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:**

WEB-приложение «SpaceX»

Выполнил студент Грунковский Роман Иванович

(Ф.И.О.)

Руководитель проекта преп.-стаж. Сенюк В.К.

(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 преп.-стаж. Сенюк В.К.

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

Нормоконтролер преп.-стаж. Сенюк В.К.

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Введение………………………………………………………………………..3](#_Toc41075293)

[1. Аналитический обзор литературы…………………….……………………4](#_Toc41075294)

[1.1. Аналитический обзор источников……………………………………….4](#_Toc41075295)

[1.2. Обзор аналогов……………………………………………………………4](#_Toc41075296)

[2. Проектирование архитектуры……………………………………………...6](#_Toc41075297)

[2.1. Обобщенная структура……………………………………………………6](#_Toc41075298)

[2.2. Взаимосвязь всех компонентов…………………………………………..6](#_Toc41075299)

[3. Разработка функциональной модели и модели данных ПС………………8](#_Toc41075300)

[3.1. Проектирование базы данных……………………………………………8](#_Toc41075301)

[3.2. Проектирование архитектуры проекта…………………………………..8](#_Toc41075302)

[4. Создание программного средства………………………………………...11](#_Toc41075303)

[4.1. EJS………………………………………………………………………...11](#_Toc41075304)

[4.2. Express…………………………………………………………………….11](#_Toc41075305)

[4.3. Gulp…………………………………………..……………………………11](#_Toc41075306)

[4.4. ESLint и nodemon………………………………………………………...11](#_Toc41075307)

[4.5. Sharp………………………………………………………………………12](#_Toc41075308)

[5. Тестирование………………………………………………………………13](#_Toc41075309)

[5.1. Посты…………………………………………………………………….13](#_Toc41075310)

[5.2. Комментарии…………………………………………………………….17](#_Toc41075311)

[6. Руководство пользователя………………………………………………...20](#_Toc41075312)

[Заключение……………………………………………………………………23](#_Toc41075313)

[Список литературы…………………………………………………………...24](#_Toc41075314)

[ПРИЛОЖЕНИЕ А…………………………………………………………….25](#_Toc41075315)

[ПРИЛОЖЕНИЕ Б……………………………………………………………..26](#_Toc41075316)

[ПРИЛОЖЕНИЕ В…………………………………………………………….27](#_Toc41075317)

[ПРИЛОЖЕНИЕ Г……………………………………………………….…….28](#_Toc41075318)

[ПРИЛОЖЕНИЕ Д…………………………………………………………….29](#_Toc41075319)

# Введение

Цель моего курсового проекта – разработать ресурс для размещения статей пользователями. На этом ресурсе пользователи могут обсуждать посты других пользователей, комментируя посты, а так же создавать свои собственные статьи, добавлять картинки, редактировать в случае необходимости.

Пояснительная записка состоит из нескольких разделов.

1. Аналитический обзор литературы. Данный раздел содержит обзор литературы по теме курсового проекта, примеры решения аналогичных задач.
2. Анализ и проектирование архитектуры приложения. В результате работы над этим разделом должны быть сформулированы функциональные требования для проектирования программного средства.
3. Проектирование программного средства. Описание работы программы. Описание данных.
4. Создание программного средства. Описание некоторых модулей приложения.
5. Тестирование. Проверка приложения на соответствие с требованиями в ТЗ.
6. Заключение. Результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.
7. Список использованных источников.
8. Приложения. Фрагменты исходного кода приложения.

# Аналитический обзор литературы

# Аналитический обзор источников

В ходе подготовки пояснительной записки была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Общие теоретические и практические знания были получены из книги «JavaScript. Шаблоны» и «jQuery. Подробное руководство по продвинутому JavaScript». В книгах приведены примеры работы с языком JavaScript, введены основные понятия для работы с JavaScript [1].

Принцип создания страниц был взят из статьи «Делаем веб приложение на Node.js». В статье были рассмотрены алгоритм работы веб приложения, подключение к базе данных [2].

Принцип работы с MongoDB были получены из официальной документации по mongoose [3].

# Обзор аналогов

Для создания принципиально нового решения в виде программного продукта для решения вопросов автоматизации деятельности интернет магазина необходимо ознакомиться с существующими аналогами в данной сфере. Анализ достоинств и недостатков этих аналогов позволит сформировать требования к проектируемому программному средству, учитывающие опыт существующих разработок и внести в них улучшения или изменения. Источником информации послужили электронные базы в сети Интернет.

В результате поиска были обнаружены следующие ресурсы:

* Vc. Как пишут сами о себе — «крупнейшая в рунете площадка для предпринимателей нового поколения. Мы пишем о стартапах, растущих бизнесах, новых моделях заработка, неординарных героях со всего мира и технологиях роста» [4].

С прошлого года активно использует UGC. Есть раздел [«Блоги»](https://vc.ru/blog), где каждый желающий может завести свой блог и публиковать материалы. Лучшие из них редакция Vc выводит на главную страницу ресурса. Самые резонансные публикации собирают тысячи просмотров и сотни комментариев.

Инструкция от главного редактора сайта Константина Панфилова [«Как написать хорошую статью в блоги на vc.ru»](https://vc.ru/26741-blogs) вам в помощь.

Периодически проводятся конкурсы публикаций, за которыми имеет смысл следить. Вот, например, сейчас идет [конкурс на лучшее интервью с предпринимателем](https://vc.ru/36671-na-vsyu-golovu), который продлится до 15 июня. Берете интервью с любым предпринимателем, размещаете в блогах, получаете публикацию и, возможно, еще и приз, если победите в конкурсе.

* Habr. Бывший Habrahabr. Крупнейший ресурс для ИТ-специалистов. Здесь публикуются кейсы, инструкции, новости отрасли. Структура сайта может показаться сложной для тех, кто только хочет завести здесь свой блог. Большое значение имеют оценки публикаций пользователями, рейтинги и т. д. Прежде чем начать работать с ресурсом, обязательно ознакомьтесь с [требованиями к публикациям](https://habr.com/info/help/posts/), [правилами сайта](https://habr.com/info/help/rules/), а также почитайте о том, ч[то такое карма, рейтинг](https://habr.com/info/help/karma/) и на что они влияют [5].
* Spark. Блог-платформа для общения малого бизнеса из технологического сегмента. Одно время сайт несколько «сдулся» — его продвижением никто не занимался, в итоге авторы, которые вели здесь свои блоги, постепенно их забросили. Но в ноябре 2018-го ИД «Комитет» продал ресурс разработчику софта «Интеллектуальные облачные решения». В проект взяли [новую команду](https://spark.ru/startup/spark/blog/43235/bolshie-novosti-ot-proekta-spark), и теперь Spark потихоньку возрождается, авторы возвращаются. Вот [здесь](https://spark.ru/startup/spark/blog/43902/faq-chto-takoe-spark) можно посмотреть основные требования — о чем и как писать в блоге на Spark, правила оформления и т. д. Тут же команда сайта выкладывает новости про последние обновления платформы [6].

# Проектирование архитектуры

Архитектура проекта – это его строение как оно видно (или должно быть видно) извне его, т.е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью программного средства, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры проекта:

– Выделение программных подсистем и отображение на них внешних функций (заданных по внешнему описанию) программного средства;

– Определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

# 2.1. Обобщенная структура

Курсовой проект построен на архитектуре MVC. Для данного курсового проекта были разработаны следующие уровни:

* уровень «Модель» – описание таблиц в базе данных;
* уровень «Представление» – отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* уровень «Контроллер» – описание классов, методы которых предназначены для создания ответа пользователю по его запросу.

Каждый запрос клиента проходит как минимум через уровень контроллера, после чего клиент получает ответ.

# 2.2. Взаимосвязь всех компонентов

В решении курсового проекта была использована MVC архитектура, которая представлена на рисунке 2.1. Эта архитектура выбрана, так как она идеально подходит для web-приложений.

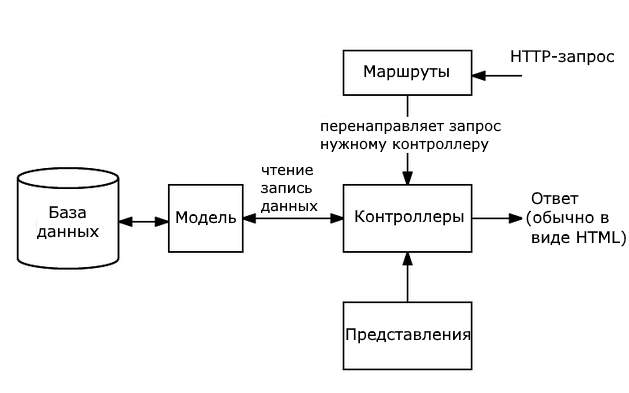


Рисунок 2.1 – общая схема MVC архитектуры

Данный курсовой проект построен по примеру выше приведенной схемы.

1. **Разработка** **функциональной модели и модели данных ПС**

# Проектирование базы данных

В данном курсовом проекте была спроектирована база данных используя mongoose. Mongoose представляет специальную ODM-библиотеку (Object Data Modelling) для работы с MongoDB, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных. Грубо говоря, Mongoose работает подобно инструментам ORM.

Созданы 5 таблиц (коллекций) для работы приложения:

* comments. Таблица для работы с комментариями постов;
* posts. Таблица для работы с постами;
* sessions. Таблица для работы с сессиями;
* uploads. Таблица для работы с загружаемыми картинками пользователем;
* users. Таблица для работы с пользователями. Авторизация, регистрация.

# 3.2. Проектирование архитектуры проекта

На рисунке 3.2 изображена диаграмма вариантов использования данного приложения.

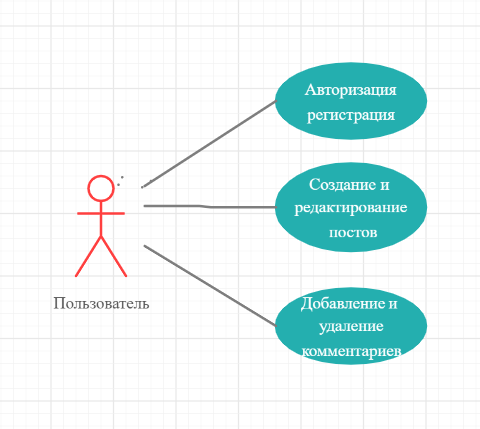


Рисунок 3.2 – диаграмма использования

На рисунке 3.3 была спроектирована диаграмма последовательности для авторизации и регистрации. На диаграмме последовательности отображаются только те объекты, которые непосредственно принимают участие во взаимодействии.

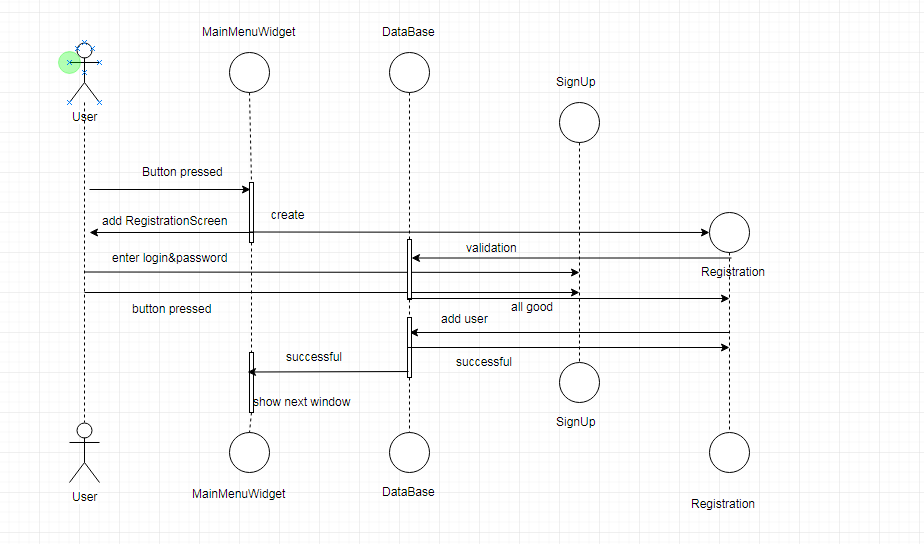


Рисунок 3.3 – диаграмма последовательности для авторизации и регистрации.

На рисунках 3.4 и 3.5 представлены блок-схемы для алгоритмов авторизации и регистрации пользователя.



Рисунок 3.4 – схема алгоритма авторизации



Рисунок 3.5 – схема алгоритма регистрации

# Создание программного средства

Для более детального изучения программного средства в данной главе остановимся поподробнее на некоторых фреймворках и технологиях, которые использовались в приложении.

# 4.1. EJS

Весь графический интерфейс разработан используя движок EJS. Этот движок использует синтаксис, который во многом был заимствован из движка представлений Web Forms на платформе ASP.NET, который разрабатывался в компании Microsoft. В приложении А представлен код на движке EJS, который отображается на главной странице.

# 4.2. Express

Express - это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Express – ключевой фреймворк, который отвечает за бэкэнд часть приложения. В приложении Б представлен код, который и реализует работу express сервера.

# 4.3. Gulp

Gulp — это таск-менеджер для автоматического выполнения часто используемых задач, написанный на языке программирования JavaScript. В данное проекте Gulp выполняет следующие задачи:

* «склеивание» всех js-файлов в один js файлов, удаляя лишние пробелы, переносы строк;
* генерация из scss файлов css файла;
* сжатие css файла.

В приложении В представлен код gulp-файла.

# 4.4. ESLint и nodemon

ESLint - это инструмент статического анализа кода для выявления проблемных шаблонов, обнаруженных в коде JavaScript. Проще говоря, это инструмент, который позволяет проводить анализ качества вашего кода, написанного на любом выбранном стандарте JavaScript. Он приводит код к более-менее единому стилю, помогает избежать глупых ошибок, умеет автоматически исправлять многие из найденных проблем и отлично интегрируется со многими инструментами разработки. Кстати, он, как и другие линтеры, не обязывает вас к одному какому-то конкретному стилю. Наоборот — вы можете выбрать что-то из лучших практик и доработать по своему усмотрению. В приложении Г представлен код настройки ESLint.

Nodemon – нужен для удобной разработки, чтобы не перезапускать проект после изменений файлов. Nodemon делает это сам.

# 4.5. Sharp

Sharp используется для изменения размеров изображений, которые пользователь может добавить в свой пост. Ниже представлена ключевая функция, которая это реализует.

const resizer =  
 Sharp()  
 .resize(1024, 768, {fit: 'inside', withoutEnlargement: true})  
 .toFormat('jpg')  
 .jpeg({  
 quality: 40,  
 progressive: true  
 });

На этом с обзором некоторых ключевых технологий закончим и перейдем к тестированию приложения.

# Тестирование

Для автоматизации тестирования было принято решение создать отдельный проект, назначением которого будут являться API-тесты. В качестве тестового фреймворка был выбран Jest. В качестве модуля, который будет выполнять запросы к нашему серверу был выбран axios. Для проверки ответов сервера были добавлены json-схемы, а также AJV (Another JSON Schema Validator). Результат работы проекта, а также тестирования части авторизации и регистрации представлен на рисунке 5.1. В приложении Д представлен код одной из JSON-схем, используемых для проверки ответа сервера.

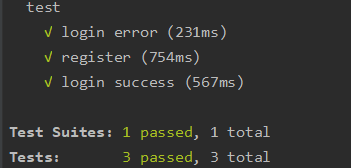


Рисунок 5.1 – результат тестирования auth-части

Далее приступим к более привычному для тестирования способу, а именно мануальному тестированию.

# Посты

Начнём с ключевой части нашего приложения – работа с постами. Приступим к негативным тестам. Попробуем оставить поля не заполненными и опубликуем пост. На рисунке 5.2 представлен результат этих действий, а именно ошибки, которая не позволяет пользователю добавить пост. А на рисунке 5.3 результат добавления картинки, которая не является картинкой (проверка формата файла).

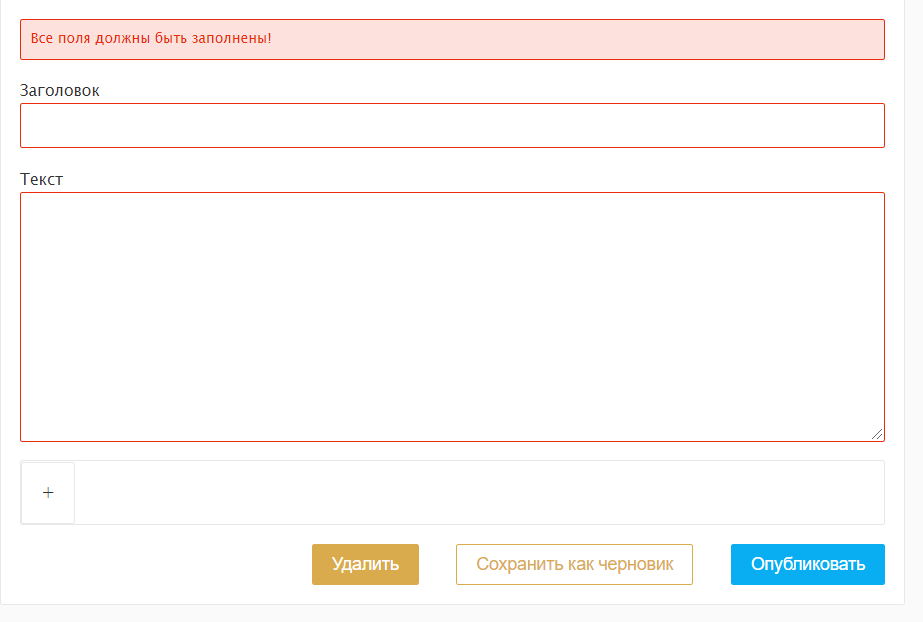


Рисунок 5.2 – поля не заполнены

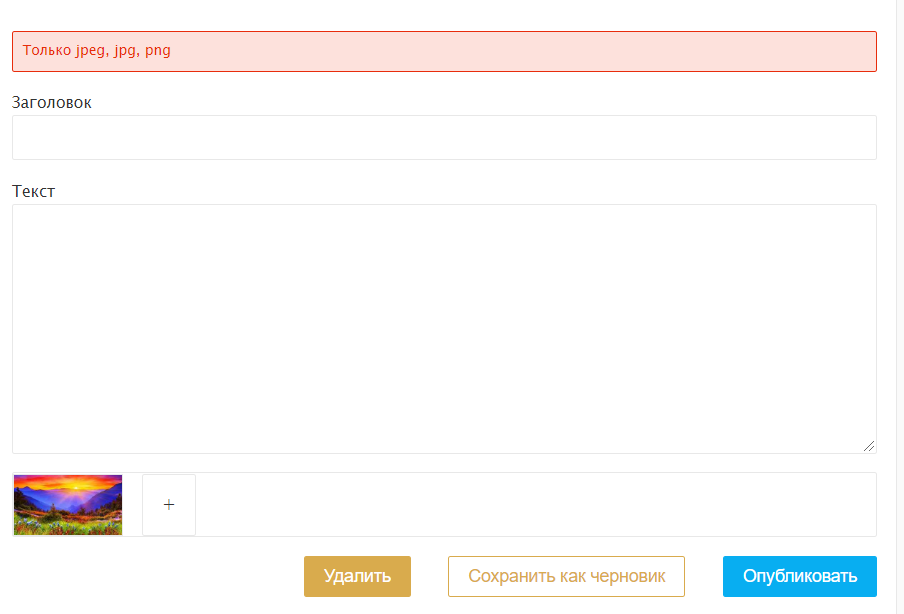


Рисунок 5.3 – формат изображения

Как видим, у нас всё хорошо работает. Далее перейдём к позитивным тестам. Картинку мы уже добавили выше к посту, теперь добавим её в сам пост, нажатием на неё. Появляется разметка markdown, что видно на рисунке 5.4, а на рисунке 5.5 видно отображение этой картинки, когда уже опубликовали пост в ленте.

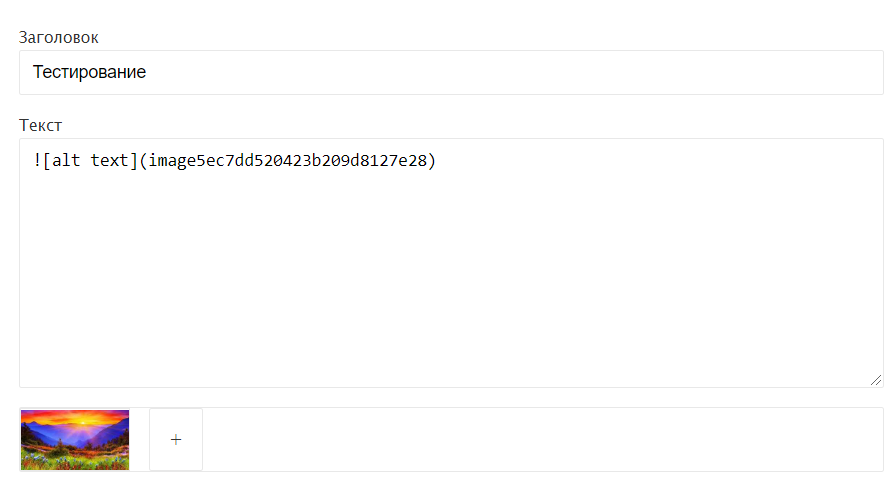


Рисунок 5.4 – картинка в markdown

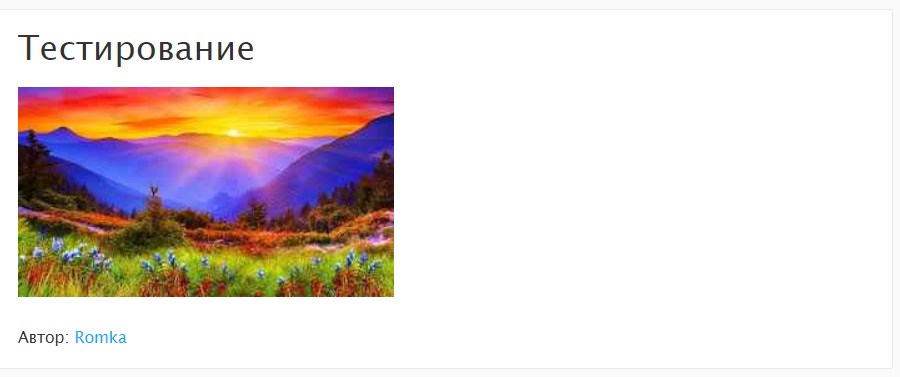


Рисунок 5.5 – отображение картинки

Также можно увидеть, что и автор отображается у нас корректно со ссылкой на автора, где можно посмотреть все посты автора. Страница пользователя представлена на рисунке 5.6.

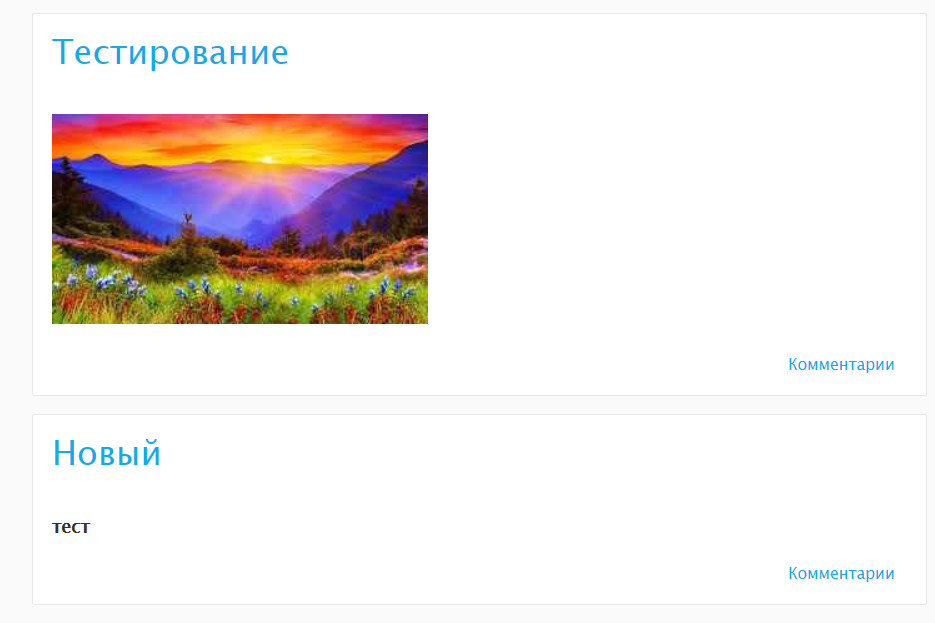


Рисунок 5.6 – страница пользователя

Далее проверим функцию редактирования поста. Добавим к нашей картинке какой-нибудь текст и сохраним. На рисунке 5.7 видно, что пост отредактирован, а также время последнего обновления, которое отображается на главной странице.

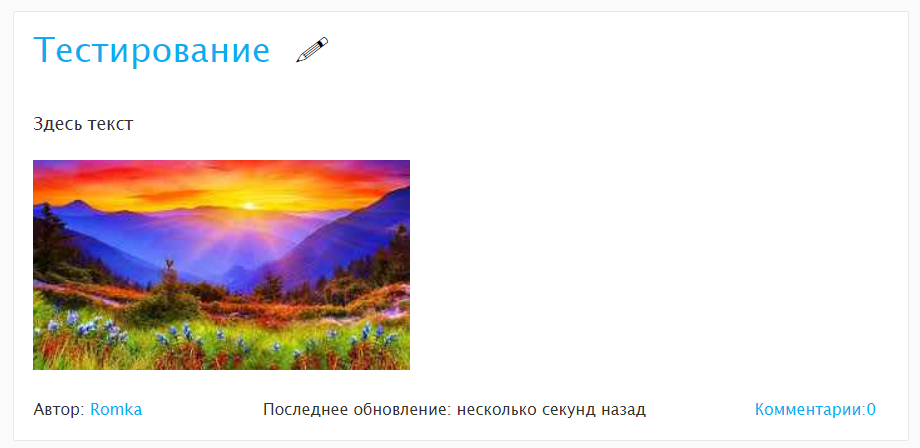


Рисунок 5.7 – редактирование поста

Теперь добавим новый пост, только теперь его сохраним как черновик. Для этого нажмем соответствующую кнопку. В результате видим на рисунке 5.8, что поста нет на главной странице. Однако если мы нажмем «добавить пост», то на рисунке 5.9 увидим наш заготовленный шаблон.

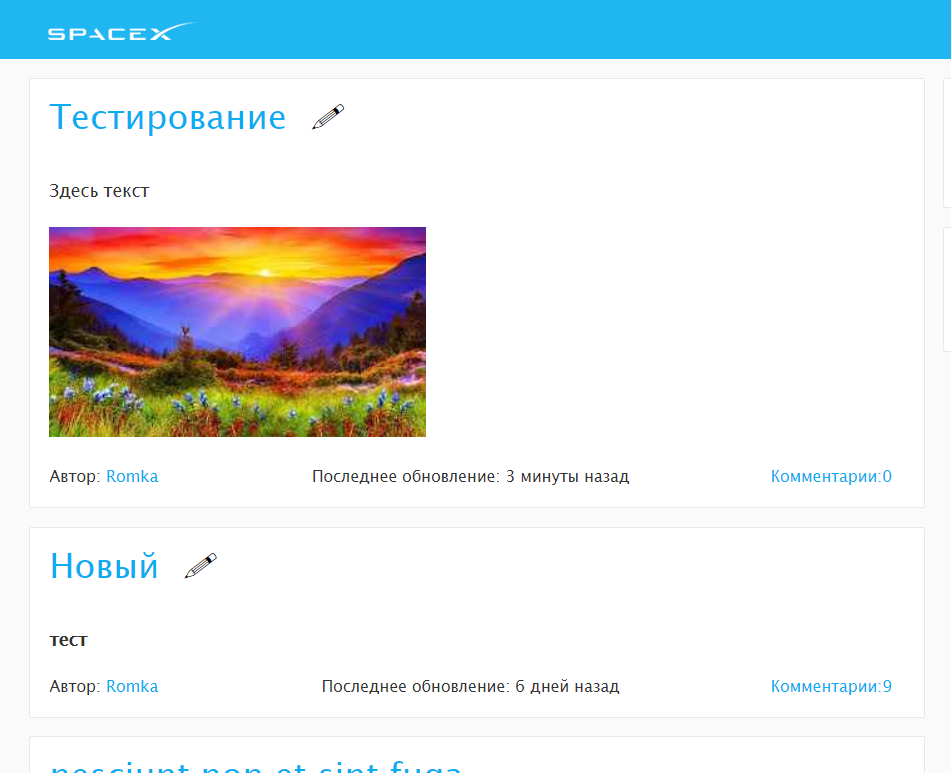


Рисунок 5.8 – главная страница

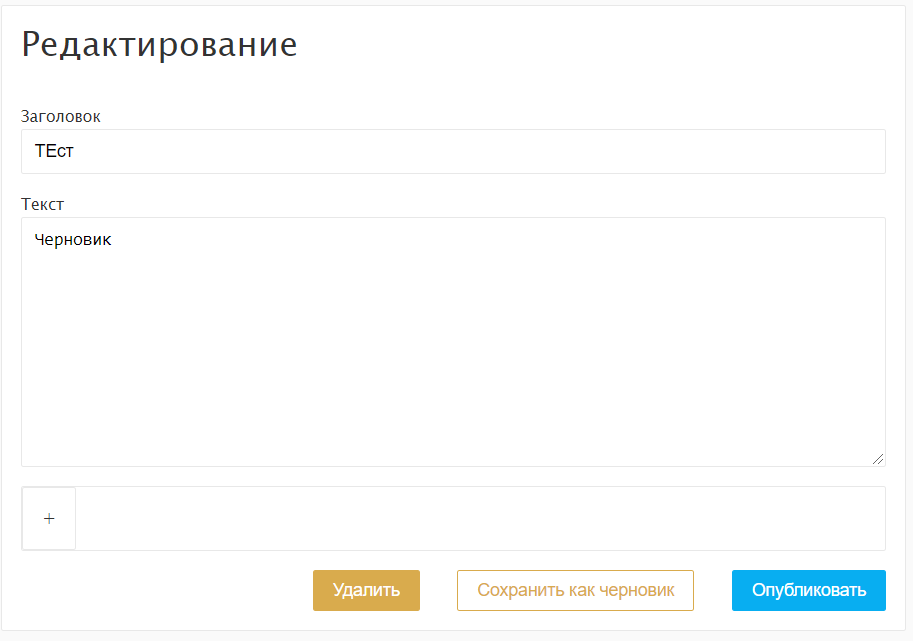


Рисунок 5.9 – сохраненный ранее шаблон

После публикации шаблона удалим его и убедимся в отсутствии поста. Так и есть. Пост отсутствует на главной странице.

# Комментарии

Следующая часть приложения, которую мы протестируем, будет комментарии и всё, что с ними связано. На рисунке 5.10 мы видим, что количество комментариев у поста ноль.

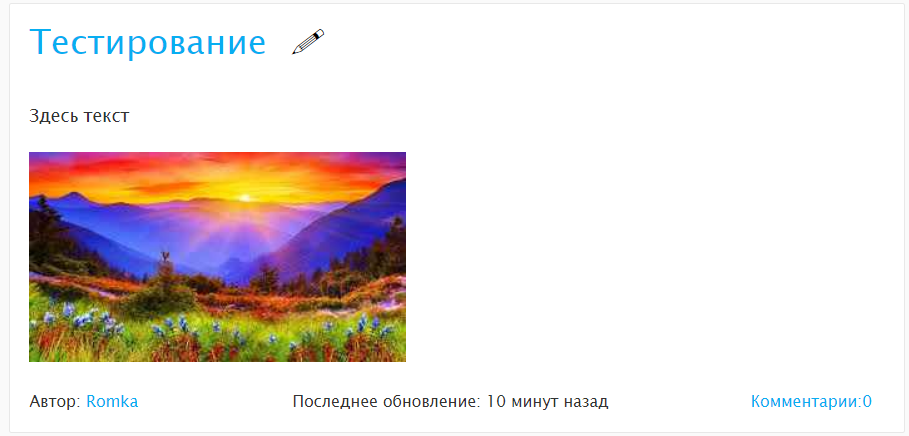


Рисунок 5.10 – количество комментариев

Далее перейдем на страницу поста и попробуем добавить пустой комментарий, в результате видим на рисунке 5.11, что возникла ошибка.

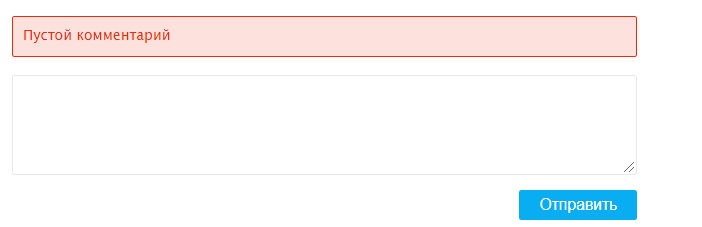


Рисунок 5.11 – пустой комментарий

Далее добавим уже какой-нибудь комментарий и увидим на рисунке 5.12, что комментарий успешно добавился, а на рисунке 5.13, что изменилось количество комментариев.

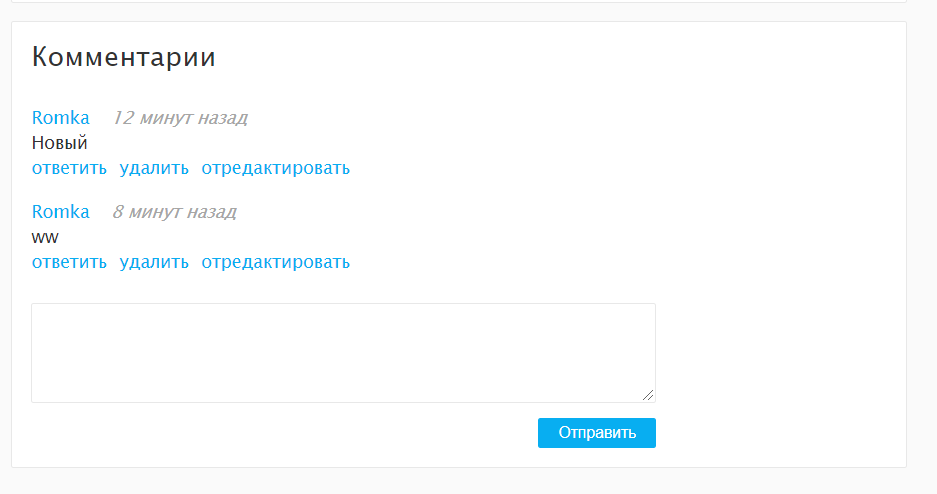


Рисунок 5.12 – комментарии к посту

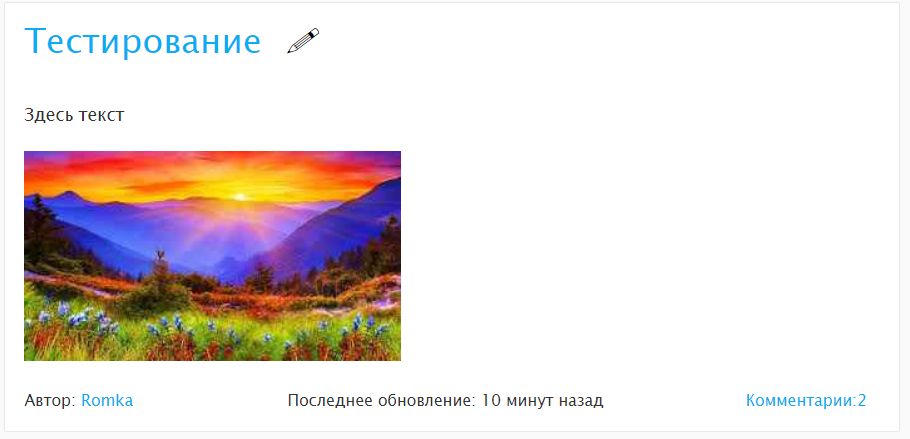


Рисунок 5.13 – измененное количество комментариев

Далее добавим дочерний комментарий и проверим его отображение. Для этого нажмем на кнопку «ответить». После отправки ответа на комментарий, ответ корректно отображается, заодно проверим редактирование комментария. Оно тоже корректно работает, что видно на рисунке 5.14.

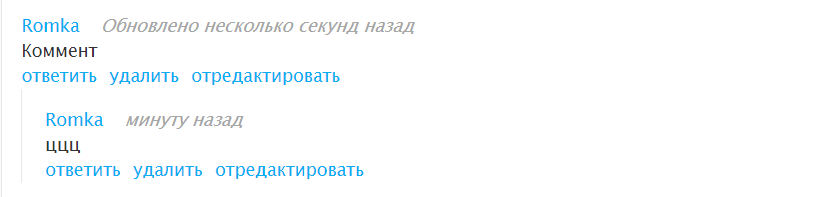


Рисунок 5.14 – ответ и редактирование комментария.

Теперь удалим комментарий, который имеет дочерний комментарий. Как видим на рисунке 5.15, комментарий бесследно не пропал, дабы остался ответ пользователя, а также на этот комментарий больше нельзя ответить и отредактировать. А на рисунке 5.16 представлен результат удаления комментария, который не имеет дочерних. В результате видим, что наши компоненты работают корректно.

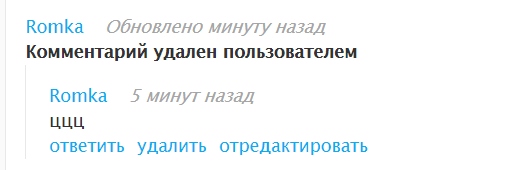


Рисунок 5.15 – удаление комментария с дочерними

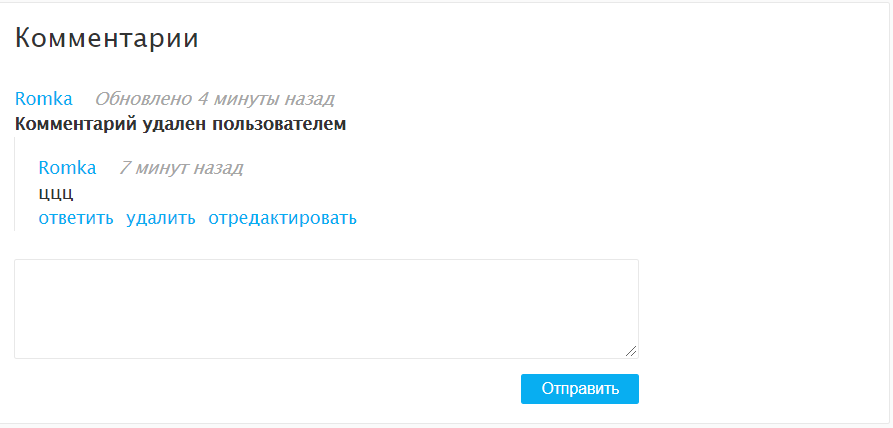


Рисунок 5.16 – удаление комментария без дочерних

# Руководство пользователя

При открытии главной странице сайта пользователь оказывается на стартовой странице, где видны все посты, а также присутствует форма регистрации и авторизации, которую можно переключать. Вид главной страницы представлен на рисунке 6.1.

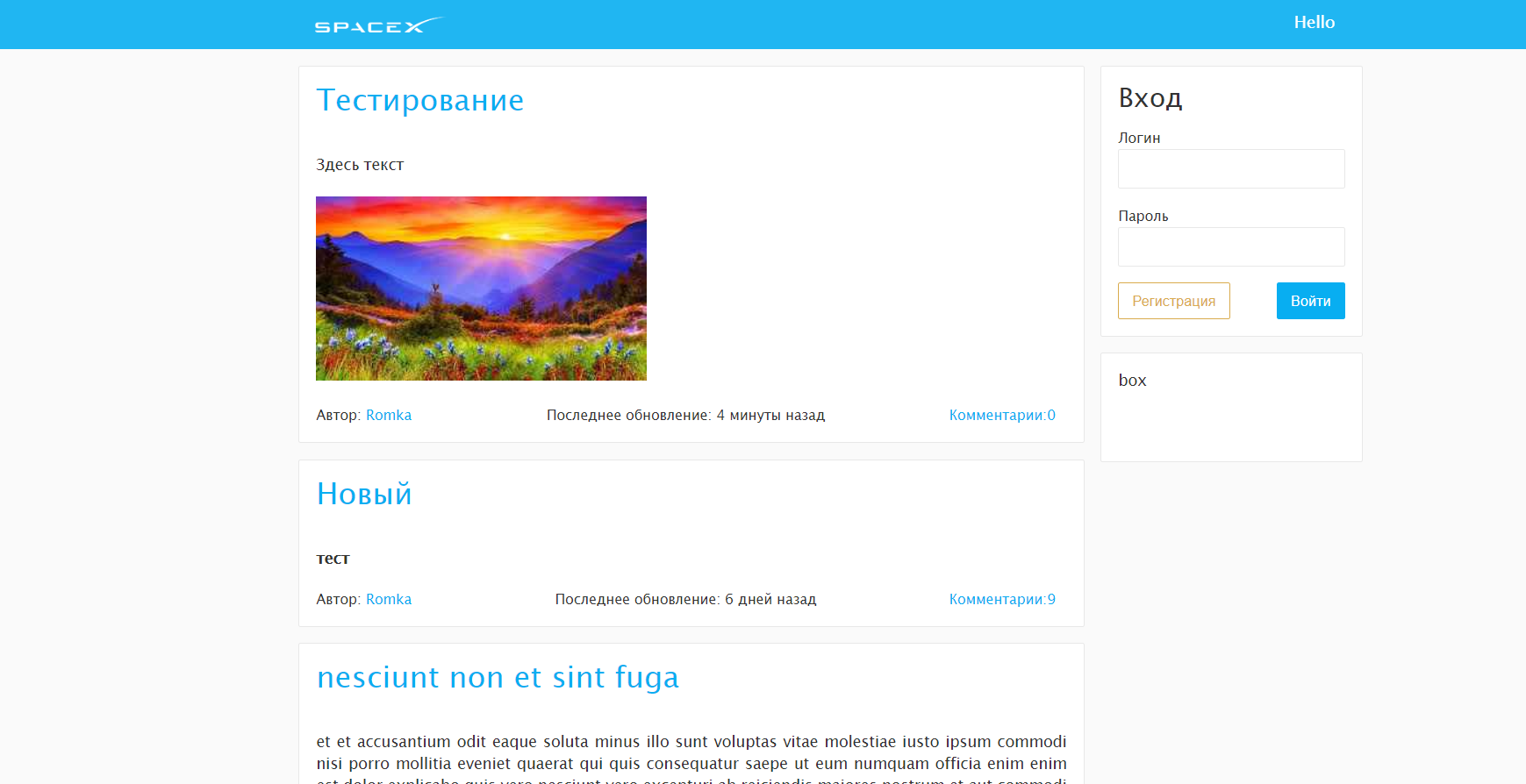


Рисунок 6.1 – главная страница

Для того чтобы пользоваться всеми возможностями сайта, пользователю необходимо зарегистрировать аккаунт. Зарегистрируем пользователя manual с паролем manual. После регистрации вместо формы регистрации у нас отображается блок, представленный на рисунке 6.2.

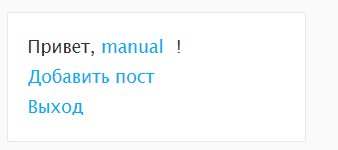


Рисунок 6.2 – блок пользователя

В этом блоке пользователь может перейти на свою страницу, где может увидеть все свои посты, нажать на кнопку добавления поста для перехода на страницу поста, а также нажать кнопку выход, если пользователь хочет войти под другими данными.

Рассмотрим страницу, где пользователь может добавить свой пост на рисунке 6.3.

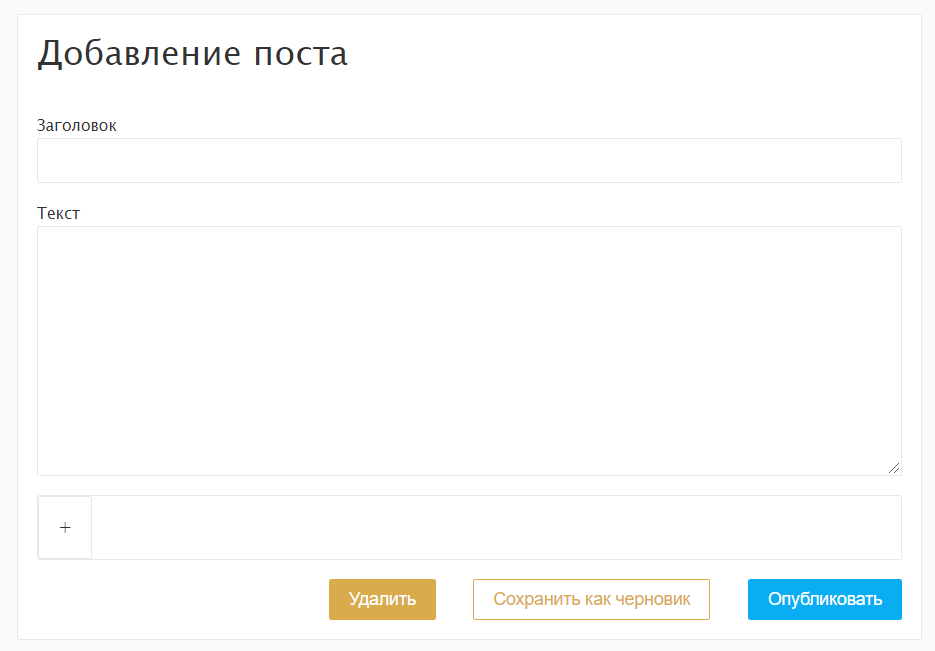


Рисунок 6.3 – добавление поста

На этой странице пользователь может разместить заголовок и текст для своего будущего поста, а так же при необходимости добавить картинку к своему посту. Нажав на кнопку сохранить как черновик, пост сохраняется, но не отображается на главной странице. Нажав кнопку «опубликовать» пост публикуется и появляется на главной странице, а также пользователю становится доступно редактирование поста, что видно на рисунке 6.4.

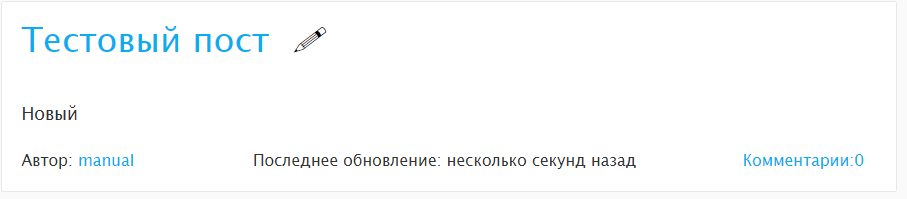


Рисунок 6.4 – пост на главной странице

На рисунке 6.5 изображена страница поста, где у авторизованного пользователя есть возможность добавить комментарий к посту, ответить на другие комментарии, отредактировать или удалить свой комментарий.

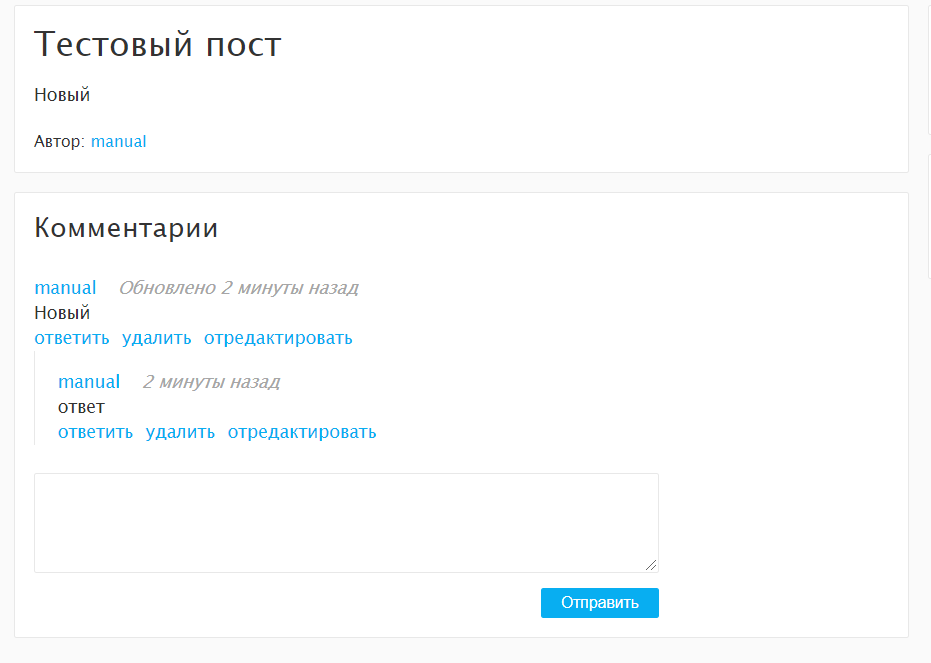


Рисунок 6.5 – страница поста

# Заключение

В самом начале была поставлена задача: разработать web-приложение на определенную тему, в данной работе тематикой стал некий портал, где пользователи могут делиться своими мыслями, рассказывать о чем-либо. Перед началом были изучены некоторые аспекты JavaScript, Express.js, jQuery, SCSS и т.д., которые и позволили реализовать то, что было задумано изначально.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию приложения «SpaceX». При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения. Таким образом, была достигнута цель и создан портал.

В программном средстве были реализованы следующие функции:

* авторизация и регистрация;
* добавление постов;
* удаление и редактирование постов;
* добавление картинок к своим постам;
* комментирование постов;
* редактирование и удаление комментариев.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# Список литературы

[1] Стоян Стефанов “JavaScript. Шаблоны” – Символ, 2011 – 265 с.

[2] Делаем веб приложение на Node.js [Электронный ресурс]. – Режим доступа: <https://nodeguide.ru/doc/dailyjs-nodepad/> – Дата доступа: 22.05.2020.

[3] Mongoose [Электронный ресурс]. – Режим доступа: <https://mongoosejs.com/docs/api/mongoose.html> – Дата доступа: 22.05.2020.

[4] Vc [Электронный ресурс]. – Режим доступа: <https://vc.ru/> – Дата доступа: 22.05.2020.

[5] Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/> – Дата доступа: 22.05.2020.

[6] Spark [Электронный ресурс]. – Режим доступа: <https://spark.ru/> – Дата доступа: 22.05.2020.

# ПРИЛОЖЕНИЕ А

<%- include ('../layout/header.ejs') %>  
  
<div class="main">  
 <div class="container">  
 <div class="content">  
 <% for(var ***i*** = 0; ***i*** < posts.length; ***i***++){ %>  
 <div class="post">  
 <div class="top">  
 <h2>  
 <a href="/posts/<%= posts[***i***].url %>">  
 <%= posts[***i***].title %>  
 </a>  
 <% if (posts[***i***].owner.login === user.login) { %>  
 <a style="color: black; text-decoration: none"  
 href="/post/edit/<%= posts[***i***].id %>">🖉</a>  
 <% } %>  
 </h2>  
 </div>  
 <div class="body">  
 <p>  
 <%- posts[***i***].body %>  
 </p>  
 </div>  
 <div class="bottom">  
 <div class="author">  
 Автор:  
 <a href="/users/<%= posts[***i***].owner.login %>"><%= posts[***i***].owner.login %></a>  
 </div>  
 <div>  
 Последнее обновление:  
 <%= moment(posts[***i***].updatedAt).fromNow(); %></div>  
 <a href="/posts/<%= posts[***i***].url %>#<%= posts[***i***].id %>">Комментарии:<%= posts[***i***].commentCount %> </a>  
 </div>  
 </div>  
 <% } %>  
 <div class="pagination">  
 <% if (current > 2){ %>  
 <a class="new" href="/archive/<%= ***Number***(current) - 1 %>">← К новым</a>  
 <% } %>  
 <% if (current == 2){ %>  
 <a class="new" href="/">← На главную</a>  
 <% } %>  
 <% if (pages > 0 && current < pages){ %>  
 <a class="old" href="/archive/<%= ***Number***(current) + 1 %>">К старым →</a>  
 <% } %>  
 </div>  
 </div>  
 <%- include ('../layout/sidebar.ejs') %>  
 </div>  
</div>  
  
<%- include ('../layout/footer.ejs') %>

# ПРИЛОЖЕНИЕ Б

const express = require('express');  
const ***https*** = require('https');  
const fs = require('fs');  
const bodyParser = require('body-parser');  
const ***path*** = require('path');  
const staticAsset = require('static-asset');  
const mongoose = require('mongoose');  
const session = require('express-session');  
const MongoStore = require('connect-mongo')(session);  
  
require('./webSocket');  
  
const config = require('./config');  
const routes = require('./routes');  
  
const cert = {  
 // Replace private key and cert with the appropriate names of the credentials you use  
 key: fs.readFileSync('./certificates/key.pem', 'utf8'),  
 cert: fs.readFileSync('./certificates/cert.pem', 'utf8')  
};  
  
//Database  
mongoose.Promise = ***global***.Promise;  
mongoose.set('debug', config.IS\_PRODUCTION);  
mongoose.connection  
 .on('error', error => ***console***.log(error))  
 .on('close', () => ***console***.log('Database connection closed.'))  
 .once('open', () => {  
 const info = mongoose.connections[0];  
 ***console***.log(`Connected to ${info.host}:${info.port}/${info.name}`);  
 //require('./mocks')();  
 });  
mongoose.connect(config.MONGO\_URL, {useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true, useFindAndModify: false});  
  
//Express  
const ***app*** = express();  
// sessions  
  
***app***.use(  
 session({  
 secret: config.SESSION\_SECRET,  
 resave: true,  
 saveUninitialized: false,  
 store: new MongoStore({  
 mongooseConnection: mongoose.connection  
 })  
 })  
);  
//Sets and uses  
***app***.set('view engine', 'ejs');  
***app***.use(bodyParser.urlencoded({extended: true}));  
***app***.use(bodyParser.json());  
***app***.use(staticAsset(***path***.join(\_\_dirname, 'public')));  
***app***.use(express.static(***path***.join(\_\_dirname, 'public')));  
***app***.use(  
 `/${config.UPLOADS\_ROUTE}`,  
 express.static(***path***.join(\_\_dirname, config.DESTINATION))  
);  
***app***.use(  
 '/javascripts',  
 express.static(***path***.join(\_\_dirname, 'node\_modules', 'jquery', 'dist')));  
  
//routers  
  
***app***.use('/api/auth', routes.***auth***);  
***app***.use('/post', routes.***post***);  
***app***.use('/', routes.***archive***);  
***app***.use('/comment', routes.***comment***);  
***app***.use('/upload', routes.***upload***);  
  
// catch 404  
***app***.use((req, res, next) => {  
 const err = new ***Error***('Not found');  
 err.status = 404;  
 next(err);  
});  
  
//error handler  
// eslint-disable-next-line no-unused-vars  
***app***.use((error, req, res, next) => {  
 res.status(error.status || 500);  
 res.render('error', {  
 message: error.message,  
 error: !config.IS\_PRODUCTION ? error : {}  
 });  
});  
const httpsServer = ***https***.createServer(cert, ***app***);  
httpsServer.listen(config.PORT, () => ***console***.log(`App listening on port ${config.PORT}`));  
  
module.exports = ***app***;

# ПРИЛОЖЕНИЕ В

/\* eslint-disable node/no-unpublished-require \*/  
const ***gulp*** = require('gulp');  
const sass = require('gulp-sass');  
const autoprefixer = require('gulp-autoprefixer');  
const cssnano = require('gulp-cssnano');  
const plumber = require('gulp-plumber');  
const concat = require('gulp-concat');  
const uglify = require('gulp-uglifyjs');  
/\* eslint-enable node/no-unpublished-require \*/  
  
***gulp***.task('scss', () => {  
 return ***gulp*** .src('dev/scss/\*\*/\*.scss')  
 .pipe(plumber())  
 .pipe(sass())  
 .pipe(  
 autoprefixer(['last 15 versions', '> 1%', 'ie 8', 'ie 7'], {  
 cascade: true  
 })  
 )  
 .pipe(cssnano())  
 .pipe(***gulp***.dest('public/stylesheets'));  
});  
  
***gulp***.task('scripts', () =>  
 ***gulp*** .src([  
 'dev/js/\*.js',  
 //  
 ])  
 .pipe(concat('scripts.js'))  
 .pipe(uglify())  
 .pipe(***gulp***.dest('public/javascripts'))  
);  
  
***gulp***.task('watch', () => {  
 ***gulp***.watch('dev/scss/\*\*/\*.scss', ***gulp***.series('scss'));  
 ***gulp***.watch('dev/js/\*\*/\*.js', ***gulp***.series('scripts'));  
});  
  
***gulp***.task('default', ***gulp***.parallel('scss', 'scripts', 'watch'), () => {  
  
});

# ПРИЛОЖЕНИЕ Г

{  
 "plugins": ["node"],  
 "extends": [  
 "eslint:recommended",  
 "plugin:node/recommended",  
 "prettier"  
 ],  
 "parserOptions": {  
 "ecmaVersion": 2020  
 },  
 "rules": {  
 "node/exports-style": [  
 "error",  
 "module.exports"  
 ],  
 "node/file-extension-in-import": [  
 "error",  
 "always"  
 ],  
 "node/prefer-global/buffer": [  
 "error",  
 "always"  
 ],  
 "node/prefer-global/console": [  
 "error",  
 "always"  
 ],  
 "node/prefer-global/process": [  
 "error",  
 "always"  
 ],  
 "node/prefer-global/url-search-params": [  
 "error",  
 "always"  
 ],  
 "node/prefer-global/url": [  
 "error",  
 "always"  
 ],  
 "node/prefer-promises/dns": "error",  
 "node/prefer-promises/fs": "error"  
 }  
}

# ПРИЛОЖЕНИЕ Д

{  
 "definitions": {},  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "$id": "schemaRegister",  
 "type": "object",  
 "title": "Register user",  
 "required": [  
 "ok",  
 "user"  
 ],  
 "properties": {  
 "ok": {  
 "$id": "#/properties/ok",  
 "type": "boolean",  
 "title": "Status"  
 },  
 "user": {  
 "$id": "#/properties/user",  
 "type": "object",  
 "title": "User obj",  
 "items": {  
 "$id": "#/properties/data/items",  
 "type": "object",  
 "title": "The Items Schema",  
 "required": [  
 "\_id",  
 "login",  
 "password",  
 "createdAt",  
 "updatedAt",  
 "\_\_v",  
 "id"  
 ]  
 }  
 }  
 }  
}