МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка веб-приложения «Медиа-галерея»

Выполнил студент Чернявский Михаил Александрович

(Ф.И.О.)

Руководитель проекта ассистент Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ассистент Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ассистент Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2017

**Реферат**

МЕДИА-ГАЛЕРЕЯ, NodeJS, HEROKU, REST-API, ORM, НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ, МОДУЛЬНЫЕ-ТЕСТЫ

Основной целью курсового проекта является разработка веб-приложения «Медиа-галерея».

Пояснительная записка состоит из введения, шести разделов и заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается архитектура приложения и базы данных.

Во втором разделе представлен процесс и результаты разработки программного средства.

В третьем разделе описывается разработка веб-интерфейса.

В четвертом разделе представляется тестирование приложения.

В пятом разделе описывается процесс развертывания приложения.

В шестом разделе описывается руководство пользователя

Заключение состоит из итогов курсового проекта и задач, решенных в ходе проектирования и разработки приложения.

Пояснительная записка курсового проекта содержит 26 страниц, 23 рисунков, 2 таблицы, 6 источников литературы и 3 приложения.

Содержание

[Определения 6](#_Toc483036390)

[Введение 7](#_Toc483036391)

[1. Архитектура приложения и базы данных 8](#_Toc483036392)

[2. Разработка веб-приложения 10](#_Toc483036393)

[3. Разработка веб-интерфейса 14](#_Toc483036394)

[4. Тестирование 15](#_Toc483036395)

[5. Развертывание приложения 16](#_Toc483036396)

[6. Руководство пользователя 17](#_Toc483036397)

[Заключение 21](#_Toc483036398)

[Список использованных источников 22](#_Toc483036399)

[ПРИЛОЖЕНИЕ А 23](#_Toc483036400)

[ПРИЛОЖЕНИЕ Б 25](#_Toc483036401)

[ПРИЛОЖЕНИЕ В 26](#_Toc483036402)

# Определения

БД – база данных.

СУБД – система управления базой данных.

URL – Uniform Resource Locator. Определитель местонахождения ресурса.

ORM – Object-Relational Mapping. Объектно-реляционное отображение.

REST – Representational State Transfer. Передача состояния представления.

API – Application Programming Interface. Интерфейс программирования приложений.

PaaS - Platform as a Service. Платформа как услуга.

Бизнес-логика – реализация предметной области в информационной системе.

CRUD-операции – create, read, update, delete - создать, прочесть, обновить, удалить. Четыре базовые функции, используемые при работе с персистентными хранилищами данных.

# Введение

В настоящее время – время, когда запечатлеть жизненные моменты не составляет труда, время, когда нет необходимости в покупке фотопленок, фотокамер, фотоаппаратов, для совершения, хотя бы примитивного качества, снимков. Во время информационной одаренности встает проблема хранения тех самых зафиксированных моментов, к примеру, загородных посиделок с друзьями. Хотелось бы, чтобы всё лежало в одном месте, и ,при малейшем желании вспомнить какое-то яркое событие, можно было бы с легкостью к нему обратиться. Безусловно, выход есть.

А что, если хранить видео, фото в одной единой персональной галереи. Всё же станет намного проще, уже не надо будет метаться в вечном поиске забытой где-то флеш-карты, уже не надо волноваться, что, вдруг кто-то что-то удалил. Теперь, все ваши жизненные моменты в сохранности – о них позаботится «Медиа-галерея». Остается лишь нажать на кнопку «Загрузить».

В соответствии с заданием курсового проекта следует разработать веб-приложение «Медиа-галерея».

# 1. Архитектура приложения и базы данных

Для разработки веб-приложения использовалась программная платформа Node.js. Данная платформа  позволяет  с  помощью  JavaScript  создавать  веб-сервера [5].

Серверная часть приложения состоит из контроллеров, сервисов, и модели используемых сущностей. Подробней о каждом компоненте будет рассказано в разделе 2.

ORM позволяет преобразовывать несовместимые типы моделей в ООП, в частности, между хранилищем данных и объектами программирования. ORM используется для упрощения процесса сохранения объектов в реляционную базу данных и их извлечения, при этом ORM сама заботится о преобразовании данных между двумя несовместимыми состояниями.

*База данных*

База данных – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами. База данных нашего программного средства состоит из 5 таблиц, и ее схема изображена на рисунке 1.1.

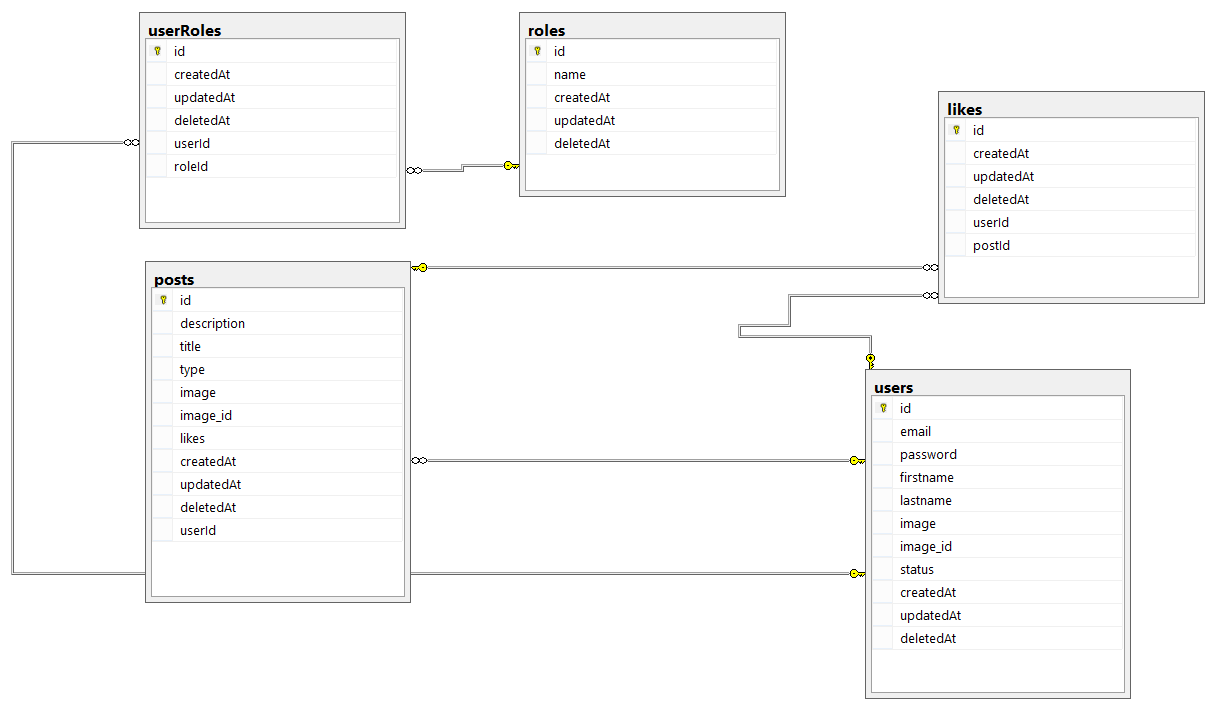


Рисунок 1.1 – Диаграмма базы даных

* *USERS* *–*  содержит информацию о зарегистрированных пользователях. В ней содержаться следующие поля:
  1. id – идентификатор пользователя (первичный ключ);
  2. email – e-mail адрес пользователя;
  3. password – пароль пользователя;
  4. firstname – имя пользователя;
  5. lastname – фамилия пользователя;
  6. image – ссылка на персональное фото пользователя;
  7. image\_id – уникальное наименование персонального фото пользователя;
  8. status – статус блокировки пользователя.
* *POSTS*  *–*  содержит   информацию о публикациях пользователей. В ней содержаться следующие поля:
  1. id – идентификатор публикации (первичный ключ);
  2. description – описание публикации;
  3. title – заголовок публикации;
  4. type – тип публикации;
  5. image – ссылка на медиа-файл;
  6. image\_id – уникальное наименование медиа-файла;
  7. likes – рейтинг публикации;
  8. userId – идентификатор зарегистрированного пользователя (внешний ключ на колонку id в таблице *USERS*);
* *ROLES*  *–*  содержит   информацию о ролях приложения. В ней содержаться следующие поля:

1. id – идентификатор роли (первичный ключ);
2. name – название роли.

* *LIKES*  *–*  содержит   информацию о рейтингах публикаций пользователей. В ней содержаться следующие поля:

1. id – идентификатор записи в таблице (первичный ключ);
2. userId – идентификатор зарегистрированного пользователя (внешний ключ на колонку id в таблице *USERS*);
3. postId – идентификатор опубликованной записи (внешний ключ на колонку id в таблице *POSTS* ).

* *USERROLES*  *–*  содержит   информацию о ролях пользователей. В ней содержаться следующие поля:

1. id – идентификатор записи в таблице (первичный ключ);
2. userId – идентификатор зарегистрированного пользователя (внешний ключ на колонку id в таблице *USERS.*
3. roleId – идентификатор роли (внешний ключ на колонку id в таблице *ROLES* ).

Каждая таблица имеет 3 системных поля: *createadAt, updatedAt,  deletedAt*  (данные поля необходимы для работы ORM). Диаграмма базы данных представлена в приложении А.

При разработке приложения использовалась локальная база данных, а при развертывании на сервере Heroku подключался дополнительный модуль, который позволял создавать БД на сервере.

Базы данных Heroku имеют широкую совместимость с различными облаками. К ней могут подключаться приложения, построенные в Google App Engine, Microsoft Azure, Amazon Web Services' Elastic Compute Cloud (EC2), а так же приложения созданные на собственном компьютере разработчика.

# 2. Разработка веб-приложения

При разработке RESTful сервисов удобно разделять проект на несколько основных частей [6]. В данном курсовом проекте таковыми являются контроллеры и сервисы [1]. Все компоненты проекта и описание каждого из них представлены в таблице 2.1.

Таблица 2.1 – Компоненты проекта

|  |  |
| --- | --- |
| **Компонент** | **Описание** |
| Context | Настройка ORM и установка связей между моделями |
| Controllers | Реализация всех контроллеров, в том числе базового |
| Models | Содержит описание сущностей для ORM-фреймворка |
| Services | Сервисы для работы веб-приложения |
| Public | Хранит все статические веб-ресурсы, в том числе документацию к API |
| Tests | Совокупность тестов |
| Utils | Вспомогательные инструменты |
| Views | Клиентская часть веб-приложения |

*Контроллеры*

Основной функцией контроллеров является обеспечение связи между пользователем и системой. Они контролируют и направляют данные от пользователя к системе, а так же реакцию системы – пользователю. В основном контроллеры не занимаются бизнес-логикой, но в случае пассивной модели они могут реализовывать и бизнес-слой.

В данном проекте всего 4 контроллера. Далее будет расписан каждый из них.

Контроллер Auth. Обрабатывает все запросы по пути */auth.* Позволяет производить регистрацию и аутентификацию пользователей в системе. Вызывает методы описанные в сервисе Auth.

Контроллер Posts. Обрабатывает все запросы по пути */posts.* Позволяет создавать новые публикации, просматривать уже существующие, изменять, удалять, оценивать и искать публикации по ключевому слову. Вызывает методы описанные в сервисе Posts.

Контроллер User. Обрабатывает все запросы по пути */users.* Позволяет блокировать, удалять пользователей, предоставляет панель администратора и пользователя. Вызывает методы описанные в сервисе User.

Контроллер Api. Собирает все ранее описанные контроллеры и добавляет их к маршрутизатору под своими путями. Вызывает методы описанные в сервисе Api.

Примеры контроллеров находятся в приложении А.

*Сервисы*

Сервисы занимаются основной логикой проекта. И обеспечивают правильную работу приложения. Содержатся все необходимые методы для работы с маршрутами. Описание каждого сервиса представлено в таблице 2.2.

Таблица 2.2 - Сервисы

|  |  |
| --- | --- |
| **Сервис** | **Описание** |
| Auth | Осуществляет регистрацию и аутентификацию пользователей в системе. |
| Posts | Позволяет создавать новые публикации, просматривать уже существующие, удалять, изменять, оценивать и и искать публикации по ключевому слову. |
| User | Позволяет блокировать, удалять пользователей. Предоставляет панель администратора и пользователя. |

Примеры сервисов находятся в приложении Б.

*Модели*

Курсовой проект включает в себя 5 модели: *like*, *post, role, user, userRole*. На основе этих моделей будут созданы таблицы в базе данных. Пример описания модели *users* представлен на рисунке 2.1.

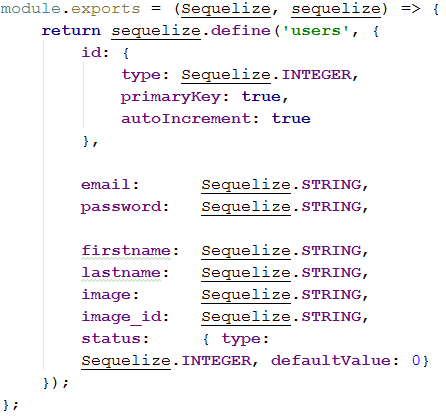


Рисунок 2.1 – Пример описания модели *users*

Пример описания модели *posts* представлен на рисунке 2.2.

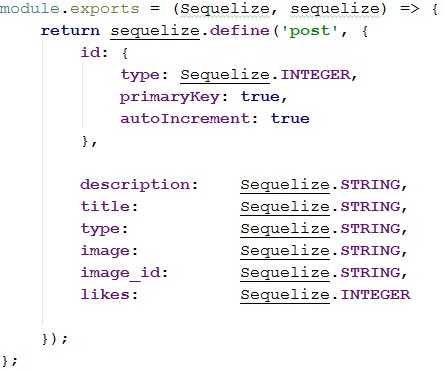


Рисунок 2.2 – Пример описания модели *post*

Пример описания модели *like* представлен на рисунке 2.3.

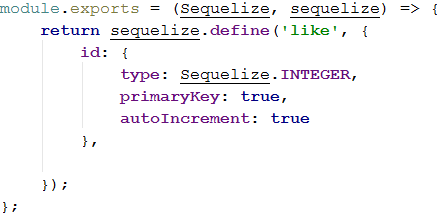


Рисунок 2.3 – Пример описания модели *like*

Пример описания модели *roles* представлен на рисунке 2.4.

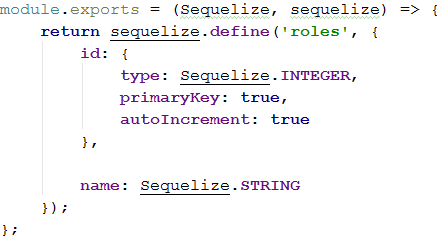


Рисунок 2.4 – Пример описания модели *roles*

Пример описания модели *userRoles* представлен на рисунке 2.5.

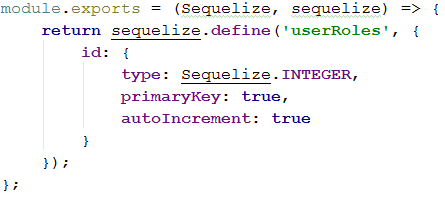


Рисунок 2.5 – Пример описания модели *userRoles*

*Документация*

Для написания документации использовался язык разметки YAML. Используется как формат для файлов конфигурации. Создавался с целью акцентирования внимания на данных, а не на разметке документа. Исходный код документации и ее графическое представление представлены на рисунках 2.6, 2.7.

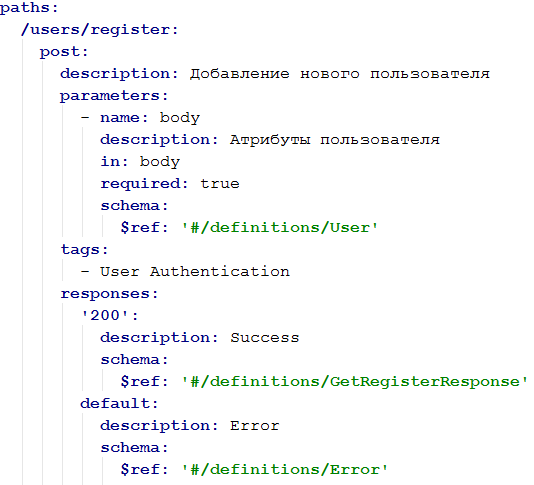


Рисунок 2.6 – Пример документации для контроллера

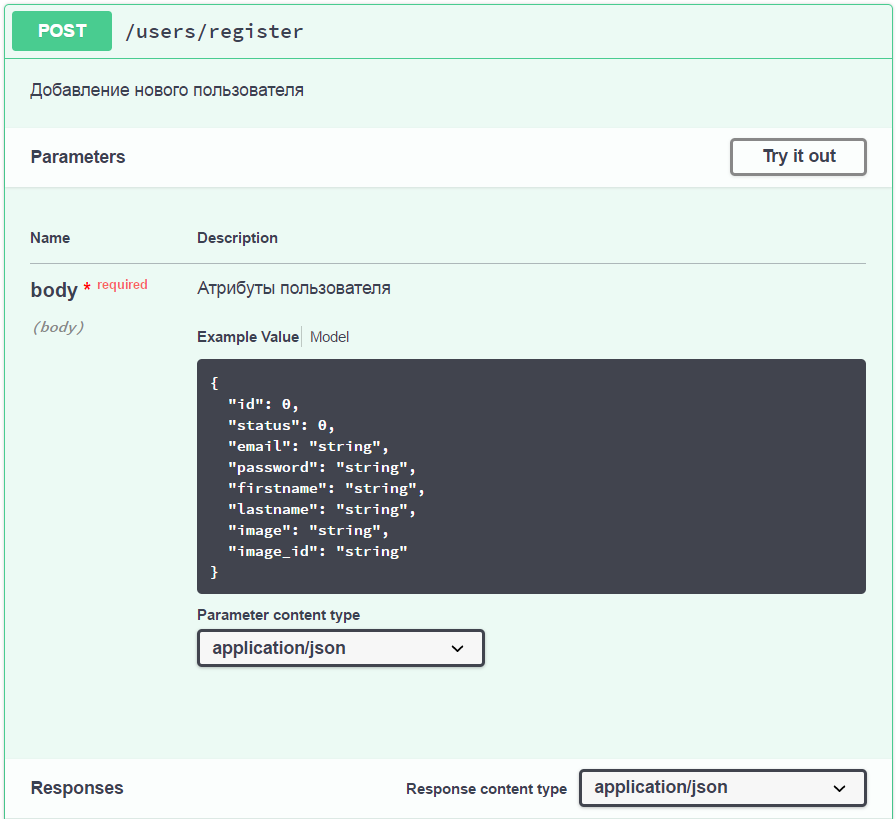


Рисунок 2.7 – Интерфейс готовой документации

Для создания документации использовалась технология swagger [3], которая позволяет документировать REST-сервисы. Swagger поддерживает множество языков программирования и фреймворков. Также Swagger предоставляет UI для просмотра документации.

# 3. Разработка веб-интерфейса

Для разработки веб-интерфейса использовался  [HTML](https://ru.wikipedia.org/wiki/HTML) с применением [CSS](https://ru.wikipedia.org/wiki/CSS) и [JavaScript](https://ru.wikipedia.org/wiki/JavaScript). Для стилизации интерфейса использовался фреймворк для создания переносимых интерфейсов – Semantic UI. Пример реализации компонента и его отображение на веб-странице представлены на рисунках 3.1 и 3.2.



Рисунок 3.1 – Реализация интерфейса

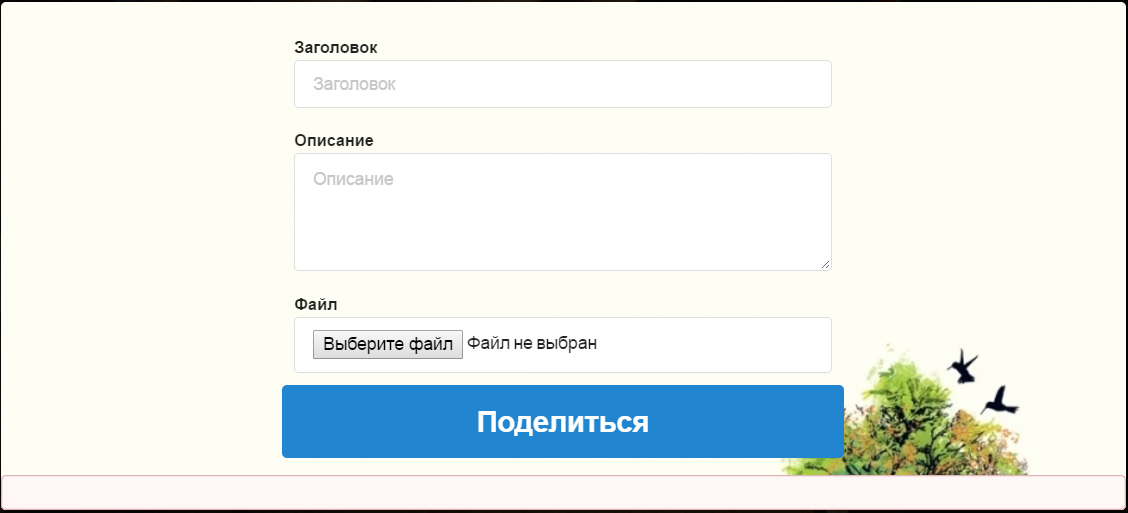


Рисунок 3.2 – Отображение на веб-странице

Для удобной работы применялся шаблонизатор ejs для платформы Node.js. Этот движок использует синтаксис, который во многом был заимствован из движка представлений Web Forms на платформе ASP.NET, который разрабатывался в компании Microsoft. Его преимуществами являются: простота в использовании и простая интеграция с модулем Express для Node.js. Недостатками этого шаблонизатора заключаются в его малой функциональности и отсутствии официального поставщика. Применялся фреймворк Semantic UI для создания переносимых интерфейсов. Главной особенностью которого является: гибкость отображение страниц на всех устройствах.

# 4. Тестирование

При разработке и тестировании приложения использовался подход BDD – разработка, основанная на функционировании. Как правило, такие тесты иллюстрируют и тестируют различные сценарии, которые интересны непосредственно клиенту системы. Вся реализация курсового проекта покрыта модульными тестами. В них тестируются только отдельные методы, не затрагивая всю логику. Так же для всех обращений к другим ресурсам, например, запрос к базе данных, чтение с диска, необходимо создавать заглушки, которые будут эмулировать обращение к ресурсу.

За основу тестового окружения был взят фреймворк Mocha. Это javascript фреймворк для Node.js, который позволяет проводить асинхронное тестирование.

Основными его преимуществами является:

* Простая поддержка асинхронности, включая Promise.
* Поддержка таймаутов асинхронного выполнения.
* before, after, before each, after each хуки.
* Использование любой assertion библиотеки.

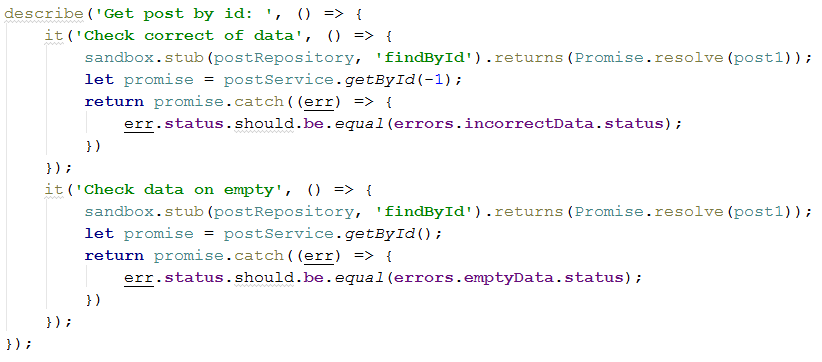


Рисунок 4.1 – Модульный тест

Пример успешно пройденных тестов показан на рисунке 4.2.

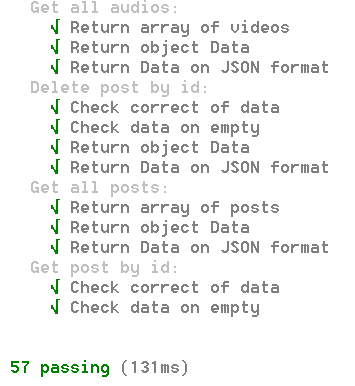


Рисунок 4.2 – Успешно пройденные тесты

# 5. Развертывание приложения

Веб-приложение разворачивалось на Heroku. Это облачная PaaS-платформа с поддержкой многих языков программирования. Основным преимуществом платформы является наличие бесплатного тарифа с ограниченными возможностями.

Для предотвращения ошибок на рабочем сервере использовался распределённый веб-сервис для сборки и тестирования программного обеспечения Travis CI. Подключаясь к системе контроля версий, Travis CI выполняет тестирование приложения до того, как изменения будут приняты в основную ветку. В случае успешно пройденных тестов, все изменения сливаются в главную ветку, иначе изменения не принимаются и сервис оповещает пользователя о неудачных тестах. Пример успешно пройденных тестов на сервисе изображен на рисунке 5.1.

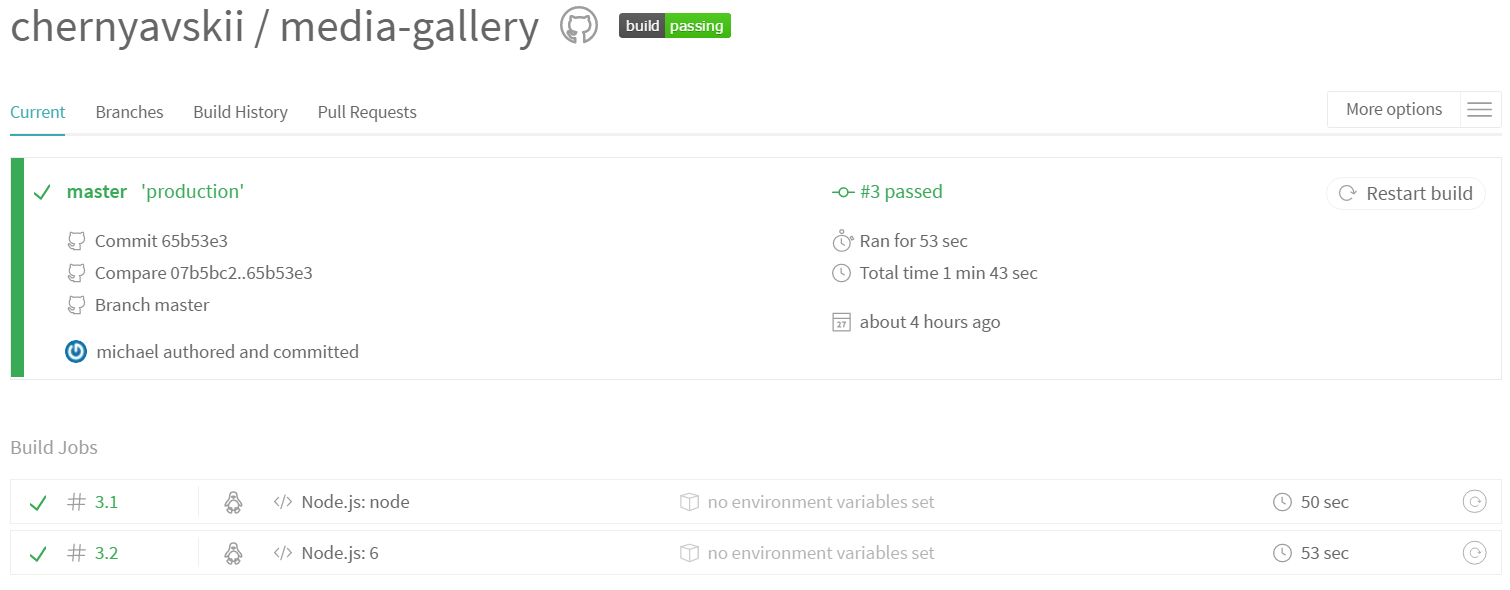


Рисунок 5.1 – Успешно пройденные тесты в Travis CI

Heroku предоставляет автоматическое развертывание при помощи систем контроля версий GitHub и DropBox. Пример включения автоматического развертывания после успешного прохождения тестов изображен на рисунке 5.2.

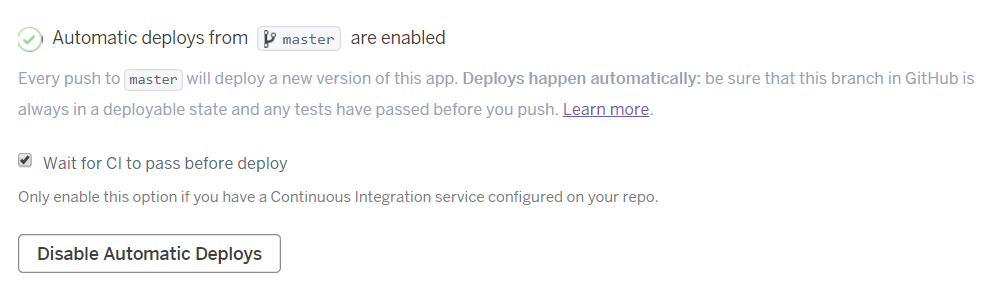


Рисунок 5.2 – Включение автоматического развертывания после прохождения тестов

# 6. Руководство пользователя

При переходе на сайт пользователь попадает на главную страницу. Если данный пользователь не авторизован либо не зарегистрирован в системе, ему необходимо перейти на страницу логина. Только после успешной авторизации, пользователям предоставляется возможность использовать данный ресурс. Ограничение входа, наложенное на не авторизованных пользователей в системе, представлено на рисунке 6.1.

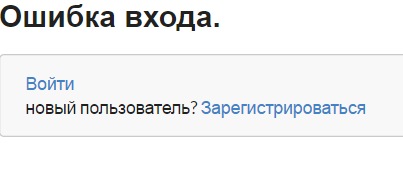


Рисунок 6.1 – Ограничение входа для не авторизованных пользователей

На странице логина возможно создание нового либо авторизация уже существующего пользователя в системе. После успешного прохождения данного этапа, пользователю присваивается необходимая роль для пользования данным ресурсом. Формы входа и регистрации представлены на рисунках 6.2, 6.3.

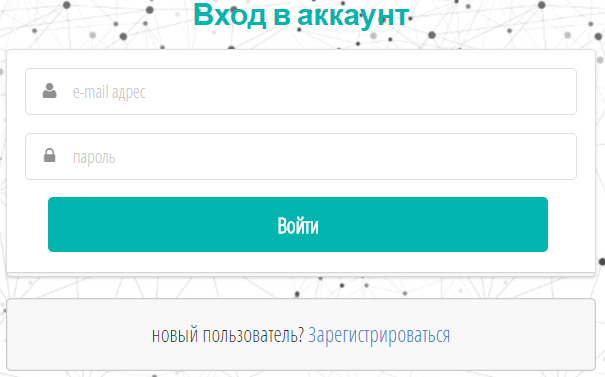


Рисунок 6.2 – Форма входа существующего пользователя

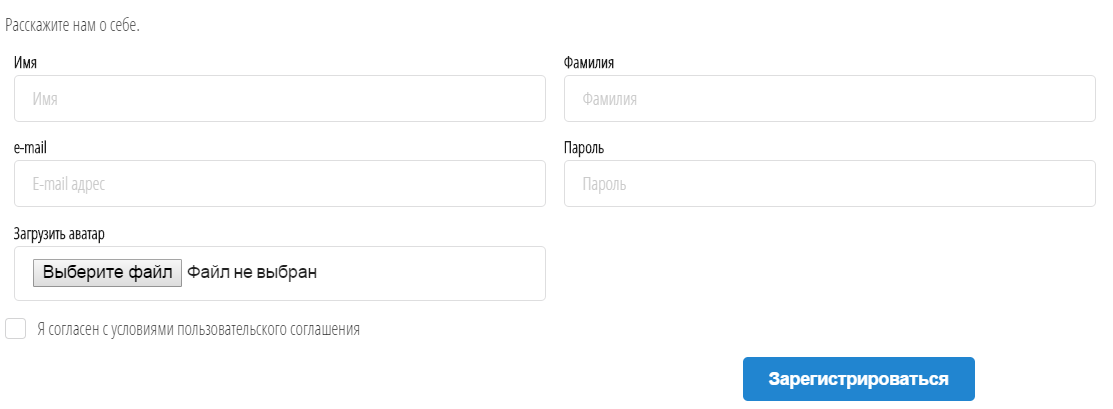


Рисунок 6.3 – Форма регистрации нового пользователя

*Пользователь*

Пользователю предоставляется возможность делиться со своими медиа-файлами с пользователями. Просматривать и оценивать публикации других пользователей, делится ссылками на медиа-файлы в социальных сетях. Для создания новой публикации необходимо заполнить форму. Окно создания новой публикации предоставлено на рисунке 6.4.

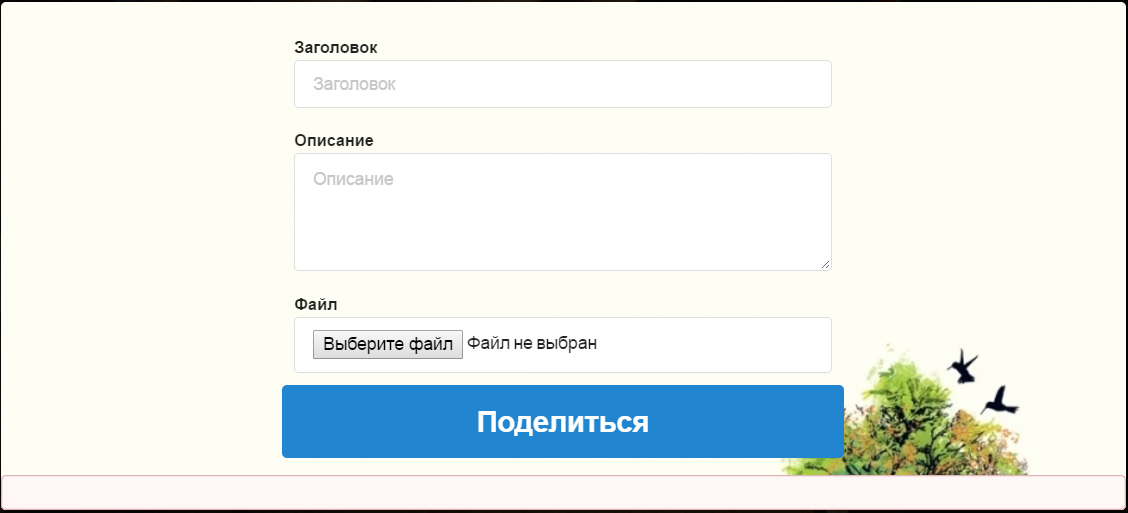


Рисунок 6.4 – Форма создания новой публикации

Для удаления или изменения записи необходимо нажать соответствующие кнопки. Кнопка «Удалить» удалит выбранную запись безвозвратно. Кнопка «Изменить» перенаправит пользователя на страницу изменения публикации. Кнопки удаления и изменения записи представлены на рисунке 6.5.



Рисунок 6.5 – Кнопки удаления и изменения записи

Право на удаление и изменение записей распространяется только на самих авторов данных записей. Страница «Изменения публикации» содержит в себе необходимые поля для обновления существующей записи. Форма изменения записи представлена на рисунке 6.6.

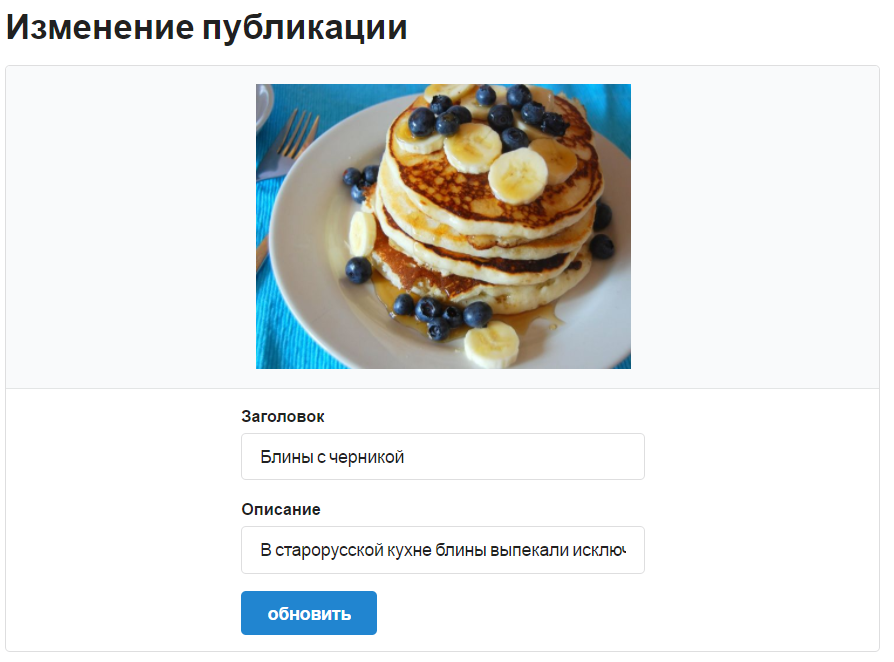


Рисунок 6.6 – Форма изменения записи

Для поиска публикации необходимо вписать в поле ключевое слово, по которому будет осуществляется поиск по записям. Важно знать, что данная форма поиска учитывает регистр символов. Поле ввода ключевого слово для поиска изображено на рисунке 6.7.

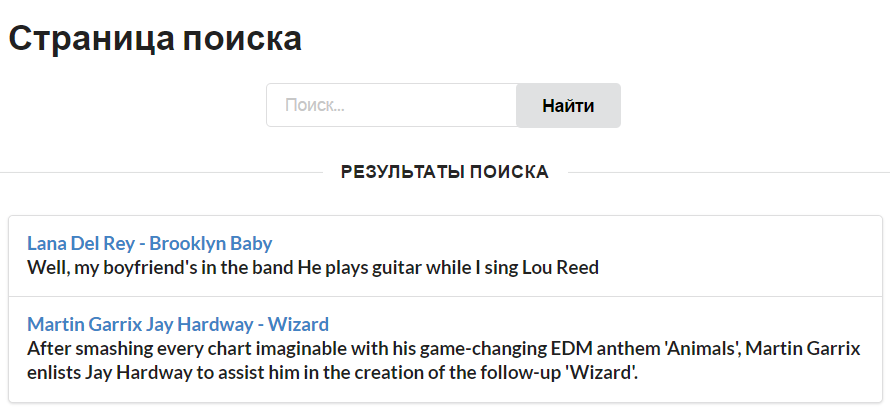


Рисунок 6.7 – Форма поиска публикаций

Для того чтобы поделится ссылкой в социальных сетях необходимо нажать на кнопку и подтвердить отправку ссылки. Кнопка, позволяющая отправлять ссылки на медиа-файлы в другие социальные сети, изображена на рисунке 6.8.

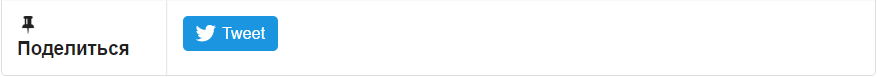


Рисунок 6.8 – Кнопка отправки ссылки на медиа-файл

*Администратор*

Администратору предоставляется право на удаление и изменение статуса пользователя. Другие пользователи к панели администратора доступа не имеют. Панель администратора представлена на рисунке 6.9.

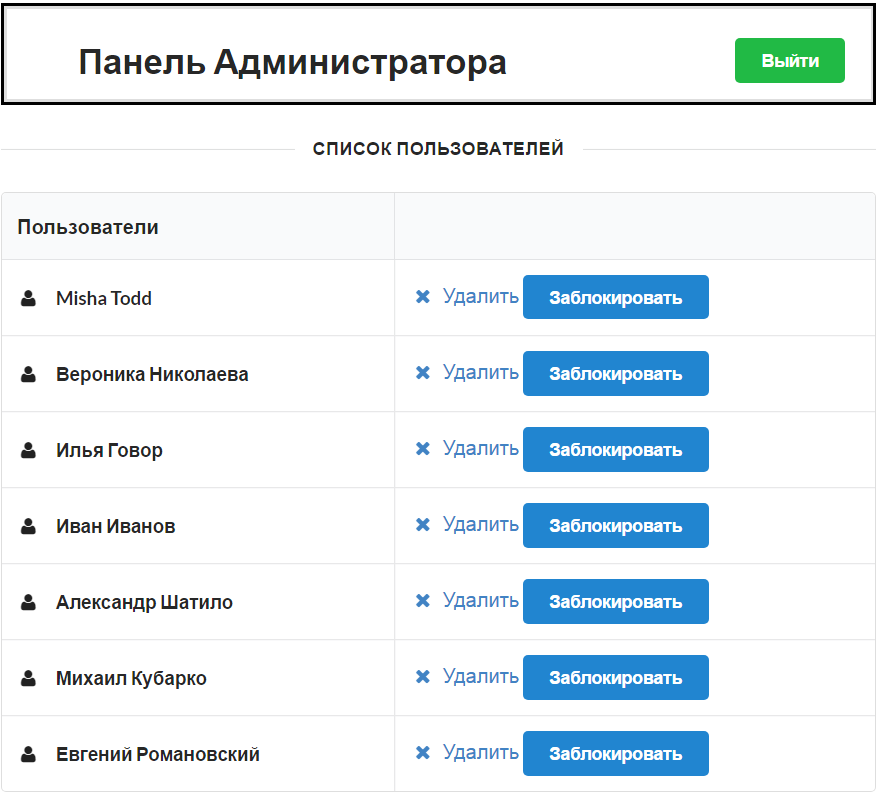


Рисунок 6.9 – Панель администратора

Клиентская часть приложения не является полноценной реализацией созданного API. Для полного ознакомления с возможностями данного веб-приложения необходимо обратиться к документации, которая расположена по адресу <https://my-media-gallery.herokuapp.com/public/swagger/public.html>

# Заключение

В ходе разработки курсового проекта были решены все поставленные задачи. Таким образом, была достигнута цель и создано веб-приложение «Медиа-галерея».

В соответствии с полученным результатом работы можно сделать вывод, что разработанное веб-приложение работает верно, а требования технического задания выполнены в полном объеме.

В ходе выполнения курсового проекта приложение способно выполнять следующие функции:

* Выполнять регистрацию и авторизацию пользователей.
* Сохранять рабочую информацию в централизованной базе данных.
* Добавлять, удалять и редактировать медиа-файлы.
* Обеспечивать возможность просматривать и оценивать медиа-файлы пользователей.
* Осуществлять поиск медиа-файлов по ключевому слову.
* Управлять ролями пользователей.
* Наделять администратора правами на просмотр, добавление, удаление, редактирование всех записей базы данных приложения.
* Позволять возможность пользователям делиться ссылками на медиа-файлы в социальных сетях.

Не смотря на огромное количество аналогичных проектов в интернете, данное веб-приложение способно составить конкуренцию за счет некоторых особенностей, а именно:

* Хорошо спроектированное и общедоступное API.
* Удобный интерфейс пользователя.
* Сервер написан на кроссплатформенной программной платформе Node.js.

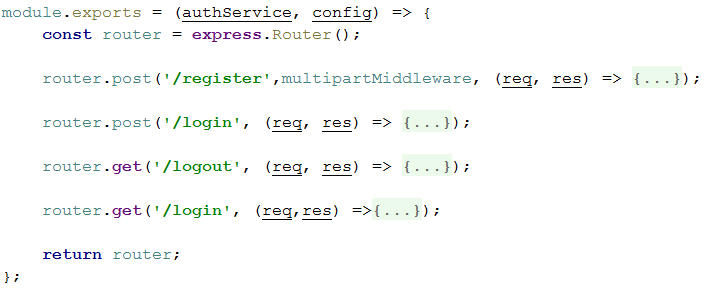
На сегодняшний день Node.js начинает набирать популярность как клатформа для разработки веб-серверов, а язык JavaScript уже стал одним из популярнейших языков программирования.

# Список использованных источников

1. Цыганенко Н.П. Курс лекций по программированию серверных кроссплатформенных приложений / Н.П. Цыганенко
2. Cloudinary.com [Электронный ресурс] / cloudinary.com – Режим доступа: [https:// http://cloudinary.com/documentation/](https://stackoverflow.com/questions/29698676/create-document-from-swagger-node-express-spec-js-and-model-js%20) Дата доступа 17.02.2017
3. Swagger.io [Электронный ресурс] / swagger.io – Режим доступа: <http://swagger.io/specification/> Дата доступа 27.03.2017
4. Sinonjs.org [Электронный ресурс] / sinonjs.org – Режим доступа: <http://sinonjs.org/> Дата доступа 21.03.2017
5. Swagger.io [Электронный ресурс] / swagger.io – Режим доступа: <http://swagger.io/specification/> Дата доступа 27.03.2017
6. «Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript» Итан Браун.

# ПРИЛОЖЕНИЕ А

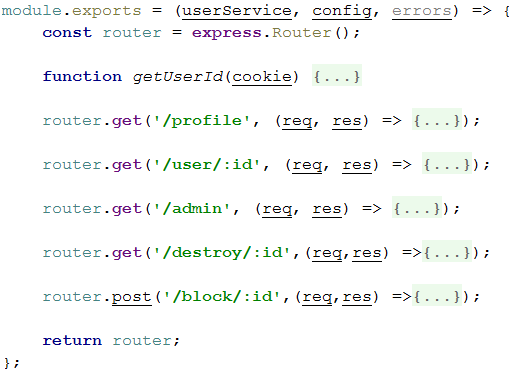
**Контроллер Auth**

****

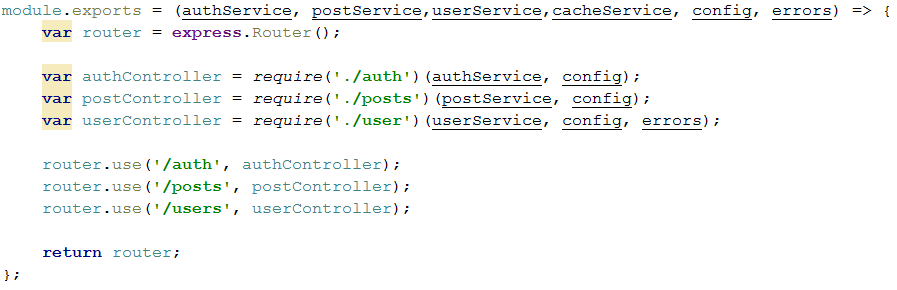
**Контроллер Posts**

****

**Контроллер User**

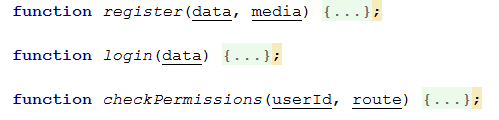
****

**Контроллер Api**

****

# ПРИЛОЖЕНИЕ Б

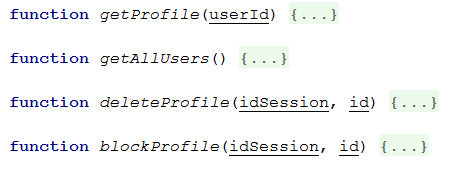
**Сервис Auth**

****

**Сервис Posts**

****

**Сервис User**

****

# ПРИЛОЖЕНИЕ В

**Методы API**

Таблица Г.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Путь** | **Параметры** | **Ответ** | **Коды ошибок** | **Описание** |
| POST | /auth/register | data, media | {  roleId: 0, userId: 0  } | 401,407 | Регистрация нового пользователя |
| POST | /auth/login | data | {  roleId: 0, userId: 0  } | 405, 401,407 | Авторизация пользователя |
| POST | /posts/create | userId,post,media,media\_type | {  userId:0,  title: “string”, description: “string”, image: “string”, image\_id: “string”,  type: “string”,  } | 401,408 | Создание новой публикации |
| GET | /posts/post/:id | id | **{**  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: “string”,  type: “string”,  } | 408,401 | Получение публикации по id |
| GET | /posts/ |  | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] |  | Получучение всех публикации |

Продолжение таблицы Г.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Путь** | **Параметры** | **Ответ** | **Коды ошибок** | **Описание** |
| GET | / posts/destroy/:id | id | {  success: "true"  } | 408,401 | Удаление записи |
| GET | /posts/audio |  | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] |  | Получение всех аудиозаписей |
| GET | /posts/video |  | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] |  | Получение всех видеозаписей |
| GET | /posts/images |  | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] |  | Получение всех изображений |
| GET | /posts/like/:id | id | **{**  **userId**:0,  **postId**:0  } | 405, 401,407 | Оценивание публикации |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Путь** | **Параметры** | **Ответ** | **Коды ошибок** | **Описание** |
| GET | /posts/ userimages | id | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] | 401,407 | Получение всех изображений пользователя |
| GET | /posts/ uservideo | id | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] | 401,407 | Получение всех видеозаписей пользователя |
| GET | /posts/useraudio | id | [  {  **id**:0, title: “string”, description: “string”, image: “string”, image\_id: }] | 401,407 | Получение всех аудиозаписей пользователя |
| POST | /posts/search | word | {  id:0,  title: “string”, description: “string”, image: “string”, image\_id: } | 401 | Поиск среди публикаций |