**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

1. **Общие сведения**
   1. **Полное наименование системы**

Веб-приложение агрегатор услуг по ремонту

* 1. **Наименования предприятий разработчика и заказчика**

Разработчик : студент группы ИУК5-72Б Кузнецов Р.С. Заказчик : КФ МГТУ им. Баумана кафедра ИУК5.

* 1. **Перечень документов, на основании которых создается система**

Методические указания к выполнению курсовой работы по предмету “Вычислительные средства АСОИУ ”.

* 1. **Плановые сроки начала и окончания работы по созданию системы.**

Начало работы: 14.09.2022

Окончание работы: 30.12.2022

* 1. **Сведения об источниках и порядке финансирования работ**

Источников и порядков финансирования не имеется.

1. **Назначение и цели создания (развития) системы**
   1. **Назначение системы.**

Система предназначена для предоставления следующих возможностей:

* размещения услуг по ремонту для исполнителей;
* поиска услуг по ремонту для заказчиков;
* выбора подходящего исполнителя услуги;
* связи и общения по поводу услуги с выбранным исполнителем.
  1. **Цели создания системы.**
* Формирование теоретического представления, а также

практических навыков по разработке и реализации веб-приложения с использованием базы данных;

* разработка веб-приложения, реализующего двустороннюю связь в реальном времени между заказчиком и исполнителем;
* разработка серверной части веб-приложения при помощи Node.js, Express.js, MongoDB;
* разработка клиентской части веб-приложения при помощи Tailwind CSS, React.js, Redux toolkit, Axios;
* разработка бизнес-логики, которая будет реализовывать эффективное и грамотное взаимодействие с входными и выходными данными;
* получение первичных навыков ведения научно- исследовательской, проектной и производственно- технологической деятельности;
* обучение грамотному оформлению технической документации.

1. **Характеристики объекта автоматизации**

**3.1 Краткие сведения об объекте автоматизации.**

Объектом автоматизации являются процессы нахождения, размещения услуг по ремонту, а также коммуникации между исполнителем и заказчиком услуги.

1. **Требования к системе**
   1. **Требования к системе.**

Система должна работать в ОС Windows с помощью браузера Google Chrome, иметь удобный и понятный графический интерфейс. Предоставлять возможности размещения и нахождения услуг по ремонту, а также обмена информацией между исполнителем и заказчиком. При разработке должна быть использована база данных MongoDB для хранения информации. Средой разработки программного обеспечения является Visual Studio Code. Язык программирования – JavaScript.

* + 1. **Требования к структуре и функционированию системы**

Приложение должно быть разбито на 2 слоя:

* Слой графического пользовательского интерфейса – клиентская часть приложения: отображение навигационной панели (шапки), предоставляемых услуг, их исполнителей и цены;
* Слой бизнес логики – серверная часть приложения: реализация двусторонней связи между клиентами и сервером, обработка событий при взаимодействии пользователя с графическим интерфейсом веб-приложения, а также работа с базой данных.

**4.1.2 Требования к надежности**

Программа должна обеспечивать корректную обработку исключительных ситуаций, вызванных вводом недопустимых символов при регистрации, отсутствием доступа к базе данных.

* 1. **Требования к функциям (задачам), выполняемым системой**

Система должна позволять размещать и просматривать услуги по ремонту, а также предоставлять возможность коммуникации напрямую между заказчиком и исполнителем.

* + 1. **Слой пользовательского интерфейса.**

Пользовательский интерфейс должен быть реализован используя фреймворки Tailwind CSS, Redux Toolkit и React.js.

Состоит пользовательский интерфейс из двух частей:

* Статическая часть – навигационная панель (шапка веб-приложения);
* Динамическая часть – услуги, выставленные исполнителями, форма регистрации и авторизации, личный кабинет, чаты, отзывы.

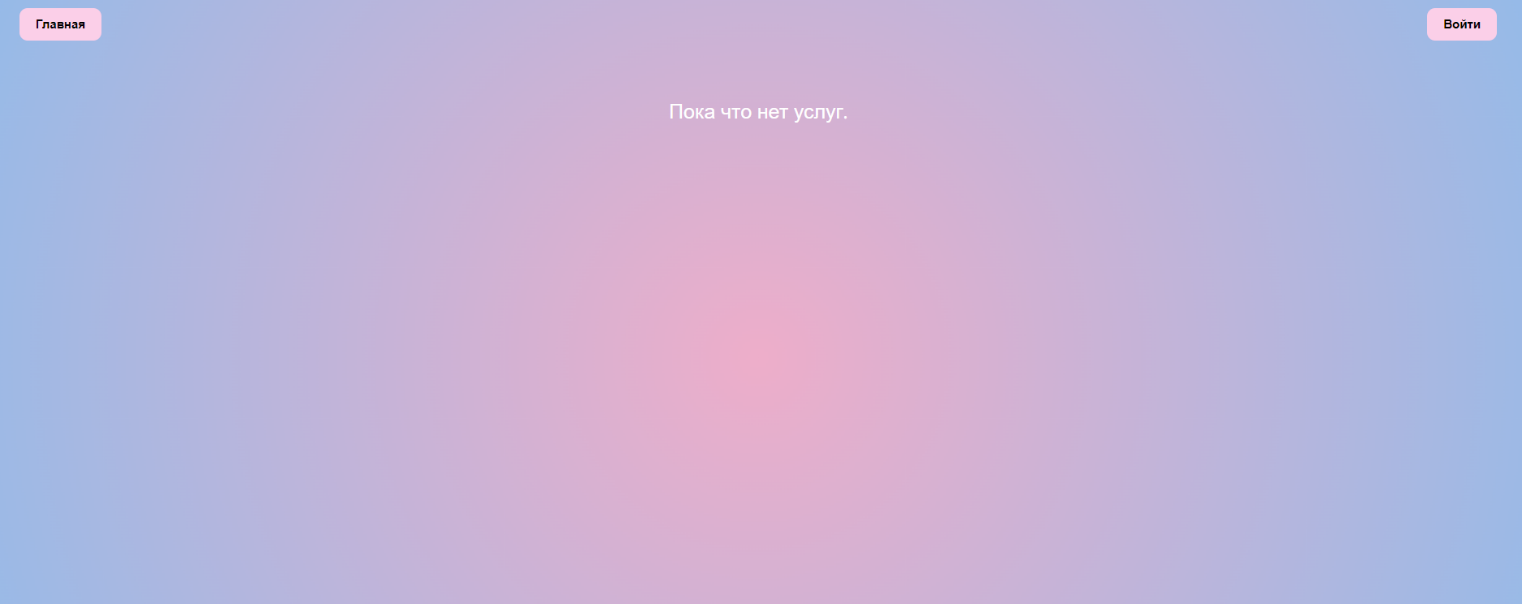


Рисунок 1.1 - Пример пользовательского интерфейса

* + 1. **Слой бизнес логики**

На уровне этого слоя должны быть реализованы следующие функции:

* + - * Обработка полученных данных;
      * Обеспечение создания двусторонней связи между клиентами и сервером;
      * Передача информации на клиентскую часть;
      * Взаимодействие пользователя с интерфейсом веб-приложения – реакция при нажатии на кнопки.

1. **Состав и содержание работ по созданию системы**

|  |  |  |
| --- | --- | --- |
| Стадии | Этапы работ | Сроки исполнения |
| 1. Формирование требований к АС | * 1. Исследование объекта и подтверждение   необходимости создания АС.   * 1. Формирование требований пользователя к АС.   2. Оформление отчёта о   выполненной работе и заявки на разработку АС. | 21.03.2022 |
| 2. Разработка концепции АС. | * 1. Изучение выбранного объекта.   2. Проведение научно- исследовательских работ.   3. Проектирования концепции АС, удовлетворяющей   потребности пользователя.   * 1. Оформление отчёта о выполненной работе. | 04.04.2022 |
| 3. Техническое задание. | Разработка и утверждение технического задания на  создание АС. | 18.04.2022 |
| 4. Разработка системы. | Разработка системы согласно  техническому заданию. | 23.05.2022 |
| 5. Защита курсовой работы. | * 1. Создание грамотной презентации и речи для защиты курсовой работы.   2. Защита курсовой работы. | 08.06.2022 |

1. **Порядок контроля и приемки системы.**
   1. **Состав, объем и методы испытаний системы и ее составных частей.**

Приложение должно пройти предварительные испытания, состоящие из отладки и минимального набора тестов.

В результате предварительных испытаний, должны быть исправлены недочёты, замечания на которые были получены в ходе предварительных испытаний. Для проверки корректной работы внесённых изменений должны быть проведены повторные испытания разработанной программы.

* 1. **Общие требования к приемке работ.**

В процессе приемки работ должна быть осуществлена проверка на соответствие требованиям настоящего «Технического задания». По результатам испытаний возможны доработки и исправления.

При приёмке работ осуществляется проверка на соответствие требованиям актуального технического задания «Технического задания». По

результатам приёмки можно исправить недочёты и внести некоторые доработки.

1. **Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие.**

Работ по подготовке объекта автоматизации проводить не требуется.

1. **Требования к документированию**

По окончанию работы предъявлена расчетно-пояснительная записка в состав которой входят:

* техническое задание;
* научно-исследовательская часть;
* проектно-конструкторская часть;
* проектно-технологическая часть.

Также должна быть предоставлена графическая часть работы, выполненная формате А1 на 2 листах, в которую входят:

* демонстрационные чертежи;
* алгоритмические схемы.

1. **Источники разработки**

Гост 34.601-90

Гост 34.602-89

1. **Научно исследовательская часть**

**2.1 Постановка задачи проектирования**

Ежедневно у людей выходит из строя, ломается огромное количество вещей (телефон, стиральная машина, холодильник), из-за этого человек, у которого что-то сломалось, будет искать специалиста, способного починить необходимую вещь. Отсюда возникает необходимость: специалисту - найти заказчика и предоставить ему услугу по ремонту; потребителю – найти исполнителя и получить услугу по ремонту. Намного удобнее и эффективнее исполнителю найти заказчика, а заказчику исполнителя, позволит специальный сайт-агрегатор, на котором будут отображаться услуги, их исполнители и цена, а также который предоставит возможность связаться заказчику с исполнителем услуги напрямую, чтобы удобнее обеспечить процесс согласования цены, времени и других мелочей, относящихся к конкретному случаю. Поэтому необходимо реализовать такое веб-приложение, которое предоставляло бы вышеописанные возможности.

Для реализации веб-приложения необходимо использовать базу данных MongoDB, серверную платформу Node.js, язык программирования JavaScript, пакет Redux toolkit, фреймворки React.js, Express.js, Tailwind CSS.

**2.2 Описание предметной области.**

Процессы аккумулирования и отображения услуг по ремонту от различных исполнителей, а также коммуникации между заказчиком и исполнителем по поводу конкретной услуги относительно несложные, но комплексные. Чтобы успешно реализовать эти процессы необходимо проделать много работы и грамотно подойти к этому.

Сперва нужно предоставить возможность исполнителям добавлять свои услуги, а также изменять или удалять их при необходимости. Затем в удобной и понятной форме отображать их. Кроме того, для заказчика возможность сортировки по категориям сделает процесс поиска услуг гораздо удобнее. Следовательно, также нужно реализовать функционал сортировки. Далее необходимо упростить процесс выбора конкретного специалиста для заказчика. Сделать это можно с помощью комментариев (отзывов) к конкретной услуге исполнителя, а также непосредственной коммуникацией между исполнителем и заказчиком используя отдельный чат. Так как комментарии могут оставлять любые пользователи, то необходимо реализовать фильтрацию нецензурных и спам слов. Кроме того, нужно иметь уполномоченный персонал (модерация), который будет иметь возможность удаления недопустимых услуг и комментариев к ним. Следовательно, необходимо разграничить обычных пользователей с модерацией, а также их полномочия на сайте.

Самым логичным и универсальным вариантом для реализации вышеперечисленного функционала будет разработка «клиент-серверного» веб-приложения с использованием базы данных.

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.

Клиентская часть будет реализовывать пользовательский интерфейс, формировать запросы к серверу и обрабатывать ответы от него.

Серверная часть — получать запрос от клиента, выполнять вычисления, после этого формировать веб-страницу и отправлять её клиенту по сети с использованием протокола HTTP.

Для создания веб-приложений на стороне сервера используются разнообразные технологии и языки программирования, способные осуществлять вывод в стандартную консоль.

На стороне клиента будут использоваться:

* Для реализации GUI
* Tailwind CSS
* React.js
* Для формирования и отправки запросов на сервер:
* JavaScript
* Axios
* Redux toolkit

На стороне сервера:

* Для хранения данных
* MongoDB
* Для формирования запросов, обработки и отправки данных на клиент:
* JavaScript
* Node.js
* Express.js
* Mogoose

**2.3 Анализ аналогов и прототипов**

Для того чтобы сделать хороший проект, необходимо проанализировать и сравнить уже существующие, самые часто используемые веб-приложения агрегаторы, и выделить их основные преимущества и недостатки. Воспользовавшись недочётами других разработчиков, не повторим их в своем проекте и сделаем более удобное веб-приложение.

Посмотрим на преимущества и недостатки Яндекс Услуг https://uslugi.yandex.ru/

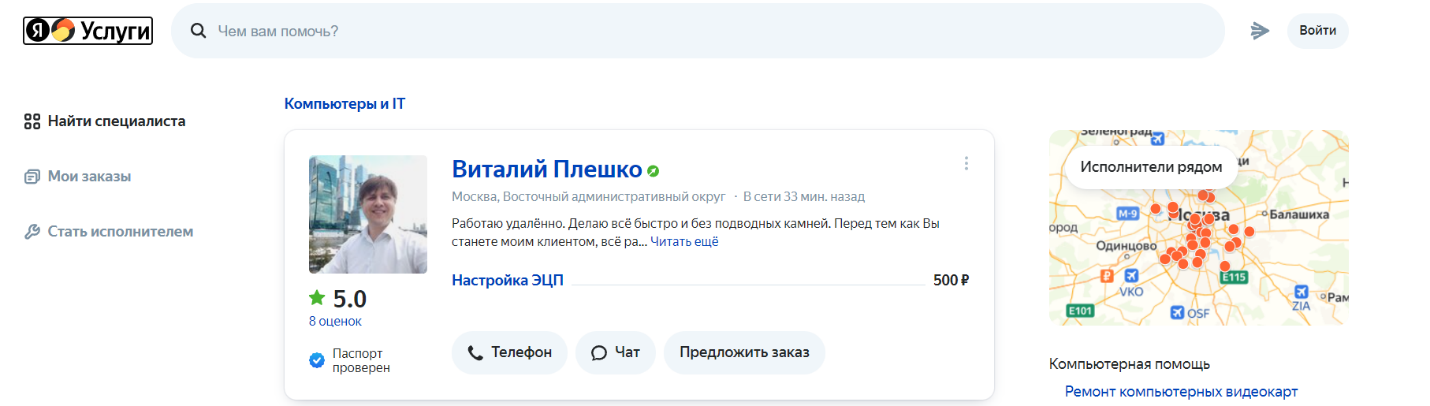


Рисунок 2.1 - Интерфейс сайта Яндекс Услуги

Преимущества:

* возможность стать исполнителем, а также разместить заказ;
* возможность сортировки услуг по различным критериям;
* информативная карточка исполнителя со всей необходимой информацией;

Недостатки:

* загруженный интерфейс, изобилие функционала => сложно ориентироваться на сайте;
* отсутствие возможности связаться с исполнителем напрямую через сайт, а только через дополнительные приложения (WhatsApp, Яндекс Мессенджер и т.д.).

Далее проанализируем сайт https://youdo.com/

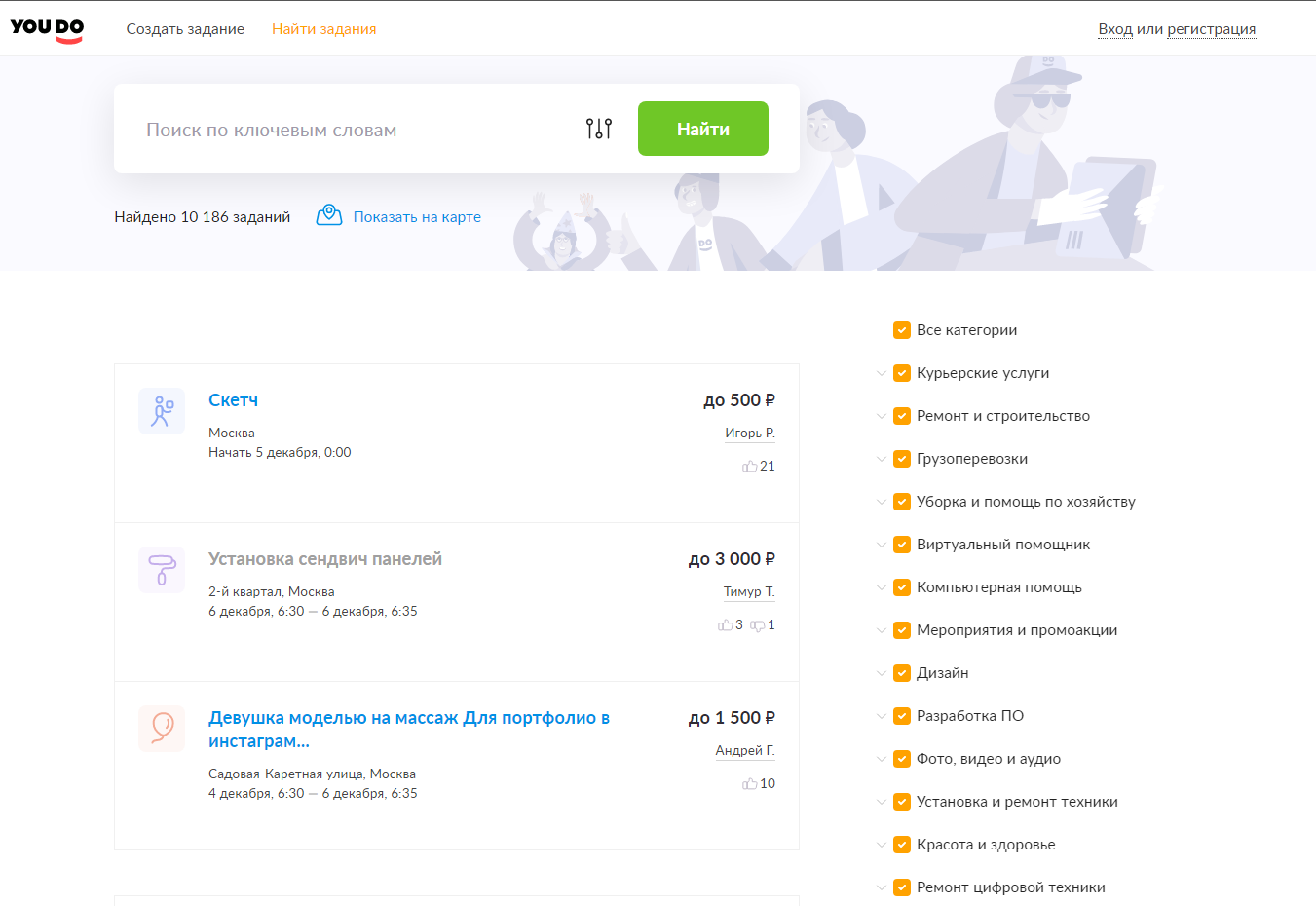


Рисунок 2.2 – Интерфейс сайта https://youdo.com/

Преимущества:

* приятный и удобный интерфейс;
* лаконичная карточка услуги со всей необходимой информацией, но без излишеств;

Недостатки:

* отсутствие фотографии исполнителя на карточке услуги;
* отсутствие возможности связаться с заказчиком/исполнителем напрямую на сайте:
* отсутствие быстрой возможности искать исполнителей собственноручно (для этого нужно создавать конкретный заказ на услугу, который не факт что увидят и захотят делать).

Проанализировав аналоги и прототипы, можно сделать вывод, что существующие веб-приложения агрегаторы не позволяют быстро решать все поставленные задачи, поэтому необходимо разработать собственное веб-приложение, которое бы предоставляло удобные возможности размещения и отображения услуг по ремонту, а также быстрой связи с конкретным исполнителем.

**2.4 Перечень задач, подлежащих решению в процессе разработки.**

* разработка пользовательского интерфейса с использованием React.js и Tailwind CSS;
* разработка базового функционала веб-приложения (пользовательский интерфейс, клиентская часть с использованием React.js, Tailwind CSS и JavaScript, серверная часть с использованием Node.js, Express.js, MongoDB);
* разработка модели базы данных;
* разработка запросов с использованием Axios, Redux, Mongoose;
* привязка запросов к пользовательскому интерфейсу;
* привязка базы данных к серверной части веб-приложения для хранения данных;
* тестирование разработанного приложения;
* исправление выявленных ошибок.

**2.5 Обоснование выбора инструментов и платформы для разработки.**

Для успешной разработки проекта, используется операционная система Windows 10. Windows 10 — [операционная система](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) для [персональных компьютеров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) и [рабочих станций](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B1%D0%BE%D1%87%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D0%BD%D1%86%D0%B8%D1%8F), разработанная корпорацией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в рамках семейства [Windows NT](https://ru.wikipedia.org/wiki/Windows_NT).

Данная платформа предоставляет простой и удобный доступ к использованию широкого набора инструментов для разработки:

* редакторы кода;
* менеджеры пакетов;
* базы данных;
* интегрированные среды для разработки.

Согласно Statcounter.com, общий рейтинг операционных систем, показывает, что лидером является Windows, который установлен на 76,58% всех компьютеров, OS X — 18,93% и Linux — 1,62%. Платформа Windows является самой популярной и доступной, также для этой платформы создано много удобных средств, для облегчения выполнения проекта.

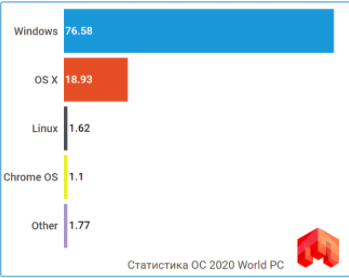


Рисунок 2.3 - Статистика использования операционных систем в мире.

**VS Code**

Для успешной разработки проекта, используется редактор кода Visual Studio Code.

Visual Studio Code (VS Code) от Microsoft – [текстовый редактор](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) для [кроссплатформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) разработки. Распространяется бесплатно, разрабатывается как программное обеспечение с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), но готовые сборки распространяются под [проприетарной лицензией](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%80%D0%B8%D0%B5%D1%82%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Функциональные возможности позволяют создавать, тестировать и разворачивать полноценные проекты.

Ключевые особенности среды разработки VS Code:

* подсветка и дополнение кода при указании типа данных, создании языковых конструкций (классов, методов) – технология IntelliSense;
* функционал отладки кода;
* встроенные команды Git;
* расширение функционала добавлением новых языковых модулей и тем;
* простота развёртывания проектов.

Ниже перечислены некоторые популярные возможности Visual Studio Code, которые помогут вам повысить продуктивность разработки программного обеспечения.

* волнистые линии и [быстрые действия](https://docs.microsoft.com/ru-ru/visualstudio/ide/quick-actions?view=vs-2019).

Волнистые линии обозначают ошибки или потенциальные проблемы кода прямо во время ввода. Эти визуальные подсказки позволяют устранять проблемы немедленно и не ждать, пока ошибка будет обнаружена во время сборки или запуска программы. Если навести указатель мыши на волнистую линию, на экран будут выведены дополнительные сведения об ошибке. Кроме того, в поле слева может появляться значок лампочки с быстрыми действиями по устранению ошибки.

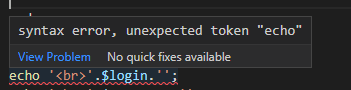


Рисунок 2.4 - Волнистые линии и быстрые действия

* поиск в Visual Studio Code

Среда Visual Studio Code может показаться сложной, ведь там столько разных меню, параметров и свойств. Чтобы быстро находить функции интегрированной среды разработки и элементы кода, в Visual Studio Code представлен единый компонент поиска (**CTRL**+**Q**).

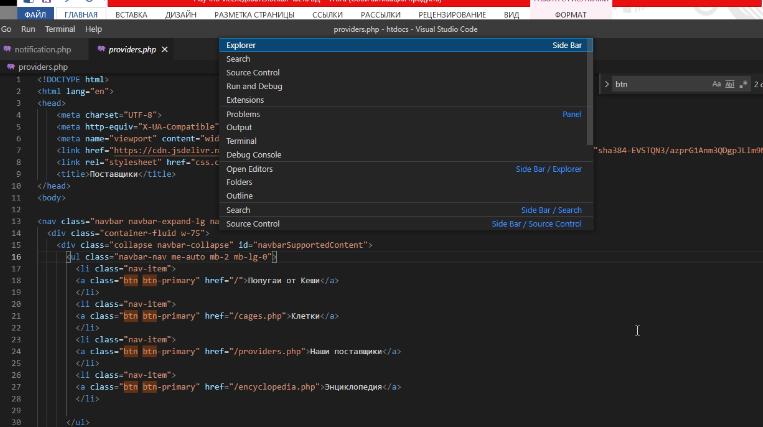


Рисунок 2.5 - Поиск в Visual Studio Code

Согласно опросу, проведенного среди 3240 разработчиков в феврале 2018 организацией Standard, разработчики выделили такие средства разработки, как наиболее часто используемые:

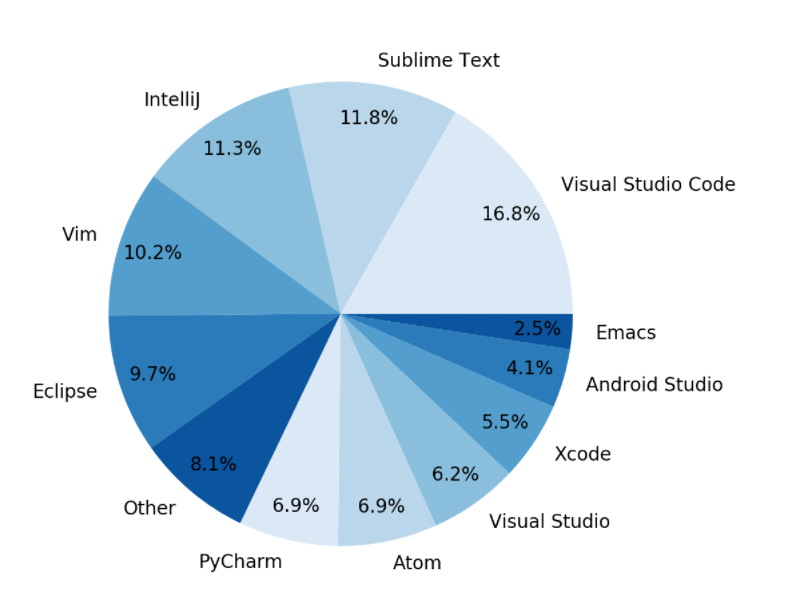


Рисунок 2.6 - Статистика использования платформ для разработки систем в мире.

Первое место среди IDE у Visual Studio Code с 16.8% использующих его разработчиков.

Также, Visual Studio Code обладает хорошим отладчиком, с помощью которого можно легко отследить ошибки в коде и без проблем их исправить, позволяет удобно и быстро создавать проекты и работать с ними.

По сравнению со средой разработки Eclipse, Visual Studio Code имеет более простой интерфейс в использовании.

**NPM**

Кроме того, следует выбрать *менеджер пакетов NPM*. Менеджеры пакетов представляют собой наборы инструментов для автоматизации различных процессов: установка, настройка, обновление и удаление приложений. NPM предлагает расширенный функционал, включая следующее:

* менеджер пакетов Node.js, позволяющий разработчикам делиться пакетными модулями;
* реестр NPM, представляющий собой открытую библиотеку пакетов с открытым исходным кодом для Node.js, веб-приложений для фронтенда и прочего;
* NPM также является менеджером пакетов в интерфейсе командной строки для их установки и публикации.

**React.js**

React — это JavaScript фреймворк и библиотека для разработки пользовательского интерфейса. Он универсален и гибок; JSX делает написание пользовательских компонентов проще; поддерживает мобильную разработку; более простое и дружелюбное написание SEO разметки; используется минимум в 1.6 миллионах сайтах; предлагает лучшую экосистему, больше шаблонов и дополнительных инструментов. Поэтому для разработки клиентской части выбор был сделан в пользу данного фреймворка.

**Node.js**

Одним из наиболее известных средств для back-end разработки программных решений на JavaScript считается платформа Node.js. Она очень проста и понятна интуитивно, что является огромным плюсом для начинающих разработчиков. При этом Node.js позволяет работать с серверными технологиями.

Сегодня Node считается одной из лидирующих платформ для веб-разработки, и большая часть веб-инструментов, серверных и клиентских, работают с этим инструментом.

Одна из причин, по которой рекомендуется Node.js по сравнению с другими фреймворками: поскольку JavaScript также используется на стороне клиента, то получается единый язык для всего приложения, вследствие чего нужно учить на один язык меньше.

Один из важнейших плюсов платформы – это асинхронность в сочетании с событийным подходом. В отличие от потокового, событийно-ориентированное программирование основано на каких-то внешних действиях. Т.е. выполнение программы зависит от действий пользователя или поступившего в программу сетевого пакета и т.д.

Такое решение упрощает программирование интерактивных приложений, т.е. основанных на работе с использованием ввода-вывода. Это могут быть онлайн-чаты или игровые приложения, красочные интерактивные веб-сервисы или всевозможные калькуляторы, рейтинги, голосования и т.д.

Кроме того, к преимуществам платформы относят:

* простой и широко известный JavaScript;
* богатая стандартная библиотека;
* огромное внешних библиотек и готовых модулей.

Над постоянным улучшением платформы работают тысячи заинтересованных программистов из разных стран. Проект распространяется с открытым кодом, а потому развивается огромными темпами.

**MySQL/NoSQL MongoDB**

*Базы данных SQL*

Также известные как реляционные базы данных, определяют и управляют данными на основе языка структурированных запросов (SQL). Они чаще всего используются для обработки структурированных данных, которые организуют элементы данных и стандартизируют то, как они будут связаны друг с другом и с их различными свойствами.

*Базы данных NoSQL*

Также известные как нереляционные базы данных, позволяют хранить и извлекать неструктурированные данные с использованием динамической схемы. NoSQL широко распространены из-за своей гибкой способности создавать уникальную структуру, которая может быть в форме документа, графика, столбца, организованные как структура данных.

SQL на протяжении десятилетий значительно превосходил нереляционные альтернативы, но NoSQL быстро сокращает разрыв с популярными базами данных. Хотя многие организации предпочитают переходить с устаревших баз данных, таких как Oracle, но не все переходят на NoSQL.

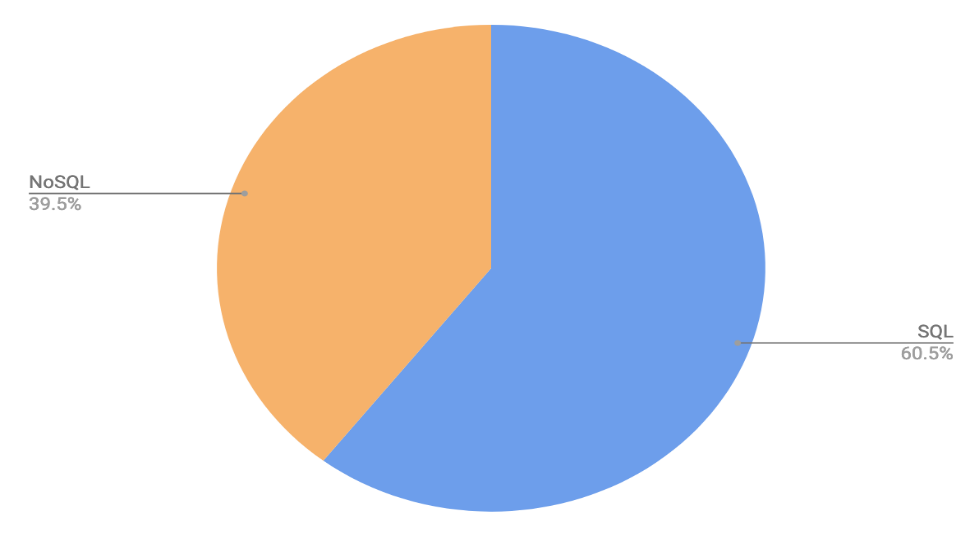


Рисунок 2.7 — Использование SQL и NoSQL

Используя данные исследования ScaleGrid — служба размещения баз данных в общедоступных и частных облаках, можно увидеть, что MySQL доминирует с 38,9% использования, за ней следуют MongoDB с 24,6%, PostgreSQL с 17,4%, Redis с 8,4% и Cassandra с 3,0%. Oracle отстал всего на 1,8%, а пользователи CouchDB, Berkeley DB, Microsoft SQL Server, Redshift, Firebase, Elasticsearch и InfluxDB объединили категорию «Другие» с 2,4%. Нельзя не отметить рост популярности MySQL, MongoDB и PostgreSQL.

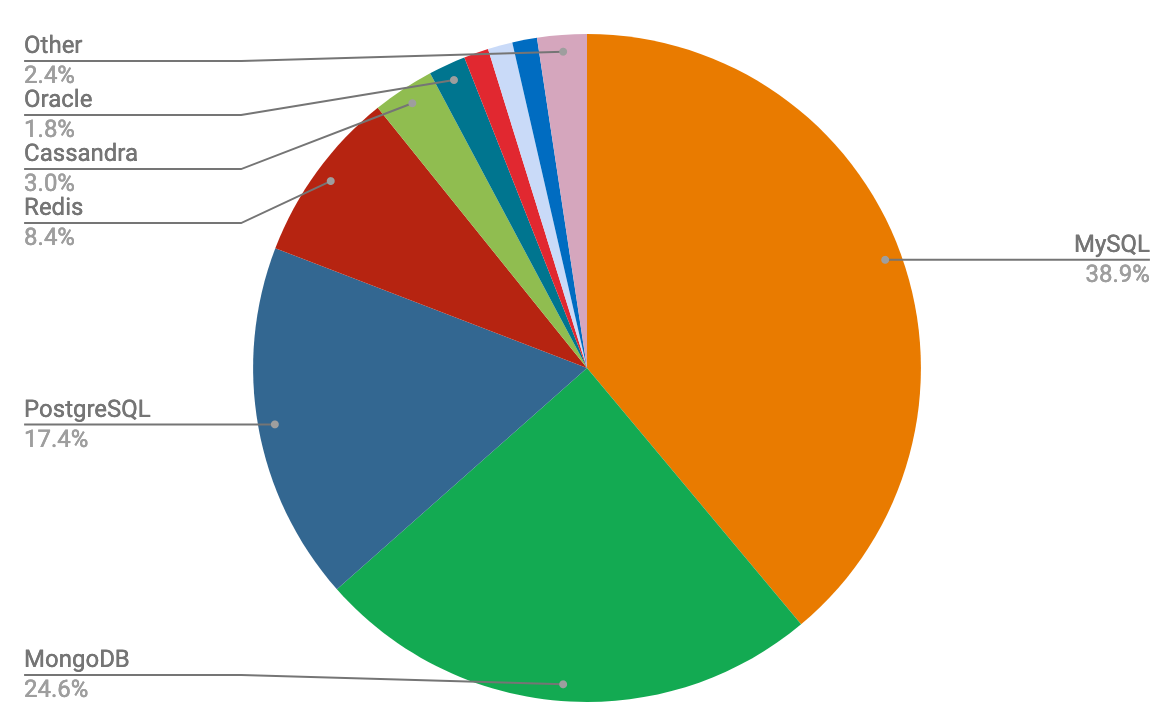


Рисунок 2.8 — Самые популярные базы данных

MongoDB построена на масштабируемой архитектуре, которая стала популярной среди разработчиков всех видов для разработки масштабируемых приложений с развивающимися схемами данных. В качестве базы данных документов, MongoDB позволяет разработчикам легко хранить структурированные или неструктурированные данные. Она использует формат, подобный JSON, для хранения документов. Этот формат напрямую сопоставляется с нативными объектами в большинстве современных языков программирования, что делает его естественным выбором для разработчиков, поскольку им не нужно думать о нормализации данных. MongoDB был создан для людей, создающих интернет-приложения и бизнес-приложения.

В MongoDB записи хранятся в виде документов в сжатых файлах BSON. Документы можно получить непосредственно в формате JSON, который имеет много преимуществ:

* это естественная форма для хранения данных;
* он удобочитаем;
* структурированная и неструктурированная информация может храниться в одном и том же документе;
* можно вложить JSON для хранения сложных объектов данных;
* JSON имеет гибкую и динамическую схему, поэтому добавление полей или удаление поля не является проблемой.
* документы сопоставляются с объектами в большинстве популярных языков программирования.

Исходя из вышеописанных преимуществ, было принято решение использовать базу данных MongoDB для разработки данного веб-приложения агрегатора.

**3. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ**

1. **Разработка структуры приложения**

Для представления принципа работы приложения ниже приведена ее

UML диаграмма вариантов использования (рис. 3.1).

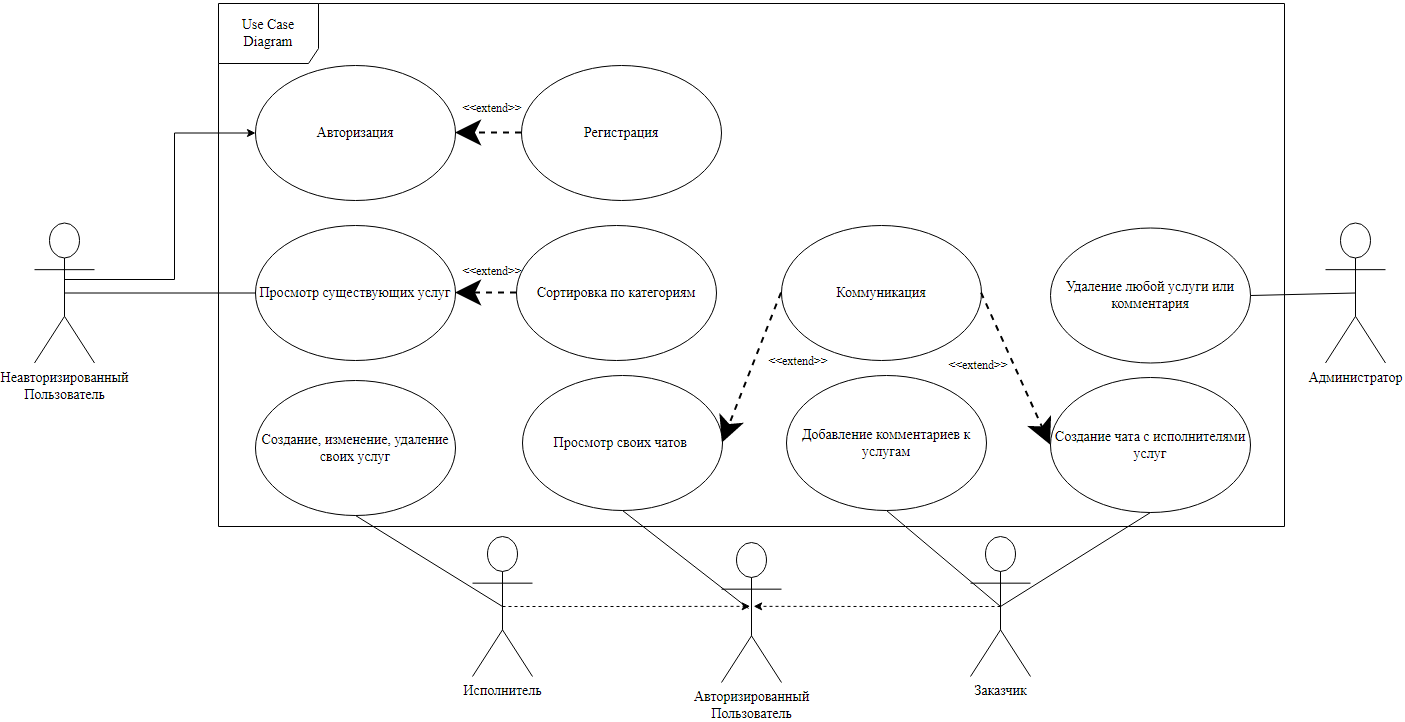


Рисунок 3.1 – UML диаграмма вариантов использования

При попадании на веб-приложение агрегатор услуг по ремонту пользователь сам решает, что ему необходимо сделать. Например, он может просматривать все добавленные услуги другими пользователями-исполнителями без необходимости регистрироваться. Однако, если пользователь-заказчик захочет связаться с пользователем-исполнителем услуги (с помощью чата) или оставить комментарий на услугу, то ему придётся пройти процесс регистрации, а затем авторизации. Если пользователь-исполнитель захочет добавить свою услугу, которая будет отображаться всем другим пользователям, то ему также необходимо пройти процессы регистрации и авторизации.

В любой момент работы с веб-приложением, пользователь может авторизоваться, зарегистрироваться или выйти из аккаунта.

Модератор (администратор) поддерживает порядок на сайте и контролирует корректность информации, предоставляемой пользователями. Модератор может удалять любые услуги и комментарии к ним, если сочтет их содержание неприемлемым или вводящим в заблуждение.

Общее назначение программного средства – предоставление возможностей размещения и просмотра услуг по ремонту, а также коммуникации между заказчиком и исполнителем.

Реализуемая задача состоит в том, чтобы хранение всей информации, необходимой для существования веб-приложения осуществлялось в базе данных, а также производилось удобное взаимодействие с пользовательским интерфейсом.

1. **Разработка интерфейса взаимодействия пользователя с системой**

Проектирование макета веб-приложения происходило в Figma. Верстка сайта и проектирование веб-интерфейса осуществлялась с помощью React, Tailwind CSS, JavaScript.

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.  React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

Tailwind CSS — это CSS-фреймворк с открытым исходным кодом. Основная особенность этой библиотеки заключается в том, что, в отличие от других фреймворков CSS, таких как Bootstrap, она не предоставляет ряд предопределенных классов для таких элементов, как кнопки или таблицы.

На рисунке 3.2 представлен разработанный интерфейс взаимодействия пользователя с системой.

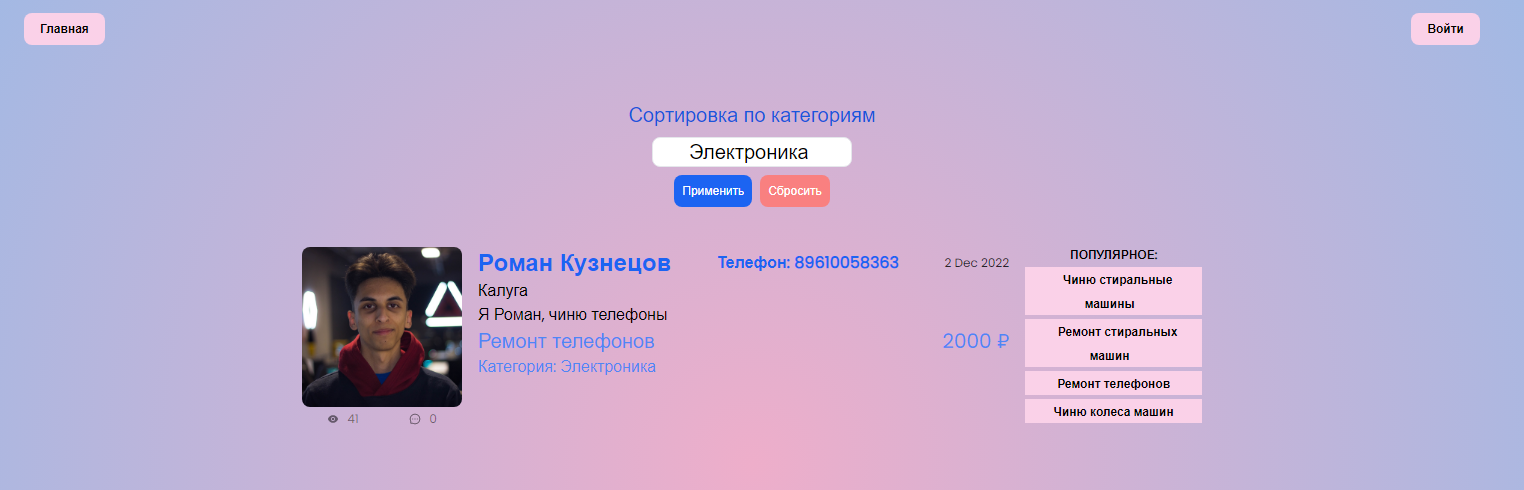


Рисунок 3.2 – Пользовательский интерфейс веб-приложения

1. **Логическая схема базы данных**

Документ «User» содержит информацию о пользователе: логин (тип данных – строка, необходимое поле, уникальный), пароль (тип данных – строка, необходимое поле), имя, фамилию и город (тип данных – строка), номер телефона (тип данных – число), является ли он администратором или нет (тип данных – логический), услуги этого пользователя (тип данных – массив ObjectId, 12-байтовый шестнадцатеричных строк типа BSON, например, ObjectId("6009c0eee65f6dce28fb3e50")).

Документ «Service» содержит информацию об услуге: логин владельца, заголовок, основной текст, категория (типы данных – строка, необходимые поля), изображение пользователя (тип данных – строка), цена на услугу, просмотры (типы данных – число), владелец услуги (тип данных – ObjectId), комментарии к услуге (тип данных – массив ObjectId).

Документ «Comment» содержит информацию о комментариях. Здесь происходит хранение текста комментария и имени отправителя (типы данных – строки, необходимые поля), автор комментария (тип данных – ObjectId).

Документ «Chat» содержит информацию о чатах. Здесь хранится информация об обоих пользователях (типы данных – ObjectId) и сообщения этого чата (тип данных – строка).

Документ «ChatMessage» содержит информацию о сообщении чата. Здесь осуществляется хранение идентификаторов чата и отправителя, имени отправителя и само сообщение. Все поля обязательные, типы данных - строки

На рисунке 3.3 представлена логическая схема базы данных.

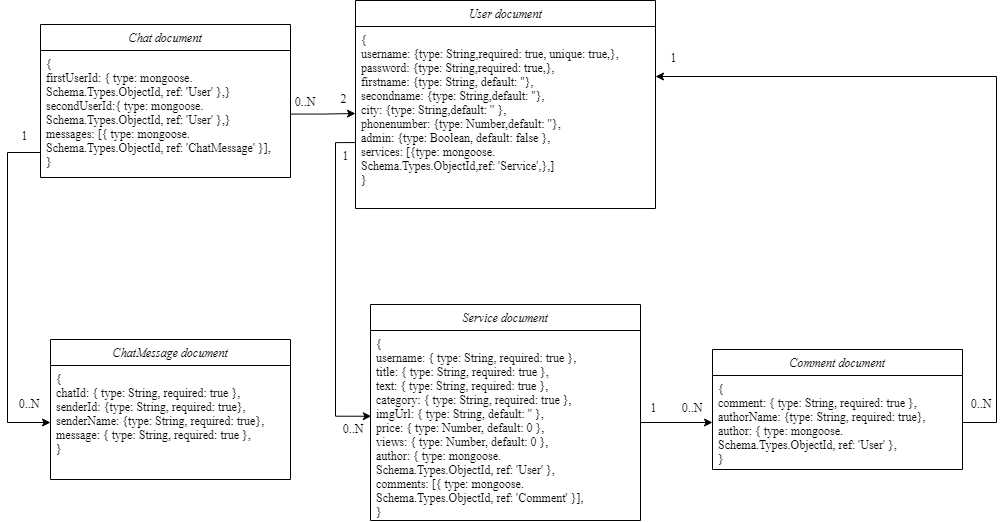


Рисунок 3.3 – Логическая модель данных

1. **Разработка архитектуры приложения**

Данная система представляет собой веб-приложение. Это особый тип программ, построенных по архитектуре "клиент-сервер". Особенность их заключается в том, что само веб-приложение находится и выполняется на сервере, а клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Передача запросов и результатов их обработки происходит через Интернет.

За счет наличия исполняемой части, веб-приложения способны выполнять практически те же операции, что и обычные Windows-приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными — Интернет.

К наиболее типичным операциям, выполняемым Web приложениями, относятся:

• прием данных от пользователя и сохранение их на сервере;

• выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД, проведение сложных вычислений;

• аутентифицирование пользователя и отображение интерфейса системы, соответствующего данному пользователю;

• отображение постоянно изменяющейся оперативной информации и т.д.

При разработке данного веб-приложения в курсовой работе было принято решение использовать клиент-серверную архитектуру с использованием паттерна MVC.

Model-View-Controller (MVC) – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Общий вид паттерна представлен на рисунке 3.4.

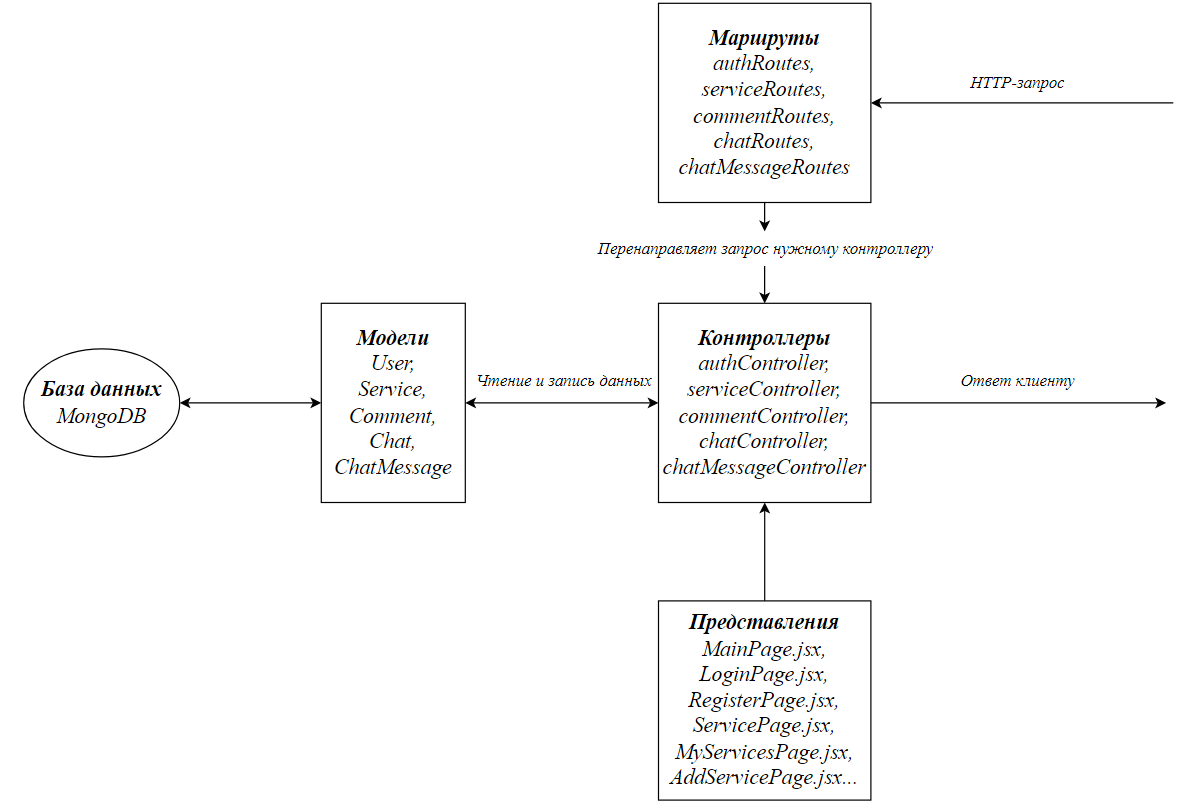


Рисунок 3.4 – Архитектура приложения

В этой схеме модель является независимым компонентом - любые изменения контроллера или представления никак не влияют на модель. Контроллер и представление являются относительно независимыми компонентами. Так, из представления можно обращаться к определённому контроллеру, а из контроллера генерировать представления, но при этом нередко их можно изменять независимо друг от друга.

Такое разграничение компонентов приложения позволяет реализовать концепцию разделение ответственности, при которой каждый компонент отвечает за свою строго очерченную сферу. В связи с чем легче построить работу над отдельными компонентами. И благодаря этому приложение легче разрабатывать, поддерживать и тестировать отдельные компоненты.

Концепция MVC разделяет данные, представление и обработку действий пользователя на компоненты:

• ***Модель / Model*** — предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние. При этом модель не содержит в себе информации о способах визуализации данных или форматах их представления, а также не взаимодействует с пользователем напрямую.

• ***Представление / View*** — отвечает за отображение информации (визуализацию). Одни и те же данные могут представляться различными способами и в различных форматах. Например, коллекцию объектов при помощи разных представлений можно представить на уровне пользовательского интерфейса как в табличном виде, так и списком; на уровне API можно экспортировать данные как в JSON, так в XML или XSLX.

• ***Контроллер / Controller*** — обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя. Как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация — проверяются права пользователя на выполнение действий или получение информации.

• ***Система маршрутизации*** как дополнительный компонент сопоставляет запросы с маршрутами и выбирает для обработки запросов определенный контроллер.

В общем случае, когда к приложению приходит запрос, система маршрутизации выбирает нужный контроллер для обработки запроса. Контроллер обрабатывает запрос. В процессе обработки он может обращаться к данным через модели и для рендеринга ответа использовать представления. Результат обработки контроллера отправляется в ответ клиенту. Ответ представляет html-страницу, которую пользователь видит в своем браузере.

В приложении определены роутеры, которые сопоставляются с адресами, например, "localhost:3002/". В файле index.js:

// http://localhost:3002

app.use('/api/auth', authRoutes)

app.use('/api/services, serviceRoutes)

app.use('/api/comments', commentRoutes)

app.use('/api/chat', chatRoutes)

app.use('/api/messages', messagesRoutes)

Контроллеры подключаются как стандартные модули, и затем функции контроллеров используются для обработки маршрутов. Кроме того, для упрощения логической организации маршруты, которые обрабатываются конкретными контроллерами authController, serviceController и другие организованы в отдельных роутерах – authRoutes, serviceRoutes … .

Однако в данном случае опять же стоит отметить, что внутри каждого роутера может быть определено множество маршрутов. В этом случае для упрощения управления маршрутами каждый роутер организуется и подключается в виде отдельного модуля. Для этого в проекте организована папка routes.

В папке routes соответственно хранятся данные роутеры. Для обработки запросов в нашем приложении используется Express. В нём определено ряд встроенных функций, и одной из таких является функция app.get(). Она обрабатывает GET-запросы протокола HTTP и позволяет связать маршруты с определенными обработчиками. Для этого первым параметром передается маршрут, а вторым - обработчик, который будет вызываться, если запрос к серверу соответствует данному маршруту.

Вся логика обработки запросов лежит в контроллерах. Для этого в каталоге приложения определена папка, которая называется controllers. В данной папке хранятся следующие контроллеры нашего приложения: authController, serviceController, commentController, chatController, chatMessageController. Каждый файл представляет собой отдельный контроллер, который содержит набор функций-обработчиков маршрутов.

Так в контроллере serviceController содержатся следующие функции-обработчики: createService(), getAll(), sortedServices(), getById(), getMyServices(), removeService(), updateService() и др. Чтобы получить список всех услуг, по адресу "localhost:3002/" определен маршрут и обработчик, представленный функцией контроллера getAll():

router.get('/', getAll)

Также нужно подключить конкретную функцию-обработчик, например, import { getAll} from '../controllers/serviceController.js'

Когда фреймворк Express получает запрос, этот запрос передается в конвейер обработки. Конвейер состоит из набора компонентов или middleware, которые получают данные запроса и решают, как его обрабатывать.

Здесь конвейер обработки состоял из вызовов router.get(), которые сравнивали запрошенный адрес с маршрутом, и если между адресом и маршрутом было соответствие, то данный запрос обрабатывался методом router.get().

При переходе по адресу / будет срабатывать метод router.get, который отправит пользователю файл MainPage.jsx.

При обработке запросов фреймворк Express опирается на систему маршрутизации. В приложении определяются маршруты, а также обработчики этих маршрутов. Если запрос соответствует определенному маршруту, то вызывается для обработки запроса соответствующий обработчик.

Для обработки данных по определенному маршруту можно использовать ряд функций, в частности: use, get, post, put, delete. Так, данные удаляются с помощью метода DELETE, поэтому для обработки определяем функцию:

router.delete('/:id', checkAuth, removeService)

Первый параметр функции - адрес, на который идет отправка - /services/:id, второй параметр – небольшой middleware, который проверят авторизирован ли пользователь, третий параметр – функция-обработчик.

Для получения отправленного id услуги используем выражение типа request.params.id, где request.params инкапсулирует передаваемые параметры, а id - ключ данных, который соответствует значению id соответствующей услуги для удаления. Функция-обработчик removeService контроллера serviceController:

export const removeService = async (req, res) => {

    try {

        const service = await Service.findByIdAndDelete(req.params.id)

        if (!service) return res.json({ message: 'Такой услуги не существует' })

        await User.findByIdAndUpdate(req.userId, {

            $pull: { services: req.params.id },

        })

        res.json({ message: 'Услуга была удалена.' })

    } catch (error) {

        res.json({ message: 'Что-то пошло не так.' })

    }

}

Таким образом, за счет выноса логики обработки маршрутов и организации маршрутов в роутеры в отдельные модули общий код приложения стал проще и яснее.

В данном приложении для управления между состояниями данных и представлениями пользовательского интерфейса используется React и Redux toolkit. Redux Toolkit - это пакет, облегчающий работу с Redux. Он был разработан для решения трех главных проблем:

* слишком сложная настройка хранилища (store);
* для того чтобы заставить Redux делать что-то полезное, приходится использовать дополнительные пакеты;
* слишком много шаблонного кода (boilerplate).

Redux Toolkit предоставляет инструменты для настройки хранилища и выполнения наиболее распространенных операций, а также содержит полезные утилиты, позволяющие упростить код.

Разработка любого Redux-приложения предполагает создание и настройку хранилища. configureStore() позволит упростить процесс настройки:

import { configureStore } from '@reduxjs/toolkit'

import authSlice from './features/auth/authSlice'

import serviceSlice from './features/service/serviceSlice'

import commentSlice from './features/comment/commentSlice'

export const store = configureStore({

    reducer: {

        auth: authSlice,

        service: serviceSlice,

        comment: commentSlice,

    },

})

Редукторы (reducers) - это самая важная часть Redux. Как правило, редуктор отвечает за:

* определение характера ответа на основе поля type объекта операции;
* обновление состояния посредством копирования части состояния и модификации этой копии.

createSlice() автоматически генерирует типы и создателей операции на основе переданного названия редуктора. Создателя операции определяют action creators. Параметры, необходимые для выполнения операции, обычно, помещаются в поле payload. createSlice() анализирует функции, определенные в поле reducers (в нашем случае createComment), создает редуктор для каждого случая и генерирует создателя, использующего название редуктора в качестве типа операции. Таким образом, редуктор createComment становится типом операции comment/createComment, а создатель createComment() возвращает операцию с этим типом.

Хранилище Redux ничего не знает об асинхронной логике. Оно знает только о том, как синхронно отправлять операции, обновлять состояние посредством вызова корневого редуктора и уведомлять UI об изменениях. Любые асинхронные операции должны выполняться за пределами хранилища.

Существует несколько посредников для реализации асинхронности в Redux. Рекомендуемым является Redux Thunk. Он прекрасно подходит для большинства случаев, а использование синтаксиса async/await делает его еще лучше. createAsyncThunk() упрощает процесс выполнения асинхронных запросов - мы передаем ему строку для префикса типа операции и колбек создателя полезной нагрузки (payload), выполняющего реальную асинхронную логику и возвращающего промис с результатом. createAsyncThunk() возвращает преобразователя, который заботится об отправке правильных операций на основе возвращенного промиса, и типы операции, которые можно обработать в редукторах.

Таким образом, создание комментария createSlice вместе с action creator и createAsyncThunk будет выглядеть следующим образом:

const initialState = {

    comments: [],

    loading: false,

}

// Создание преобразователя

export const createComment = createAsyncThunk(

    'comment/createComment',

    async ({ serviceId, comment, author }) => {

        try {

            console.log('author', author)

            const { data } = await axios.post(`/comments/${serviceId}`, {

                serviceId,

                comment,

                author,

            })

            return data

        } catch (error) {

            console.log(error)

        }

    },

)

// Определяем редуктор и комментарии с помощью `createSlice()` и Обрабатываем операции в редукторе

export const commentSlice = createSlice({

    name: 'comment',

    initialState,

    reducers: {},

    extraReducers: {

        // Создание комментария

        [createComment.pending]: (state) => {

            state.loading = true

        },

        [createComment.fulfilled]: (state, action) => {

            state.loading = false

// Добавляем комментарий в массив состояния

            state.comments.push(action.payload)

        },

    },

})

export default commentSlice.reducer

// Позже в PostPage.jsx, отправляем `thunk`

dispatch(createComment({ serviceId, comment, author }))

// На сервере в commentController принимаем параметры из request.body

export const createComment = async (req, res) => {

        const { serviceId, comment, author } = req.body

…

        const newComment = new Comment({ comment, authorName, author })

        await newComment.save()

        await Service.findByIdAndUpdate(serviceId, {$push: { comments: newComment.\_id },})

…

Для интеграции MongoDB с проектом используется Object Document Mapper (ODM) Mongoose для создания схем и моделей данных приложения. В приложении определены следующие модели: User, Service, Comment, Chat, ChatMessage. Mongoose выступает в качестве внешнего интерфейса для MongoDB, базы данных NoSQL, использующей документно-ориентированную модель данных.

Модели определяются с помощью Schema. Схема позволяет определять поля, хранящиеся в каждом документе, вместе с их требованиями к проверке и значениями по умолчанию. Также можно определить статические вспомогательные методы, для того, чтобы упростить работу с типами данных.

Например, модель комментария в нашем приложении будет:

import mongoose from 'mongoose'

const CommentSchema = new mongoose.Schema(

{

comment: { type: String, required: true },

authorName: {type: String, required: true},

author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

},

    { timestamps: true },

)

export default mongoose.model('Comment', CommentSchema)

Затем схемы «компилируются» в модели с помощью mongoose.model() метода. Метод save() сохраняет текущий обьект в базу данных. Когда у нас есть модель, мы можем использовать ее для поиска, создания, обновления и удаления объектов данного типа.

Организация взаимодействия приложения с сервером посредством протокола HTTP осуществляется за счет принципа REST (Representational state transfer).

Другой отличительной чертой REST является то, что API-интерфейсы REST фокусируются на ресурсах (то есть вещах, а не действиях) и способах доступа к ресурсам. Ресурсы, как правило, являются разными типами информации. Мы получаем доступ к ресурсам через URL (Uniform Resource Locators), так же как переход к URL-адресу в вашем браузере позволяет подключиться к информационному ресурсу. URL-адреса сопровождаются методом, который указывает, как вы хотите взаимодействовать с ресурсом

REST работает на уровне HTTP и использует нижеприведённые методы для работы с ресурсами на сервере:

POST — для создания ресурса;

GET — для его получения;

PUT — для его обновления;

DELETE — для его удаления.

Разумеется, этот перечень сопровождается такими опциями, как обработка ошибок в запросе, разграничение доступа к данным и валидация входящих данных, в общем, всеми возможными проверками, которые сервер выполняет перед тем, как выполнить желание клиента.

В данном случае, например, необходимо иметь возможность добавлять новые услуги, просматривать все существующие услуги и конкретную услугу пользователя.

Эти операции соответствуют следующим маршрутам:

• POST /services— создание новой услуги;

• GET /services/:id — получение одной услуги;

• GET /services — получение всех услуг.

Что происходит, когда клиент хочет получить подробную информацию о услуге с идентификатором 6389fabd6ab54b47891c88db из списка всех услуг с помощью REST API веб-приложения агрегатора? Клиент должен обмениваться данными с сервером, на котором размещен API, используя протокол HTTP, как показано на рисунке 3.5

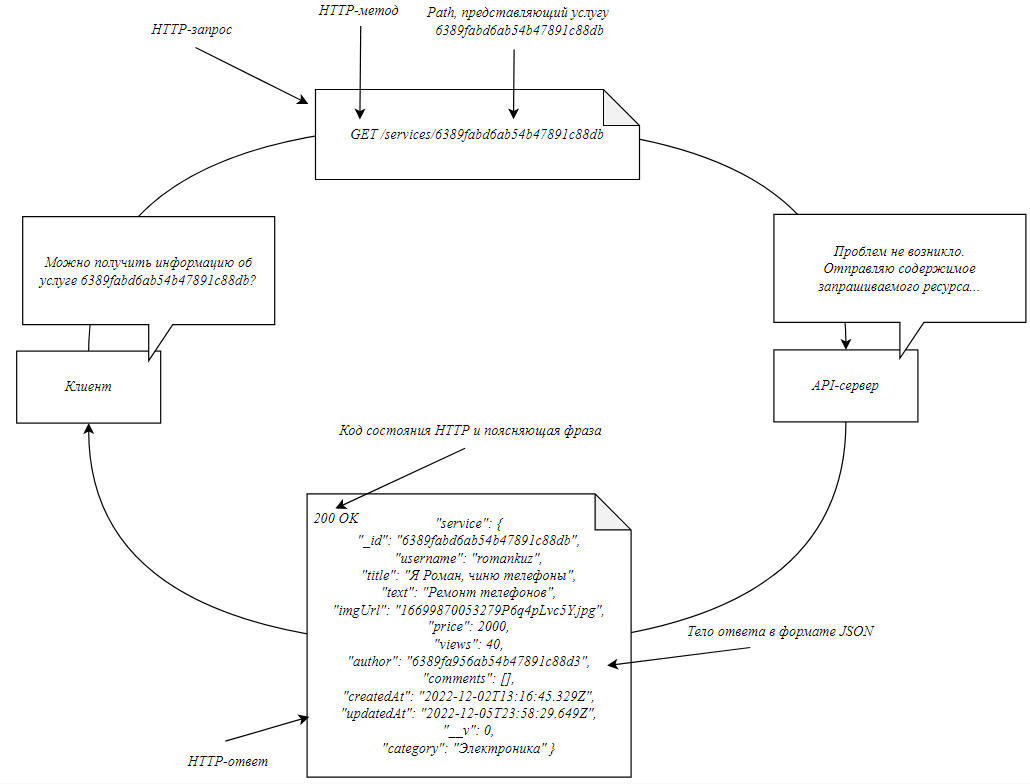


Рисунок 3.5 - Вызов REST API с использованием протокола HTTP

Поскольку получение информации об услуге представлено GET /services/:id, клиент отправляет HTTP-запрос GET /services/6389fabd6ab54b47891c88db на сервер API веб-приложения. В ответ сервер возвращает ответ, 200 OK, за которым следует информация о запрашиваемой услуге. (Обратите внимание, что мы упростили этот обмен данными по протоколу HTTP, чтобы сосредоточиться только на тех элементах, которые важны для нас).

Запрос состоит из HTTP-метода GET и пути /services/6389fabd6ab54b47891c88db. Путь – это адрес, идентифицирующий ресурс на сервере; в данном случае услуга 6389fabd6ab54b47891c88db находится в services. HTTP-метод указывает на то, что потребитель хочет сделать с этим ресурсом: GET означает, что он хочет получить ресурс. С функциональной точки зрения такой запрос означает что-то вроде: «Привет, могу ли я получить информацию об услуге с идентификатором 6389fabd6ab54b47891c88db»? Но с точки зрения протокола HTTP это означает: «Привет, могу ли я определить ресурс по пути /services/6389fabd6ab54b47891c88db»?

Первая часть ответа состоит из кода состояния HTTP 200 и поясняющей фразы OK. Код состояния сообщает нам, как прошла обработка запроса. Благодаря поясняющей фразе мы можем догадаться, что код состояния HTTP 200 означает, что все прошло нормально. Вторая часть ответа называется телом ответа. Она содержит содержимое ресурса, идентифицированного путем в запросе, который в данном случае представляет собой информацию об услуге 6389fabd6ab54b47891c88db в виде данных в формате JSON. С функциональной точки зрения ответ, возвращаемый сервером API, в основном означает: «Конечно, вот информация о запрашиваемой услуге». С точки зрения HTTP имеется в виду: «Проблем не возникло. Отправляю содержимое запрашиваемого ресурса...».

Преимущества использования данного подхода заключаются в следующем:

• Независимость сервера от клиента — серверы и клиенты могут быть мгновенно заменены другими независимо друг от друга, так как интерфейс между ними не меняется. Сервер не хранит состояний клиента.

• Уникальность адресов ресурсов — каждая единица данных (любой степени вложенности) имеет свой собственный уникальный URL, который, по сути, целиком является однозначным идентификатором ресурса.

• Независимость формата хранения данных от формата их передачи — сервер может поддерживать несколько различных форматов для передачи одних и тех же данных (JSON, XML и т. д.), но хранит данные в своем внутреннем формате, независимо от поддерживаемых.

• Присутствие в ответе всех необходимых метаданных — помимо самих данных сервер должен возвращать детали обработки запроса, например, сообщения об ошибках, различные свойства ресурса, необходимые для дальнейшей работы с ним, например, общее число записей в коллекции для правильного отображения постраничной навигации

**4. Проектно-технологическая часть**

* 1. **Проектирование начального и тестового наполнения базы данных. Порядок развертывания системы.**

Содержимое базы данных является основным источником информации для данного веб-приложения.

На странице регистрации с помощью специальной формы вручную можно добавить информацию о новом пользователе в базу данных, на странице добавления услуги — информацию о услуге, на странице уже существующей услуги — создать чат с исполнителем или оставить отзыв об услуге, а на странице своего профиля — изменить личные данные.

Формы, представленные на рисунках 4.1-4.5, позволяют создавать аккаунт, добавлять, изменять или удалять услуги, создавать чаты для коммуникации с исполнителями, добавлять комментарии (отзывы) об услугах без использования других сервисов работы с базой данных и помощи разработчиков.

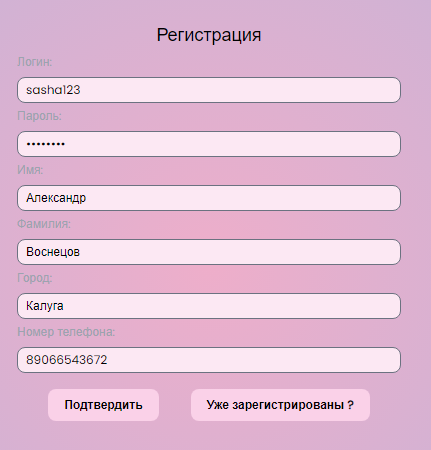
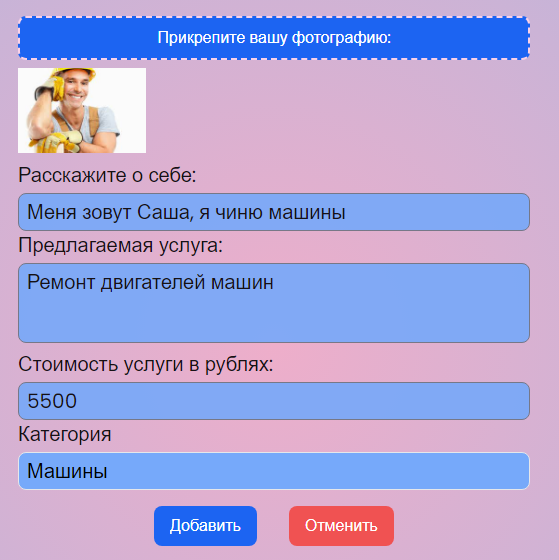


Рисунок 4.1 – Создание аккаунта



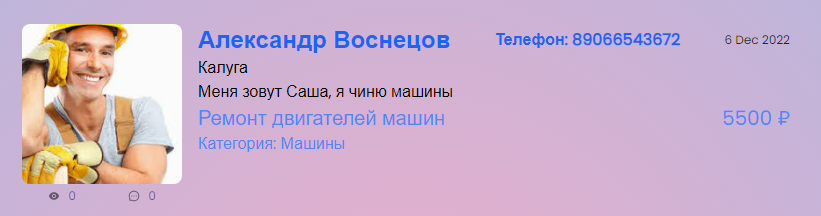


Рисунок 4.2 – Создание услуги

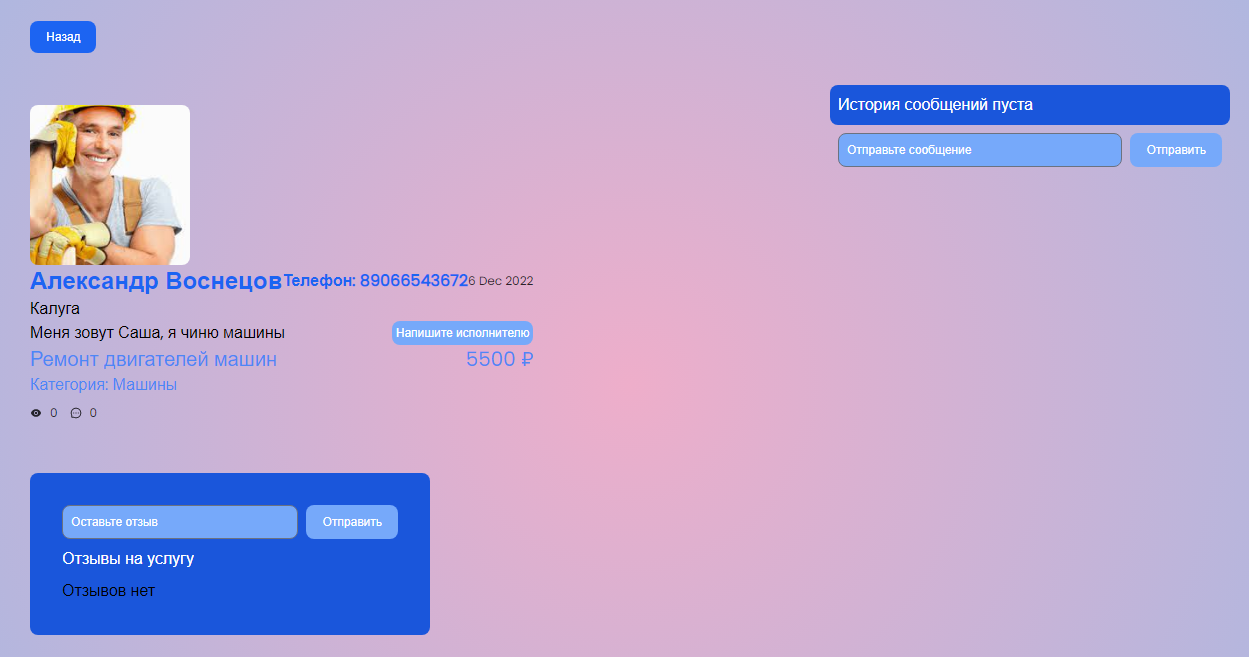


Рисунок 4.3 – Создание чата с исполнителем

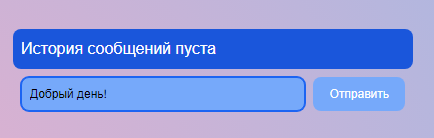
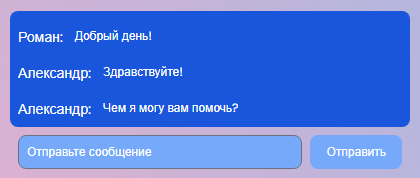
 

Рисунок 4.4 – Создание сообщений в чате

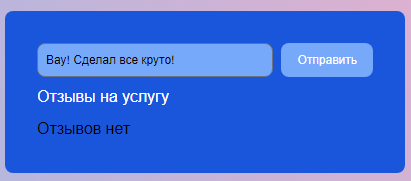
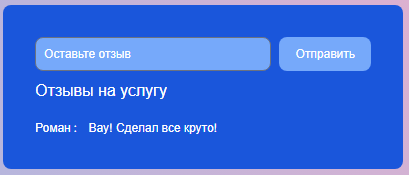
 

Рисунок 4.5 – Создание отзыва об услуге

Самая простая процедура развертывания сайта состоит из следующих этапов:

1. Перемещение файлов на сервер.
2. Запуск команды npm install для установки новых зависимостей.
3. Запуск node при помощи команды npm run dev. Также можно развернуть приложение на сервер PaaS-провайдера (Heroku).
4. Данная система тестировалась в следующих браузерах: Google Chrome, Firefox, Microsoft Edge.
   1. **Тестирование и отладка макета рабочей программы.**

Тестирование веб-приложения. Для начала необходимо запустить веб-приложение агрегатор услуг по ремонту. После запуска веб-приложения, открывается веб-страница с пользовательским интерфейсом (рисунок 4.6).

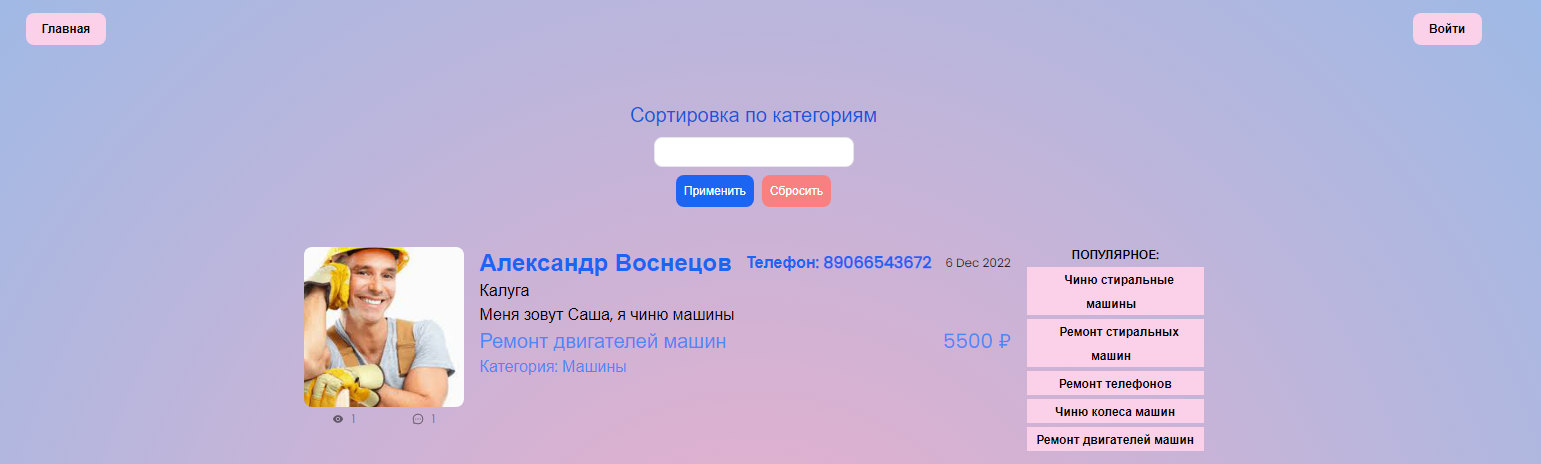
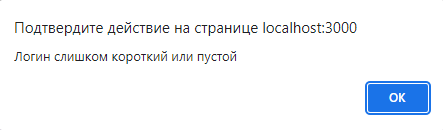


Рисунок 4.6 – Пример пользовательского интерфейса

Одной из самых первоначальных и главных задач веб-приложения является авторизация и регистрация пользователей. Для успешного выполнения этих задач, необходимо проверить правильность введенных данных. Например, длина пароля должна быть не менее 8 символов, а логина — не менее 3-ех. Для этого приведем пример ввода с ошибкой и проверим, как отреагирует на такую ситуацию веб-приложение (рис. 4.7).



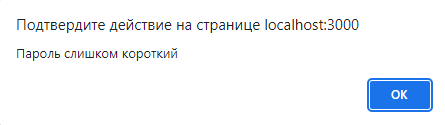


Рисунок 4.7 – Обработка ситуаций с некорректным вводом при регистрации

Следовательно, можно сделать вывод, что при создании пользователя, при вводе некорректных данных происходит успешная обработка исключительной ситуации.

Кроме того, при попытки входа под несуществующими данными, или при допущении ошибки в логине или пароле, приложение должно сообщить об этом пользователю, проверим это. Для этого введем несуществующий логин или пароль в нашей базе данных и посмотрим, что произойдет (рис. 4.8).

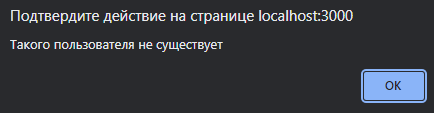


Рисунок 4.8 – Обработка несуществующего логина или пароля

Таким образом, веб-приложение выдает предупреждение о том, что пользователя с таким логином и паролем не существует.

Кроме того, может возникнуть ситуация, из-за которой соединение с базой данных не удастся произвести. Если нет соединения с базой данных – то корректная работа с сайтом невозможна. Следовательно, нужна проверка на успешное установление соединения с базой данных (рис. 4.9)

if (!services.length) {

        return (

            <div className='text-xl text-center text-white py-10'>

                Пока что нет услуг.

            </div>

        )

    }

Рисунок 4.9 – Проверка соединения с БД

В случае отсутствия соединения с БД, длина массива с услугами будет нулевой, так как от сервера не пришло ответа из-за отсутствия соединения с БД, поэтому дальнейшая работа с сайтом будет невозможна, однако пользователю будет показано, что на данный момент услуг нет (рис. 4.10).

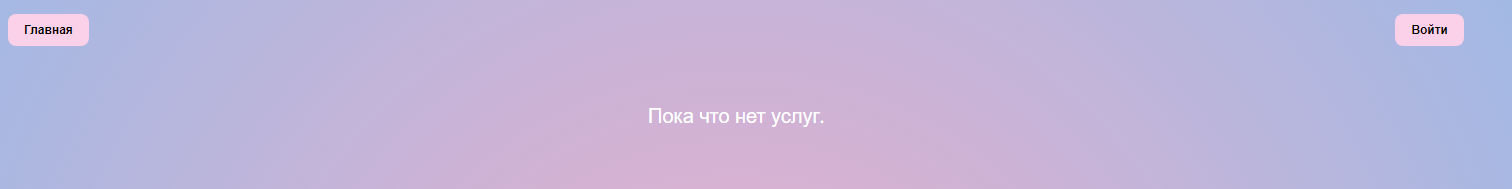


Рисунок 4.10 – Сообщение об ошибке

Так как ресурс является открытым, то появляется риск создания спаммерских или нецензурных комментариев под услугами. Чтобы этого избежать, используется фильтр в виде регулярного выражения, который будет блокировать создание нежелательных комментариев. Так, при попытке оставить нецензурный комментарий, пользователю отобразится предупреждение, и комментарий не будет создан (рис. 4.11).

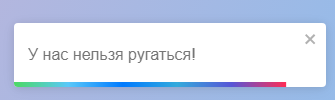


Рисунок 4.11 – Попытка добавления нецензурного отзыва

Основополагающую роль каждого веб-приложения играет пользовательский интерфейс. Его работа должна быть адаптивной под разную диагональ устройства, на котором будет открыто приложение (рис. 4.12).



Рисунок 4.12 – Пример адаптивности интерфейса под маленькое разрешение экрана

Таким образом, после проверки некорректной авторизации и регистрации, нежелательных комментариев, ситуации с отсутствием соединения с БД, а также проверки адаптивности под маленькое разрешение, можно сделать вывод, что веб-приложение работает корректно, а также осуществляет успешную обработку исключительных ситуаций.

**4.3. Разработка руководства пользователя**

Порядок работы:

После запуска открывается страница с пользовательским интерфейсом (рисунок 4.13).

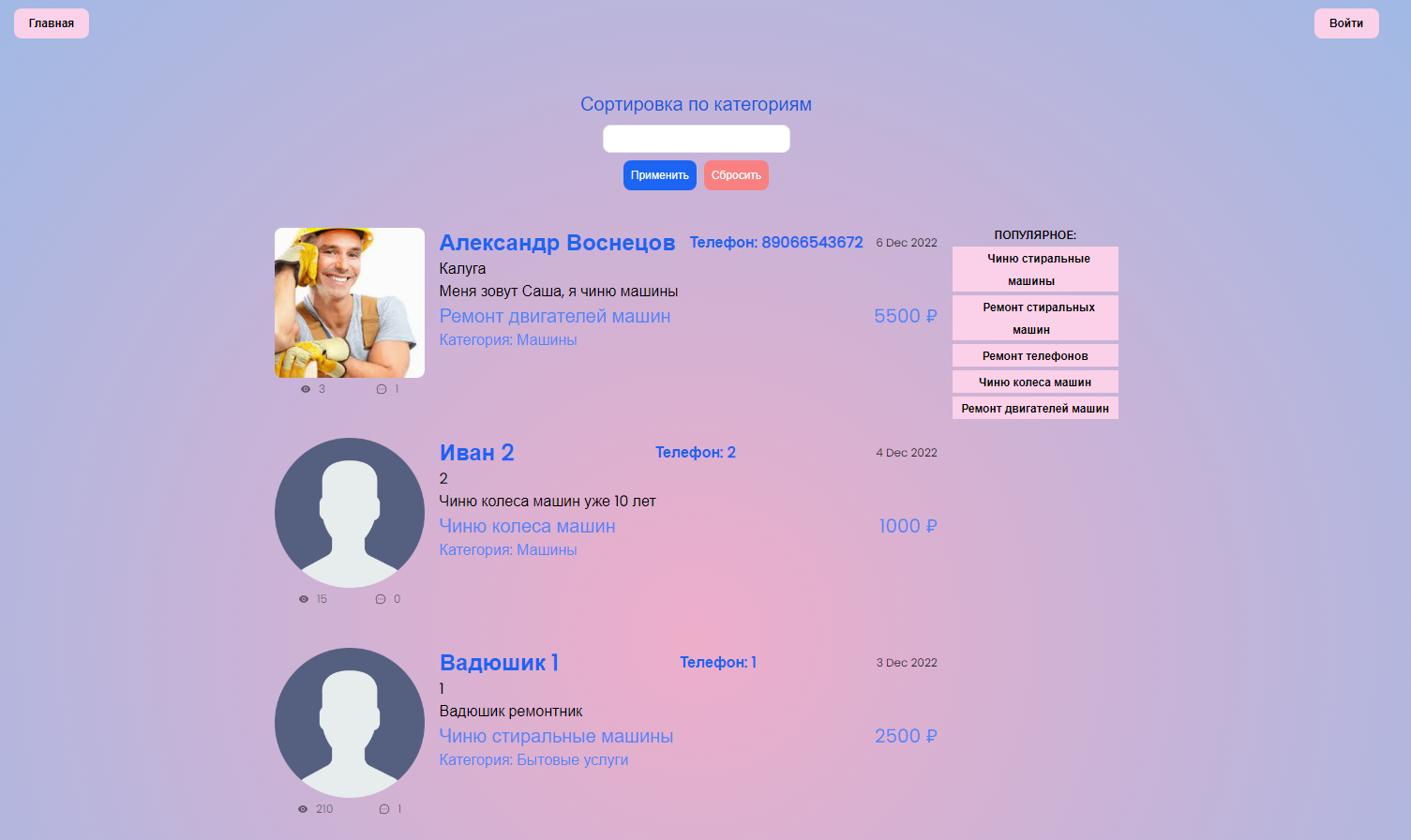


Рисунок 4.13 – Веб-приложение после открытия

Просматривать все существующие услуги на сайте (рис. 4.13), а также открыть конкретную услугу (рис. 4.14), может даже неавторизированный пользователь (гость).

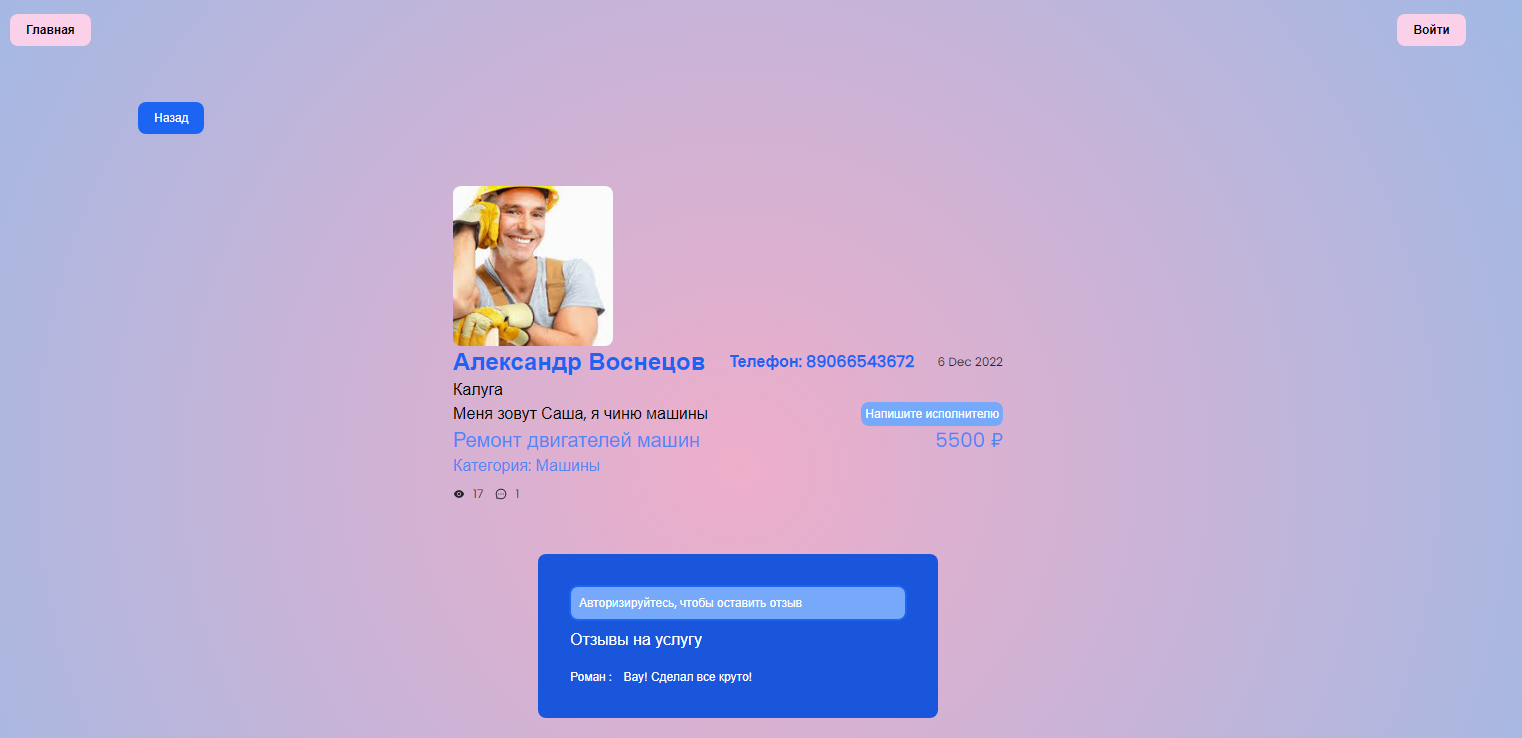


Рисунок 4.14 – Страница с услугой у гостя

Далее следует разделить пользователей данного сайта на 2 категории: исполнитель и заказчик. Однако это разделение является условным, так как абсолютно любой пользователь может быть как исполнителем, так и заказчиком услуги.

Чтобы появилась возможность связываться с исполнителями, а также добавлять услуги, необходимо пройти процессы регистрации и авторизации. При входе в систему пользователю надо ввести логин и пароль (Рисунок 4.15). Если ранее пользователь не был зарегистрирован, он может заполнить соответствующую форму для регистрации (Рисунок 4.16). При некорректном вводе данных пользователю выводится сообщение об ошибке (Рисунок 4.17). Например, указан неправильный логин (менее 3-ех символов) или пароль (менее 8 символов) или при входе указан незарегистрированный логин или неправильный пароль. Сообщение об ошибке появляется в правом нижнем углу.



Рисунок 4.15 - Форма авторизации



Рисунок 4.16 - Форма регистрации

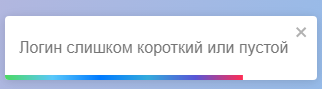
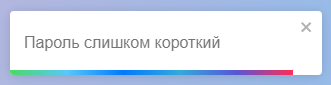
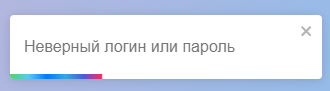
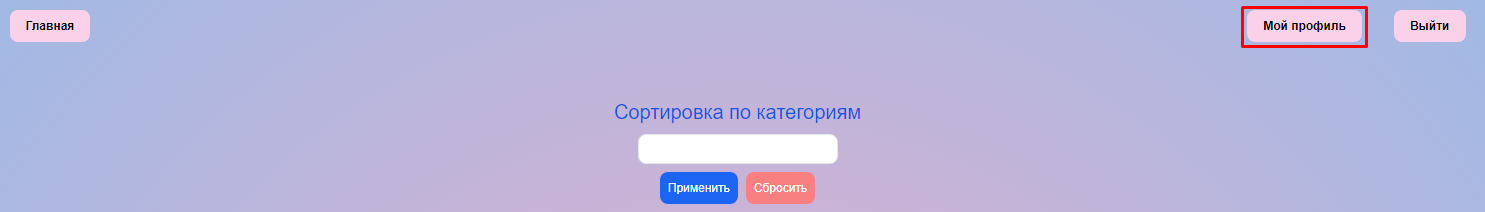
  

Рисунок 4.17 – Сообщения об ошибке при авторизации или регистрации

После регистрации и авторизации исполнитель может добавить услугу. Для этого надо нажать на кнопку «Мой профиль», а затем «Добавить услугу» (рис. 4.18).



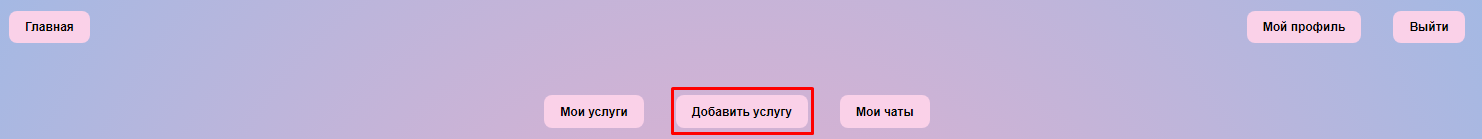


Рисунок 4.18 – Путь для добавления услуги

После этого исполнитель попадет на страницу добавления услуги (рис. 4.19).

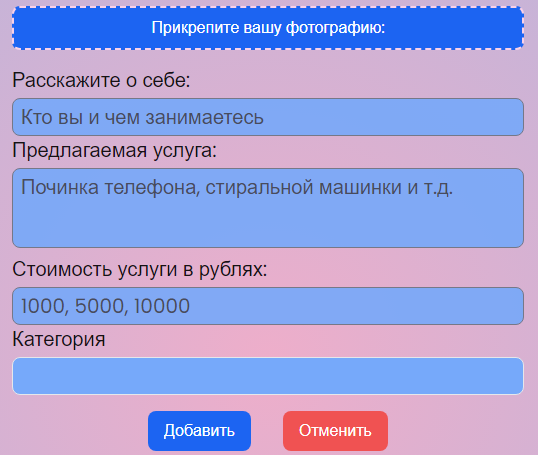
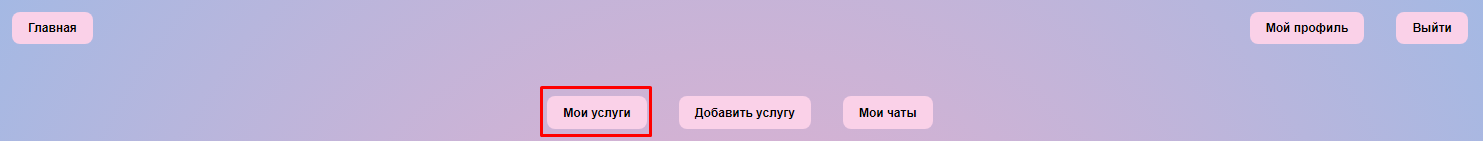


Рисунок 4.19 – Форма для добавления услуги

Кроме того, он может просмотреть все свои услуги (4.20).



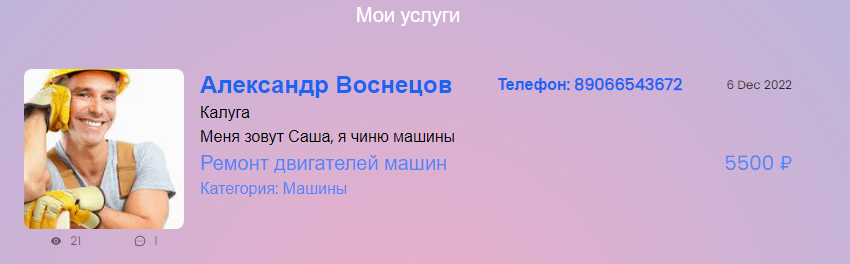


Рисунок 4.20 – Просмотр всех услуг исполнителя

Если исполнитель перейдет на свою конкретную услугу, нажав на нее, то откроется страница данной услуги и отзывы к ней, а также появится возможность редактирования или удаления услуги (рис. 4.21).

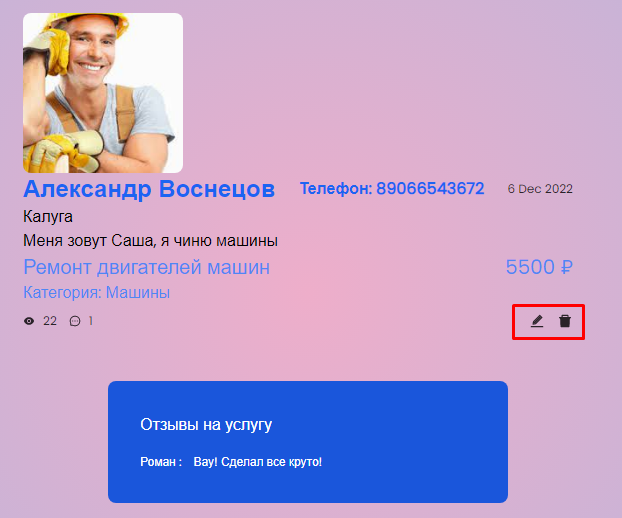


Рисунок 4.21 – Возможность редактирования или удаления своей услуги

Заказчик, в свою очередь, перейдя на конкретную услугу, может создать чат с исполнителем или оставить отзыв к ней (рис. 4.22).

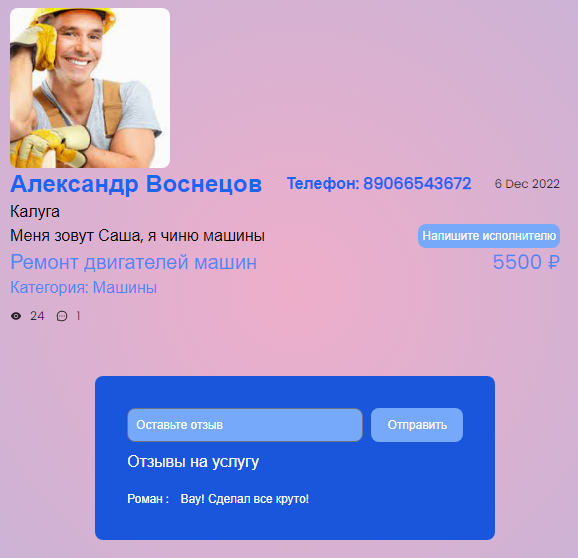


Рисунок 4.22 – Возможность создания чата с исполнителем или добавления отзыва об услуге

При нажатии на кнопку «Напишите исполнителю» создастся чат, там можно общаться и договариваться по поводу конкретной услуги (4.23)



Рисунок 4.23 – Создание чата с исполнителем

И исполнитель, и заказчик могут посмотреть все свои чаты на странице своего профиля, нажав на кнопку «Мои чаты» (рис. 4.24). Кроме того, на этой странице можно изменять информацию о пользователе (имя, фамилию, город и номер телефона).

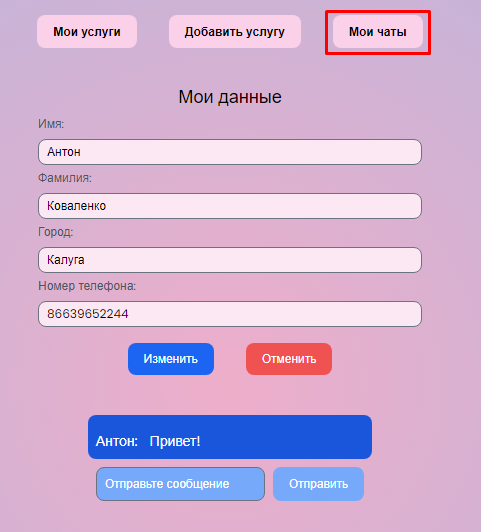


Рисунок 4.24 – Отображение всех чатов пользователя и возможность изменения данных пользователя

Администратор (модератор) может удалять, изменять любые услуги, а также удалять отзывы к ним (рис. 4.25).

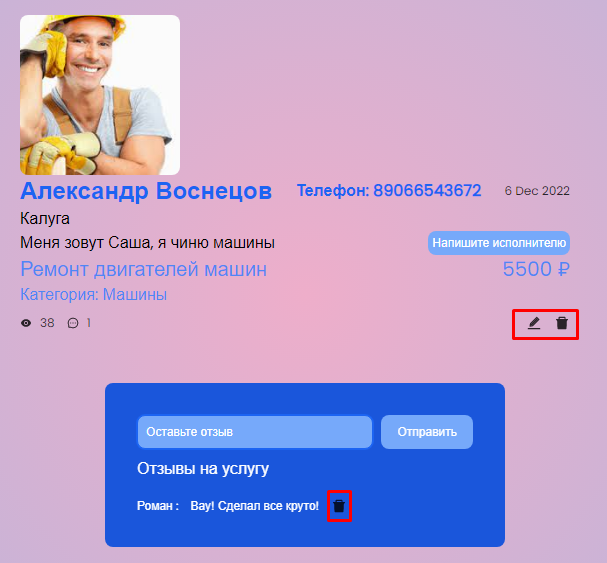


Рисунок 4.25 – Возможность удаления, изменения услуг и удаления отзывов у администратора

**Заключение**

При выполнении курсовой работы на тему «веб-приложение агрегатор услуг по ремонту» была исследована и описана предметная область, проведен анализ аналогов данной системы, а также выбор инструментов и платформы для разработки.

Проведен анализ объектов автоматизации и разработаны методы решения технических задач. Также была разработана и реализована структура базы данных, архитектура приложения, серверная часть приложения и веб-интерфейс системы. База данных была наполнена тестовыми данными.

Результатом данной курсовой работы является рабочий сайт-агрегатор услуг по ремонту.

**Список использованных источников**

1. Арно Лоре Проектирование веб-API / Пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2020.– 440 с

2. Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. - СПб.: Питер, 2013. - 240 c.

3. Янг А., Мек Б., Кантелон М. Я Node.js в действии. 2-е изд. — СПб.: Питер, 2018. — 432 с.: ил. — (Серия «Для профессионалов»).

4. https://mongoosejs.com/docs/queries.html

5. Ревунков Г.И., Ковалева Н. А., Силантьева Е. Ю. Проектирование баз данных. [Электронный ресурс] – МГТУ им. Н.Э.Баумана, 2018. – 48 с. Режим доступа: <https://e.lanbook.com/reader/book/103499/>

6. <https://nodeguide.ru>

7. Руководство по программированию на JavaScript [Электронный ресурс] – режим доступа: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express\_Nodejs/skeleton\_website

8. Изучаем Node.js [Электронный ресурс] – режим доступа: https://metanit.com/web/nodejs/7.1.php 8. Learning JavaScript Design Patterns — Addy Osmani 2020 – 439c

9. https://reactdev.ru/libs/redux-toolkit/