МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологи

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка Web-приложения «Хабр»

Выполнил студент Муха А.А.

(Ф.И.О.)

Руководитель проекта асс. Дубовик 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.)

Курсовой проект защищен с оценкой

Минск 2021

**Реферат**

Пояснительная записка курсового проекта содержит 50 страниц пояснительной записки 45 иллюстраций, 4 источника литературы, 1 приложение.

JAVASCRIPT, NODE.JS, EXPRESS.JS, SQLITE, MVC.

На сегодняшний день новостные ленты по типу “Хабра” являются важнейшим средством донесения технической информации, своевременно обеспечивающим этой информацией большую часть людей. Они представляют услуги, которые немыслимы без применения существующих веб-приложений. Эти приложения позволяют упорядочить и избежать многих ошибок, свойственных человеку. В настоящее время услугами подобных новостных лент пользуются все ведущие страны. Количество информации проходящих, через работников достигает огромных масштабов, потому так актуальным и необходимым является предоставить автоматизированный процесс управлением материалов.

В соответствии с заданием курсового проекта следует разработать web-приложение способное хранить данные о материалах и пользователях, комментарии с использованием программных продуктов SQLite, NodeJS, Express, MVC. А интерфейс программы будет представлен web-приложением, написанным с помощью Node.JS.

В первой главе проводится аналитический обзор литературы по тематике курсового проекта и постановку задачи, и некоторые сравнения с аналогами.

Вторая глава посвящена процессу проектирования приложения.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвёртой главе описаны руководства для существующих в веб-приложении групп пользователей.

В пятой главе описаны результаты тестирования.

В заключении приведены результаты проделанной работы.

**Содержание**

[Введение 4](#_Toc42280158)

[1. Постановка задачи 5](#_Toc42280159)

[1.1 Алгоритмы решения 5](#_Toc42280160)

[1.2 Обзор прототипов 5](#_Toc42280161)

[1.3 Актуальность задачи 7](#_Toc42280162)

[2. Проектирование веб-приложения 9](#_Toc42280163)

[2.1 Архитектура проекта 9](#_Toc42280164)

[2.2 Общая структура проекта 9](#_Toc42280165)

[2.3 Схемы и описание работы веб-приложения 11](#_Toc42280166)

[2.4 Модель базы данных 14](#_Toc42280167)

[2.5 Использование ORM 16](#_Toc42280168)

[3. Разработка веб-приложения 17](#_Toc42280169)

[3.1 Физическая структура и функции модулей 17](#_Toc42280170)

[3.2 Реализация хранения и хеширования пароля 20](#_Toc42280171)

[3.3 Реализация пагинации 21](#_Toc42280172)

[3.4 Реализация уведомления на сообщения 23](#_Toc42280173)

[4. Руководство пользователя 26](#_Toc42280174)

[4.1 Программно-аппаратные ресурсы ПК 26](#_Toc42280175)

[4.2 Руководство по запуску веб-приложения 26](#_Toc42280176)

[4.3 Руководство для гостя 26](#_Toc42280177)

[4.4 Руководство для авторизированного пользователя 28](#_Toc42280178)

[4.5 Руководство для администратора 34](#_Toc42280179)

[5. Тестирование, экспериментальные исследования и анализ полученных результатов 38](#_Toc42280180)

[5.1 Тестирование выводимых системой сообщений 38](#_Toc42280181)

[5.2 Тестирование производительности на больших объемах данных 39](#_Toc42280182)

[5.3 Анализ 42](#_Toc42280183)

[Заключение 44](#_Toc42280184)

[Список литературы 45](#_Toc42280185)

[Приложение 46](#_Toc42280186)

[Листинг моделей 46](#_Toc42280187)

# Введение

Развитие технологий привнесло ощутимые изменения в области развития новостных лент.

В настоящее время реализовано много современных компьютерных приложений, которые предоставляют удобный интерфейс для работы с данными в отрасли статей, изданий и администрирования новостных лент.

Однако, с учетом глобализации и урбанизации, а также активным развитием рыночной экономики, спрос на web-приложения новостных лент не убывает. Тяга к информации о происходящем в мире стала настоящей актуальной проблемой, а веб-приложения новостных лент упрощают задачу привлечения клиентов легким доступом к информации, а также облегчает администрирование и учет статей.

С учетом вышеописанных факторов, целью своего курсового проекта мною была выбрана разработка web-приложения – новостной ленты, функционал которой будет давать возможность вести статьи, управлять ими, позволить пользователям общаться друг с другом а также возможность участия читателей в виде комментирования.

В основе разработки курсового проекта лежит удобство пользования и изменения базы данных, наглядность изменений и статусов статей, простота интерфейса, понятная человеку, напрямую не связанному с информационными технологиями.

В записке далее представлена информация о аналитическом разборе литературы по теме данного курсового проекта с целью формирования требований к программному продукту, также будет рассмотрен анализ существующих на рынке продуктов, аналогичных заданному, с целью оптимизации курсового проекта. В последующих главах описывается функционал моей новостной ленты, а также тестирование его работоспособности. Завершается записка кратким руководством пользователя для быстрого и эффективного понимания возможностей программы и работы с ней.

# Постановка задачи

## 1.1 Алгоритмы решения

Постановка задачи – это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются функциональные требования, определяющие функции, которые должно выполнять программное обеспечение, и эксплуатационные требования, определяющие характеристики его функционирования. Один из этапов построение задачи это формирование диаграммы использования продукта.

Сначала будет проведено базовое сравнение с аналогами на рынке, определена актуальность задачи. Затем будет проведено проектирование приложения и определена модель базы данных. На основе спроектированной и разработанной модели базы данных будет построен процесс разработки веб-приложения, по окончанию которого будет приведено краткое руководство пользователя с тестированием некоторых основных функций приложения.

## 1.2 Обзор прототипов

На данный момент, сфера подобных новостных лент наполнена многочисленными веб-приложениями, различными по своему исполнению, но имеющими общий основной интерфейс и спектр услуг.

Рассмотрим 3 наиболее интересных и популярных решения в области новостных лент типа “Хабр”:

Первый вариант, который собственно и является сайтом «habr.com» можно увидеть на рисунке 1.1.

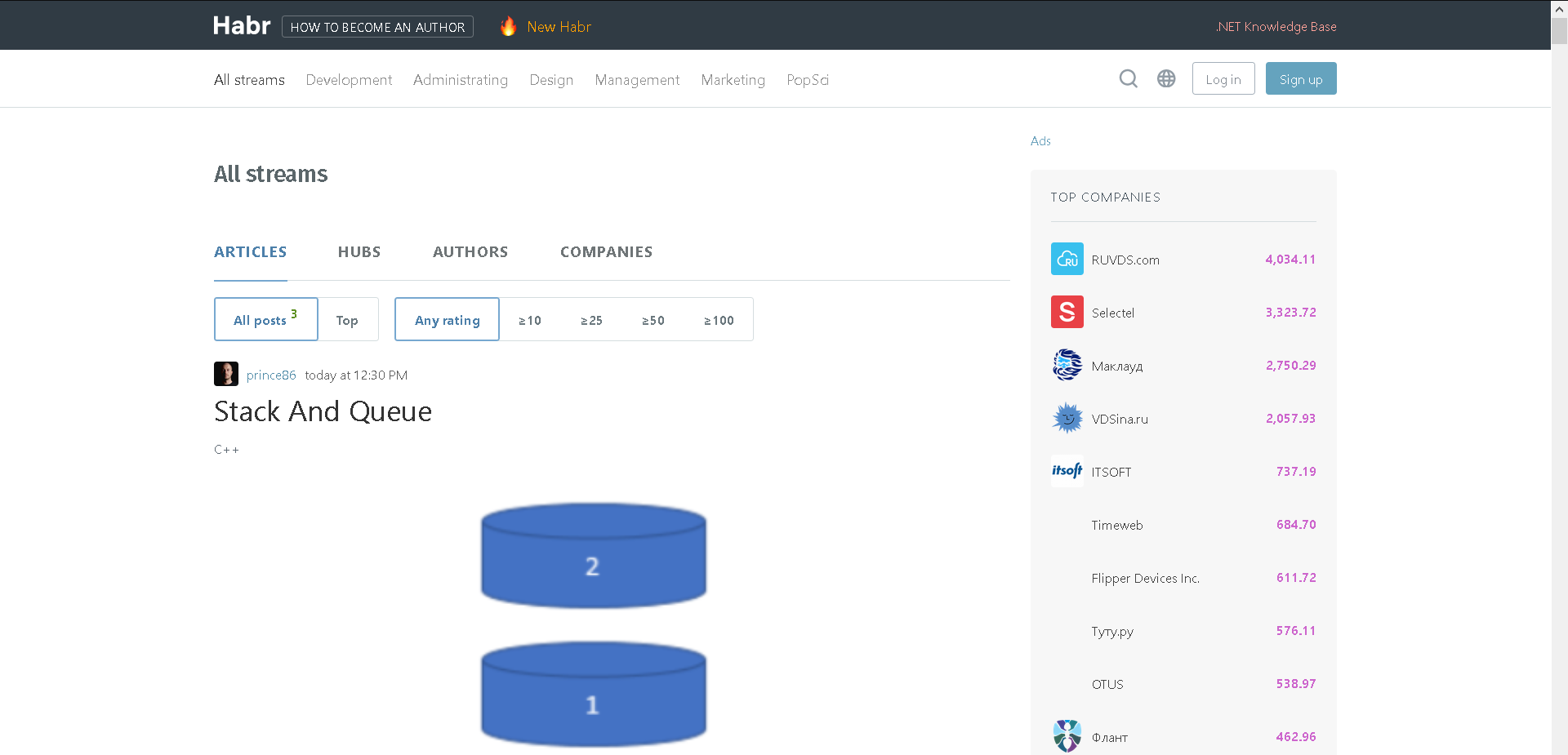


Рисунок 1.1 – Главная страница «habr»

К функциям данного приложения относятся:

* вход в личный кабинет;
* предложение идей;
* просмотр новостей;
* возможность сортировать новости по хештегам;

К достоинствам «habr» можно отнести его функциональность, не перегруженность и ненавязчивый дизайн, а также наглядность. Главной чертой является демонстрация последних новостей, касающихся непосредственно самого сайта на главной странице приложения.

К недостаткам же можно отнести необходимость совершать множество переходов для простых действий.

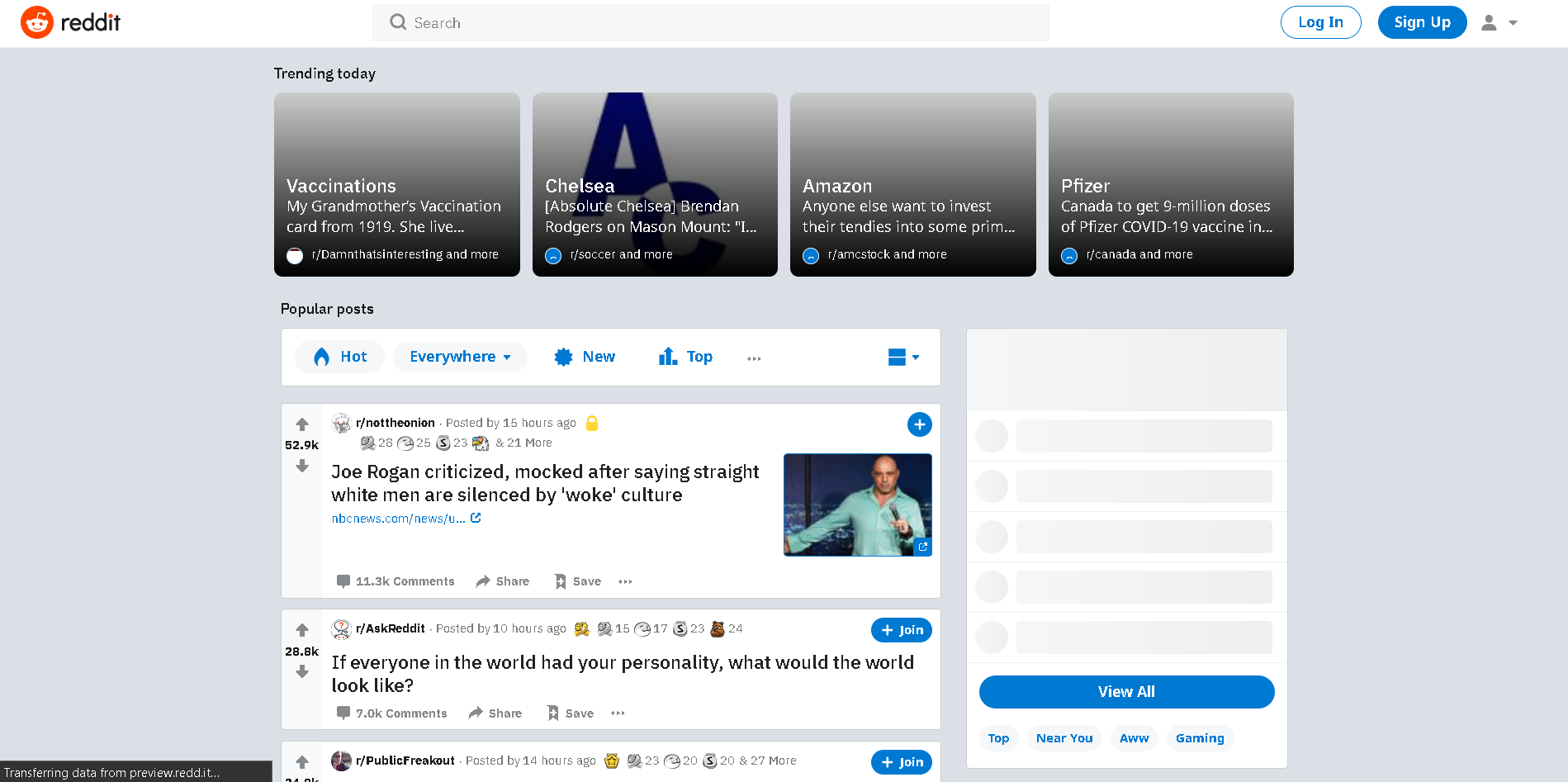


Рисунок 1.2 – Интерфейс сайта «reddit»

«*Reddit*», на мой взгляд, наиболее удобен в функциональном отношении с точки зрения незарегистрированного пользователя, так как она сразу имеет возможность на главной странице попасть в основные разделы или перенаправить в раздел приобретения подписки. Главную страницу можно увидеть на рисунке 1.2.

К недостаткам можно причислить отсутствие русского языка.

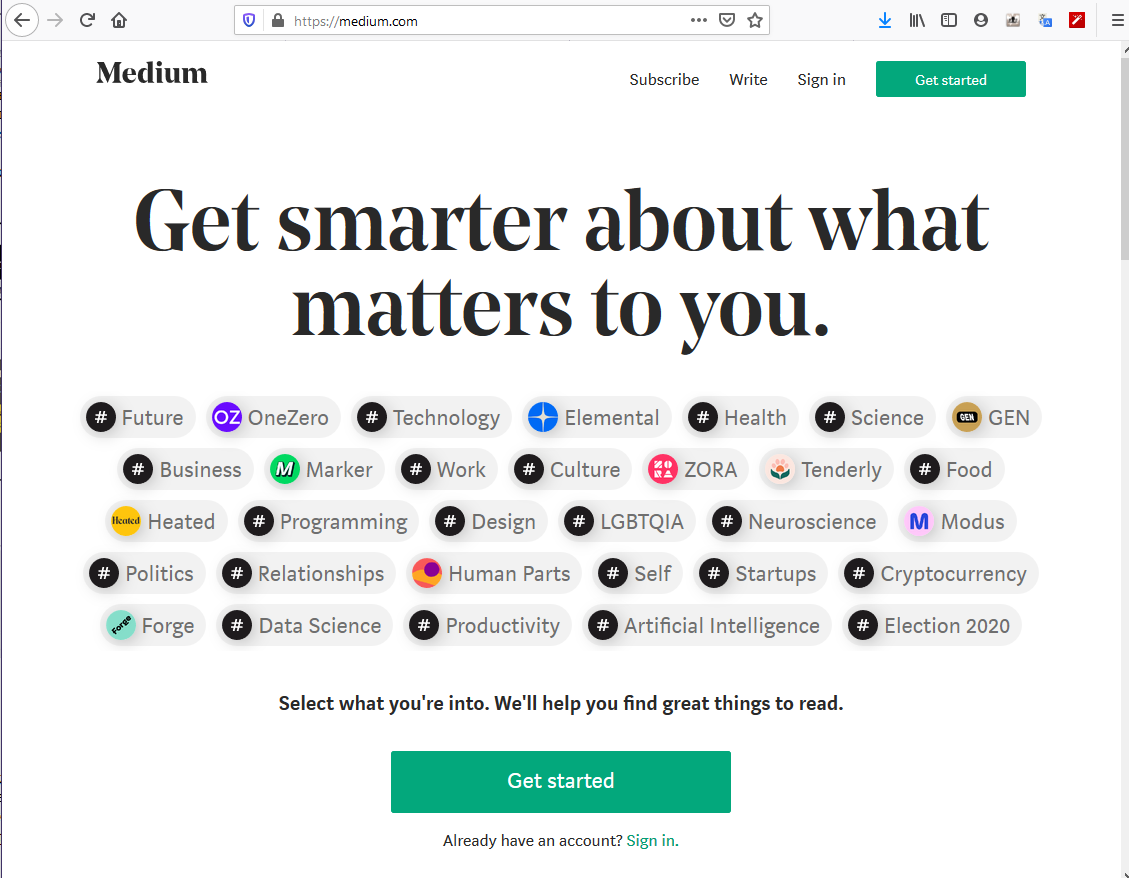


Рисунок 1.3 – Главная страница сайта «Medium»

На первый взгляд сразу в глаза бросается облако тегов. Это не спроста так как именно на них и завязано приложение. Приложение сходу предлагает зайти в свой аккаунт Google, чтобы максимально быстро приступить к работе с приложением. На данный момент по моим личным наблюдением это один из самых популярных порталов в данной сфере. Единственный недостаток, который может доставить неудобства конечному пользователю это, что чтение статей недоступно пользователю, не вошедшему в аккаунт.

Плюсом может являться поддержка абсолютно любых разрешений экранов, в том числе и мобильных, приложение автоматически адаптируется при изменении размера. Если сравнивать с предыдущим примером, реклама здесь является совсем ненавязчивой.

## 1.3 Актуальность задачи

Просмотрев и проанализировав 3 примера реализации веб-приложений новостных лент, можно сделать выводы:

* стоит уделить внимание цветовым решениям и удобности чтения текста;
* необходимо уделить внимание безопасности приложения, а также защите от sql injections и нажатия клавиши F5;

На сегодняшний день услуги, предоставляемые подобными новостными лентами, увеличиваются в объеме, а сама отрасль приобретает все больше динамики. Полный контроль над всеми процессами в данной сфере способны обеспечить только надежные информационные технологии. В силу этих причин издательские компании вынуждены внедрять инструменты, которые способны обеспечить наглядность, прозрачность и учет производимых операций.

Важно сделать текст легкочитаемым, для этого используется целый набор правил. Очень сложно считать тонкую надпись на пестром фоне — она в нем растворится. Шрифт на пару тонов темнее/светлее фона тоже будет неразличим — буквы сольются с фоновыми фрагментами своего цвета, и получится каша.

Читатель не должен всматриваться и расшифровывать надпись на анонсе, у него нет времени играть в угадалки. Информацию нужно донести с первого взгляда. Если ради этого нужно пожертвовать «красотой» и убрать фон, то придется жертвовать.

Если правильного фона нет, а менять фотографию не хочется, добавьте контрастную плашку и поставьте текст на нее. Плашка может быть строгой геометрической формы или «пятном», ее можно сделать общей на весь блок, под каждую строку или даже под каждое слово. Необязательно делать плашку непрозрачной. Даже небольшая контрастность отделит текстовую область от фотографии и кардинально повысит читабельность.

Если текст плохо смотрится на фоне в том объеме, в котором он был задуман изначально, то стоит разбить его на фрагменты и распределить по контрастным фонам внутри картинки. Так будут убиты сразу два зайца – сделано слова частью визуальной истории и решен вопрос с контрастным фоном.

# 2. Проектирование веб-приложения

## 2.1 Архитектура проекта

В качестве архитектуры для веб-приложения был выбран паттерн MVC (Model View Controller) – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Связь компонентов этого типа архитектуры можно рассмотреть на рисунке 2.1.

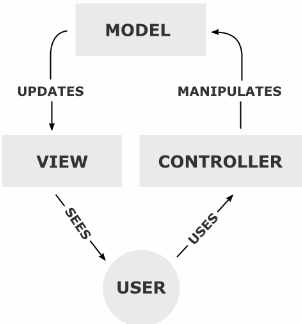


Рисунок 2.1 – Структура паттерна MVC

Основные компоненты:

* Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

## 2.2 Общая структура проекта

Так как первоначальная задача веб-приложения эта обработка HTTP запросов, то ориентироваться можно на схему, представленную на рисунке 2.2.

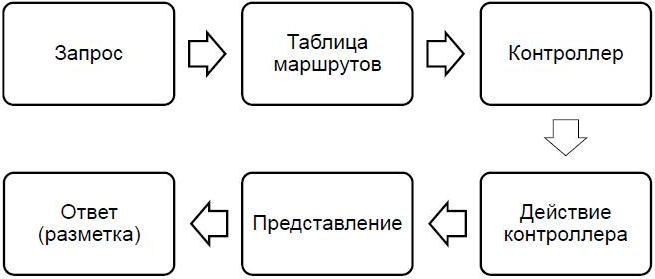


Рисунок 2.2 – Процесс обработки запроса

Так как веб-приложение помимо стандартных для MVC трех компонентов имеет и четвертый, называемый таблицей маршрутов, определим и его. Часть таблицы показана на рисунке 2.3.

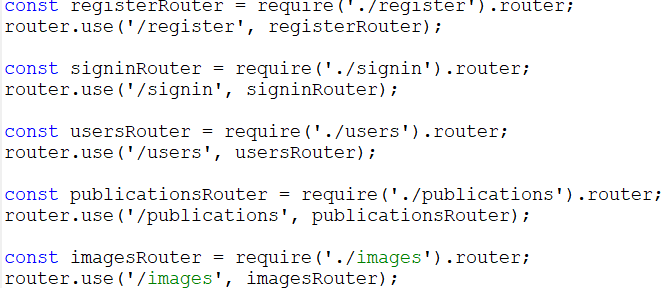


Рисунок 2.3 – Часть кода таблицы маршрутизации.

Общая структура того, как приложение будет обрабатывать запросы от пользователей показана на рисунке 2.4.

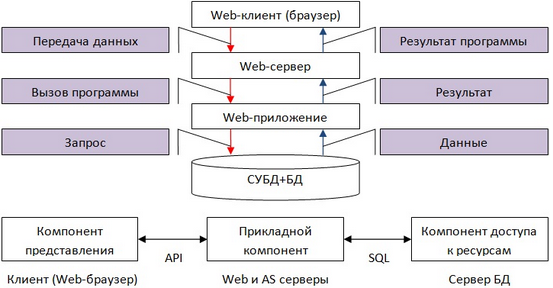


Рисунок 2.4 – Общая структура

## 2.3 Схемы и описание работы веб-приложения

Веб-приложение «Менеджер конспектов студенческих лекций» должно обладать следующим функционалом:

* регистрация, авторизация пользователей;
* возможность создавать статьи;
* поддержка markdown разметки в статьях;
* возможность предпросмотра статьи при публикации;
* для роли администратора возможность подтверждать или отвергать предложенные статьи;
* возможность прикрепления изображений и их вставка;
* комментирование статей авторизированными пользователями;
* панель администрирования для роли администратора.

Для реализации веб-приложения были выбраны следующие технологии:

* SQLite[1] –  компактная встраиваемая СУБД;
* Node.js – платформа для исполнения кода javascript;
* Express[3] ­– фреймворк node.js, обеспечивающий работу с протоколом http;
* Materialize – css фреймворк с сеткой и готовыми компонентами для построения интерфейса;

Также в проекте используются следующие паттерны проектирования:

* MVC – схема разделения данных приложения, [пользовательского интерфейса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.
* Singleton.

В этом подразделе будет описан базовый функционал web-приложения «менеджер конспектов студенческих лекций». Система подразумевает хранение двух типов пользователей, это: пользователь и администратор. Пользователь имеет возможность просматривать статьи, писать статьи, управлять своими статьями, смотреть чужие и оставлять под ними комментарии, а также писать сообщения другим пользователям, хранить и редактировать черновики и многое другое, более подробно с возможностями пользователя можно ознакомиться на рисунке 2.5.

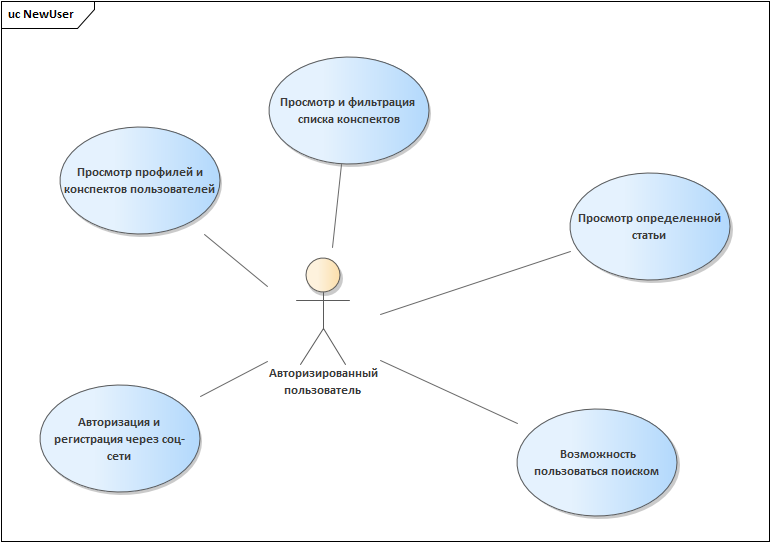


Рисунок 2.5 – Диаграмма использования обычного пользователя

Возможности незарегистрированного пользователя не такие богатые, их можно увидеть на рисунке 2.6.

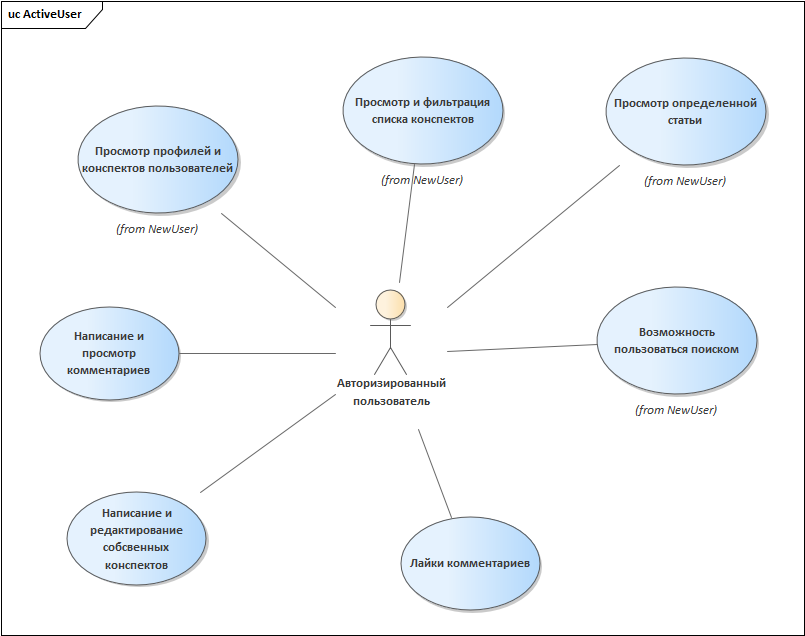


Рисунок 2.6 – Диаграмма использования незарегистрированного пользователя

Разница между пользователем и администратором заключается в том, что администратор дополнительно имеет доступ к списку пользователей ресурса, всем публикациям, может следить за состоянием сервера, а также подтверждать или отменять те или иные предложенные статьи. Более подробно возможности администратора можно рассмотреть на диаграмме использования, представленной на рисунке 2.7.

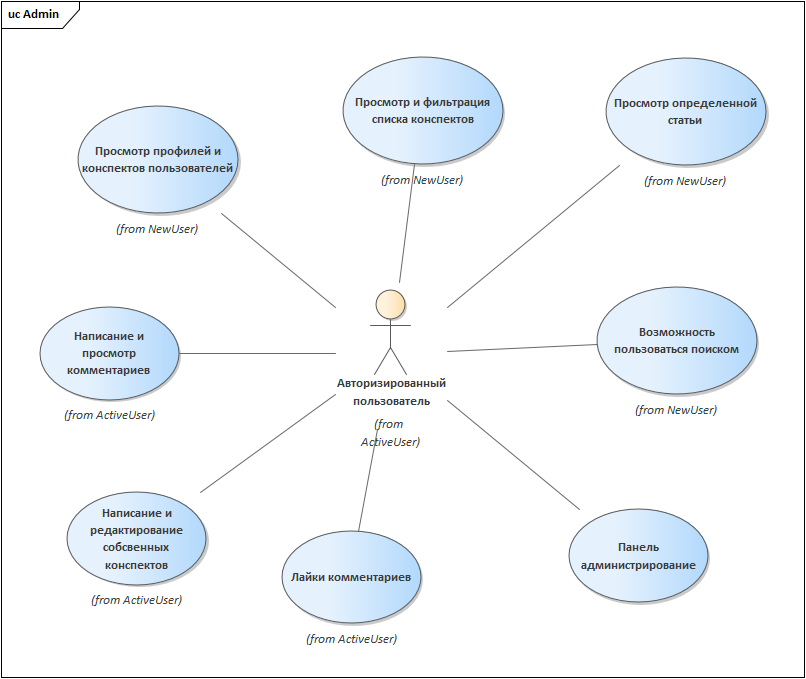


Рисунок 2.7 – Диаграмма использования администратора приложения

Таким образом в разделе был описан функционал двух основных групп пользователей, используемых в приложении.

Языком разработки мной был выбран nodejs[2] с фреймворком express, поскольку express предоставляет полный функционал для разработки веб-приложений.

Express — это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Имея в своем распоряжении множество служебных методов HTTP и промежуточных обработчиков, создать надежный API можно быстро и легко.

Express предоставляет тонкий слой фундаментальных функций веб-приложений, которые не мешают вам работать с давно знакомыми и любимыми вами функциями Node.js.

Основная идея Node.js: использовать неблокирующий событийно-ориентированный ввод/вывод, чтобы оставаться легковесным и эффективным при обращении с приложениями, обрабатывающими большие объемы данных в реальном времени и функционирующими на распределенных устройствах.

Язык javascript имеет множество простых возможностей обработки json и xml файлов, т.к. проект по-большей части построен на api быстрая и удобная обработка json пакетов является важной задачей, чем вкупе с фреймворком express проект успешно справляется.

В качестве СУБД мною была выбрала SQLite3, поскольку на данный момент будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных. Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций.

## 2.4 Модель базы данных

Для создания, хранения и управления данными, была использованная реляционная встраиваемая бд SQLite. Ниже на рисунке 2.8 представлена модель базы данных (таблицы, отмеченные на них первичных и вторичных ключей) курсового проекта. База данных была подвергнута процедуре нормализации данных.

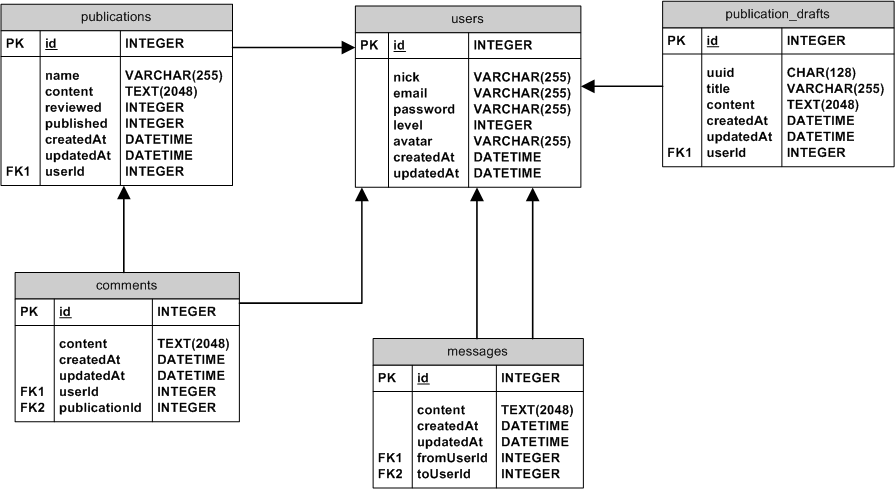


Рисунок 2.8 – Модель базы данных

База данных на выше представленной схеме в полной мере описывает необходимые таблицы для решения поставленной задачи данного курсового проекта.

Для создания понимания следует разобрать поле каждой таблицы.

Таблица USERS, каждая запись которой представляет собой отдельного пользователя системы. Состоит из следующих полей:

* id – уникальный айди пользователя системы;
* nick – имя пользователя, используемое для отображения на сайте;
* email – адрес электронной почты пользователя;
* password – пароль пользователя, хранящийся в SHA-256 хеше;
* level – уровень привилегий пользователя;
* avatar – ссылка на аватар пользователя;
* createdAt – дата и время создания аккаунта;
* updatedAt – дата и время последнего обновления данных аккаунта.

Пользователь имеет возможность перед созданием публикаций держать у себя в профиле набор черновиков, которые представлены таблицей publication\_drafts, ее поля следующие:

* id – уникальный айди черновика;
* uuid – случайным образом генерирующийся uuid;
* title – заголовок или же название планируемой публикации;
* content – содержимое публикации в виде markdown разметки;
* createdAt – дата и время создания публикации;
* updatedAt – дата и время обновления публикации;
* userId – автор черновика, которому он принадлежит.

Так как у пользователя имеется возможность делать черновики публикаций, так у него и есть возможность делать сами публикации, для этого в приложении используется таблица publications, которая содержит следующие поля:

* id – уникальный айди публикации;
* name – заголовок публикации;
* content – содержимое публикации;
* reviewed – проверена ли была публикация после добавления;
* published – опубликована ли публикация по результатам проверки;
* createdAt – дата и время создания публикации;
* updatedAt – дата и время последнего обновления публикации;
* userId – айди автора публикации.

Пользователи имею возможность писать комментарии к публикациям, за них отвечает соответствующая таблица с именем comments, поля которой следующие:

* id – уникальный айди комментария;
* content – содержимое комментария;
* createdAt – дата и время создания комментария;
* updatedAt – дата и время обновления комментария;
* userId – айди автора комментария;
* publicationId – айди публикации, к которой написан комментарий.

Одним из способов взаимодействия между пользователем является возможность писать сообщения друг другу, эту роль в системе выполняет таблица messages, поля которой:

* id – уникальный айди пользователя;
* content – содержимое комментария;
* createdAt – дата время создания комментария;
* updatedAt – дата и время обновления комментария;
* fromUserId – айди пользователя отправителя;
* toUserId – айди пользователя получателя.

База данных данного курсового проекта содержит в большинстве своем только таблицы. Основная причина такого подхода в том, что в проекте используется sequelize orm[4], которая дает возможность не задумываться о том, как именно данные будут представлены в базе данных.

## 2.5 Использование ORM

В общем понятии ORM – это технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Sequelize — это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в бд и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами. Причем Sequelize может работать с рядом СУБД - MySQL, Postgres, MariaDB, SQLite, MS SQL Server. В этом курсовом проекте использовался sequelize в связке со встраиваемой базой данных SQLite.

Листинги моделей базы будут приведены в приложении.

# 3. Разработка веб-приложения

## 3.1 Физическая структура и функции модулей

Приложение проектировалось по архитектуре MVC, Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Содержимое корня проекта можно увидеть на рисунке 3.1.



Рисунок 3.1 – Содержимое корня проекта.

Касаемо архитектуры MVC, в проекте можно увидеть папки controllers, models и views. Стоит подробнее изучить каждую из них. Содержимое папки controllers, содержит набор контроллеров, которые осуществляют промежуточную подготовку данных из models перед подачей их на views, то же самое и в обратную сторону. Содержимое папки controllers представлено на рисунке 3.2.

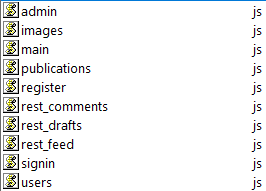


Рисунок 3.2 – Содержимое папки Controllers.

Каждый контроллер в итоге конечному пользователю возвращает тот или иной view из папки views, содержимое которой можно увидеть на рисунке 3.3.

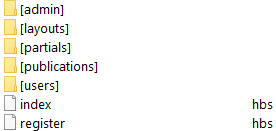


Рисунок 3.3 – Содержимое папки views.

Стоит обратить внимание на расширение файлов, это hbs. Handlebars (или сокращенно hbs) – это шаблонизатор для JavaScript, который помогает снизить сложность создания таких страниц. Страница представляет собой код html, но содержит дополнительные объявления для hbs engine, который подменит их на данные на этапе обработки.

Папка models содержит файлы моделей, каждая модель – это отдельный тип в базе данных. В случае этого приложения каждая модель является моделью sequelize. Это позволяет на ходу изменять модели, а проведенные изменения сразу же отобразятся в базе данных с первой же миграцией. Содержимое папки models можно увидеть на рисунке 3.4.

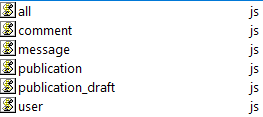


Рисунок 3.4 – Содержимое папки models.

Оставшиеся папки не представляет особого интереса т.к. относятся скорее к nodejs приложению, чем его структуре, однако следует рассмотреть то, как осуществляется маршрутизация между страницами проекта. Для этого в корне проекта можно увидеть файл routes. Содержимое папки routes можно увидеть на рисунке 3.5.

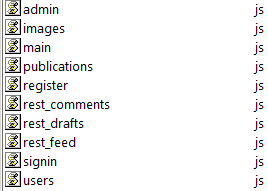


Рисунок 3.5 – Содержимое папки routes.

Каждый файл осуществляет маршрутизацию первого уровня, а в каждом из этих файлов уже реализована реализация маршрутизации дальше вглубь. Исключением является файл main.js, который подтягивает остальные маршрутизаторы и устанавливает алиасы первого уровня. Рассмотрим пример на маршрутизаторе publications. Содержимое файла представлено на рисунке 3.6.

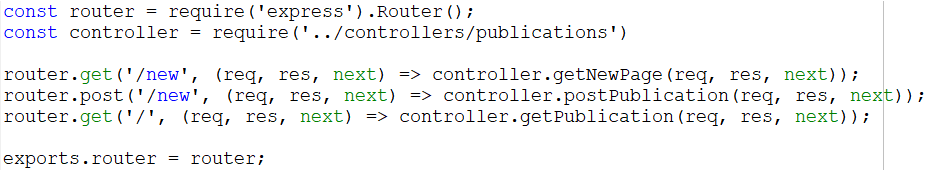


Рисунок 3.6 – Содержимое файла routes/publications.js.

По имеющимся маршрутам внутри можно увидеть, что в проекте будут задействованы следующие маршруты: «/publications/new», «/publications». Возможно, остался вопрос откуда взялся сам publications, а ответ хранится в ранее упомянутом routes/main.js, часть содержимого которого показана на рисунке 3.7.

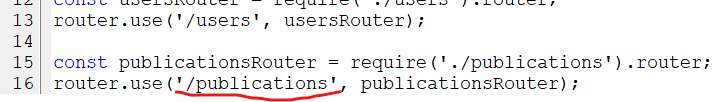


Рисунок 3.7 – Часть содержимого routes/main.js.

Здесь видно, как складывается полный URI путь к контроллеру, таким образом в проекте и работает маршрутизация.

## 3.2 Реализация хранения и хеширования пароля

В любой современной информационной системе не стоит хранить пароли пользователей в открытом виде, т.к. такой вид хранения может привести к плачевным последствиям. Например, если у злоумышленника каким-либо образом получится вытащить данные с базы данных, то эти данные все равно останутся защищенными, т.к. злоумышленник не сможет произвести авторизацию в систему благодаря тому, что пароли ему неизвестны. Плюсов у хеширования паролей насчитывается бесчисленное множество, однако это лишь один из примеров.

В реализации своей я выбрал алгоритм хеширования SHA-256, т.к. он соответствует современным требованиям к алгоритмам хеширования, а также вычисляет хеш-значение за приемлемое время и имеет фиксированную длину, что дает возможность эффективно хранить его в базе данных. Для реализации механизма хеширования был использован npm-пакет crypto. Его импорт в коде показан на рисунке 3.8.

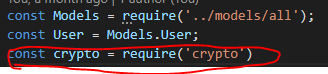


Рисунок 3.8 – Использования пакета crypto

Этот пакет предоставляет набор хеш алгоритмов, которые можно использовать, просто вызвав соответствующую функцию. Небольшой листинг кода, решающий эту задачу представлен на рисунке 3.9.

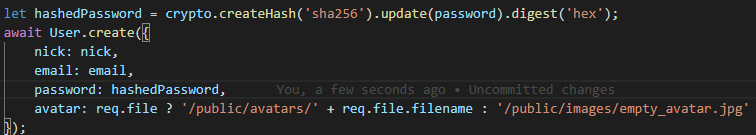


Рисунок 3.9 – Регистрация нового пользователя с хешированием пароля.

В результате внутри базы данных пароль пользователя будет хранится в виде, подобном тому, который изображен на рисунке 3.10.

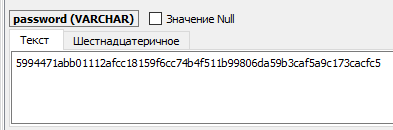


Рисунок 3.10 – Хранение пароля в базе данных.

Ко всему прочему в серьезных проектах обязательно стоит добавить к паролю так называемую соль, это может быть случайный набор байт, или даже заранее установленная последовательность байт. Дело в том, что ее наличие внесет больше хаотичности в хеш, что предотвратит потенциальную перебор паролей по хешу. Т.к. для большинства типичных паролей хеши уже посчитаны и существуют и найти соответствие при желании злоумышленнику проблем не составит.

## 3.3 Реализация пагинации

В современном мире в связи с изобилием контента, уместить всё за раз на странице невозможно, учитывая тот факт, что данные передаются по сети, нагрузка на сервер становится необоснованно большой. В добавок к этому можно добавить, что вовсе не гарантируется, что вся переданная информация будет использована конечным пользователем, поэтому грамотным решением является отдавать эту информацию порционно, в зависимости от нужд и запросов пользователя.

В проекте используется множество вариантов дозагрузки содержимого, как от пагинации так до автоматической дозагрузки содержимого по мере необходимости.

Так как в проекте используется sequelize orm ,то ее реализация не представляет собой особых проблем. Дело в том, что все запросы в sequelize поддерживают две дополнительные настройки, это limit и offset, использование которых показано на рисунке 3.11.

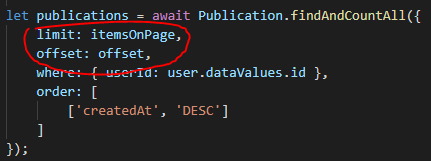


Рисунок 3.11 – Реализация пагинации с помощью sequelize.

В этом случае поле limit устанавливает количество строк, которое следует отобрать, а поле offset соответственно задает смещение относительно начала базы. На рисунке 3.11 используется метод findAndCountAll вместо find, потому что дополнительно с отобранными данными, метод возвращает количество всех записей попавших под условие where без ограничителей в виде limit и offset. Это дает возможность посчитать количество страниц и соответственно оценить, на какой странице сейчас находится пользователь, чтобы номер этой страницы подсветить. Реализация подсчета показана на рисунке 3.12.

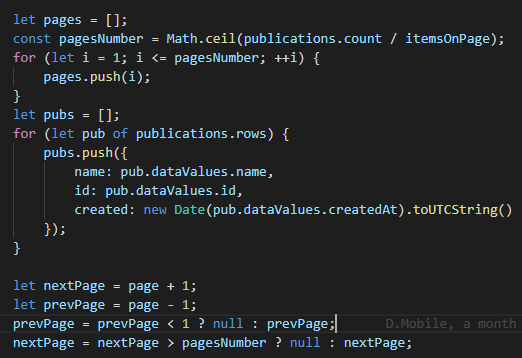


Рисунок 3.12 – Подсчет количества страниц и оценка ближайших.

Внутри приложения так же используется автопрогрузка некоторых сущностей, например новости на главной странице. Для этого была создана функция, которая отлавливает момент, когда пользователь пролистал до самого низа страницы и вызывает функцию загрузки новостей. Участок кода на клиенте, реализующий этот механизм показан на рисунке 3.13.

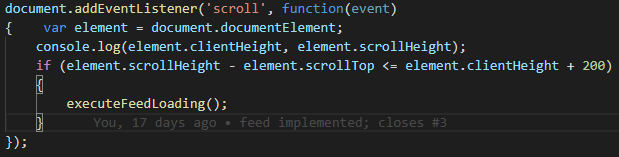


Рисунок 3.13 – Реализация автозагрузки на клиенте.

Новости на главной странице загружаются через REST запросы, поэтому при загрузке главной страницы первая пачка новостей загружается автоматически, дальше только по пролистыванию страницы в самый низ.

## 3.4 Реализация уведомления на сообщения

Так как приложение дает возможность пользователям писать друг другу сообщения, было решено добавить уведомления, показывающиеся у получателя сообщения, когда то доходит. Написать сообщение можно со страницы пользователя, которому это сообщение отправляется. Страница, дающая возможность отправлять сообщения показана на рисунке 3.14.

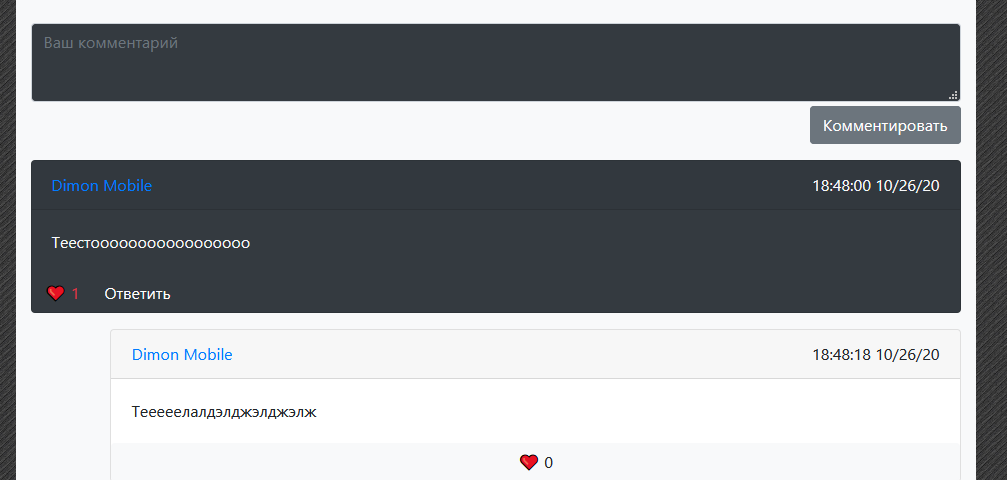


Рисунок 3.14 – Написание сообщения другому пользователю.

Для создания уведомлений используется websocket сервер. Авторизовавшийся клиент на клиентской части помимо основного канала передачи данных, так же подключается к websocket серверу, который сообщает о наступившем событии. Результат такой работы можно увидеть на рисунке 3.15.

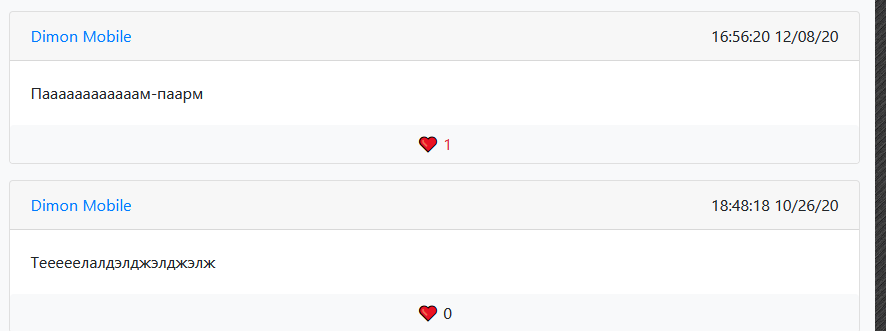


Рисунок 3.15 – Демонстрация уведомления на комментарий.

Серверная websocket часть написана в основном файле и стартует вместе с http сервером, часть кода, реализующая websocket сервер показана на рисунке 3.16.



Рисунок 3.16 – Websocket сервер для сообщений.

Существует две версии, но в данном проекте помимо вебсокета, был принят вариант и с использоваением longpoll.

Long Polling — это технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдет какое-либо событие (например, придёт новое сообщение), либо истечет заданное время ожидания.

Другими словами, получая от вас запрос, сервер ждет, когда произойдет событие, о котором он должен вас уведомить, и, когда оно происходит, Long Poll сервер отправляет ответ на ваш запрос, содержащий информацию о случившемся событии.  
Мы напишем программу, которая будет уведомлять пользователя о некоторых изменениях в его аккаунте, которые мы будем получать от Long Poll сервера. Чтобы начать получать ответы от сервера, необходимо получить три обязательных параметра, необходимых для работы Long Poll-a: server, key и ts.

Клиентская часть еще проще и состоит всего из пары функций, листинг которых частично изображен на рисунке 3.17.



Рисунок 3.17 – Клиентская часть websocket-уведомлений.

4. Руководство пользователя

## 4.1 Программно-аппаратные ресурсы ПК

В процессе работы использовались следующие технические и программные средства:

Технические средства:

* ноутбук Lenovo K53S;
* Windows 10 Professional (64 бит);
* Visual Studio Code;
* Oracle VirualBox;
* Node.js;
* Sequelize
* NodeJS
* npm

## 4.2 Руководство по запуску веб-приложения

После запуска виртуальной машины для старта сервера достаточно в консоли из папки с кодом выполнить «npm start».

## 4.3 Руководство для гостя

На стартовой странице незарегистрированному пользователю предоставляется список последних материалов, как это показано на рисунке 4.1.

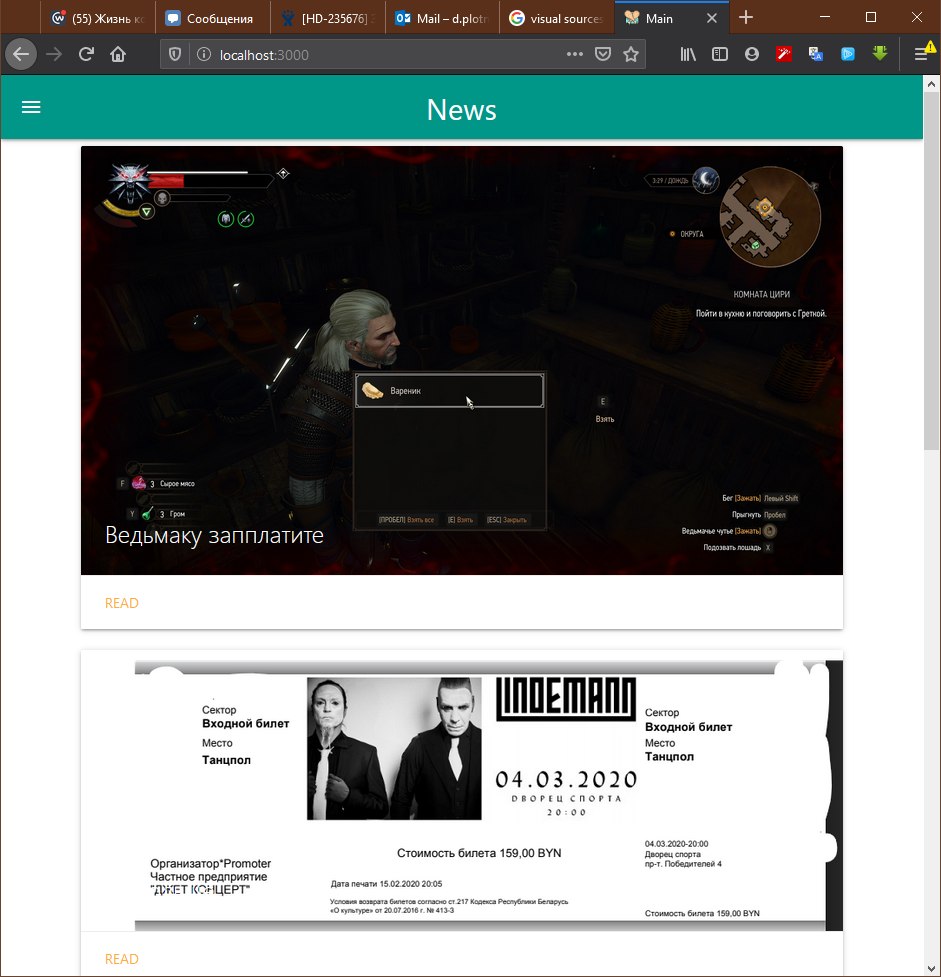


Рисунок 4.1 – Главная страница для неавторизированного пользователя

Клик по любому из материалов откроет его и сделает возможным его просмотр, как это продемонстрировано на рисунке 4.2.

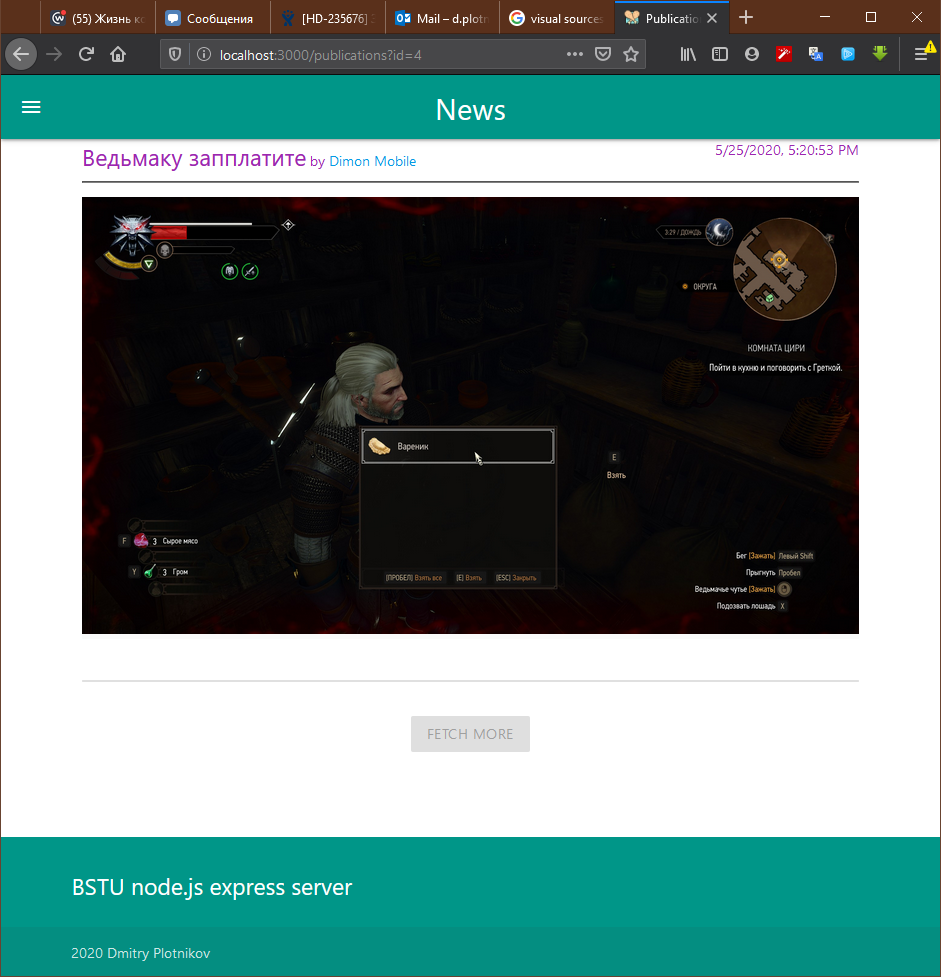


Рисунок 4.2 – Просмотр материала анонимным пользователем

Как видно из рисунка, для того чтобы оставлять комментарии пользователю требуется осуществить вход в систему.

## 4.4 Руководство для авторизированного пользователя

Каждая новая роль не исключает предыдущие возможности, а только добавляет их. У авторизированных пользователей на страницах материалов появляется возможность оставлять комментарии и просматривать их. Так как этот момент показывался и расписывался ранее, здесь изображения прилагаться не будут.

В правом нижнем углу имеется кнопка, позволяющая создать новый материал, ее можно увидеть на рисунке 4.3.

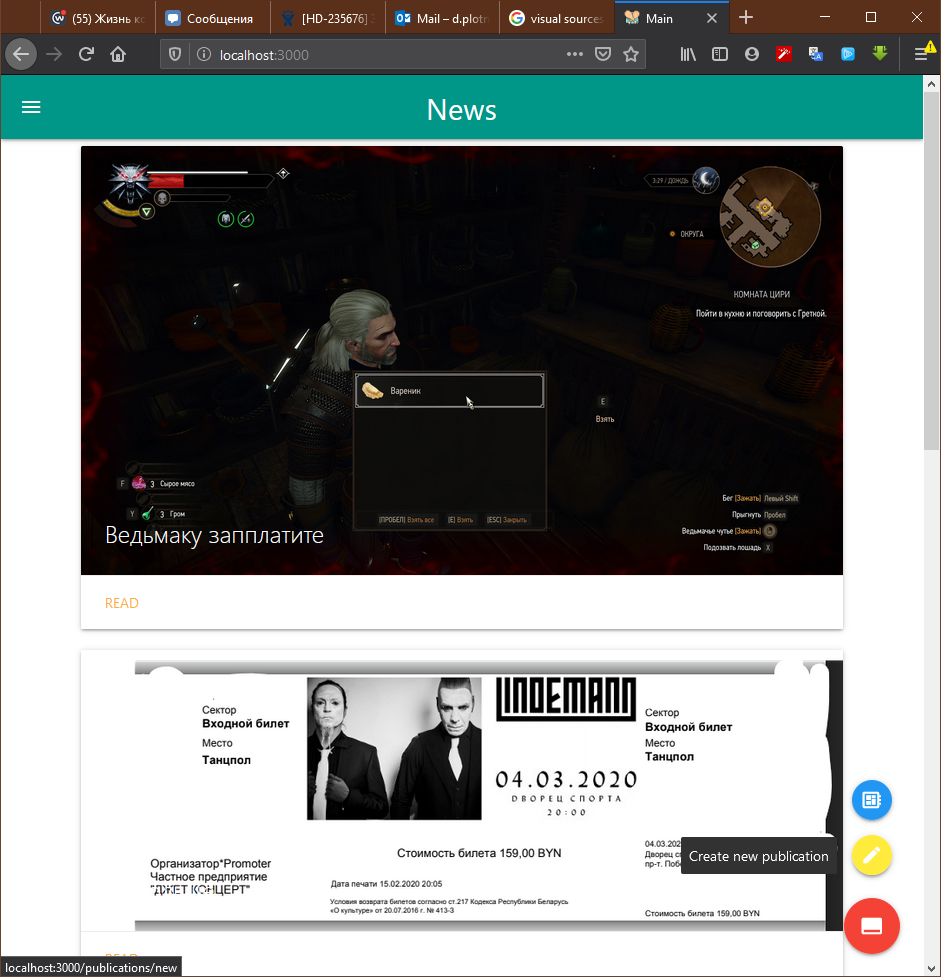


Рисунок 4.3 – Демонстрация кнопки создания материала

Кнопка доступна с любой страницы и не привязана лишь к главной. По нажатию на нее происходит перемещение на страницу создания материала. Здесь существует одно единственное текстовое поле, в котором имеется возможность написать статью с использование форматирования, обеспечиваемого применением markdown-разметки. Краткие подсказки отображены ниже этого поля, что и показано на рисунке 4.4.

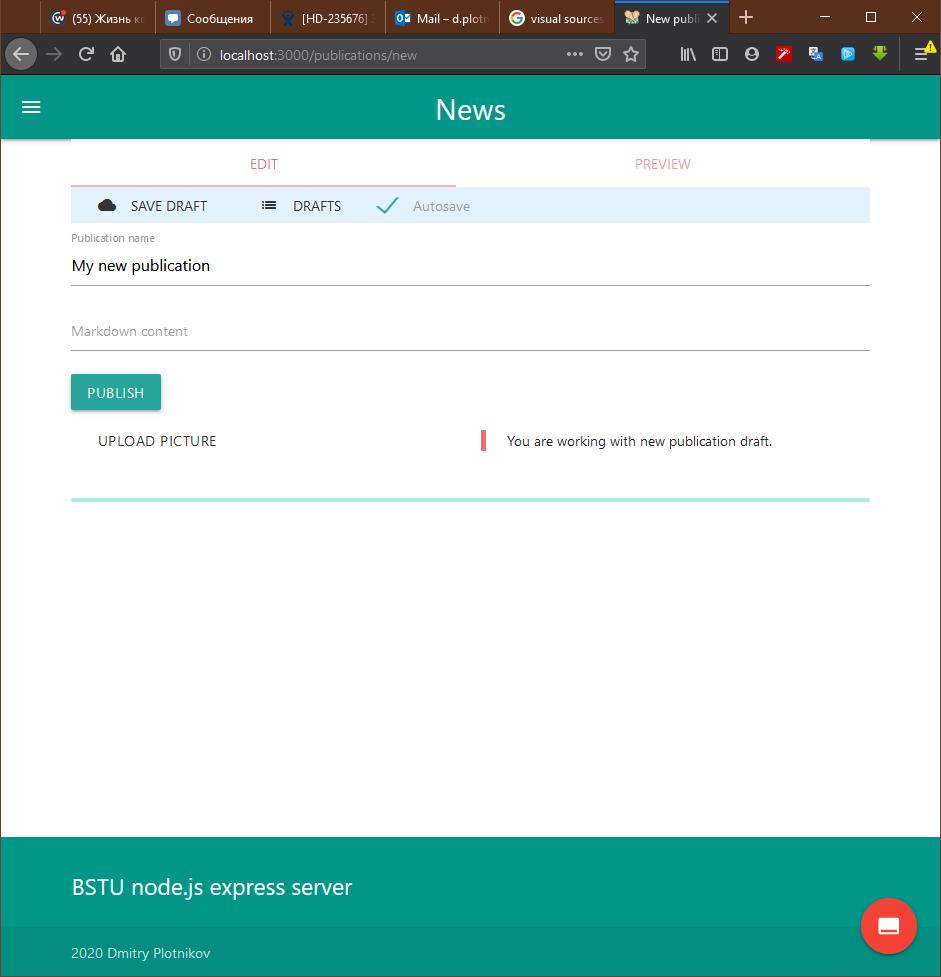


Рисунок 4.4 – Демонстрация страницы создания материала

Очень удобным моментом является возможность просмотреть как будет выглядеть материал еще до того, как он будет сохранен. Сверху для этого и предусмотрена вкладка PREVIEW. К статье можно прикрепить фотографию, а то и несколько и вставлять их по ходу, как это и подразумевается в markdown-синтаксисе. После загрузки изображения на сервер можно вставить его код в редактор, лишь кликнув по его имени. Эти действия продемонстрированы на рисунке 4.5.

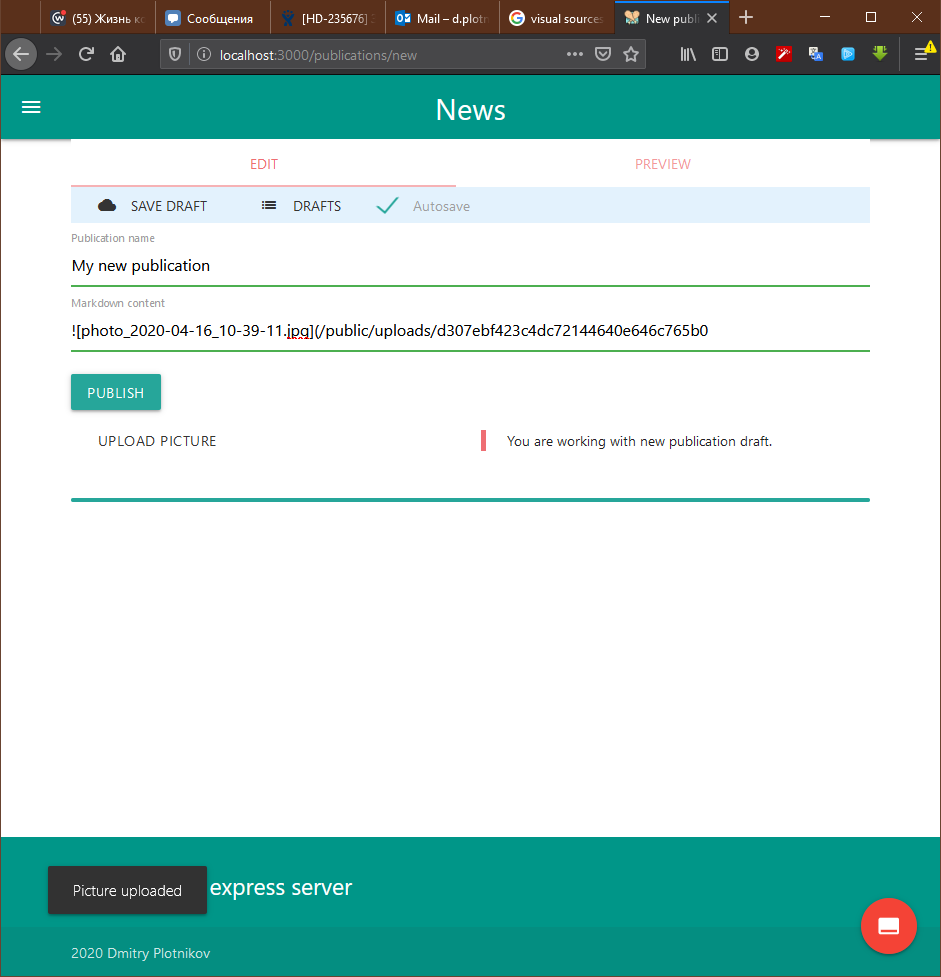


Рисунок 4.5 – Демонстрация работы редактора материалов

А по нажатию на кнопку PREVIEW можно увидеть результат, как и показано на рисунке 4.6.

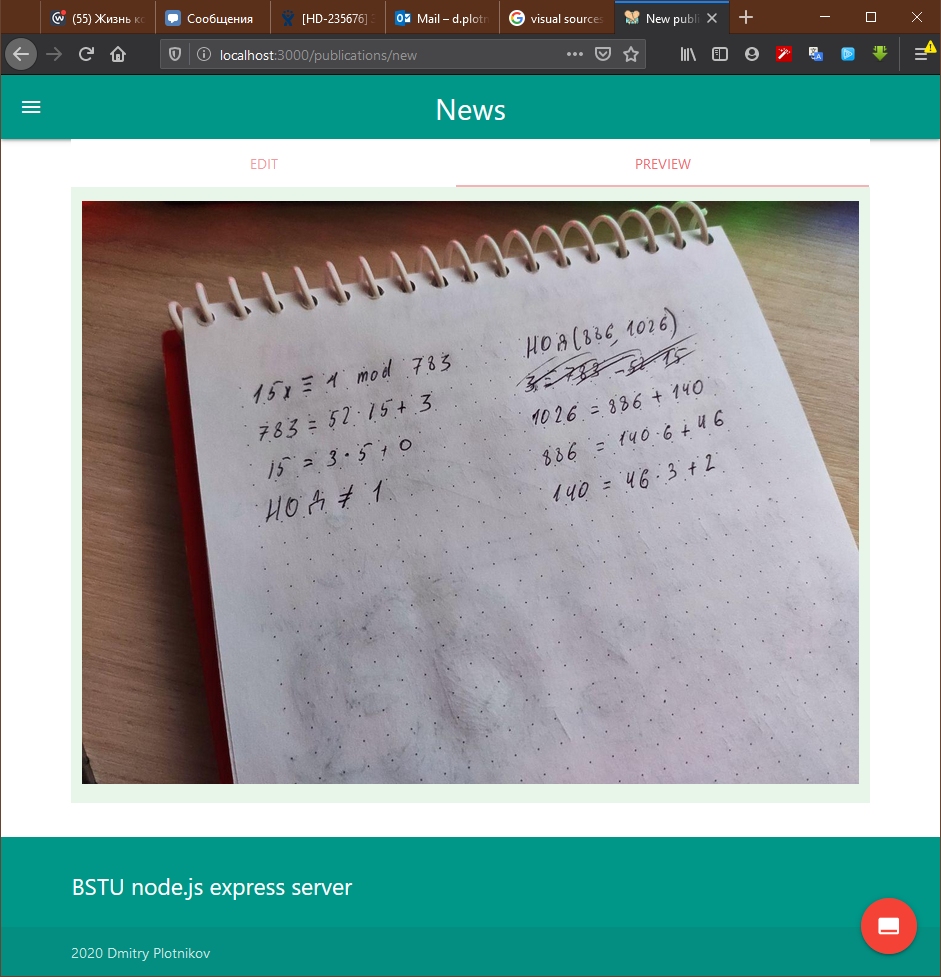


Рисунок 4.6 – Демонстрация предпросмотра материала

Далее по нажатию на кнопку готовности, и вводу имени можно сохранить материал в системе, как показано на рисунке 4.7.

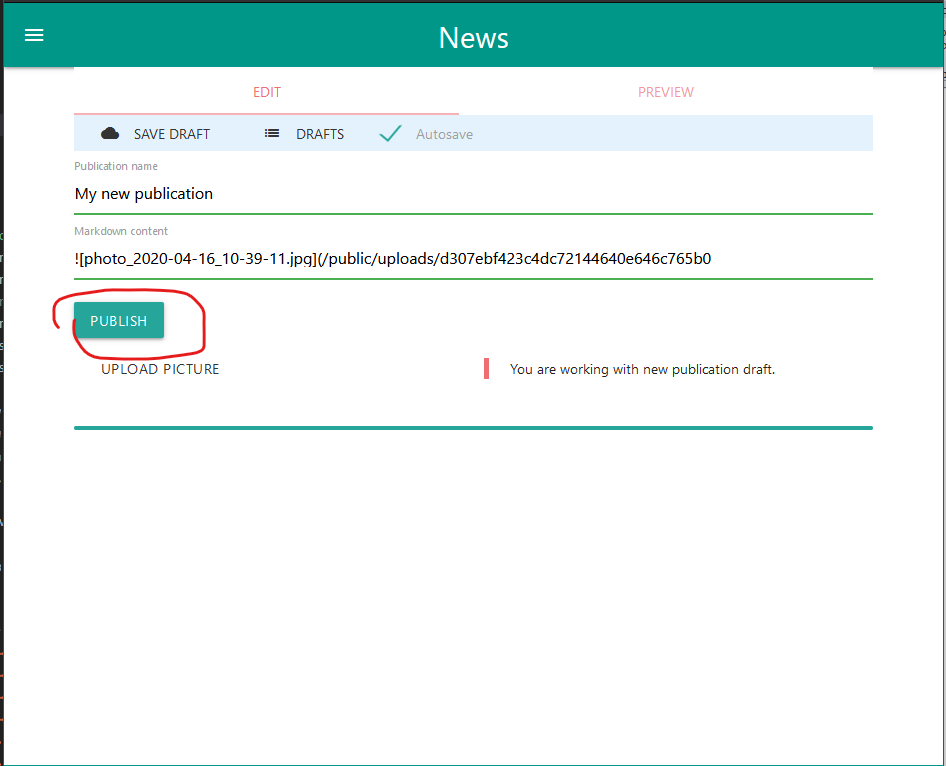


Рисунок 4.7 – Создание нового материала

После создания пользователь попадает к себе в профиль в новую вкладку, где видит список своих материалов, что и продемонстрировано на рисунке 4.8.

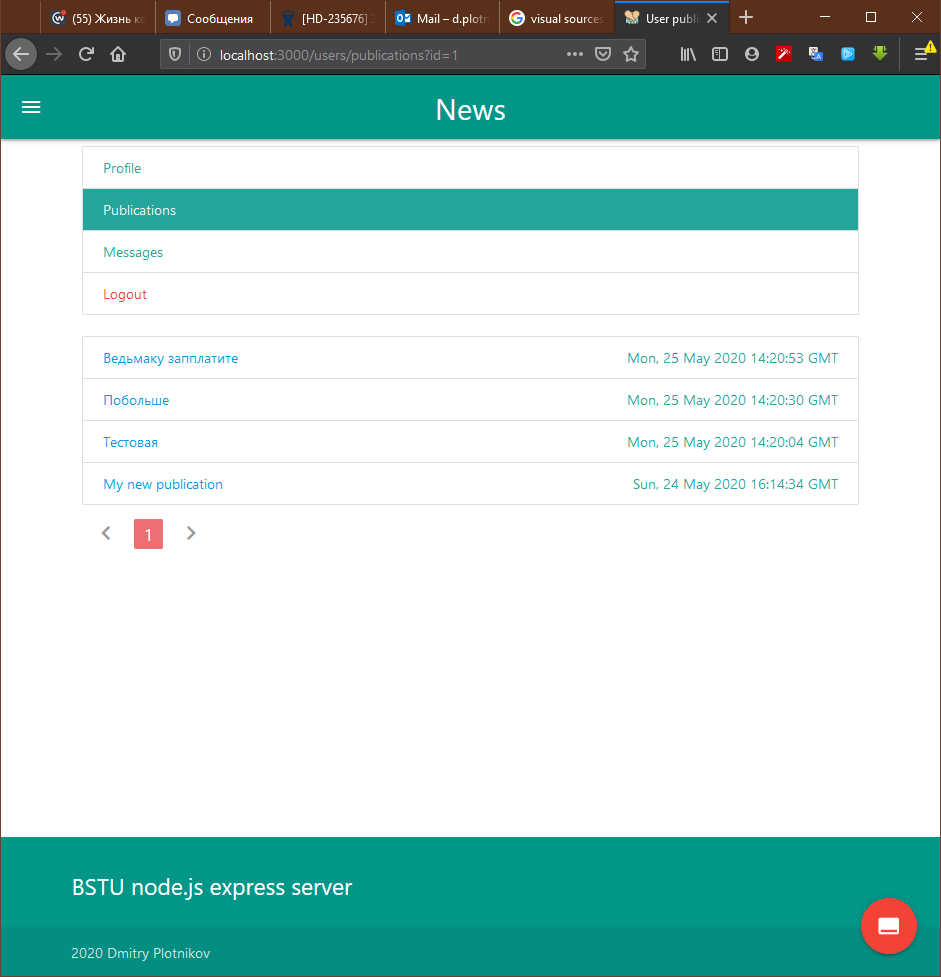


Рисунок 4.8 – Вкладка пользователя с материалами

Созданный материал не попадает на главную страницу, перед этим он должен быть проверен и одобрен редактором или администратором сайта.

## 4.5 Руководство для администратора

На странице профиля появляется раздел связанный с администрированием, как это показано на рисунке 4.9, однако внутри него пользователь увидит только материалы, в то время как для администратора будут доступны все функции.

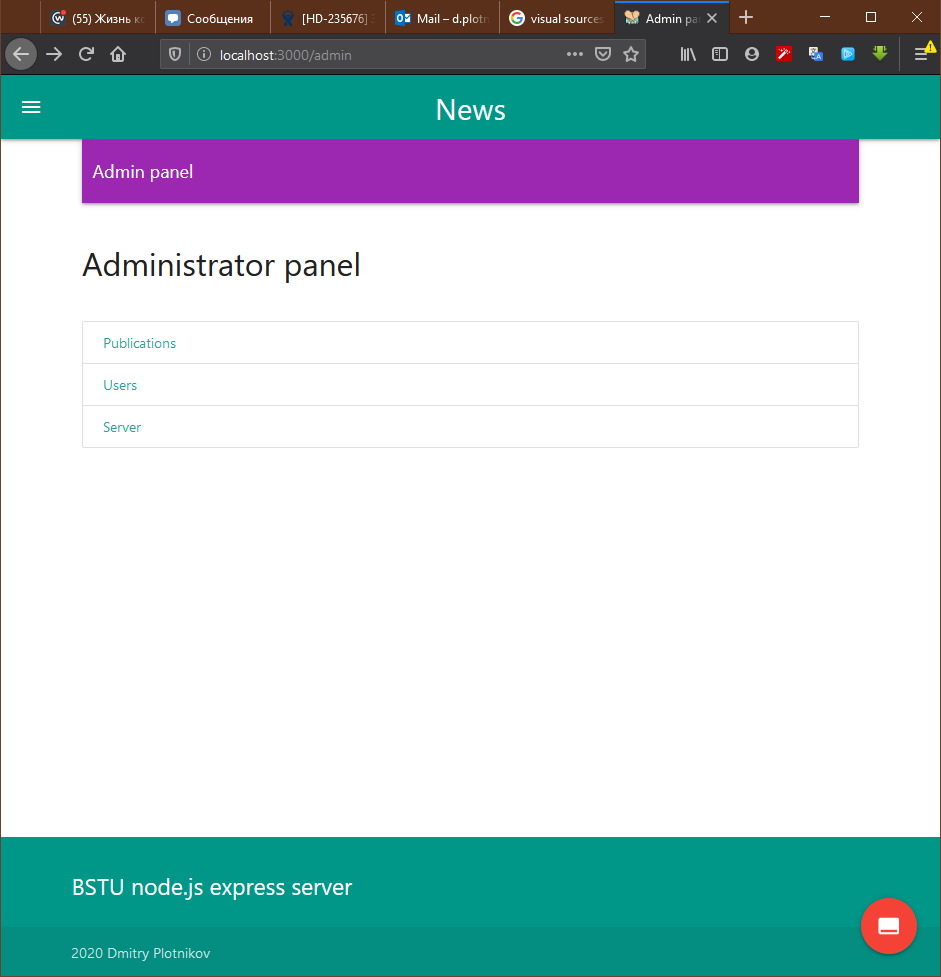


Рисунок 4.9 – Демонстрация раздела администратора

В каждом из них, кроме экспорта можно просмотреть список соответствующего типа данных, хранящихся на сервере и получать таким образом к ним быстрый доступ. В разделе материалов для пользователя и администратора существует фильтр, позволяющий увидеть список материалов, нуждающихся в предварительном просмотре перед публикацией. Продемонстрировано на рисунке 4.10.

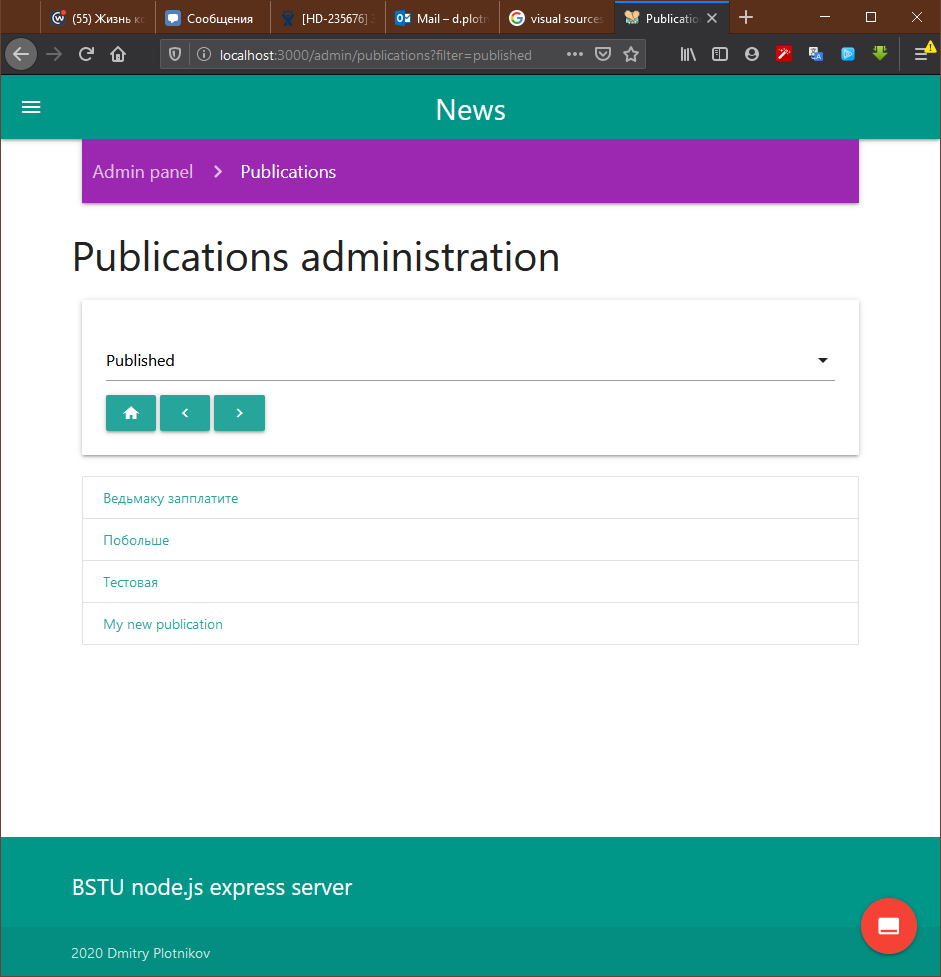


Рисунок 4.10 – Демонстрация фильтра по материалам

Открыв такой материал справа появляется сообщение, предупреждающее о том, что этот материал следует подробно просмотреть и либо одобрить его, либо отвергнуть. Страница, демонстрирующая это показана на рисунке 4.11.

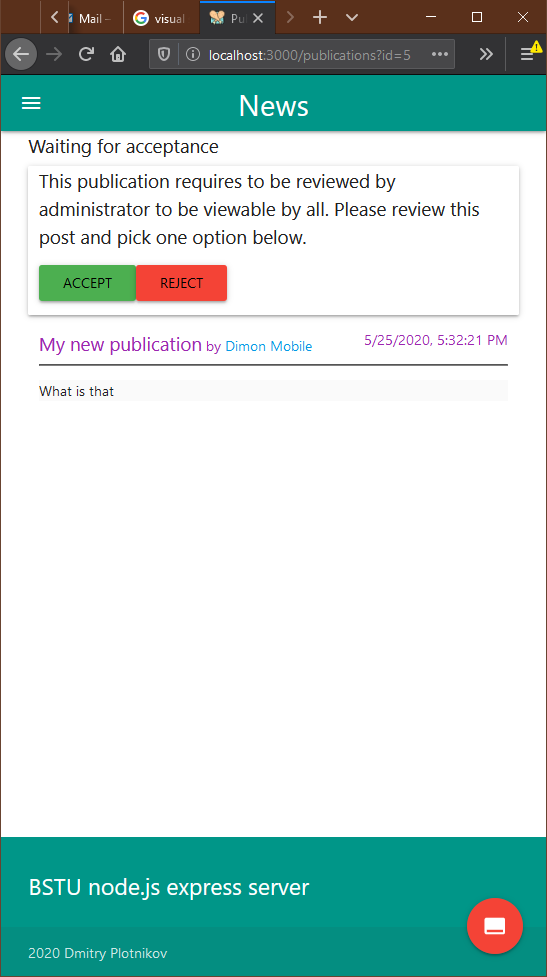


Рисунок 4.11 – Демонстрация страницы одобрения

После подтверждения в материале появляется блок комментариев, а сама новость начинает отображается на главной странице веб-приложения.

# 5. Тестирование, экспериментальные исследования и анализ полученных результатов

## 5.1 Тестирование выводимых системой сообщений

Система корректно обрабатывает ситуации, когда искомые или необходимые данные не были найдены, или же были введены некорректно. Подтверждением может служить вывод сообщений, когда системе не удалось найти логин в базе данных, как это отображено на рисунке 5.1.

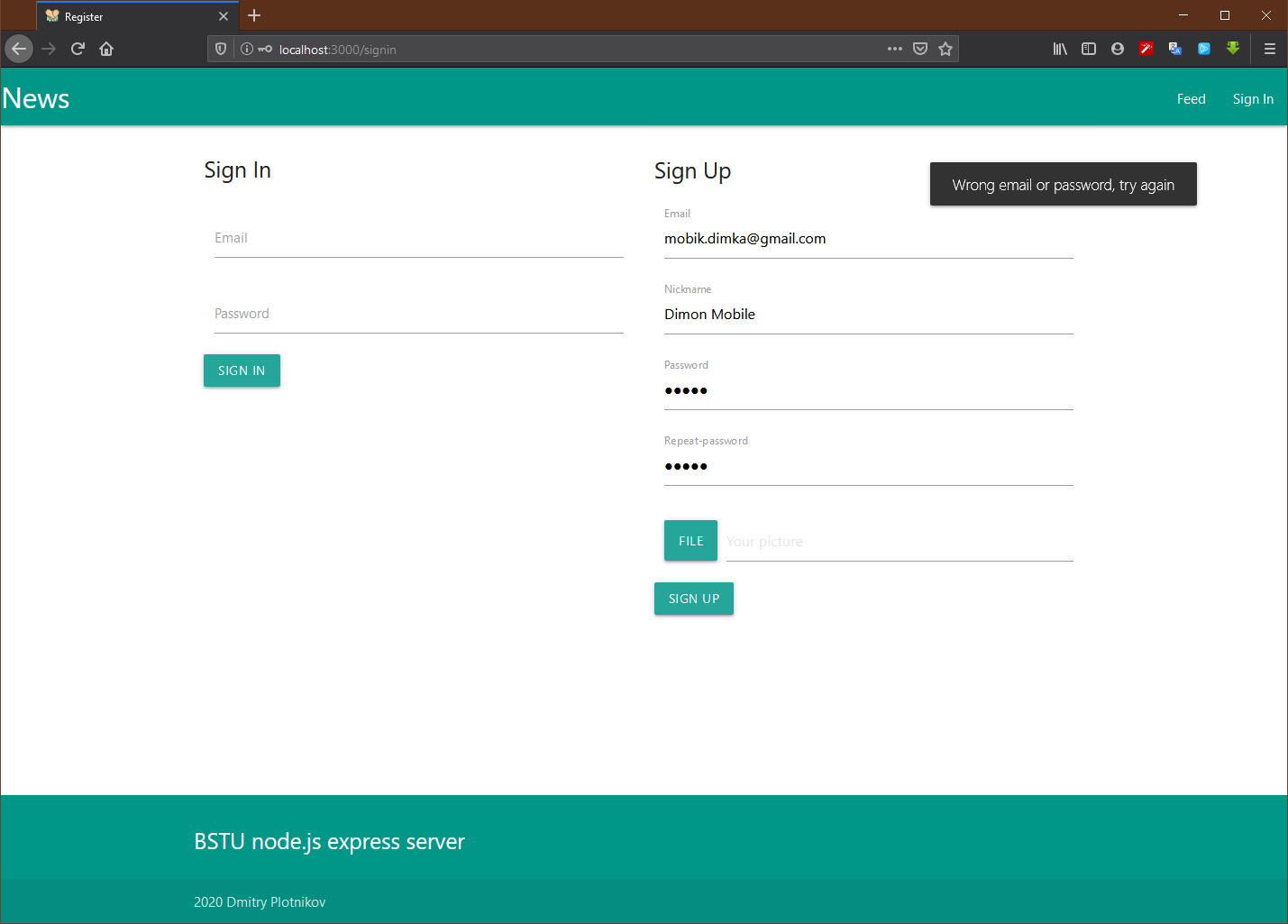


Рисунок 5.1– Предупреждение о неправильно введенном пароле или логине

В случае же если пользователь например попытается зарегистрироваться с электронной почтой, которая уже занята, он тут же увидит сообщение, информирующее его о том, что такое действие невозможно. Убедиться в этом можно по рисунку 5.2.

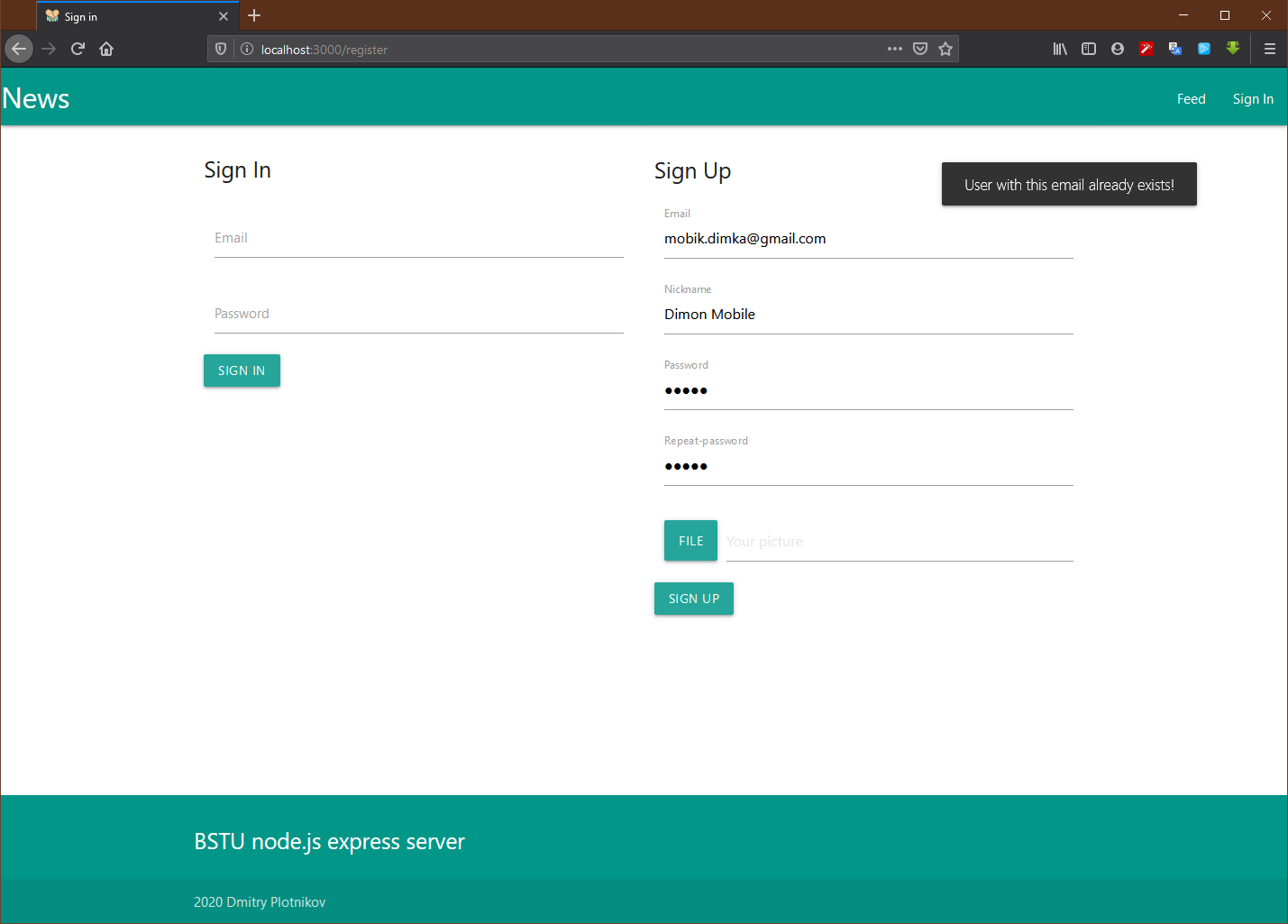


Рисунок 5.2 – Предупреждение о неверной длине комментария

Таким образом можно сделать вывод, что в практически в любой ситуации если что-то пойдет не так пользователь получит вразумительное сообщение, о происшедшем и сможет сделать соответствующие выводы или действия. В случае каких-либо непредвиденных обстоятельств, ошибка будет выведена на отдельной странице, но ее трактовка так же будет понятной.

Такой подход даст возможность пользователю сообщить администратору приложения о происшедшем, что может позволить избежать возникновения таких ошибок в дальнейшем.

## 5.2 Тестирование производительности на больших объемах данных

Для тестирования данного пункта было принято решение выбрать таблицу комментариев. Связано это с тем, что таблица имеет наибольшее число зависимостей, такие как: автор, создавший комментарий, и материал, к которому этот комментарий относится, поля предоставлены на рисунке 5.3.



Рисунок 5.3 – Поля таблицы COMMENTS

Для отображения комментариев используется внутреннее соединение таблиц с таблицей USERS, поля которой предоставлены на рисунке 5.4.

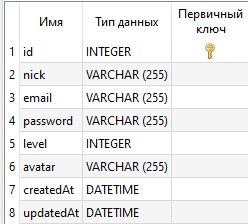


Рисунок 5.4 – Поля таблицы USERS

Вкупе работы с sequelize orm и использованием SQLite получается довольно затратная операция, поэтому ее и было принято решение протестировать. Код функции можно увидеть на рисунке 5.5.

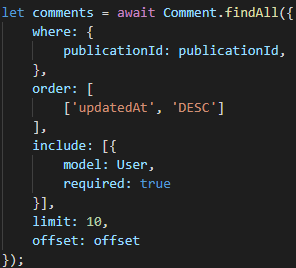


Рисунок 5.5 – Функция, забирающая из базы комментарии

Для проведения тестирования база была наполнена данными (около 600 тыс. строк), после чего выполнялся вызов этой функции. В результате были получены следующие данные:

* В случае, когда в таблице 10 полей, скорость извлечения составила 4мс.;
* В случае наполнения таблицы полумиллионом записи, скорость извлечения увеличилась до 1.5-2с.

Данный результат можно считать приемлемым, особенно учитывая тот факт, что сервер запущен на виртуальной машине с очень ограниченным числом ресурсов. Убедиться в результатах теста можно и воспользовавшись самим приложением, как это показано на рисунке 5.6.

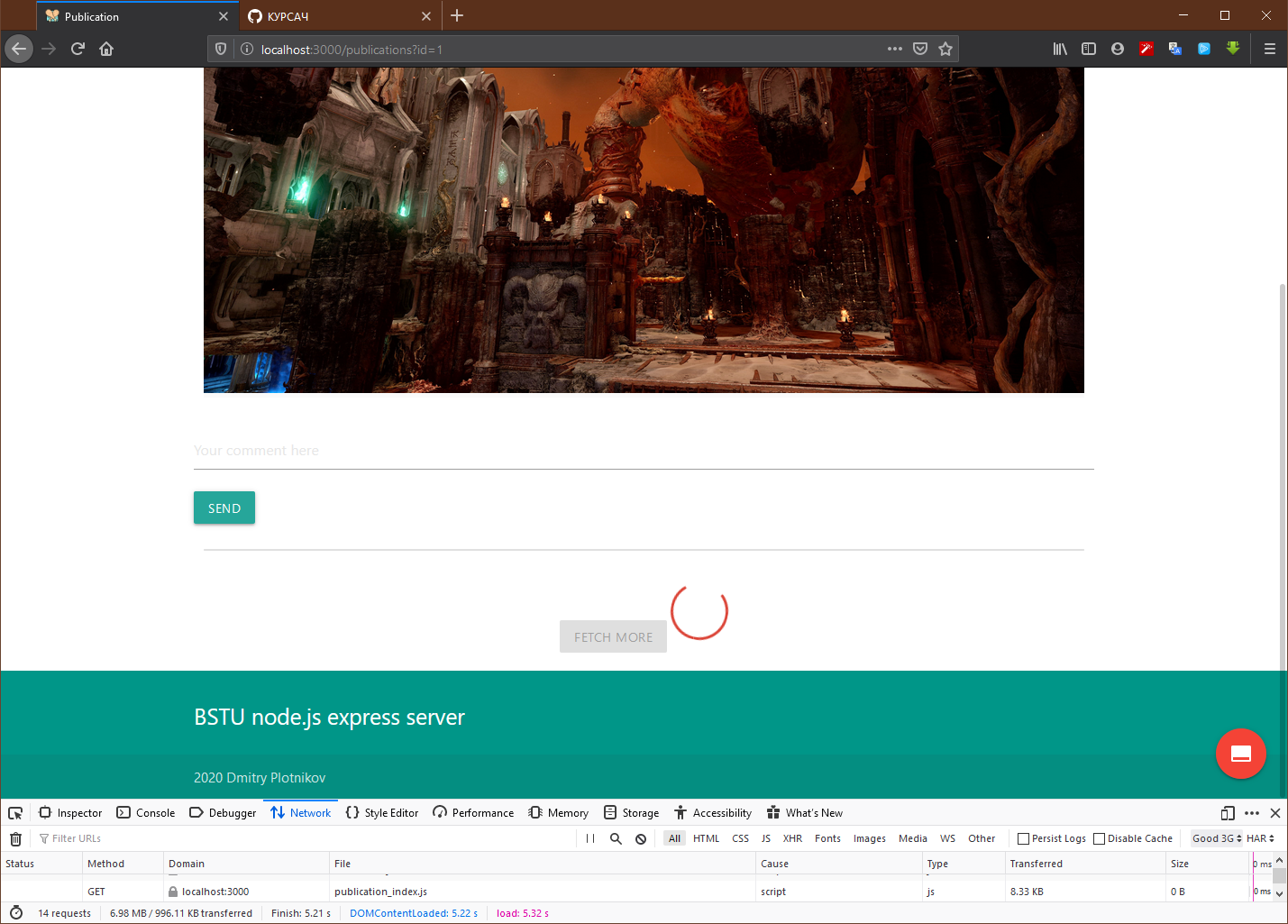


Рисунок 5.6 – Тестирование большого кол-ва комментариев

Индикатор загрузки при полумиллионном количестве комментариев можно наблюдать около 2-х секунд.

## 5.3 Анализ

Результаты тестирования данного веб-приложения показывают, что не было обнаружено таких исключительных ситуаций, при которых приложение самопроизвольно завершает свою работу, не отобразив при этом сообщения о причине исключительной ситуации. Все ошибки невалидного ввода, неправильной логики использования веб-приложения отображаются на экране в понятном для пользователя виде.

Также двухсторонняя валидация и на стороне сервера, и на стороне клиента гарантирует отсутствие в приложении полномасштабных сбоев. В случае же ошибки в логах можно просмотреть полный путь и место возникновения ошибки.

В результате подробного тестирования удалось выявить слабые места и закрыть их. Где-то это были дыры в виде багов валидаторов, а где-то невероятно большой объем времени ожидания выполнения SQL-запросов, что делало бы работу с веб-приложением не просто неудобной, а даже невозможной.

На примере таблицы с комментариями была установлена приблизительная зависимость роста выполнения времени запроса к количеству данных.

В данном случае тестирование помогло:

* в создании лучшей архитектуры: когда часть приложения трудно тестировать, это обычно происходит из-за того, что оно тесно связано с другими частями или функциональность вашего приложения слишком сложна. При их тестировании вам нужно будет сделать их слабосвязанными, применить делегирование и паттерны проектирования, чтобы сделать приложение максимально простым и тестируемым;
* улучшить качество кода: продукт стал менее подвержен сбоям в работе, поскольку тесты помогли написать более надёжный и хороший код, который менее подвержен ошибкам;
* сделали рефакторинг простым и безопасным: создание программного обеспечения — это итеративный процесс. Требования меняются с течением времени, следовательно, меняется и функциональность. Наличие хорошего тестового покрытия позволило модифицировать определённый код, проверяя, что тесты все ещё успешно проходят. Если это не так, делались такие правки в коде таким образом, чтобы тесты прошли.

# Заключение

В самом начале была поставлена задача: реализовать веб-приложение с использованием базы данных и технологии работы с мультимедиа, которое в будущем получило название «Новостная лента». Перед началом были изучены некоторые аспекты SQLite и Node.js, которые и позволили реализовать то, что было задумано изначально.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию веб-приложения «Новостная лента». При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения. Таким образом, была достигнута цель и создано веб-приложение.

В веб-приложении были реализованы следующие функции:

Функционально веб-приложение должно выполнять следующие задачи:

* отображать информацию базы данных;
* работать и редактировать (некоторые) параметры таблиц базы данных: пользователи, новости, сообщения, комментарии, черновики;
* иметь 2 роли с определенной функциональностью;
* отправка сообщений пользователей;
* написание комментариев;
* управление собственными для пользователя черновиками;
* просмотр 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. Официальная документация SQLite - https://www.sqlite.org/docs.html
2. Официальная документация node.js - <https://nodejs.org/en/docs/>
3. Официальная документация express - <https://expressjs.com/en/api.html>
4. Официальная документация Sequelize – https://sequelize.org/v5/

# Приложение

# Листинг моделей

const Sequelize = require('sequelize');

const db = require('../config/db');

let comment = db.define('comment', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

content: {

type: Sequelize.TEXT

}

}, {

indexes: [{

unique: false,

fields: ['userId'] // foreign key

}, {

unique: false,

fields: ['publicationId'] // foreign key

}]

});

module.exports = comment;

const Sequelize = require('sequelize');

const db = require('../config/db');

let message = db.define('message', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

content: {

type: Sequelize.TEXT

},

}, {

indexes: [{

unique: false,

fields: ['fromUserId'] // foreign key

}, {

unique: false,

fields: ['toUserId'] // foreign key

}, {

unique: false,

fields: ['createdAt']

}]

});

module.exports = message;

const Sequelize = require('sequelize');

const db = require('../config/db');

let publication\_draft = db.define('publication\_draft', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

uuid: {

type: Sequelize.UUIDV4

},

title: {

type: Sequelize.STRING,

allowNull: false

},

content: {

type: Sequelize.TEXT

},

}, {

indexes: [{

unique: false,

fields: ['userId'] // foreign key

}, {

unique: true,

fields: ['uuid']

}]

});

module.exports = publication\_draft;

const Sequelize = require('sequelize');

const db = require('../config/db');

let model = db.define('publication', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

name: {

type: Sequelize.STRING,

allowNull: false,

},

content: {

type: Sequelize.TEXT,

allowNull: false

},

published: {

type: Sequelize.BOOLEAN,

allowNull: false,

defaultValue: false

},

reviewed: {

type: Sequelize.BOOLEAN,

allowNull: false,

defaultValue: false

}

}, {

indexes: [{

unique: false,

fields: ['userId']

}]

});

module.exports = model;

const Sequelize = require('sequelize');

const db = require('../config/db');

let user = db.define('user', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

nick: {

type: Sequelize.STRING,

allowNull: false,

},

email: {

type: Sequelize.STRING,

allowNull: false,

},

password: {

type: Sequelize.STRING,

allowNull: false

},

level: {

type: Sequelize.INTEGER,

allowNull: false,

defaultValue: 1

},

avatar: {

type: Sequelize.STRING,

allowNull: true,

}

}, {

indexes: [{

unique: true,

fields: ['email']

}]

});

module.exports = user;

const PublicationDraft = require('./publication\_draft');

const User = require('./user');

const Publication = require('./publication');

const Comment = require('./comment');

const Message = require('./message');

// Drafts

User.hasMany(PublicationDraft, {

foreignKey: 'userId'

});

//PublicationDraft.belongsTo(User);

// Publications

Publication.belongsTo(User, {

foreignKey: 'userId',

onDelete: 'CASCADE',

onUpdate: 'CASCADE',

});

Comment.belongsTo(User, {

foreignKey: 'userId',

onDelete: 'CASCADE',

onUpdate: 'CASCADE',

});

Comment.belongsTo(Publication, {

foreignKey: 'publicationId',

onDelete: 'CASCADE',

onUpdate: 'CASCADE',

});

Message.belongsTo(User, {

foreignKey: 'fromUserId',

as: 'sourceUser'

});

Message.belongsTo(User, {

foreignKey: 'toUserId',

as: 'targetUser'

});

// Exports

exports.PublicationDraft = PublicationDraft;

exports.User = User;

exports.Publication = Publication;

exports.Comment = Comment;

exports.Message = Message;