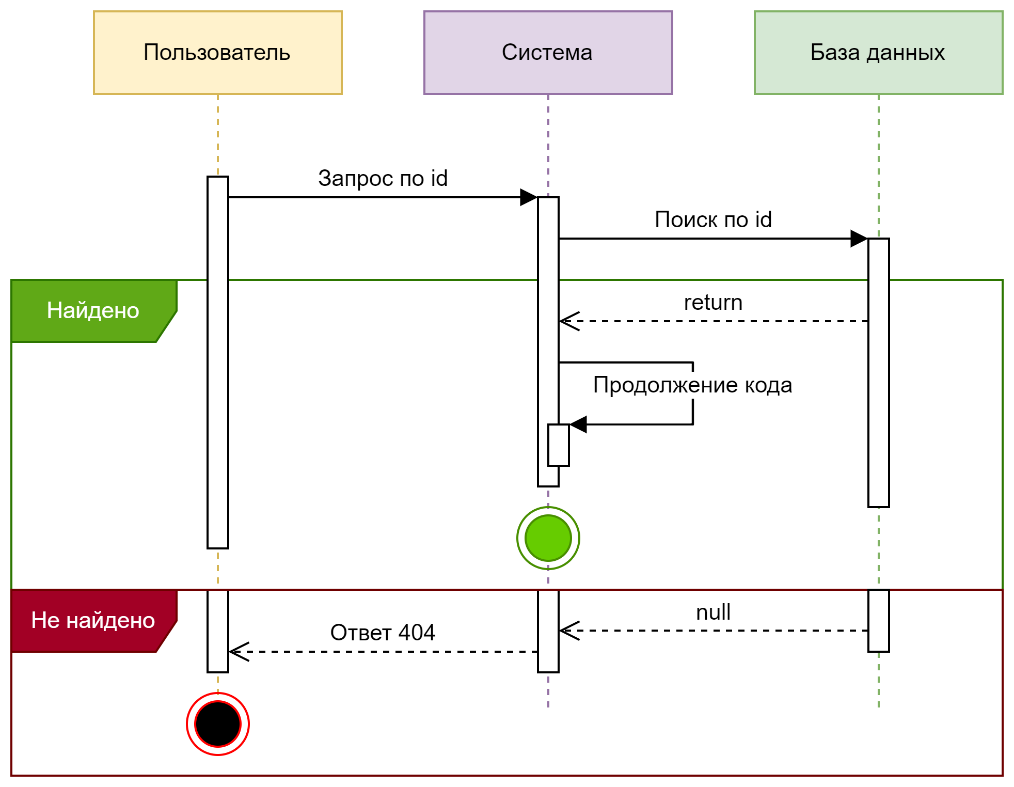


Рисунок 4 — Проверка наличия объекта в базе данных

Пользователи

Функциональные последовательности запросов для взаимодействия с пользовательскими данными:

1. Аутентификация пользователя представлена на *Рисунке 5*.
2. Регистрация пользователем в системе представлена на *Рисунке 6*.
3. Выход и удаление токена для текущей сессии пользователя представлена на *Рисунке 7*.
4. Выход посредством удаления всех токенов пользователя представлен на *Рисунке 8*.
5. Операция просмотра пользователя представлена на *Рисунке 9*.
6. Редактирование пользователя представлено на *Рисунке 10*.
7. Загрузка изображения аватарки пользователя представлена на *Рисунке 11*.

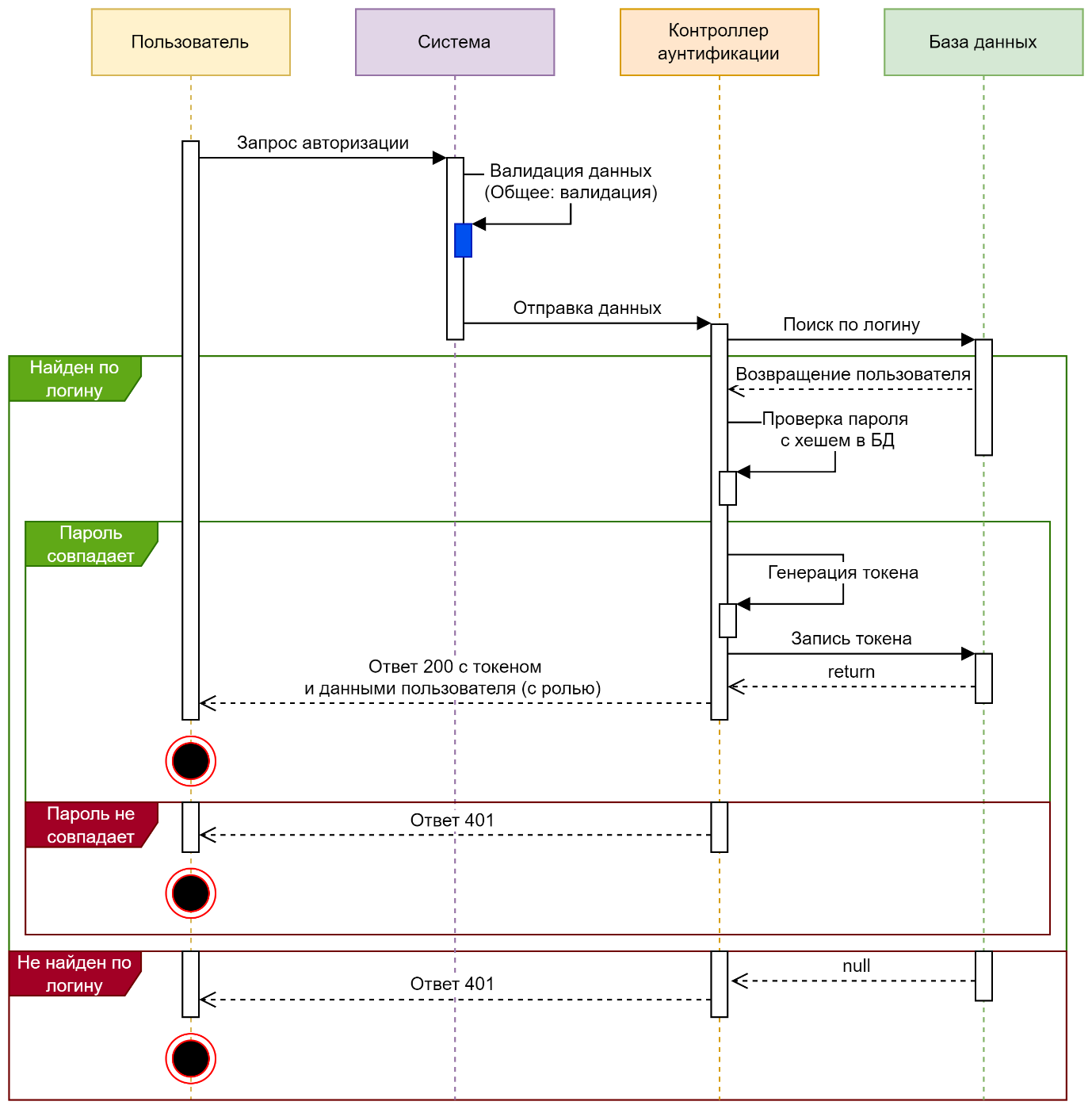


Рисунок 5 — Последовательность авторизации пользователя в системе

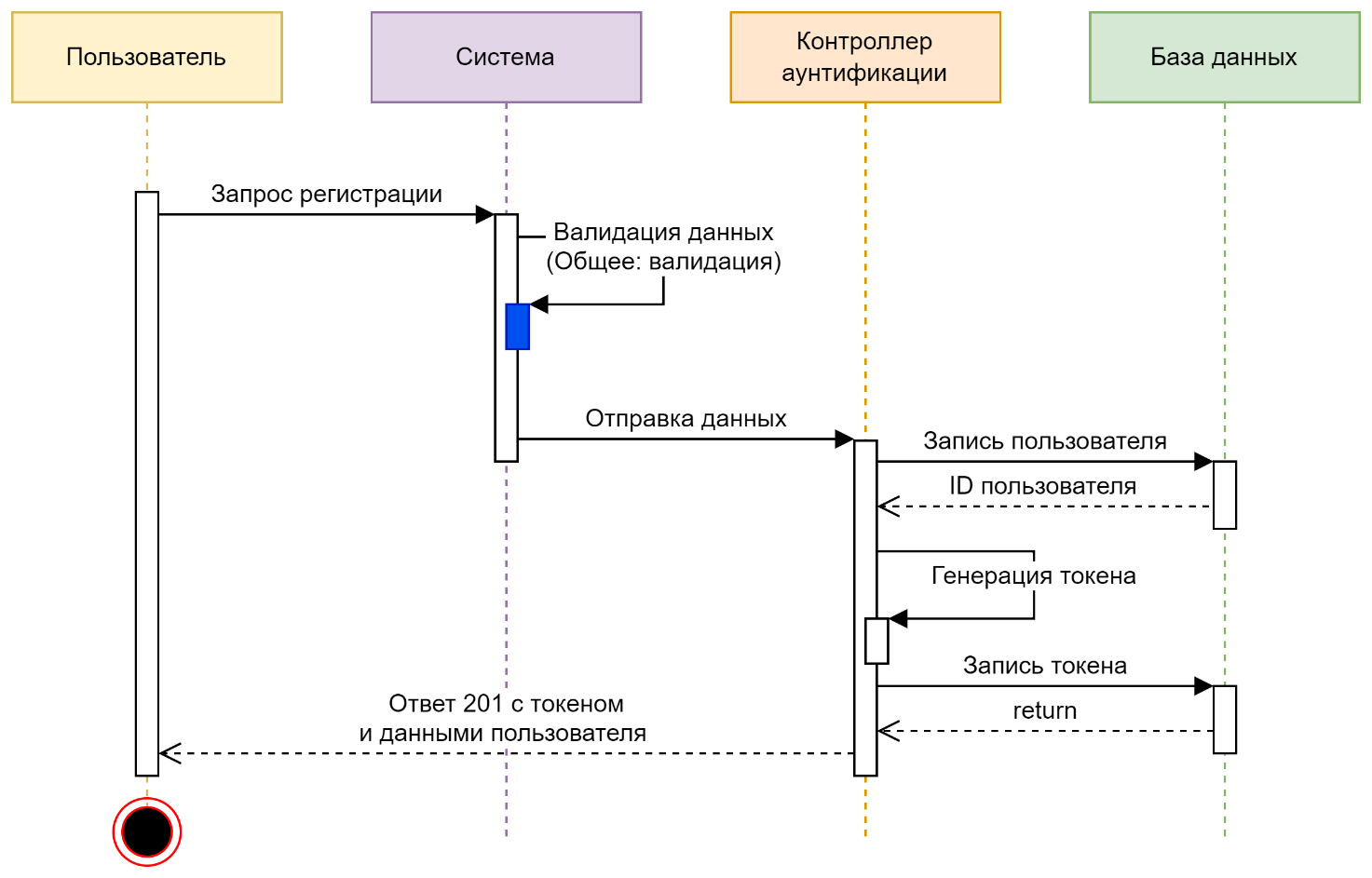


Рисунок 6 — Последовательность запроса регистрации пользователя в системе

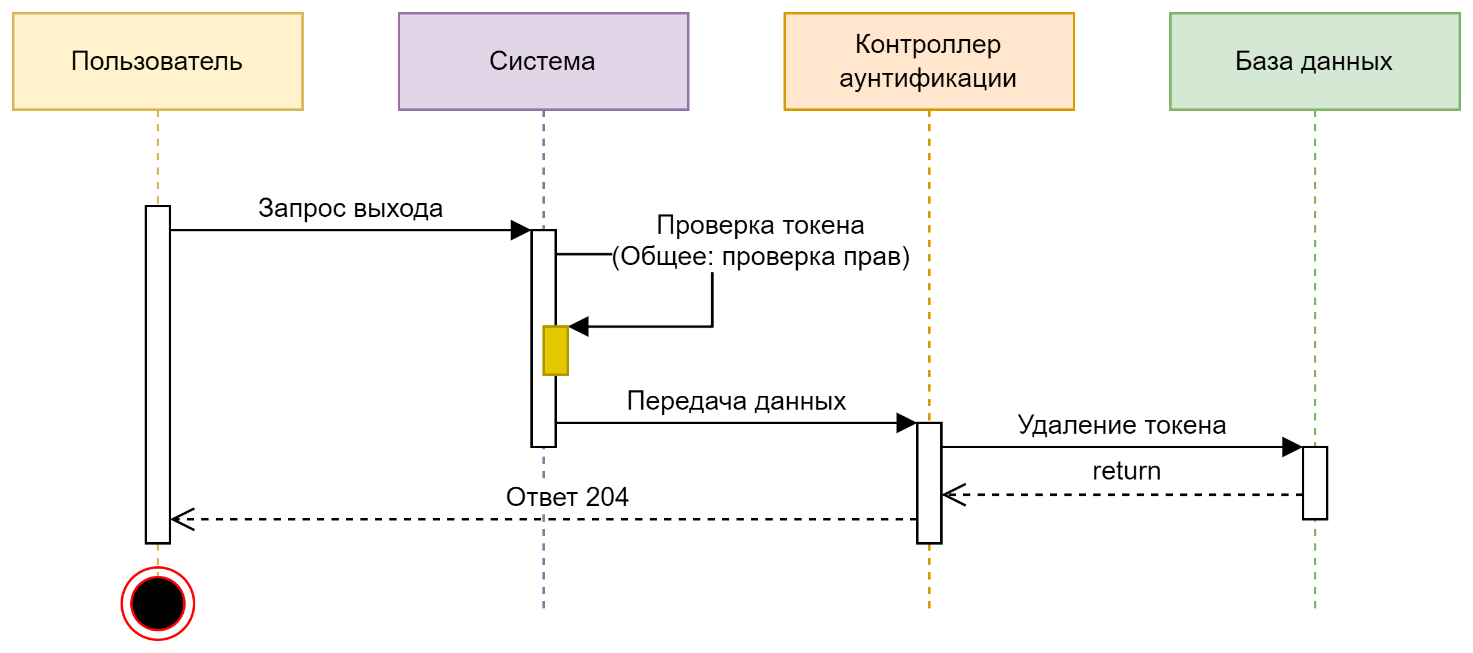


Рисунок 7 — Последовательность запроса выхода

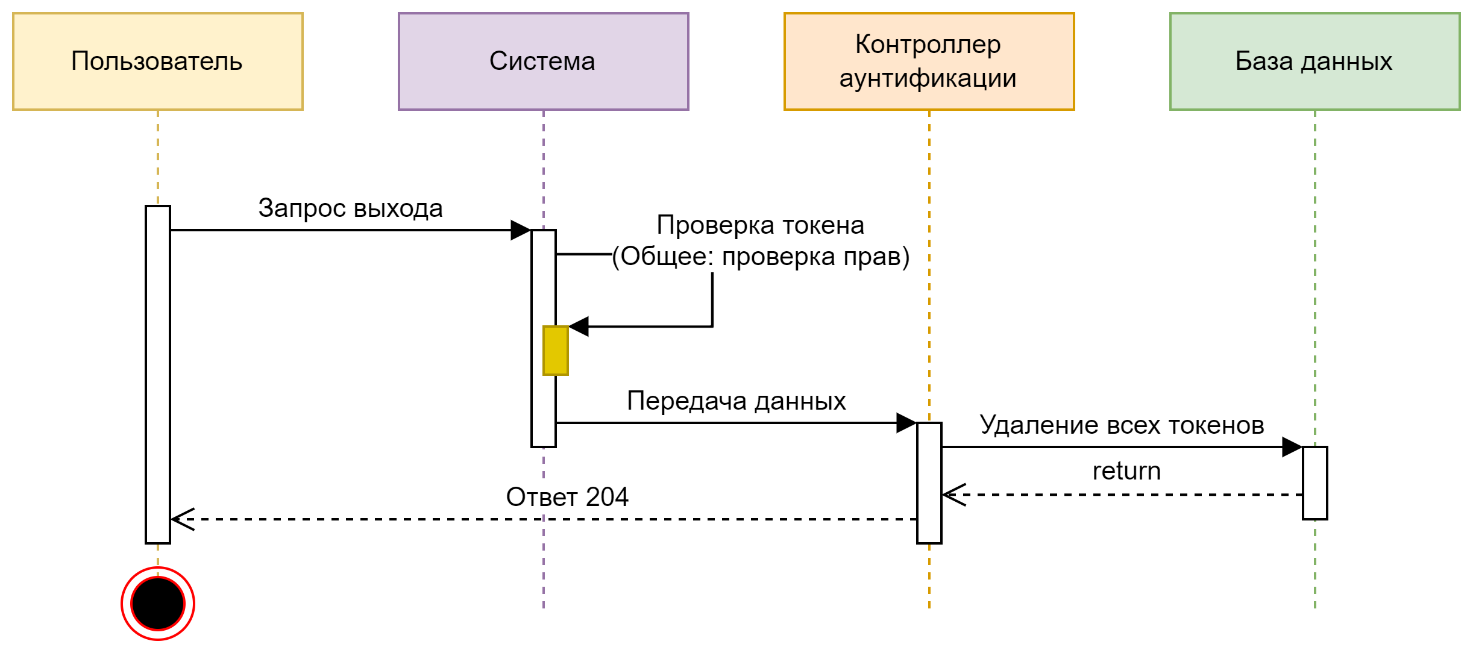


Рисунок 8 — Последовательность запроса выхода со всех устройств

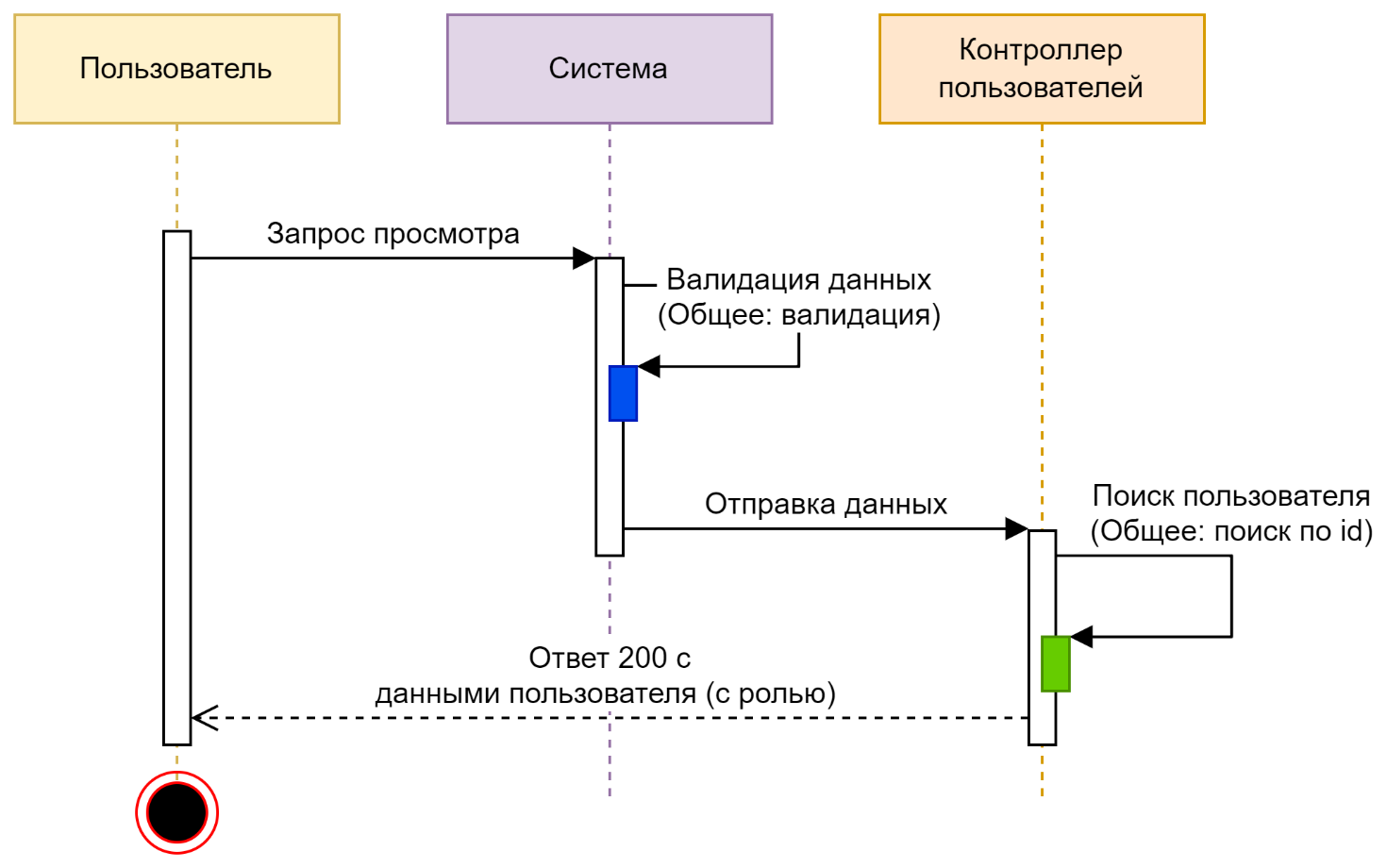


Рисунок 9 — Последовательность запроса просмотра информации о пользователе

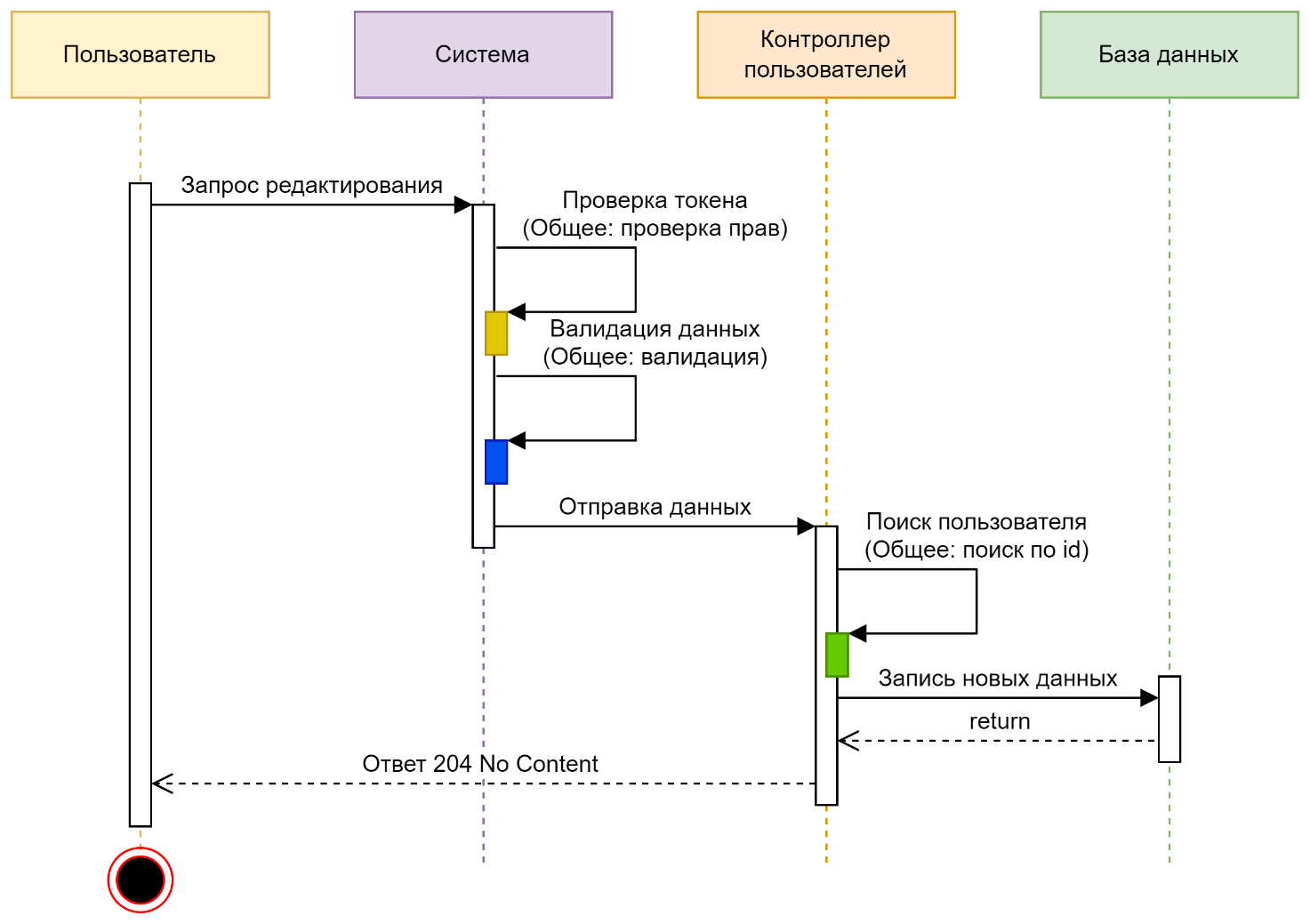


Рисунок 10 — Последовательность запроса редактирование информации о пользователе

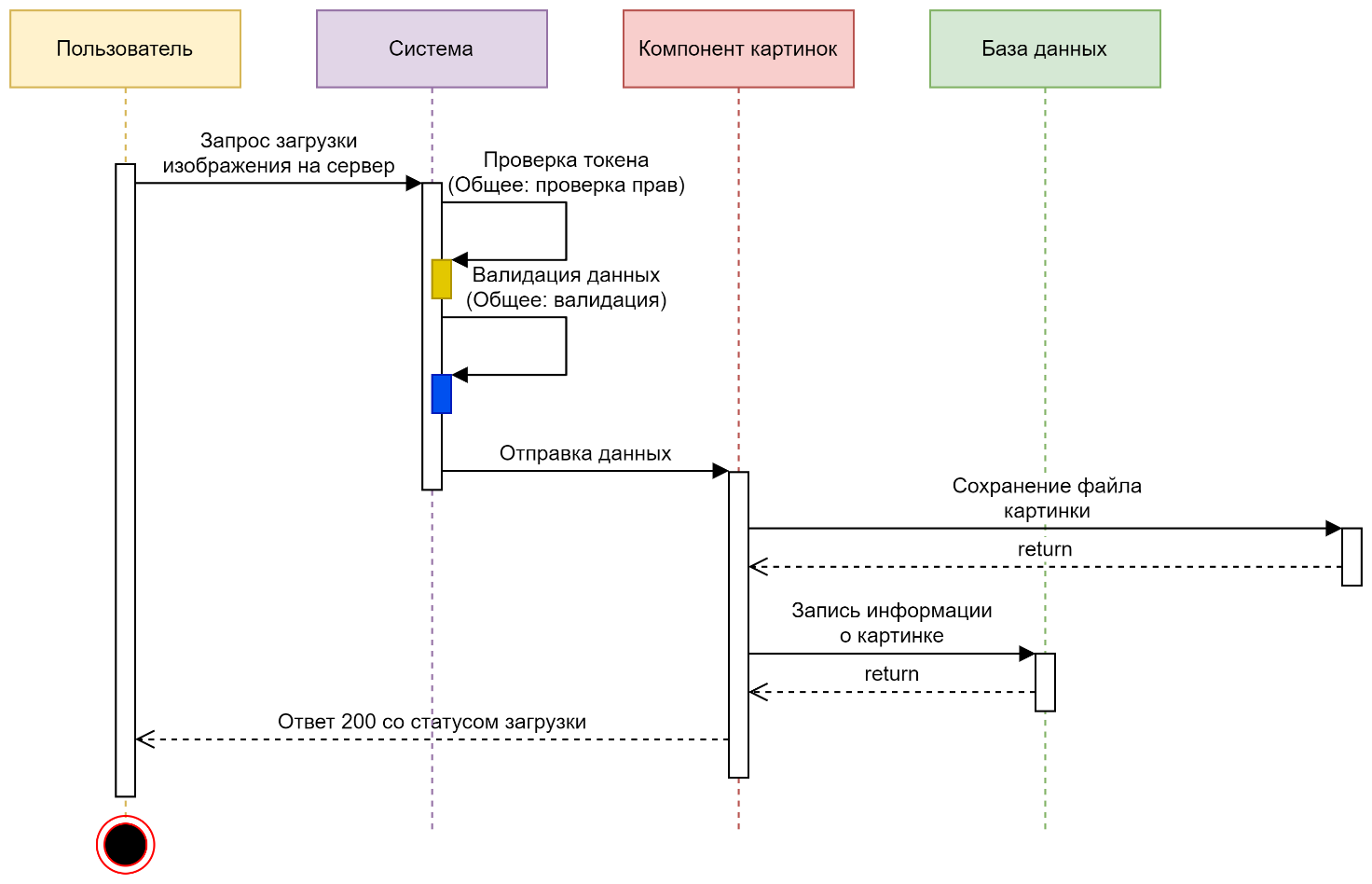


Рисунок 11 — Последовательность запроса загрузки и обновления аватарки пользователя

Теги

Функциональные последовательности запросов для взаимодействия с тегами:

1. Создание тега (*см. Рисунок 12*).
2. Просмотр тегов (*см. Рисунок 13*)
3. Удаление тега (*см. Рисунок 14*).

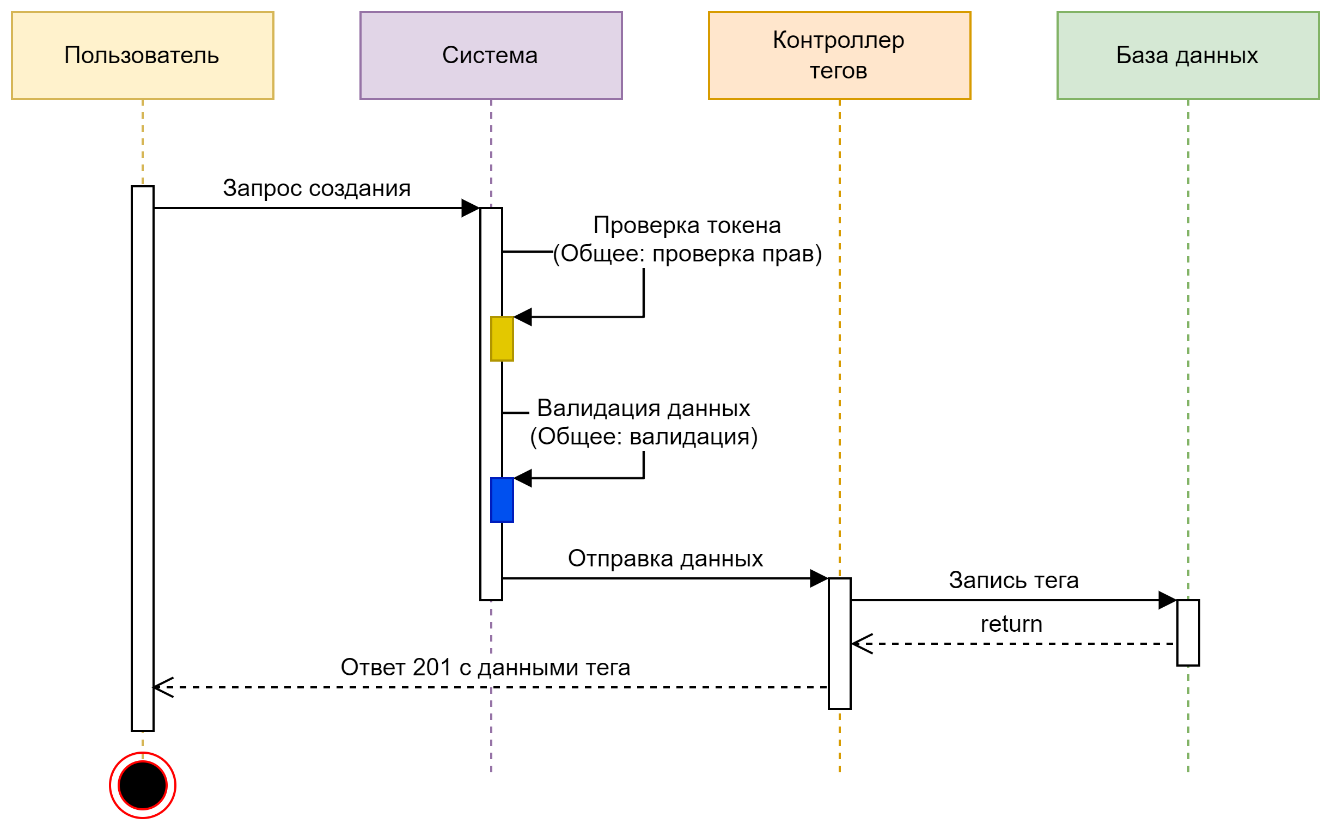


Рисунок 12 — Последовательность запрсоа создания тега

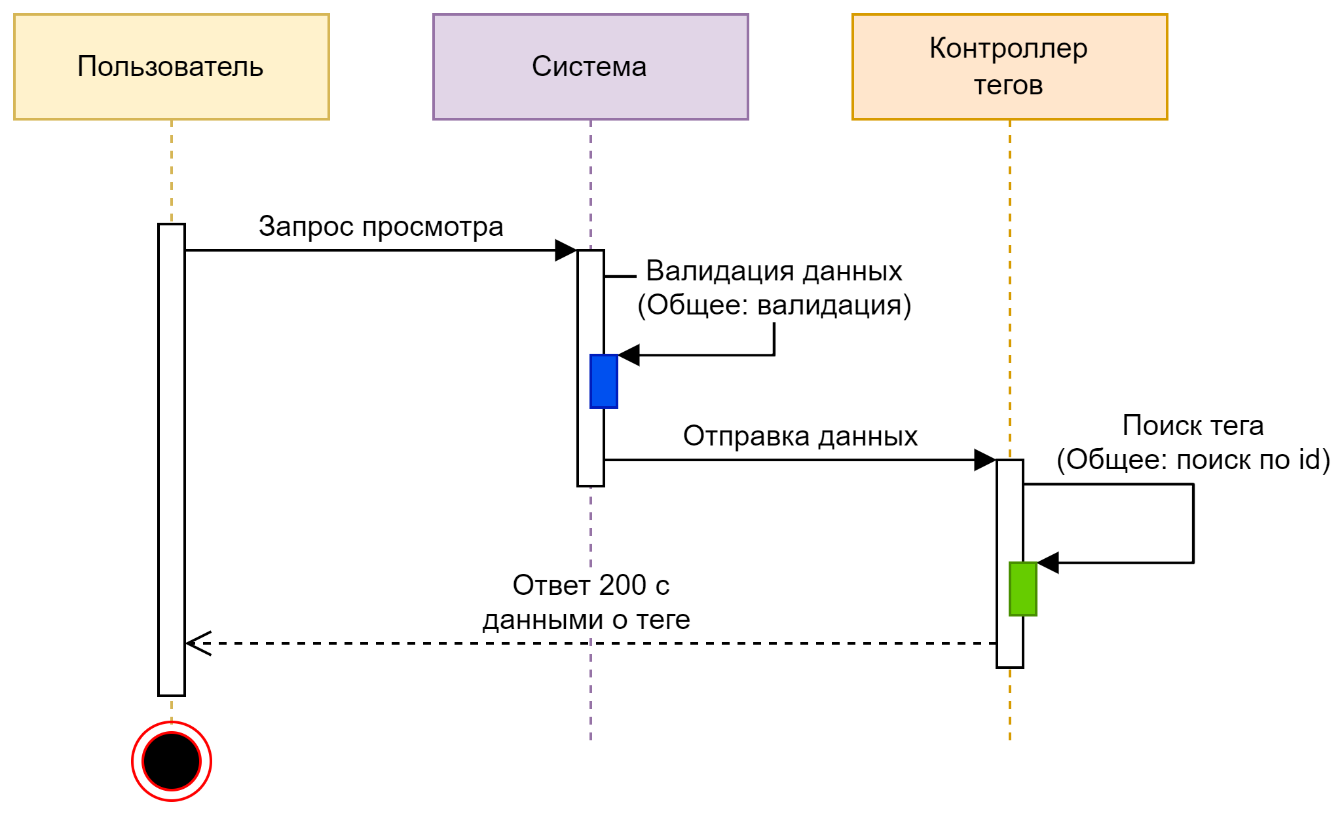


Рисунок 13 — Последовательность запроса просмотра всех тегов в системе

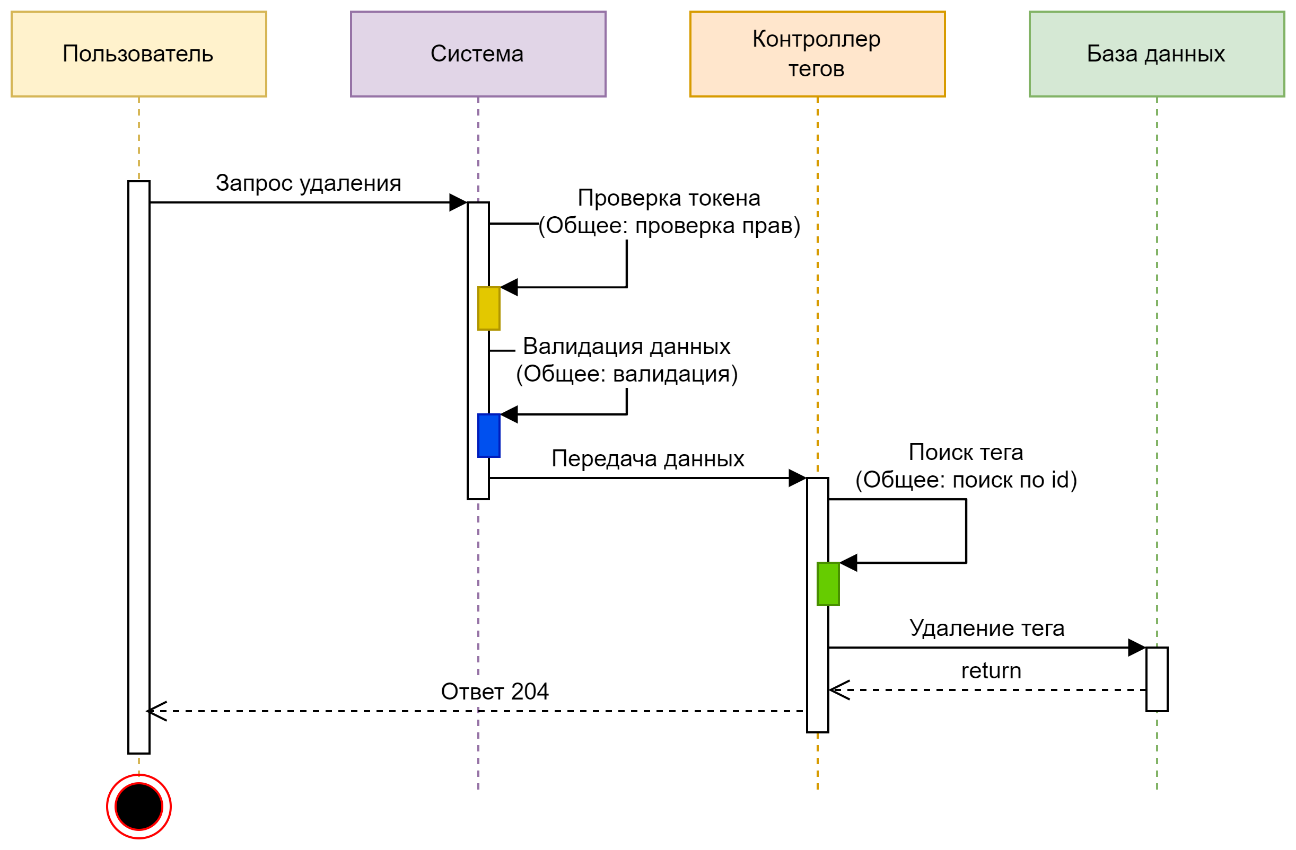


Рисунок 14 — Последовательность запроса удаления тега из системы

Комментарии

Функциональные последовательности запросов для взаимодействия с комментариями к новостям:

1. Создание нового комментария к новости (*см. Рисунок 15*).
2. Просмотр комментария (*см. Рисунок 16*).
3. Редактирование комментария (*см. Рисунок 17*).
4. Удаление комментария (*см. Рисунок 18*).

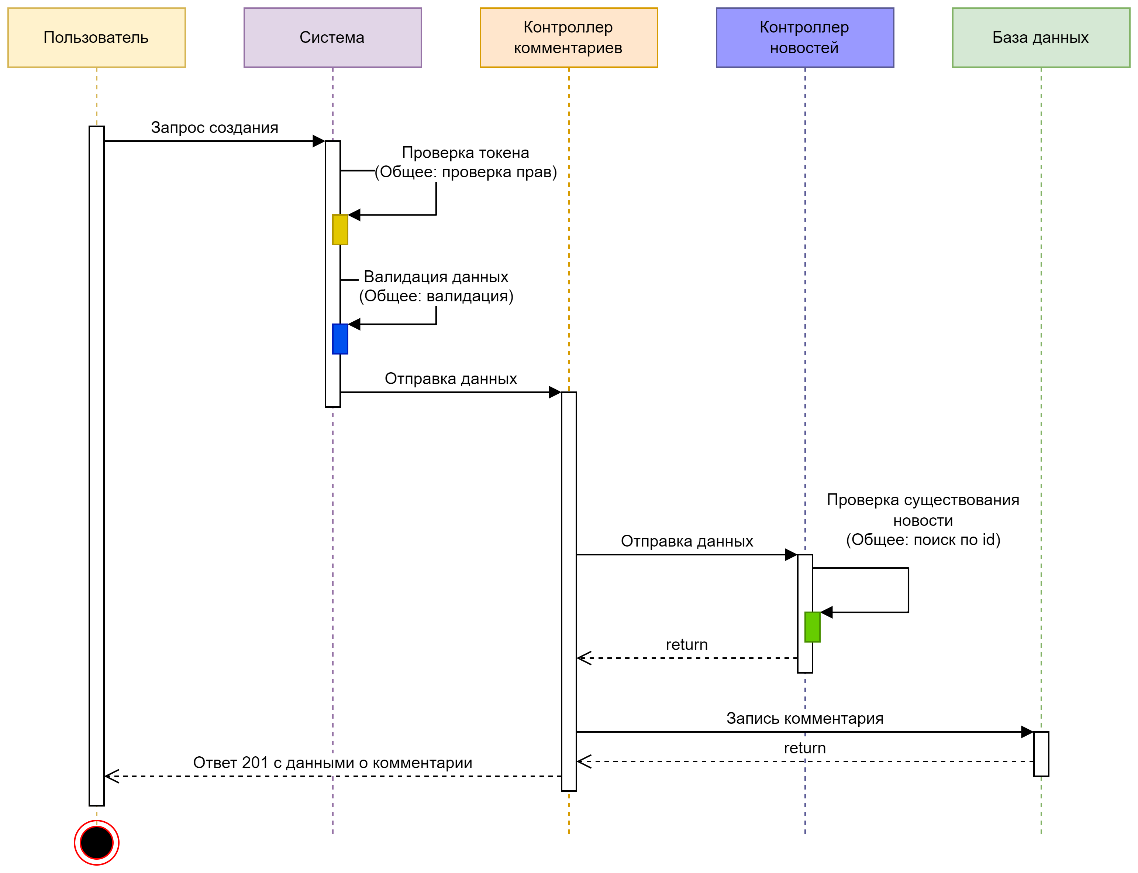


Рисунок 15 — Последовательность запроса создания комментария

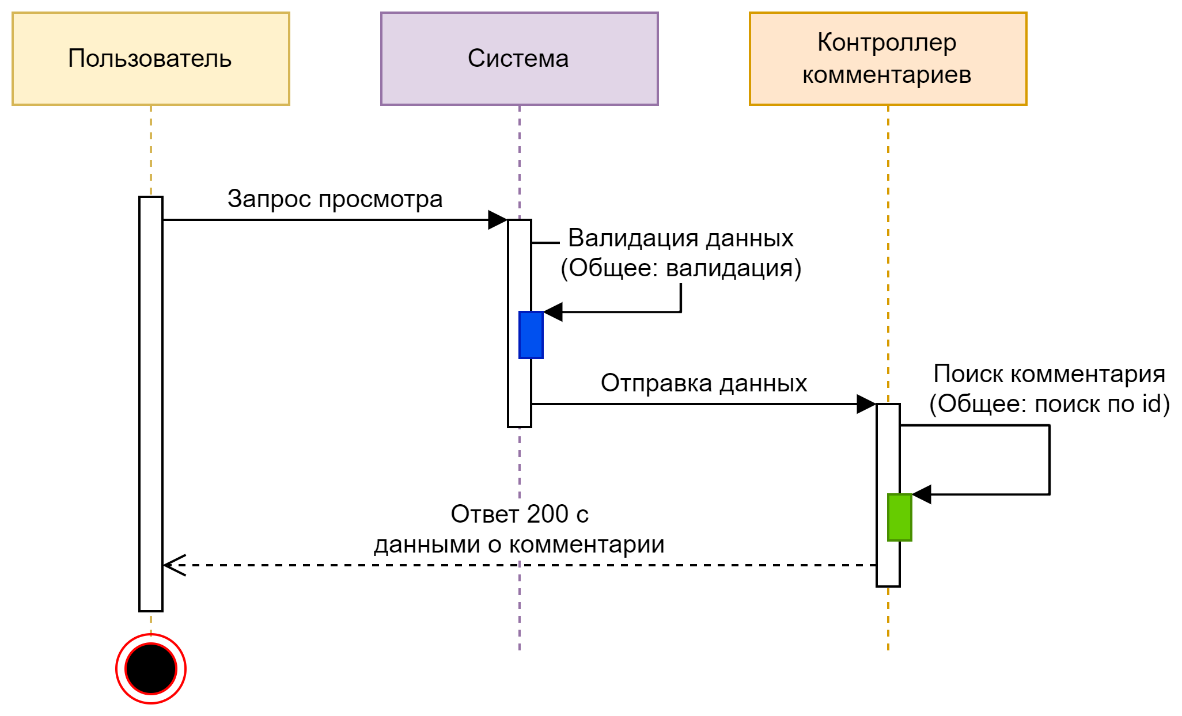


Рисунок 16 — Последовательность запроса просмотра комментария

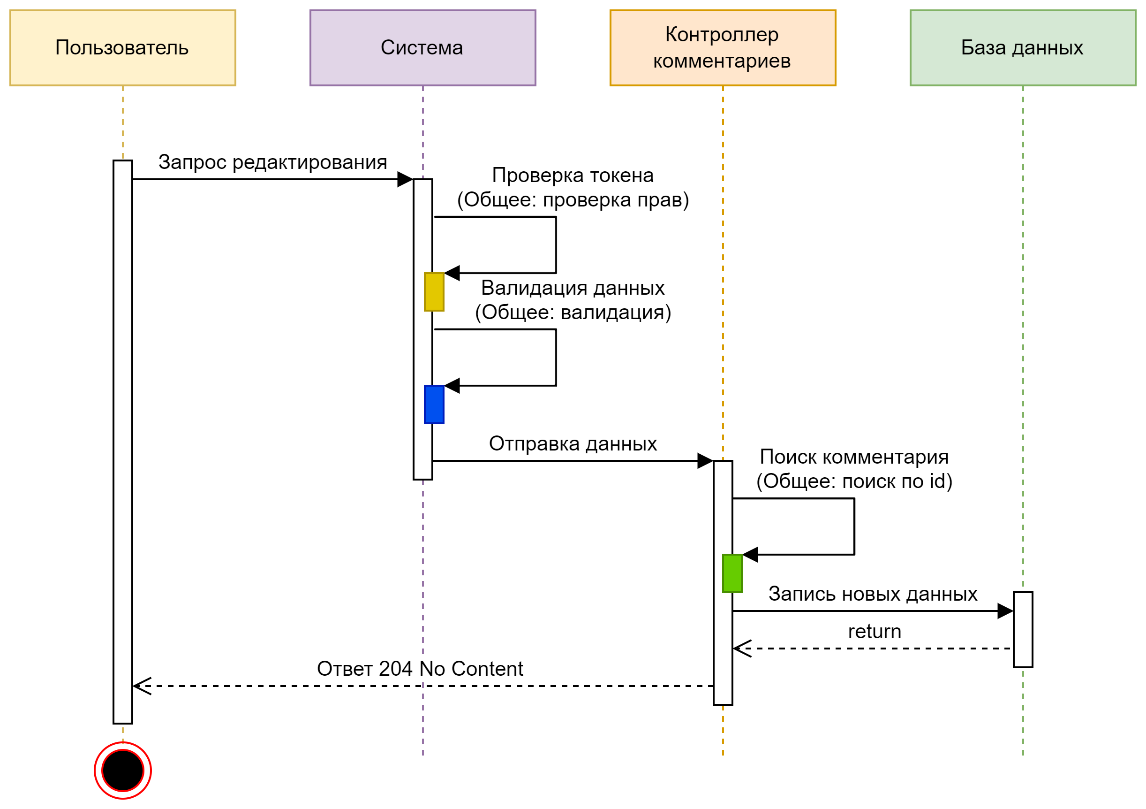


Рисунок 17 — Последовательность запроса редактироавания комментария

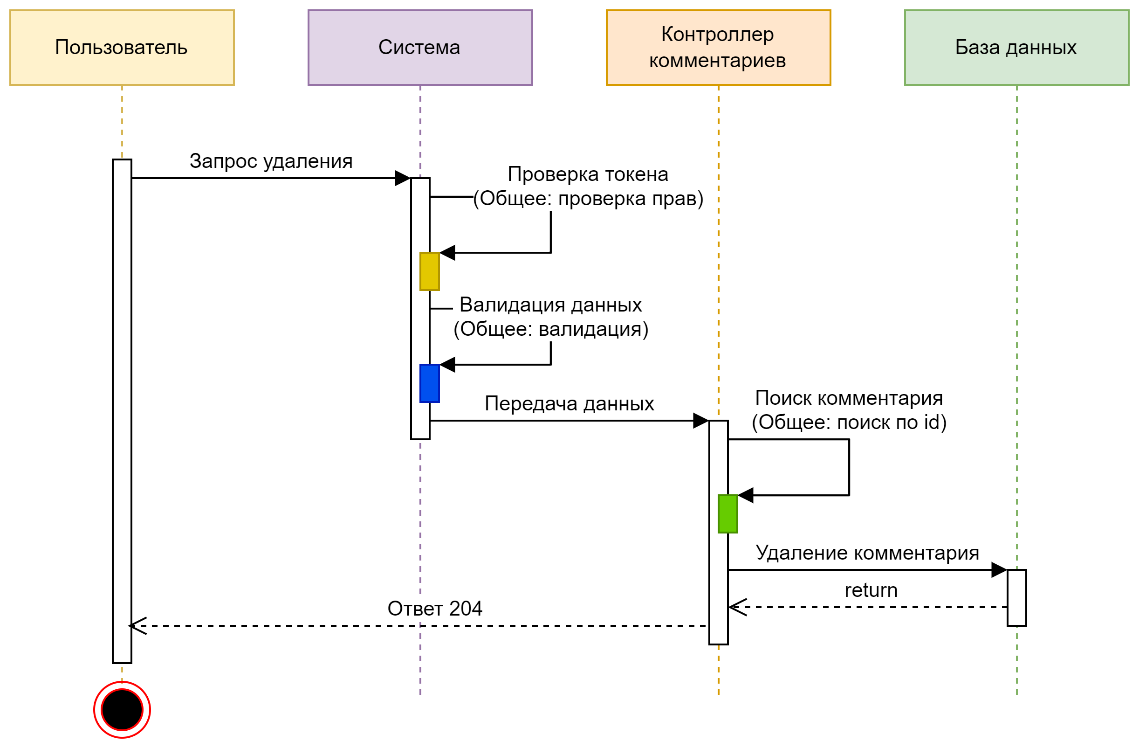


Рисунок 18 — Последовательность запроса удаления комментария

Новости

Функциональные последовательности запросов для взаимодействия с новостями:

1. Создание новой новости (*см. Рисунок 19*).
2. Редактирование (*см. Рисунок 20*).
3. Просмотр (*см. Рисунок 21*).
4. Удаление (*см. Рисунок 22*).

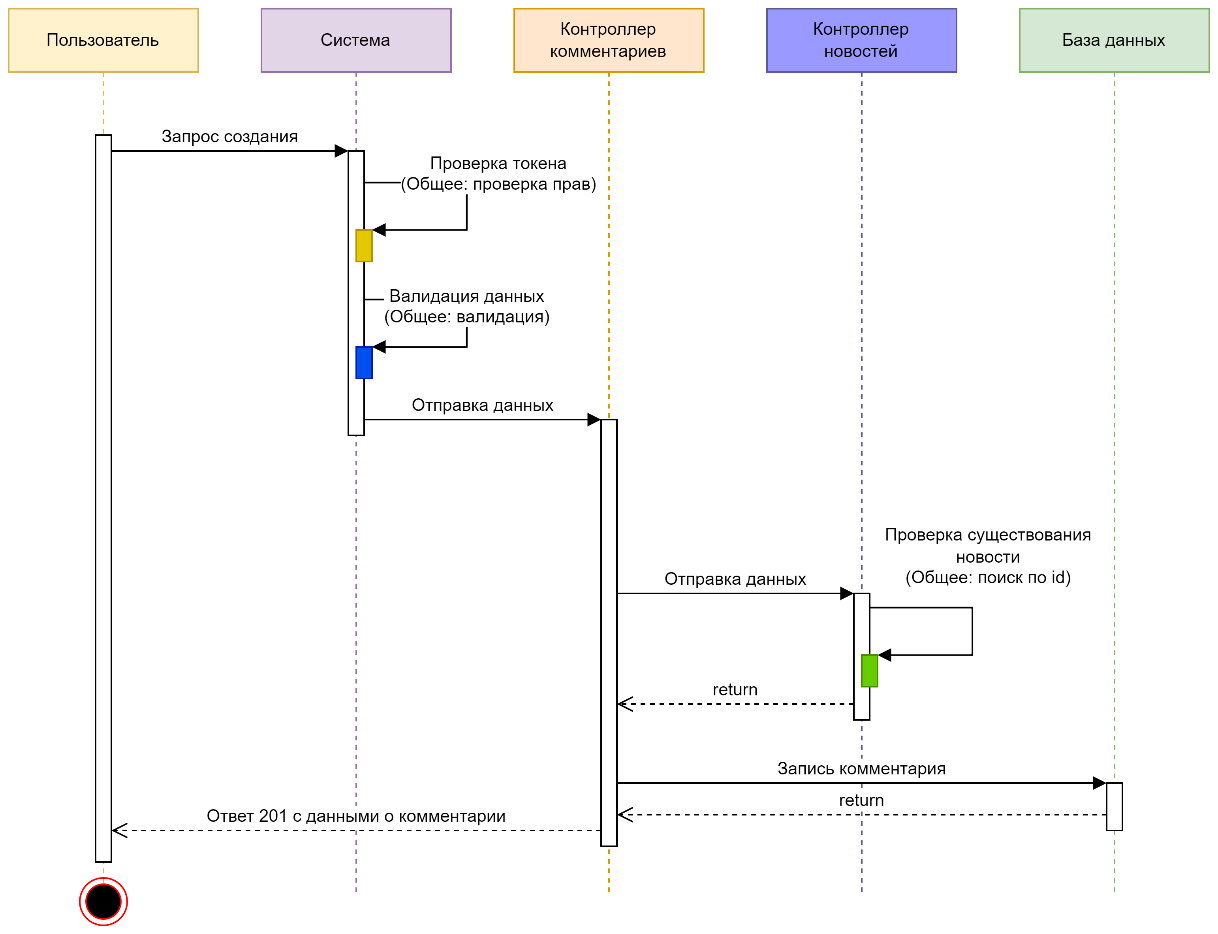


Рисунок 19 — Последовательность создания новости

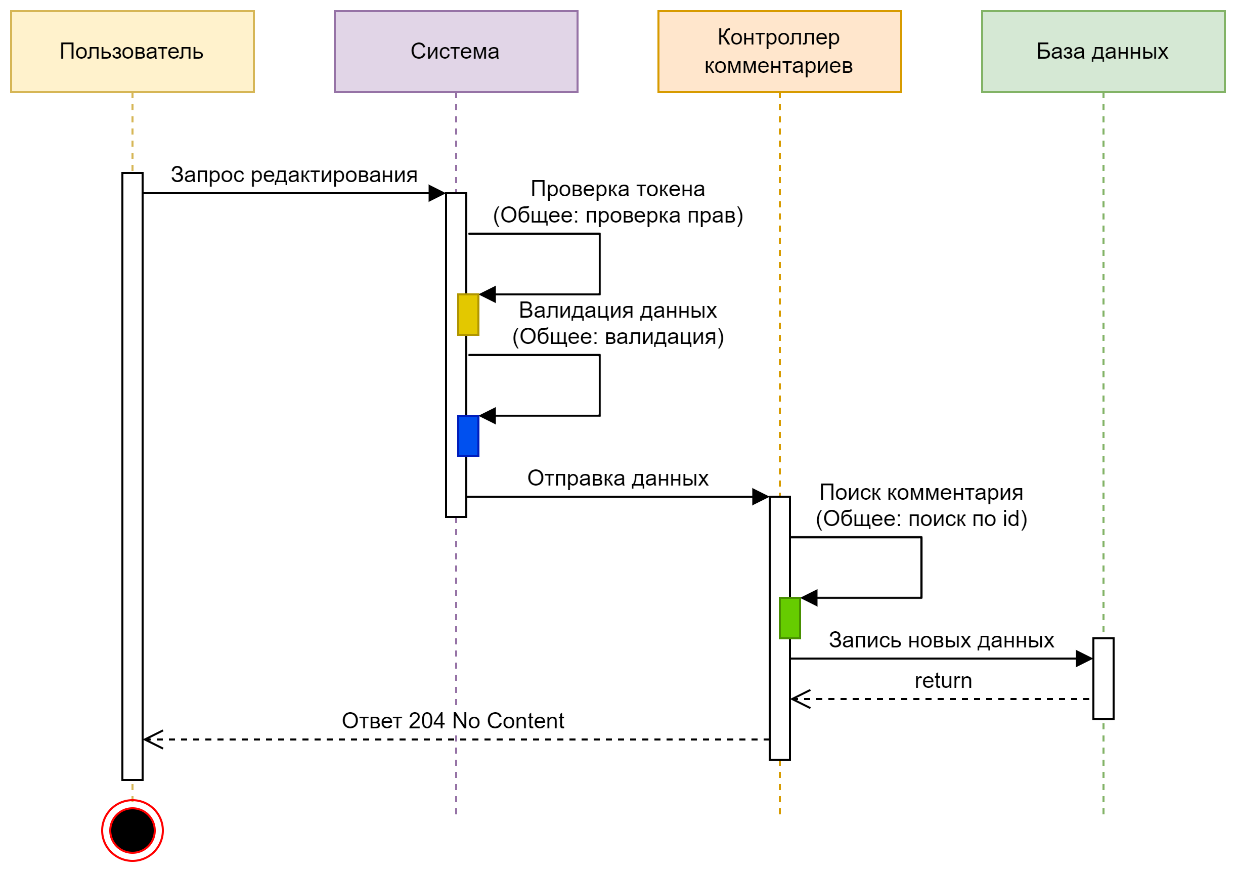


Рисунок 20 — Последовательность запроса редактирования новости

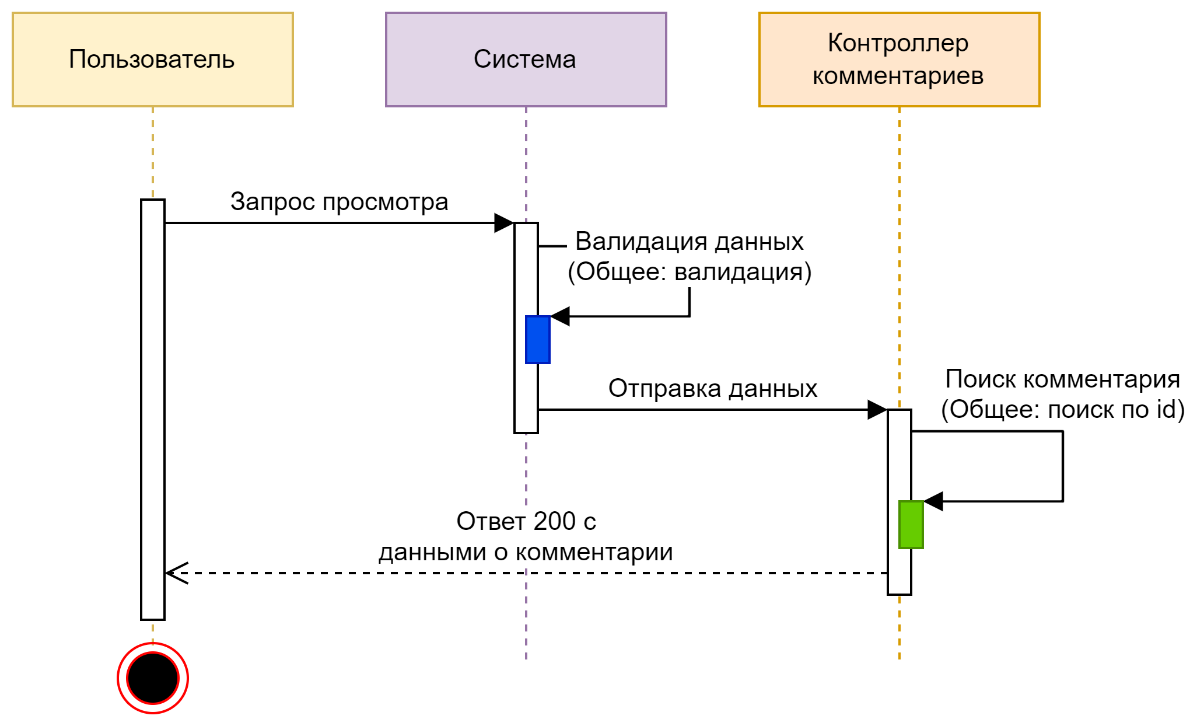


Рисунок 21 — Последовательность запроса просмотра новости



Рисунок 22 — Последовательность запроса удаления новости

Изображения

Последовательность запроса загрузки изображения на сервер представлена на Рисунке 23.

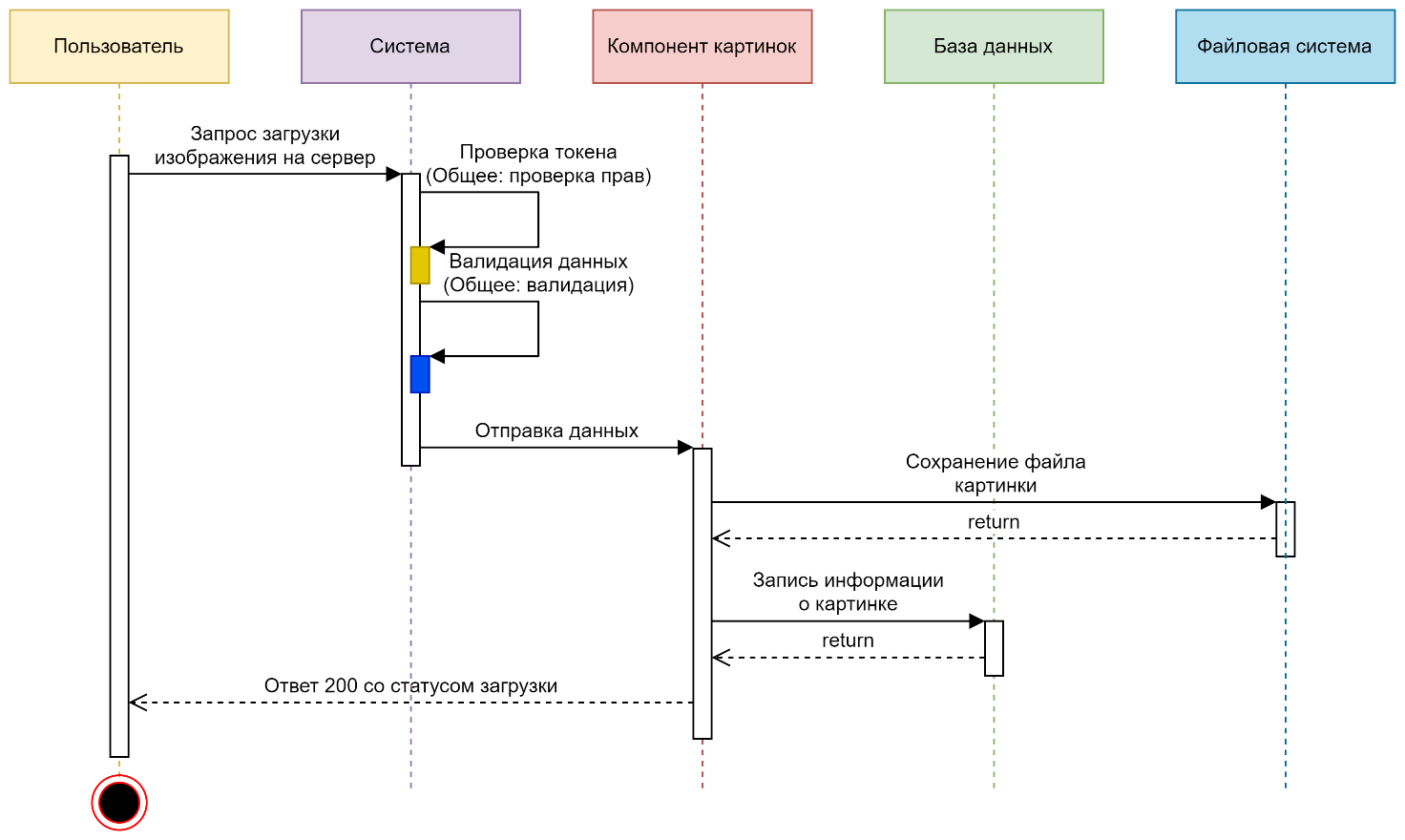


Рисунок 23 — Псоледовательность запроса загрузки изображения на сервер

Категории

Функциональные последовательности запросов для взаимодействия с категориями:

1. Создание (см. Рисунок 24).
2. Просмотр списка категорий (см. Рисунок 25).
3. Удаление (см. Рисунок 26).

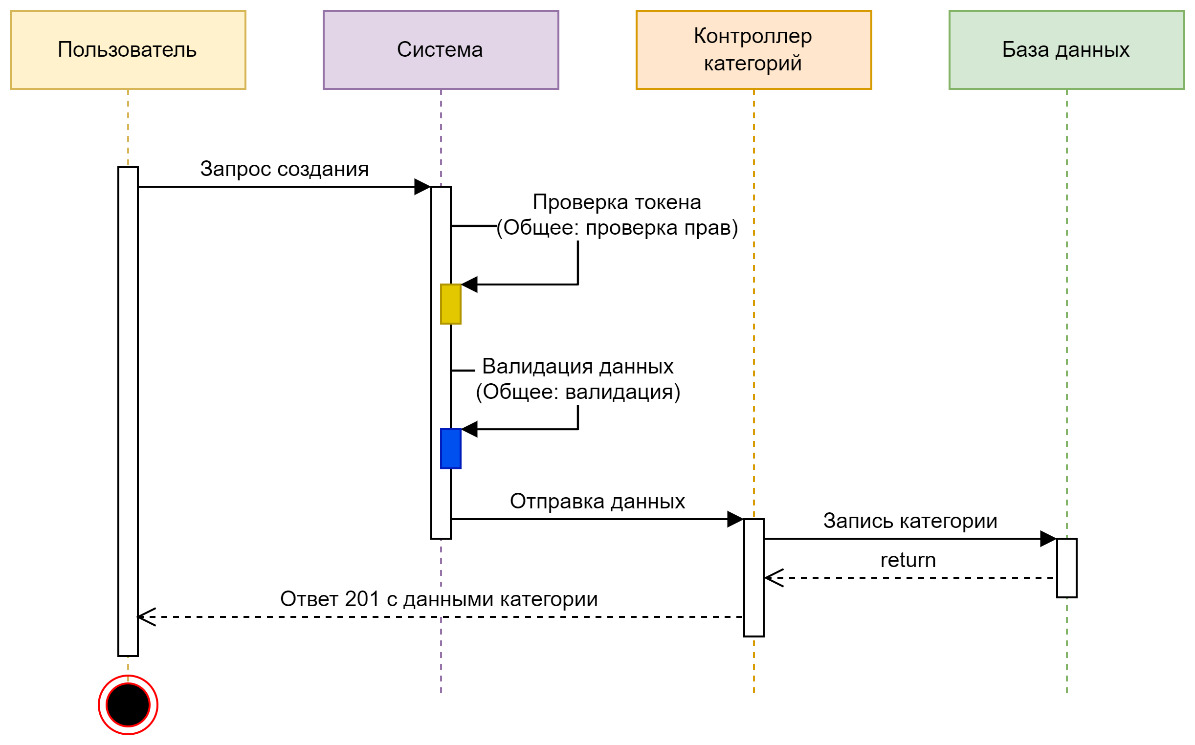


Рисунок 24 — Последовательность запроса создания категории

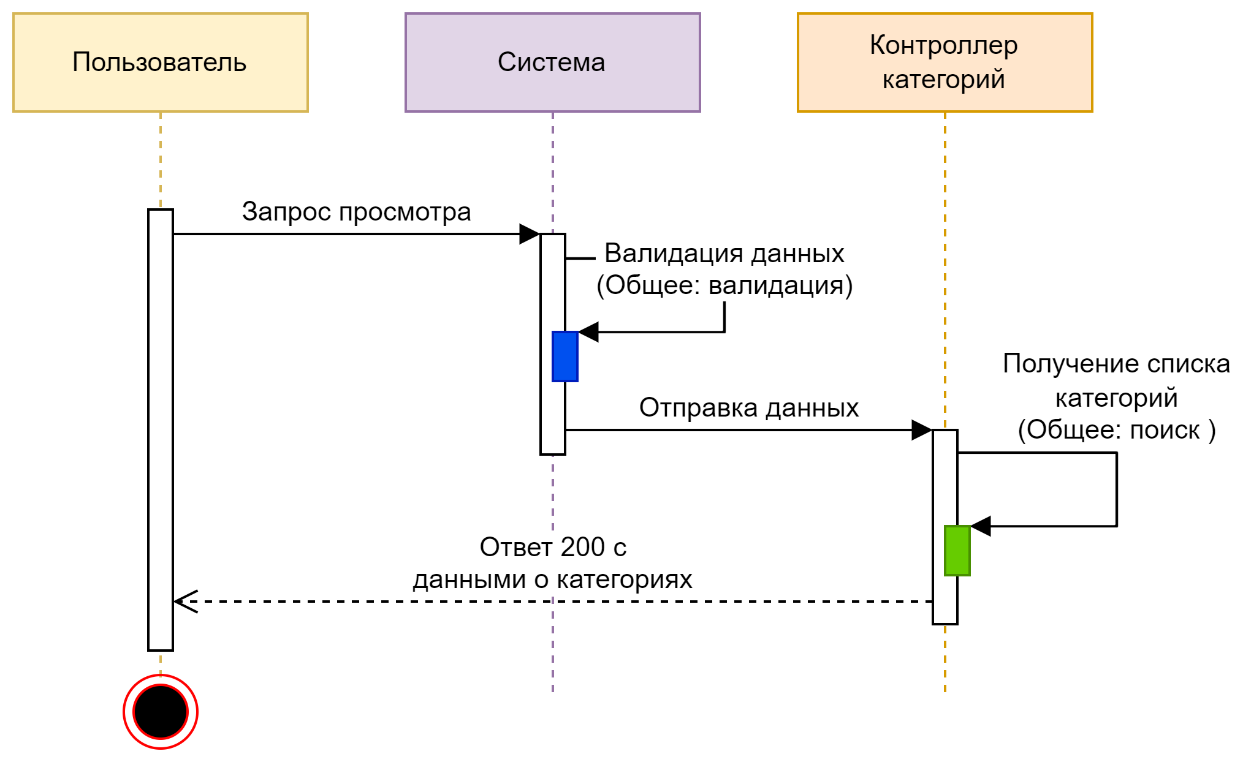


Рисунок 25 — Последовательность запроса просмотра списка категорий

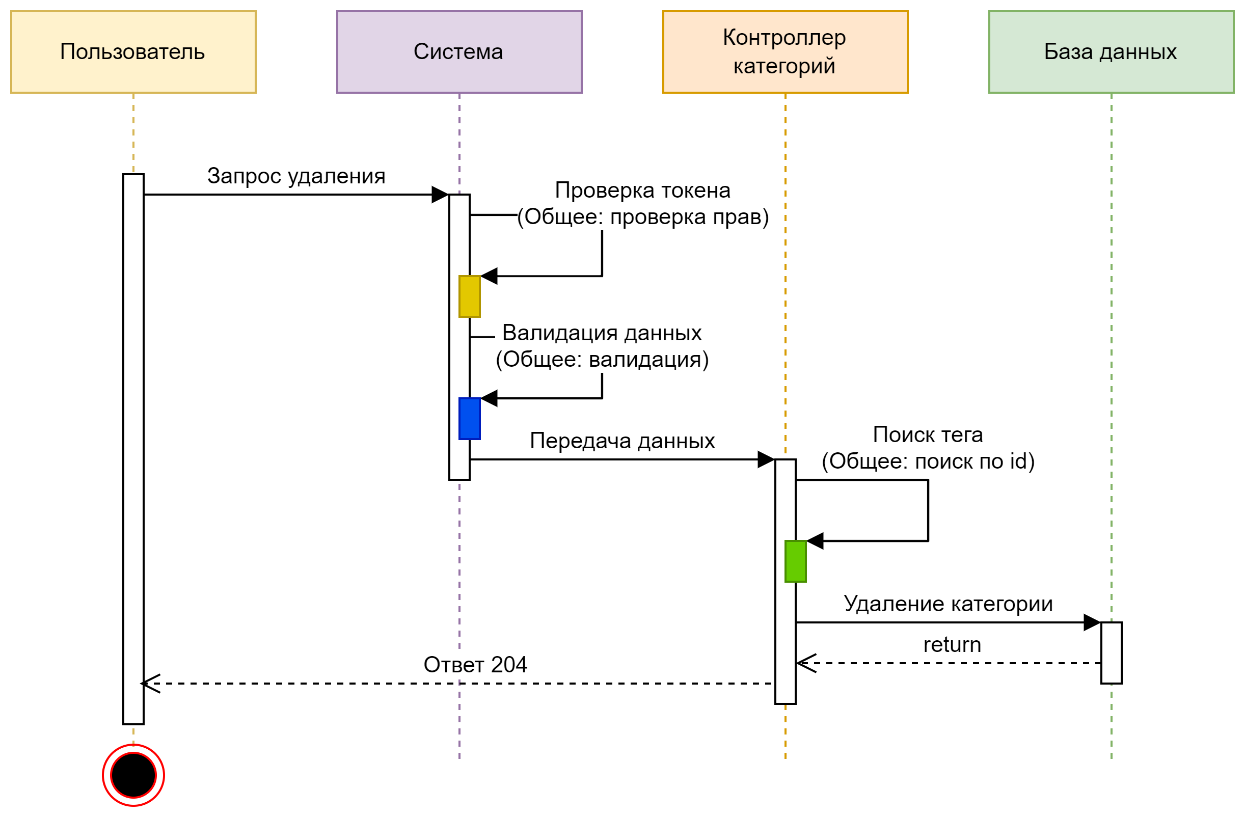


Рисунок 26 — Последовательность запроса удаления категории

Технологический стек

*Таблица 1. Используемый технологический стек*

|  |  |  |
| --- | --- | --- |
| *Серверная часть* | | |
| **Описание** | **Наименование** | **Версия** |
| Язык программирования | PHP | 8.3 |
| Фреймворк для разработки | Laravel | 11 |
| Система управления базами данных | MySQL | 8.0.30 |
| *Мобильное приложение* | | |
| **Описание** | **Наименование** | |
| Язык программирования | С# | |
| Фреймворк для разработки | .NET MAUI | |

Хранимые данные

1. **Токены:** идентифицирующая строка и связь с пользователем.
2. **Пользователи:** фамилия, имя, дата рождения, email, никнейм, логин, хеш пароля.
3. **Роли:** кодовое наименование.
4. **Новости:** заголовок, основной текст (содержимое новости), код пользователя (автора), дата создания, дата редактирования, причина редактирования.
5. **Города:** наименование.
6. **Теги:** наименование.
7. **Реакции:** наименование, путь к изображению.
8. **История просмотров:** связанная новость и пользователь, дата и время просмотра.
9. **Комментарии:** текст комментария, связанный пользователь, связанная новость.
10. **Жалобы:** пояснение, тип, пожаловавшийся пользователь, связанный комментарий, статус рассмотрения.

Требования к REST API

Общие требования

API разрабатываемого сервиса «MeNews» должно быть доступно с других доменов.

Все функции, кроме аутентификации, просмотра списка новостей, информации о конкретной новости и поиска, доступны только авторизированным пользователям.

Идентификация пользователя организована посредством передачи уникальной строки (далее *токена*) в header’е запроса в поле *Authorization*, полученной после успешной авторизации в системе.

В примерах будут использоваться следующие переменные:

* **{{host}}** – адрес информационного продукта;
* **{{path}}** – основная часть URL, обозначающая конкретное местоположение ресурсов и обеспечивающая абстрактную идентификацию расположения ресурса.

Виды ответов при ошибках

1. При попытке получить доступ к функциям системы, требующим авторизации, и использовании недействительного токена, во всех запросах необходимо возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 403 Forbidden **Content-Type:** application/json  **Body:** {  "message": "INVALID\_TOKEN",  "code": 401,  "path": "/{{path}}"  } |

1. В случае ошибок с валидацией данных во всех запросах необходимо возвращать следующее тело ответа:

|  |
| --- |
| **Response** |
| **Status:** 400 Bad Request **Content-Type:** application/json  **Body:** {  "message": "VALIDATION",  "code": 400,  "errors": [  {  "schema": {  <validation\_restriction\_name>: <validation\_restriction>  "type": <validation\_field\_type>  },  "path": <field>,  "value": <field\_value>,  "message": <error\_message>  },  <…>  ],  "path": <request\_path>  } |

В поля <validation\_restriction\_name> и <validation\_restriction> подставляется наименование ограничения, вызвавшего ошибку, и значение ограничения соответственно.

В поля <field>, <field\_value>, <error\_message> указываются имя поля, вызвавшего ограничение, значение этого поля и сообщение об ошибке соответственно.

В <request\_path> подставляется путь относительно домена, где произошла ошибка.

1. При попытке получить несуществующий ресурс возвращать ответ следующего вида:

|  |
| --- |
| **Response** |
| **Status:** 404 Not Found |

Пользователи

Авторизация

Запрос для аутентификации пользователя в системе. При отправке передаётся логин и пароль.

Если переданы корректные данные, то есть переданная пара логин/пароль присутствует в сервисе «MeNews» (далее *ИС*), то ИС возвращает клиенту сгенерированный токен со следующими полями, приведёнными ниже в примере запроса-ответа.

В случае возникновения ошибок, вернуть соответствующий ответ, описанный в *Видах ответов при ошибках*.

|  |  |
| --- | --- |
| **Request** | Response |
| **URL**: {{host}}/login  **Method**: POST  **Body**: {  "login": {{login}},  "password": {{password}}  } | Успех  **Status:** 200 OK  **Body:** {  "id": 1,  "role": {  "id": 1,  "code": "user"  },  "email": "zamelane@vk.com",  "firstName": "Евгений",  "lastName": "Малинин",  "nickname": "zamelane",  "birthday": "2004-07-20T00:00:00.0000000",  "avatar": 1,  "token": "0DlTeqXqI1RcO7Q7JhOppLrl"  }  Ошибка валидации полей  **Формат ответа из общих требований**  Неправильные логин и/или пароль  **Status:** 401 Unauthorized  **Content-Type:** application/json  **Body:** {  "message": "INVALID\_DATA",  "code": 401,  "path": "/login"  } |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Описание** | **Пример** |
| 1 | login | строка | да | Логин | admin |
| 2 | password | строка | да | Пароль | Admin! |

Регистрация

Запрос для регистрации нового пользователя в системе.

Если переданы корректные данные и переданный логин/никнейм не присутствуют в ИС, то клиенту возвращается сообщение об успешной регистрации, приведённом ниже в примере запроса-ответа, который также содержит токен для аутентификации пользователя.

В случае возникновения ошибок, вернуть ответ, описанный в *Видах ответов при ошибках*.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: {{host}}/reg  **Method**: POST  **Body**: {  "login": {{login}},  "password": {{password}},  "firstName": {{firstName}},  "lastName": {{lastName}},  "birthday": {{birthday}}  } | Успех  **Status:** 200 OK  **Body:** {  "id": 1,  "role": {  "id": 1,  "code": "user"  },  "email": "zamelane@vk.com",  "firstName": "Евгений",  "lastName": "Малинин",  "nickname": "zamelane",  "birthday": "2004-07-20T00:00:00.0000000",  "token": "0DlTeqXqI1RcO7Q7JhOppLrl"  }  Ошибка валидации полей  **Формат ответа из общих требований**  Логин/никнейм занят  **Status:** 400  **Content-Type:** application/json  **Body:** {  "message": "USERNAME\_IS\_BUSE",  "code": 401,  "path": "/reg"  } |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | login | строка | да | мин: 2, макс: 75, уникальный | Логин | admin |
| 2 | password | строка | да | мин: 8, макс: 75 | Пароль | Admin! |
| 3 | firstName | строка | да | мин: 2, макс: 75 | Имя | Иванов |
| 4 | lastName | строка | да | мин: 2, макс: 75 | Фамилия | Пупкин |
| 5 | birthday | строка | да | Дата в формате DD-MM-YYYY | Дата рождения | 21-12-2024 |

Выход с одного устройства

Запрос для закрытия активной сессии путём удаления токена пользователя. При отправке передаётся токен, полученный при аутентификации, для завершения авторизации пользователя в системе.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: {{host}}/logout  **Method**: GET  **Headers**:   * Content-Type: application/json * Secret: <token> | Успех  **Status:** 204 No Content  Токен отсутствует в системе  **Формат ответа из общих требований** |

Выход со всех устройств

Запрос для закрытия всех активных сессий путём удаления токенов пользователя. При отправке передаётся любой активный на момент запроса токен, полученный при аутентификации, для завершения всех авторизаций пользователя в системе.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: {{host}}/logout/all  **Method**: GET  **Headers**:   * Content-Type: application/json * Secret: <token> | Успех  **Status:** 204 No Content  Токен отсутствует в системе  **Формат ответа из общих требований** |

Создание пользователя

Запрос на добавление пользователя в систему. При отправке передаются:

* **{login}** – логин пользователя, без пробелов. Строка, от 5 до 15 символов.
* **{password}** –пароль пользователя. Строка, от 7 до 70 символов.
* **{firstName}** – имя пользователя. Строка, от 2 до 70 символов.
* **{lastName}** – фамилия пользователя. Строка, от 2 до 70 символов.
* **{birthday}** –дата рождения пользователя. Форматированная строка даты.
* **{role}** – роль пользователя. Строка. Должна быть в системе.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: {{host}}/users  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body**: {  "login": **{login}**,  "password": **{password}**,  "firstName": **{firstName}**,  "lastName": **{lastName}**,  "birthday": **{birthday}**,  "role": **{role}**  } | Успех  **Status:** 204 No Content  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований**  Пользователь уже существует  **Формат ответа из общих требований**  Недостаточно прав  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | login | строка | да | мин: 2, макс: 75, уникальный | Логин | admin |
| 2 | password | строка | да | мин: 8, макс: 75 | Пароль | Admin! |
| 3 | firstName | строка | да | мин: 2, макс: 75 | Имя | Иванов |
| 4 | lastName | строка | да | мин: 2, макс: 75 | Фамилия | Пупкин |
| 5 | birthday | строка | да | Дата в формате  DD-MM-YYYY | Дата рождения | 21-12-2024 |
| 6 | role | строка | нет | Код одной из ролей: user, reader, admin | Роль пользователя. По умолчанию: user | reader |

Удаление пользователя

Запрос на удаление пользователя из системы. При запросе передаётся идентификатор пользователя **{user\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/users/**{user\_id}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Пользователь не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Недостаточно прав  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | userId | число | да | существует | Код пользователя | 25 |

Просмотр пользователя

Запрос для просмотра пользователя в системе по его идентификатору **{user\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/users/**{user\_id}**  **Method**: GET  **Headers**:   * Content-Type: application/json | Успех  **Status:** 200 OK  **Body:** {  "id": 1,  "role": {  "id": 1,  "code": "user"  },  "firstName": "Евгений",  "lastName": "Малинин",  "nickname": "zamelane",  "birthday": "2004-07-20T00:00:00.0000000",  "avatar": 1  }  Пользователь не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | userId | число | да | существует | Код пользователя | 25 |

Редактирование пользователя

Запрос для редактирования пользователя в системе по его идентификатору **{user\_id}**. При отправке передаётся хотя бы одно из перечисленных полей (см. входные параметры).

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/users/**{user\_id}**  **Method**: PUT  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "login": "Zamelane"  } | Успех  **Status:** 204 No Content  Пользователь не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Недостаточно прав  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | login | строка | да | мин: 2, макс: 75, уникальный | Логин | Zamelane |
| 2 | password | строка | да | мин: 8, макс: 75 | Пароль | Admin! |
| 3 | firstName | строка | да | мин: 2, макс: 75 | Имя | Иванов |
| 4 | lastName | строка | да | мин: 2, макс: 75 | Фамилия | Пупкин |
| 5 | birthday | строка | да | Дата в формате  DD-MM-YYYY | Дата рождения | 21-12-2024 |
| 6 | role | строка | нет | Код одной из ролей: user, reader, admin. Должен иметь право на смену роли | Роль пользователя. По умолчанию: user | reader |

Загрузить аватар пользователю

Запрос на загрузку изображения-аватара пользователя по его идентификатору **{user\_id}.**

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/users/**{user\_id}**/avatar  **Method**: POST  **Headers**:   * Content-Type: multipart/form-data * Authorization: Bearer **<token>**   **Body:** {  "image": **{{картинка}}**  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "successful": [  "id": 1,  "file\_info": {  "name": test.jpg,  "upload\_date": "2024-10-29",  "size": 127345,  "width": 1280,  "height": 720  }  "author": {  "id": 1,  "nickname": "zamelane"  }  ],  "errored": [  "name": "test.jpg".  "message": "large size"  ]  }  Пользователь не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Недостаточно прав  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | userId | число | да | существует | Код пользователя | 25 |
| 2 | image | файл | да | mimes: jpeg,png,gif | Картинка | <image> |

Теги

Создание (добавление) тега

Запрос для создания нового тега в системе. При отправке передаются:

* **{name}** – текст тега, без пробелов. Строка, от 1 до 15 символов.
* **{description}** – описание тега. Строка, от 10 до 70 символов.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: {{host}}/tags  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body**: {  "name": **{name}**,  "description": **{description}**  } | Успех  **Status:** 204 No Content  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | name | строка | да | мин: 2, макс: 15, без пробелов | Название тега | РЕГИОН70 |
| 2 | description | строка | да | мин: 10, макс: 70 | Описание тега | Новость взята из паблика «РЕГИОН 70» по Томской области |

Удаление тега

Запрос для удаления тега по его идентификатору. Необходимо передать **{id}** тега, который нужно удалить, в адресе запроса.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/tags/**{id}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 No Content  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований**  Тег не найден  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | id | число | да | существует | Код тега | 12 |

Просмотр информации о теге

Запрос для получения информации о теге по его идентификатору **{id}**.

При успешном запросе будет возвращена информация о теге, включая его идентификатор, название и описание. Если тег с указанным идентификатором не найден, система вернет ошибку *404*.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/tags/**{id}**  **Method**: GET  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "name": "Россия",  "description": "Новость относится к стране Россия"  }  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований**  Тег не найден  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | id | число | да | существует | Код тега | 12 |

Города

Создание

Запрос для добавления нового города в систему.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/cities  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "name": "Томск"  } | Успех  **Status:** 204 No Content  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | name | строка | да | мин: 2, макс: 25 | Наименование города или населённого пункта | Томск |

Просмотр

Запрос для получения информации о городе по его идентификатору **{cityId}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/cities/**{cityId}**  **Method**: GET  **Headers**:   * Content-Type: application/json | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "name": "Томск"  }  Город не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Токен отсутствует в системе  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | cityId | число | да | существует | Код населённого пункта | 10 |

Редактирование

Запрос для редактирования информации о городе по его идентификатору **{cityId}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/cities/**{cityId}**  **Method**: PUT  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "name": "Новосибирск"  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "name": "Новосибирск"  }  Город не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | cityId | число | да | существует | Код населённого пункта | 10 |
| 2 | name | строка | да | не существует (уникально) | Наименование населённого пункта | Новосибирск |

Удаление

Запрос для удаления города по его идентификатору. Необходимо передать идентификатор **{cityId}** города, который нужно удалить.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/cities/**{cityId}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Город не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | cityId | число | да | существует | Код населённого пункта | 10 |

История

Просмотр истории чтения

Запрос для добавления новости в историю просмотра.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/history  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "data": [  {  "news": {  "id": 1,  "title": "Заголовок новости"  },  "read\_date": "21.12.2024",  "read\_time ": "20:10:00.00 "  },  <…>  ],  "limit": 5,  "total": 25,  "page": 1  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Избранное

Создание

Запрос для добавления новости в избранное.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/favourites  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "news\_id": "1"  } | Успех  **Status:** 201 CREATED  **Content-Type:** application/json  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | news\_id | число | да | существует | Код новости | 1 |

Просмотр избранного

Запрос для просмотра всех избранных записей.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/favourites  **Method**: GET  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "data": [  {  "news": {  "id": 1,  "title": "Заголовок новости"  },  "added\_date": "21.12.2024"  },  <…>  ],  "limit": 5,  "total": 25,  "page": 1  }  Реакция не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Реакции

Создание

Запрос для создания нового типа реакции в системе.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/reactions  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "symbol": "😄",  "description": "Радость"  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "symbol": "😄",  "description": "Радость"  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | symbol | строка | да | Мин: 1, макс: 1, не существует | Эмодзи реакции | 😄 |
| 2 | description | строка | да | мин: 5, макс: 70 | Описание реакции | Радость |

Удаление

Запрос для удаления типа реакции по его идентификатору. Необходимо передать **{reactionId}** реакции, которую нужно удалить.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/reactions/**{reactionId}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Реакция не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | reactionId | число | да | существует | Код реакции | 5 |

Просмотр

Запрос для получения информации о типе реакции по его идентификатору **{reactionId}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/reactions/**{reactionId}**  **Method**: GET  **Headers**:   * Content-Type: application/json | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "symbol": "😄",  "description": "Радость"  }  Реакция не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | reactionId | число | да | существует | Код реакции | 5 |

Реакции к новости

Добавление

Запрос для добавления реакции к конкретной новости по её идентификатору **{news\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/**{news\_id}**/reactions  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "reaction\_id": "1"  } | Успех  **Status:** 204 No Content  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | newsId | число | да | существует | Код новости | 1 |
| 2 | reactionId | число | да | существует | Код реакции | 5 |

Удаления

Запрос для удаления реакции к конкретной новости по ее идентификатору.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/**{news\_id}**/reactions  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Реакция уже удалена  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | newsId | число | да | существует | Код новости | 1 |

Новости

Создание

Запрос для создания новости.

Если данные корректны, система создаст новость в системе и вернёт её идентификатор, заголовок, содержание и связанные теги.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "title": "Некий заголовок",  "content": "Некий текст markdown",  "tags": [1, 2, 3, 4, 5]  } | Успех  **Status:** 201 Created  **Content-Type:** application/json  **Body:** {  "id": 1,  "title": "Некий заголовок",  "content": " Некий текст markdown",  "tags": [1, 2, 3, 4, 5]  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | title | строка | да | мин: 10, макс: 75 | Заголовок новости | Центробанк неожиданно сохранил ключевую ставку на уровне 21% |
| 2 | content | текст | да | мин: 50 | Содержимое новости | Томский районный суд вынес решение… |
| 3 | tags | массив чисел | нет | должны существовать | Теги новости | 1,2,3 |

Редактирование

Запрос для редактирования существующей новости по её идентификатору **{news\_id}**.

Если данные корректны, система обновит информацию о новости и вернет её идентификатор, заголовок, содержание и связанные теги.

Если новость не найдена, будет возвращена ошибка 404.

Если переданы некорректные данные, вернется ошибка 400. Если токен не предоставлен, будет возвращена ошибка 401.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/**{news\_id}**/reactions  **Method**: PATCH  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "title": "Некий заголовок",  "content": "Некий текст markdown",  "tags": [1, 2, 3, 4, 5]  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "title": "Новая история",  "content": "Это содержание новой новости.",  "tags": [2, 3]  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований**  Новость не найдена  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | news\_id | число | да | существует | Код новости | 55 |
| 2 | title | строка | нет | мин: 10, макс: 75 | Заголовок новости | Центробанк неожиданно сохранил ключевую ставку на уровне 21% |
| 3 | content | текст | нет | мин: 50 | Содержимое новости | Томский районный суд вынес решение… |
| 4 | tags | массив чисел | нет | должны существовать | Теги новости | 1,2,3 |

Удаление

Запрос для удаления новости по её идентификатору. Необходимо передать **{news\_id}** новости, которую нужно удалить.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/**{news\_id}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Delete  Новость не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | news\_id | число | да | существует | Код новости | 55 |

Комментарии

Создание

Запрос на создание (написание) комментария от имени активного пользователя.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/comments  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "comment": "Очень важная новость, хорошо, что она была освещена!"  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "author": {  "id": 1,  "nickname": "zamelane"  }  "comment": "Очень важная новость, хорошо, что она была освещена!",  "createDateTime": "2024-10-29T10:15:10.0000000",  "updateDateTime": "null"  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | comment | строка | да | мин: 1, макс: 255 | Комментарий | Очень важная новость, хорошо, что она была освещена! |

Просмотр

Запрос на просмотр информации о комментарии по его идентификатору **{commentId}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/comments/**{commentId}**  **Method**: GET  **Headers**:   * Content-Type: application/json | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "author": {  "id": 1,  "nickname": "zamelane"  }  "comment": "Очень важная новость, хорошо, что она была освещена!",  "createDateTime": "2024-10-29T10:15:10.0000000",  "updateDateTime": "null"  }  Комментарий не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | commentId | число | да | существует | Код комментария | 75 |

Редактирование

Запрос на редактирование комментария к новости по его идентификатору **{commentId}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/comments/**{commentId}**  **Method**: PUT  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "comment": "Очень важная новость, хорошо, что она была освещена! UPD: я обновил этот комментарий."  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "author": {  "id": 1,  "nickname": "zamelane"  }  "comment": "Очень важная новость, хорошо, что она была освещена! UPD: я обновил этот комментарий.",  "createDateTime": "2024-10-29T10:15:10.0000000",  "updateDateTime": "2024-10-30T09:10:40.0000000"  }  Комментарий не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | commentId | число | да | существует | Код комментария | 75 |
| 2 | comment | строка | да | мин: 1, макс: 255 | Комментарий | Очень важная новость, хорошо, что она была освещена! UPD: я обновил этот комментарий. |

Удаление

Запрос на удаление комментария к новости.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/news/comments/**{commentId}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Новость не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | commentId | число | да | существует | Код комментария | 75 |

Жалобы

Просмотр списка

Запрос на получение списка жалоб на комментарии.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/complaints  **Method**: GET  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** [  {  "id": 1,  "author": {  "id": 1,  "nickname": "zamelane"  },  "comment": "Очень важная новость, хорошо, что она была освещена!",  "createDateTime": "2024-10-29T10:15:10.0000000",  "updateDateTime": "null",  "complaint": {  "id": 1,  "reason": {  "id": 1,  "code": "offence"  },  "author": {  "id": 5,  "nickname": "chelovek"  },  "createDateTime": "29T10:15:10.0000000"  }  },  { … }  ]  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Создание

Запрос на создание жалобы на комментарий по его идентификатору **{comment\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/comments/**{comment\_id}**/ complaints  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>**   **Body:** {  "description": "Очень обидно!",  "reason\_id": 1  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "description": "Очень обидно!",  "reason\_id": 1  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | comment\_id | число | да | существует | Код комментария | 75 |
| 2 | reason\_id | число | да | существует | Код причины | 1 |
| 3 | description | строка | нет | мин: 2, макс: 255 | Описание проблемы | Очень обидно! |

Удаление

Запрос на удаление жалобы из системы по её идентификатору **{complaint\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/complaints/**{complaint\_id}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Комментарий не найден  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | complaint\_id | число | да | существует | Код жалобы | 1 |

Баны

Просмотр бана

Запрос для просмотра бана пользователя.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/bans/**{banId}**  **Method**: POST  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 201 CREATED  **Content-Type:** application/json  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | userId | число | да | существует | Код пользователя | 1 |
|  | banId | число | да | существует | Код бана | 1 |

Просмотр банов

Запрос для просмотра всех банов пользователя.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/users/**{userId}**/bans  **Method**: GET  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "data": [  {  "complaint": {  "id": 1,  "reason": {  "id": 1,  "name": "Оскорбление"  },  "author": {  "id": 1,  "firstName": "Геннадий",  "lastName": "Кукин",  "role": "user",  "avatar": null,  "login": "ivanov",  "email": "ivanov@mail.ru",  "birthDay": "21.12.2000"  },  "comment": {  "id": 1,  "user": {  "id": 1,  "firstName": "Иванов",  "lastName": "Пупкин",  "role": "user",  "avatar": null  },  "content": "Вы все плохиши!",  "created\_at": "10.12.2024",  "updated\_at": "10.12.2024"  },  }  "ban\_end\_date": "21.12.2024"  },  <…>  ],  "limit": 5,  "total": 25,  "page": 1  }  Реакция не найдена  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | userId | число | да | существует | Код пользователя | 1 |

Изображения

Информация об изображении

Получение информации об изображении по идентификатору изображения **{img\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/imgs/**{img\_id}**  **Method**: GET  **Headers**:   * Content-Type: application/json | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "id": 1,  "file\_info": {  "name": test.jpg,  "upload\_date": "2024-10-29",  "size": 127345,  "width": 1280,  "height": 720  }  "author": {  "id": 1,  "nickname": "zamelane"  }  }  Изображение не найдено  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | img\_id | число | да | существует | Код изображения | 15 |

Просмотр изображения

Запрос для скачивания изображения по его идентификатору **{img\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/imgs/**{img\_id}**/file  **Method**: GET | Успех  **Status:** 200 OK  **Content-Type:** image/<"jpeg"|"png"|…>  **Content-Disposition:** attachment; filename=<имя\_файла>  Изображение не найдено  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | img\_id | число | да | существует | Код изображения | 15 |

Загрузка изображения

Запрос для загрузки на сервер изображения.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/imgs  **Method**: POST  **Headers**:   * Content-Type: multipart/form-data * Authorization: Bearer **<token>**   **Body:** {  "image": **{{картинка}}**  } | Успех  **Status:** 200 OK  **Content-Type:** application/json  **Body:** {  "successful": [  "id": 1,  "file\_info": {  "name": test.jpg,  "upload\_date": "2024-10-29",  "size": 127345,  "width": 1280,  "height": 720  }  "author": {  "id": 1,  "nickname": "zamelane"  }  ],  "errored": [  "name": "test.jpg".  "message": "large size"  ]  }  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** |
| 1 | image | файл | да | mimes: jpeg,png,gif | Картинка |

Удаление

Запрос для удаления изображения по его идентификатору **{img\_id}**.

|  |  |
| --- | --- |
| Request | Response |
| **URL**: **{{host}}**/imgs/**{img\_id}**  **Method**: DELETE  **Headers**:   * Content-Type: application/json * Authorization: Bearer **<token>** | Успех  **Status:** 204 Deleted  Изображение не найдено  **Формат ответа из общих требований**  Ошибка валидации полей  **Формат ответа из общих требований**  Недействительный токен  **Формат ответа из общих требований** |

Входные параметры:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Тип** | **Обяз** | **Условия** | **Описание** | **Пример** |
| 1 | img\_id | число | да | существует | Код изображения | 15 |

Требования к мобильному приложению клиента

При открытии приложения пользователю доступны страницы, в зависимости от его роли:

* Гость:
  + Главная – страница с недавними и популярными новостями.
  + Каталог – просмотр всех новостей с возможностью их сортировки и фильтрации (через поиск или выбор категории).
  + Страница новости – отображает содержимое новости, информацию о ней.
  + Комментарии – выводит список комментариев к новости.
  + Авторизация – страница входа пользователя с кнопкой регистрации.
  + Регистрация – страница регистрации пользователя с кнопкой входа.
* Читатель:
  + Всё то же самое, что и у гостя, но без страницы авторизации и регистрации.
  + Избранное – список отложенных новостей.
  + Профиль – информация об авторизированном пользователе с возможностью редактирования.
* Репортёр:
  + Всё то же самое, что и у читателя.
  + Мастерская – список новостей, написанных репортёром.
  + Редактор новости – страница редактирования и создания новости.

Далее подробно рассматривается содержание и приблизительное оформление каждой страницы.

Главная страница

После запуска приложения, первою из страниц, которую видит пользователь, должна быть Главная страница.

На ней отображаются следующие секции:

* В самом верху страницы карусель из случайных новостей *(см. Рисунок 27)*.
* Ниже прокручиваемый список категорий с кнопкой открытия страницы категорий *(см. Рисунок 28)*.
* Затем секция «Самое читаемое» с последними популярными новостями *(см. Рисунок 29)*.
* В самом конце должен быть раздел «Последние новости», отображающем недавно добавленные репортёрами новостями *(см. Рисунок 30)*.

На каждой карточке небольшой карточки новости нужно выводить изображение, заголовок, количество просмотров и комментариев *(см. Рисунок 29)*. Для больших карточек новостей нужно дополнительно отображать описание (краткое содержимое) и увеличить занимаемое изображением новости пространство *(примерное оформление смотри на Рисунке 27)*.

|  |  |
| --- | --- |
| Рисунок 27 — Примерное оформление новости из карусели случайных новостей | Рисунок 28 — Примерное оформление блока категорий |

|  |  |
| --- | --- |
| Рисунок 29 — Примерное оформление блока популярных новостей | Рисунок 30 — Примерное оформление блока последних новостей |

Каталог

На странице каталога, доступной через NavBar (нижнюю навигационную панель), должна присутствовать поисковая строка и элементы выбора для фильтрации по категориям, тегам и выбора режима сортировки.

При вводе текста в поисковой строке, необходимо выполнять повторный поиск новостей по ключевым словам, введённым в поисковую строку, но через 500мс после окончания ввода пользователем. Пример оформления страницы каталога смотри на *Рисунке 31.*

Избранное

На странице избранного необходимо показывать отложенные пользователем новости, а в «шапке» страницы отображать их количество. При нажатии на новость нужно открывать обычную страницу просмотра новости *(см. Рисунок 32)*.

|  |  |
| --- | --- |
| Рисунок 31 — Примерное оформление страницы каталога новостей | Рисунок 32 — Примерное оформление страницы избранных новостей |

Страница просмотра новости

На странице новости (*см. Рисунок 33*) необходимо показывать:

* Шапку новости: основное изображение, количество просмотров, комментариев, название категории и заголовок новости.
* Автора новости: аватарку автора, его фамилию и имя, дату публикации и время чтения новости.
* Содержимое новости и кнопка для его редактирования (доступной только автору новости).
* Список рекомендованных к прочтению новостей (в конце страницы), *см. Рисунок 34*.

Также на этой странице должны присутствовать кнопки, дающие возможность поделиться новостью и добавить её в избранное.

|  |  |
| --- | --- |
| Рисунок 33 — Верхняя часть страницы просмотра новости | Рисунок 34 — Блок "смотреть также", нижняя часть страницы просмотра новости |

Страница создания (редактирования) новости

На странице создания (редактирования) новости должны присутствовать следующие элементы:

* Постер новости (основное изображение) и возможность загрузки нового изображения.
* Поле ввода заголовка новости.
* Многострочное поле для ввода содержимого новости в формате markdown.
* Элементы выбора категории и тегов.

В случае ошибок прохождения валидации во время публикации новости, под соответствующими полями должно выводиться сообщения об соответствующей ошибке, выделенной красным цветом, согласно *цветовой палитре*.

Страница авторских новостей репортёра

Страница авторских новостей аналогична по оформлению странице избранного, за исключением наличия в верхней её части (шапке) кнопки для создания новости (переход на страницу создания новости). При нажатии на новость, необходимо открывать её на соответствующей странице просмотра новостей.

Проектирование и разработка базы данных

Для упрощения просмотра и понимания взаимосвязей между данными и уменьшения количества ошибок при разработке было проведено проектирование базы данных с использованием инструмента визуального моделирования и проектирования draw.io.

Далее описан процесс создания схем базы данных для всех трёх моделей (концептуальная, логическая, физическая).

Концептуальная модель данных

Во время проектирования концептуальной модели были проведены исследования и собраны требования к базе данных, а также определено, какую информацию нужно хранить и как она будет связана.

Были выделены основные сущности, которые будут представлять информацию в базе данных. Для каждой сущности были определены соответствующие атрибуты.

В результате была составлена диаграмма концептуальной модели базы данных на основе определённых сущностей, атрибутов и связей *(см. Рисунок 35)*.

1. Сущность «Пользователь» с атрибутами «Логин», «Имя», «Фамилия», «Хеш пароля», «Адрес электронной почты», «Роль».
2. Сущность «Изображение» с атрибутами «дата загрузки», «хеш файла», «расширение».
3. Сущность «Токен».
4. Сущность «Тег».
5. Сущность «Категория» с атрибутами «Название», «Цвет», «ИД Изображения».
6. Сущность «Новость» с атрибутами «Заголовок», «Содержимое», «Автор», «Категория».
7. Сущность «Реакции» с атрибутами «Эмодзи», «Описание».
8. Сущность «История чтения» с атрибутами «Новость», «Пользователь», «Дата чтения».
9. Сущность «Избранное» с атрибутами «Новость», «Пользователь», «Дата добавления».
10. Сущность «Комментарий» с атрибутами «Автор», «Сообщение», «Дата добавления», «Дата обновления».
11. Сущность «Жалоба» с атрибутами «ИД Комментария», «Причина», «Пояснение», «Автор жалобы».
12. Сущность «Бан» с атрибутами «Пользователь», «Пояснение», «Дата окончания».
13. Сущность «Причина» (бана).
14. Связующая сущность «Тег-Новость».
15. Связующая сущность «Реакция-Новость» с дополнительным атрибутом «ИД пользователя».

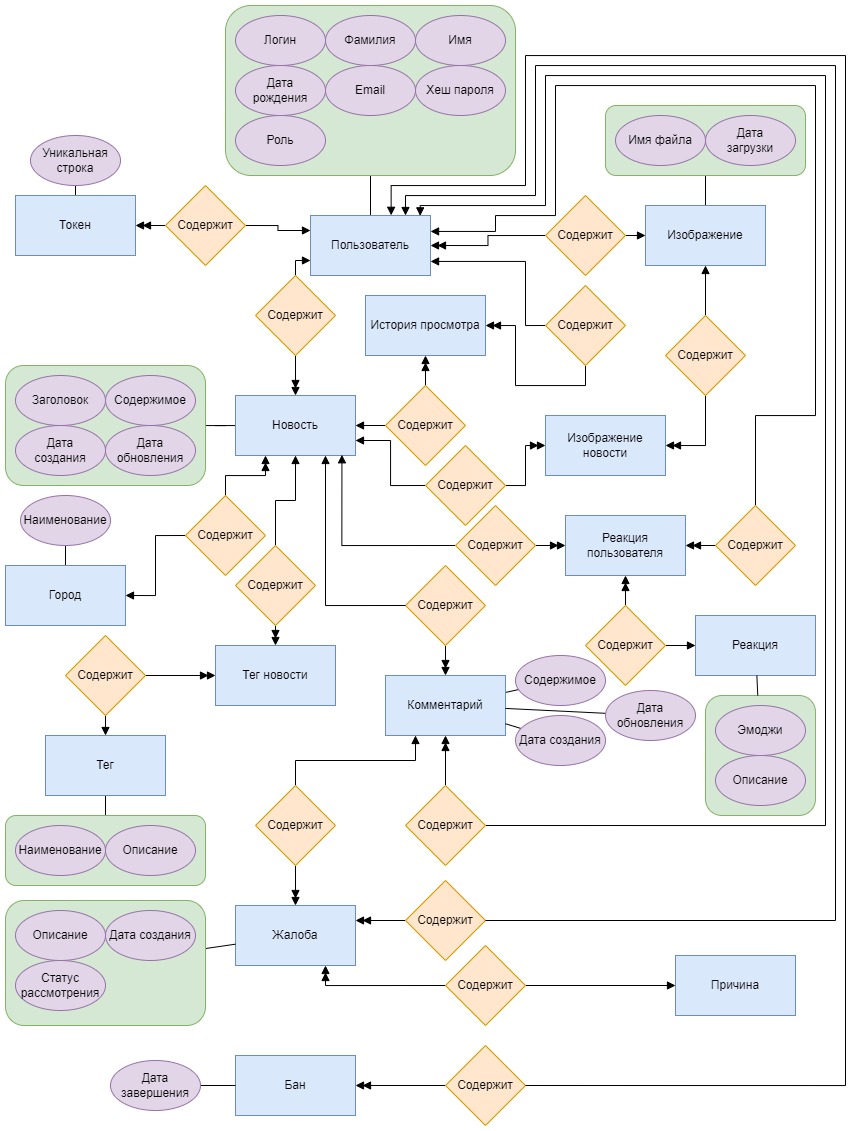


Рисунок 35 — Концептуальная модель данных

Логическая модель данных

На этапе проектирования логической модели была осуществлена трансформация концептуальной модели в вид, соответствующий выбранной реляционной модели данных, определенных таблиц. Для каждой таблицы были определены соответствующие атрибуты и их типы данных. Была проведена нормализация для устранения избыточности и аномалий данных. Таблицы были приведены к нормальным формам.

В конечном итоге была создана диаграмма баз данных, которая отображает структуру базы данных на логическом уровне *(см. Рисунок 36).*

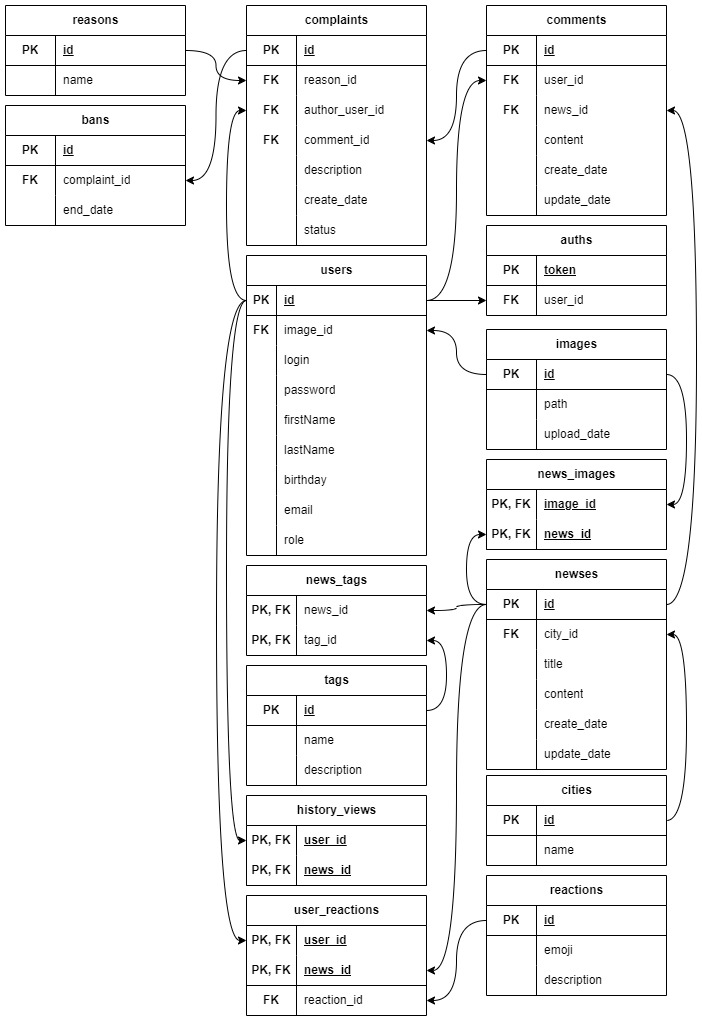


Рисунок 36 — Логическая модель данных

Физическая модель данных

На этапе проектирование физической модели была выбрана подходящая система управления базами данных для её реализации. Было решено как данные будут храниться на физическом уровне, определено предполагаемое количество данных и выделены соответствующие ресурсы.

После завершения проектирования физической модели была получена диаграмма базы данных, отображающая организацию данных на самом низком уровне *(см. Рисунок 37 — Рисунок 39).*

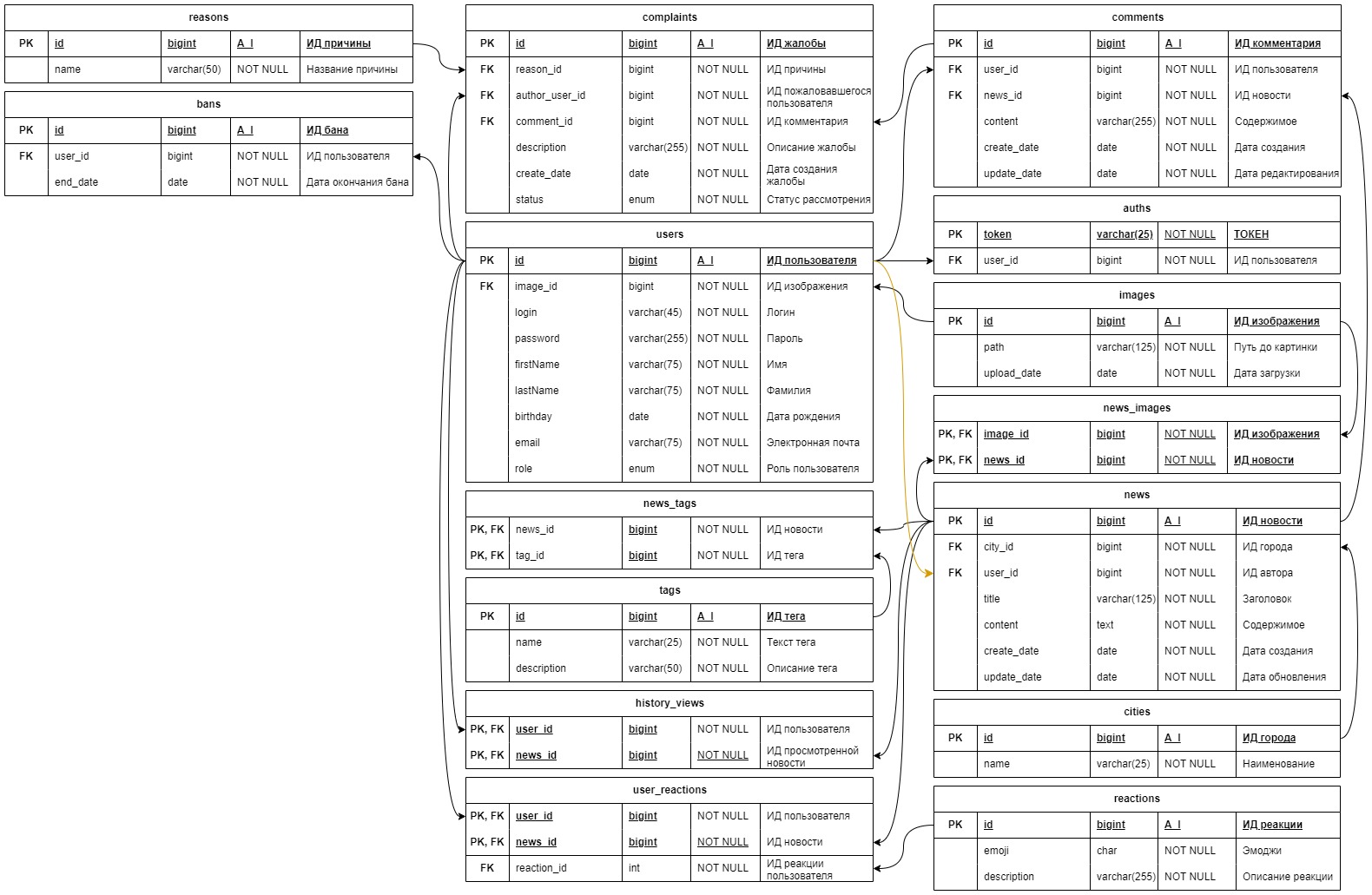


Рисунок 37 — Физическая модель данных, часть 1

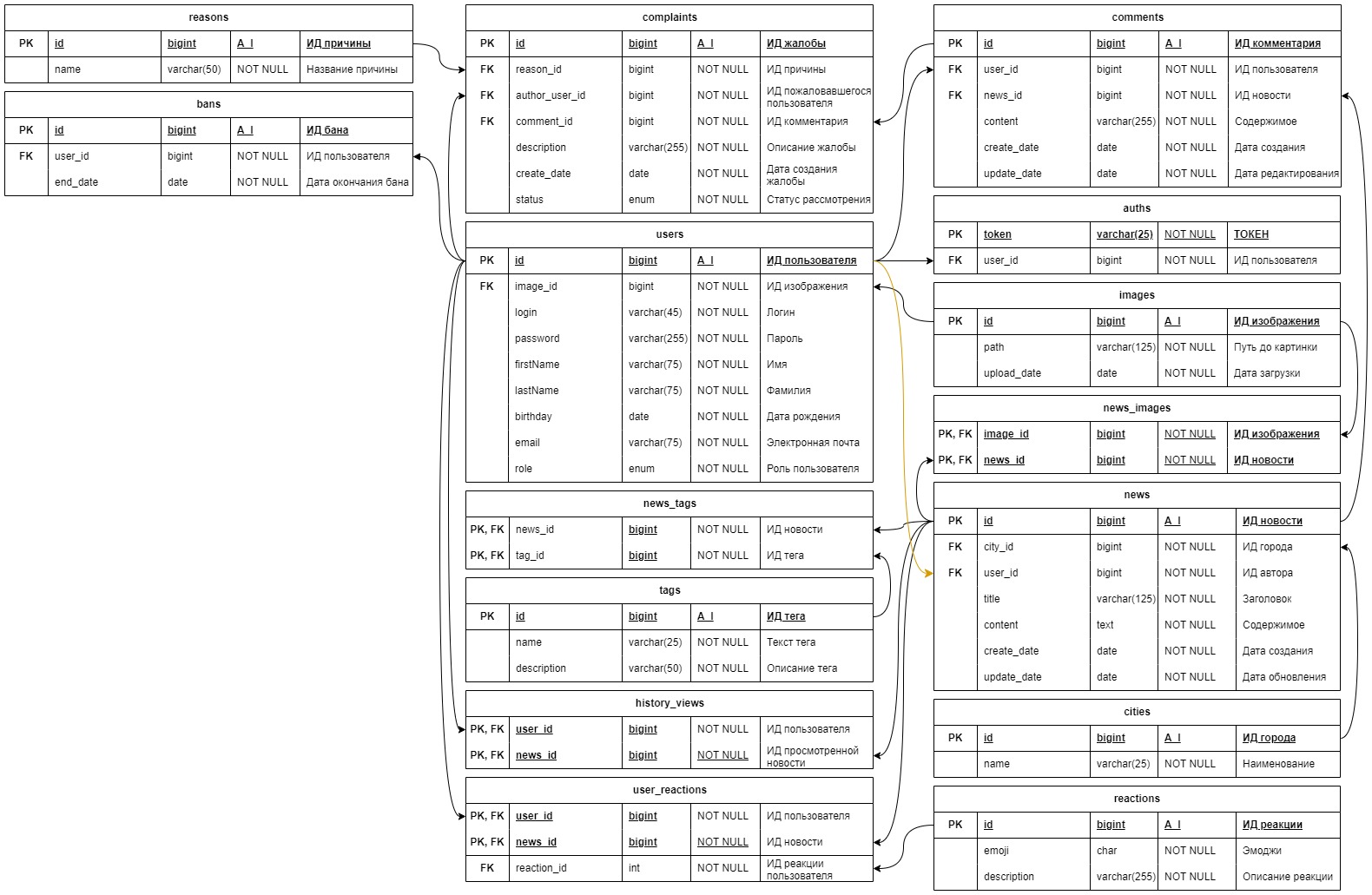


Рисунок 38 — Физическая модель данных, часть 2

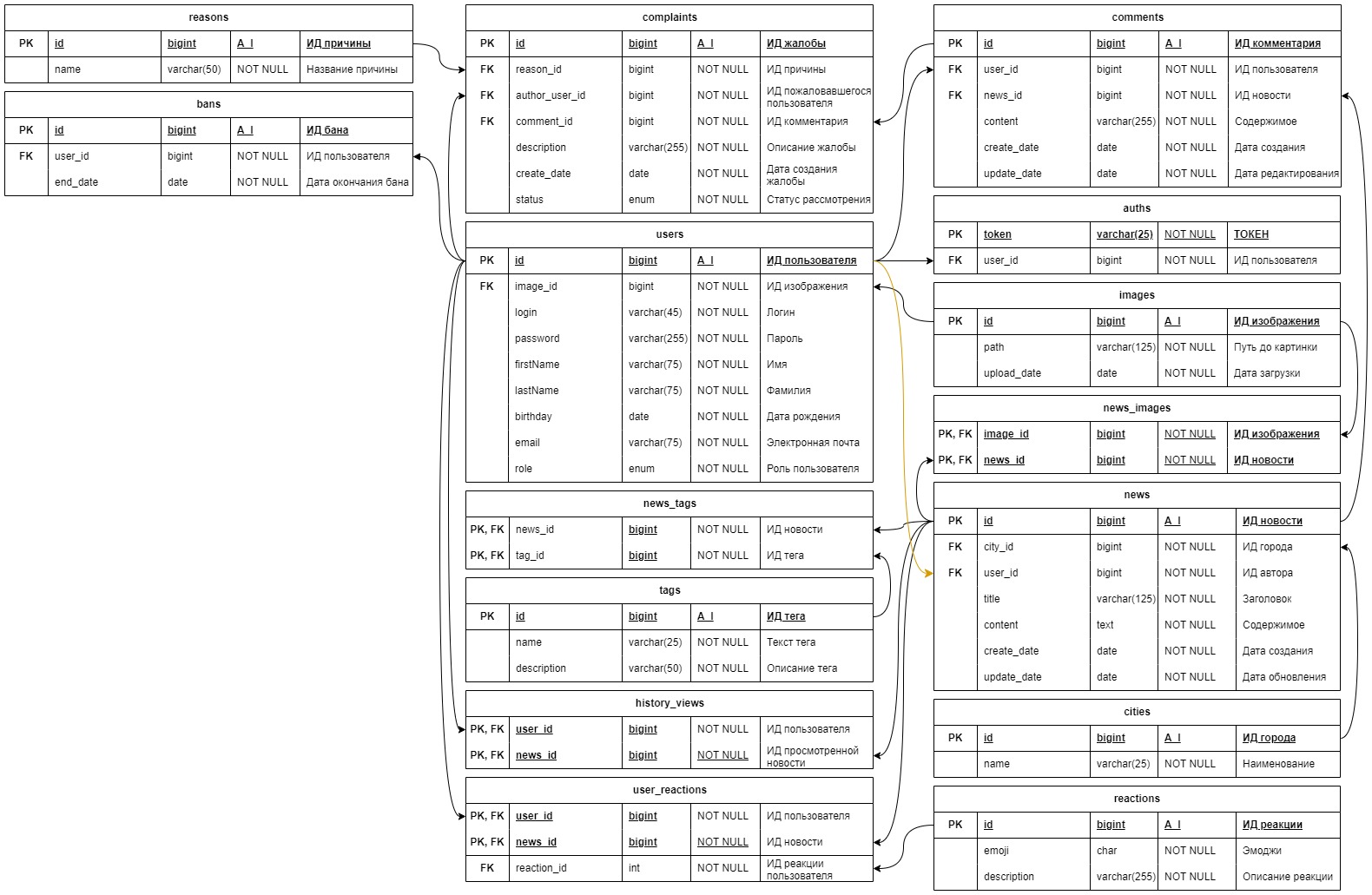


Рисунок 39 — Физическая модель данных, часть 3

Разработка серверной части

Для разработки api будущего сервиса выбран и взят фреймворк «Laravel». Перед началом написания кода были выделены основные этапы разработки:

1. Написание миграций.
2. Описание моделей основных сущностей базы данных.
3. Определение маршрутов (end-points) api.
4. Написание контроллеров, обрабатывающих входящие запросы для каждой сущности.
5. Тестирование.

Далее подробно описан каждый этап.

Миграции

Миграции являются неотъемлемой часть в информационных системах, работающих с базами данных. Они позволяют контролировать версии состояний вашей базы данных и, при необходимости, вносить изменения в существующую структуру базы данных без необходимости полностью пересоздавать таблицы в базе данных.

Каждая миграция описывается через использование фасада Schema, который позволяет нам создавать и управлять таблицами, хранящимися в нашей базе данных. Используя его, по выделенным ранее сущностями и составленной физической модели, были описаны миграции для ей генерации. Итоговые миграции расположены в директории *database/migrations* и представлены в *Листингах 1 — 14*.

Листинг 1 — Миграция базы данных database/migrations/0001\_01\_01\_000000\_create\_users\_table.php (таблица users)

|  |
| --- |
| Schema::create('users', function (Blueprint $table) {  $table->id();  $table->foreignId('image\_id')->nullable()->constrained()->cascadeOnUpdate();  $table->string('firstName');  $table->string('lastName');  $table->string('login')->unique();  $table->string('password');  $table->string('email')->unique();  $table->enum('role', ['reader', 'reporter', 'admin'])->default('reader');  $table->timestamps(); }); |

Листинг 2 — Миграция базы данных 0000\_00\_00\_000000\_create\_images\_table.php (таблица images)

|  |
| --- |
| Schema::create('images', function (Blueprint $table) {  $table->id();  $table->string('hash', 256);  $table->enum('extension', config('settings.allowed\_upload\_mimes'));  $table->date('upload\_date'); }); |

Листинг 3 — Миграция базы данных 2024\_11\_12\_062411\_create\_categories\_table.php (таблица categories)

|  |
| --- |
| Schema::create('categories', function (Blueprint $table) {  $table->id();  $table->string('name', 30);  $table->string('background\_color', 6);  $table->foreignId('image\_id')->nullable()->constrained()->cascadeOnUpdate();  $table->timestamps(); }); |

Листинг 4 — Миграция базы данных 2024\_11\_12\_062412\_create\_news\_table.php (таблица news)

|  |
| --- |
| Schema::create('news', function (Blueprint $table) {  $table->id();  $table->foreignId('city\_id')->nullable()->constrained()->cascadeOnUpdate()->nullOnDelete();  $table->foreignId('user\_id')->nullable()->constrained()->cascadeOnUpdate()->nullOnDelete();  $table->foreignId('image\_id')->nullable()->constrained()->cascadeOnUpdate()->nullOnDelete();  $table->foreignId('category\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->string('title', 125);  $table->text('content');  $table->date('create\_date')->default(now());  $table->date('update\_date')->nullable();  $table->timestamps(); }); |

Листинг 5 — Миграция базы данных 2024\_11\_12\_062457\_create\_comments\_table.php (таблица comments)

|  |
| --- |
| Schema::create('comments', function (Blueprint $table) {  $table->id();  $table->foreignId('user\_id')->nullable()->constrained()->cascadeOnUpdate()->nullOnDelete();  $table->foreignId('news\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->text('content');  $table->timestamps(); }); |

Листинг 6 — Миграция базы данных 2024\_11\_12\_062522\_create\_reactions\_table.php (таблица reactions)

|  |
| --- |
| Schema::create('reactions', function (Blueprint $table) {  $table->id();  $table->char('emoji');  $table->string('description');  $table->timestamps(); }); |

Листинг 7 — Миграция базы данных 2024\_11\_12\_062539\_create\_user\_reactions\_table.php (таблица user\_reactions)

|  |
| --- |
| Schema::create('user\_reactions', function (Blueprint $table) {  $table->foreignId('user\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->foreignId('news\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->foreignId('reaction\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->primary(['user\_id', 'news\_id']);  $table->timestamps(); }); |

Листинг 8 — Миграция базы данных 2024\_11\_12\_062638\_create\_history\_views\_table.php (таблица history\_views)

|  |
| --- |
| Schema::create('history\_views', function (Blueprint $table) {  $table->foreignId('user\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->foreignId('news\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->date('read\_date');  $table->time('read\_time');  $table->primary(['user\_id', 'news\_id', 'read\_date']); }); |

Листинг 9 — Миграция базы данных 2024\_11\_12\_062657\_create\_tags\_table.php (таблица tags)

|  |
| --- |
| Schema::create('tags', function (Blueprint $table) {  $table->id();  $table->string('name', 25);  $table->string('description', 50)->nullable();  $table->timestamps(); }); |

Листинг 10 — Миграция базы данных 2024\_11\_12\_062706\_create\_news\_tags\_table.php (таблица news\_tags)

|  |
| --- |
| Schema::create('news\_tags', function (Blueprint $table) {  $table->foreignId('news\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->foreignId('tag\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->primary(['news\_id','tag\_id']);  $table->timestamps(); }); |

Листинг 11 — Миграция базы данных 2024\_11\_12\_062750\_create\_reasons\_table.php (таблица reasons)

|  |
| --- |
| Schema::create('reasons', function (Blueprint $table) {  $table->id();  $table->string('name', 50);  $table->timestamps(); }); |

Листинг 12 — Миграция базы данных 2024\_11\_12\_062754\_create\_complaints\_table.php (таблица complaints)

|  |
| --- |
| Schema::create('complaints', function (Blueprint $table) {  $table->id();  $table->foreignId('reason\_id')->constrained()->cascadeOnUpdate();  $table->foreignId('author\_user\_id')->nullable()->constrained('users')->cascadeOnUpdate()->nullOnDelete();  $table->foreignId('comment\_id')->nullable()->constrained()->cascadeOnUpdate()->nullOnDelete();  $table->string('description')->nullable();  $table->enum('status', ['pending', 'accepted', 'rejected'])->default('pending');  $table->timestamps(); }); |

Листинг 13 — Миграция базы данных 2024\_11\_14\_062259\_create\_bans\_table.php (таблица bans)

|  |
| --- |
| Schema::create('bans', function (Blueprint $table) {  $table->id();  $table->foreignId('complaint\_id')->unique()->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->date('end\_date');  $table->timestamps(); }); |

Листинг 14 — Миграция базы данных 2024\_11\_12\_062706\_create\_news\_tags\_table.php (таблица favourites)

|  |
| --- |
| Schema::create('favourites', function (Blueprint $table) {  $table->foreignId('user\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->foreignId('news\_id')->constrained()->cascadeOnUpdate()->cascadeOnDelete();  $table->dateTime('added\_date');  $table->primary(['user\_id', 'news\_id']); }); |

Модели

Для взаимодействия с таблицами базы данных необходимы модели. Они позволяют через Eloquent ORM взаимодействовать с соответствующими таблицами, определять связи между данными и управлять бизнес-логикой. Каждая модель наследует класс *Model* и связана с определённой таблицей, а также содержит свойства и методы, который значительно упрощают работу с данными.

Модель «User»

Данная модель является представлением таблицы «users» и описывает поля присущие каждой записи, находящейся в ней. Она располагается в файле *app/Models/User.php*. Через неё происходит управление пользователями и реализация интерфейсов аутентификации, авторизации и сброса пароля. В ней содержатся поля login, password, firstName, lastName, email, image\_id, а также скрытые поля password и image\_id.

Для описания полей, содержащихся в модели, используется атрибут *fillable* с уровнем доступа *protected*, которых хранит массив из строк-имён полей соответствующей таблице модели *(см. Листинг 15)*.

Листинг 15 — Описание атрибутов (полей таблицы) модели через поле fillable для модели «User»

|  |
| --- |
| protected $fillable = [  'firstName',  'lastName',  'login',  'password',  'email',  'role',  'image\_id' ]; |

Так как в модели содержатся поля, которые не должны быть включены в массив и JSON-представление модели, нужно определить чёрный список полей. Сделать это можно через атрибут *hidden* с уровнем доступа *protected*, который также, как и атрибут *fillable,* содержит массив строк-имён поле, которые должны быть исключены из представления *(см. Листинг 16)*.

Листинг 16 — Описание исключённых из представления атрибутов (полей таблицы) модели через поле hidden для модели «User»

|  |
| --- |
| protected $hidden = [  'password',  'image\_id' ]; |

После определения публичных и скрытых полей, необходимо настроить связи между моделями, т.е. то, где и каким образом они хранят ссылки (связи) друг с другом.

Для модели «User» необходимы связи с таблицами (моделями) Bans *(модель Ban)* и Images *(Модель Image)*. Одни из основных видов связи между таблицами – *один-ко-многим* и *много-через-одного*. В нашем случае обе модели связаны с моделью «User» через *один-ко-многим*. Пример реализации такой связи смотри в *Листинге 17*.

Листинг 17 — Связь модели Image с полем «image\_id», хранящимся в модели «User»

|  |
| --- |
| public function avatar(): BelongsTo {  return $this->belongsTo(Image::class, 'image\_id'); } |

Также определим вспомогательные методы, к которым мы всегда сможем обратиться в любой момент, имея доступ к экземпляру класса.

В данном случае определим методы для проверки, является ли пользователь администратором (метод *«isAdministrator»*), а также методы *«casts»* (для определения поведения атрибутов модели) и *«history\_views»*.

Итоговый файл модели смотри в *Листинге 18*.

Листинг 18 — Итоговое содержимое файла модели «User», User.php

|  |
| --- |
| class User extends *Authenticatable* {  use HasApiTokens, HasFactory, Notifiable;   protected $fillable = [  'firstName',  'lastName',  'login',  'password',  'email',  'role',  'image\_id'  ];   protected $hidden = [  'password',  'image\_id'  ];   protected function casts(): array  {  return [  'password' => 'hashed',  ];  }   public function isAdministrator(): bool  {  return $this->role == 'admin';  }   public function avatar(): BelongsTo  {  return $this->belongsTo(Image::class, 'image\_id');  }   public function history\_views(int $count = 20, int $offset = 0): Collection  {  return News::*whereHas*('reactions', function ($query) {  $query->where('user\_id', '=', $this->id)->get();  });}   public function bans()  {return Ban::*join*('complaints', 'bans.complaint\_id', '=', 'complaints.id')  ->join('comments', 'complaints.comment\_id', '=', 'comments.id')  ->where('comments.user\_id', '=', $this->id)->get();  } } |

Модель «Tag»

Подобно предыдущей модели, мы сформируем представление таблицы тегов. Поскольку эта таблица не содержит связей с другими моделями (таблицами), мы определим только общедоступные поля *(name, description)* и скрытые поля *(created\_at, updated\_at)*, описав их аналогично модели «User» *(см. Листинги 15–16)*.

Итоговое содержание файла модели представлено в *Листинге 19*.

Листинг 19 — Итоговое содержимое файла модели «Tag», Tag.php

|  |
| --- |
| class Tag extends Model {  protected $fillable = [  'name',  'description'  ];   protected $hidden = [  'created\_at',  'updated\_at'  ]; } |

Модель «Reason»

Реализация данной модели аналогична модели «Tag», но у неё нет скрытых полей и есть только 1 публичное поле – «name». Она представляет собой простую коллекцию причин отправки жалоб и, поэтому, не имеет связей с другими моделями внутри себя, т.к. ожидается, что относительно неё не будет необходимости получать отношения с другими таблицами.

В результате должен получиться файл модели, аналогично представленному в *Листинге 20*.

Листинг 20 — Итоговое содержимое файла модели «Reason», Reason.php

|  |
| --- |
| class Reason extends Model { protected $fillable = ['name']; } |

Модель «Reaction»

Эта модель хранит виды (или типы) реакций, который пользователь может установить для новости, т.е. *отреагировать*. Она описывается аналогично предыдущим моделям и дополнительно имеет методы «setReaction» и «deleteReaction» которые в качестве аргумента принимают новость и пользователь, от чьего имени нужно установить реакцию.

Итоговое содержимое модели представлено в *Листинге 21*.

Листинг 21 — Итоговое содержимое файла модели «Reaction», Reaction.php (описание параметров опущено)

|  |
| --- |
| class Reaction extends Model {  protected $fillable = [  'emoji',  'description'  ];  public function setReaction(User $user, News $news)  {  return UserReaction::*updateOrCreate*([  'user\_id' => $user->id,  'news\_id' => $news->id  ], [  'user\_id' => $user->id,  'news\_id' => $news->id,  'reaction\_id' => $this->id  ]);  }public static function deleteReaction(User $user, News $news)  {  return UserReaction::*where*([  'user\_id' => $user->id,  'news\_id' => $news->id  ])->delete();  } } |

Модель «Image»

Хранит информацию о успешно загруженных на сервер изображениях.

Через установленное значение *«false»* у переменной *«timestamps»*, мы сообщаем, что не хотим использовать временные метки создания и обновления для записей.

Также, в этой модели реализованы методы *«getPathByDate»* (для получения пути картинки на основе даты), *«getPathUrl»* (для получения адреса изображения относительно url сервера) *«savePhoto»* (для сохранения загружаемого сообщения в базе данных и на диске) и *«saveLocalPhoto»* (используется для миграций, загружая локальные изображения в базу).

Итоговое содержимое модели представлено в *Листинге 22*.

Листинг 22 — Итоговое содержимое файла модели «Image», Image.php (описание параметров опущено)

|  |
| --- |
| class Image extends Model {  public $timestamps = false;   protected $fillable = [  'hash',  'extension',  'upload\_date'  ]; public static function getPathByDate(DateTime $date = new DateTime()): string  {  *// Извлекаем конкретные числа* $day = $date->format('d');  $month = $date->format('m');  $year = $date->format('Y');   return "images/$year/$month/$day";  }  public static function getPathUrl(string|Image $value, ImgType $t = ImgType::*Uploaded*): string  {  $image = $value instanceof Image  ? $value  : Image::*where*(['hash' => $value])->first();   if ($t === ImgType::*Uploaded*)  $path = Image::*getPathByDate*(new DateTime($image->upload\_date));  else  $path = "images/categories";   return 'storage/' . $path . "/$image->hash.$image->extension";  }  public static function savePhoto($parameter, $customName = null)  {  $pathToSave = Image::*getPathByDate*();   *// Обрабатываем загрузку файла* $file = request()->file("$parameter");  $filename = $file->getClientOriginalName();  try {  *// Проверяем, разрешён ли загружаемый тип файла (картинки)* $validator = Validator::*make*(request()->all(), [  $parameter => 'mimes:' . implode(',', config('settings.allowed\_upload\_mimes'))  ]);  if ($validator->fails()) {  return [  'name' => $filename,  'message' => 'Validation failed',  'errors' => $validator->errors()->toArray()['file']  ];  }   *// Проверяем, не слишком ли большое изображение* $filesize = $file->getSize();  if (config('settings.max\_file\_size') < $filesize) {  return [  'name' => $filename,  'message' => 'File is too large'  ];  }   *// Подготавливаем уникальность изображения* $tmpPath = $file->getRealPath();  $extension = $file->getClientOriginalExtension();  $hash = hash\_file('xxh3', $tmpPath);   *// Сохраняем изображение* $fileNameSave = "$hash.$extension";   $file->storeAs($pathToSave, $fileNameSave, ['disk' => 'public']);   $imageExists = Image::*where*('hash', '=', $hash)->first();  if (!$imageExists)  $imageExists = Image::*create*([  'hash' => $hash,  'extension' => $extension,  'upload\_date' => date('Y-m-d')  ]);   return $imageExists;  } catch (Exception) {  return [  'name' => $filename,  'message' => "Something went wrong"  ];  }  }  public static function saveLocalPhoto(string $path, string $savePrefix = ""): Image  {  *// Определяем путь до файла* if (!file\_exists($path))  $tmpPath = $\_SERVER['DOCUMENT\_ROOT'] . ($path[0] === '/' ? '' : '/') . $path;  else  $tmpPath = $path;   *// Подготавливаем уникальность изображения* $extension = pathinfo($tmpPath, *PATHINFO\_EXTENSION*);  $hash = hash\_file('xxh3', $tmpPath);   *// Сохраняем изображение* $fileNameSave = ($savePrefix === "" ? "" : $savePrefix . "/") . "$hash.$extension";   Storage::disk('public')->put($fileNameSave, file\_get\_contents($tmpPath));    $imageExists = Image::*where*('hash', '=', $hash)->first();  if (!$imageExists)  $imageExists = Image::*create*([  'hash' => $hash,  'extension' => $extension,  'upload\_date' => date('Y-m-d')  ]);   return $imageExists;  } } |

Модель «HistoryView»

Отвечает за хранение истории чтения пользователя. Содержит обратную связь с новостью, к которой хранит историю просмотра (обратная связь с пользователем опущена).

Итоговое содержимое модели представлено в *Листинге 23*.

Листинг 23 — Итоговое содержимое файла модели «HistoryView», HistoryView.php

|  |
| --- |
| class HistoryView extends Model {  public $timestamps = false;    protected $fillable = ['user\_id', 'news\_id', 'read\_date', 'read\_time'];   protected $hidden = [];   public function news() { return $this->belongsTo(<News::class>); } } |

Модель «Favourite»

Аналогичная модели «HistoryView», но имеет другое назначение – хранение избранных новостей пользователя, т.е. тех, которые он захотел отложить (пометить, оставить закладку).

Итоговое содержимое модели представлено в *Листинге 24*.

Листинг 24 — Итоговое содержимое файла модели «Favourite», Favourite.php

|  |
| --- |
| class Favourite extends Model {  public $timestamps = false;   protected $fillable = ['news\_id', 'user\_id', 'added\_date'];   public function news() { return $this->belongsTo(<News::class>); } } |

Модель «Category»

Хранит информацию о категория новостей. Содержит название категории, основной цвет её карточки и изображение (иконку).

Итоговое содержимое модели представлено в *Листинге 25*.

Листинг 25 — Итоговое содержимое файла модели «Category», Category.php

|  |
| --- |
| class Category extends Model {  protected $fillable = ['name', 'background\_color', 'image\_id'];   protected $hidden = ['created\_at', 'updated\_at', 'image\_id'];   public function image() { return $this->belongsTo(Image::class); } } |

Модель «UserReaction»

Отвечает за хранение реакций пользователей на новости. Функционально почти никак не участвует, поэтому не содержит никаких описанных обратных связей.

Итоговое содержимое модели представлено в *Листинге 26*.

Листинг 26 — Итоговое содержимое файла модели «UserReaction», UserReaction.php

|  |
| --- |
| class UserReaction extends Model { protected $fillable = ['news\_id', 'user\_id', 'reaction\_id']; } |

Модель «Ban»

Аналогична предыдущей, но содержит блокировки пользователей и их сроки. Содержит причину блокировки и дату окончания.

Итоговое содержимое модели представлено в *Листинге 27*.

Листинг 27 — Итоговое содержимое файла модели «Ban», Ban.php

|  |
| --- |
| class Ban extends Model {  protected $fillable = [  'complaint\_id',  'end\_date'  ];   protected $hidden = [];   public function complaint(): BelongsTo  {  return $this->belongsTo(Complaint::class);  } } |

Модель «News»

Хранит информацию о написанных новостях. Описывает заголовок, автора, изображение (постер), содержимое, категорию, дату создания и обновления.

Также, в этой модели реализованы методы «isBookmarked» (проверяет, есть ли новость в закладках), «commentsCount» (возвращает количество комментариев), «viewsCount» (возвращает количество просмотров).

Итоговое содержимое модели представлено в *Листинге 28*.

Листинг 28 — Итоговое содержимое файла модели «News», News.php

|  |
| --- |
| class News extends Model {  protected $fillable = [ 'title', 'user\_id', 'city\_id', 'image\_id', 'content', 'category\_id', 'create\_date', 'update\_date' ];   protected $hidden = [];  protected static function newFactory(): NewsFactory { return NewsFactory::*new*(); }   public function city(): BelongsTo { return $this->belongsTo(City::class); }  public function image() { return $this->belongsTo(Image::class, 'image\_id');}   public function images(): BelongsToMany  {  return $this->belongsToMany(Image::class, NewsImage::class);  }   public function author(): BelongsTo  {  return $this->belongsTo(User::class, 'user\_id');  }   public function tags(): BelongsToMany  {  return $this->belongsToMany(Tag::class, NewsTag::class);  }   public function reactions(): BelongsToMany  {  return $this->belongsToMany(Reaction::class, UserReaction::class);  }   public function comments(): HasMany  {  return $this->hasMany(Comment::class);  }   public function category()  {  return $this->belongsTo(Category::class);  }   public function isBookmarked()  {  $user = Auth::user();   if (!$user)  return false;   return Favourite::*where*([  ["news\_id", "=", $this->id],  ["user\_id", "=", $user->id]  ])->first() != null;  }   public function commentsCount() { return Comment::*where*("news\_id", $this->id)->count(); }  public function viewsCount()  {  return HistoryView::*where*("news\_id", $this->id)->count();  } } |

Модель «NewsTag»

Эта модель связывает новость с тегами. Хранит идентификаторы тега и новости.

Итоговое содержимое модели представлено в *Листинге 29*.

Листинг 29 — Итоговое содержимое файла модели «NewsTag», NewsTag.php

|  |
| --- |
| class NewsTag extends Model {  protected $fillable = [  'news\_id',  'tag\_id'  ];   protected $hidden = []; } |

Маршрутизация

Для сопоставления HTTP-запросов с соответствующими обработчиками, необходимо определить маршруты API в файле *routes/api.php*. Маршруты, определённые в нём, вложены в группу маршрутов *RouteServiceProvider*, внутри которой применяется префикс *«api»*, поэтому нет необходимости вручную добавлять его к каждому маршруту.

Исходя из технического задания, в файле *«routes/api.php»* были описаны маршруты (см. Листинг 30).

Листинг 30 — Маршрутизация API «routes/api.php»

|  |
| --- |
| */\*  \* [Авторизация]  \* /login | post  \* /register | post  \* /logout | get  \* /logout/all | get  \*/* Route ::controller(AuthController::class) ->group(function () {  Route::post('login', 'login' );  Route::post('register', 'register');  Route::prefix('logout')  ->middleware(OptionalAuth::class)  ->group(function () {  Route::get('', 'logout' );  Route::get('all', 'logoutAll');  }); });  */\*  \* [Пользователи]  \* /users | post  \* /users/{user} | get, put, delete  \*/* Route ::controller(UserController::class) ->prefix('users') ->group(function() {  *// Создание пользователя* Route::post('', 'store');  *// Работа с собой* Route::get('me', 'me');  *// Работа с конкретным пользователем* Route::group(['prefix' => '{user}'], function () {  Route::get ('', 'show' );  Route::post ('update', 'update' );  Route::delete('', 'destroy');  }); });  */\*  \* [Тэги]  \* /tags | get, post  \* /tags/{tag} | get, delete  \*/* Route ::controller(TagController::class) ->prefix('tags') ->group(function () {  Route::get ('', 'index');  Route::post('', 'store');  Route::group(['prefix' => '{tag}'], function () {  Route::get ('', 'show' );  Route::delete('', 'destroy');  }); });  */\*  \* [Города]  \* /cities | get, post  \* /cities/{city} | get, put, delete  \*/* Route ::controller(CityController::class) ->prefix('cities') ->group(function () {  Route::get ('', 'index');  Route::post('', 'store');  Route::group(['prefix' => '{city}'], function () {  Route::get ('', 'show' );  Route::put ('', 'update' );  Route::delete('', 'destroy');  }); });  */\*  \* [Реакции]  \* /reactions | get, post  \* /reactions/{reaction} | get, put, delete  \*/* Route ::controller(ReactionController::class) ->group(function () {  Route::group(['prefix' => 'reactions'], function () {  Route::get ('', 'index');  Route::post('', 'store');  Route::group(['prefix' => '{reaction}'], function () {  Route::get ('', 'show' );  Route::put ('', 'update' );  Route::delete('', 'destroy');  });  });  Route::group([  'prefix' => '/news/{news}/reactions',  'controller' => ReactionController::class  ], function () {  Route::post ('{reaction}', 'userReactionStore' );  Route::delete('', 'userReactionDestroy');  }); });  */\*  \* [Новости]  \* /news | get, post  \* /news/{news} | get, put, delete  \*/* Route ::controller(NewsController::class) ->prefix('news') ->group(function () {  Route::get ('', 'index');  Route::post('', 'store');  Route::group(['prefix' => '{news}'], function () {  Route::get ('', 'show' );  Route::post ('', 'update' );  Route::delete('', 'destroy');  }); });  */\*  \* [История]  \* /history | get  \*/* Route ::controller(HistoryViewController::class) ->prefix('history') ->group(function () {  Route::get ('', 'index'); });  */\*  \* [Избранное]  \* /favourites | get  \*/* Route ::controller(FavouriteController::class) ->prefix('favourites') ->group(function () {  Route::get ('', 'index');  Route::post('', 'store');  Route::delete('{news}', 'destroy'); });  */\*  \* [Комментарии]  \* /news/{news}/comments | get, post  \* /news/{news}/comments/{comment} | put, delete  \*/* Route ::prefix('news') ->group(function () {  Route::group([  'prefix' => '{news}/comments',  'controller' => CommentController::class  ], function () {  Route::get ('', 'index');  Route::post ('', 'store');  Route::group(['prefix' => '{comment}'], function () {  Route::put ('', 'update' );  Route::delete('', 'destroy');  });  }); });  */\*  \* [Жалобы]  \* /news/{news}/comments/{comment}/complaints | get, post, put  \* /news/{news}/comments/{comment}/complaints/{complaint} | get  \*/* Route ::prefix('news') ->controller(ComplaintController::class) ->group(function () {  Route::group(['prefix' => '{news}/comments/{comment}/complaints'], function () {  Route::get ('', 'index' );  Route::post('', 'store' );  Route::put ('{complaint}', 'updateStatus');  Route::get ('{complaint}', 'show' );  }); });  */\*  \* [Баны]  \* /users/{user}/bans | get, post  \* /users/{user}/bans/{ban} | get, put, delete  \*/* Route ::prefix('users/{user}/bans') ->controller(BanController::class) ->group(function () {  Route::get ('', 'index' );  Route::get ('{ban}', 'show' ); });  */\*  \* [Изображения]  \* /images/{image] | get  \*/* Route ::prefix('images') ->controller(ImageController::class) ->group(function () {  Route::post('', 'upload'); });  */\*  \* [Категории]  \* /categories | get, post  \* /categories/{category} | delete  \*/* Route ::prefix('categories') ->controller(CategoryController::class) ->group(function () {  Route::get('', 'index');  Route::post('', 'store');  Route::delete('{category}', 'destroy'); }); |

Контроллеры

Вместо того чтобы определять логику обработки запросов в виде замыканий в файлах маршрутов, мы организуем её с помощью классов контроллеров. Контроллеры позволяют сгруппировать связанную логику обработки запросов в одном классе.

В Laravel контроллеры 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 содержать методы для обработки различных HTTP-запросов, таких как GET, POST, PUT, DELETE.

В *Листинге 31* представлен контроллер авторизации (AuthController), который расположен в app/Http/Controllers/Routes/AuthController.php. Этот контроллер реализовывает следующие методы:

1. login() – аутентифицирует пользователя, проверяя переданные им логин и пароль. В случае успеха возвращает токен авторизации и данные о пользователе. Если же аккаунты, в который пытается войти пользователь, заблокирован, то выбрасывается соответствующая ошибка.
2. register() – регистрирует пользователя по переданным данным с присваиванием ему роли «reader» (читатель), если не указана иная («reporter»). После успешной регистрации возвращается токен и данные пользователя.
3. logout() – удаляет авторизацию (токен) текущего пользователя, завершив тем самым его сессию.
4. logoutAll() – удаляет все токены текущего пользователя, завершив все сессии его аккаунта.

Листинг 31 — Контроллер авторизации «AuthController.php»

|  |
| --- |
| class AuthController extends Controller {  public function login(Request $request)  {  if (!Auth::attempt($request->only('login', 'password')))  throw new ApiException('Invalid credentials', 401);   $user = Auth::user();   $currentDate = (new DateTime())->format('Y-m-d');   $ban = User\Ban::*join*('complaints', 'complaints.id', '=', 'bans.complaint\_id')  ->join('comments', 'comments.id', '=', 'complaints.comment\_id')  ->where('end\_date', '>', $currentDate)  ->where('user\_id', '=', $user->id)  ->first();   if ($ban)  throw new ApiException('Your account is blocked', 403);   $token = $user->createToken('token')->plainTextToken;   return response([  'success' => true,  'token' => $token,  'user' => FullUserResource::*make*($user)  ]);  }   public function register(UserRegistrationRequest $request)  {  $user = User::*create*($request->validated());  $token = $user->createToken('token')->plainTextToken;   return response([  'success' => true,  'token' => $token,  'user' => $user  ], 201);  }   public function logout()  {  auth()->user()->currentAccessToken()->delete();  return response(null, 204);  }   public function logoutAll()  {  foreach (auth()->user()->tokens as $token)  $token->delete();  return response(null, 204);  } } |

В *Листинге 32* представлен контроллер блокировок (BanController), который расположен в app/Http/Controllers/Routes/BanController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает все блокировки, выданные пользователю.
2. show() – показывает подробную информацию об конкретной блокировке.

Листинг 32 — Контроллер блокировок «BanController.php»

|  |
| --- |
| class BanController extends Controller {  public function index(User $user): JsonResponse  {  return response()->json(BanMinResource::*collection*($user->bans()));  }   public function show(User $user, Ban $ban): JsonResponse  {  return response()->json(BanResource::*make*($ban));  } } |

В *Листинге 33* представлен контроллер категорий (CategoryController), который расположен в app/Http/Controllers/Routes/CategoryController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает все категории новостей.
2. store() – метод для создания новой категории новостей.
3. destroy() – метод для удаления категории новости.

Листинг 33 — Контроллер категорий «CategoryController.php»

|  |
| --- |
| class CategoryController extends Controller {  public function index(): JsonResponse  {  return response()->json(CategoryResource::*collection*(Category::*all*()));  }   public function store(CategoryRequest $request): JsonResponse  {  $data = $request->validated();  if (isset($data['image']))  {  $image = Image::*savePhoto*('image', "images/categories");  if (!isset($image['message']))  $data['image\_id'] = $image->id;  }   $category = Category::*create*($data);   return response()->json($category, 201);  }   public function destroy(Category $category): JsonResponse  {  $category->delete();  return response()->json(null, 204);  } } |

В *Листинге 34* представлен контроллер категорий (CommentController), который расположен в app/Http/Controllers/Routes/CommentController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает все комментарии конкретной новости.
2. store() – метод для создания комментария к новости.
3. update() – метод обновления конкретного комментария (его изменения)
4. destroy() – метод для удаления конкретного комментария.

Листинг 34 — Контроллер комментариев «CommentController.php»

|  |
| --- |
| class CommentController extends Controller {  public function index(News $news): JsonResponse  {  $paginateResponse = $news  ->comments()  ->orderBy("comments.id", "DESC")  ->simplePaginate();  return response()->json(CommentResource::*collection*($paginateResponse));  }   public function store(CommentCreateRequest $request, News $news): JsonResponse  {  $comment = Comment::*create*([  ...$request->validated(),  'user\_id' => auth()->id(),  'news\_id' => $news->id  ]);  return response()->json(CommentResource::*make*($comment), 201);  }   public function update(CommentCreateRequest $request, News $news, Comment $comment): JsonResponse  {  $comment->update($request->validated());  return response()->json([  'code' => 200,  'data' => $comment  ]);  }   public function destroy(News $news, Comment $comment): JsonResponse  {  $comment->delete();  return response()->json(null, 204);  } } |

В *Листинге 35* представлен контроллер жалоб (ComplaintController), который расположен в app/Http/Controllers/Routes/ComplaintController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает все жалобы новости для комментария.
2. store() – метод для создания жалобы на комментарий новости.
3. updateStatus() – метод обновления статуса жалобы к комментарию.
4. show() – метод для просмотра конкретной жалобы.

Листинг 35 — Контроллер жалоб «ComplaintController.php»

|  |
| --- |
| class ComplaintController extends Controller {  public function index(News $news, Comment $comment): JsonResponse  {  return response()->json(ComplaintResource::*collection*($comment->complaints));  }   public function store(ComplaintRequest $request, News $news, Comment $comment): JsonResponse  {  if ($comment->checkComplaintExists(auth()->user()))  throw new ApiException('Complaint exists', 400);   $complaint = Complaint::*create*([  ...$request->validated(),  'author\_user\_id' => auth()->user()->id,  'comment\_id' => $comment->id  ]);   return response()->json(ComplaintResource::*make*($complaint));  }   public function updateStatus(ComplaintUpdateStatusRequest $request, News $news, Comment $comment, Complaint $complaint): JsonResponse  {  if ($request->get('status') === 'accepted')  {  $res = Validator::make(  request()->all(),  ['ban\_end\_date' => 'required|date|date\_format:Y-m-d|after:today']  );  if ($res->fails())  throw new ApiException('Request validation error',422, $res->errors());  Ban::*updateOrCreate*(  ['complaint\_id' => $complaint->id],  ['end\_date' => $request->get('ban\_end\_date')]  );  }  else  Ban::*where*('complaint\_id', '=', $complaint->id)->delete();   $complaint->update($request->validated());   return response()->json(ComplaintResource::*make*($complaint));  }   public function show(News $news, Comment $comment, Complaint $complaint): JsonResponse  {  return response()->json(ComplaintResource::*make*($complaint));  } } |

В *Листинге 36* представлен контроллер избранного (FavouriteController), который расположен в app/Http/Controllers/Routes/FavouriteController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает все избранные новости пользователя.
2. store() – метод для добавления новости в избранное.
3. destroy() – метод для удаления новости из избранного.

Листинг 36 — Контроллер избранного «FavouriteController.php»

|  |
| --- |
| class FavouriteController extends Controller {  public function index()  {  $favouritesRequest = Favourite::*where*([  'user\_id' => auth()->id()  ])->orderBy('added\_date', 'desc');   $favourites = $favouritesRequest->paginate(15);   return response()->json([  'favourites' => FavouriteResource::*collection*($favourites),  'total' => $favourites->total()  ]);  }   public function store(FavouriteRequest $request)  {  $favourite = Favourite::*firstOrCreate*([  ...$request->validated(),  'user\_id' => auth()->id()  ], [  'added\_date' => now()  ]);   return response()->json(null, 201);  }   public function destroy(News $news)  {  Favourite::*where*([  ["news\_id", "=", $news->id],  ["user\_id", "=", auth()->id()]  ])->delete();   return response()->json(null, 204);  } } |

В *Листинге 37* представлен контроллер истории чтения (HistoryViewController), который расположен в app/Http/Controllers/Routes/HistoryViewController.php. Этот контроллер реализовывает следующие методы:

1. index() – возвращает историю чтения новостей пользователя.

Листинг 37 — Контроллер избранного «HistoryViewController.php»

|  |
| --- |
| class HistoryViewController extends Controller {  public function index(HistoryRequest $request)  {  $user = auth()->user();  $limit = intval($request->query('limit', 15));  $readDate = $request->get('date') ?? (new DateTime())->format('Y-m-d');   $historyQuery = HistoryView  ::*where*([  'read\_date' => $readDate,  'user\_id' => $user->id  ])  ->orderBy('read\_time', 'desc');   $historyViewPage = $historyQuery->paginate($limit);   return response()->json(  formatPaginate($historyViewPage, new HistoryViewResource(null))  );  } } |

В *Листинге 38* представлен контроллер изображений (ImageController), который расположен в app/Http/Controllers/Routes/ImageController.php. Этот контроллер реализовывает следующие методы:

1. upload() – метод для загрузки изображения на сервер.

Листинг 38 — Контроллер избранного «ImageController.php»

|  |
| --- |
| class ImageController extends Controller {  public function upload(ImageUploadRequest $request)  {  *// Текущая дата* $date = new DateTime();   *// Извлекаем конкретные числа* $day = $date->format('d');  $month = $date->format('m');  $year = $date->format('Y');   *// Сохраняем картинку по своему пути* $result = Image::*savePhoto*('image', "images/$year/$month/$day");   return response()->json(ImageResource::*make*($result));  } } |

В *Листинге 39* представлен контроллер новостей (NewsController), который расположен в app/Http/Controllers/Routes/NewsController.php. Этот контроллер реализовывает следующие методы:

1. index() – метод для получения всех новостей, из сортировки и фильтрации, а также поиска по ключевым словам.
2. show() – метод для просмотра полной версии новости.
3. store() – метод для создания новой новости репортёром.
4. update() – метод для обновления существующей новости.
5. destroy() – метод для удаления новости.

Листинг 39 — Контроллер новостей «NewsController.php»

|  |
| --- |
| class NewsController extends Controller {  public function index()  {  $query = News::*query*();   if (isset(request()->sort))  {  switch (request()->sort) {  case 'FirstOld':  $query->orderBy('id', 'asc');  break;  case 'Random':  $query->inRandomOrder();  break;  case 'FirstMoreViews':  $query->leftJoin('history\_views', 'history\_views.news\_id', '=', 'news.id')  ->select('news.\*', \DB::*raw*('COUNT(history\_views.user\_id) as views\_count'))  ->groupBy('news.id')  ->orderBy('views\_count', 'desc');  break;  case 'FirstNew':  $query->orderBy('id', 'desc');  }  }   if (isset(request()->search))  $query->where('title', 'like', '%' . request()->search . '%')  ->orWhere('content', 'like', '%' . request()->search . '%');   if (isset(request()->categories))  $query->whereIn('category\_id', explode(',', request()->categories));   if (request()->has("me")) {  $user = auth()->user();  if (isset($user))  $query->where('news.user\_id', '=', $user->id);  }   $paginate = $query->paginate(isset(request()->limit) ? request()->limit : 15); return response()->json([  "news" => NewsMinResource::*collection*($paginate),  "total" => $paginate->total()  ]);  }   public function show(News $news)  {  $user = auth()->user();  if ($user) {  $historyData = [  'news\_id' => $news->id,  'user\_id' => $user->id,  'read\_date' => (new DateTime())->format('Y-m-d'),  ];  HistoryView::*firstOrCreate*($historyData,  [  'read\_time' => now()->format('H:i:s')  ]);  }   return response()->json(NewsFullResource::*make*($news));  }   public function store(NewsCreateRequest $request)  {  $validatedData = $request->validated();   if (isset($validatedData['cover']))  {  $image = Image::*where*("hash", $validatedData['cover'])->first();  if ($image != null)  $cover\_id = $image->id;  }  else  $cover\_id = null;   *// Создаём новость* $news = News::*create*([  ...$request->validated(),  'user\_id' => auth()->id(),  'image\_id' => $cover\_id  ]);   *// Привязываем теги* if ($tags = $request->validated('tags'))  foreach ($tags as $tag)  NewsTag::*create*([  'news\_id' => $news->id,  'tag\_id' => $tag  ]);return response()->json([  'code' => 201,  'data' => NewsMinResource::*make*($news)  ], 201);  }   public function update(NewsUpdateRequest $request, News $news)  {  $updateData = [];  $validatedData = $request->validated();   if (isset($validatedData['title']))  $updateData['title'] = $validatedData['title'];   if (isset($validatedData['content']))  $updateData['content'] = $validatedData['content'];   if (isset($validatedData['cover']))  {  $image = Image::*where*("hash", $validatedData['cover'])->first();  if ($image != null)  $updateData['image\_id'] = $image->id;  }  else  $updateData['image\_id'] = null;   if (isset($validatedData['category\_id']))  $updateData['category\_id'] = $validatedData['category\_id'];   $news->update($updateData);   $tags = $request->validated('tags');  NewsTag::*whereNotIn*('tag\_id', $tags ?: [])->delete();  if ($tags) {  foreach ($tags as $tag) {  NewsTag::*updateOrCreate*([  'news\_id' => $news->id,  'tag\_id' => $tag  ]);  }  }return response()->json([  'code' => 200,  'data' => NewsFullResource::*make*($news)  ]);  }   public function destroy(News $news)  {  $news->delete();  return response()->json(null, 204);  } } |

В *Листинге 40* представлен контроллер реакций (ReactionController), который расположен в app/Http/Controllers/Routes/ReactionController.php. Этот контроллер реализовывает следующие методы:

1. index() – метод для просмотра всех типов реакций.
2. show() – метод просмотра конкретного типа реакции.
3. store() – метод для создания нового типа реакций.
4. update() – метод для обновления существующего типа реакций.
5. destroy() – метод для удаления типа реакций.
6. userReactionStore() – метод для добавления реакции пользователя на новость.
7. userReactionDestroy() – метод для удаления реакции пользователя с новости.

Листинг 40 — Контроллер реакций «ReactionController.php»

|  |
| --- |
| class ReactionController extends Controller {  public function index(): JsonResponse  {  return response()->json(Reaction::*all*());  }   public function show(Reaction $reaction): JsonResponse  {  return response()->json($reaction);  }   public function store(ReactionCreateRequest $request): JsonResponse  {  $reaction = Reaction::*create*($request->validated());  return response()->json([  'code' => 201,  'data' => $reaction  ], 201);  }   public function update(ReactionUpdateRequest $request, Reaction $reaction): JsonResponse  {  $reaction->update($request->validated());  return response()->json([  'code' => 200,  'data' => $reaction  ]);  }   public function destroy(Reaction $reaction): JsonResponse  {  $reaction->delete();  return response()->json(null, 204);  }   public function userReactionStore(News $news, Reaction $reaction)  {  $reaction->setReaction(auth()->user(), $news);  return response()->json(ReactionResource::*make*($reaction), 201);  }   public function userReactionDestroy(News $news)  {  Reaction::*deleteReaction*(auth()->user(), $news);  return response()->json(null, 204);  } } |

В *Листинге 41* представлен контроллер тегов (TagController), который расположен в app/Http/Controllers/Routes/TagController.php. Этот контроллер реализовывает следующие методы:

1. index() – метод для просмотра всех тегов.
2. store() – метод для создания нового тега.
3. show() – метод для просмотра информации о теге.
4. destroy() – метод для удаления тега.

Листинг 41 — Контроллер тегов «TagController.php»

|  |
| --- |
| class TagController extends Controller {  public function index(): JsonResponse  {  return response()->json(Tag::*all*());  }   public function store(TagCreateRequest $request): JsonResponse  {  $tag = Tag::*create*($request->validated());  return response()->json([  'code' => 201,  'data' => $tag  ]);  }   public function show(Tag $tag): JsonResponse  {  return response()->json($tag);  }   public function destroy(Tag $tag): JsonResponse  {  $tag->delete();  return response()->json(null, 204);  } } |

В *Листинге 42* представлен контроллер тегов (UserController), который расположен в app/Http/Controllers/Routes/UserController.php. Этот контроллер реализовывает следующие методы:

1. me() – метод для информацию пользователем о себе.
2. show() – метод для просмотра доступной информации о конкретном пользователе.
3. update() – метод для редактирования пользователя.
4. destroy() – метод для удаления пользователя.

Листинг 42 — Контроллер пользователей «UserController.php»

|  |
| --- |
| class UserController extends Controller {  public function me()  {  return response()->json(FullUserResource::*make*(auth()->user()));  }  public function show(Request $request, User $user)  {  $collection = $request->user()?->isAdministrator()  ? FullUserResource::*make*($user)  : MinUserResource::*make*($user);  return response()->json($collection);  }   public function update(UserUpdateRequest $request, User $user)  {  $data = $request->validated();  $dataToUpdate = [];   if (isset($data['avatar']))  {  $image = Image::*where*(['hash' => $data['avatar']])->first();  $dataToUpdate['image\_id'] = $image->id;  }   if (isset($data['firstName']))  $dataToUpdate['firstName'] = $data['firstName'];   if (isset($data['lastName']))  $dataToUpdate['lastName'] = $data['lastName'];   if (isset($data['email']))  $dataToUpdate['email'] = $data['email'];   if (isset($data['password']))  $dataToUpdate['password'] = $data['password'];   $user->update($dataToUpdate);   return response()->json(  $request->user()->isAdministrator()  ? new FullUserResource($user)  : new MinUserResource($user)  );  }   public function destroy(User $user)  {  $user->delete();  return response(null, 204);  } } |

Посредники

В Laravel существует механизм, который позволяет обрабатывать HTTP-запросы перед их передачей в контроллер. Этот механизм называется middleware (посредник).

Он выполняет роль фильтра, который может изменять запрос или отклонять его, если он не соответствует определённым условиям.

Посредник полезен для:

* аутентификации (проверки прав доступа пользователя);
* авторизации (разрешения доступа к определённому ресурсу);
* логирования и мониторинга запросов;
* фильтрации и изменения запроса перед обработкой контроллером.

Каждый middleware в Laravel — это обычный PHP-класс, который реализует метод handle($request, Closure $next). Метод handle принимает два параметра: запрос и функцию, которая будет вызвана после обработки запроса. Если функция не вызывается, запрос останавливается.

В *Листинге 43* показан посредник OptionalAuth, который проверяет авторизацию пользователя, если она передана, или возвращает пустоту в противном случае (явно сигнализирует об отсутствии авторизации). Добавляет необязательность токена (авторизации), где это возможно.

Листинг 43 — Посредник необязательности авторизации «OptionalAuth.php»

|  |
| --- |
| class OptionalAuth {  public function handle(Request $request, Closure $next, $allowNull = true)  {  if (request()->bearerToken() || !$allowNull) {  if ($user = Auth::guard('sanctum')->user())  Auth::setUser($user);  else throw new UnauthorizedException();  }  return $next($request);  } } |

Политики

В Laravel политики представляют собой классы, которые организуют логику авторизации для конкретной модели или ресурса. Они содержат методы, определяющие правила доступа к связанной модели или ресурсу. Эти методы возвращают значение true или false в зависимости от того, имеет ли пользователь право доступа.

Политики позволяют ограничить доступ к определённым маршрутам приложения аутентифицированным пользователям или пользователям с определённой ролью или уровнем разрешений.

Для всех роутов были определены политики, организующие их доступность пользователю. Так, весь функционал для блокировок доступен только администраторам *(см. Листинг 44)*, а в категориях частично доступен всем пользователям *(см. Листинг 45)*.

Листинг 44 — Политика доступа роутов контроллера блокировок, файл «BanPolicy.php»

|  |
| --- |
| class BanPolicy {  public function viewAll(User $user): bool { return $user->isAdministrator(); }   public function view(User $user, Ban $ban): bool { return $user->isAdministrator(); } } |

Листинг 45 — Политика доступа роутов контроллера категорий, файл «CategoryPolicy.php»

|  |
| --- |
| class CategoryPolicy {  public function viewAll(?User $user): bool { return true; }   public function delete(User $user, Category $category): bool { return $user->isAdministrator(); }   public function create(User $user): bool { return $user->isAdministrator(); } } |

Также, чтобы не описывать вручную отношение контроллера к политике, был написан класс, управляющий их подключением и в случае неполадок выводит соответствующие сообщения *(см. Листинг 46)*.

Листинг 46 — Автоматический регистратор политик, файл «PolicyMapRegister.php»

|  |
| --- |
| trait PolicyMapRegister {  use AuthorizesRequests;   protected string|null $modelToReg = null;  protected array $abilityMap = [  'index' => 'viewAll',  'store' => 'create',  'show' => 'view',  'update' => 'update',  'destroy' => 'delete'  ];  protected array $methodsWithoutModels = [];  *// К какой модели относится контроллер* protected string|null $target = null;   */\*\* Возвращает имя модели из строки-класса модели \*/* protected function getModelName(string $model): string  {  $separatedPath = explode('\\', $model);  return $separatedPath[array\_key\_last($separatedPath)];  }   */\*\* Связывает контроллер с политикой указанной модели, если она была задана \*/* public function \_\_construct()  {  $this->target ??= static::class;  $targetClass = $this->target;  $generatedAbilityMap = []; *// Итоговые правила мэппинга* $generatedWithoutModels = []; *// Перечисление моделей без параметров* $middleware = []; *// Для отладки   // Извлекаем имя для общего именования через контроллер* $name = str\_replace('Controller', '', $this->getModelName($targetClass));   $this->modelToReg ??= $this->searchModelByName($name);  $policyClass = "\App\Policies\\{$name}Policy";   *// Если политики не существует, то не привязываем ability* if (!class\_exists($policyClass))  return;   if (!class\_exists($this->modelToReg))  throw new Exception('Model ' . $this->modelToReg . ' not found');   $policyInfo = $this->getMapForClassMethods($policyClass);  $controllerInfo = $this->getMapForClassMethods($targetClass);   *// Перебираем ability политики* foreach ($policyInfo as $method) {  $abilityName = $method['name'];  $modelName = $this->modelToReg;   *// Имя текущего метода контроллера, связанного со способностью* $controllerMethodName = $this->getControllerMethodNameByAbilityName($abilityName);   *// Если способность и метод политики содержит модель с тем же именованием, как у контроллера* $isContains = $this->checkPolicyContains($method, $controllerMethodName, $controllerInfo, $name);   $generatedAbilityMap[$controllerMethodName] = $abilityName;   if (!$isContains)  $generatedWithoutModels[] = $controllerMethodName;  else  $modelName = lcfirst($name);   $method = null;  foreach ($controllerInfo as $controllerMethod)  if ($controllerMethod['name'] === $controllerMethodName)  $method = $controllerMethod;   if (!$method)  throw new Exception("The controller ({$controllerMethodName}) method is not set for the corresponding policy method");   $models = $this->getModelsNamesForMethodParams($method['params'], $name);  if (count($models))  $modelName = join(',', [$modelName, ...$models]);   $middleware["can:{$abilityName},{$modelName}"][] = $controllerMethodName; *// Для отладки* $this->middleware("can:{$abilityName},{$modelName}", [])->only($controllerMethodName);  }   $this->methodsWithoutModels = $generatedWithoutModels;  $this->abilityMap = $generatedAbilityMap;  }   public function searchModelByName($name)  {  $directories = new RecursiveDirectoryIterator(app\_path('Models'), \FilesystemIterator::*FOLLOW\_SYMLINKS*);  $iterator = new RecursiveIteratorIterator($directories);  foreach ($iterator as $dir)  if (!$dir->isDir() && substr($dir->getFileName(), 0, -(strlen($dir->getExtension()) + 1)) === $name) {  $path = $dir->getPath();  $path = str\_replace('/', '\\', $path);  while (strpos($path, 'app\\app') !== false) {  $path = str\_replace('app\\app', 'app', $path);  }  return 'App\\' . explode('app\\', $path)[1] . "\\{$name}";  }  return null;  }   */\*\* Получает имена моделей используемых в параметрах метода контроллера \*/* public function getModelsNamesForMethodParams($params, $nameToContinue): array  {  foreach ($params as $param)  if (str\_contains($param['typeHint'], '\Models\\')  && ($modelName = $this->getModelName($param['typeHint'])) !== $nameToContinue)  $names[] = lcfirst($modelName);   return $names ?? [];  }   */\*\* Получает имя связанного метода в контроллере со способностью \*/* public function getControllerMethodNameByAbilityName($controllerMethodName): mixed  {  return array\_search($controllerMethodName, $this->abilityMap) ?: $controllerMethodName;  }   */\*\* Проверяет, принимает ли политика и метод контроллера модель с именем контроллера \*/* public function checkPolicyContains($policyMethod, $controllerMethodName, $controllerMethods, $checkContainsName): bool  {  foreach ($policyMethod['params'] as $param)  if (str\_contains($param['typeHint'], $checkContainsName))  *// Если содержит, то перебираем методы контроллера* return $this->checkControllerContains($controllerMethods, $controllerMethodName, $checkContainsName);  return false;  }   */\*\* Проверяет, принимает ли метод контроллера модель с именем контроллера \*/* public function checkControllerContains($controllerMethods, $searchMethodName, $checkContainsName): bool  {  *// Если содержит, то перебираем методы контроллера* foreach ($controllerMethods as $controllerMethod) {  if ($controllerMethod['name'] != $searchMethodName)  continue;  *// Если метод контроллера содержится в карте способностей, то перебираем параметры этого метода* foreach ($controllerMethod['params'] as $controllerParameter)  *// Если метод и вправду использует этот класс, то помечаем* if (!str\_contains($controllerParameter['typeHint'], 'Request')  && str\_contains($controllerParameter['typeHint'], $checkContainsName)  && ($controllerParameter['position'] != 0 || count($controllerMethod['params']) == 1))  return true;  }  return false;  }   */\*\* Составляет карту методов у класса и их параметры \*/* public function getMapForClassMethods($class)  {  $map = []; *// Карта методов класса* $reflection = new ReflectionClass($class);   foreach ($reflection->getMethods() as $method) {  *// Перебираем все параметры методов* foreach ($method->getParameters() as $parameter)  $params[] = [  'name' => $parameter->getName(),  'typeHint' => $parameter->getType()?->getName(),  'position' => count($params ?? [])  ];  *// Сохраняем в карту методов класса* $map[] = [  'name' => $method->getName(),  'params' => $params ?? []  ];  }  return $map;  }   */\*\* Сопоставление методов политики с методами запросов \*/* protected function resourceAbilityMap(): array  {  return $this->abilityMap;  }   */\*\* Перечисление методов запросов, не принимающих модели \*/* protected function resourceMethodsWithoutModels(): array  {  return $this->methodsWithoutModels;  } } |

После необходимо зарегистрировать его, подключив к контроллеру *(см. Листинг 47)*.

Листинг 47 — Абстрактный класс контроллера, файл «Controller.php»

|  |
| --- |
| abstract class Controller extends \Illuminate\Routing\Controller {  use PolicyMapRegister; } |

Конфигурация

Файлы конфигурации в Laravel (config) используются для настройки приложения. Они позволяют централизованно хранить параметры, которые могут изменяться в зависимости от среды выполнения. Это обеспечивает гибкость и удобство при развёртывании проекта.

Файлы конфигурации находятся в папке config/. Значения в них можно задать через переменные окружения, которые хранятся в файле .env. Это позволяет легко адаптировать приложение под различные серверные окружения без изменения исходного кода.

В *Листинге 48* представлен файл конфигурации config/settings.php, который содержит настройки сервера. В нём определены допустимые форматы загружаемых файлов (allowed\_upload\_mimes), а также параметр, регулирующие ограничение размера файла для загрузки (max\_file\_size). Все значения загружаются из переменных окружения и могут быть изменены без редактирования кода приложения.

Листинг 48 — Конфигурация настроек сервера, файл «config/settings.php»

|  |
| --- |
| return [  'allowed\_upload\_mimes' => explode(',', env('ALLOWED\_UPLOAD\_MIMES', 'jpeg,jpg,png,gif,svg')),  'max\_file\_size' => env('MAX\_FILE\_SIZE', 1024 \* 1024 \* 5) *// 5 MiB* ]; |

Заполнители

В Laravel для инициализации базы данных используются специальные инструменты — заполнители (seeders). Они позволяют автоматически добавлять тестовые или стандартные данные в таблицы, что значительно упрощает процесс разработки и тестирования.

Заполнители особенно полезны при работе в команде, поскольку они обеспечивают согласованность данных на всех этапах разработки.

Для работы с данными в Laravel есть фасад DB и модель Eloquent, которые 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 49* 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 49 — Общий обязательный заполнитель, файл «database/seeders/DatabaseSeeder.php»

|  |
| --- |
| class DatabaseSeeder extends Seeder {  private $cities;  private $tags;  private $reasons;  private $reactions;  private $users;   public function run(): void  {  $this->call(CategoriesSeeder::class);User::*factory*()->create([  'login' => 'test',  'password' => 'test',  ]);   $reader = User::*factory*()->create([  'login' => 'reporter',  'password' => 'reporter',  'role' => 'reporter'  ]);   User::*factory*()->create([  'login' => 'admin',  'password' => 'admin',  'role' => 'admin'  ]);   $this->users = User::*factory*()->createMany(10);   $this->cities = City::*factory*()->createMany(10);  $this->tags = Tag::*factory*()->createMany(10);  $this->reasons = Reason::*factory*()->createMany(10);  $this->reactions = Reaction::*factory*()->createMany(10); $this->make\_news($reader);  }   private function make\_news(User $reporter)  {$image2\_1 = $this->UploadImage("news2\_1.jpg")["path"];  $image4\_1 = $this->UploadImage("news4\_1.jpeg")["path"];  $image5\_1 = $this->UploadImage("news5\_1.jpg")["path"];  $image5\_2 = $this->UploadImage("news5\_2.jpg")["path"];  $image5\_3 = $this->UploadImage("news5\_3.jpg")["path"];  $image5\_4 = $this->UploadImage("news5\_4.jpg")["path"];  $image5\_5 = $this->UploadImage("news5\_5.jpg")["path"];  $image7\_1 = $this->UploadImage("news7\_1.jpg")["path"];   $news = [  [  'title' => 'Все те же деньги: томский эксперт развеяла мифы о цифровом рубле',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_1.png'  ],  [  'title' => 'Ученый ТГУ: полностью избавиться от микропластика не получится',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_2.jpg'  ],  [  'title' => '"Магия больших зарплат" у курьеров больше не действует на томичей',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_3.jpg'  ],  [  'title' => 'Больше всего томичей ищут работу менеджеров, продавцов и водителей',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_4.jpeg'  ],  [  'title' => 'От купцов к инвесторам: 21 "дом за рубль" в Томске выставят на торги',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_5.jpg'  ],  [  'title' => 'Более 4,7 тыс томичей заявились на "Лыжню России" через Госуслуги',  'content' => "<текст\_новости>",  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_6.jpg'  ],  [  'title' => 'ТГУ закупил оборудование на 10 млн руб для изучения стресса растений',  'content' => "<текст\_новости>"  'user\_id' => $reporter->id,  'category\_id' => Category::*inRandomOrder*()->first()->id,  'image' => 'news\_7.jpg'  ]  ];   *// Сохраняем картинки в storage и базе* foreach ($news as &$item) {  $image = $this->UploadImage($item["image"])["image"];  $item["image\_id"] = $image->id;  unset($item["image"]);  }   *// Сохраняем категории в базе* foreach ($news as &$item) {  $createdNews = News::*create*($item);  $comments = Comment::*factory*(10)->create([  'news\_id' => $createdNews->id  ]);  }  }   public function UploadImage(string $filename)  {  *// Текущая дата* $date = new \DateTime();   *// Извлекаем конкретные числа* $day = $date->format('d');  $month = $date->format('m');  $year = $date->format('Y');   $image = Image::*saveLocalPhoto*("database/seeders/images/" . $filename, "images/$year/$month/$day");  return [  "image" => $image,  "path" => "storage/images/$year/$month/$day/" . $image->hash . "." . $image->extension  ];  } } |

Также есть заполнитель категорий новостей, представленный в *Листинге 50*.

Листинг 50 — Заполнитель категорий, файл «database/seeders/CategoriesSeeder.php»

|  |
| --- |
| class CategoriesSeeder extends Seeder {  public function run(): void  {  $categories = [  [ "name" => "Бизнес", "background\_color" => "E4F9E0", "image" => "Handshake\_green.svg" ],  [ "name" => "Еда & Культура", "background\_color" => "E6F0FD", "image" => "ForkKnife.svg" ],  [ "name" => "Офис & Работа", "background\_color" => "FFE5E5", "image" => "Coffee.svg" ],  [ "name" => "Экономика", "background\_color" => "FFF8E0", "image" => "CreditCard.svg" ],  [ "name" => "Работа", "background\_color" => "E6F0FD", "image" => "Buildings.svg" ],  [ "name" => "IT & Технологии", "background\_color" => "F7E6FD", "image" => "BugDroid.svg" ],  [ "name" => "Разработки", "background\_color" => "F5F7FA", "image" => "Handshake\_gray.svg" ],  [ "name" => "Дизайн", "background\_color" => "E4F9E0", "image" => "PenNib.svg" ],  [ "name" => "Образ жизни", "background\_color" => "FFF8E0", "image" => "Notepad.svg" ],  [ "name" => "Фото & Видео", "background\_color" => "E6F0FD", "image" => "Camera.svg" ],  [ "name" => "Здоровье & Фитнес", "background\_color" => "E4F9E0", "image" => "Heartbeat.svg" ],  [ "name" => "Строительство", "background\_color" => "FFE5E5", "image" => "Bug.svg" ],  ];   *// Сохраняем картинки в storage и базе* foreach ($categories as &$category) {  $image = Image::*saveLocalPhoto*("database/seeders/images/" . $category["image"], "images/categories");  $category["image\_id"] = $image->id;  unset($category["image"]);  }   *// Сохраняем категории в базе* foreach ($categories as &$category)  Category::*create*($category);  } } |

Автотестирование API

В процессе создания системы было проведено автоматическое тестирование API с использованием Hoppscotch. Для проверки корректности работы конечных точек были созданы тестовые сценарии, которые охватывали основные аспекты взаимодействия с сервером.

Тестирование включало в себя:

* проверку правильности обработки запросов (GET, POST, DELETE);
* оценку соответствия полученных данных ожидаемому формату;
* тестирование обработки ошибок и проверки данных;
* измерение времени отклика API.

Разработанные тесты позволили выявить и устранить возможные проблемы на ранних этапах разработки, что обеспечило стабильность и предсказуемость работы сервиса.