##### Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Базы и банки данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ОНЛАЙН-ПЛАТФОРМА ДЛЯ ОТЗЫВОВ,**

**СОВЕТОВ И ЛАЙФХАКОВ**

Студент гр.620604 С.С. Ежгунович

Руководитель А.Ф. Трофимович

Минск 2019

**РЕФЕРАТ**

ОНЛАЙН-ПЛАТФОРМА ДЛЯ ОТЗЫВОВ, СОВЕТОВ И ЛАЙФХАКОВ: курсовой проект / С.С. Ежгунович – Минск: БГУИР, 2019, – п.з. –61 с.

Основной задачей курсового проекта является создание сервиса для возможности оставлять отзывы о товарах, давать советы и делиться лайфхаками.

Для разработки проекта системы был выбран JavaScript.

JavaScript— [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), который поддерживает [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [функциональный](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стили.

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам [приложений](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Наиболее широкое применение находит в [браузерах](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) как язык сценариев для придания [интерактивности](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) веб-страницам.

Основные архитектурные черты: [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [слабая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [прототипное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), функции как [объекты первого класса](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0).

В результате выполнения курсовой работы было разработано приложение, а также написан глоссарий.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc8955993)

[1 аНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc8955994)

[1.1 Описание традиционной технологии 7](#_Toc8955995)

[1.2 Постановка задачи 11](#_Toc8955996)

[2 Проектирование системы 12](#_Toc8955997)

[2.1 Проектирование бизнес логики 12](#_Toc8955998)

[2.2 Проектирование базы данных 14](#_Toc8955999)

[2.3 Проектирование пользовательского интерфейса 16](#_Toc8956000)

[3 Описание процесса реализации 17](#_Toc8956001)

[3.1 Выбор СУБД 17](#_Toc8956002)

[3.2 Выбор языка программирования 21](#_Toc8956003)

[3.3 Реализация ресурса 24](#_Toc8956004)

[4 Руководство пользователя 46](#_Toc8956005)

[ЗАКЛЮЧЕНИЕ 52](#_Toc8956006)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc8956007)

[ПРИЛОЖЕНИЕ А 54](#_Toc8956008)

[ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА 59](#_Toc8956009)

# Введение

Основная цель проекта заключается в том, чтобы предоставить качественный и быстрый сервис для возможности оставлять отзывы о товарах, давать советы и делиться лайфхаками.

Сервис осуществляет исключительно информативные услуги. То есть запуск проекта не подразумевает создание производства.

Огромное количество разновидностей товара заставляет покупателей задумываться о том, какой товар лучше? У покупателей возникает потребность посоветоваться с знающими людьми или почитать отзывы о товаре, что делает сервисы предоставляющие возможность оставлять и читать отзывы, делиться советами или лайфхаками весьма популярными.

Внутренними предпосылками для реализации проекта являются низкий уровень инвестиций в проект, отсутствие коммерческих рисков, простота организации и ведения бизнеса.

# аНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Описание традиционной технологии

Огромное количество разновидностей товара заставляет покупателей задумываться о том, какой товар лучше? У покупателей возникает потребность посоветоваться с знающими людьми или почитать отзывы о товаре, что делает сервисы предоставляющие возможность оставлять и читать отзывы, делиться советами или лайфхаками весьма популярными.

В качестве примера можно рассмотреть сайт otzovik.com

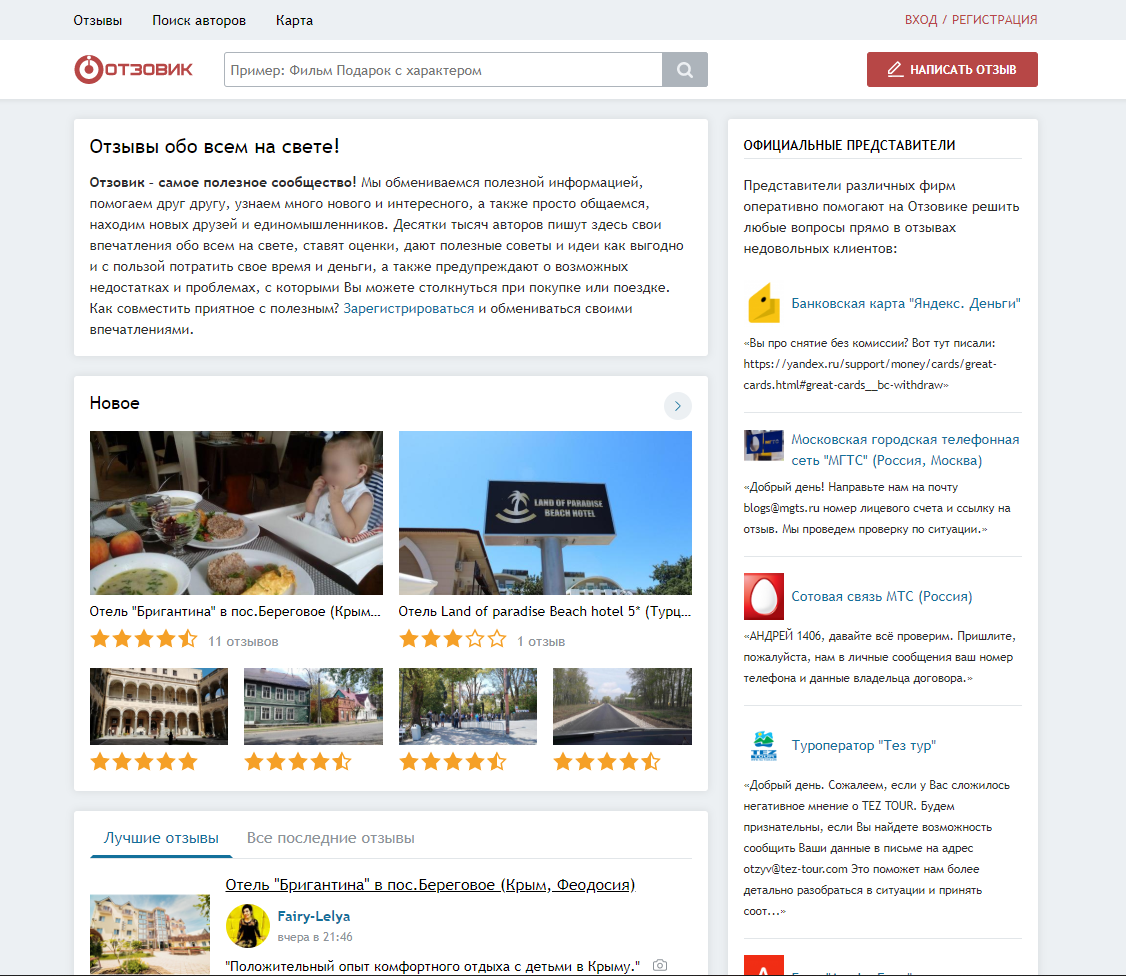


Рисунок 1.1 – Сайт “отзовик”

Существует также схожий сайт recommend.ru

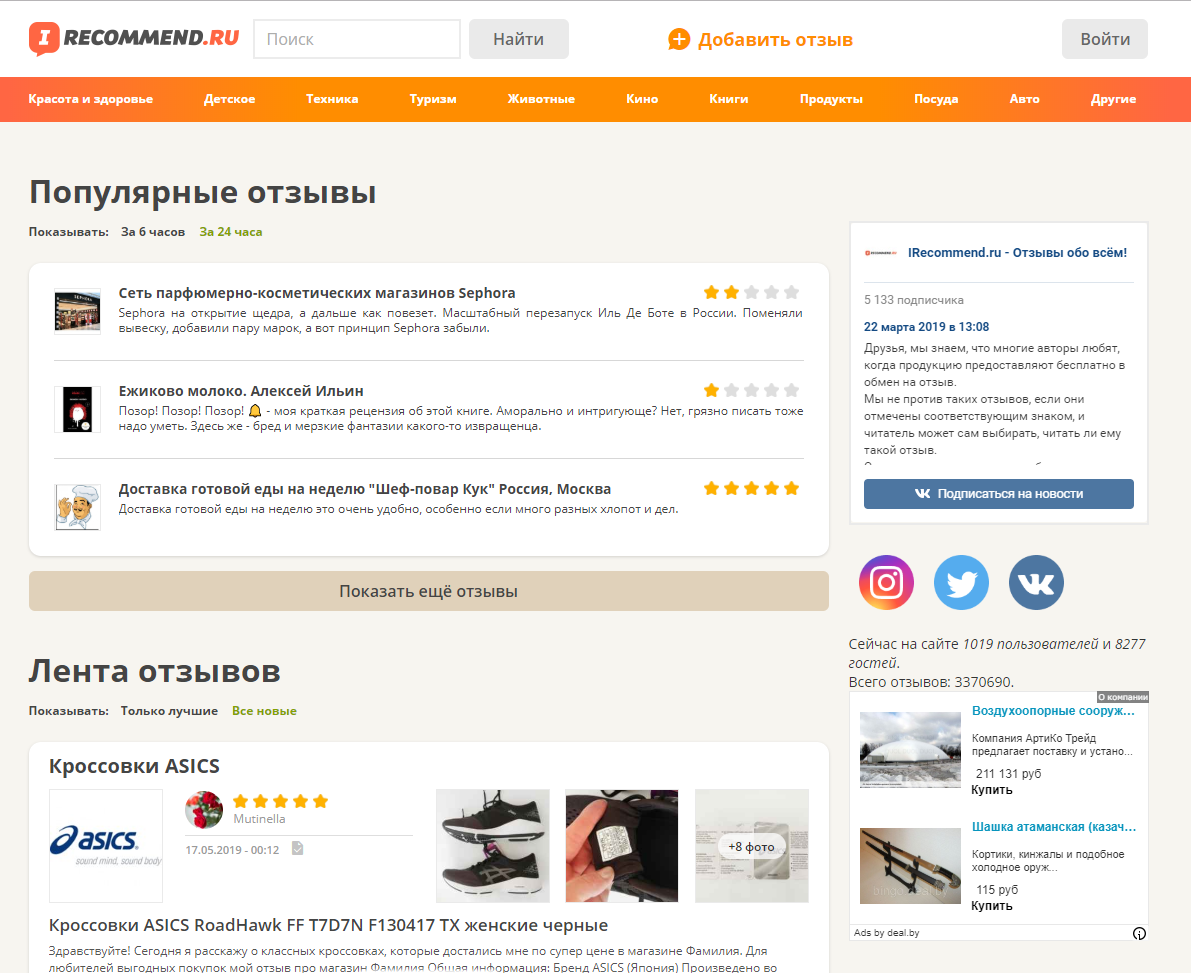


Рисунок 1.2 – Сайт “recommend.ru”

Данные сайты позволяют оставлять отзывы о любом товаре или услуге, возможность комментирования и имеют довольно удобную панель навигации.



Рисунок 1.3 – панель навигации сайта “recommend.ru”

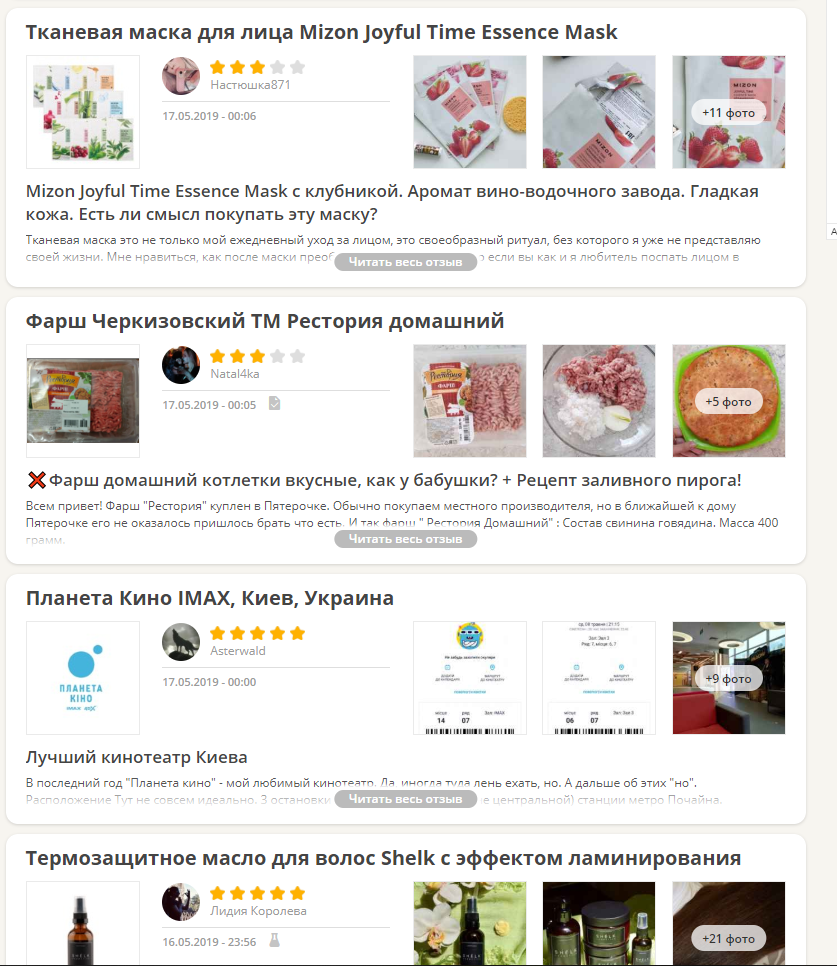


Рисунок 1.4 – лента постов сайта “recommend.ru”

Сайты адаптивны, хорошо структурированы. Пользователь имеет возможность сортировать посты по различным критериям.

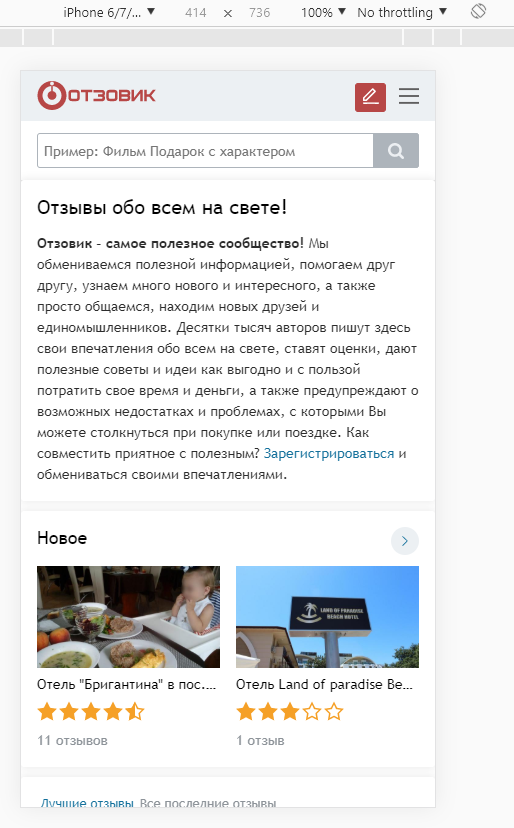


Рисунок 1.5 – пример адаптивности для сайта “отзовик”

## Постановка задачи

Курсовой проект преследует задачу создания сервиса, содержащего отзывы, советы и лайфхаки. На домашней странице должны присутствовать: список постов, формы регистрации и входа в систему. Пользователь-администратор может удалять ненужные посты или комментарии.

Персональный компьютер должен быть подключен к сети. Скорость передачи информации должна быть выше 0.5 мегабит в секунду. На персональном компьютере требуется иметь редактор кода и браузер для просмотра сайта. Требование к браузеру: браузер Internet Explorer 10 и выше, все остальные браузеры последних либо предпоследних версий.

# Проектирование системы

## Проектирование бизнес логики

**Бизнес-логика** ("Уровень бизнес логики") - [уровень абстракции](http://fkn.ktu10.com/?q=node/9256) системы (по сути "выше некуда"), в котором рассматриваются только поставленные перед ней задачи, без вопроса реализации.

Например, бизнес логика рассматривает:

* действующих лиц системы (пользователей с разными правами, разные типы существ в мире игры или иные сущности, о которых может говорить обычный человек (не программист), когда формулирует задачу);
* связи между сущностями (опять же понятные обычному человеку, когда он говорит о системе).

Именно исходя из бизнес-логики начинают формировать [архитектуру проекта](http://fkn.ktu10.com/?q=node/9261).

Для данного проекта наиболее важным этапом в проектировании бизнес-логики является разработка требований к сервису. Ниже приводятся основные функциональные требования, на основе которых будет строиться бизнес-логика:

* Система должна разделять пользователей на авторизованных, неавторизованных и пользователя-администратора.
* Система позволяет пользователю пройти регистрацию или авторизацию с помощью соответствующих форм.
* Система отображает текущего пользователя в верхней боковой панели навигации.
* Система удаляет данные о авторизованном пользователе из браузера и делает его неавторизованным, при нажатии кнопки выход.
* При попытке регистрации пользователем система проверяет на отсутствие уже существующего пользователя с такими же данными.
* Система сохраняет данные о пользователе в браузере во избежание повторного входа в систему им при каждом обновлении или закрытии страницы.
* Система предоставляет доступ к страницам со всеми блюдами при нажатии на какой-либо ресторан.
* Система идентифицирует пользователя-администратора по специальному логину и паролю.
* Пользователь-администратор, по нажатии на кнопку удалить пост или удалить комментарий имеет возможность удалить их
* Пользователь-администратор, по нажатии на кнопку редактировать данные пользователей, перенаправляется на страницу с формой изменения данных пользователей, которая содержит все поля соответствующей таблицы из базы данных позволяет их изменять.
* Система позволяет сохранять посты, добавлять посты, оставлять комментарии, ставить оценки.

Бизнес-логика строится на основе требований, указанных выше, и вся функциональность реализует их. Основная логика взаимодействия пользователей с системой отображается на рисунке 2.1.

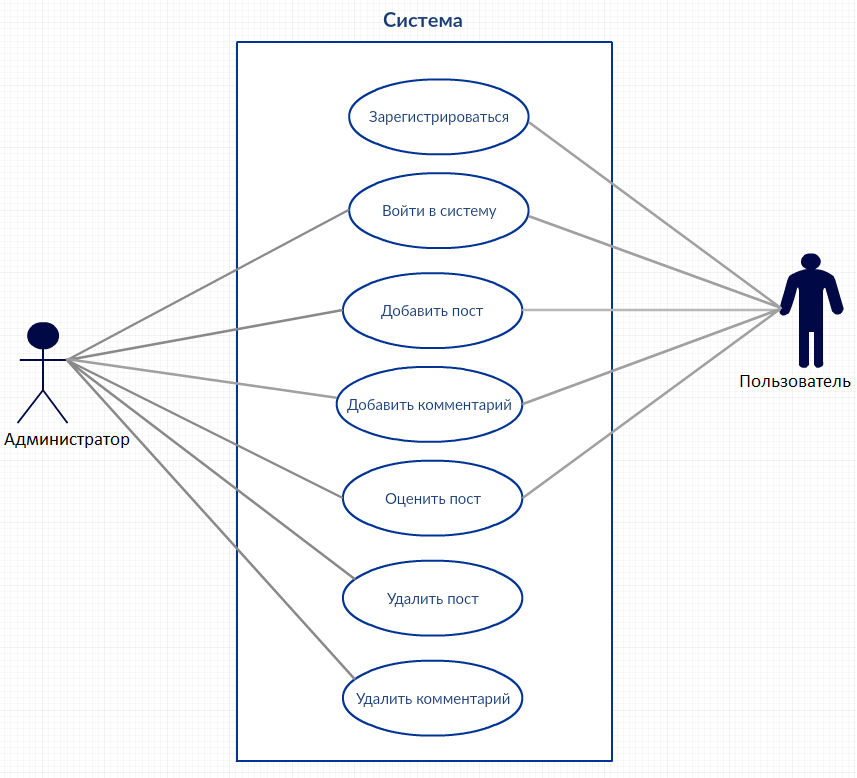


Рисунок 2.1 – Диаграмма вариантов использования

## Проектирование базы данных

Базой данных (БД) называется организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность сведений об объектах, процессах, событиях или явлениях, относящихся к некоторой предметной области, теме или задаче. Она организована таким образом, чтобы обеспечить информационные потребности пользователей, а также удобное хранение этой совокупности данных, как в целом, так и любой ее части.

Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных.

База данных предназначена для хранения данных о постах, комментариях, пользователей, сохранениях и оценках. Пользователями системы, использующей БД, являются посетители сайта и администратор. Функции системы, связанные с использованием БД: регистрация пользователей, работа со списком постов и комментариев.

Требуется соблюсти соответствие базы данных трём нормальным формам. Нормальная форма – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Исходя из функций и задач системы и требований 1НФ выделим сущности базы данных: Тип пользователя (user\_types), Пользователи (users), Посты (posts), Комментарии (comments), Категории (category), Оценки (likes), Сохранённые посты (saves). Так же при создании таблиц необходимо определить атрибуты сущностей (поля таблиц).

Сущность «Тип пользователя»: уникальный идентификатор (IDUT), название типа (name);

Сущность «Пользователи»: уникальный идентификатор (IDU), логин (login), пароль (password), идентификатор типа пользователя (IDUT).

Сущность «Категории»: уникальный идентификатор (IDC), название категории (name).

Сущность «Посты»: уникальный идентификатор (IDP), заголовок (header), идентификатор пользователя, который создал пост (IDU), текст поста (text), идентификатор категории (IDC), дата и время создания поста (datetime).

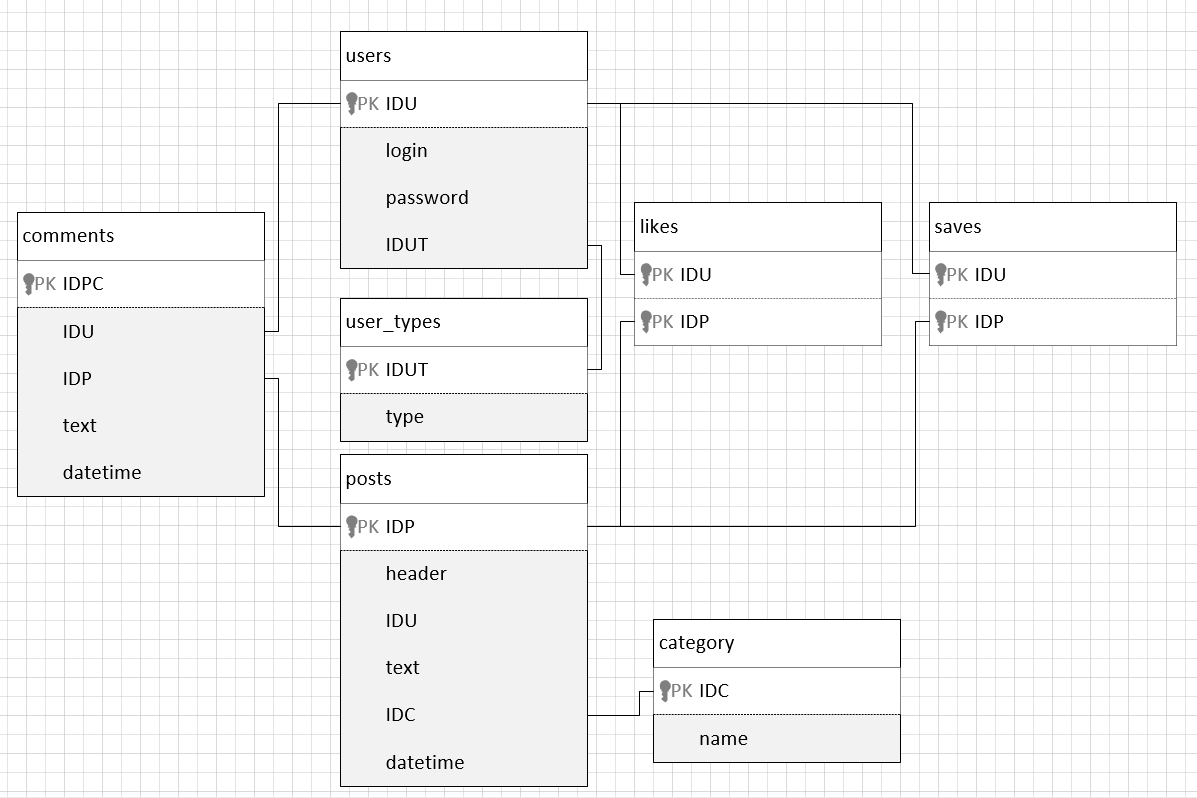
Сущность «Оценки»: идентификатор пользователя (IDU), идентификатор поста(IDP).

Сущность «Сохранённые посты»: идентификатор пользователя (IDU), идентификатор поста(IDP).

Отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут неприводимо зависит от Первичного Ключа (ПК).

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы. Третья нормальная форма соблюдается за счёт создания таблиц с едой и поставщиками, чем мы исключаем возможность появления транзитивных связей.

Схема базы данных представлена на рисунке 2.2.

Рисунок 2.2 – Схема базы данных

## Проектирование пользовательского интерфейса

Интерфейсы существуют, чтобы люди могли взаимодействовать с нашим миром. Через интерфейс мы можем прояснить, проиллюстрировать, дать возможность, показать взаимосвязь, объединить людей или разделить, управлять ожиданиями и давать доступ к услугам.

Дизайн пользовательского интерфейса является фактором, оказывающим влияние на три основных показателя качества программного продукта: его функциональность, эстетику и производительность [6].

Чтобы интерфейс эффективно помогал людям добиваться своих целей, он должен обладать следующими характеристиками:

* Во-первых, он должен быть узнаваемым, а его назначение — очевидным для пользователя.
* Во-вторых, люди должны понимать, с чем они взаимодействуют через интерфейс.
* Наконец, процесс взаимодействия с интерфейсом должен быть предсказуемым. В интерфейс можно внести некую загадочность или элементы игры, но вот путаницы быть не должно.

В данной системе всё это реализуется за счёт того, что все кнопки и ссылки обозначены словами действий, которые они осуществляют; сообщения об ошибках содержат подробную информацию о причине её возникновения.

# Описание процесса реализации

## Выбор СУБД

Несмотря на то, что все системы управления базами данных выполняют одну и ту же основную задачу (т.е. дают возможность пользователям создавать, редактировать и получать доступ к информации, хранящейся в базах данных), сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности каждой СУБД могут существенно отличаться. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка.

При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также убедиться, что она будет хорошо интегрироваться с другими продуктами, которые уже используются. Кроме того, во время выбора следует принять во внимание стоимость системы и поддержки, предоставляемой разработчиком.

Сравниваются некоторые из популярных СУБД и на результате данного анализа выбирается подходящая для данного проекта.

* + 1. **Microsoft SQL Server**

Это система управления базами данных, движок которой работает на облачных серверах, а также локальных серверах, причем можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска Microsoft SQL Server 2016, Microsoft адаптировала продукт для операционной системы Linux, а на Windows-платформе он работал изначально. Одной из уникальных особенностей версии 2016 года является temporal data support (временная поддержка данных), которая позволяет отслеживать изменения данных с течением времени. Последняя версия Microsoft SQL-сервер поддерживает dynamic data masking (динамическую маскировку данных), которая гарантирует, что только авторизованные пользователи будут видеть конфиденциальные данные.

Достоинства:

* продукт очень прост в использовании;
* текущая версия работает быстро и стабильно;
* движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
* вы сможете получить доступ к визуализации на мобильных устройствах;
* он очень хорошо взаимодействует с другими продуктами Microsoft.

Недостатки:

* цена для юридических лиц оказывается неприемлемой для большей части организаций;
* даже при тщательной настройке производительности корпорация SQL Server способен занять все доступные ресурсы;
* сообщается о проблемах с использованием службы интеграции для импорта файлов;
* идеально подходит для крупных организаций, которые уже используют ряд продуктов Microsoft.
  + 1. **PostgreSQL**

PostgreSQL является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая Linux. Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

Движок БД может быть размещен в ряде сред, в том числе виртуальных, физических и облачных. Самая свежая версия, PostgreSQL 9.5, предлагает обработку больших объемов данных и увеличение числа одновременно работающих пользователей. Безопасность была улучшена благодаря поддержке DBMS\_SESSION.

Достоинства:

* является масштабируемым и способен обрабатывать терабайты данных.
* поддерживает формат json;
* существует множество предопределенных функций;
* доступен ряд интерфейсов;
* цена и открытость.

Недостатки:

* документация туманна, поэтому, возможно, ответы на некоторые вопросы придется искать в интернете;
* скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения;
* идеально подходит для организаций с ограниченным бюджетом, но квалифицированными специалистами, когда требуется возможность выбрать свой интерфейс и использовать json.
  + 1. **MongoDB**

Бесплатная база данных, которая имеет коммерческую версию - MongoDB, она предназначена для приложений, которые используют как структурированные, так и неструктурированные данные. Ядро является очень гибким и работает при подключении базы данных к приложениям через драйверы MongoDB. Существует широкий выбор доступных драйверов, поэтому легко найти драйвер, который будет работать с требуемым языком программирования.

Поскольку изначально система MongoDB не была разработана для обработки моделей реляционных данных (хотя может это выполнять), могут возникнуть проблемы производительности, если вы попытаетесь использовать её таким образом. Однако, движок предназначен для обработки различных данных, которые нельзя отнести к реляционным, и может хорошо справляться там, где другие движки работают медленно или бессильны.

MongoDB 3.2 - это последняя версия, и она имеет новую подключаемую систему движков хранения. Документы могут быть проверены в процессе обновления или выполнения вставок, а функции текстового поиска были улучшены. Новая способность частичного индексирования может привести к более высокой производительности, уменьшая размер индексов.

Достоинства:

* скорость и простота в использовании;
* движок поддерживает json и другие традиционные документы NoSQL;
* данные любой структуры могут быть сохранены/прочитаны быстро и легко.

Недостатки:

* SQL не используется в качестве языка запросов;
* инструменты для перевода SQL-запросов в MongoDB доступны, но их следует рассматривать именно как дополнение;
* программа установки может занять много времени;
* подходит для организаций, работающих с разнородными данными, которые тяжело поддаются классификации. Для внедрения потребуются высококлассные специалисты.
  + 1. **MySQL**

MySQL - одна из самых популярных баз данных для веб-приложений. Фактически, является стандартом de facto для веб-серверов, которые работают под управлением операционной системы Linx. MySQL - это бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность. Существуют специальные платные версии, предназначенные для коммерческого использования. В бесплатной версии наибольший упор делается на скорость и надежность, а не на полноту функционала, который может стать и достоинством и недостатком - в зависимости от области внедрения.

Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Эта СУБД позволяет выбирать различные движки для системы хранения, которые позволяют менять функционал инструмента и выполнять обработку данных, хранящихся в различных типах таблиц. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц. Она также имеет простой в использовании интерфейс, и пакетные команды, которые позволяют удобно обрабатывать огромные объемы данных. Система невероятно надежна и не стремится подчинить себе все доступные аппаратные ресурсы.

Достоинства:

* распространяется бесплатно;
* прекрасно документирована;
* предлагает много функций, даже в бесплатной версии;
* пакет MySQL включен в стандартные репозитории наиболее распространённых дистрибутивов операционной системы Linux, что позволяет устанавливать её элементарно;
* поддерживает набор пользовательских интерфейсов;
* может работать с другими базами данных, включая DB2 и Oracle.

Недостатки:

* придётся потратить много времени и усилий, чтобы заставить MySQL выполнять несложные задачи, хотя другие системы делают это автоматически, например, создавать инкрементные резервные копии;
* отсутствует встроенная поддержка XML или OLAP;
* для бесплатной версии доступна только платная поддержка;
* идеально подходит для: организаций, которым требуется надежный инструмент управления базами данных, но бесплатный.

База данных данного проекта не требует очень сложных операций над данными, необходим удобный рабочий интерфейс для работы с базой данных, проект предусматривает создание реляционной базы данных. Учитывая приведённый выше анализ различных СУБД и опыт работы с одной из СУБД разработчика системы, что так же является важным фактором, так как от этого зависит качество созданной базы данных и скорость её созданий, выбирается СУБД MySQL.

## Выбор языка программирования

Данный проект реализуется на языке программирования JavaScript. Так же используется язык разметки HTML и язык для определения стилей CSS. Данный выбор обусловлен тем, что JS является одним из ведущих языков для разработки web-ресурсов. Язык имеет очень большое сообщество, за счёт чего большую поддержку и быстрое эффективное развитие. Бизнес логика будет реализована с помощью фреймворка NodeJS. Пользовательский интерфейс – библиотека ReactJS/Redux. Ниже приведены описания данных технологий, их плюсы и минусы в сравнении с другими языками программирования.

React - это библиотека JavaScript, которая используется для создания пользовательского интерфейса. React был создан компанией Facebook, а первый релиз библиотеки увидел свет в марте 2013 года. Текущей версий на данный момент (октябрь 2017 года) является версия React v16.0. Первоначально React предназначался для веба, для создания веб-сайтов, однако позже появилась платформа React Native, которая уже предназначалась для мобильных устройств.

React представляется идеальный инструмент для создания масштабируемых веб-приложений (в данном случае речь идет о фронтенде), особенно в тех ситуациях, когда приложение представляет SPA (одностраничное приложение).

React относительно прост в освоении, имеет понятный и лаконичный синтаксис.

Достоинства:

* простота и ясность кода за счёт технологии JSX и компонентного подхода;
* размер фреймворка: ядро react + flux занимает в 3 раза меньше angular2 либо незначительно меньше angular1. Зачем вместо того чтобы подгружать необходимый функционал с помощью модулей, создавать сборку-монстра;
* скорость работы: react благодаря подходу с виртуальны DOM и jsx значительно опережает конкурентов;
* однонаправленный поток данных: каждый кто создавал полноценное крупное приложение на angular/ember так или иначе сталкивались с проблемой потока данных и связей хранилище-модель-представления (а также их комбинацией). С аналогичной проблемой столкнулись и инженеры facebook, результатом их работы стал flux. Стоит отдать должное - решение справляется со своей задачей;
* компиляция в html на сервере: огромное преимущество решений на базе react. Да существует angular-server но он не полноценен, и работает с определенной натяжкой. Что касается React, то существует возможность (и инструменты которые ее реализуют) выдавать полностью автономную html страницу которая не требует повторной компиляции на клиенте (что еще и ускоряет процесс визуализации). Собственно, это позволяет приложению оставаться отчасти рабочим даже с выключенным или недоступным JavaScript на клиенте;
* сущность jsx: даже если опустить сам факт, что это более продуманное и соответственно более быстрое решение (в плане скорости работы), сама сущность jsx дает еще преимущество при обработке ошибок. Что же произойдет, если вы допустите ошибку в JSX - приложение не пройдет сборку (не скомпилируется), выдав при этом номер строки, где допущена ошибка, а также перечень предупреждений, где потенциально могут возникнуть проблемы;
* философия: React в отличии от ember/angular создает впечатление программы, которая делает что-то одно, но делают это хорошо. В React вы не найдете "стандартно маршрутизатора" либо других заранее навязанных DI зависимостей;
* фреймворк на все случаи жизни.

Недостатки:

* отсутствие системы событий (отличную от нативных DOM событий);
* отсутствие работы с AJAX;
* плохая и непонятная документация.

Пользовательский интерфейс мог реализовываться с помощью фреймворка Angular, он быстрее на начальных этапах, так как реализует почти всевозможные функциональности, представленные в React npm-модулями. Однако за счёт небольшого размера проекта и необходимости простого и читабельного кода, так же компонентная философия позволяет создавать динамические страницы и легко их изменять, используется библиотека React.

Redux является предсказуемым контейнером состояния для JavaScript приложений. (Не путайте с WordPress фреймворком – [Redux Framework](https://reduxframework.com/).) Это позволяет вам создавать приложения, которые ведут себя одинаково в различных окружениях (клиент, сервер и нативные приложения), а также просто тестируются. Кроме того, это обеспечивает большой опыт отладки, например [редактирование кода в реальном времени в сочетание с time traveling](https://github.com/gaearon/redux-devtools). Вы можете использовать Redux вместе с [React](https://facebook.github.io/react/) или с любой другой view-библиотекой.

Для реализации бизнес-логики используется программная платформа NodeJS, пакетный менеджер NPM и фреймворк Express.

Node.js – программная платформа, основанная на движке [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) (транслирующем [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) взаимодействовать с устройствами [ввода-вывода](https://ru.wikipedia.org/wiki/%D0%92%D0%B2%D0%BE%D0%B4-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4) через свой [API](https://ru.wikipedia.org/wiki/API) (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль [веб-сервера](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи [NW.js](https://ru.wikipedia.org/wiki/NW.js), AppJS или [Electron](https://ru.wikipedia.org/wiki/Electron) для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит [событийно-ориентированное](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [асинхронное (или реактивное)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программирование с [неблокирующим вводом/выводом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F).

Основной фактор для использования данного фреймворка в проект – однопоточность, что обеспечивает меньшую нагрузку на сервер.

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.

В данном фреймворке очень удобно реализована система маршрутизации для сервера, что является одной из основных причин его использования.

## Реализация ресурса

Основная идея архитектуры «клиент-сервер» состоит в разделении сетевого приложения на несколько компонентов, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом. Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

В проекте используется Прокси-сервер. Во-первых, действует как посредник, помогая пользователям получить информацию из Интернета и при этом обеспечивая защиту сети. Во-вторых, сохраняет часто запрашиваемую информацию в кэш-памяти на локальном диске, быстро доставляя ее пользователям без повторного обращения к Интернету.

Для доступа к тем или иным сетевым сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Рассмотрим возможные граничные значения:

«Тонкий» клиент. Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами статического HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере. Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент. Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных). Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

Для проекта сначала необходимо создать базу данных, чтобы прописывать логику приложения по работе с данными.

Создаётся база данных в визуальном интерфейсе phpMyAdmin. Для созданий таблиц, их ключей и внешних ключей не требуется прописывать код вручную.

Строится структура проекта, которая соответствует философии использующихся технологий. В данном случае все модули разделены на относящиеся к серверу и к клиентской части (см. рисунок 3.1). Клиентская часть находится в папке “src”. В клиентской части файлы помещены в папки страниц сайта, на которых они используются. Так же есть папка с элементами, использующимися на всех страницах. Здесь же содержится файл хранящий модуль хранилища для технологии Redux. Код с созданием этого хранилища находится в файле “store/index.js”. В папке “store” лежит “reducers” который тоже необходимы для технологии Redux. Так же папке “src” лежат компоненты которые соответствуют философии фреймворка React (см. рисунок 3.2). В серверной части находятся модули, содержащие в себе реализацию бизнес-логики и настройки подключения к базе данных (см. рисунок 3.3).

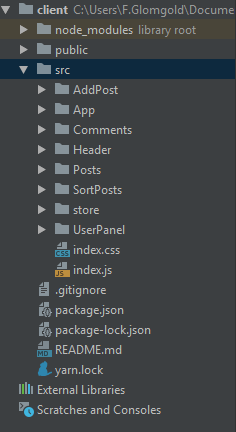


Рисунок 3.1 – Основная иерархия папок

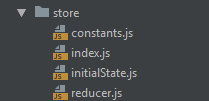


Рисунок 3.2 – Файловая иерархия папки store

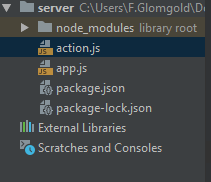


Рисунок 3.3 – Иерархия папки сервер

Создаётся домашняя страница и её визуальные компоненты, а именно навигация и ссылки на формы регистрации и авторизации. Ниже приведён пример реализации главного компонента навигации:

***import*** *React* ***from*** *'react';****import*** *{render}* ***from*** *'react-dom';****import*** *{Provider}* ***from*** *'react-redux'****import*** *'./index.css';****import*** *App* ***from*** *'./App/App';****import*** *storeFactory* ***from*** *'./store/index'****const*** *store=storeFactory();  
  
render(  
 <Provider store={store}>  
 <App/>  
 </Provider>,  
 document.getElementById('root')  
);*

Где “App” представляет собой реализацию провайдера (компонент redux для управления состоянием компонента). Код для реализации главного компонента App представлен ниже:

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

*import Header from '../Header/Header'*

*import Posts from '../Posts/Posts'*

*import UserPanel from '../UserPanel/UserPanel'*

*import './App.css';*

*class App extends Component {*

*render(){*

*return(*

*<div className='App'>*

*<Header/>*

*<main className='App\_\_main'>*

*<Posts/>*

*<UserPanel/>*

*</main>*

*</div>*

*)*

*}*

*}*

*export default App*

“Header” – шапка сайта. В ней находится название приложения. Код компонента для представлен ниже. Собственно этот компонент не несёт за собой никакой функциональности. Код этого компонента является не сложным и несёт в себе практически обычную разметку.:

*import React from 'react';*

*import './Header.css'*

*const Header=()=>{*

*return(*

*<header className='Header'>*

*<h1 className='Header\_h1'>callbackHUB</h1>*

*</header>*

*)*

*};*

*export default Header;*

“Posts” - это, собственно, и есть главная часть которая предоставляет список постов с комментариями. Эта часть перерендывается когда изменяется тип сортировки или пользователь входит в систему. Полный код для этого компонента представлен в приложении А. Код метода render для компонента представлен ниже:

*render() {*

*return (*

*<section className='Posts'>*

*<SortPosts/>*

*{*

*(this.props.login && this.props.password) ?*

*<AddPost/> :*

*''*

*}*

*<ul>*

*{this.props.posts.map((p) =>*

*<li key={p.IDP} className='Posts\_\_post' value={p.IDP}>*

*<div className='Posts\_\_likes'>*

*<div*

*className={p.liked ? 'Posts\_\_like-icon Posts\_\_like-icon\_liked' : 'Posts\_\_like-icon'}*

*onClick={this.submitLike}*

*>*

*</div>*

*{p.likes}*

*</div>*

*<section className='Posts\_\_block '>*

*<header className='Posts\_\_header'>*

*<h2 className='Posts\_\_h'>{p.header}</h2>*

*<span className='Posts\_\_category'>{p.category}</span>*

*</header>*

*<article className='Posts\_\_text'>*

*{p.text}*

*</article>*

*<footer className='Posts\_\_footer'>*

*<div className='Posts\_\_saves'>*

*<div*

*className={p.saved ? 'Posts\_\_save-icon Posts\_\_save-icon\_saved' : 'Posts\_\_save-icon'}*

*onClick={this.submitSave}*

*>*

*</div>*

*{p.saves}*

*</div>*

*<div className='Posts\_\_comments'>*

*<div*

*className={p.show\_comments ? 'Posts\_\_comments-icon Posts\_\_comments-icon\_show' : 'Posts\_\_comments-icon'}*

*onClick={this.submitComments}*

*>*

*</div>*

*{p.comments\_l}*

*</div>*

*<div className='Posts\_\_footer\_info-block'>*

*<span className='Posts\_\_author'>{p.author}</span>*

*<span className='Posts\_\_date'>{p.date}</span>*

*</div>*

*</footer>*

*{p.show\_comments ? <Comments IDP={p.IDP}/> : ''}*

*</section>*

*</li>*

*)}*

*</ul>*

*</section>*

*)*

*}*

Компонент “SortPost” отвечает за сортировку постов. Может сортировать посты по дате и по рейтингу. Так же, если пользователь выполнил вход в систему, то выполнить отображение только сохранённых постов. Код представлен ниже:

*const SortPosts = ({sort=C.SORT\_TYPES.SORT\_BY\_DATE,sortBy=f=>f,login='',password='',loadPosts=f=>f})=>{*

*const change=e=>{*

*const by=e.target.value;*

*fetch(`http://localhost:3110/?action=${C.LOAD\_POSTS}&sort=${by}&login=${login}&password=${password}`)*

*.then(j=>j.json())*

*.then(posts=>sortBy(by,posts))*

*.catch((err)=>console.error(err));*

*};*

*const loadSavePosts=()=>{*

*fetch(`http://localhost:3110/?action=${C.LOAD\_POSTS}&saves=true&login=${login}&password=${password}`)*

*.then(j=>j.json())*

*.then(posts=>loadPosts(posts))*

*.catch((err)=>console.error(err));*

*};*

*return(*

*<section className='SortPosts'>*

*<div>*

*<span>Сортировка по: </span>*

*<label className='SortPosts\_\_sort-type'>*

*<input*

*type="radio"*

*name="sort"*

*onClick={change}*

*value={C.SORT\_TYPES.SORT\_BY\_DATE}*

*defaultChecked*

*className='SortPosts\_\_input'*

*/>*

*<span>дате</span>*

*</label>*

*<label className='SortPosts\_\_sort-type'>*

*<input*

*type="radio"*

*name="sort"*

*onClick={change}*

*value={C.SORT\_TYPES.SORT\_BY\_REIT}*

*className='SortPosts\_\_input'*

*/>*

*<span>рейтингу</span>*

*</label>*

*</div>*

*{*

*(login && password) ?*

*<label className='SortPosts\_\_sort-type'>*

*<input*

*type="radio"*

*name="sort"*

*onClick={loadSavePosts}*

*value={C.SORT\_TYPES.SORT\_BY\_REIT}*

*className='SortPosts\_\_input'*

*/>*

*<span>сохранённое</span>*

*</label> :*

*''*

*}*

*</section>*

*)*

*};*

*export default connect(*

*state=>({*

*sort:state.sort,*

*login:state.userPanelDate.login,*

*password:state.userPanelDate.password*

*}),*

*dispatch=>({*

*sortBy:((by,posts)=>{*

*dispatch({type:C.SORT,by});*

*dispatch({type:C.LOAD\_POSTS,posts})*

*}),*

*loadPosts:(posts=>{*

*dispatch({type:C.LOAD\_POSTS,posts})*

*})*

*})*

*)(SortPosts);*

Компонент “UserPanel” отвечает за регистрацию и авторизацию пользователей, а также вывод предупреждений при неправильном логине или пароле при входе или уже заняты данных при регистрации. Код представлен ниже:

*const UserPanel=({*

*userPanelDate={},*

*posts=[],*

*showHint=f=>f,*

*loggingIn=f=>f,*

*exit=f=>f,*

*showRegistrationList=f=>f,*

*showAuthorizationList=f=>f,*

*loadPosts=f=>f,*

*resetPosts=f=>f})=>*

*{*

*let userPanelType=()=>{*

*let \_login,\_password;*

*const submitAuthorization=(e)=>{*

*e.preventDefault();*

*fetch(`http://localhost:3110/?action=${C.GET\_INFO\_ABOUT\_USER}&login=${\_login.value}&password=${\_password.value}`)*

*.then(j=>{*

*if(j.status===404) showHint();*

*else {*

*fetch(`http://localhost:3110/?action=${C.LOAD\_POSTS}&sort=${C.SORT\_TYPES.SORT\_BY\_DATE}&login=${\_login.value}&password=${\_password.value}`)*

*.then(j=>j.json())*

*.then(posts=>{*

*loggingIn(\_login.value,\_password.value);*

*loadPosts(posts);*

*})*

*.catch((err)=>console.error(err));*

*}*

*});*

*};*

*const submitRegistration=(e)=>{*

*e.preventDefault();*

*fetch(`http://localhost:3110/?action=${C.ADD\_USER}&login=${\_login.value}&password=${\_password.value}`)*

*.then(j=>{*

*if(j.status===404) showHint();*

*else {*

*loggingIn(\_login.value,\_password.value);*

*}*

*});*

*};*

*const submitExit=()=>{*

*exit();*

*resetPosts();*

*};*

*switch (userPanelDate.userPanelType) {*

*case C.USER\_PANEL\_TYPES.AUTHORIZATION:*

*return <form onSubmit={submitAuthorization} className='UserPanel\_\_form'>*

*<h3 className='UserPanel\_\_h3'>Авторизация</h3>*

*<div className='UserPanel\_\_input-container'>*

*<input className='UserPanel\_\_input' ref={input=>\_login= input} type="text" placeholder='Логин' required />*

*</div>*

*<div className='UserPanel\_\_input-container'>*

*<input className='UserPanel\_\_input' ref={input=>\_password= input} type="password" placeholder='Пароль' required/>*

*</div>*

*{userPanelDate.hint ? <span className='UserPanel\_\_error'>Неправильнй <br/>логин логин или пароль</span> : <span> </span>}*

*<button className='UserPanel\_\_button'>ВОЙТИ</button>*

*<span className='UserPanel\_\_span' onClick={showRegistrationList}>регистрация</span>*

*</form>;*

*case C.USER\_PANEL\_TYPES.REGISTRATION:*

*return <form onSubmit={submitRegistration} className='UserPanel\_\_form'>*

*<h3 className='UserPanel\_\_h3'>Регистрация</h3>*

*<div className='UserPanel\_\_input-container'>*

*<input className='UserPanel\_\_input' ref={input=>\_login= input} type="text" placeholder='Логин' required/>*

*</div>*

*<div className='UserPanel\_\_input-container'>*

*<input className='UserPanel\_\_input' ref={input=>\_password= input} type="password" placeholder='Пароль' required/>*

*</div>*

*{userPanelDate.hint ? <span className='UserPanel\_\_error'>Ошибка регистрации</span> : <span> </span>}*

*<button className='UserPanel\_\_button'>СОЗДАТЬ АККАУНТ</button>*

*<span className='UserPanel\_\_span' onClick={showAuthorizationList}>войти</span>*

*</form>;*

*case C.USER\_PANEL\_TYPES.LOGGED\_IN:*

*return <section className='UserPanel\_\_section'>*

*<h3 className='UserPanel\_\_h3'>{userPanelDate.login}</h3>*

*<button className='UserPanel\_\_button' onClick={submitExit}>ВЫХОД</button>*

*</section>;*

*default: return <div>error</div>*

*}*

*};*

*return(*

*<aside className='UserPanel'>*

*{userPanelType()}*

*</aside>*

*)*

*};*

*export default connect(*

*state=>({*

*userPanelDate:state.userPanelDate,*

*posts:state.posts*

*}),*

*dispatch=>({*

*showHint:(()=>{*

*dispatch({type:C.SHOW\_HINT})*

*}),*

*loggingIn:((login,password,posts)=>{*

*dispatch({type:C.LOGGING\_IN,login,password,posts});*

*}),*

*exit:(()=>{*

*dispatch({type:C.EXIT})*

*}),*

*showRegistrationList:(()=>{*

*dispatch({type:C.REGISTRATION\_LIST})*

*}),*

*showAuthorizationList:(()=>{*

*dispatch({type:C.AUTHORIZATION\_LIST})*

*}),*

*loadPosts:(posts=>{*

*dispatch({type:C.LOAD\_POSTS,posts})*

*}),*

*resetPosts:(()=>{*

*dispatch({type:C.RESET\_POSTS})*

*})*

*})*

*)(UserPanel);*

Все остальные глупые компоненты, то есть те, которые только отображают полученные без какой-либо функциональности свойства, реализуются по тому же принципу. Далее «глупые» компоненты будут называться компонентами, а компоненты, содержащие в себе методы – контейнерами.

Панель пользователя отображает текущего пользователя, так как при установке компонента на страницу проверяет наличие данных в локальном хранилище данных браузера(localStorage), что означает, что пользователь ранее зашёл в систему и не выходил из неё. Имеется метод для выхода из системы, при его вызове все данные из локального хранилища браузера удаляются и пользователь становится не авторизованным. Логика прописана в файле “reducer”, который находится в папке “store”. Как говорилось ранее эта папка нужна для управления данными. Здесь же осуществляется и управление локальными хранилищем браузера:

*import C from './constants'*

*//import fetch from 'isomorphic-fetch'*

*export const posts = (state=[],action) =>{*

*switch (action.type) {*

*case C.LOAD\_POSTS:*

*return [...action.posts];*

*case C.RESET\_POSTS:*

*return state.map(p=> {return{...p,liked:false,saved:false}});*

*case C.LIKE:*

*return state.map(p=>p.IDP===action.IDP ? {...p,liked:true,likes:(p.likes+1)} : {...p});*

*case C.DELETE\_LIKE:*

*return state.map(p=> p.IDP===action.IDP ? {...p,liked:false,likes:p.likes-1} : {...p});*

*case C.SAVE:*

*return state.map(p=>p.IDP===action.IDP ? {...p,saved:true,saves:(p.saves+1)} : {...p});*

*case C.DELETE\_SAVE:*

*return state.map(p=> p.IDP===action.IDP ? {...p,saved:false,saves:p.saves-1} : {...p});*

*case C.COMMENTS\_SHOWN:*

*return state.map(p=>p.IDP===action.IDP ? {...p,show\_comments:true, comments:action.comments, comments\_l:action.comments.length}:{...p});*

*case C.COMMENTS\_HIDDEN:*

*return state.map(p=>p.IDP===action.IDP ? {...p,show\_comments:false}:{...p});*

*default:*

*return state;*

*}*

*};*

*export const addPostPanel=(state='false', action)=>{*

*console.log(state,action);*

*switch (action.type) {*

*case C.ADD\_POST\_PANEL:*

*return action.show;*

*default: return state;*

*}*

*};*

*export const sort = (state=C.SORT\_TYPES.SORT\_BY\_DATE,action) =>{*

*switch (action.type) {*

*case C.SORT:*

*return action.by;*

*default:*

*return state;*

*}*

*};*

*export const userPanelDate = (state={},action) =>{*

*switch (action.type) {*

*case C.SHOW\_HINT:*

*return {...state,hint:true};*

*case C.LOGGING\_IN:*

*localStorage.setItem('login',action.login);*

*localStorage.setItem('password',action.password);*

*return {...state,login:action.login,password:action.password,userPanelType:C.USER\_PANEL\_TYPES.LOGGED\_IN,hint:false};*

*case C.EXIT:*

*localStorage.removeItem('login');*

*localStorage.removeItem('password');*

*return {...state,login:'',password:'',userPanelType: C.USER\_PANEL\_TYPES.AUTHORIZATION};*

*case C.REGISTRATION\_LIST:*

*return {...state,hint:false,userPanelType:C.USER\_PANEL\_TYPES.REGISTRATION};*

*case C.AUTHORIZATION\_LIST:*

*return {...state,hint:false,userPanelType:C.USER\_PANEL\_TYPES.AUTHORIZATION};*

*default:*

*return state;*

*}*

*};*

Реализация функции, которая устанавливает соединение с БД на серверной стороне:

***const*** *pool = mysql.createPool({  
 database: 'kursach',  
 host: "127.0.0.1",  
 user: "root",  
 password: "qwerty"  
});*

Все остальные функции get-запросов со стороны клиента к серверу будут реализованы аналогично, в некоторых из них будут передаваться параметры в строке url, защита от sql-инъекций реализована с помощью внутреннего метода escape объекта pool.

Форма регистрации и авторизации абсолютно аналогичны в плане визуального интерфейса. При регистрации и авторизации происходит проверка данных форм на стороне браузера, что не нагружает сервер.

Прописаны правила проверки, соответствующие требованиям. При авторизации в базе данных проверяется наличие пользователя с таким именем и паролем, после при нахождении соответствующей записи возвращается положительный результат, в обратном случае, если пользователь не найден – ошибка о несуществующем пользователе, если пароль не верен – ошибка о неверности пароля:

const getInfoAboutUser=(data)=>{

return new Promise((resolve, reject) => {

const {login,password} = data;

const query = pool.query(`

SELECT users.IDU, user\_types.type

FROM users

INNER JOIN user\_types

ON users.IDUT=user\_types.IDUT

WHERE login = ${pool.escape(login)} AND password = ${pool.escape(password)}`, (err, res) => {

if (err) reject(err);

if (!res.length) {

console.log('getInfoAboutUser-');

reject('user not found');

}

else {

const IDU = res[0].IDU;

const type = res[0].type;

console.log('getInfoAboutUser+');

resolve({...data,IDU,type});

}

});

});

};

Компонент со списком постов при загрузке делает запрос к серверу на получение всех ресторанов из базы данных в виде массива объектов со свойствами название, описание, изображение, имя, адрес, телефон:

componentWillMount() { fetch(`http://localhost:3110/?action=${C.LOAD\_POSTS}&sort=${C.SORT\_TYPES.SORT\_BY\_DATE}&login=${this.props.login}&password=${this.props.password}`)

.then(j=>j.json())

.then(posts=> {

this.props.loadPosts(posts)

})

}

Таким образом при обновлении страницы список так же обновляется в соответствии с актуальной информацией из базы.

В каждом посте содержится информация о заголовке, тексте, авторе, дате, количестве оценок и людей, которые сохранили пост. Так же в посте находится элемент с комментариями который изначально отображает их количество и при нажатии на этот элемент выпадает компонент с комментариями. Комментарии содержат автора, текст и дату. Код компонента представлен ниже:

import React from 'react';

import {connect} from 'react-redux';

import C from "../store/constants";

import './Comments.css'

const Comments = ({IDP='',getComments=f=>f,login='',password='',show\_comments=f=>f})=>{

const comments=getComments(IDP);

let \_text;

const addComments=(e)=>{

e.preventDefault();

fetch(`http://localhost:3110/?action=${C.ADD\_COMMENT}&IDP=${IDP}&text=${\_text.value}&login=${login}&password=${password}`)

.then(j=>j.json())

.then((comments)=>{

\_text.value="";

show\_comments(IDP,comments);

})

.catch((err)=>console.error(err));

};

return (

<section className='Comments'>

<ul className='Comments\_\_ul'>

{

comments.map((c=>

<li key={c.IDPC} className='Comments\_\_ul\_\_li'>

<h4 className='Comments\_\_ul\_\_li\_\_h4'>{c.login}</h4>

<p className='Comments\_\_ul\_\_li\_\_p'>{c.text}</p>

<span className='Comments\_\_ul\_\_li\_\_datetime'>{c.datetime}

</span>

</li>

))

}

</ul>

{

(login && password) ?

<form onSubmit={addComments} className='Comments\_\_form'>

<div className='Comments\_\_form\_\_input-container'>

<input type="text" className='Comments\_\_form\_\_input' ref={input=>\_text=input} required/>

</div>

<button className='Comments\_\_form\_\_button'>Отправить</button>

</form> :

''

}

</section>

)

};

export default connect(

state=>({

login:state.userPanelDate.login,

password:state.userPanelDate.password,

getComments:((IDP)=>{

return state.posts.filter((p)=>p.IDP===IDP)[0].comments;

})

}),

dispatch=>({

addComment:(comment=>{

dispatch({type:C.ADD\_COMMENT,comment})

}),

show\_comments:((IDP,comments)=>{

dispatch({type:C.COMMENTS\_SHOWN,IDP,comments})

})

})

)(Comments);

Контейнер, который отображает текущие заказы пользователя обращается к конкретной таблице из базы данных и так же проделывает несколько проверок включая проверку прошёл ли пользователь авторизацию:

Последовательность функций вызова при загрузке постов:

case 'LOAD\_POSTS':

if(login && password) {

getInfoAboutUser({...req.query,res})

.then(loadPosts)

.then(sendAnswer)

.catch(sendError)

}

Функция загрузки постов:

*const loadPosts=(data)=>{*

*return new Promise((resolve, reject) => {*

*const {sort,IDU,saves} = data;*

*let searchLikes='';*

*let searchLikesAndSaves='';*

*if (IDU) searchLikesAndSaves = `*

*,(SELECT count(\*) FROM likes WHERE likes.IDU=${pool.escape(IDU)} AND likes.IDP=posts.IDP) as liked*

*,(SELECT count(\*) FROM saves WHERE saves.IDU=${pool.escape(IDU)} AND saves.IDP=posts.IDP) as saved*

*`;*

*let saveSearchInnerJoin='';*

*let saveSearchCondition='';*

*if(saves && IDU) {*

*saveSearchInnerJoin=`INNER JOIN saves ON posts.IDP=saves.IDP`;*

*saveSearchCondition=`WHERE saves.IDU=${pool.escape(IDU)}`*

*}*

*let querySort;*

*switch(sort){*

*case 'SORT\_BY\_DATE':*

*querySort='date';*

*break;*

*case 'SORT\_BY\_REIT':*

*querySort='likes';*

*break;*

*default: querySort='date';*

*}*

*const query = pool.query(*

*`SELECT*

*posts.IDP AS IDP,*

*posts.header AS header,*

*posts.text AS text,*

*posts.datetime as date,*

*users.login as author,*

*category.name AS category,*

*(SELECT COUNT(\*) FROM saves WHERE saves.IDP=posts.IDP) as saves,*

*(SELECT COUNT(\*) FROM likes WHERE likes.IDP=posts.IDP) as likes,*

*(SELECT COUNT(\*) FROM comments WHERE comments.IDP=posts.IDP) as comments\_l*

*${searchLikesAndSaves}*

*FROM posts*

*INNER JOIN users ON posts.IDU = users.IDU*

*INNER JOIN category ON posts.IDC = category.IDC*

*${saveSearchInnerJoin}*

*${saveSearchCondition}*

*ORDER BY ${querySort} DESC`,*

*(err, res) => {*

*if (err) {*

*console.log('loadPosts-');*

*reject(err);*

*} else {*

*const newM= res.map((e)=>{*

*const pD = new Date(e.date);*

*console.log(e.date,pD);*

*const newDate = (`${pD.getFullYear()}-${pD.getMonth()+1}-${pD.getMonth()}-${pD.getHours()}:${pD.getMinutes()}:${pD.getSeconds()}`);*

*return {...e,date:newDate};*

*});*

*console.log('parseDate+');*

*console.log(`loadPosts+ (${res.length})`);*

*resolve({...data, posts: newM})*

*}*

*});*

*});*

*};*

Функции добавления постов и комментирования, что регулируется переменной хранилища, которая имея логическое значение, определяет авторизован ли пользователь. Для неавторизованного пользователя появляются предупреждения о необходимости.

Для административной версии сайта добавлены специальные функциональности, доступные только администратору, это регулируется специальной переменной хранилища приложения, которая устанавливается в положение правды, при входе в систему администратора. Все иконки дополнительных возможностей находятся в невидимом состоянии, пока данная переменная не будет иметь значение правды.

При входе администратора возле каждого поста и комментария будет отображён крестик который не будет виден при входе обычного пользователя. Крестик несёт под собой функциональную возможность удалить соответствующий пост или комментарий.

# Руководство пользователя

Ниже приведен алгоритм действий пользователя при работе с готовым продуктом.

При входе на сайт пользователь попадает на домашнюю страницу (см. рисунок 4.1).

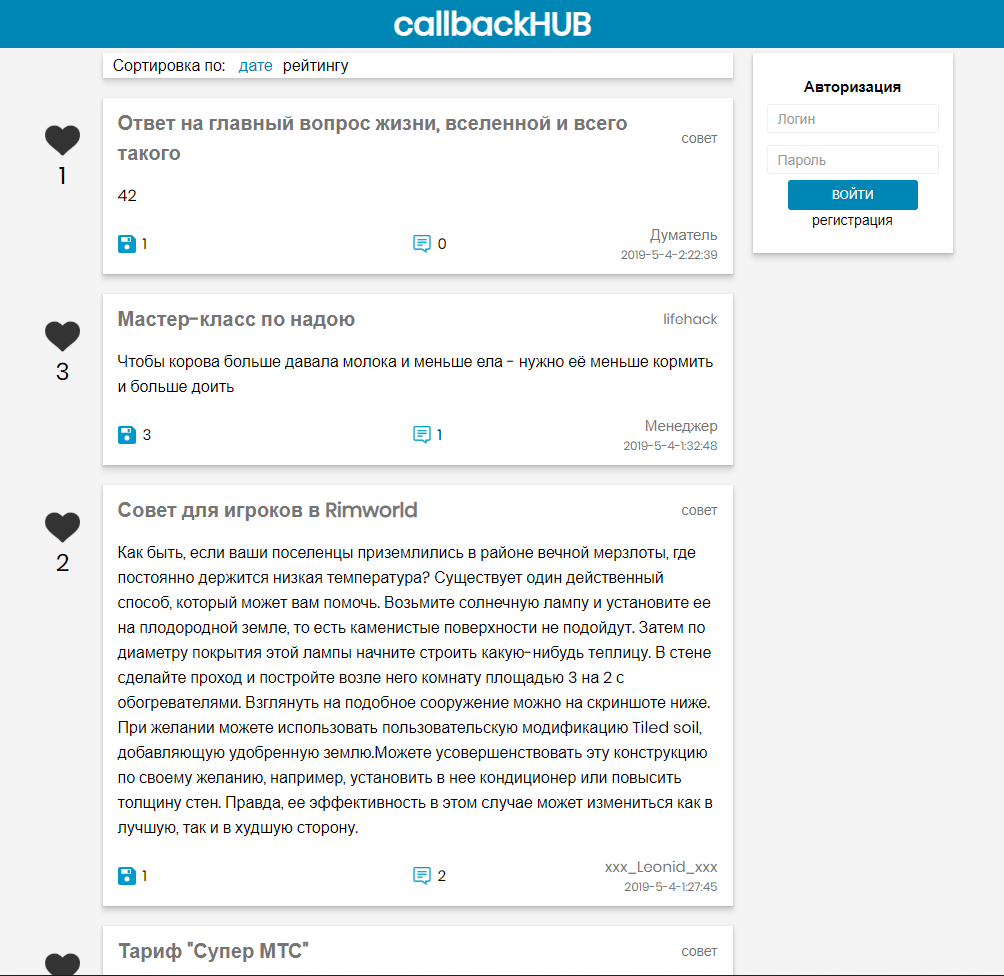


Рисунок 4.1 – Домашняя страница сайта

В верхней панели пользователю видно название сайта, форма авторизации, где можно перейти на форму регистрации, и видна лента постов. Если пользователь авторизован то форма сменится на поле с логином и кнопкой выхода (см. рисунок 4.2).



Рисунок 4.2 – Изображение вместо кнопок регистрации и войти

Для того что бы оценить пост нужно нажать на сердецко. При нажатии оно сменить цвет на розово-красный и оценка увеличится на 1. При повторном нажатии цвет снова сменится на нейтральный и оценка уменьшится на 1(см. рисунок 4.3).

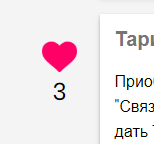


Рисунок 4.3 – Активированное сердечко

Для того что бы сохранить пост нужно нажать на иконку с дискетой. При нажатии оно также сменит цвет на розово-красный и число, которое означает количество людей сохранивших пост, увеличится на 1. При повторном нажатии цвет также снова сменится на нейтральный и число уменьшится на 1 (см. рисунок 4.4).

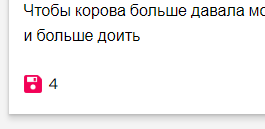


Рисунок 4.4 – Активированный значок сохранения

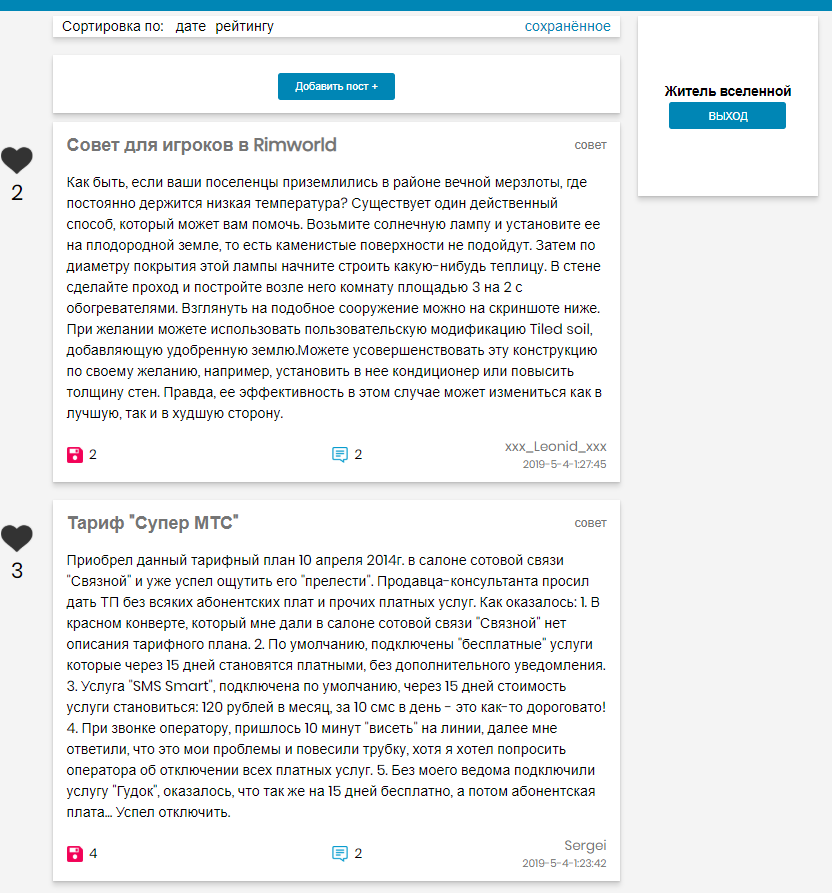
Что бы убедиться, что пост сохранился, можно в верхней панели сортировки нажать на 'сохранённые’ и увидеть список постов которые вы сохранили (см. рисунок 4.5). 

Рисунок 4.5 – Окно управления содержание корзины

Отсортировать посты можно только при условии входа в систему.

Для регистрации и входа предусмотрены формы сменяющие друг на друга при нажатии на соответствующие элементы в верхней части сайта. В формах предусмотрен вывод ошибок при вводе некорректных данных:

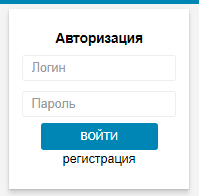


Рисунок 4.6 – Форма авторизации

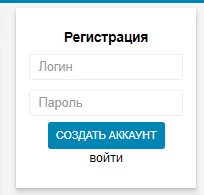


Рисунок 4.7 – Форма регистрации

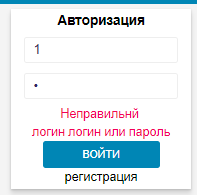


Рисунок 4.8 – Форма авторизации при попытке ввести неправильные данные

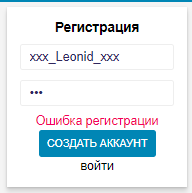


Рисунок 4.9 – Форма регистрации при попытке ввести неправильные данные

В постах предусмотрены комментарии. При загрузке поста рядом с иконкой комментариев выводится их количество. При нажатии на иконку открывается область с комментариями. При повторном нажатии область пропадает.

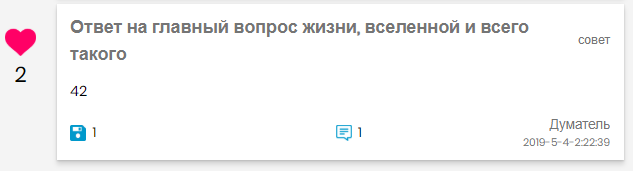


Рисунок 4.10 – Пост с закрытыми комментариями

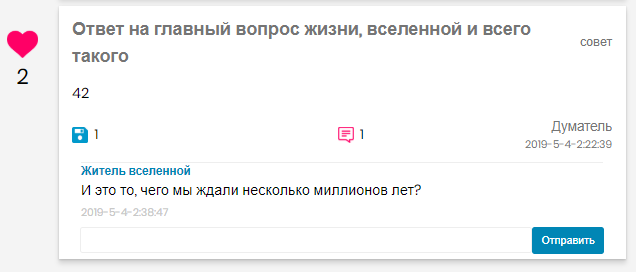


Рисунок 4.11 – Пост с открытыми комментариями

Авторизованный пользователь имеет возможность оставлять комментарии самостоятельно. Для этого необходимо открыть комментарии и воспользоваться формой в самом низу.



Рисунок 4.12 – Форма для комментариев

Также авторизованный пользователь имеет возможно самостоятельно добавлять посты открыв формы для ввода информации нажав на кнопку “Добавить пост+”

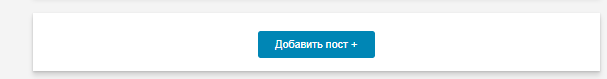


Рисунок 4.12 – Кнопка для открытия формы

После открытия формы пользователь может создать пост введя заголовок, текст поста и выбрав категорию из выпадающего списка. После нажатия на кнопку ‘Добавить’ пост будет отправлен на сервер, а клиент обновит список с новым постом.

## ЗАКЛЮЧЕНИЕ

В результате проделанной работы был создан ресурс для просмотра отзывов, советов и лайфхаков. Данный проект представляет собой пользовательский интерфейс, предназначенный для просмотра и изменения данных из базы. Соблюдены все бизнес-требования к системе: разделение пользователей по правам доступа, выдача пользователю всей информации о ресторанах и блюдах из базы данных, возможность авторизованных пользователей добавлять посты и комментировать записи, возможность для администратора удаления различных данных. Бизнес-логика создана в соответствии с требованиями, представленными выше.

Соблюдены правила создания базы данных, в следствие чего предупреждены возможности неожиданного исчезновения или редактирования данных системой.

Так же соблюдались предписания по созданию пользовательского интерфейса, интерфейс получился интуитивно понятным и лояльным.

Использовались современные технологии по созданию веб-ресурсов, что обеспечило широкое отображение ресурсов, практически без ограничений по браузерной части.

Пояснительная записка оформлена в соответствии с требованиями стандарта СТП 01-2013.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Томас Конолли. Базы данных. Проектирование, реализация и сопровождение. Теория и практик/ Томас Конолли, Карролин Бегг – Вильямс, 2003. – 1430 с.
2. К. Дж. Дейт. Введение в системы баз данных/ К. Дж. Дейт - Вильямс, 2017. – 1328 с.
3. Нормализация отношений. Шесть нормальных форм [Электронный ресурс]. – Режим доступа: https://habr.com/post/254773/, свободный.
4. Этапы проектирования данных [Электронный ресурс] – Режим доступа: <http://www.mstu.edu.ru/study/materials/zelenkov/ch_5_1.html>, свободный.
5. Итан Браун. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript/ Итан Браун – O'Reilly, 2017. -336 с.
6. [Тереза Нейл](https://oz.by/people/more9036515.html). Проектирование веб-интерфейсов/ [Тереза Нейл](https://oz.by/people/more9036515.html), [Билл Скотт](https://oz.by/people/more9036514.html) – O’Reilly, 2014/ - 352с.
7. СТП 01-2013 – Стандарт предприятия. Дипломные проекты(работы). Общие требования – Минск, БГУИР. – 174 с.

## ПРИЛОЖЕНИЕ А

**(справочное)**

**Код компонента Posts**

**import** React, {Component} **from** 'react';  
**import** {connect} **from** 'react-redux';  
**import** SortPosts **from** '../SortPosts/SortPosts'  
**import** AddPost **from** '../AddPost/AddPost'  
**import** Comments **from** '../Comments/Comments'  
**import** C **from** "../store/constants";  
**import** './Posts.css'  
  
**class** Posts **extends** Component{  
 category; liked; saves; author; likes; IDP; value; saved; show\_comments; comments\_l;  
 componentWillMount() {  
 fetch(`http://localhost:3110/?action=${C.LOAD\_POSTS}&sort=${C.SORT\_TYPES.SORT\_BY\_DATE}&login=${**this**.props.login}&password=${**this**.props.password}`)  
 .then(j=>j.json())  
 .then(posts=> {  
 **this**.props.loadPosts(posts)  
 })  
 .catch((err)=>console.error(err));  
 **this**.like=**this**.like.bind(**this**);  
 **this**.deleteLike=**this**.deleteLike.bind(**this**);  
 **this**.submitLike=**this**.submitLike.bind(**this**);  
 **this**.save=**this**.save.bind(**this**);  
 **this**.deleteSave=**this**.deleteSave.bind(**this**);  
 **this**.submitSave=**this**.submitSave.bind(**this**);  
 **this**.submitComments=**this**.submitComments.bind(**this**);  
 **this**.loadComments=**this**.loadComments.bind(**this**);  
 }  
  
 submitLike(e){  
 **const** IDP=e.target.parentElement.parentElement.value;  
 **const** post = **this**.props.posts.filter(p=>p.IDP===IDP)[0];  
 post.liked ? **this**.deleteLike(IDP) : **this**.like(IDP);  
 }  
  
 like(IDP){  
 **if**(**this**.props.login!=='' && **this**.props.password!==''){  
 fetch(`http://localhost:3110/?action=${C.LIKE}&IDP=${IDP}&login=${**this**.props.login}&password=${**this**.props.password}`)  
 .then(()=>{  
 **this**.props.like(IDP);  
 })  
 .catch((err)=>console.error(err));  
 }  
  
 };  
  
 deleteLike(IDP){  
 **if**(**this**.props.login!=='' && **this**.props.password!==''){  
 fetch(`http://localhost:3110/?action=${C.DELETE\_LIKE}&IDP=${IDP}&login=${**this**.props.login}&password=${**this**.props.password}`)  
 .then(()=>{  
 **this**.props.deleteLike(IDP);  
 })  
 .catch((err)=>console.error(err));  
 }  
 };  
  
 submitSave(e){  
 **const** IDP=e.target.parentElement.parentElement.parentElement.parentElement.value;  
 **const** post = **this**.props.posts.filter(p=>p.IDP===IDP)[0];  
 post.saved ? **this**.deleteSave(IDP) : **this**.save(IDP);  
 }  
  
 save(IDP){  
 **if**(**this**.props.login!=='' && **this**.props.password!==''){  
 fetch(`http://localhost:3110/?action=${C.SAVE}&IDP=${IDP}&login=${**this**.props.login}&password=${**this**.props.password}`)  
 .then(()=>{  
 **this**.props.save(IDP);  
 })  
 .catch((err)=>console.error(err));  
 }  
  
 };  
  
 deleteSave(IDP){  
 **if**(**this**.props.login!=='' && **this**.props.password!==''){  
 fetch(`http://localhost:3110/?action=${C.DELETE\_SAVE}&IDP=${IDP}&login=${**this**.props.login}&password=${**this**.props.password}`)  
 .then(()=>{  
 **this**.props.deleteSave(IDP);  
 })  
 .catch((err)=>console.error(err));  
 }  
 };  
  
 submitComments(e){  
 **const** IDP=e.target.parentElement.parentElement.parentElement.parentElement.value;  
 **const** post = **this**.props.posts.filter(p=>p.IDP===IDP)[0];  
 post.show\_comments ? **this**.props.hide\_comments(IDP) : **this**.loadComments(IDP);  
 }  
  
 loadComments(IDP){  
 fetch(`http://localhost:3110/?action=${C.LOAD\_COMMENTS}&IDP=${IDP}`)  
 .then(j=>j.json())  
 .then((comments)=>{  
 **this**.props.show\_comments(IDP,comments);  
 })  
 .catch((err)=>console.error(err));  
 }  
  
  
 render() {  
 **return** (  
 <section className='Posts'>  
 <SortPosts/>  
 {  
 (**this**.props.login && **this**.props.password) ?  
 <AddPost/> :  
 ''  
 }  
 <ul>  
 {**this**.props.posts.map((p) =>  
 <li key={p.IDP} className='Posts\_\_post' value={p.IDP}>  
 <div className='Posts\_\_likes'>  
 <div  
 className={p.liked ? 'Posts\_\_like-icon Posts\_\_like- icon\_liked' : 'Posts\_\_like-icon'}  
 onClick={**this**.submitLike}  
 >  
 </div>  
 {p.likes}  
 </div>  
 <section className='Posts\_\_block '>  
 <header className='Posts\_\_header'>  
 <h2 className='Posts\_\_h'>{p.header}</h2>  
 <span className='Posts\_\_category'>{p.category}</span>  
 </header>  
 <article className='Posts\_\_text'>  
 {p.text}  
 </article>  
 <footer className='Posts\_\_footer'>  
 <div className='Posts\_\_saves'>  
 <div  
 className={p.saved ? 'Posts\_\_save-icon Posts\_\_save-icon\_saved' : 'Posts\_\_save-icon'}  
 onClick={**this**.submitSave}  
 >  
 </div>  
 {p.saves}  
 </div>  
 <div className='Posts\_\_comments'>  
 <div  
 className={p.show\_comments ? 'Posts\_\_comments-icon Posts\_\_comments-icon\_show' : 'Posts\_\_comments-icon'}  
 onClick={**this**.submitComments}  
 >  
 </div>  
 {p.comments\_l}  
 </div>  
 <div className='Posts\_\_footer\_info-block'>  
 <span className='Posts\_\_author'>{p.author}</span>  
 <span className='Posts\_\_date'>{p.date}</span>  
 </div>  
 </footer>  
 {p.show\_comments ? <Comments IDP={p.IDP}/> : ''}  
 </section>  
 </li>  
 )}  
 </ul>  
 </section>  
 )  
 }  
}  
  
**export default** connect(  
 state=>({  
 posts:state.posts,  
 login:state.userPanelDate.login,  
 password:state.userPanelDate.password  
 }),  
 dispatch=>({  
 loadPosts:(posts=>{  
 dispatch({type:C.LOAD\_POSTS,posts})  
 }),  
 like:(IDP=>{  
 dispatch({type:C.LIKE,IDP})  
 }),  
 deleteLike:(IDP=>{  
 dispatch({type:C.DELETE\_LIKE,IDP})  
 }),  
 save:(IDP=>{  
 dispatch({type:C.SAVE,IDP})  
 }),  
 deleteSave:(IDP=>{  
 dispatch({type:C.DELETE\_SAVE,IDP})  
 }),  
 hide\_comments:(IDP=>{  
 dispatch({type:C.COMMENTS\_HIDDEN,IDP})  
 }),  
 show\_comments:((IDP,comments)=>{  
 dispatch({type:C.COMMENTS\_SHOWN,IDP,comments})  
 })  
 })  
)(Posts);

# ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | *Текстовые документы* |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | | *Пояснительная записка* | *61с.* | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л* | *№ докум.* | *Подп.* | *Дата* | *Онлайн-платформа для отзывов, советов и лайфхаков*  *Ведомость курсового проекта* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Ежгунович* |  |  |  | *Т* |  | *61* | *61* |
| *Провер.* | | *Трофимович* |  |  | *Кафедра ИТАС*  *гр. 620604* | | | | |
| *Т.контр.* | |  |  |  |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |
| *Реценз.* | |  |  |  |