##### 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](#_Toc6319421)

[1 аНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc6319422)

[1.1 Описание традиционной технологии 7](#_Toc6319423)

[1.2 Выявление достоинств и недостатков 9](#_Toc6319424)

[1.3 Постановка задачи 11](#_Toc6319425)

[2 Проектирование системы 12](#_Toc6319426)

[2.1 Проектирование бизнес логики 12](#_Toc6319427)

[2.2 Проектирование базы данных 14](#_Toc6319428)

[2.3 Проектирование пользовательского интерфейса 16](#_Toc6319429)

[3 Описание процесса реализации 17](#_Toc6319430)

[3.1 Выбор СУБД 17](#_Toc6319431)

[3.2 Выбор языка программирования 21](#_Toc6319432)

[3.3 Реализация ресурса 24](#_Toc6319433)

[4 Руководство пользователя 50](#_Toc6319434)

[ЗАКЛЮЧЕНИЕ 57](#_Toc6319435)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 58](#_Toc6319436)

[ПРИЛОЖЕНИЕ А 59](#_Toc6319437)

[ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА 61](#_Toc6319438)

# 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

В наши времена уже всем известно, что для того, чтобы заказать себе какое-либо блюдо из любого заведения не обязательно даже звонить, а достаточно лишь сделать несколько кликов на специализированном сайте в интернете. Главный вопрос, все-таки, что же это за специализированные сайты.

Сейчас практически у любого заведения по приготовлению пищи (пиццы, суши, воги и т.д.) есть свои сайты. Сайты таких заведений как раз и являются наиболее популярными для осуществления заказа какой-либо пищи на себя или на компанию. К примеру популярная пиццерия “Пицца Лисицца” на своём сайте предоставляет пользователям всю нужную информацию, весь каталог своих блюд, информацию о цене и доставке, их акциям и скидкам (см. рисунок 1.1).

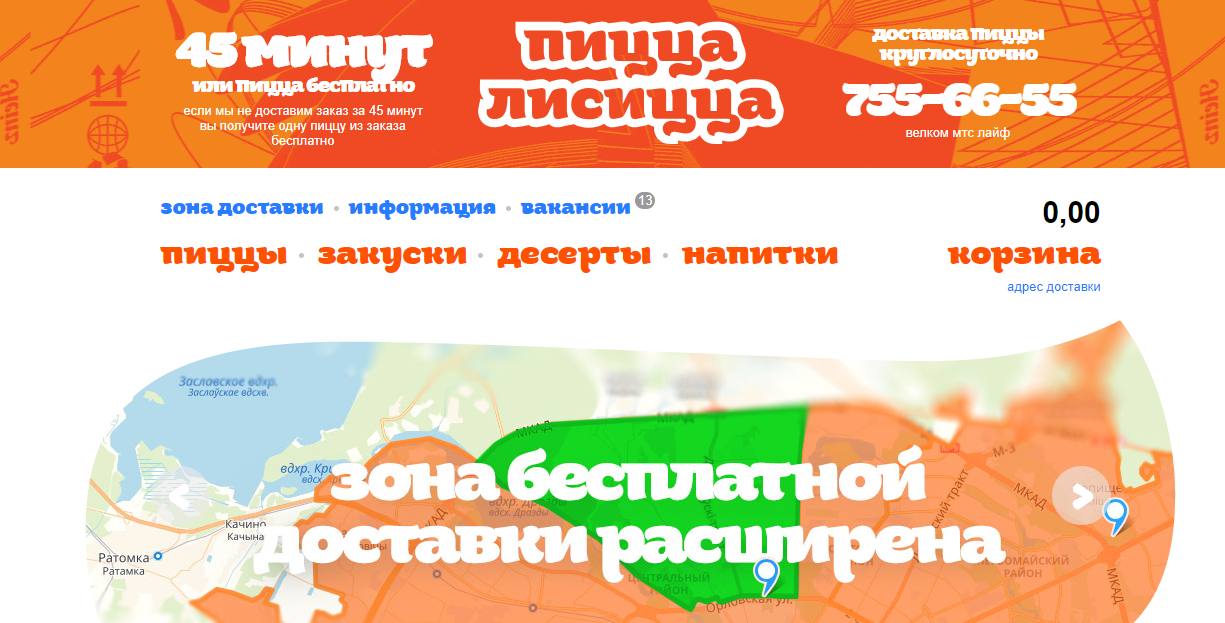


Рисунок 1.1 – Сайт пиццерии “Пицца лисицца”

Аналогичный сайт у ресторана “Суши вёсла” со всей необходимой для пользователя информацией. На сайте присутствуют также вакансии, адреса ресторанов, т.к. в отличии от “Пиццы Лисиццы” ресторан “Суши Вёсла” работает не только на вынос (см. рисунок 1.2).



Рисунок 1.2 – Сайт ресторана “Суши Вёсла”

Но что, если кушать хочется, а решить, что именно из-за большого выбора ресторанов, кафе и их сайтов кажется невозможным. Ведь у каждого из них свои блюда, цены, скидки и акции. Пользователю хочется всё же сделать анализ, решить где интереснее предложение. В общем хотелось бы видеть на одном сайте предложения сразу от всех ресторанов и кафе.

В Беларуси такой сервис уже есть и называется он “Menu.by”. Сайт компании как раз и предоставляет пользователям выбор любого блюда из любого заведения на одном сайте (см. рисунок 1.3).

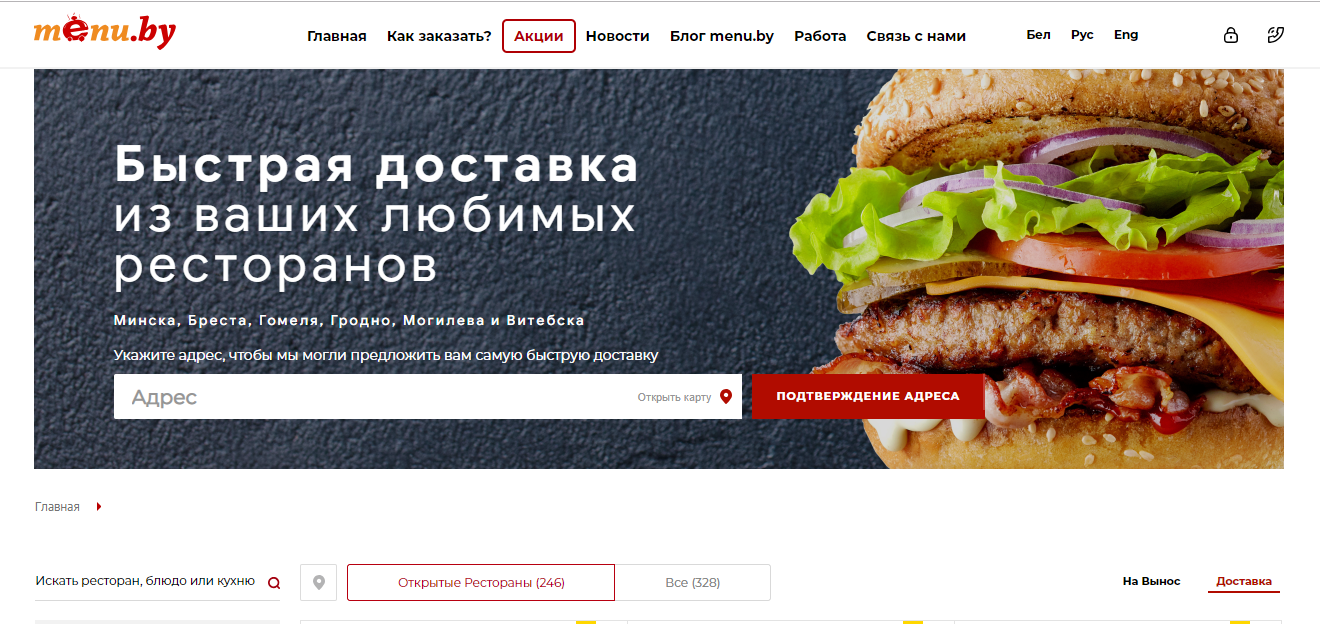


Рисунок 1.3 – Главная страница сайта “menu.by”

Самое интересное что ресторанов на сайте больше 325, и наверняка есть те о которых пользователь даже не слышал, но возможно зайдёт и попробует что-то новое. Сайт также позволяет делать удобные сортировки по различным видам блюд и кухонь для того, чтобы не показывать, то, что не интересно и сосредоточиться только на важном для пользователя.

## Выявление достоинств и недостатков

Всё же каким удобным сервисом не был бы сайт со всеми своими достоинствами, у него наверняка есть недостатки и их выявление очень важно, ведь из-за них можно потерять не мало пользователей.

Для этой цели будем рассматривать сайт “menu.by”, так как он максимально похож на курсовой проект именно тем, что объединяет в себе сразу несколько ресторанов с их предложениями.

Так как курсовой проект не имеет всей функциональности, которую имеет сервис “menu.by”, мы будем рассматривать именно то что у них общее.

Список ресторанов на данном сайте расположен в виде галереи, но также можно переключиться и на режим в виде списка (см. рисунок 1.4). С левой стороны на сайте находится панель с фильтрами (см. рисунок 1.5). На верхней навигации сайта находятся новости, и специальные списки – информация по работе, для тех, кто хочет присоединиться к команде и пункт “как заказать” для той части пользователей у кого возникли трудности со входом.

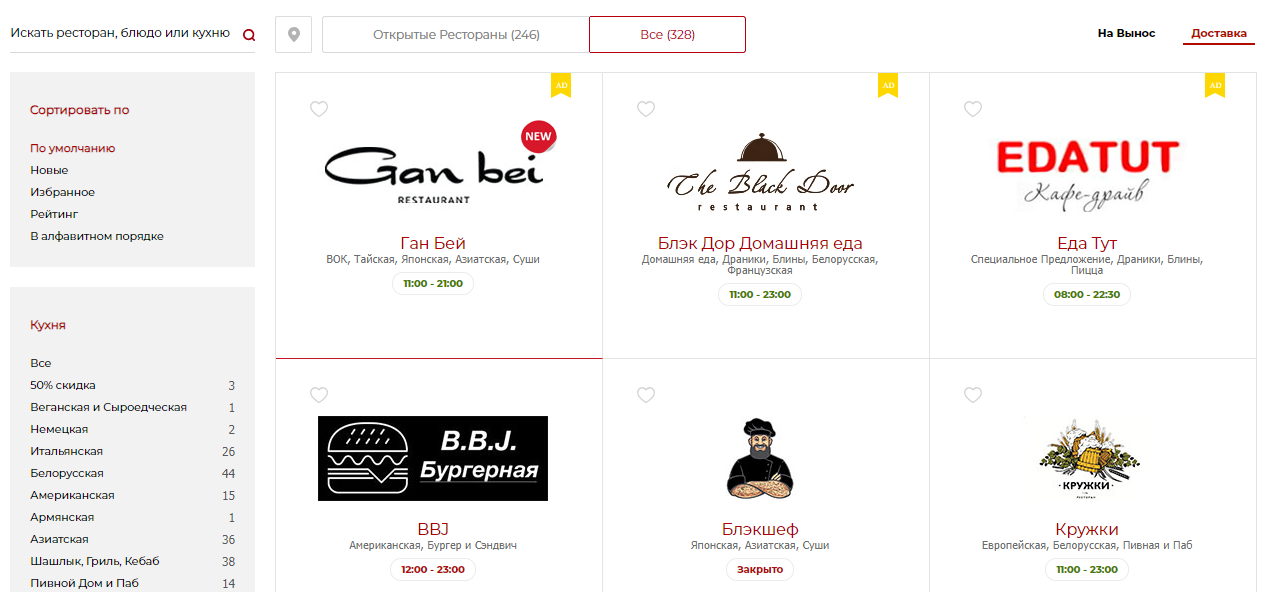


Рисунок 1.4 – Список ресторанов



Рисунок 1.5 – Фильтры



Рисунок 1.6 – Верхняя панель сайта

Достоинства сервиса: адаптивная вёрстка, на сайте осуществлён поиск, что позволяет молниеносно искать блюда и рестораны; возможность менять отображение ресторанов (список, галерея); возможность добавлять в закладки для авторизованных пользователей.

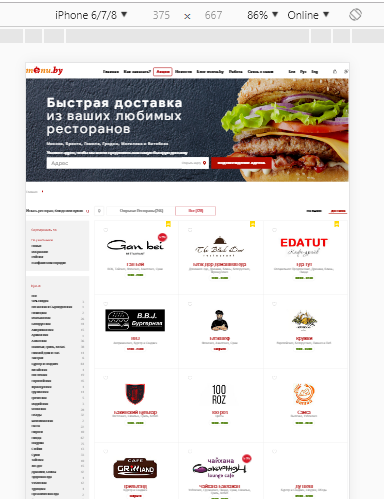


Рисунок 1.7 – Вид сайта на iPhone 6

## Постановка задачи

Курсовой проект преследует задачу создания сервиса, содержащего список ресторанов и страницы с блюдами из них. На домашней странице должны присутствовать: список ресторанов, ссылки на формы регистрации и входа в систему. Пользователь-администратор может удалять, редактировать и добавлять информацию о ресторанах и блюдах.

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 исходят 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.

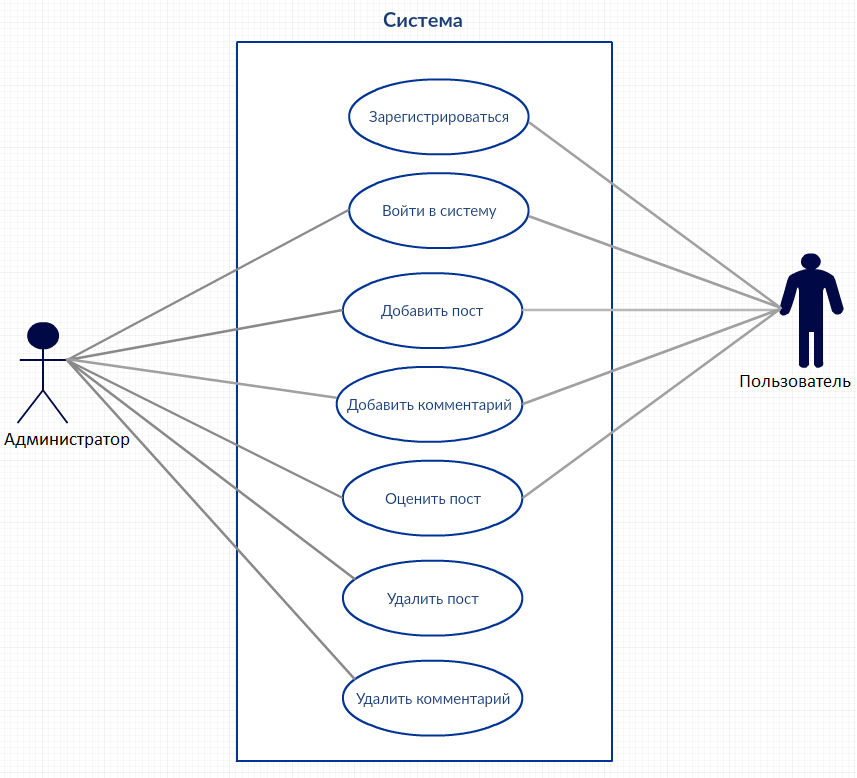


Рисунок 2.1 – Диаграмма вариантов использования

## 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.

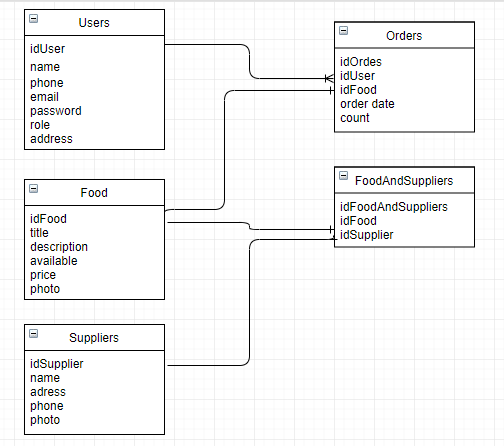


Рисунок 2.2 – Схема базы данных

## 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).

В данной работе клиент является «Тонким».

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

Создаётся база данных в визуальном интерфейсе pgAdmin PostgreSQL tools. 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 “routes.js”. Так же в папке “src” лежат “action” и “reducers” которые тоже необходимы для технологии Redux. В папке “components” лежат компоненты которые соответствуют философии фреймворка React (см. рисунок 3.2). В серверной части находятся папки и модули, содержащие в себе реализацию бизнес-логики (контроллеры) и так же маршрутизаторы (роутеры), модели(models), файл с настройками подключения к базе данных (см. рисунок 3.4).

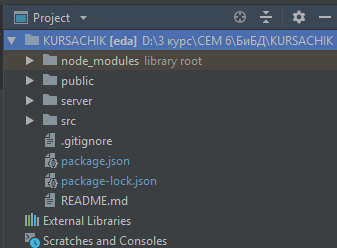


Рисунок 3.1 – Основная иерархия папок

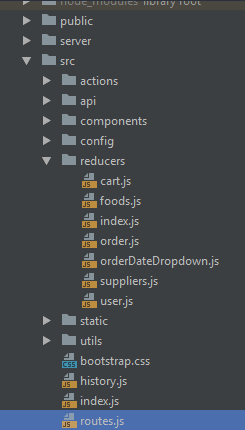


Рисунок 3.2 – Файловая иерархия папки клиент

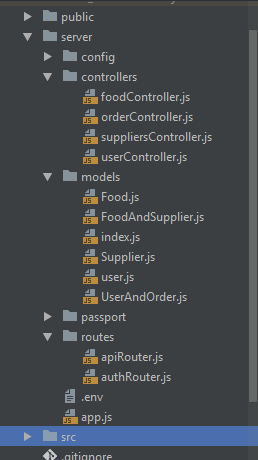


Рисунок 3.3 – Иерархия папки сервер

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

*const Routes = () => (  
 <****Provider*** *store={store}> //важный компонент для соединения  
 <****Router*** *history={history}> // хранилищем  
 <****App****>  
 <****Header*** */> // объяснение этих компонентов будет   
 <****Main*** */> // далее  
 <****Footer*** */>  
 </****App****>  
 </****Router****>  
 </****Provider****>  
);  
  
export default Routes;*

Где “header” представляет собой верхнюю панель сайта. Код для реализации хедера представлен ниже:

*class Header extends React.Component {  
  
 render() {  
 let numberOfItemsInCart = this.props.items.length > 0 ? this.props.items.length : '';  
 return (  
 <****header*** *className="header">  
 <****Link*** *exact={true} to='/' className="header\_\_logo" >****Restaurants****</****Link****> //компонент ссылки перехода на главную  
 <****div*** *className="header-right">  
 <****button*** *onClick={() => {this.props.dispatch(toggleCart())}} className="header\_\_btn-cart">  
 <****span*** *className='header\_\_num-of-items'> {numberOfItemsInCart}</****span****>  
 </****button****>  
 { this.props.user.isAuthenticated ? // проверка на  
 <****button*** *onClick={() => //авторизованного {this.props.dispatch(toggleUserModal())}} //пользователя className="header\_\_btn-user"></****button****> //в соотвествии с этим   
 : <****div****> // отрисовываюся кнопки различного вида  
 <****Link*** *to='/signin' className="btn btn-secondary my-2 my-sm-0">****Sign in****</****Link****>  
 <****Link*** *to='/signup' className="btn btn-secondary my-2 my-sm-0">****Sign up****</****Link****>  
 </****div****>  
 }  
 </****div****>  
 </****header****>  
 );  
 }  
}  
  
function mapStateToProps (state) {  
 return {  
 items: state.cart.items, // соединение с хранилищем  
 user: state.user // которое предоставляет Redux  
 }  
}  
  
export default connect(mapStateToProps)(Header)*

“main” - это, собственно, и есть главная часть которая предоставляет список ресторанов, блюд. Эта часть перерендывается когда нужно перевести пользователя на другую страницу, к примеру для подтверждения заказа. Код для этого компонента представлен ниже:

*const Main = () => {  
 return (  
 <****div*** *className='main'>  
 <****Switch****>  
 <****Route*** *exact='true' path='/' component={SuppliersList} />  
 <****Route*** *path='/signin' component={LoginForm} />  
 <****Route*** *path='/signup' component={RegisterForm} />  
 <****Route*** *path='/userInfo' component={UserInformation}/>  
 <****Route*** *path='/userOrders' component={UserOrders}/>  
 <****Route*** *path='/orderConfirmation' component={OrderConfirmation}/>  
 <****Route*** *path='/userProfiles' component={UserProfiles}/>  
 <****Route*** *path='/suppliersEditing' component={SuppliersEditingList}/>  
 <****Route*** *path={`/suppliers/:uuid/edit`} component={FoodEditingList} />  
 <****Route*** *path={`/suppliers/:uuid/food`} component={FoodList} />  
 </****Switch****>  
 <****Cart****/>  
 <****UserModal****/>  
 </****div****>  
 );  
};  
  
export default Main;*

Компонент “footer” отвечает за нижнюю панель сайта. Собственно этот компонент не несёт за собой никакой функциональности. Его код этого компонента является не сложным и несёт в себе практически обычную разметку.

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.

Маршрутизация сервиса, где прописаны конечные точки маршрутов и их обработчики, представлена в приложении А.

Навигация 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 “userAction”, который находится в папке “action”. Как говорилось ранее эта папка нужна для управления данными которые лежат в хранилище Redux. Здесь же осуществляется и управление локальными хранилищем браузера:

*export function setCurrentUser(user) { // редаксовские редьюсеры  
 return { // только они и могут изменять хранилище   
 type: SET\_CURRENT\_USER, // которое предоставляет Redux  
 payload: user  
 }  
  
}  
  
export function registerUser(userData) {  
 return dispatch => {  
 dispatch({  
 type: REGISTER\_REQUEST // функии проверяют тип  
 }); // и в соответствии с типом выполняются   
 axios.post(`http://localhost:3000/signup/`, userData)  
 .then(res => {  
 if (res) {  
 dispatch({  
 type: REGISTER\_SUCCESS,  
 });  
 const token = res.data.token;  
 localStorage.setItem('token', token);  
 setAuthorizationToken(token);  
 dispatch(setCurrentUser(jwt.decode(token)));  
 history.goBack();  
 }  
 })  
 .catch(() => {  
 dispatch({  
 type: REGISTER\_FAILURE,  
 payload: 'User with that email already exists!'  
 });  
 })  
 }  
}  
export function loginUser(userData) {  
 return dispatch => {  
 dispatch({  
 type: LOGIN\_REQUEST,  
 });  
 axios.post(`http://localhost:3000/signin/`, userData)  
 .then(res => {  
 if (res) {  
 const token = res.data.token;  
 localStorage.setItem('token', token);  
 setAuthorizationToken(token);  
 dispatch(setCurrentUser(jwt.decode(token)));  
 dispatch({  
 type: LOGIN\_SUCCESS,  
 });  
 history.goBack();  
 }  
  
 })  
 .catch(()=> {  
 dispatch({  
 type: LOGIN\_FAILURE,  
 payload: 'Incorrect username or password!'  
 });  
 })  
 }  
}  
export function logoutUser() {  
 return dispatch => {  
 localStorage.removeItem('token'); // соединения с хранилищем, и здесь же вызов  
 setAuthorizationToken(false); // соответствующих редбюсеров  
 dispatch(setCurrentUser({}));  
 dispatch(clearOrders());  
 history.push('/')  
 }  
}  
export function toggleUserModal(payload) {  
 return {  
 type: TOGGLE\_USER\_MODAL,  
 payload: payload  
 }  
}  
export function getUserProfiles() {  
 return dispatch => {  
 axios.get(`http://localhost:3000/api/admin/userProfiles`)  
 .then(res => {  
 dispatch({  
 type: GET\_ALL\_USERS,  
 payload: res.data,  
 });  
 });  
 }  
}  
export function updateUser(userData) {  
 return dispatch => {  
 axios.put(`http://localhost:3000/api/admin/userProfiles`, userData)  
 .then(res => {  
 dispatch({  
 type: GET\_ALL\_USERS,  
 payload: res.data,  
 });  
 });  
 }  
}*

Реализация функции экспорта данных из БД на серверной стороне:

const fs = *require*('fs'); // модуль для работы с файлами  
const path = *require*('path'); // модуль для определения пути для какого либо файла  
const Sequelize = *require*('sequelize'); // модулья для объектного отображения бд в js  
const basename = path.basename(*\_\_filename*);  
const env = *process*.env.NODE\_ENV || 'development';  
const config = *require*(*\_\_dirname* + '/../config/config.json')[env];  
const db = {};  
  
if (config.use\_env\_variable) {  
 var sequelize = new Sequelize(*process*.env[config.use\_env\_variable], config);  
} else {  
 var sequelize = new Sequelize(config.database, config.username, config.password, config);  
}  
sequelize  
 .authenticate() // модуль sequelize работает как Promis, что даёт большое удобство при  
 .then(() => { // цепочке функций и отлавливания ошибок  
 *console*.log('Connection has been established successfully.'); //происходит подключения к   
 }) // БД  
 .catch(*err* => {  
 *console*.error('Unable to connect to the database:', *err*);  
 });  
fs  
 .readdirSync(*\_\_dirname*)  
 .filter(*file* => {  
 return (*file*.indexOf('.') !== 0) && (*file* !== basename) && (*file*.slice(-3) === '.js');  
 })  
 .forEach(*file* => {  
 const model = sequelize['import'](path.join(*\_\_dirname*, *file*));  
 db[model.name] = model;  
 });  
  
*Object*.keys(db).forEach(*modelName* => {  
 if (db[*modelName*].associate) {  
 db[*modelName*].associate(db);  
 }  
});  
  
db.sequelize = sequelize;  
  
*module*.exports = db;

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 функции привидения к числу.

Форма регистрации и авторизации абсолютно аналогичны в плане визуального интерфейса, за исключение лишь некоторых данных, т.к. для того чтобы войти в систему пользователю не обязательно набирать адрес или телефон. При регистрации и авторизации происходит проверка данных форм на стороне браузера, что не нагружает сервер. Прописаны правила проверки, соответствующие требованиям. Код ниже:

const authRouter = new express.Router();  
  
const SignUpSchema = *Joi*.object().keys({ //правила для полей регистрации  
 name: *Joi*.string().min(2).max(30),  
 phone: *Joi*.string().required(),  
 password: *Joi*.string().regex(/^[a-zA-Z0-9]{6,30}$/).required(), //регулярное выражение  
 email: *Joi*.string().email().required(),  
 address:*Joi*.string().required(),  
});  
  
const SignInSchema = *Joi*.object().keys({ //правила для полей авторизации  
 email: *Joi*.string().email().required(),  
 password: *Joi*.string().regex(/^[a-zA-Z0-9]{6,30}$/).required()  
});  
  
authRouter.post('/signup', celebrate({ // отправка запроса регистрацию  
 body: SignUpSchema  
 }), (*req*, *res*, *next*) => passport.authenticate('localSignUp', (*err*,*user*) => {  
 if (*err*) {  
 return *res*.sendStatus(400)  
 }  
 let token = jwt.sign({ // формирования токена  
 uuid: *user*.uuid, // уникальный идентификатор   
 name: *user*.name, // PostgreSql создаёт его сама  
 email: *user*.email,  
 phone: *user*.phone,  
 address: *user*.address,  
 role: *user*.role,  
 }, config.jwtSecret);  
 return *res*.status(200).json({  
 success: true,  
 token  
 });  
 })(*req*, *res*, *next*)  
);  
  
authRouter.post('/signin', celebrate({ // отправка запроса авторизацию  
 body: SignInSchema  
 }), (*req*, *res*, *next*) => passport.authenticate('localSignIn', (*err*, *user*) => {  
 if (*err*) {  
 return *res*.sendStatus(400)  
 }  
 let token = jwt.sign({  
 uuid: *user*.uuid,  
 name: *user*.name,  
 email: *user*.email,  
 phone: *user*.phone,  
 address: *user*.address,  
 role: *user*.role,  
 }, config.jwtSecret);  
 return *res*.status(200).json({  
 success: true,  
 token  
 });  
 })(*req*, *res*, *next*)  
);

Ниже приведен код контейнера формы регистрации (код формы авторизации почти аналогичен), где описываются методы запроса к серверу, а также перехода в случае успеха на новую страницу:

render() {  
 return (  
 <**form** onSubmit={(*e*) => this.registerUser(*e*)} className="register-form" >  
 <**p** className={this.state.errorMessage ? "error-message error" : "error"}>{this.state.errorMessage}</**p**>  
 <**fieldset**>  
 <**legend**>**Sign Up**</**legend**>  
 <**div** className="form-group">  
 <**label** className="col-form-label">**Name**</**label**>  
 <**input** type="text" className="form-control" name="username" onChange={(*e*)=>{this.inputHandler(*e*)}} required/> // на каждом поле ввода стоят   
 </**div**> // обработчики   
 <**div** className="form-group">  
 <**label** className="col-form-label">**Phone**</**label**>  
 <**input** type="text" className="form-control" name="phone" onChange={(*e*)=>{this.inputHandler(*e*)}} required/>  
 </**div**>  
 <**div** className="form-group">  
 <**label** >**Address**</**label**>  
 <**input** type="text" className="form-control" name="address" onChange={(*e*)=>{this.inputHandler(*e*)}} required/>  
 </**div**>  
 <**div** className="form-group">  
 <**label** >**Email address**</**label**>  
 <**input** type="email" className="form-control" aria-describedby="emailHelp" name="email" onChange={(*e*)=>{this.inputHandler(*e*)}} required/>  
 <**small** className="form-text text-muted">**We'll never share your email with anyone else.**</**small**>  
 </**div**>  
 <**div** className="form-group">  
 <**label** >**Password**</**label**>  
 <**input** type="password" className="form-control" id="user-password" name="password" onChange={(*e*)=>{this.inputHandler(*e*)}} required/>  
 </**div**>  
 <**div** className="form-group">  
 <**label** >**Password confirmation**</**label**>  
 <**input** type="password" className="form-control" name="passwordConfirmation" onChange={(*e*)=>{this.inputHandler(*e*)}} required/>  
 </**div**>  
 <**button** type="submit" className="btn btn-primary">**Submit**</**button**> //при  
 </**fieldset**> // нажатиии на кнопку данные из инпутов отправляются на проверку  
 </**form**>  
 );  
 }  
}  
function mapStateToProps (*state*) { // соединение с хранилищем Redux  
 return {  
 user: *state*.user  
 }  
}  
export default connect(mapStateToProps)(RegisterForm);

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:

export default new PassportLocalStrategy({  
 usernameField: 'email',  
 passwordField: 'password',  
 session: false,  
 passReqToCallback: true  
}, (*req*, *email*, *password*, *done*) => {  
 const { name, phone, address } = *req*.body;  
 const salt = bcrypt.genSaltSync(10);  
 const userData = {  
 name: name.trim(),  
 phone: phone.trim(), // собираются данные  
 email: *email*.trim(),  
 password: bcrypt.hashSync(*password*.trim(), salt),  
 address: address.trim(),  
 };  
 db.User  
 .findOne({  
 where: {  
 email: userData.email  
 }  
 })  
 .then(*user* => {  
 if (!*user*) { // происходит проверка на существование пользователя  
 return db.User // при отсутствии он будет создан, при существовании показано  
 .create(userData) //сообщение  
 .then(*user* => *done*(null,*user*.dataValues))  
 .catch(*error* => *done*(*error*));  
 }  
  
 return *done*({  
 error: 'That email is already taken.'  
 })  
 })  
 .catch(*error* => *done*(*error*));  
});

Страница со списком ресторанов 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:

componentDidMount() { // специальный жизненный цикл в React для подтягивания данных  
 this.props.dispatch(getSuppliersList())  
}

Таким образом при обновлении страницы список так же обновляется в соответствии с актуальной информацией из базы. Её визуальная составляющая состоит из списка ресторанов с динамической подгрузкой. Список при прокрутке страницы вниз или вверх элементы динамически удаляются или добавляются компонент списка ресторанов:

render() {  
 let {suppliers} = this.props;  
 let suppliersContainer = suppliers.map((*supplier*) => // перебирается массив  
 <**SupplierItem** name={*supplier*.name} // и в каждый новый компонент будут вставлены новые  
 address={*supplier*.address} // свойства  
 phone={*supplier*.phone}  
 photo={*supplier*.photo}  
 uuid={*supplier*.uuid}  
 key={*supplier*.uuid}  
 />  
 );  
 return (  
 <**div**>  
 <**span** className='blocks-title'>**Choose your supplier**</**span**>  
 <**div** className='blocks'>  
 {suppliersContainer} // но отрисовывается весь массив из компонентов  
 </**div**> // что очень удобно в работе  
 </**div**>  
 );  
 }  
}  
  
function mapStateToProps(*state*) {  
 return {  
 suppliers: *state*.supplier.suppliers,  
 }  
}  
  
export default connect(mapStateToProps)(SuppliersList);

В каждом элементе списка содержится информация о названии ресторана, его изображение, адрес, телефон, и так же сам элемент списка является кликабельной ссылкой при нажатии на которую, пользователю будет предоставлен список блюд, которые есть в этом ресторане. Код для реализации списка блюд:

componentDidMount() {  
 this.props.dispatch(getSuppliersFoods(this.props.match.params.uuid));  
 }  
  
 render() {  
 let {food, foodItems} = this.props;  
 let foodItem = !food.isLoaded  
 ? <**PulseLoader** color={'#252525'}  
 loading={!food.isLoaded}  
 /> : foodItems.length > 0 ?  
 foodItems.map((*food*) =>  
 <**FoodItem** title={*food*.title}  
 description={*food*.description}  
 available={*food*.available}  
 price={*food*.price}  
 photo={*food*.photo}  
 uuid={*food*.uuid}  
 key={*food*.uuid}  
 />)  
 : <**h5** className="card-title">**Sorry, this supplier haven't any food yet**</**h5**>;  
  
 return (  
 <**div** className={food.isLoaded ? "blocks" : "spinner"}>  
 {foodItem}  
 </**div**>  
 );  
 }  
}  
function mapStateToProps (*state*) {  
 return {  
 foodItems: *state*.foods.food,  
 food: *state*.foods  
 }  
}  
  
export default connect(mapStateToProps)(FoodList)

Страница с подробной информацией о пользователе. Визуальная составляющая представлена именем пользователя, его email-ом, телефоном, адресом. Код компонента с пользовательской информацией:

render() {  
  
 return (  
 <**div** className="jumbotron user-info">  
 {this.props.user.isAuthenticated  
 ? <**div**>  
 <**h1** className="display-6 user-info-header">**Your information**</**h1**>  
 <**table** className="table table-hover user-info-table">  
 <**thead**>  
 <**tr**>  
 <**th** scope="col">**Name**</**th**> // статические поля  
 <**th** scope="col">**Email**</**th**>  
 <**th** scope="col">**Phone**</**th**>  
 <**th** scope="col">**Address**</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr**>  
 <**td**>{this.props.user.info.name}</**td**> // динамически поля, данные  
 <**td**>{this.props.user.info.email}</**td**> // для них подягиваются из   
 <**td**>{this.props.user.info.phone}</**td**> // хранилища Redux  
 <**td**>{this.props.user.info.address}</**td**>  
 </**tr**>  
 </**tbody**>  
 </**table**>  
 </**div**>  
 : <**h3** className="display-6 user-info-header">**Sorry, you must sign in first**</**h3**>}  
 </**div**>  
 );  
 }  
}  
function mapStateToProps (*state*) {  
 return {  
 user: *state*.user,  
 }  
}  
  
export default connect(mapStateToProps)(UserInformation)

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

render() {  
 const { isVisible} = this.props.orderDateDropdown;  
 const {orders} = this.props;  
 const filteredOrders = orders.filter((*order*) => {  
 if (this.props.user.info.role === 'admin') {  
 return *order*.orderDate.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.Food.title.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.User.name.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.User.phone.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.User.email.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.User.address.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.count.toString().includes(this.state.searchValue)  
 } else {  
 return *order*.orderDate.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.Food.title.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.Food.description.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 *order*.count.toString().includes(this.state.searchValue)  
 }  
 });   
 const orderItemsContainer = orders.length > 0  
 ? filteredOrders.map((*order*) => {  
 if (this.props.user.info.role === 'admin') {   
 return <**UserOrderItems** orderDate={*order*.orderDate} // проверка на админа и отрисовка компонента  
 title={*order*.Food.title} // с соответствующими свойствами  
 price={*order*.Food.price}  
 photo={*order*.Food.photo}  
 count={*order*.count}  
 username={*order*.User.name}  
 userPhone={*order*.User.phone}  
 userEmail={*order*.User.email}  
 userAddress={*order*.User.address}  
 key={*order*.uuid}  
 />  
 }  
 else {  
 return <**UserOrderItems** orderDate={*order*.orderDate}  
 title={*order*.Food.title}  
 description={*order*.Food.description}  
 price={*order*.Food.price}  
 photo={*order*.Food.photo}  
 count={*order*.count}  
 key={*order*.uuid}  
 />  
 }})  
 : <**th** className="card-title">**Sorry, you haven't any order**</**th**>;  
  
 return (  
 <**div** className="jumbotron user-info">  
 <**button** type="button" onClick={() => {  
 history.goBack();  
 }} className="close btn-delete btn-go-back" aria-label="Go back">  
 <**span** aria-hidden="true">&larr;</**span**>  
 </**button**>  
 {!this.props.user.isAuthenticated ?  
 <**h3** className="display-6 user-info-header">**Sorry, you must sign in first**</**h3**>  
 : this.props.user.info.role === 'admin' ?  
 <**div**>  
 <**div** className="user-orders\_\_header">  
 <**h1** className="display-6 user-info-header for-print">**Users orders**</**h1**>  
 <**form**>  
 <**input** onChange={(*e*) => this.onInputChange(*e*)} className="form-control form-control-sm mr-sm-2 search-input" type="search" placeholder="Search for an order..."/>  
 <**button** type="button" className="btn btn-outline-primary btn-sm btn-admin"  
 onClick={() => {*window*.print()}}>**Print orders** </**button**>  
 </**form**>  
  
 </**div**>  
 <**table** className="table table-hover user-info-table for-print">  
 <**thead**>  
 <**tr**>  
 <**th** scope="col">**Item**</**th**>  
 <**th** scope="col"></**th**>  
 <**th** scope="col" className="nav-item dropdown show">  
 <**a** className="nav-link dropdown-toggle" onClick = {() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 }}  
 data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="true">**Order date**</**a**>  
 <**div** className="dropdown-menu show" style={isVisible ? {display: 'block'} : {display: 'none'}}>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getAllOrders());  
 }}>**All**</**a**>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getAllOrdersForToday());  
 }}>**Today**</**a**>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getAllOrdersForWeek());  
 }}>**Last week**</**a**>  
 </**div**>  
 </**th**>  
 <**th** scope="col">**Quantity**</**th**>  
 <**th** scope="col">**User name**</**th**>  
 <**th** scope="col">**User phone**</**th**>  
 <**th** scope="col">**User address**</**th**>  
 <**th** scope="col">**Total price**</**th**>  
 </**tr**>  
 </**thead**>  
 {orderItemsContainer}  
 </**table**>  
 </**div**>  
 : <**div**>  
 <**div** className="user-orders\_\_header">  
 <**h1** className="display-6 user-info-header">**Your orders**</**h1**>  
 <**form**>  
 <**input** onChange={(*e*) => this.onInputChange(*e*)} className="form-control form-control-sm mr-sm-2" type="search" placeholder="Search for an order"/>  
 </**form**>  
 </**div**>  
 <**table** className="table table-hover user-info-table">  
 <**thead**>  
 <**tr**>  
 <**th** scope="col">**Item**</**th**>  
 <**th** scope="col"></**th**>  
 <**th** scope="col">**Description**</**th**>  
 <**th** scope="col" className="nav-item dropdown show">  
 <**a** className="nav-link dropdown-toggle" onClick = {() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 }}  
 data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="true">**Order date**</**a**>  
 <**div** className="dropdown-menu show" style={isVisible ? {display: 'block'} : {display: 'none'}}>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getUserOrders(this.props.user.info.uuid));  
 }}>**All**</**a**>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getUserOrdersForToday(this.props.user.info.uuid));  
 }}>**Today**</**a**>  
 <**a** className="dropdown-item" onClick={() => {  
 this.props.dispatch(toggleOrderDateDropdown());  
 this.props.dispatch(getUserOrdersForWeek(this.props.user.info.uuid));  
 }}>**Last week**</**a**>  
 </**div**>  
 </**th**>  
 <**th** scope="col">**Quantity**</**th**>  
 <**th** scope="col">**Total price**</**th**>  
 </**tr**>  
 </**thead**>  
 {orderItemsContainer} // динамическое поле, которое будет готова  
 </**table**> // непосредственно перед отрисовкой  
 </**div**>  
 }  
 </**div**>  
 );  
 }  
}  
function mapStateToProps (*state*) {  
 return { // соединение с Redux  
 orders: *state*.order,  
 user: *state*.user,  
 orderDateDropdown: *state*.orderDateDropdown,  
 }  
}  
  
export default connect(mapStateToProps)(UserOrders)

Функции оформления заказов и их просмотр доступны только авторизованным пользователем, что регулируется переменной хранилища, которая имея логическое значение, определяет авторизован ли пользователь. Для неавторизованного пользователя появляются предупреждения о необходимости регистрации и система переносит его на форму регистрации.

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. Код добавленных элементов в компоненте UserModal:

*render() {  
 const { isVisible} = this.props.user;  
 return (  
 <****div*** *className=" modal user-modal" style={isVisible ? {display: 'block'} : {display: 'none'}} >  
 <****div*** *className="modal-dialog">  
 {this.props.user.isAuthenticated && this.props.user.info.role === 'admin' ?  
 <****div*** *className="modal-content">  
 <****Link*** *to='/userOrders' onClick={() => {this.props.dispatch(toggleUserModal())}}>****View user orders****</****Link****>  
 <****Link*** *to='/userProfiles' onClick={() => {this.props.dispatch(toggleUserModal())}}>****Edit user profile****</****Link****>  
 <****Link*** *to='/suppliersEditing' onClick={() => {this.props.dispatch(toggleUserModal())}}>****Edit supplier data****</****Link****>  
 <****div*** *className="modal-footer">  
 <****a*** *onClick={() => { this.props.dispatch(logoutUser()) }}>****Log out****</****a****>  
 </****div****>  
 </****div****>  
 : <****div*** *className="modal-content">  
 <****Link*** *to='/userInfo' onClick={() => {this.props.dispatch(toggleUserModal())}}>****Your information****</****Link****>  
 <****Link*** *to='/userOrders' onClick={() => {this.props.dispatch(toggleUserModal())}}>****Orders history****</****Link****>  
 <****div*** *className="modal-footer">  
 <****a*** *onClick={() => { this.props.dispatch(logoutUser()) }}>****Log out****</****a****>  
 </****div****>  
 </****div****>  
 }  
 </****div****>  
 </****div****>  
 );  
 }  
}  
function mapStateToProps (state) {  
 return {  
 user: state.user* // соединение с Redux *}  
}  
  
export default connect(mapStateToProps)(UserModal)*

В этом окне администратор может путём нажатия на кнопку “Edit supplier data”. При нажатии на неё администратор перенаправляется на страницу редактирования данных ресторанов. Код компонента отображения страницы для редактирования:

*addSupplier(e) {  
 const supplierData = {  
 name: this.refs.name.value,  
 phone: this.refs.phone.value,  
 address: this.refs.address.value,  
 photo: this.refs.photoLink.value,  
 };  
 this.props.dispatch(createSupplier(supplierData));  
 this.props.dispatch(toggleAddSupplierField());  
 this.refs.name.value = ''; // переменные для хранения введённых значений  
 this.refs.phone.value = '';  
 this.refs.address.value = '';  
 this.refs.photoLink.value = '';  
 e.preventDefault()  
 }  
 onInputChange (e) {  
 this.setState({ // функция, которая отвечает за изменения значения  
 searchValue: e.target.value // в инпуте  
 });  
 e.preventDefault();  
 };  
 render() {  
 let filteredSuppliers = this.props.supplier.suppliers.filter((supplier) => {  
 return supplier.name.toLowerCase().includes(this.state.searchValue.toLowerCase()) ||  
 supplier.phone.includes(this.state.searchValue.toLowerCase()) ||  
 supplier.address.toLowerCase().includes(this.state.searchValue.toLowerCase())  
 });  
 return (  
 <****div*** *className="jumbotron user-info">  
 <****button*** *type="button" onClick={() => {  
 history.goBack();  
 }} className="close btn-delete btn-go-back" aria-label="Go back">  
 <****span*** *aria-hidden="true">&larr;</****span****>  
 </****button****>  
 {!this.props.user.isAuthenticated ?  
 <****h3*** *className="display-6 user-info-header">****Sorry, you must sign in first****</****h3****>  
 : this.props.user.info.role === 'admin' ?  
 <****div****>  
 <****div*** *className="user-orders\_\_header">  
 <****h1*** *className="display-6 user-info-header for-print">****All suppliers****</****h1****>  
 <****form****>  
 <****input*** *onChange={(e) => this.onInputChange(e)} className="form-control form-control-sm mr-sm-2 search-input" type="search" placeholder="Search supplier"/>  
 <****button*** *type="button"  
 onClick={() => {this.props.dispatch(toggleAddSupplierField())}}  
 className="btn btn-outline-primary btn-sm btn-admin">  
 {this.props.supplier.addFieldIsVisible ? 'Cancel': 'Add supplier'}  
 </****button****>  
 </****form****>  
 </****div****>  
 <****div*** *className="user-profile">  
 <****div*** *className="user-profile\_\_label">  
 <****label*** *className="control-label" htmlFor="nameInput">****Name****</****label****>  
 <****label*** *className="control-label" htmlFor="phoneInput">****Phone****</****label****>  
 <****label*** *className="control-label" htmlFor="addressInput">****Address****</****label****>  
 <****label*** *className="control-label" htmlFor="photoLinkInput">****Photo link****</****label****>  
 </****div****>  
 <****form*** *onSubmit={(e) => {this.addSupplier(e)}} className="user-profile\_\_info add-supplier" style={!this.props.supplier.addFieldIsVisible ? {display: "none"} :{}}>  
 <****input*** *className="form-control" id="nameInput" type="text" ref="name" required />  
 <****input*** *className="form-control" id="phoneInput" minLength="7" type="number" ref="phone" />  
 <****input*** *className="form-control user-profile\_\_address" id="addressInput" type="text" ref="address" required/>  
 <****input*** *className="form-control user-profile\_\_address" minLength="10" id="photoLinkInput" type="text" ref="photoLink"/>  
 <****button*** *type="submit" className="btn btn-outline-primary btn-sm">****Save****</****button****>  
 </****form****>  
 {filteredSuppliers.map( (supplier) => {  
 return <****SupplierEditingItem*** *name={supplier.name} // перебор свойств и составление   
 phone={supplier.phone} // списка из компонентов  
 address={supplier.address}  
 photoLink = {supplier.photo}  
 uuid={supplier.uuid}  
 key={supplier.uuid}  
 />  
 }) }  
 </****div****>  
 </****div****>  
 : <****div****>  
 <****h3*** *className="display-6 user-info-header">****Your haven't permission to view this page****</****h3****>  
 </****div****>  
 }  
 </****div****>  
 );  
 }  
}  
function mapStateToProps (state) {  
 return { // соединение с Redux  
 user: state.user,  
 supplier: state.supplier  
 }  
}  
export default connect(mapStateToProps)(SuppliersEditingList);*

Практически аналогичным образом выглядит и страница для редактирования пользователей в режиме “Admin”. За исключение того, что админ может редактировать различные данные пользователей для этого добавлены кнопки “Edit” возле каждого юзера и появляются специальные поля ввода и обработчики событий на них, что добавит всё-таки некоторую логику в обычную разметку. Но будем считать, что этот код аналогичен коду для изменения данных о ресторанах и к объяснению и печати он не обязателен.

# 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).

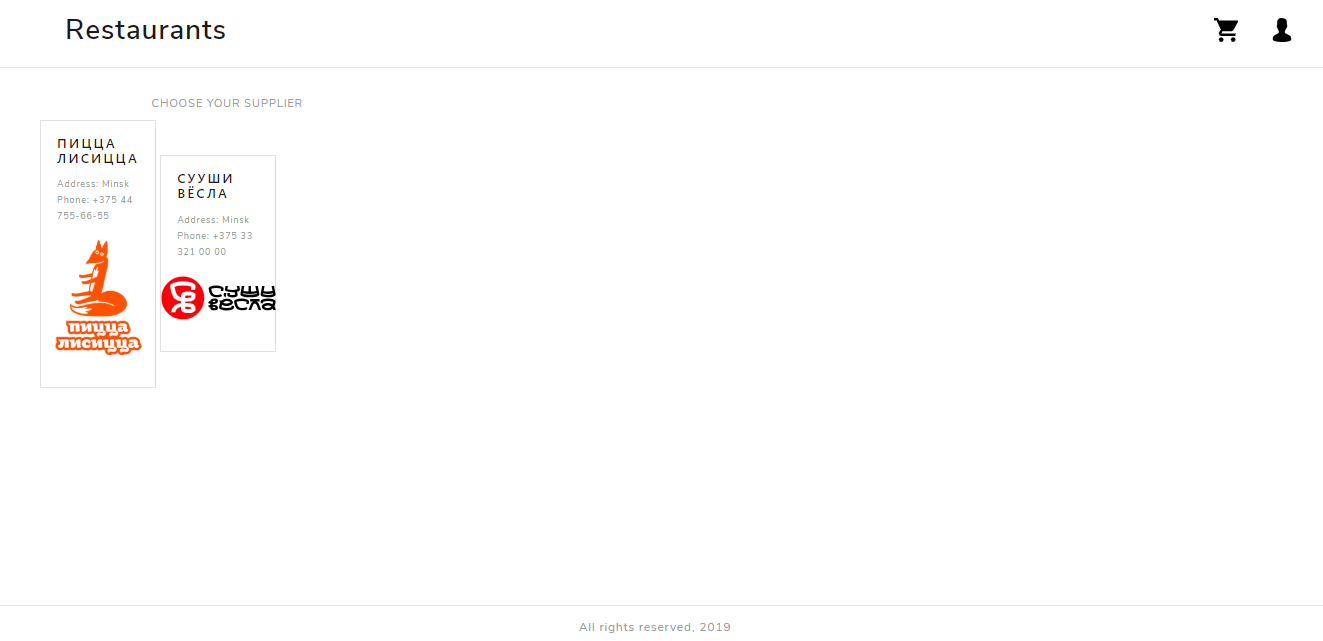


Рисунок 4.1 – Домашняя страница сайта

В верхней панели пользователю видно изображение корзины, при нажатии на которое отобразится корзина, и видны кнопки входа и регистрации, что говорит о том, что пользователь не авторизован. Если пользователь авторизован эти кнопки изменятся на одну иконку пользователя (см. рисунок 4.2).



Рисунок 4.2 – Изображение вместо кнопок регистрации и войти

Для перехода на страницу с едой требуется нажать на изображение какого-либо ресторана.

Для того что бы положить какое-либо блюдо в корзину необходимо под изображением блюда нажать на кнопку добавить в корзину

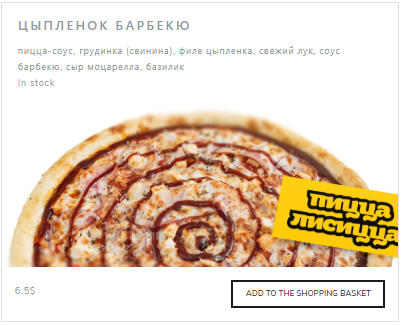


Рисунок 4.3 – Изображение одного блюда выбранного ресторана

Что бы убедиться, что товар добавлен в корзину или для того, чтобы изменить количество этого товара или убрать его из корзины необходимо нажать на значок корзины на верхней панели. В появившемся окне можно проделать все эти действия и оформить заказ (см. рисунок 4.4).

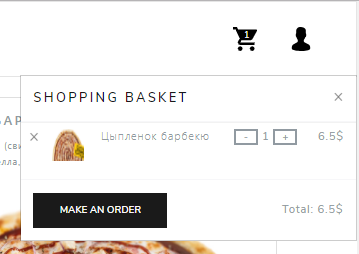


Рисунок 4.4 – Окно управления содержание корзины

Для регистрации в системе. Пользователь может нажать на кнопку «Sign Up» находясь на любой странице (см. рисунок 4.5). Или же, если вдруг пользователь нажмёт на “Sign in” не имея аккаунта снизу будет ссылка “ [Don't have account?](http://localhost:3000/signup)”, которая перенаправит на форму для регистрации(см.

рисунок 4.6).



Рисунок 4.5 – Кнопка перехода на форму регистрации и авторизации



Рисунок 4.6 – Страница формы для входа и ссылка на переход к форме регистрации

Находясь на странице с формой для регистрации, вводится электронная почта и пароль. После первого ввода данных могут появиться ошибки (см. рисунок 4.7. Для устранения данных ошибок надо просто начать вводить другие данные или исправить текущие. Возможные ошибки при отправке данных пользователя: «User with that email already exists!» (см. рисунок 4.8) - означает, что пользователь с такой электронной почтой уже существует, тогда пользователю надо либо зайти в систему, либо ввести другую электронную почту.

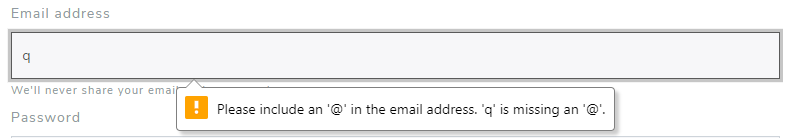


Рисунок 4.7 – Ошибки проверки данных формы

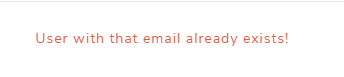


Рисунок 4.8 – Ошибка существования пользователя с таким же email

Для регистрации в системе новым пользователем надо перейти по ссылке «Sign up» на любой странице. В данной форме ошибки, связанные с вводом пароля и электронной почты описаны в пункте 7.

После авторизации в системе пользователь имеет возможность просматривать свою информацию, которую вводил при регистрации. Для этого необходимо нажать на иконку пользователя и выбрать “Your information” (см. рисунок 4.9).

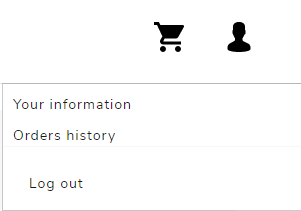


Рисунок 4.9 – Окно при нажатии на иконку пользователя

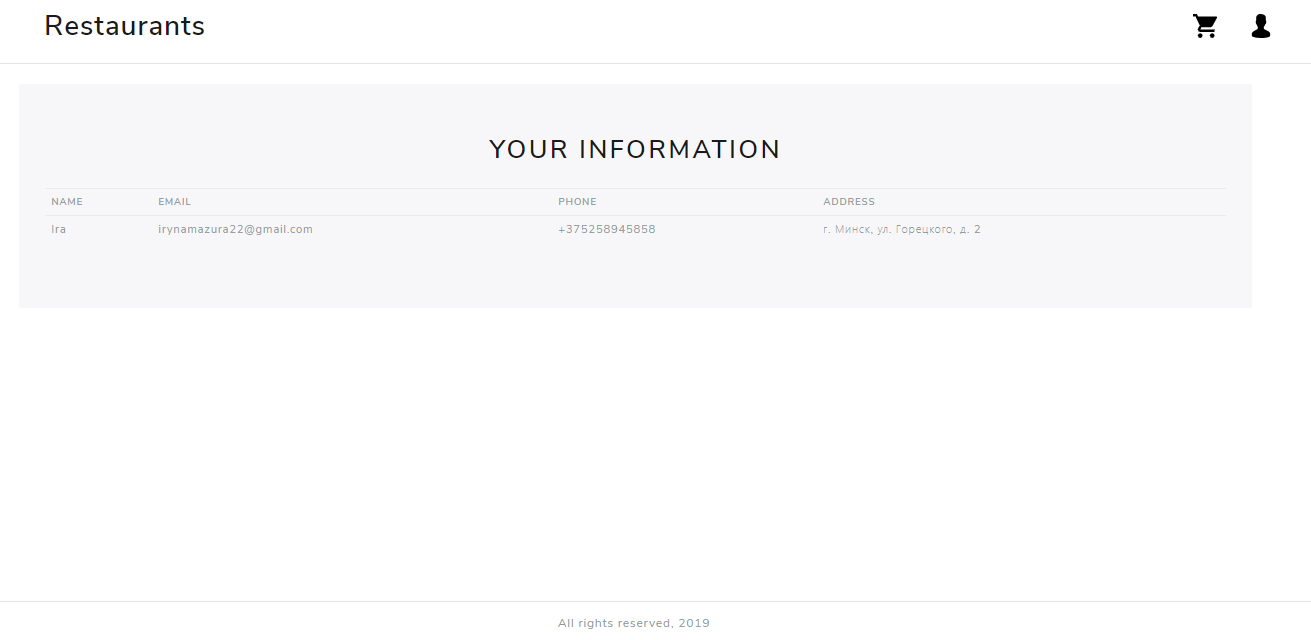


Рисунок 4.10 – Успешное открытие информации пользователя

Авторизованный пользователь имеет возможность делать заказы. Для этого необходимо проделать пункт 5 и нажать на кнопку “make an order”.

После этого откроется страницы с подтверждением заказа.

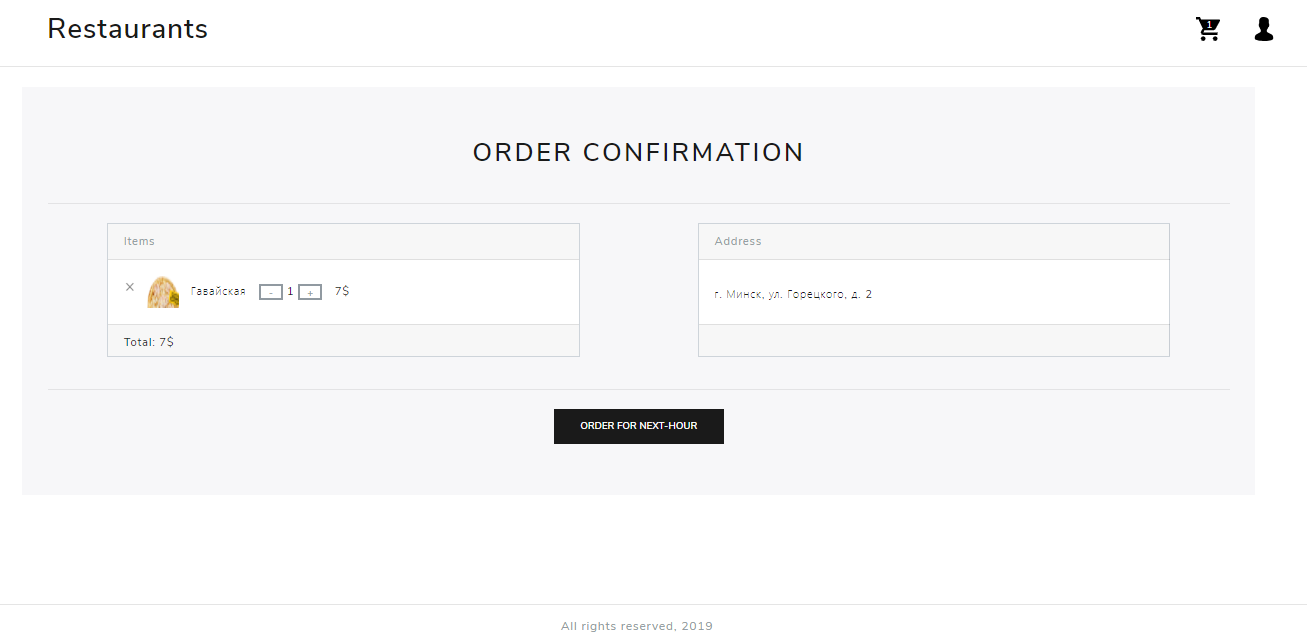


Рисунок 4.11 – Страница подтверждения заказа

Чтобы воспользоваться расширенными возможностями администратора необходимо войти в систему с электронной почтой – admin@gmail.com и паролем – root123.

Чтобы изменить какие-либо данные ресторанов необходимо нажать на иконку пользователя и выбрать “edit supplier data”. После этого откроется страница для редактирования (см. рисунок 4.12)

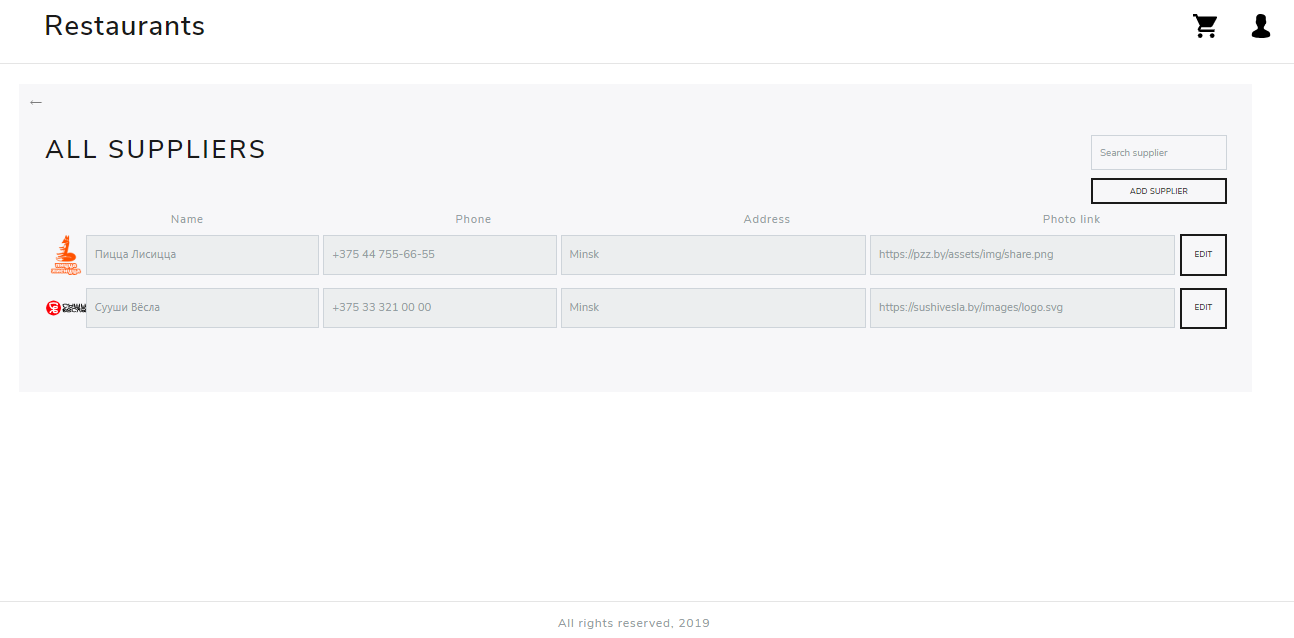


Рисунок 4.12 – Страница для редактирования данных о ресторанах

Для удаления ресторана требуется нажать по кнопке “Edit”. В таком случае напротив каждого ресторана появляется крестик для удаления, все поля становятся изменяемые (см. рисунок 4.13).

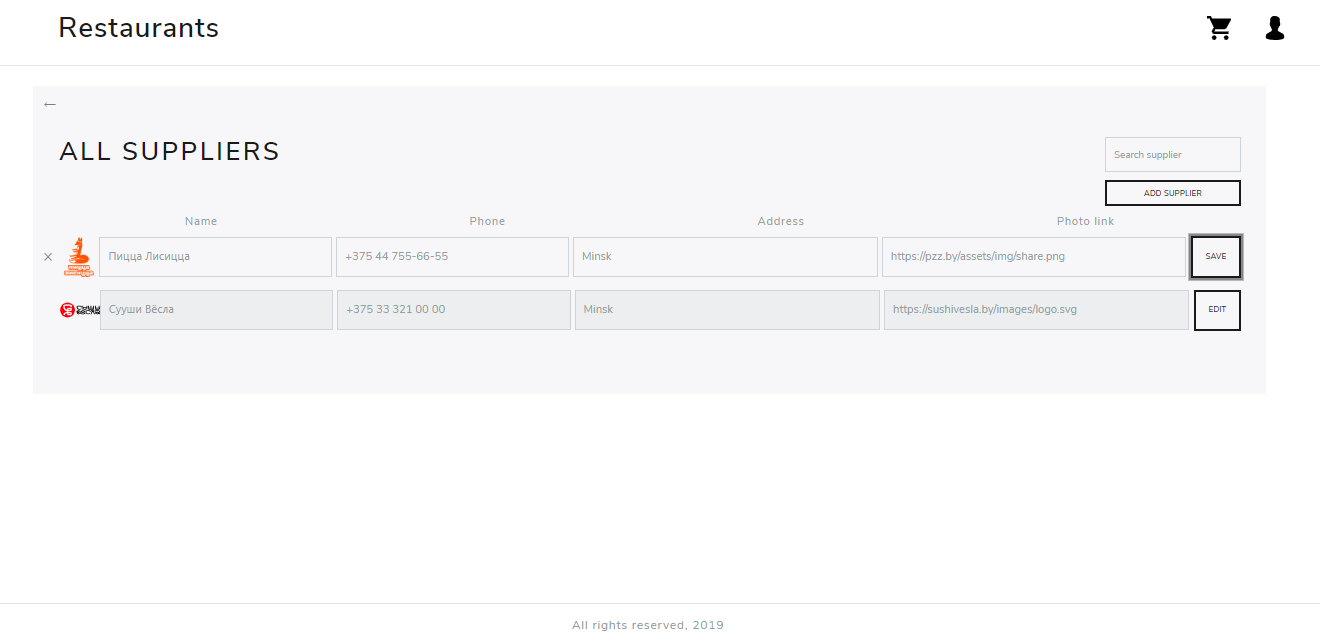


Рисунок 4.13 – Способ редактирования и удаления ресторана

Для редактирования пользователей необходимо нажать на иконку пользователя и выбрать “edit user profile”. После необходимых необходимо нажать на кнопку “edit” и все поля станут изменны (см. рисунок 4.14).

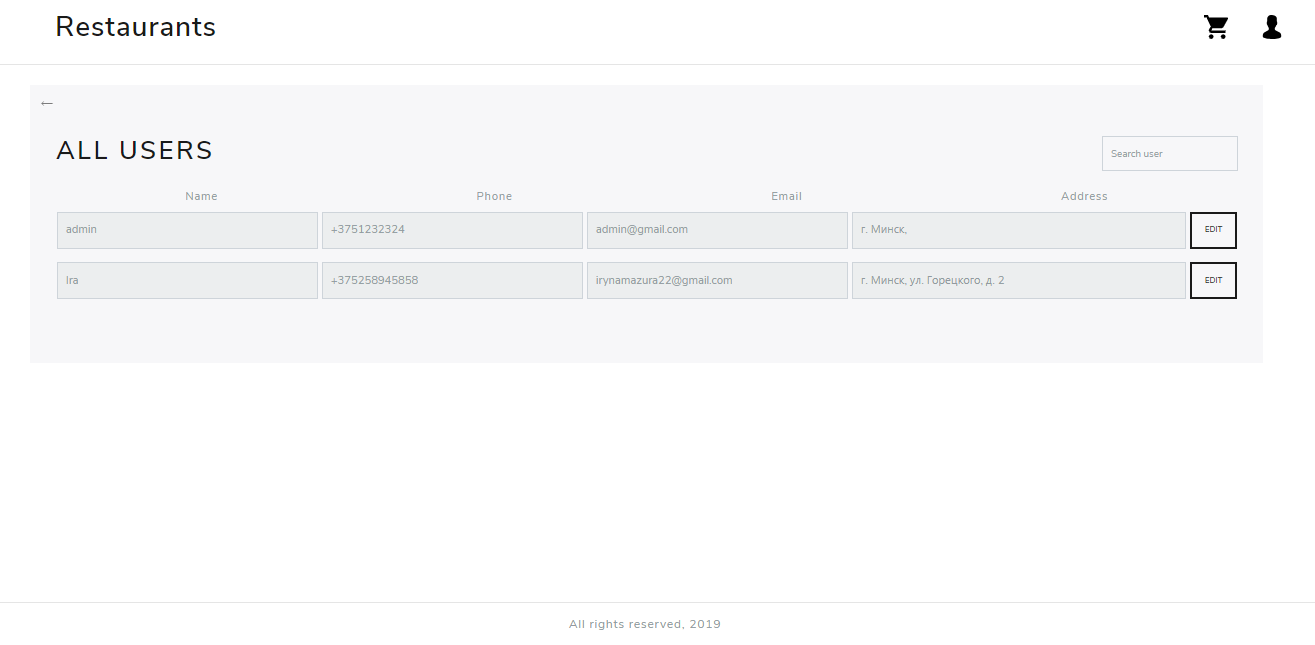


Рисунок 4.14 – Форма редактирования пользователей

Для выхода пользователем из системы требуется нажать кнопку «Logout» в окне которое открывается при нажатии на иконку пользователя. Произойдёт перенаправление на главную страницу (см. рисунок 4.15).

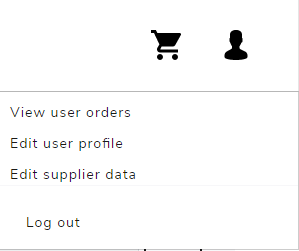


Рисунок 4.15 – Кнопка выхода из системы

Таким образом, работа с сайтом достаточно удобная и интуитивно понятна даже неопытным пользователям.

## ЗАКЛЮЧЕНИЕ

В результате проделанной работы был создан ресурс для управления каталогом ресторанов и их пищей. 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 делать заказы, возможность для администратора удаления, добавления, редактирования 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)**

**Маршрутизация сервиса**

import express from 'express';  
import \* as foodController from '../controllers/foodController.js';  
import \* as suppliersController from '../controllers/suppliersController.js';  
import \* as orderController from '../controllers/orderController.js';  
import \* as userController from '../controllers/userController.js';  
  
const apiRouter = new express.Router();  
  
apiRouter.get('/suppliers', suppliersController.getSuppliers);  
  
apiRouter.post('/admin/suppliers', suppliersController.createSupplier);  
  
apiRouter.put('/admin/suppliers', suppliersController.updateSupplier);  
  
apiRouter.delete('/admin/suppliers/:idSupplier', suppliersController.deleteSupplier);  
  
apiRouter.get('/suppliers/:idSupplier/food', foodController.getSuppliersFood);  
  
apiRouter.put('/admin/suppliers/food', foodController.updateSupplierFood);  
  
apiRouter.post('/admin/suppliers/:idSupplier/food', foodController.createSupplierFood);  
  
apiRouter.delete('/admin/food/:idFood', foodController.deleteSupplierFood);  
  
apiRouter.get('/admin/userOrders', orderController.getAllOrders);  
  
apiRouter.get('/admin/userOrders/today', orderController.getAllOrdersForToday);  
  
apiRouter.get('/admin/userOrders/week', orderController.getAllOrdersForWeek);  
  
apiRouter.get('/users/:idUser/orders', orderController.getUserOrders);  
  
apiRouter.get('/users/:idUser/orders/today', orderController.getUserOrdersForToday);  
  
apiRouter.get('/users/:idUser/orders/week', orderController.getUserOrdersForWeek);  
  
apiRouter.post('/users/:idUser/orders', orderController.createOrder);  
  
apiRouter.delete('/users/:idUser/orders', orderController.deleteOrder);  
  
apiRouter.get('/admin/userProfiles', userController.getUsers);  
  
apiRouter.put('/admin/userProfiles', userController.updateUser);

export default apiRouter;

# ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | *Текстовые документы* |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | | *Пояснительная записка* | *61с.* | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  |  | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л* | *№ докум.* | *Подп.* | *Дата* | *Автоматизированная система онлайн-заказов продуктов питания*  *Ведомость курсового проекта* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Мазура* |  |  |  | *Т* |  | *61* | *61* |
| *Провер.* | | *Трофимович* |  |  | *Кафедра ИТАС*  *гр. 620604* | | | | |
| *Т.контр.* | |  |  |  |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |
| *Реценз.* | |  |  |  |