Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А.Замышляева
«____»_____2018 г.

Разработка новостного агрегатора с рекомендательной системой

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2017.97.ПЗ ВКР

Руководитель работы, доцент
_____/Т.Ю. Оленчикова
«____»_____2018 г.

Автор работы
Студент группы ЕТ-414
_____/ И.А.Марченко
«____»_____2018 г.

Нормоконтролер, доцент
_____/Т.Ю. Оленчикова
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Марченко И.А. Разработка новостного агрегатора с рекомендательной системой – Челябинск: ЮУрГУ, ЕТ-414, 37 с., 30 ил., 6 табл., библиогр. список – 17 наим., 2 прил.

Данная работа посвящена разработке новостного агрегатора, способного с помощью анализа профиля пользователя и его активности подбирать новости, которые могут быть интересны.

В работе выполнен обзор существующих приложений подобного типа, наиболее востребованных средств разработки серверных приложений и систем управления базами данных.

Разработано веб-приложение, база данных и алгоритм для подбора интересующих новостей. Реализован графический интерфейс, база данных для их хранения. Также был проведен анализ предметной области, рассмотрены различные логистические концепции.

Программа реализована на языке программирования JavaScript с использованием нереляционной СУБД MongoDB. В приложении приведены описание и текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время вокруг нас находится огромное количество самой разной информации. Ее настолько много, что самолично разбирать огромное множество статей в поисках интересных новостей может занять огромное количество времени. Потому в ходе своей дипломной работы было выбрано решение о создании сайта - новостного агрегатора, который на основе анализа данных активности каждого пользователя, мог бы подбирать интересные для него новости.

Сайт будет сам анализировать профиль и активность каждого пользователя и отображать все новости, которые могут быть ему интересны. Доступ же к сайту могут иметь все пользователи, у которых есть компьютер с доступом в интернет.

Работа посвящена разработке сайта агрегатора новостей. Проект состоит из двух частей – серверной части и самого сайта.

Первая глава посвящена разработке требований к системе. Рассматриваются различные методы и подходы к разработке веб-приложений.

Во второй главе приведена разработка архитектуры системы в целом, приведена диаграмма устройства базы данных.

Третья глава посвящена разработке NoSQL базы данных системы, состоящая из концептуального проектирования.

И наконец, в четвертой главе приведена реализация системы, включающая в себя описание алгоритмов, а также отладку и тестирование системы.

1 ОБЗОР ТЕХНОЛОГИЙ СОЗДАНИЯ НОВОСТНОГО АГРЕГАТОРА ДЛЯ ПОЛЬЗОВАТЕЛЕЙ

1.1 Анализ требований к приложению

Целью данной работы является создание сайта новостного агрегатора с использование рекомендательной системы.

Создаваемый ресурс упростит и ускорит получение полезной информации для пользователя на основе его предпочтений.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать требования к приложению;
- изучить и проанализировать современные технологии и инструментарий; для создания сайтов и рекомендательной системы.
- разработать архитектуру его серверной части;
- разработать необходимые алгоритмы для подбора новостей;
- разработать рекомендательную систему;
- разработать NoSQL базу данных;
- реализовать и отладить программу.

1.2 Существующие новостные агрегаторы

На данный момент существуют различные новостные агрегаторы. Для сравнения разберем два наиболее похожих: Гугл Новости и Яндекс Новости.
Оба сайта существуют достаточно давно и имеют большую аудиторию, но при этом совершенно разные принципы работы. (см рисунки 1.1 и 1.2).

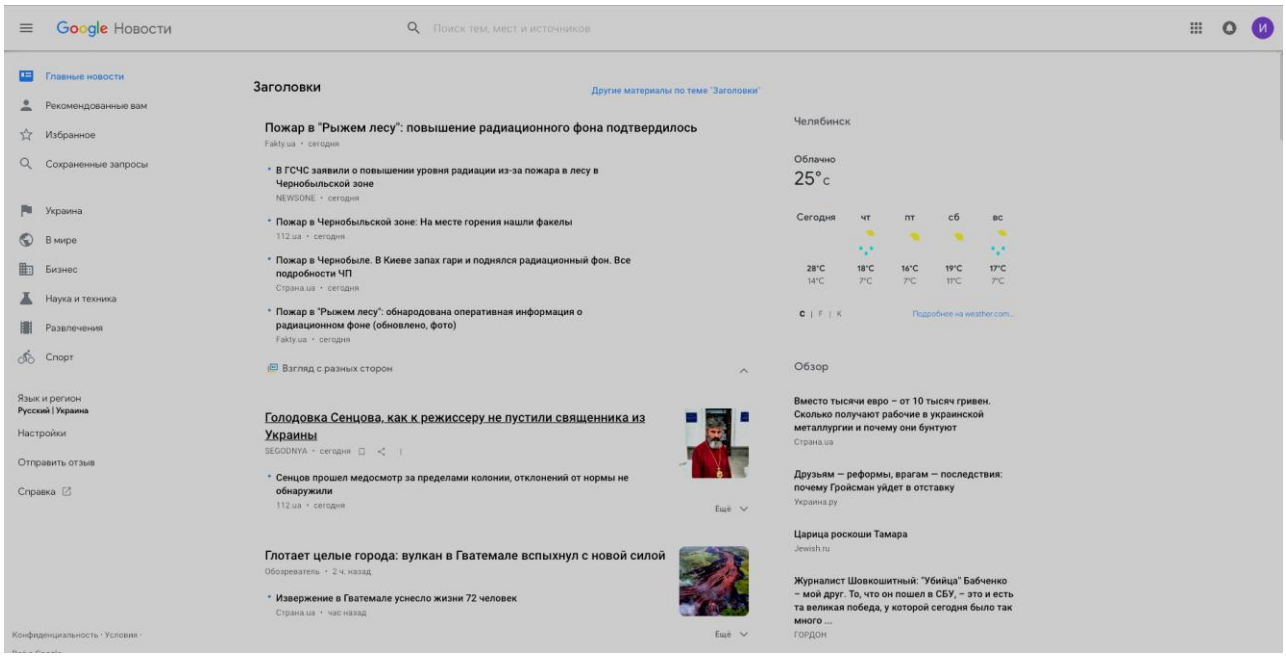

Рисунок 1.1 – Интерфейс Гугл Новости

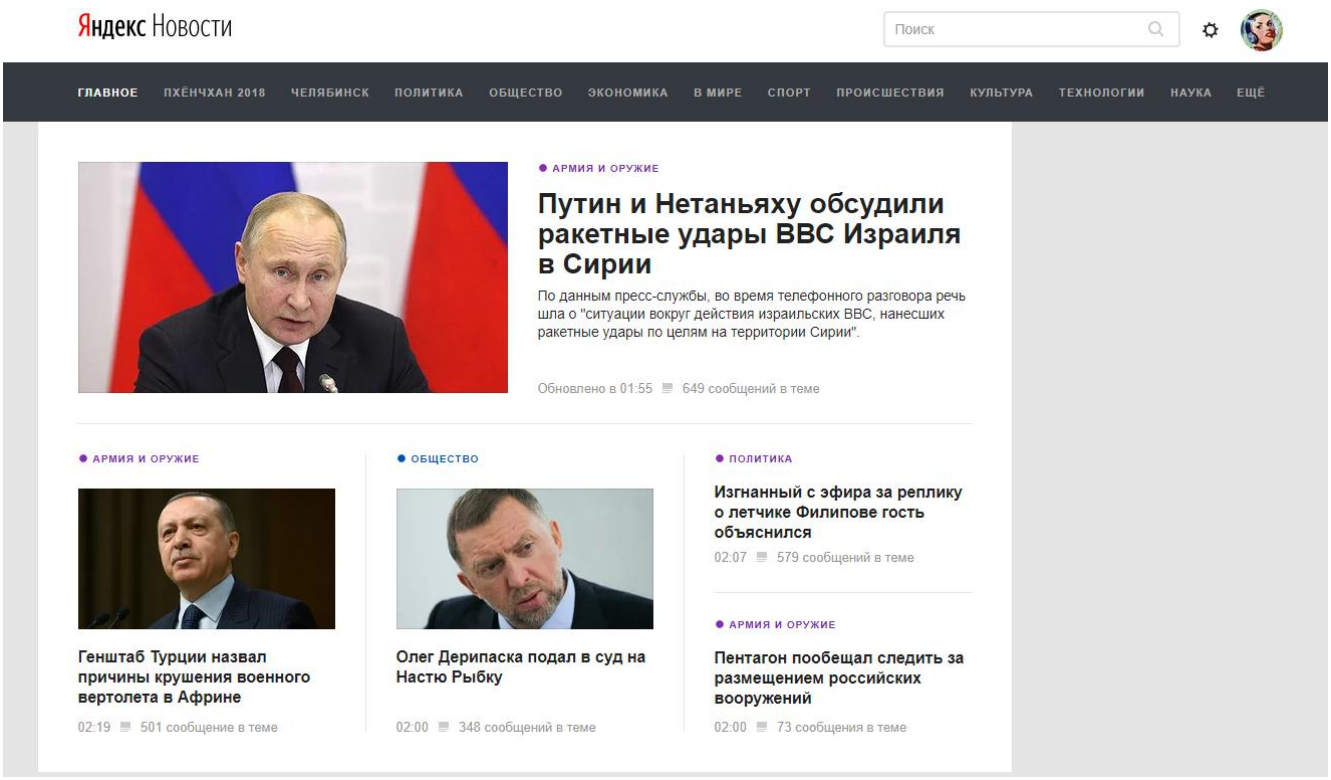

Рисунок 1.2 – Интерфейс Яндекс Новости

Первый агрегатор, Гугл Новости, разделяет новости на категории и приводит подборки новостей с одного сайта, объединяя их в блоки. А также показывает в приоритете новости на основе ваших запросов в гугле.

Второй агрегатор, Яндекс Новости, самый популярный в РФ на данный момент. Он не имеет рекомендательной системы для разных пользователей, но имеет свой рейтинг сайтов, распределяющий новости и их попадание на главную.

Разрабатываемый агрегатор будет отличаться от первого конкурента алгоритмом для подбора интересующих новостей, а второй же не имеет персональных подборок на момент написания данной работы.

1.3 Требования к программе или программному изделию

1.3.1 Требования к функциональным характеристикам

Система должна представлять совокупность методических и программных средств решения следующих задач:

- индивидуальные подбора интересующих новостей;
- ввод новой информации в БД;
- изменение существующей информации в БД;
- быстрый анализ БД для построения рекомендации;
- удаление информации из БД;
- осуществлять кластеризацию информации и выводить ее в соответствии с запросами пользователя.

1.3.2 Требования к надежности

Основные требования к надежности следующие:

− обеспечение сохранности и конфиденциальности информации, хранящейся на сервере;

− обеспечение корректного вывода информации с сервера;

− обеспечение своевременного обновления информации на сервере.

1.3.3 Требования к составу и параметрам технических средств

Система должна работать на ПК с возможностью выхода в интернет через приложение браузера. Технические характеристики компьютера:

− компьютер на базе процессора Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2;

− оперативная память не менее 400 Мб;

− свободная память на жестком диске не менее 512 Мб.

1.3.4 Требования к информационной и программной совместимости

Для работы необходим персональный компьютер с наличием ОС Windows или MacOS

1.3.5 Распределение функций в Web-приложении

Разработка Web-приложений зачастую трудоемкий и сложный процесс, в котором приходится много решать, что же положить на плечи клиентской части, а что оставить решать серверу [17].

Решение этой задачи зачастую приводит к появлению двух или даже трех звеньев между клиентом и сервером.

В основе Интернета находится классическая схема «клиент-сервер». В базовом виде она реализуется так:

− клиент должен формировать и отправлять запрос на сервер, где расположены базы данных;

− сервер по запросу производит необходимую обработку данных, компонует результат и отправляет его обратно клиенту;

− получив результат, клиент отображает его на используемом устройстве вывода данных, после чего переходит в режим ожидания следующих действий пользователя.

Этот цикл непрерывно повторяется до тех пор, пока пользователь не завершит работу с сервером.

В HTTP осуществляются следующие транзакции.

1. Браузер производит декодировку первой части URL, после чего устанавливается соединение с сервером.
2. На сервер передается остальная часть URL.
3. Сервер согласно URL определяет путь и имя файла.

4. Сервер осуществляет пересылку указанного файла браузеру.

5. Сервер прерывает соединение.

6. Браузер отображает документ.

В сервисе WWW для передачи информации применяется протокол HTTP (см. рисунок 1.3).

В процессе осуществления этих транзакций сервер не получает никакой информации о состоянии браузера, т.е. HTTP можно назвать протоколом «с одним направлением», позволяющим осуществлять взаимодействие с сервером только через механизм URL, что порождает сложности при реализации клиентской составляющей (см. рисунок 1.4).

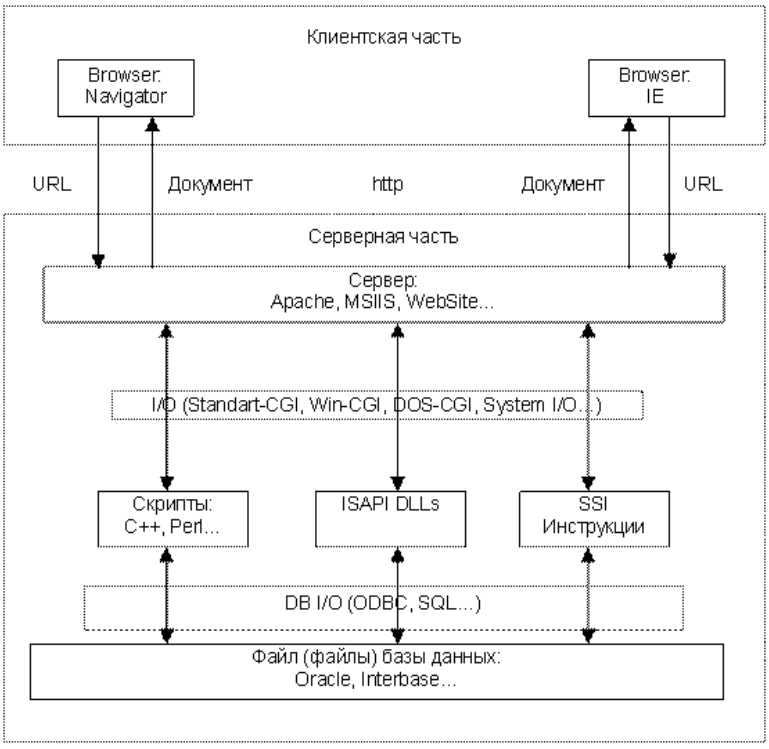

Рисунок 1.3 – Схема «клиент-сервер» WWW-HTTP

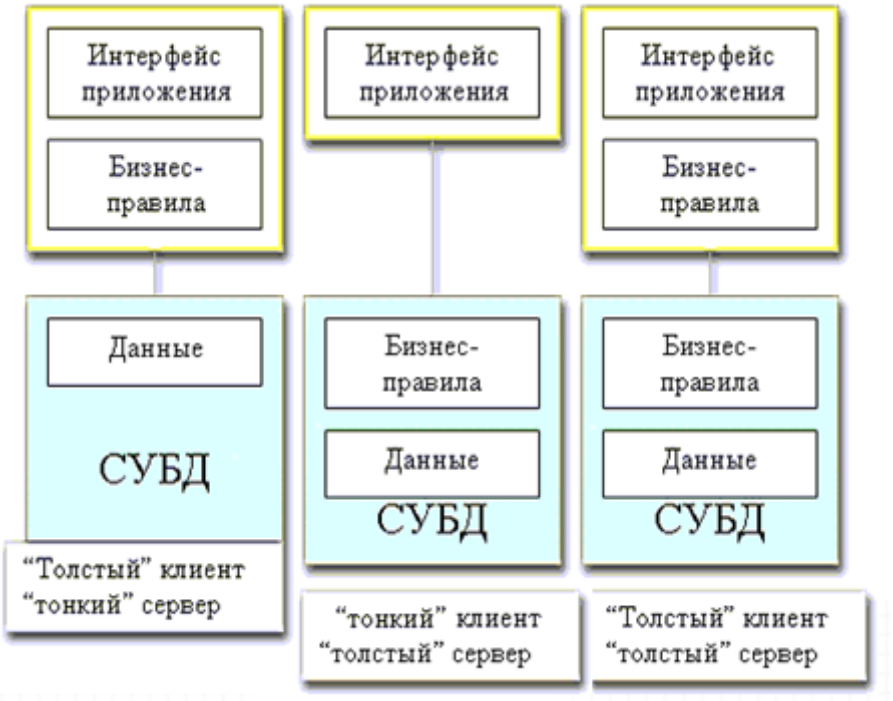

Рисунок 1.4 – Распределение функций в архитектуре «клиент-сервер»

Тенденция в технологиях «клиент-сервер» связана с активным ростом использования различных распределенных вычислений. Для реализации таких вычислений применяется модель сервера приложений. В этой модели сетевое приложение разделяется несколько частей, при этом любая из этих частей способна выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, происходит обмен информацией в формате, который согласован заранее. И тогда двухзвенная клиент-серверная архитектура превращается в трехзвенную, или three-tier.

Как правило, роль третьего звена в такой архитектуре выполняет сервер приложений, и составляющие распределяются следующим образом:

− представление данных происходит на стороне клиента;

− прикладной компонент находится на специально выделенном сервере приложений (или, например, выполняющем функции промежуточного программного обеспечения);

− управление ресурсами происходите на сервере базы данных, который и представляет запрашиваемые данные.

Изолированность звеньев друг от друга позволяет более точно сконфигурировать приложение.

Более того, если выделять дополнительные сервера, каждый из которых способен представлять собственные сервисы, то трехзвенная архитектура может быть расширена до многозвенной (см рисунок 1.7).

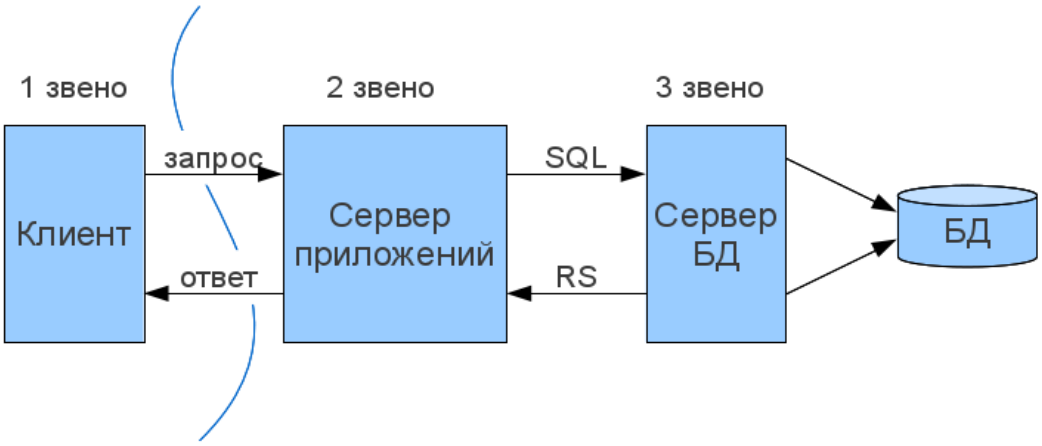

Рисунок 1.6 – Трехзвенная архитектура приложения

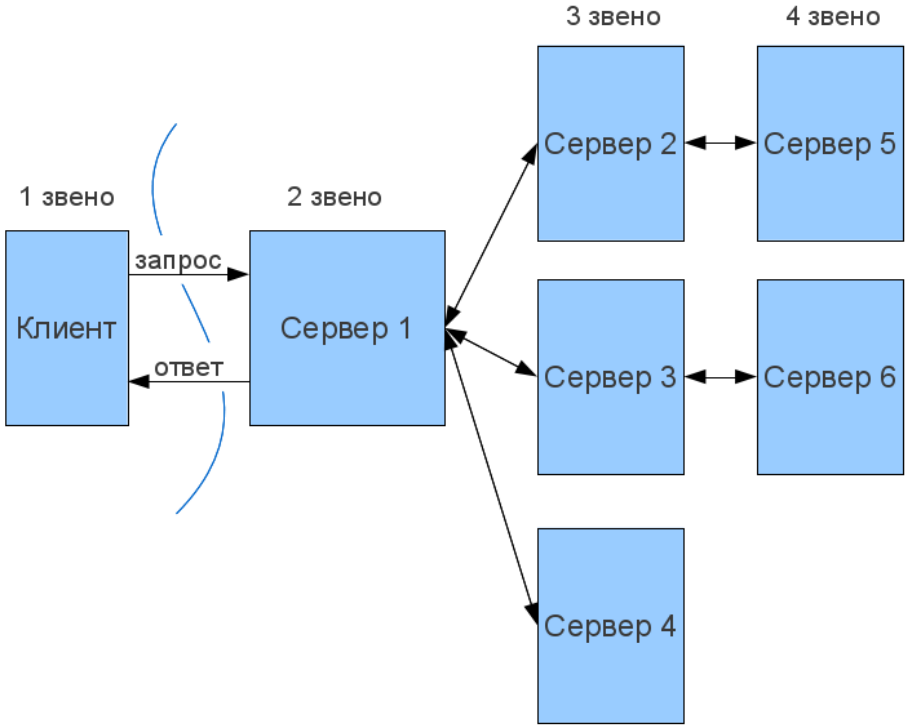

Рисунок 1.7 – Пример многозвенной архитектуры приложения

Для разработки приложения была выбрана трехзвенная архитектура с использованием NoSQL. NoSQL обеспечит необходимую скорость работы с данными, которые будут обрабатываться во время парсинга новостей и извлечения их для пользователя.

1.4 Инструментарий разработки

Создание Web-приложений для сервера создается на различных языках и с помощью различных технологий. Список языков и их лицензирования приведен в таблице 1.1.

Таблица 1.1

Инструментарий для веб-приложений

Название	**Лицензия**	**Веб-сервер**
ASP	проприетарная	специализированный
ASP.NET	проприетарная	специализированный
ASP.NET vNext	Apache 2	практически любой
C/C++	свободная	практически любой
Java	свободная	множество, в том числе свободных
Node.js	MIT License	собственный
Perl	свободная	практически любой
PHP	свободная	практически любой
Python	свободная	практически любой
Ruby	свободная	практически любой

Рассмотрим три наиболее популярных языка программирования, применяемых в веб-разработке: PHP, Ruby и Node.js, а также сравним SQL и NoSQL базы данных и рассмотрим несколько последних.

1.4.1 Язык PHP

PHP (Hypertext PreProcessor) – скриптовый язык общего назначения, исполняемый на стороне веб-сервера, созданный Расмусом Лердорфом в качестве инструмента создания динамических и интерактивных веб-сайтов [1].

В настоящее время является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

Преимущества PHP:

− свободное программное обеспечение, распространяемое под особой лицензией (PHP license);

− легок для освоения;

− имеет огромную популярность и широкий круг пользователей и разработчиков, потому активно поддерживается и развивается;

− сильно развита поддержка БД;

− огромное кол-во расширений и библиотек;

− пригоден для работы в изолированной среде.

− обладает программным интерфейсом расширений и средствами для организации веб-сессий;

− разворачивается практически на любом сервере;
− поддерживает все популярные ОС на сегодняшний день.

Недостатки PHP:

− непригоден для системных и desktop приложений;
− плохо работает с исключениями из-за ограниченности средств;
− настройка сервера и развертывание приложений затруднены из-за того, что на базовый синтаксис языка оказывают влияние глобальные параметры конфигурации;
− в PHP передача объектов происходит по значению, к чему приходится приспосабливаться основной массе программистов, поскольку в большинстве других языков объекты передаются по ссылке;
− у web-приложений, написанных на данном языке, низкий уровень безопасности.

PHP применяется в множестве известных проектов, в числе которых есть Yahoo, Google и Facebook.

1.4.2 Язык Ruby

Ruby – объектно-ориентированный язык программирования, разработанный Юкихиро Матсумото. Ruby создан под влиянием таких языков, как Perl, Eiffel и Smalltalk [2].

Он характеризуется динамической типизацией и автоматическим управлением памятью. Для веба-приложений применяется фрейворк Rails.

Преимущества Ruby:

− открытая разработка;
− Ruby прост на вид, но очень сложен внутри, подобно человеческому телу;
− поддерживается большинством современных платформ;
− можно внедрять HTML-разметку;
− полностью объектно-ориентированный язык программирования;
− имеет в наличии продвинутые методы для манипуляций со строками и текстом;
− легко интегрировать высокопроизводительные серверы баз данных;
− просто в освоении;
− при помощи Ruby легко создавать многопоточные приложения благодаря простому интерфейсу;
− расширяется с помощью библиотек, написанных на языке C.
− имеет свой собственный отладчик;
− высокий уровень безопасности.

Недостатки Ruby:

− очень сложен переход с начального уровня на более высокие из-за недостаточного кол-ва качественной информации;
− низкий уровень производительности, относительно других языков web-программирования;
− медленное развитие.

Ruby применяется в GitHub и Indiegogo.

1.4.3 Платформа Node.js

Node.js - это программная платформа движке V8 для языка JavaScript, созданная Райаном Далем, которая предназначена для создания масштабируемых распределённых сетевых приложений, таких как веб-сервер. В отличие других приложений на JS данный “каркас” исполняется не в браузере, а на сервере.[3]

Преимущества Node.js:

- высокий уровень интеграции для работы с NoSQL базами данных;
- высокая масштабируемость, позволяющую справляться с большим потоком трафика;
- отлично справляется с построением легковесных REST / JSON интерфейсов;
- из пункта выше вытекает, что Node отличный вариантом для написания обертки вокруг базы данных или веб-сервиса, которая общается с клиентом в формате JSON;
- отлично справляется с обработкой огромного кол-ва запросов и имеет низкое время отклика;
- отлично подходит для системы мягкого реального времени;
- огромное сообщество, которое растет с гигантской скоростью;
- множество документаций и руководств.

Недостатки Node.js:

- высокий порог входа, относительно конкурентов;
- проблемная реализация тяжелых математических вычислений;
- из-за событийно-ориентированной природы довольно сложно на Node.js делать вещи, требующие последовательного выполнения.

Node.js применяется в Ebay,PayPal,Google,Netflix,Uber,LinkedIn,VK.

1.4.4 Реляционные базы данных

Реляционные базы данных – это набор данных с предопределенными связями между ними. Для работы с реляционными БД используется стандартный язык SQL, предназначенный для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

1.4.5 NoSQL

Термин NoSQL появился в 2009 году стихийно, он не имеет научного определения за спиной, а расшифровать его можно как "Not Only SQL”. Это

название – характеристика вектора развития IT в сторону от реляционных баз данных [5].

Общих характеристик у NoSQL баз данных не очень много, а под этим названием скрывается множество разнородных систем.

Особенности:

− share nothing. Для повышения производительности при масштабировании или обработке данных достаточно добавить новый сервер – кластер. Процедурами шардинга, репликации, обеспечением отказоустойчивости (результат будет получен даже если одна или несколько серверов перестали отвечать), перераспределения данных в случае добавления узла сети занимается сама NoSQL база;

− используют query languages, который хоть и похож на SQL, но полностью реализовать данный язык не удалось;

− слабая типизация; в отдельной строке или документе можно добавить произвольное поле без предварительного декларативного изменения структуры всей таблицы;

− представления в виде агрегатов; в отличие от реляционных БД, где есть физические таблицы, NoSQL оперирует ими как сущностями (см. рисунок 1.8);

− слабые ACID свойства; ими зачастую пренебрегают, чтобы добиться низкого времени отклика и обработки команд при огромном количестве данных;

− неограниченная горизонтальная масштабируемость.

Популярность NoSQL БД растет огромными темпами, и уже большинство крупнейших корпораций мира используют их. Это не означает, что реляционные базы данных уходят. Скорее всего, в будущем эти две модели будут всегда использоваться вместе, чтобы обеспечить высокие скорости работы и иметь высокую стабильность данных [4].

На данный момент существует огромное количество NoSQL баз, которые активно развиваются (самый большой список можно найти на http://nosql-database.org/). Многие теоретики и практики создавали свои собственные классификации, но наиболее простой и общеупотребительной можно считать систему, основанную на используемой модели данных, предложенной Риком Кейтелем.

1) **Хранилища ключ-значение**. Отличительной особенностью является простая модель данных – ассоциативный массив или словарь, позволяющий работать с данными по ключу. Основная задача подобных хранилищ – максимальная производительность, поэтому никакая информация о структуре значений не сохраняется.
2) **Документо-ориентированные хранилища**. Модель данных подобных хранилищ позволяет объединять множество пар ключ-значение в абстракцию, называемую «документ». Документы могут иметь вложенную структуру и объединяться в коллекции. Однако это скорее удобный способ

логического объединения, т.к. никакой жесткой схемы у документов нет и множества пар ключ- значение, даже в рамках одной коллекции, могут быть абсолютно произвольными. Работа с документами производится по ключу, однако существуют решения, позволяющие осуществлять запросы по значениям атрибутов.

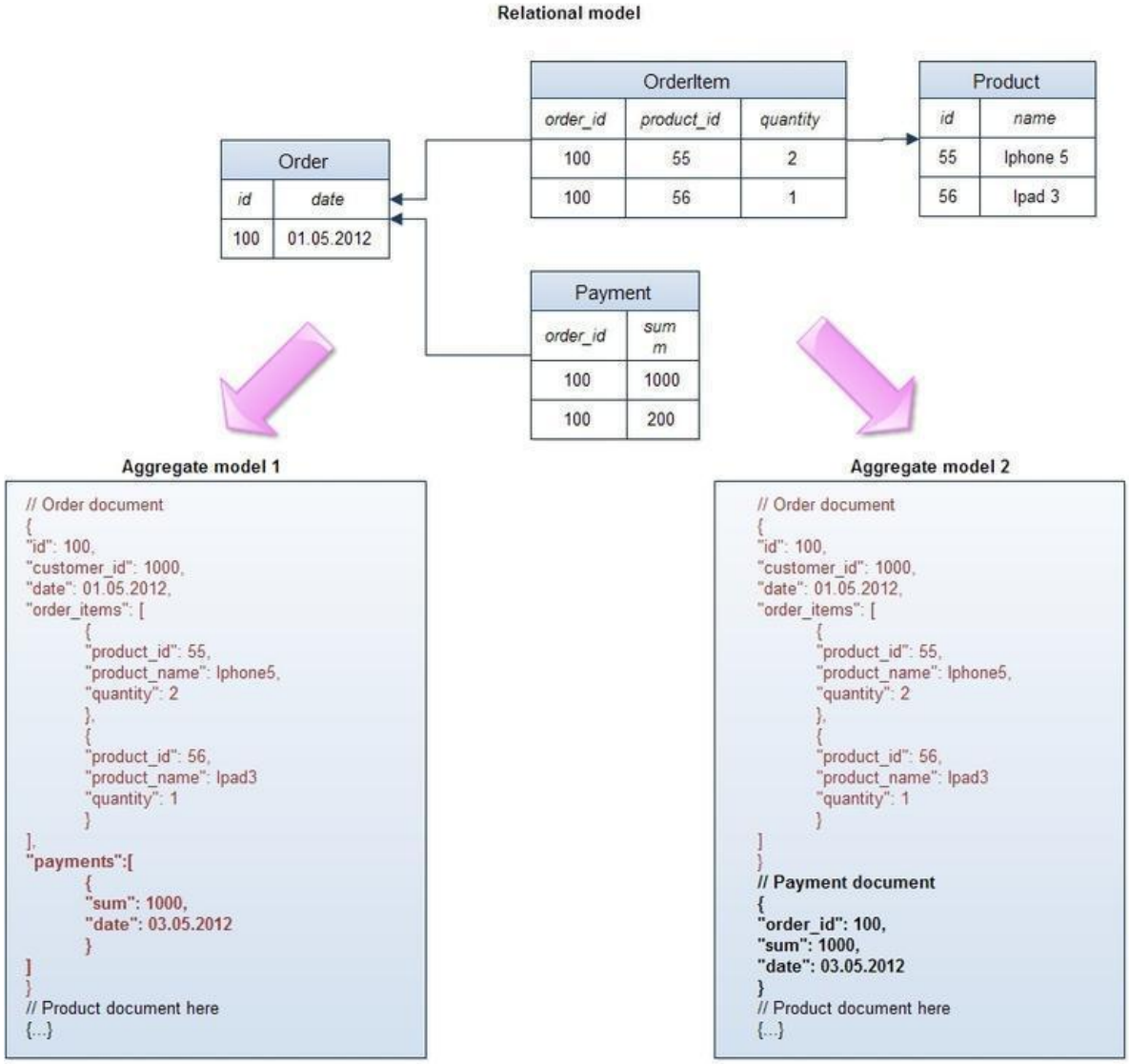

Рисунок 1.8 – Представления данных в виде агрегатов

3) **Колоночные хранилища.** Этот тип кажется наиболее схожим с традиционными реляционными СУБД. Модель данных хранилищ подобного типа подразумевает хранение значений как неинтерпретируемых байтовых массивов, адресуемых кортежами <ключ строки, ключ столбца, метка времени>. Основой модели данных является колонка, число колонок для одной таблицы может быть неограниченным. Колонки по ключам объединяются в семейства, обладающие определенным набором свойств.
4) **Хранилища на графах**. Подобные хранилища применяются для работы с данными, которые естественным образом представляются графами (напр.

социальная сеть). Модель данных состоит из вершин, ребер и свойств. Работа с данными осуществляется путем обхода графа по ребрам с заданными свойствами.

В ходе обзора было принято решение остановиться на документно-ориентированном хранилище, в частности на MongoDB, т.к. она полностью подходит для решения поставленной задачи, а также имеет хорошую интеграцию с NodeJS.

Таблица 1.2.

Классификация NoSQL хранилищ по модели данных

Тип	**Проекты**
Хранилища ключ-значение	Redis Scalaris Tokio Tyrant Voldemort Riak
Документо-ориентированные хранилища	SimpleDB CouchDB MongoDB
Колоночные хранилища	Bigtable HBase HyperTable Cassandra
Хранилища на графах	Neo4j

1.5 ReactJS

ReactJS – фреймворк от Facebook для создания пользовательских интерфейсов [14].

В основе данной библиотеки лежит идея создания отдельных блоков кода – компонентов.

Каждый такой блок обладает определенной степенью независимости, что позволяет повторно использовать компоненты в разных частях программы. Основным принципом является композиция компонентов. Компоненты, написанные разными разработчиками должны работать вместе, не требуя изменений в коде.

Среди главных особенностей данной библиотеки также можно выделить следующее [16].

1) JSX (JavaScript Syntactic Extension) – это расширение синтаксиса JavaScript, которое выглядит подобно XML, синтаксически повторяющего HTML.

2) Состояние (state) и свойства (props) компонента. Компоненты обладают некоторым локальным состоянием и свойствами, переданными от компонентов – родителей. Состояние как правило содержит данные, относящиеся только к данному компоненту, например, состояние интерфейса. Свойства же содержат данные, переданные от других компонентов, что позволяет наладить взаимодействие между ними. Изменения в состоянии или свойствах вызывают повторное отображение компонента(re-rendering).
3) Виртуальный DOM (Domain Object Model). HTML код возвращаемый компонентами не изменяет интерфейс сразу. Вместо этого изменения применяются к хранимой в память копии. Далее запускается алгоритм согласования (reconcilation), в процессе которого копия сравнивается с реальным DOM'ом и при необходимости вызывается отрисовка определенных узлов. Работа с виртуальным DOM'ом быстрее чем отрисовка элементов непосредственно браузером, что значительно повышает производительность. С другой стороны, хранение копии требует дополнительной оперативной памяти, что является главным недостатком данной библиотеки.
4) Методы жизненного цикла (lifecycle methods). Все компоненты наследуются от абстрактного класса Component, обладающего набором методов, которые можно переопределить, чтобы контролировать поведение компонента в определенные моменты времени.

1.6 Вывод по разделу

В данном разделе выполнен анализ требований к приложению, включающий в себя:

- требования к функциональным характеристикам;
- требования к надежности;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к программной документации.

Проведен обзор существующих языков и платформ для написания Web-приложений. Также рассмотрено распределение функций в Web-приложении, клиент-серверная организация сети Интернет, протокол HTTP и основные транзакции в этом протоколе, и типы архитектур «клиент-сервер».

Рассмотрены типы баз данных и рассмотрены наиболее популярные NoSQL решения. В ходе обзора было принято решение остановиться на MongoDB.

2 РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ

2.1 Диаграмма прецедентов

Для построения диаграмм архитектуры использовался язык UML (Unified Model Language) и приложение StarUML. Данный язык очень удобен для представления общих понятий, таких как, класс, обобщение, поведение [6].

На рисунке 2.1 приведена диаграмма вариантов использования (или диаграмма прецедентов) системы в целом, раскрывающая взаимодействия в системе и возможности для пользователя.

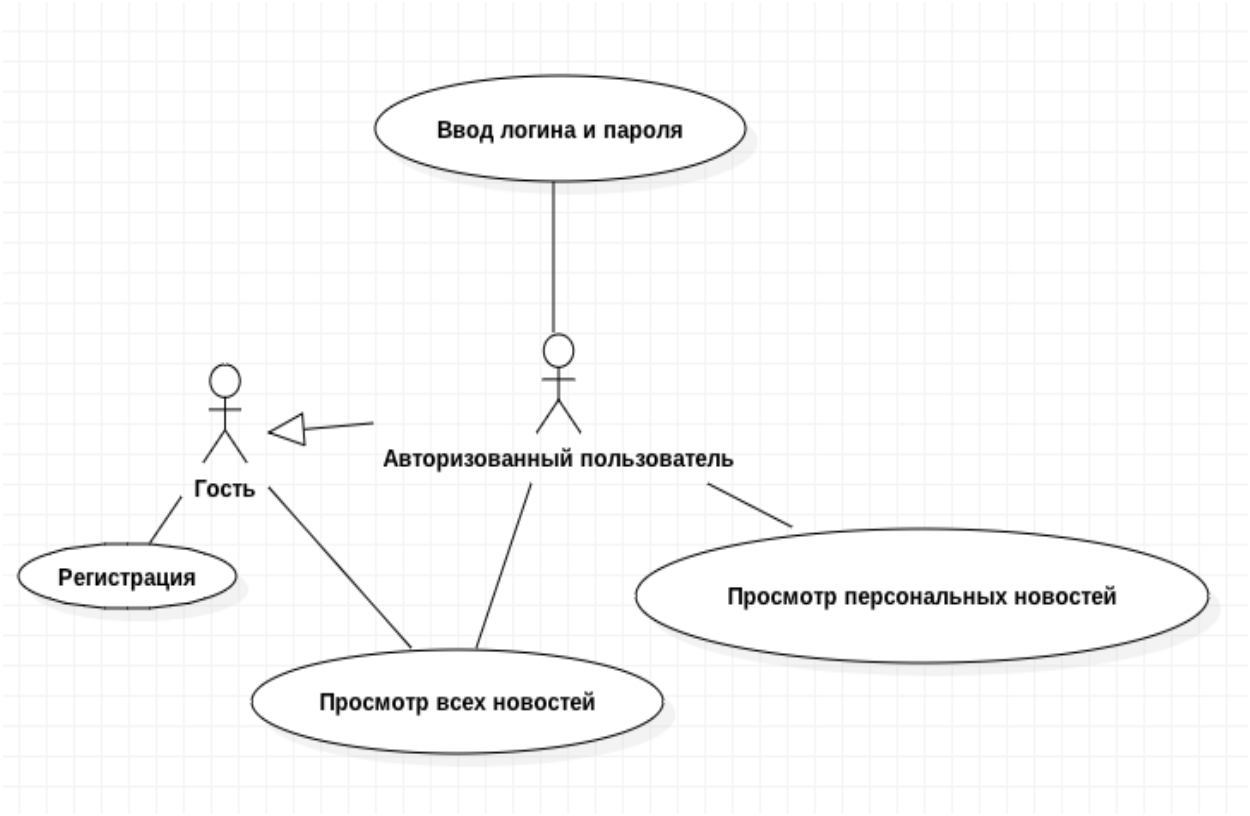

Рисунок 2.1 – Диаграмма прецедентов системы в целом

2.2 Диаграмма размещения

UML используется для абстрактного описания модели системы. Диаграмма размещения (см. рисунок.2.2) отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать размещение объектов и компонентов в распределенной системе.

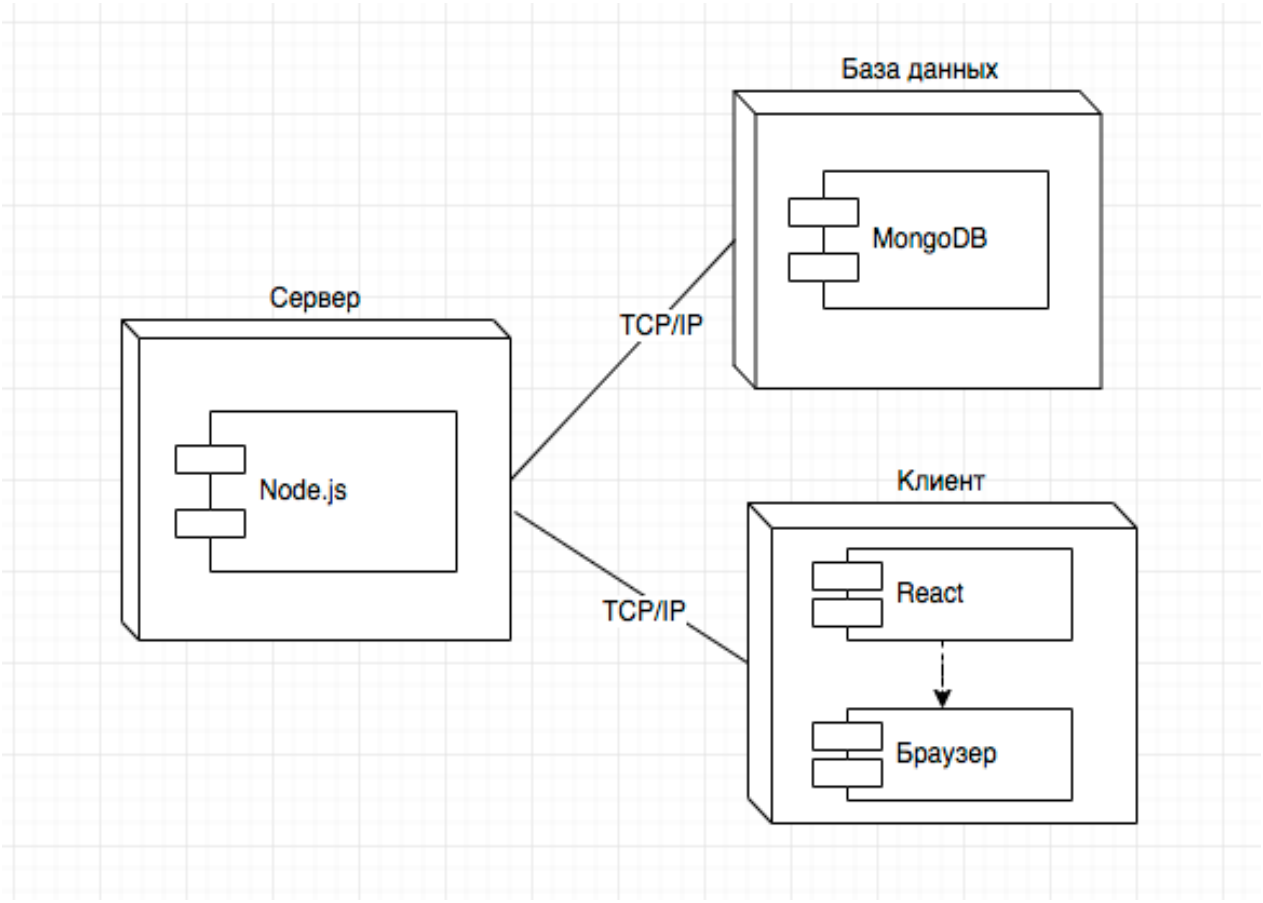

Рисунок 2.2 – Диаграмма размещения

Компонент – физически существующая часть системы. Компоненты могут включать в себя реализацию класса, реализацию функциональных возможностей системы и др. Диаграмма компонентов (см. рисунок 2.3) служит для представления системы в виде структурных компонентов связей и зависимости между ними. Таким образом в данной работе она будет разделена на две части: диаграмма для серверных компонентов (см. рисунок 2.3) и клиентских (см. рисунок 2.4)

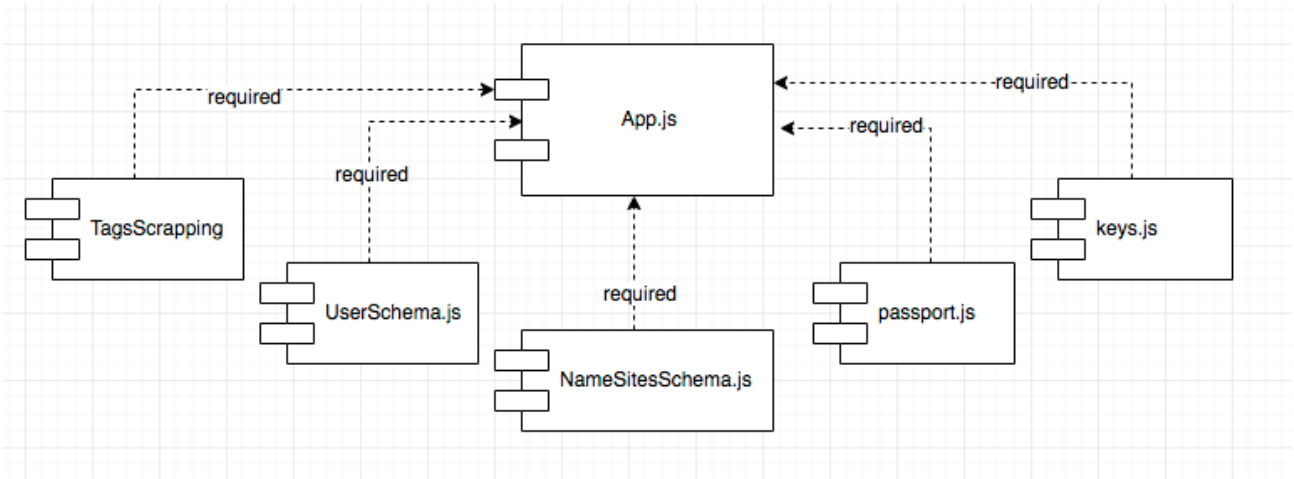

Рисунок 2.3 – Диаграмма серверных компонентов

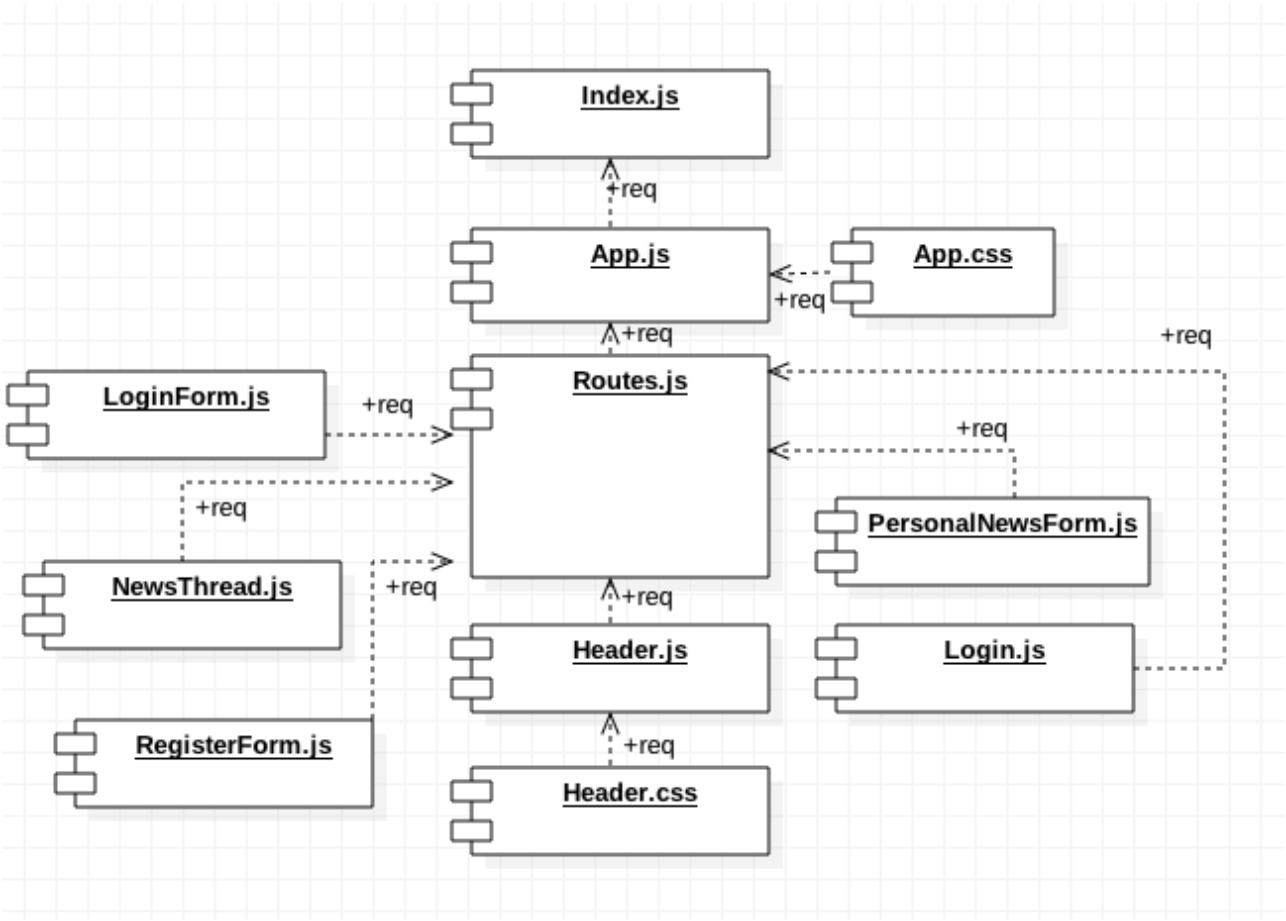

Рисунок 2.4 – Диаграмма клиентских компонентов

2.3 Диаграмма активности

На рисунке 2.5 продемонстрирована диаграмма общей активности системы, показывающая поэтапно всю работу приложения.

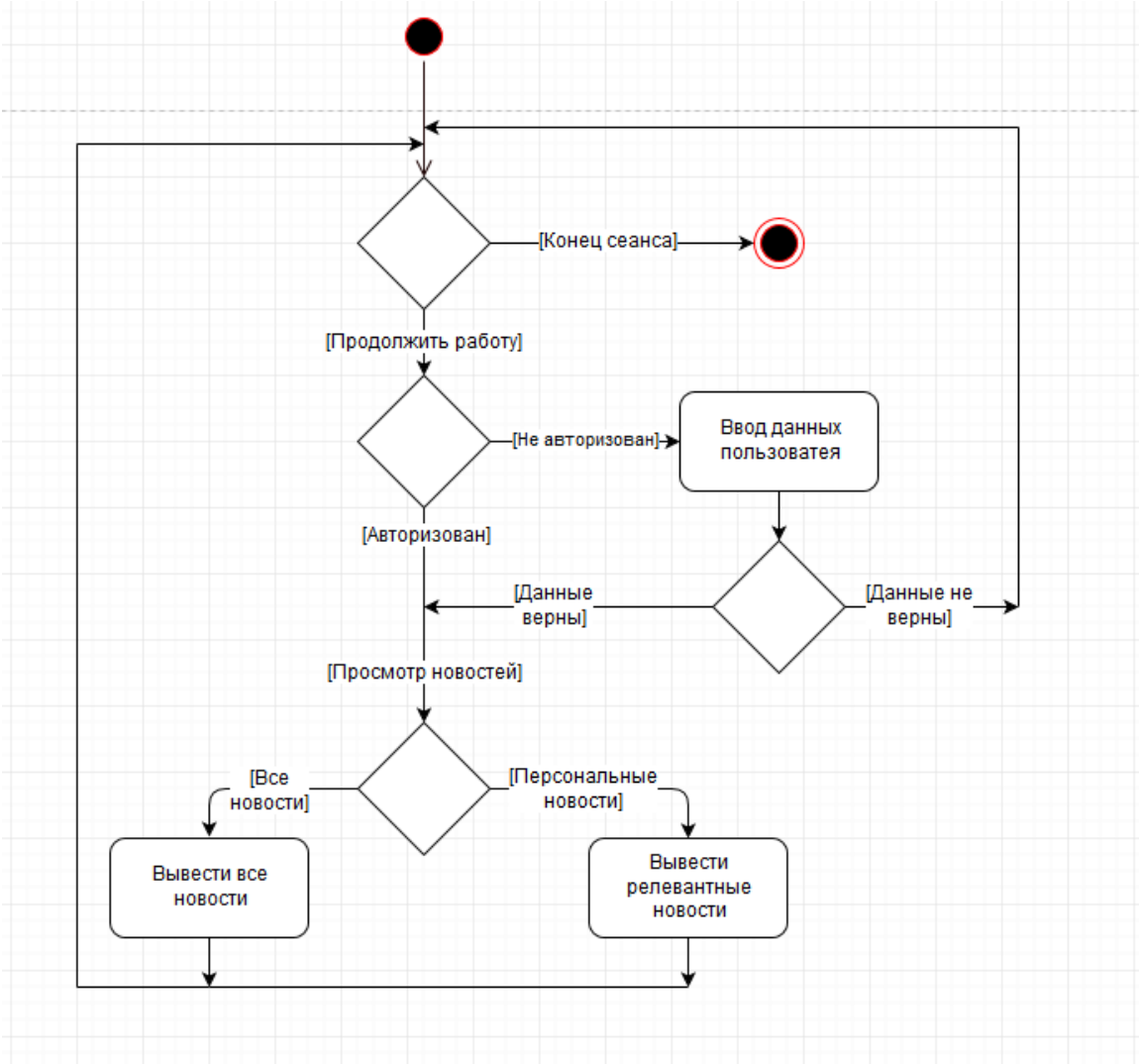

Рисунок 2.5 – Диаграмма активности системы

2.4 Построение базы данных

Разработка базы данных начинается с концептуального проектирования [16]. Для этого необходимо выделить все сущности и связи в системе. В системе имеется пять сущностей: Пользователь, Вес тем, Темы, Новости и Список сайтов. А также 4 связи «один-ко-многим».

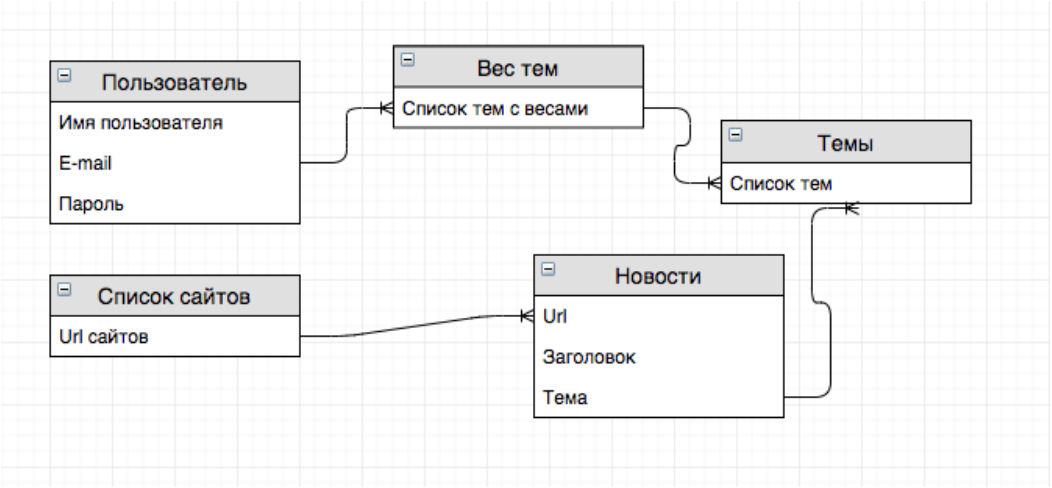

Рисунок 2.6 – ER-диаграмма базы данных

В каждой коллекции имеются свои уникальные _id для объектов в ней. В ER-диаграмме можно увидеть, что база данных имеет три коллекции - users, news_sites и fresh_news, рассмотрим их подробнее.

Таблица 2.1

Коллекция “users” – пользователи системы

	Тип данных	Значение по умолчанию	Ограничения
_id	String	-	уникальный
e-mail	String	-	шаблон
username	String	-	уникальный
password	String	-	не менее 6 симв
tags	String	-	-

В данной коллекции хранятся данные пользователя – уникальный _id, который присваивает сама MongoDB, почтовый адрес (e-mail), имя пользователя(username) и пароль (password). Таблица tags – примерная. Количество строк может увеличиваться в зависимости от количества тем, которые будет различать нейронная сеть. Имя пользователя должно быть уникальным. В целях повышения безопасности пароль не должен быть короче 6 символов.

В коллекции “fresh_news” хранятся новости, получаемые с помощью парсинга – уникальный _id, который присваивает функция на стороне сервера, заголовок

статьи(title), ссылка(url) на саму новость и наконец тэг(tag), который выдает новости нейронная сеть.

Таблица 2.2

Коллекция “fresh_news” – список новостей

	Тип данных	Значение по умолчанию	Ограничения
_id	String	-	уникальный
title	String	-	уникальный
url	String	-	уникальный
tag	String	-	-

В коллекции “news_sites” хранятся новостные порты, с которых берутся новости – уникальный _id, который присваивает функция на стороне сервера и url сайта (nameSite).

Таблица 2.3

Коллекция “news_sites” – пользователи системы

	Тип данных	Значение по умолчанию	Ограничения
_id	String	-	уникальный
nameSite	String	-	уникальный

Документы в MongoDB хранятся в виде JSON формате, представляющим из себя массив объектов. На примере коллекции со списком новостей показан внешний вид записей, которые добавляются в базу данных. (см.рисунок 2.7)

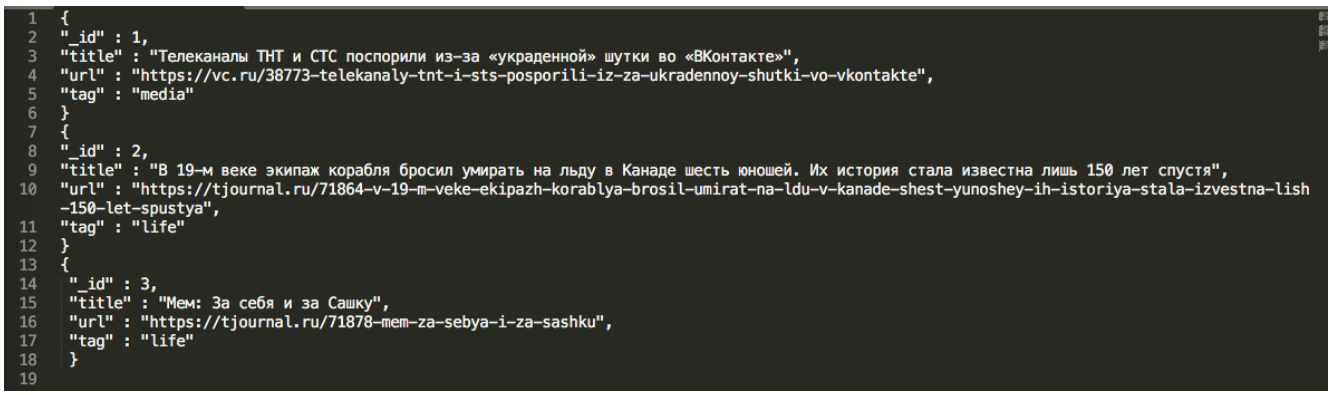

Рисунок 2.6 – Пример записей в MongoDB

2.5 Выводы по разделу

В данном разделе разработана база данных для системы.

Первым делом было выполнено концептуальное проектирование, в ходе которого сначала были выделены сущности и связи в системе, а потом на основе них была построена ER-диаграмма.

После этого построена NoSQL база данных, приведено описание всех коллекций и атрибутов в ней.

3 АЛГОРИТМЫ

3.1 Алгоритм определения тематики новости

В ходе разработки проекта была поставлена задача определения тематики новости для анализа, которым будет заниматься рекомендательная система. Принято решение остановиться на модифицированном Наивном Байесовском классификаторе реализованном через структуру MCMC, которая может использоваться для установки байесовской модели с дискретными и непрерывными параметрами [7].

В основе NBC (Naïve Bayes Classifier) лежит теорема Байеса:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)},$$

где

P(c|d) – вероятность что документ d принадлежит классу c, именно её нам надо рассчитать;
P(d|c) – вероятность встретить документ d среди всех документов класса c;
P(c) – безусловная вероятность встретить документ класса c в корпусе документов;
P(d) – безусловная вероятность документа d в корпусе документов.

Цель классификации состоит в том, чтобы понять к какому классу принадлежит документ, поэтому нам нужна не сама вероятность, а наиболее вероятный класс. Байесовский классификатор использует оценку апостериорного максимума (Maximum a posteriori estimation) для определения наиболее вероятного класса. Это класс с максимальной вероятностью [9].

$$c_{map} = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

Необходимо рассчитать вероятность для всех классов и выбрать тот класс, который обладает максимальной вероятностью. Знаменатель (вероятность документа) - константа и никак не может повлиять на ранжирование классов.

Замечаем, что O зависят только от класса C, и не зависят друг от друга. Это сильно упрощение, но зачастую это работает.

Числитель принимает вид:

$$P(C)P(o_1|C)P(o_2|Co_1)...P(o_n|Co_1o_2...o_n) = P(C)P(o_1|C)P(o_2|C)..P(o_n|C) = P(C)\prod_i(o_i|C)$$

В конце мы получаем нижеприведенную формулу.

$$c = \arg\max_{c \in C} P(c|o_1o_2...o_n) = \arg\max_{c \in C} P(c)\prod P(o_i|c)$$

Остается вычислить вероятности P(C) и P(O|C). Это будет тренировкой классификатора.

Можно заметить, что данный классификатор реализует bags of word model [8]. Он представляет документ как набор слов, вероятности которых условно не зависят друг от друга.

В дипломной работе классификатор анализирует заголовки новостей, полученные методом парсинга, а затем присуждает им одну из шести тем, к которой относится данная новость.

3.2 Алгоритм вывода новостей

Далее стояла задача вывода новостей для пользователя. Был разработан простой, но эффективный алгоритм, который основывается на приоритетных темах для каждого пользователя, а также учитывают дату вывода новости, что также влияет на приоритет вывода:

$$X * e^{-\lambda \Delta t},$$

где

X – приоритет пользователя для тематики новости;

– время, за которое у тем снижается приоритет, в данной работе приоритет снижается раз в 7 дней;

– выведенный коэффициент снижения приоритета равный 0.099021.

В ходе работы данного алгоритма приоритет каждой тематики раз в неделю снижается в 2 раза, что позволит со временем «забывать» предпочтения, которые перестают быть интересны для пользователя.

3.3 Вывод по разделу

В данном разделе были разработаны алгоритма для определения тематики новости и вывода их для пользователя.

После выбора и разработки алгоритмов они были успешно реализованы в данной работе.

4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММЫ

4.1 Интерфейс приложения

В ходе дипломной работы было написано визуальное представление новостного агрегатора. Внешний вид представляет из себя простой сайт, созданный с помощью фреймворка React. В верхней части экрана находится панель навигации, на которой располагаются три кнопки «Ваша подборка», «Все новости» и «Профиль»(см. рисунок 4.1)

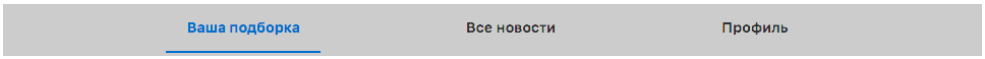

Рисунок 4.1 – Панель навигации

Первая вкладка, «Ваша подборка», отображает все персональные новости, которые подобрал агрегатор на основе предпочтений пользователя. (см. рисунок 4.2). Если же у пользователя нет учетной записи, то эта вкладка будет пустой.

Рисунок 4.2 – Вкладка «Ваша подборка»

Вкладка «Все новости» доступна для всех посетителей ресурса, даже если у них нет учетной записи. В ней отображаются все новости без разделения на тематики (см. рисунок 4.3).

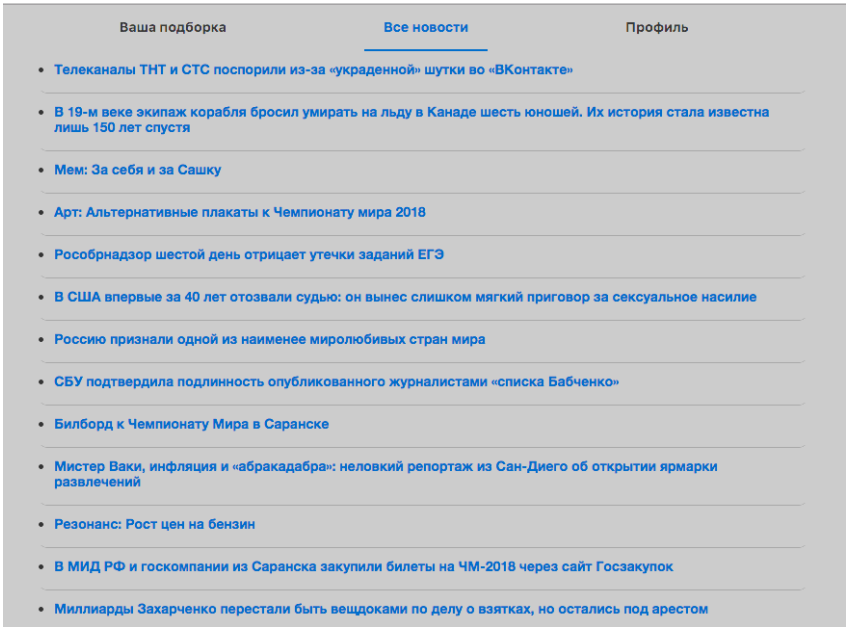

Рисунок 4.3 – Вкладка «Все новости»

Вкладка «Профиль» отвечает за регистрацию в системе и авторизацию, если учетная запись уже заведена. При первом посещении отображается форма авторизации и кнопка регистрации(см. рисунок 4.4). При нажатии на последнюю открывается форма, где можно завести свою учетную запись в системе (см. рисунок 4.5). Если же войти в систему, то на данной странице будет отображаться надпись, что пользователь авторизован, а также кнопка выхода из учетной записи (см. рисунок 4.6).

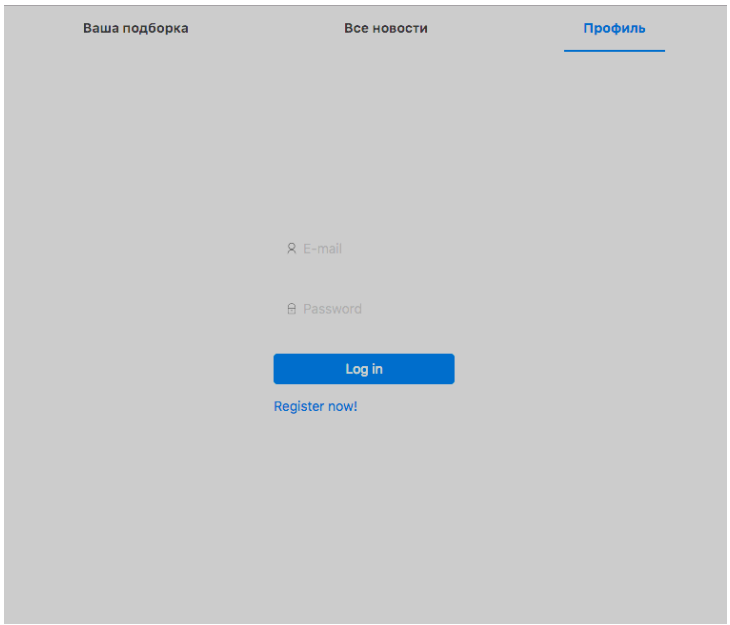

Рисунок 4.4 – Вкладка «Профиль»

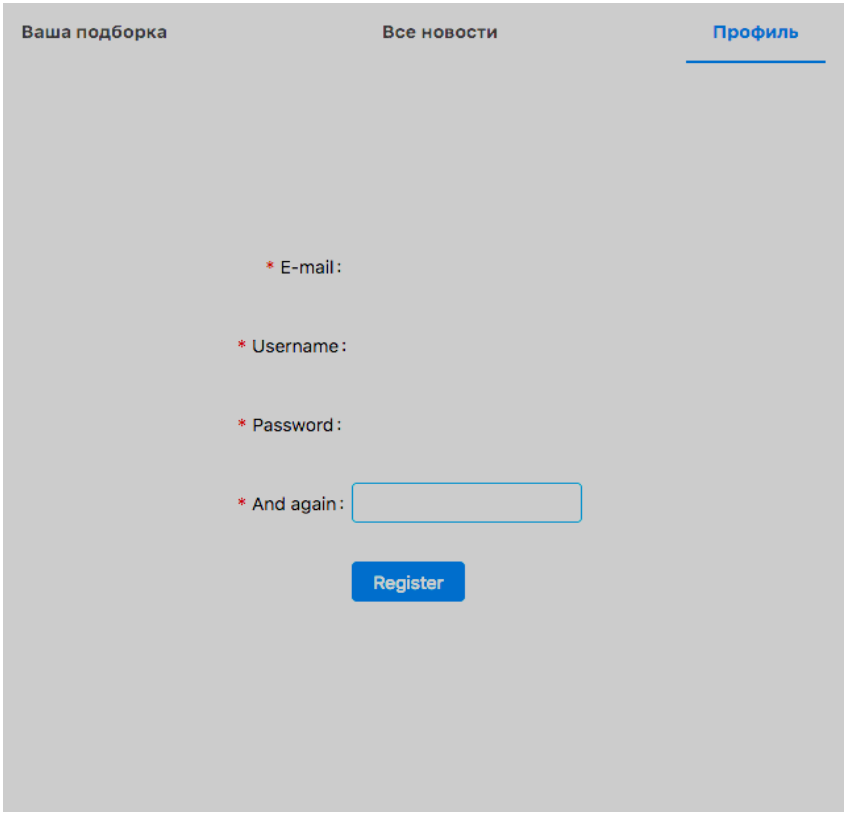

Рисунок 4.5 – Регистрация в системе

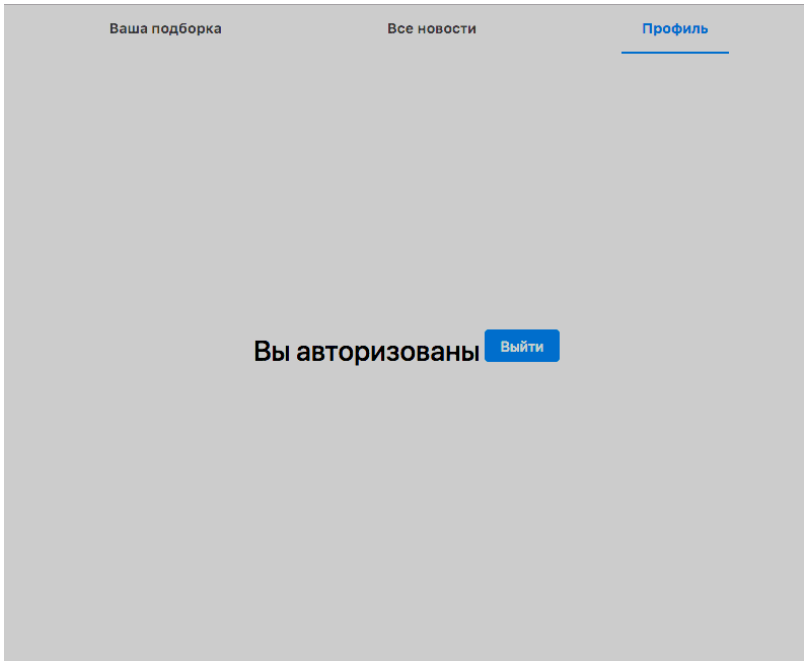

Рисунок 4.6 – Отображение для авторизированного пользователя

4.2 Методика и результаты тестирования программы

Для исследования работоспособности данного Веб-приложения отобраны несколько сайтов, с которых берется и анализируется информация. Агрегатор

будет раз в час обращаться к порталам и брать оттуда список все появившихся за это время новостей.

https://tjournal.ru/
https://dtf.ru/
https://vc.ru/

После парсинга информации в базу данных и определения тематик новости сохранялись в базу данных (см. рисунок 4.7).

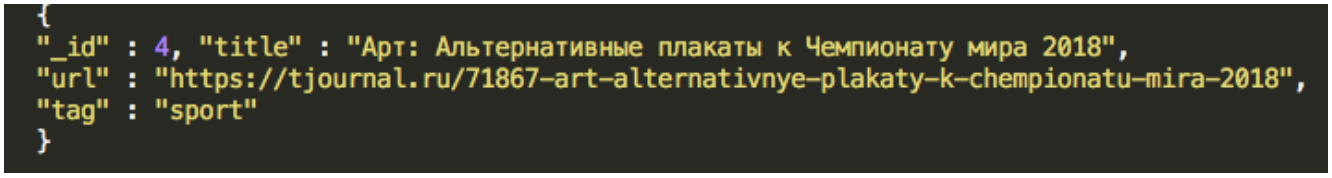

Рисунок 4.7 – Внешний вид записи в MongoDB

После этого создана тестовая учетная запись пользователя, где были наугад открыты несколько новостей, о чем занеслась информация в базу данных (см. рисунок 4.8).

Рисунок 4.8 – Информация о пользователе в MongoDB

При просмотре учетной записи пользователя видно, что пароль хранится в зашифрованном виде, а также имеет свой уникальный ObjectId. А также после взаимодействия с новостями можно заметить, что запись «media» имеет наибольший приоритет, т.к. новости, относящиеся к данной тематике, были выбраны наибольшее количество раз.

После этого страница с подборкой новостей была обновлена и новости, выводимые для пользователя, изменились (см рисунок 4.9 и 4.10)

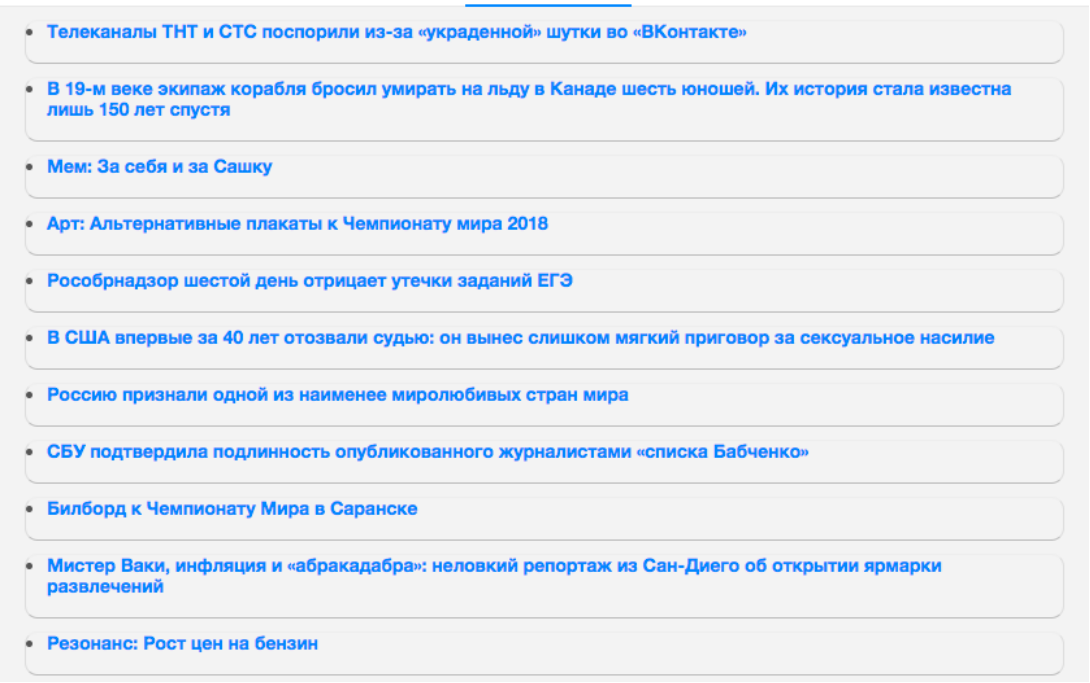

Рисунок 4.9 – Результат вывода информации до появления предпочтений

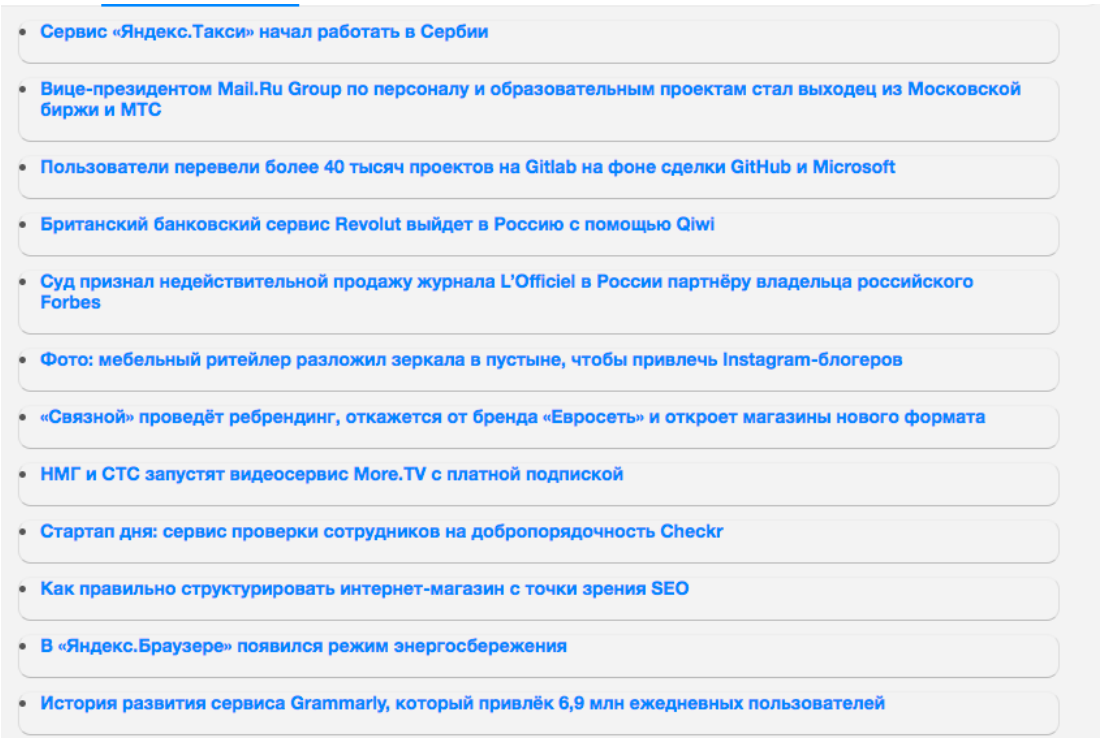

Рисунок 4.10 – Результат вывода информации после появления предпочтений

По изображению 4.10 можно заметить, что выводимая информация изменилась. Стали выводиться новости, которые в MongoDB имеют данный тэг, а также занесены туда не так давно. (см. рисунок 4.11)

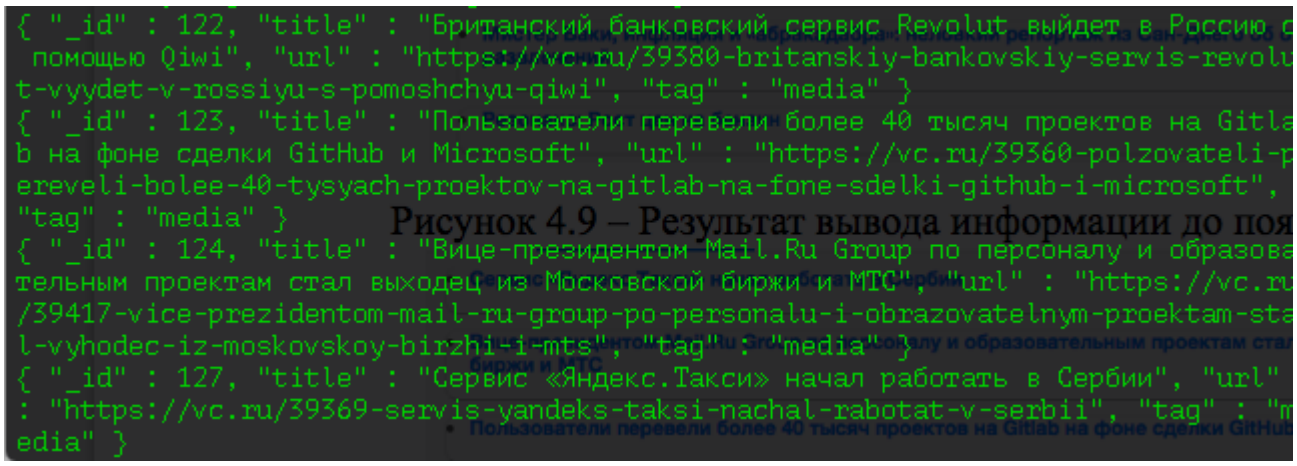

Рисунок 4.11 – Новости с темой «media»в MongoDB

4.3 Вывод по разделу

В данном разделе был произведен обзор интерфейса, а также протестировано финальное Веб-приложение.

В результате данного тестирования было выявлено, что данное веб-приложение работает исправно. Новости стабильно извлекаются в заданный промежуток времени, а также корректно выводятся для пользователя.

ЗАКЛЮЧЕНИЕ

В ходе дипломной работы проведен анализ существующих на данный момент новостных агрегаторов. Произведен обзор различных технологий для построения веб-приложения и обзор существующих направлений в построении баз данных. В ходе обзора были выбраны технологии, которые были использованы в построении дипломной работы. Также были выбраны и реализованы алгоритмы, которые помогли в выводе персональной новостной подборки. Приложение было успешно реализовано с помощью технологий NodeJS, фреймворка React и базы данных MongoDB. В ходе проведения череды тестов приложение показало хорошую работоспособность.

В дальнейшем планируется переработать интерфейс, улучшить алгоритм анализа принадлежности статей, а также добавить возможность регистрации при помощи сторонних социальных сетей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Скляр, Д. Изучаем PHP 7. Руководство по созданию интерактивных веб-сайтов / Д. Скляр – Москва: Вильямс, 2017. – 710с.
2. Макгаврен, Д. Head First. Изучаем Ruby / ред. П. Щеголев – Санкт-Петербург: Питер, 2017. – 544с.
3. Пауэрс, Ш. Изучаем Node. Переходим на сторону сервера / ред. Ю. Сергиенко – Санкт-Петербург: Питер, 2017. – 400с.
4. Клеменков, П.А. Применение NoSQL для построения рекомендательных сервисов реального времени - МГУ – 20с.
5. Seguin, K. The Little MongoDB Book / 2014. – 66p.
6. Ларман, К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / К. Ларман – Москва: Вильямс, 2014. – 732с.
7. Roberts, O., Jeffrey, S. Examples of Adaptive MCMC / 2009.– 28p.
8. Wikipedia. – URL: https://en.wikipedia.org/wiki/Bag-of-words_model
9. Блог Дениса Баженова. – URL: http://bazhenov.me/blog/2012/06/11/naive-bayes.html
10. Хабрахабр. – URL: https://habr.com/post/301426/
11. Хабрахабр. – URL: https://habr.com/post/201206/
12. Medium.com. – URL: https://medium.com/of-all-things-tech-progress/starting-with-authentication-a-tutorial-with-node-js-and-mongodb-25d524ca0359
13. Бэнкс,А. React и Redux. Функциональная веб-разработка/ ред. Ю. Тарасов – Санкт-Петербург: Питер, 2017. – 336с.
14. Medium.com. – URL: https://medium.com/@filipovskii_ru/%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-react-js-37ac23f5bb2a
15. Хабрахабр. – URL:https://habr.com/company/ruvds/blog/343022/
16. Конноли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика/ ред. К. Тарасенко – Москва: Вильямс, 2014. – 1440с.
17. Таненбаум, Э. Компьютерные сети / ред. Ю. Сергиенко – Санкт-Петербург: Питер, 2017. – 960с.

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ ПРОГРАММЫ

1. Общие сведения

Для использования клиентской части приложения необходим один из следующих веб-браузеров: Google Chrome 4.0+, Mozilla Firefox 2.0+, Safari 3.1+. Для размещения серверной части необходимы Node.js и сервер баз данных mongoDB.

2. Функциональное назначение

Программный продукт предназначен для подбора оптимальных новостей для пользователя в целях экономии и оптимизации его времени.

3. Описание логической структуры

Программу можно разбить на следующие составные части:

– компоненты интерфейса, обеспечивающие взаимодействие пользователя с системой;

– алгоритмы, отвечающие за парсинг новостей, определение их тематик, а также вывод;

– модуль, обрабатывающий запросы от клиентов;

– модуль, обеспечивающий взаимодействие с сервером баз данных.

– модуль, отвечающий за авторизацию пользователей.

4. Используемые технические средства

Для работы системы необходим компьютер с объемом оперативной памяти не менее 1Гб оперативной памяти, тактовой частотой процессора не менее 2 ГГц, подключение к интернету с пропускной способностью не менее 10 Мбит/сек, имеющий один из установленных браузеров.

ПРИЛОЖЕНИЕ 2

ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

1. App.js

```
const express = require("express");
const app = express();
const port = 2000;
const bodyParser = require('body-parser');
const bcrypt = require('bcrypt');
var ObjectId = require('mongodb').ObjectID;
const nameSitesSchema = require('./models/nameSitesSchema');
const updateUser = require('./models/updateUserSchema');
const User = require('./models/userSchema');
const mongoose = require("mongoose");
session = require('express-session')
const jwt = require('jsonwebtoken');
const jwtDecode = require('jwt-decode');
const keys = require('./config/keys');
const passport = require('passport');
const PersonalNewsScrapping = require('./PersonalNewsScrapping');
const MongoStore = require('connect-mongo')(session);
const MongoClient = require('mongodb').MongoClient;
const mongo_url = "mongodb://localhost:27017/mydb";
var arrSites = [];
let priority = [];
let firstPrior = [];
let secondPrior = [];
let thirdPrior = [];

app.set('view engine', 'ejs');

mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/nameSites');
const db = mongoose.connection;

//use sessions for tracking logins
app.use(session({
    secret: 'work hard',
    resave: true,
    saveUninitialized: false,
    store: new MongoStore({
        mongooseConnection: db
    })
}));

//handle mongo error
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function () {
    // we're connected!
});

// Passport
app.use(passport.initialize());

//Passport Config
require('./models/passport')(passport);
// parse incoming requests
app.use(bodyParser.json());
```

```
    app.use(bodyParser.urlencoded({extended: false}));

    app.use(function (req, res, next) {

        // Website you wish to allow to connect
        res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3001');

        // Request methods you wish to allow
        res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
PATCH, DELETE');

        // Request headers you wish to allow
        res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-
type');

        // Set to true if you need the website to include cookies in the requests
sent
        // to the API (e.g. in case you use sessions)
        res.setHeader('Access-Control-Allow-Credentials', true);

        // Pass to next layer of middleware
        next();
    });

    app.get("/urnews", (req, res) => {                  //Получение новостей
        nameSitesSchema.find({}, (err, newssites) => {
            if (err) return res.status(500).send(err);
            res.json(newssites);
        });
    });


    //POST route for updating data
    app.post('/api/main', function (req, res, next) {   //Авторизация
        console.log('-1');
        const email = req.body.user.username;
        const password = req.body.user.password;
        console.log(email, password);
        console.log('-2');
        User.findOne({email})
            .then(user => {
                if (!user) {
                    console.log('-4');
                    return res.status(404).json({email: 'User not found'});
                }

                bcrypt.compare(password, user.password)
                    .then(isMatch => {
                        if (isMatch) {

                            const payload = {id: user.id, name: user.name};

                            jwt.sign(
                                payload,
                                keys.secretOrKey,
                                {expiresIn: 3600},
                                (err, token) => {
                                    res.json({
                                        success: true,
                                        token: 'Bearer' + token
                                    });
                                });
```

```
                        console.log('-3');
                    } else {
                        return res.status(400).json({password: 'Password
incorrect'});
                    }
                })
        })
    });

    app.get('api/current', passport.authenticate('jwt', {session: false}), (req,
res) => {
        res.json({
            id: req.user.id,
            name: req.user.name,
            email: req.user.email,
        });
    });

    app.post('/api/register', function (req, res, next) {
        if (req.body.user.email &&
            req.body.user.username &&
            req.body.user.password &&
            req.body.user.passwordConf) {
            console.log('0');
            const userData = {
                email: req.body.user.email,
                username: req.body.user.username,
                password: req.body.user.password,
                passwordConf: req.body.user.passwordConf,
                media: 1,
                world: 1,
                life: 1,
                economics: 1,
                science: 1,
            }
            console.log(userData);
            console.log('1');
            User.create(userData, function (error, user) {
                if (error) {
                    return next(error);
                } else {
                    req.session.userId = user._id;
                    return res.sendStatus(200);
                }
            });

        }
    });

    app.get('/api/personal_news', function (req, res, next) {
        let param = jwtDecode(req.query.token);
        console.log(param);
        let id = param.id;
        console.log(id);
        function tag_promised(mongo_url) {
            return new Promise((resolve) => {
                MongoClient.connect(mongo_url, function (err, db) {
                    if (err) throw err;
                    let dbo = db.db("nameSites");
                    dbo.collection("Users").find({"_id": new ObjectId(id)}, {
                        _id: 0,
                        email: 0,
```

```
                    username: 0,
                    password: 0,
                    passwordConf: 0,
                    __v: 0
                }).toArray(function (err, result) {
                    if (err) throw err;
                    arrSites = result;
                    db.close();
                    console.log(arrSites);
                    resolve(arrSites);
                });
            });
        });
    }

    tag_promised(mongo_url).then(result => {    // Получаю сайты для их
парсинга
        console.log(result);
        function getPriority() {
            let sortable = [];
            for (let priority in result[0]) {
                sortable.push([priority, result[0][priority]]);
            }
            sortable.sort(function (a, b) {
                return b[1] - a[1];
            });
            for (let i = 0; i < 3; i++) {
                console.log(sortable[i][0]);
                priority[i] = sortable[i][0];

            }
            console.log(priority);
            return priority;
        }

        let PriorForMongo = getPriority();
        console.log('test' + PriorForMongo);

        function p1(mongo_url) {
            return new Promise((resolve) => {
                MongoClient.connect(mongo_url, function (err, db) {
                    if (err) throw err;
                    let dbo = db.db("nameSites");
                    dbo.collection("freshMeat").find({tag:
PriorForMongo[0]}).sort({_id: -1}).limit(30).toArray(function (err, result) {
                        if (err) throw err;
                        for (i = 0; i <= 30; i++) {
                            if (result[i] !== undefined) {
                                firstPrior[i] = result[i];
                            }
                        }
                        db.close();
                        resolve(firstPrior);

                    });
                });
            });
        }

        function p2(mongo_url) {
            return new Promise((resolve) => {
                MongoClient.connect(mongo_url, function (err, db) {
```

```
                        if (err) throw err;
                        let dbo = db.db("nameSites");
                        dbo.collection("freshMeat").find({tag:
PriorForMongo[1]}).sort({_id: -1}).limit(20).toArray(function (err, result) {
                            if (err) throw err;
                            for (i = 0; i <= 20; i++) {
                                if (result[i] !== undefined) {
                                    secondPrior[i] = result[i];
                                }
                            }
                            db.close();
                            resolve(secondPrior);
                        });
                    });
                });
            }

            function p3(mongo_url) {
                return new Promise((resolve) => {
                    MongoClient.connect(mongo_url, function (err, db) {
                        if (err) throw err;
                        let dbo = db.db("nameSites");
                        dbo.collection("freshMeat").find({tag:
PriorForMongo[2]}).sort({_id: -1}).limit(10).toArray(function (err, result) {
                            if (err) throw err;
                            for (i = 0; i <= 10; i++) {
                                if (result[i] !== undefined) {
                                    thirdPrior[i] = result[i];
                                }
                            }
                            db.close();
                            resolve(thirdPrior);
                        });
                    });
                });
            }

            Promise.all([p1(mongo_url), p2(mongo_url), p3(mongo_url)]).then(values
=> {
                const test = [...values[0], ...values[1], ...values[2]];
                res.send(test);
            });

        });
    });

    app.post('/api/personal_news', function (req, res, next) {
        let user = jwtDecode(req.body.news.token);
        let tag = req.body.news.tag;
        console.log(tag);
        let id = user.id;
        console.log(id);
        var key = tag;
        var obj = {};
        obj[key] = 1;
        MongoClient.connect(mongo_url, function (err, db) {
            if (err) throw err;
            let dbo = db.db("nameSites");
            dbo.collection("Users").update({"_id": new ObjectId(id)}, {$inc: obj},
function (err, result) {
                if (err) {
                    return console.log(err);
```

```
                }
                console.log("RESULT: " + result);
            });
        });
    });
    // GET route after registering
    app.get('/profile', function (req, res, next) {
        User.findById(req.session.userId)
            .exec(function (error, user) {
                if (error) {
                    return next(error);
                } else {
                    if (user === null) {
                        var err = new Error('Not authorized! Go back!');
                        err.status = 400;
                        return next(err);
                    } else {
                        return res.send('<h1>Name: </h1>' + user.username +
'<h2>Mail: </h2>' + user.email + '<br><a type="button" href="/logout">Logout</a>')
                    }
                }
            });
    });


    // GET for logout logout
    app.get('/logout', function (req, res, next) {
        if (req.session) {
            // delete session object
            req.session.destroy(function (err) {
                if (err) {
                    return next(err);
                } else {
                    return res.redirect('/main');
                }
            });
        }
    });

    app.listen(port, () => {
        console.log("Server listening on port " + port);
    });

    module.exports = app;
```

2.PersonalNewsScrapping.js

```
var request = require('request');
var cheerio = require('cheerio'),
    async = require("async");
var MongoClient = require('mongodb').MongoClient;
var mongo_url = "mongodb://localhost:27017/mydb";
var bayes = require('bayes');
var classifier = bayes();
var Converter = require("csvtojson").Converter;
var converter = new Converter({});

var arrSites =[];
var arrFlag =[];
var flag=0;
var gate = 60000;

function getPars() {
```

```
converter.fromFile("/Users/ivanmarchenko/Downloads/заголовки/lenta_1000_lem.csv",f
unction(err,result1){
    if(err){
        console.log("An Error Has Occured");
        console.log(err);
    }
    var json = result1;
    try {
        for (i = 0; i <= json.length; i++) {
            classifier.learn(json[i].title, json[i].tag);
        }
    } catch (err){
    }
//function parsing() {
    function sites_promised(mongo_url) {
        return new Promise((resolve) => {
            MongoClient.connect(mongo_url, function (err, db) {
                if (err) throw err;
                let dbo = db.db("nameSites");
                dbo.collection("newsSites").find({}).toArray(function (err,
result) {
                    if (err) throw err;
                    for (i = 0; i <= 2; i++) {
                        arrSites[i] = result[i].site;
                    }
                    db.close();
                    resolve(arrSites);
                });
            });
        });
    }
    sites_promised(mongo_url).then(result => {    // Получаю сайты для их парсинга
        console.log(result);
        function flag_promised() {                                          //highest
flag from field
            return new Promise((resolve) => {
                MongoClient.connect(mongo_url, function (err, db) {
                    if (err) throw err;
                    let dbo = db.db("nameSites");
                    console.log('1');
                    dbo.collection("freshMeat").find({}).sort({"_id": -
1}).limit(3).toArray(function (err, result) {
                        if (err) throw err;
                        console.log(result);
                        arrFlag[0] = result[0]._id;
                        db.close();
                        resolve(arrFlag);
                    });
                });
            });
        }
        flag_promised(mongo_url).then(result => {
            console.log('Результ: '+ result);
            if (flag < result) {
                flag = Number(result)+1;
            }
        });
        let q = async.queue(function (url) {
            request(url, function (error, response, html) {
                if (!error && response.statusCode === 200) {
                    let $ = cheerio.load(html);
```

```
                    $(' div.entry_content.entry_content--short > div > h2 >
span').each(async function () {
                        let a = $(this);
                        let title = a.text();
                        let url =
a.parent().parent().parent().children(".entry_content__link").attr('href');
                        let tag = await classifier.categorize(title);
                        let id = flag;
                        console.log('Флаги: '+ flag);
                        let metadata = {
                            title: title,
                            url: url,
                            _id: id,
                            tag: tag
                        };
                        function checkCount(mongo_url) {
                            return new Promise((resolve) => {
                                MongoClient.connect(mongo_url, function (err, db)
{
                                    if (err) throw err;
                                    let dbo = db.db("nameSites");

dbo.collection("freshMeat").find({}).toArray(function (err, result) {
                                        if (err) throw err;
                                        let checkCounter=false;
                                        for (i = 0; i < result.length; i++) {
                                            if (url == result[i].url) {
                                                checkCounter=true;
                                            }
                                        }
                                        db.close();
                                        resolve(checkCounter);
                                        console.log('CheckCounter проверка' +
checkCounter);
                                    });
                                });
                            });
                        }
                        checkCount(mongo_url).then(result => {
                            console.log('CheckCounter проверка финальная'+result);
                            if (result === false) {
                                console.log('Проверка');
                                MongoClient.connect(mongo_url, function (err, db)
{
                                    if (err) throw err;
                                    let dbo = db.db("nameSites");

dbo.collection("freshMeat").insertOne(metadata, function (err) {
                                        if (err) {
                                            return console.log(err);
                                        }
                                        console.log(id);
                                        console.log(metadata);
                                    });

                                    db.close();
                                });
                            }
                        });
                        await flag++;
                    });
                }
```

```
            });
        }, 10);
        let j = 0;
        while (arrSites.length > j) {
            console.log(arrSites[j]);
            q.push(arrSites[j]);
            j++;
        }
    });
});
}

setInterval(getPars, 60000*60);
```

3.nameSitesSchema.js

```
var mongoose = require('mongoose');

var nameSitesSchema = mongoose.Schema({
        nameSites: String,
        urlSites: String,
        _id: String,
        tag: String},
    {collection: 'freshMeat'});

var nameSites = mongoose.model('nameSites', nameSitesSchema);

module.exports = nameSites;
```

4.passport.js

```
const mongoose = require('mongoose');
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const User = mongoose.model('User');
const keys = require('../config/keys');

const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = keys.secretOrKey;

module.exports = passport => {
  passport.use(new JwtStrategy(opts, (jwt_payload, done) => {
      User.findById(jwt_payload.id)
          .then(user => {
              if(user){
                  return done(null, user);
              }
              return done(null, false);
          })
          .catch(err => console.log(err));
  }));
};
```

5.updateUserSchema.js

```
var mongoose = require('mongoose');

var updateUserSchema = mongoose.Schema({
         id: String,
        tag: Number},
    {collection: 'Users'});

var updateUser = mongoose.model('updateUser', updateUserSchema);

module.exports = updateUser;
```

6.userSchema.js

```
var mongoose = require('mongoose');
var bcrypt = require('bcrypt');
var UserSchema = new mongoose.Schema({
        email: {
            type: String,
            unique: true,
            required: true,
            trim: true
        },
        username: {
            type: String,
            unique: true,
            required: true,
            trim: true
        },
        password: {
            type: String,
            required: true,
        },
        passwordConf: {
            type: String,
            required: true,
        },
        life: {
            type: Number,
        },
        media: {
            type: Number,
        },
        world: {
            type: Number,
        },
        science: {
            type: Number,
        },
        economics: {
            type: Number,
        },
    },
    {collection: 'Users'}
);


//hashing a password before saving it to the database
UserSchema.pre('save', function (next) {
    var user = this;
    bcrypt.hash(user.password, 10, function (err, hash) {
        if (err) {
            return next(err);
        }
        user.password = hash;
        next();
    })
});

var User = mongoose.model('User', UserSchema);
module.exports = User;
```

7.keys.js

```
module.exports = { secretOrKey: 'secret'} ;
```

Клиентская часть:

1.App.js

```
import React, { Component } from 'react';
import {Flex} from 'grid-styled';
import styled from 'styled-components';
import Header from './components/app/Header'
import Routes from './components/app/Routes'

import './App.css';
export default class App extends Component {
    render() {
        return (
            <StyledFlex flexDirection='column' flex={1}>
                <Header />
                <Flex flex={1}>
                    <Routes />
                </Flex>
            </StyledFlex>
        );
    }
}

const StyledFlex = styled(Flex)`
      height: 100%;
      min-height: 100vh;
      background: #f3f3f3;
`;
```

2.App.css

```
@import '~antd/dist/antd.css';

.App {
    text-align: center;
}

.App-logo {
    animation: App-logo-spin infinite 20s linear;
    height: 80px;
}

.App-header {
    background-color: #222;
    height: 150px;
    padding: 20px;
    color: white;
}

.App-title {
    font-size: 1.5em;
}

.App-intro {
    font-size: large;
}

@keyframes App-logo-spin {
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
}
```

3.index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';


ReactDOM.render((
        <BrowserRouter>
            <App />
        </BrowserRouter>
    ),
    document.getElementById('root'));

4.index.css
body {
    margin: 0;
    padding: 0;
    font-family: sans-serif;
}

5.header.css

ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #b4a7d6;
    border-radius: 20px;
}

li {
    float: left;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
    border-radius: 10px;
}

li a:hover {
    background-color: #b366ff;
}

6.header.js
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import Menu from 'antd/lib/menu';
//import '../header.css';
import styled from 'styled-components';
import { Flex } from 'grid-styled';


const MenuItem = Menu.Item;
export default class Header extends Component {

    static contextTypes = {
```

```
        ...Component.contextTypes,
        router: PropTypes.object,
    };


    render() {
        return (
            <header className="header">
                <StyledFlex>
                    <Menu mode="horizontal" onClick={item =>
this.context.router.history.push(`/${item.key}`)}>
                        <MenuItem key="personal_news">
                            Ваша подборка
                        </MenuItem>
                        <MenuItem key="urnews">
                            Все новости
                        </MenuItem>
                        <MenuItem key="main">
                            Профиль
                        </MenuItem>
                    </Menu>
                </StyledFlex>
            </header>
        );
    }
}

const StyledFlex = styled(Flex)`
    justify-content: center;
      .ant-menu{
          display: flex;
          justify-content: space-between;
          width: 70%;
          padding: 0 20px;
          box-sizing: border-box;

          border-radius: 20px;
          font-weight: bolder;
      }
`;
7.login.js
import 'antd/dist/antd.css';
import styled from 'styled-components';
import { Flex } from 'grid-styled';
import React from 'react'


export default class Login extends React.Component {
    render(){
        return(
            <StyledFlex2>
            <h1>Вы вошли!</h1>
            </StyledFlex2>
        )
    }
}

const StyledFlex2 = styled(Flex)`
  .demo-loadmore-list {
  min-height: 350px;
}
    margin: auto;
```

```
`;
8.LoginForm.js
import 'antd/dist/antd.css';
import PropTypes from 'prop-types';
import './RegisterForm';
import styled from 'styled-components';
import { Flex } from 'grid-styled';
import React from 'react'
import { Form, Icon, Input, Button } from 'antd';
const FormItem = Form.Item;
const axios = require('axios');

class NormalLoginForm extends React.Component {
    state = {
        confirmDirty: false,
        autoCompleteResult: [],
        email: '',
        username: '',
        password: '',
        passwordConf: '',
        token: [],
    };
    static contextTypes = {
        router: PropTypes.object.isRequired,
    }

    handleChange (value, param){
        this.setState({[param]: value});
    }

    clearStorage() {
        sessionStorage.clear();
        this.context.router.history.push('/urnews');
    }

    handleSubmit = event => {
        event.preventDefault();

        const user = {
            email: this.state.email,
            username: this.state.username,
            password: this.state.password,
            passwordConf: this.state.passwordConf,
        };

        axios.post('/api/main', {user})
            .then(res => {
                console.log(res);
                console.log(res.data);
                this.context.router.history.push('/login');
                sessionStorage.setItem('token',res.data.token);
            })
    }
    render() {
        const { getFieldDecorator } = this.props.form;
        return (
            <StyledFlex2>
                { sessionStorage.getItem('token') == null ?
                    (<Form onSubmit={this.handleSubmit} className="login-form"
method="post" action="/login">
                    <FormItem>
```

```
                        {getFieldDecorator('userName', {
                            rules: [{required: true, message: 'Please input your
email!'}],
                        })(
                            <Input
                                prefix={<Icon
                                    type="user"
                                    style={{color: '#808080'}}/>}
                                setfieldsvalue={this.state.username}
                                placeholder="E-mail"
                                onChange={(e) => this.handleChange(e.target.value,
'username')}
                            />
                        )}
                    </FormItem>
                    <FormItem>
                        {getFieldDecorator('password', {
                            rules: [{required: true, message: 'Please input your
Password!'}],
                        })(
                            <Input
                                prefix={<Icon
                                    type="lock"
                                    style={{color: '#808080'}}/>}
                                setfieldsvalue={this.state.password}
                                type="password" placeholder="Password"
                                onChange={(e) => this.handleChange(e.target.value,
'password')}
                            />
                        )}
                    </FormItem>
                    <FormItem>
                        <Button type="primary" htmlType="submit" className="login-
form-button">
                            Log in
                        </Button>
                        <a href="" onClick={e => {
                            e.preventDefault();
                            this.context.router.history.push('/register');
                        }}
                        >
                            Register now!
                        </a>
                    </FormItem>
                </Form> ) :
                    (
                    <Flex>
                        <h1>Вы авторизованы</h1>
                        <Flex px={1}>
                        <Button
                            type="primary"
                            htmlType="submit"

                            onClick={this.clearStorage.bind(this)}
                        >
                            Выйти
                        </Button>
                        </Flex>
                    </Flex>
                    )

                }
```

```
            </StyledFlex2>
        );
    }
}

const WrappedNormalLoginForm = Form.create()(NormalLoginForm);

export default WrappedNormalLoginForm

const StyledFlex2 = styled(Flex)`
    .login-form {
    max-width: 300px;

}
.login-form-forgot {
    float: right;
}
.login-form-button {
    width: 100%;
}
    margin: auto;

`;
```

9.newsThread.js

```
import React from 'react';
import styled from 'styled-components';
import { Flex } from 'grid-styled';

export default class LoadMoreList extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            data:[]
        };
    };

    componentDidMount() {
        const url = 'http://localhost:2000/urnews';
        fetch(url)
            .then(response => response.json())
            .then((responseData) => {
                this.setState({
                    data: responseData
                });
            })
    }
    render() {
        const list = this.state.data.map(site => (
            <StyledFlex>
            <li key={site._id}>
                <a href={site.url}> {site.title} </a>
            </li>
            </StyledFlex>
        ));
        return (
            <StyledFlex2>
            <div>
                <ul>
                    {list}
                </ul>
            </div>
```

```
            </StyledFlex2>
        )
    }
}


const StyledFlex2 = styled(Flex)`
  .demo-loadmore-list {
  min-height: 350px;
}
    margin: auto;

`;

const StyledFlex = styled(Flex)`
  font-family: "Helvetica Neue", Roboto, "Segoe UI", Calibri, sans-serif;
  font-size: 14px;
  font-weight: bold;
  line-height: 16px;
  border-color: #eee #ddd #bbb;
  border-radius: 10px;
  border-style: solid;
  border-width: 1px;
  margin: 10px 5px;
  padding: 0 16px 16px 16px;
  max-width: 800px;
`;

10.PersonalNewsThread.js
import React from 'react';
import styled from 'styled-components';
import { Flex } from 'grid-styled';
const axios = require('axios');

export default class PersonalNewsThread extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            data:[]
        };

    };

    componentDidMount() {
        const url = '/api/personal_news';
        axios.get(url, {
            params: {
                token: sessionStorage.getItem('token')
            }
        })
            .then(response => {
                this.setState({
                    data: response.data
                });
            });


    }

    async postTagInProfile(tag) {
        const user = sessionStorage.getItem('token');
        const news = {
```

```
            token: user,
            tag: tag,
        };
        axios.post('/api/personal_news', {news})
            .then(res => {
                console.log(res);
                console.log(res.data);
            })
    }

    render() {
        const list = this.state.data.map(site => (
            <StyledFlex>
            <li key={site._id}>
                <a
                    href={site.url}
                    onClick={async () => this.postTagInProfile(site.tag)}
                    target='_blank'> {site.title} </a>
            </li>
            </StyledFlex>
        ));
        return (
            <StyledFlex2>
                <div>
                    <ul>
                        {list}
                    </ul>
                </div>
            </StyledFlex2>
        )
    }
}


const StyledFlex2 = styled(Flex)`
  .demo-loadmore-list {
  min-height: 350px;
}
    margin: auto;


`;

const StyledFlex = styled(Flex)`
  font-family: "Helvetica Neue", Roboto, "Segoe UI", Calibri, sans-serif;
  font-size: 14px;
  font-weight: bold;
  line-height: 16px;
  border-color: #eee #ddd #bbb;
  border-radius: 10px;
  border-style: solid;
  border-width: 1px;
  margin: 10px 5px;
  padding: 0 16px 16px 16px;
  max-width: 800px;
`;


11.RegisterForm.js
import React from 'react';
import 'antd/dist/antd.css';
import PropTypes from 'prop-types';
```

```
import {Form, Input, Button} from 'antd';
import styled from 'styled-components';
import {Flex} from 'grid-styled';

const axios = require('axios');
const FormItem = Form.Item;

class RegistrationForm extends React.Component {
    state = {
        confirmDirty: false,
        autoCompleteResult: [],
        email: '',
        username: '',
        password: '',
        passwordConf: '',
    };
    static contextTypes = {
        router: PropTypes.object.isRequired,
    }

    handleChange (value, param){
        this.setState({[param]: value});
    }

    handleSubmit = event => {
        event.preventDefault();

        const user = {
            email: this.state.email,
            username: this.state.username,
            password: this.state.password,
            passwordConf: this.state.passwordConf,
        };

        axios.post('/api/register', {user})
            .then(res => {
                console.log(res);
                console.log(res.data);
                this.context.router.history.push('/main');
            })
    }
    handleConfirmBlur = (e) => {
        const value = e.target.value;
        this.setState({confirmDirty: this.state.confirmDirty || !!value});
    }
    compareToFirstPassword = (rule, value, callback) => {
        const form = this.props.form;
        if (value && value !== form.getFieldValue('password')) {
            callback('Two passwords that you enter is inconsistent!');
        } else {
            callback();
        }
    }
    validateToNextPassword = (rule, value, callback) => {
        const form = this.props.form;
        if (value && this.state.confirmDirty) {
            form.validateFields(['confirm'], {force: true});
        }
        callback();
    }

    render() {
```

```
        const {getFieldDecorator} = this.props.form;

        const formItemLayout = {
            labelCol: {
                xs: {span: 24},
                sm: {span: 8},
            },
            wrapperCol: {
                xs: {span: 24},
                sm: {span: 16},
            },
        };
        const tailFormItemLayout = {
            wrapperCol: {
                xs: {
                    span: 24,
                    offset: 0,
                },
                sm: {
                    span: 16,
                    offset: 8,
                },
            },
        };
        return (
            <StyledFlex1>
                <Form onSubmit={this.handleSubmit} method="post"
action="/register">
                    <FormItem
                        {...formItemLayout}
                        label="E-mail"
                    >
                        {getFieldDecorator('email', {
                            rules: [{
                                type: 'email', message: 'The input is not valid E-
mail!',
                            }, {
                                required: true, message: 'Please input your E-
mail!',
                            }],
                        })(
                            <Input
                                setfieldsvalue={this.state.email}
                                onChange={(e) => this.handleChange(e.target.value,
'email')}
                            />
                        )}
                    </FormItem>
                    <FormItem
                        {...formItemLayout}
                        label="Username"
                    >
                        {getFieldDecorator('username', {
                            rules: [{ required: true, message: 'Please input your
username!', whitespace: true }],
                        })(
                            <Input
                                setfieldsvalue={this.state.username}
                                onChange={(e) => this.handleChange(e.target.value,
'username')}/>
                        )}
                    </FormItem>
```

```
                    <FormItem
                        {...formItemLayout}
                        label="Password"
                    >
                        {getFieldDecorator('password', {
                            rules: [{
                                required: true, message: 'Please input your
password!',
                            }, {
                                validator: this.validateToNextPassword,
                            }],
                        })(
                            <Input
                                setfieldsvalue={this.state.password}
                                type="password"
                                onChange={(e) => this.handleChange(e.target.value,
'password')}/>
                        )}
                    </FormItem>
                    <FormItem
                        {...formItemLayout}
                        label="And again"
                    >
                        {getFieldDecorator('passwordConf', {
                            rules: [{
                                required: true, message: 'Please confirm your
password!',
                            }, {
                                validator: this.compareToFirstPassword,
                            }],
                        })(
                            <Input
                                setfieldsvalue={this.state.passwordConf}
                                type="password"
                                onBlur={this.handleConfirmBlur}
                                onChange={(e) => this.handleChange(e.target.value,
'passwordConf')}/>
                        )}
                    </FormItem>
                    <FormItem {...tailFormItemLayout} >
                        <Button type="primary" htmlType="submit">Register</Button>
                    </FormItem>
                </Form>
            </StyledFlex1>
        );
    }
}


const WrappedRegistrationForm = Form.create()(RegistrationForm);

//ReactDOM.render(<WrappedRegistrationForm />,
document.getElementById('container'));

export default WrappedRegistrationForm

const StyledFlex1 = styled(Flex)`
    margin: auto;
`;
12.Routes.js
import Loadable from 'react-loadable';
import { Redirect, Route, Switch} from 'react-router-dom';
```

```
import React, {Component} from 'react';

const Main = Loadable({
    loader: () => import('../app/LoginForm'),
    loading() {
        return <div>Loading...</div>
    }
});

const Register = Loadable({
    loader: () => import('../app/RegisterForm'),
    loading() {
        return <div>Loading...</div>
    }
});

const UrNews =  Loadable({
    loader: () => import('../app/NewsThread'),
    loading() {
        return <div>Loading...</div>
    }
});

const Login =  Loadable({
    loader: () => import('../app/Login'),
    loading() {
        return <div>Loading...</div>
    }
});
const PersonalNews =  Loadable({
    loader: () => import('../app/PersonalNewsThread'),
    loading() {
        return <div>Loading...</div>
    }
});

export default class Routes extends Component {
    render() {
        return (
            <Switch>
                <Route
                    path='/main'
                    render={() => <Main />}
                />
                <Route
                    path='/urnews'
                    render={() => <UrNews />}
                />
                <Route
                    path='/login'
                    render={() => <Login />}
                />
                <Route
                    path='/register'
                    render={() => <Register />}
                />
                <Route
                    path='/personal_news'
                    render={() => <PersonalNews />}
                />

                <Redirect to='/main' />
```

```
                </Switch>
            );
        }
    }
```