МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация 1-40 01 01 10 Программное обеспечение информационных

технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Программное средство «Расписание транспорта»

Выполнил студент Коржова Валерия Сергеевна

(Ф.И.О.)

Руководитель проекта асс. Дубовк М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Дубовк М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Дубовк М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2022

Содержание

[Введение 3](#_Toc104210063)

[1 Постановка задачи 4](#_Toc104210064)

[1.1 Обзор аналогов 4](#_Toc104210065)

[1.2 Обзор средств разработки 8](#_Toc104210066)

[**2 Проектирование** 10](#_Toc104210067)

[2.1 Проектирование базы данных 10](#_Toc104210068)

[2.1.1 Индексы 13](#_Toc104210069)

[**2.2 Проектирование программного средства** 14](#_Toc104210070)

[3 Реализация программного средства 16](#_Toc104210071)

[3.1 Разработка серверной части 16](#_Toc104210072)

[4 Руководство пользователя 26](#_Toc104210073)

[5 Тестирование работоспособности 30](#_Toc104210074)

[5.1 Тестирование приложения 30](#_Toc104210075)

[Заключение 31](#_Toc104210076)

[Список используемых источников 32](#_Toc104210077)

[ПРИЛОЖЕНИЕ А 33](#_Toc104210078)

[ПРИЛОЖЕНИЕ Б 36](#_Toc104210079)

[ПРИЛОЖЕНИЕ В 37](#_Toc104210080)

Введение

Целью курсового проекта является разработка веб-приложения для сервиса технического обслуживания. Данное программное средство может быть использовано компаниями занимающимися авто перевозками, компаниями имеющие собственный парк авто либо иного рода транспортного средства, владельцами авто. Приложение благодаря интуитивно понятному пользовательскому интерфейсу позволяет самостоятельно выбрать компанию предоставляющую собственный набор технических услуг, предварительно ознакомившись с подробной информацией по каждой из них, не выходя из дома.

В связи с тем, что самым важным ресурсом современного мира становится информация, вполне закономерно развитие технологий в направлении хранения и управления данными.

Программное средство представляет собой веб-приложение, основанное на MVC архитектуре. Эта архитектура позволяет создать максимально независимый программный интерфейс для взаимодействия сервисов приложения.

Для серверной части приложения использовалась платформа Node.js с Фреймворком Express, для работы с базой данных использовалась ORM Sequelize. Express – это гибкий и легко расширяемый 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 надежное MVC приложение можно быстро и легко. Sequelize – это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в БД и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами. Причем Sequelize может работать с рядом СУБД - MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, MongoDB. В данном приложения использовалась СУБД PostgreSQL. [1]

Функционально необходимо разработать программное средство, предоставляющее удобный функционал для выбора, сравнения, заказа подходящего сервиса, которое взаимодействует с базой данных. Также была использована технология взаимодействия с почтой клиентов.

Таким образом при разработке приложения были использованы:

* PostgreSQL
* Node.js
* EJS

1. Аналитический обзор литературы по теме проекта

Главная задачей курсового проекта является разработка Web- приложения, позволяющего пользователю, с ролью владельца техники, просматривать и находить информацию техническом обслуживании предоставляемым разными компаниями. А также предоставление сервиса от лица компании, для пользователя с ролью владельца компании. Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных. Отображение, бизнес логика и хранилище данных должны быть максимально независимы друг от друга для возможности расширения. В данном курсовом проекте требовалось реализовать следующие задачи:

* авторизация и регистрация пользователя;
* создание записей на обслуживание;
* создание компаний, предоставляющих обслуживание;
* пользователи имеют возможность оставлять комментарии по обслуживанию;
* удаление пользователей.
  1. Обзор аналогов

В наши дни множество подобных информационных систем можно найти на сайтах с нашей тематикой. Перед тем как приступать к разработке своей системы можно изучить уже готовые решения и посмотреть, что по мнению других разработчиков является наиболее необходимым для пользователей, чтобы приложение действительно было полезным. Первым аналогом будет рассмотрено веб-приложение «Multiservice».

На рисунке 1.1 представлен внешний вид каталога фильмов и сериалов веб-приложения «Multiservice».

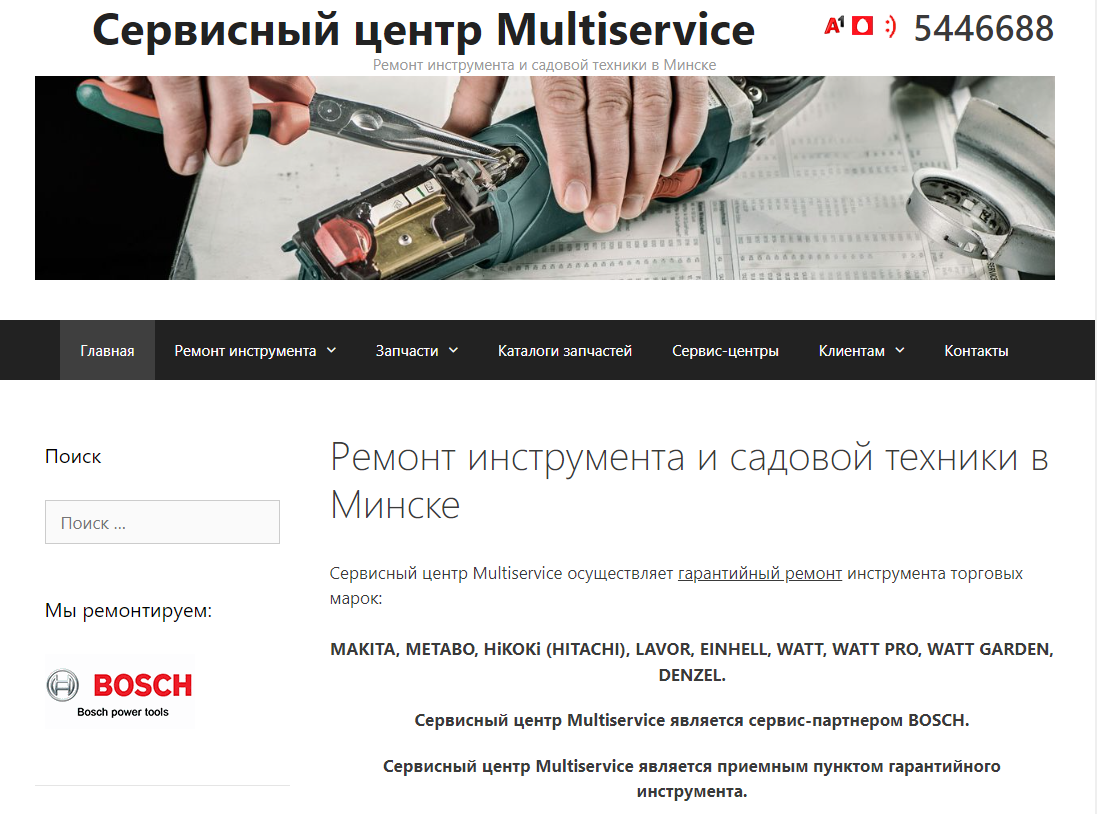


Рисунок 1.1 – Каталог сервиса сайта «https://mservice.by/»

По главной странице каталога мы можем получить информацию о том, какую информацию необходимо показывать пользователю, который просматривает каталог сервиса. На навигационной панели находится блок с поиском, а также блоки с выбором сервиса, сервис центра, блок с контактной информацией. На этой странице добавлена информация об марках производителей техники для которых поддерживается сервис: MAKITA, METABO, Hikoki (HITACHI), LAVOR, EINHELL, WATT, WATT PRO, WATT GARDEN, DENZEL. Также на странице добавлена информация об компаниях партнерах (BOSCH). На странице представлен более детальный список предоставляемых услуг 1.2.

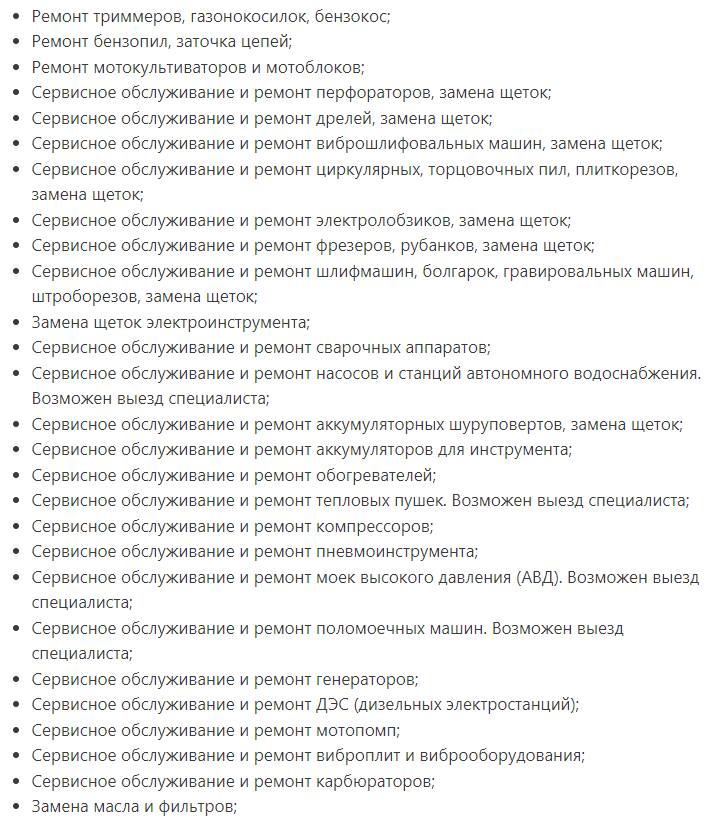


Рисунок 1.2 – Страница услуг сайта «https://mservice.by/»

Как и в любом сайте о предоставлении сервиса на данном сайте имеется компонент поиска, это необходимый атрибут для любой информационной системы данной сферы. С помощью поиска пользователь может получить набор услуг, запчастей и сервис центров для введённой поисковой строки.

Страница каталога сервисов и запчастей имеет краткую информацию о соответствующем объекте. Каждый блок сервиса содержит: название сервиса, список поддерживаемых производителей, краткое описание. При нажатии на название произойдёт переход на детальную страницу о услуге или запчасти.

Далее необходимо рассмотреть детальную страницу услуги или запчасти, для того чтобы определить наиболее важные данные которые необходимо предоставить пользователю. Данная страница представлена на рисунке 1.3.

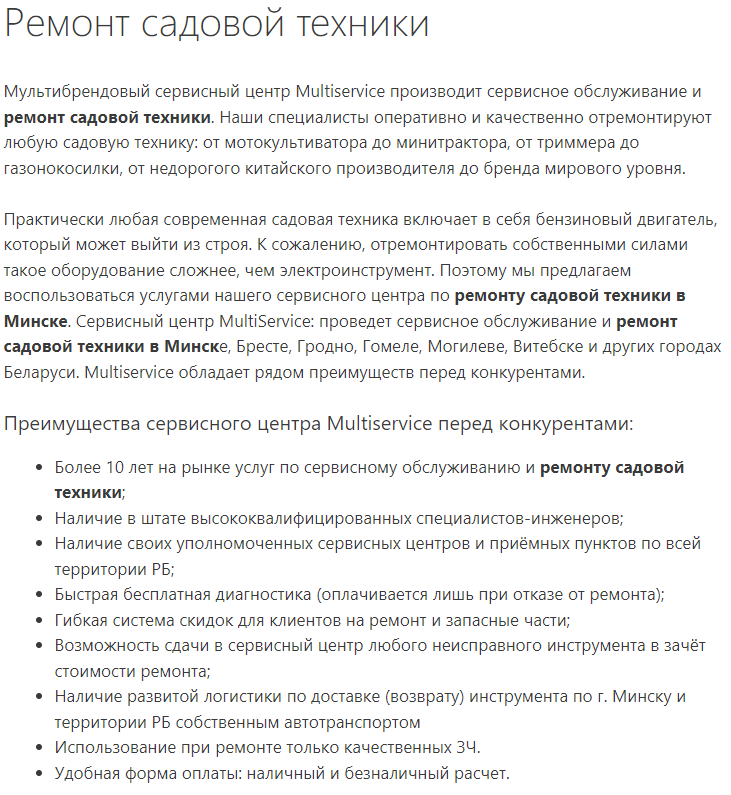


Рисунок 1.3 – Детальная страница услуги сайта «https://mservice.by/»

На рисунке видны основная информация услуги, которые необходимы пользователю, а именно: информация об категории ремонтируемой техники, название, описание проблемы ремонта данной категории техники собственными силами, преимущества использования данного сервиса начиная от имеющегося опыта, квалифицированного персонала, наличием качественной диагностики и заканчивая наличием системы скидок, возможности более выгодного использования сервиса и удобной системой оплаты (наличный и безналичный). Также есть компонент «Отправьте нам сообщение» в котором есть сообщение говорящие о предоставлении помощи по любым вопросам. приведён список из 10 актёров и кнопка для просмотра всех актёров. Также на детальной странице можно посмотреть список услуг которые часто заказывают совместно. После изучения других аналогов я могу сделать вывод о том, что рассмотренный сайт предоставляет наиболее подробную информацию о литературе и содержит все основные функции.

Главными преимуществами данного ресурса является постоянное обновление информации, динамическое добавление новых сервисов. Пользователь данного ресурса имеет возможность детально ознакомиться с каждым отдельным сервисом и категорией техники. Также пользователи данного сервиса могут просматривать контактную информацию и информацию о пунктах продаж и оказании услуг. Помимо информации о услугах, в базе данных Интернет-ресурса также содержится полная информация об центрах в которых предоставляется соответствующий сервис.

После успешной регистрации и авторизации пользователю добавляется дополнительный набор функций. Авторизованный пользователь может ставить оценки сервисам, добавлять комментарии к ним, сервисы и услуги в избранное. Также «Multiservice» даёт возможность пользователям общаться в специальном разделе «Сообщения».

Из выявленных недостатков данного ресурса необходимо выделить необходимость покупать платную подписку для дополнительных функций, невозможность забронировать услуги на определённый сервисный центр. Также немаловажным недостатком «Multiservice» является достаточно неудобный пользовательский интерфейс, недостаточный учет индивидуальных интересов и пожеланий посетителя данного ресурса.

* 1. Обзор средств разработки

PostgreSQL – объектно-ориентированная система управления базами данных. PostgreSQL является одной из наиболее популярных систем управления базами данных. Эта СУБД базируется на языке SQL и поддерживает многие из возможностей стандарта SQL:2011.

Основные особенности PostgreSQL:

* разные типы индексов;
* наследование таблиц;
* расширяемость;
* встроенный процедурный язык PL/pgSQL;
* встроенная поддержка слабоструктурированных данных в формате JSON;
* высокопроизводительные и надёжные механизмы транзакций и репликации.

По сравнению с другими СУБД PostgreSQL имеет более расширенные ограничения, который приведены в таблице 1.1.

Таблица 1.1 – Ограничения PostgreSQL

|  |  |
| --- | --- |
| Максимальный размер базы данных | Нет ограничений |
| Максимальный размер таблицы | 32 [Тбайт](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%80%D0%B0%D0%B1%D0%B0%D0%B9%D1%82) |
| Максимальный размер поля | 1 [Гбайт](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B3%D0%B0%D0%B1%D0%B0%D0%B9%D1%82) |
| Максимум записей в таблице | Ограничено размерами таблицы |
| Максимум полей в записи | 250-1600, в зависимости от типов полей |
| Максимум индексов в таблице | Нет ограничений |

Node.js – программная платформа, основанная на движке V8, для разработки веб-приложений на языке Javascript. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера. Для разработки MVC архитектуры использовался фреймворк Express. [5]

Основные возможности Node.js в совокупности с использованием фреймворка Express:

* неблокирующие базовые механизмы ввода вывода;
* поддерживает механизм асинхронности;
* удобный пакетный менеджер npm;
* система кеширования;
* большое количество библиотек;
* система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
* при помощи библиотеки Sequelize добавляется возможность использования ORM для доступа к базам данных;

EJS – JavaScript-библиотека с открытым исходным кодом для разработки frontend части приложения. Основная цель EJS – минимизировать ошибки, уменьшить время отклика во время работы пользовательских интерфейсов. Это достигается за счёт использования рендеринга – создание динамических страниц, которые описывают часть пользовательского интерфейса. Эти страницы объединяются для создания полноценного пользовательского интерфейса. EJS абстрагирует большую часть работы по визуализации, оставляя вам возможность сосредоточиться на дизайне и простом отображении бизнес информации. [2]

Visual Studio Code – интегрированная среда разработки для языка программирования Javascript. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска режима отладки и поддерживает веб-разработку на Javasctipt. Visual Studio Code разработана компанией Microsoft.

1. **Проектирование**
   1. Проектирование базы данных

Одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя. Модели базы данных для программного средства представлена на рисунке 2.1.

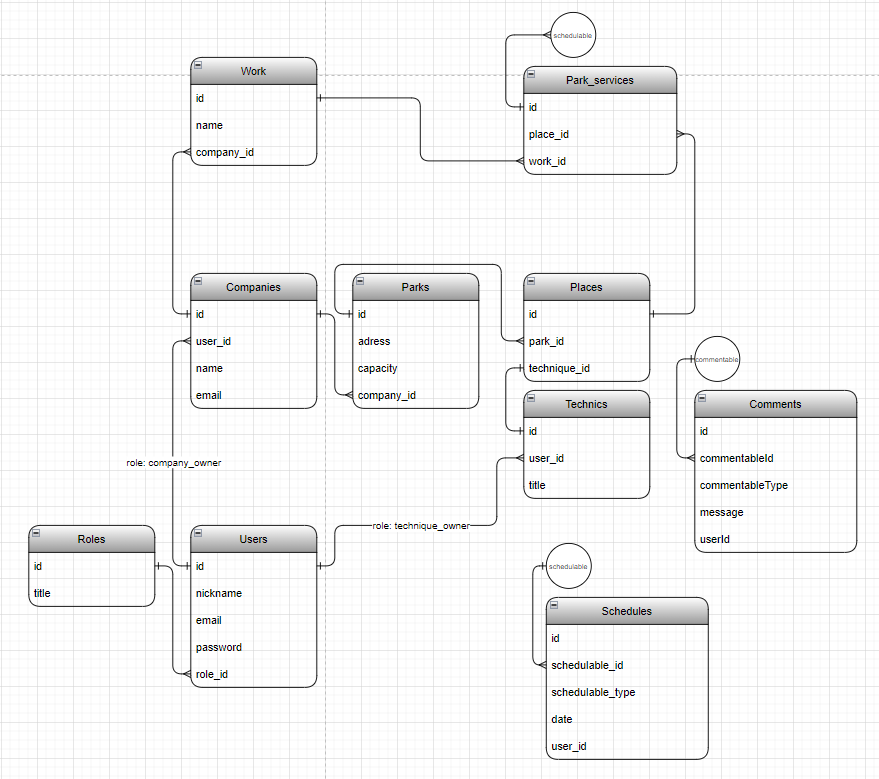


Рисунок 2.1 – Модель базы данных

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение. Основной таблицей базы данных является таблица «Park\_services» в которой хранятся все заказанные услуги. Эта таблица связана посредством связи многие-к-одному с двумя таблицами: «Works» в которой хранятся все услуги, предоставляемые компанией, и с «Places» в которой хранится вcя техника клиента, расположенная в парках компании.

Далее будет описана каждая таблица базы данных.

В таблице «Users» хранятся все зарегистрированные пользователи, разделённые на роли посредством столбца «role\_id».

Описание структуры таблицы «Users» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор пользователя |
| nickname | character varying(510) | Имя пользователя |
| email | character varying(510) | Почтовый адрес пользователя |
| password | character varying(510) | Пароль |
| role\_id | bigint, foreign key | Идентификатор роли |

В таблице «Roles» хранятся все допустимые роли в приложении. Описание структуры таблицы «Roles» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «Roles»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор роли |
| title | character varying(50), unique | Название роли |

Таблица «Companies» содержит все зарегистрированные компании. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «Companies»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор таблицы |
| user\_id | bigint, foreign key | Идентификатор создатели |
| name | character varying(50) | Название |
| email | character varying(50) | Почтовый адрес компании |

В таблице «Parks» хранятся все парки принадлежащие компании. Описание структуры таблицы «Parks» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «Parks»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор страны |
| address | character varying(50) | Адрес парка |
| capacity | bigint | Вместимость парка |
| company\_id | bigint, foreign key | Идентификатор компании |

Таблица «Techniques» содержит всю технику, принадлежащую пользователю. Её структура представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы «Techniques»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |

Окончание таблицы 2.5

|  |  |  |
| --- | --- | --- |
| id | bigint, primary key | Идентификатор таблицы |
| user\_id | bigint, foreign key | Идентификатор фильма |
| title | character varying(50) | Название техники |

В таблице «Places» хранятся все места за которыми закреплена техника расположенная в парке.

Описание структуры таблицы «Places» представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «Places»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор пользователя |
| park\_id | bigint, foreign key | Идентификатор парка |
| technique\_id | bigint, foreign key | Идентификатор техники |

Таблица «Works» содержит все услуги, предоставляемые компанией.

Описание структуры таблицы «Works» представлено в таблице 2.7.

Таблица 2.7 – Структура таблицы «Works»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор таблицы |
| name | character varying(50) | Название услуг |
| company\_id | bigint, foreign key | Идентификатор компании |

Таблица «Park\_services» является промежуточной между таблицами «Works» и «Places», она отражает заказанную услугу.

Описание структуры таблицы «Park\_services» представлено в таблице 2.8.

Таблица 2.8 – Структура таблицы «Park\_services»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор таблицы |
| place\_id | bigint, foreign key | Идентификатор места |
| work\_id | bigint, foreign key | Идентификатор услуга |

В таблице «Schedules» отображаются даты заказанных услуг. Таблица реализует полиморфические связи.

Описание структуры таблицы «Schedules» представлено в таблице 2.9.

Таблица 2.9 – Структура таблицы «Schedules»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор таблицы |
| schedulable\_id | bigint | Идентификатор модели c расписанием |
| schedulable\_type | character varying(255) | Тип модели c расписанием |

В таблице «Comments» отображаются комментарии к услугам компании.

Описание структуры таблицы «Comments» представлено в таблице 2.9.

Таблица 2.9 – Структура таблицы «Comments»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | bigint, primary key | Идентификатор таблицы |
| commetable\_id | bigint | Идентификатор комментируемой модели |
| commetable \_type | character varying(255) | Тип комментируемой модели |
| user\_id | bigint, primary key | Идентификатор комментатора |

Таким образом при проектировании базы данных получилось использовать связи многие-ко-многим, один ко многим, полиморфик и 10 таблиц.

* + 1. Индексы

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Рационально создать индексы на столбцы, которые часто используются при выборке, сортировке. Также рационально использовать индексы на столбцы с ограничением целостности foreign key и unique.

PostgreSQL поддерживает несколько типов индексов: B-Tree, хеш, GiST, SP-GiST, GIN и BRIN.

B-Tree индексы могут работать в условиях на равенство и в проверках диапазонов с данными, которые можно отсортировать в некотором порядке. Точнее, планировщик запросов PostgreSQL может задействовать индекс B-Tree, когда индексируемый столбец участвует в сравнении с одним из операторов как «<», «>», «<=», «>=», «=», «BETWEEN», «IN», «IS NULL», «IS NULL». Также оптимизатор может использовать эти индексы в запросах с операторами сравнения по шаблону LIKE и ~, если этот шаблон определяется константой, и он привязан к началу строки. Таким образом B-Tree индексы охватывают основную часть операций и соответственно будут использоваться в большинстве случаев. [4]

GIN расшифровывается как Generalized Inverted Index – это так называемый обратный индекс. Он работает с типами данных, значения которых не являются атомарными, а состоят из элементов. При этом индексируются не сами значения, а отдельные элементы; каждый элемент ссылается на те значения, в которых он встречается. Хорошая аналогия для этого метода – алфавитный указатель в конце книги, где для каждого термина приведен список страниц, где этот термин упоминается. Как и указатель в книге, индексный метод должен обеспечивать быстрый поиск проиндексированных элементов. Основная область применения GIN индекса – ускорение полнотекстового поиска.

В каждой таблицы автоматически создаётся уникальный индекс B-Tree для столбца или группы столбцов, перечисленных в первичном ключе.

* 1. **Проектирование программного средства**

Как было сказано ранее, при разработке программного средства была выбрана архитектура MVC. MVC представляет собой один из самых популярных подходов к построению архитектуры web-приложений с передачей данных по протоколу HTTP.

Основная идея MVC заключается в разделении разных операций (чаще всего CRUD) при обращении к одному и тому же URL с помощью HTTP методов, основные из которых:

* GET – используется для получения данных;
* POST - используется для создания новой записи(ей);
* PUT – используется для обновления уже существующей записи(ей);
* PATCH – используется для обновления, но только тогда, когда изменяется идентификатор записи(ей);
* DELETE – используется для удаления записи(ей).

Основными преимуществами MVC являются:

* простота реализации. Так как архитектура MVC небольшое число компонентов каждый из которых отвечает на за решение определённых, простых для понимания задач;
* гибкость. Эта архитектура весьма гибкая и дает возможность обрабатывать различные типы запросов и форматы данных.

Недостатки MVC:

* монолитное приложение. Приложения, построенные на такой архитектуру, имеют в себе одновременно backend часть и frontend часть. Что может затруднить внедрение или переход на новых стек технологий;

Всего в приложении используется 3 основных сервиса:

* EJS – сервис, реализующий frontend-часть приложения;
* Node.js – сервис, реализующий backend-часть приложения;
* PostgreSQL – сервис, реализующий доступ к данным.

Каждый сервис программного средства максимально изолирован от других сервисов, что позволяет легко расширять функционал приложения. Для общения между сервисами программного средства используется JSON. JSON – текстовый формат обмена данными, основанный на Javascript. Он представляет собой набор пар ключ-значение. Клиентские запрос изначально обрабатываются Node.js сервером, далее EJS сервис генерирует представление, которое проходит через маршрутизатор, далее отображается в на странице web-браузера.

1. Реализация программного средства

В процессе выполнения курсового проекта было разработано веб-приложение, имеющее архитектурный стиль REST API. Таким образом было разработано два отдельных компонента backend\_node, отвечающий за серверную часть программного средства, и frontend, отвечающий за клиентское представление.

* 1. Разработка серверной части

На рисунке 3.1 отображена структура серверной части разрабатываемого программного средства, включающая в себя директории, конфигурационные файлы и файлы скриптов.

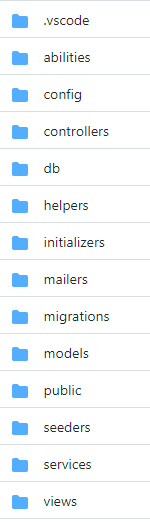


Рисунок 3.1 – Структура проекта сервера

Входной точкой в приложение является файл application.js в нем запускается прослушивание запросов посылаемых приложению работающему на определенном порту. Веб-сервер, подключает необходимые для работы сервера модули, выполняется начальная инициализация приложения, настройка ограничений, вызванных CORS и регистрация маршрутизаторов. Также вapplication.js идёт подключение к базе данных.

В директории «controllers» содержатся контроллеры. Контроллеры – это файлы, которые содержат основную маршрутизатор и сопоставляемую логику приложения. Контроллеры взаимодействуют с базой данных через модели. Пример одного из методов контроллера, а именно получение информации о копаниях, приведен в листинге 3.1. Другие примеры контроллеров находятся в Приложении А.



Листинг 3.1 – Обработка получения информации о пользователе

Директория «models» содержит файлы с классами-сущностями базы данных. Приложение использует ORM-библиотеку «Sequelize» для взаимодействия с базой данных. Конфигурацию подключения модель импортирует из файла db-connection.js. Благодаря тому, что ORM библиотека Sequelize имеет гибкий функционал, для изменения таблиц баз данных необходимо всего лишь изменить соответствующий класс модели. А для смены СУБД изменить строку подключения в файле database.js. Пример модели услуги приведён в листинге 3.2. Модели являются объектами, в приложении которые отражают таблицы в базе данных. Благодаря ORM на модели можно установить валидацию полей каждой их модели. Это обеспечивает дополнительную надежность и сохранность целостности базы данных. Это необходимо поскольку валидация полей с помощью средств html лего преодолевается. Другими словами это не способствует сохранности целостности базы данных. Другие примеры моделей находятся в Приложении Б.



Листинг 3.2 – Модель услуги

Директория «abilities» содержит файлы, реализующие авторизацию моделей, в них прописаны возможности, которыми пользователь с определённой ролью по отношению к каждой из моделей. Пример авторизации для услуг приведён в листинге 3.3

Авторизация обеспечивает изоляцию объектов приложения друг от друга, не допускает возможность изменение пользователем данных которые ему не принадлежат. Другие примеры авторизации для моделей находятся в Приложении В.



Листинг 3.3 – авторизация услуг

Директория «config» содержит JSON файл конфигурации для разны. сред управления. Пример настройки среды development приведён в листинге 3.4.



Листинг 3.4 – Настройки среды development

Директория «db» содержит JavaScript файл котором прописаны настройки подключения к базе данных. Пример настроек приведён в листинге 3.5.



Листинг 3.5 – Настройки среды development

Директория «initializers» содержит JavaScript файлы в которых прописаны настройки инициализация библиотек, используемых в приложении. Пример инициализации фреймворка «express» приведён в листинге 3.6.

Фреймворк обеспечивает необходимый функционал для приложения, но для этого необходимо правильная настройка соответствующего функционала. Например необходимо настроить использование маршрутизаторов, настроить движок рендеринга, модуль обработки тела запросов и т.д.



Листинг 3.6 – Инициализации фреймворка «express»

Директория «mailers» содержит JavaScript файлы (мэйлеры) в которых реализация взаимодействие с почтой клиентов. Пример мэйлера приведён в листинге 3.7.

Мэйлеры это один из многих способов организовать уведомление пользователя о некоторых событиях, касающихся него в приложении. Чтобы настроить мэйлер необходимо указать параметры доступа к аканту, встроенного в приложения для подобных нужд, и параметры SMTP сервера.



Листинг 3.7 – Взаимодействие с почтой пользователя

Директория «migrations» содержит JavaScript файлы (миграции) в которых описано создание базы данных при помощи ORM Sequelize. Пример миграции создающей таблицу «Works» приведён в листинге 3.8.

Подобный подход позволяет при необходимости легко менять СУБД, обеспечивая большую независимость приложения от используемых инструментов. Это возможному благодаря тому что ORM абстрагирует пользователя от прямого взаимодействия с базой данных и по средствам предоставления npm пакетов для взаимодействия с разными СУБД. К примеру если вам необходимо сменить СУБД PostgresQL на MSSQL, вам достаточно установить соответствующий пакет для работы с новой СУБД, поменять диалект базы данных и параметры доступа. Sequelize поддерживает не малый список СУБД с которыми она позволяет работать.



Листинг 3.8 – Пример миграции, создающей таблицу «Works»

Директория «public» содержит статических JavaScript файлов в которых реализовано использование технологии web-sockets. А именно оставление комментариев об услугах компании в реальном времени. Пример использования технологии web-sockets приведён в листинге 3.9.

Протокол WebSocket («веб-сокет»), описанный в спецификации [RFC 6455](http://tools.ietf.org/html/rfc6455), обеспечивает возможность обмена данными между браузером и сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.

WebSocket особенно хорош для сервисов, которые нуждаются в постоянном обмене данными, например онлайн игры, торговые площадки, работающие в реальном времени, и т.д.



Листинг 3.9 – Пример использования технологии web-sockets

Директория «seeders» содержит JavaScript файлы (сиды) в которых описано заполнение базы данных тестовыми данными. Пример сидов для таблицы «Users» приведён в листинге 3.10.

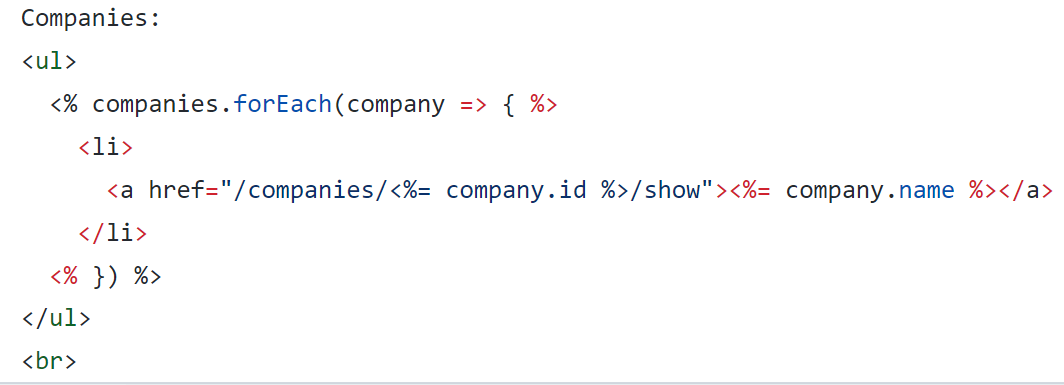
Функционал предоставляемый использованием сидов очень удобен и поддерживается всеми ORM системами. Сиды обеспечивают заполнение базы данных демонстрационными данными. Это необходимо для проверки работоспособности приложения, удобства написания тестов. Единственным недостатком такой возможности является отсутствие достаточно удобных инструментов генерации данных. Это может привести к тому что сиды станут очень большими, не читаемыми и неподдерживаемыми.

В этом проекте сиды есть на каждую таблицу из базы данных. Они задают по несколько сущностей каждой модели, что позволяет их использовать для тестирования проекта.



Листинг 3.10 – Пример сидов для таблицы «Users»

Директория «views» содержит EJS файлы (представления) в которых содержатся динамические html страницы. Представления имеют определённую структуру и пере используются при помощи механизмов «layout» и «partials». Пример динамического представления, отображающего список всех компаний приведён в листинге 3.11.



Листинг 3.10 – Пример динамического представления, отображающего список всех компаний

1. Руководство пользователя
   1. **Функциональные возможности пользователя**

Для наглядности функциональности рассмотрим UML-диаграммы для разных типов пользователей.

На рисунке 4.1 показана UML-диаграмма, отображающая возможности приложения с точки зрения неавторизованного пользователя.

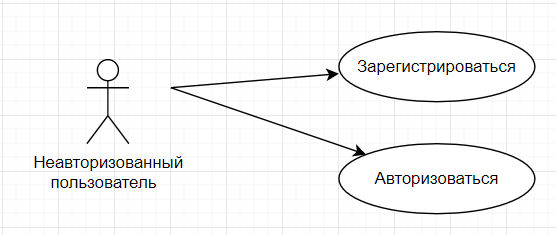


Рисунок 4.1 – UML-диаграмма вариантов использования приложения для неавторизованного пользователя

Основной функционал не доступен неавторизованному пользователю. Он может только зарегистрироваться и авторизоваться.

Если пользователь авторизуется, то ему становиться доступен основной функционал, который отличается в зависимости от того под какой ролью авторизован пользователь.

На рисунке 4.1.1 представлена страница аутентификации пользователей, на которую будет перенаправляться неавторизованный пользователь.

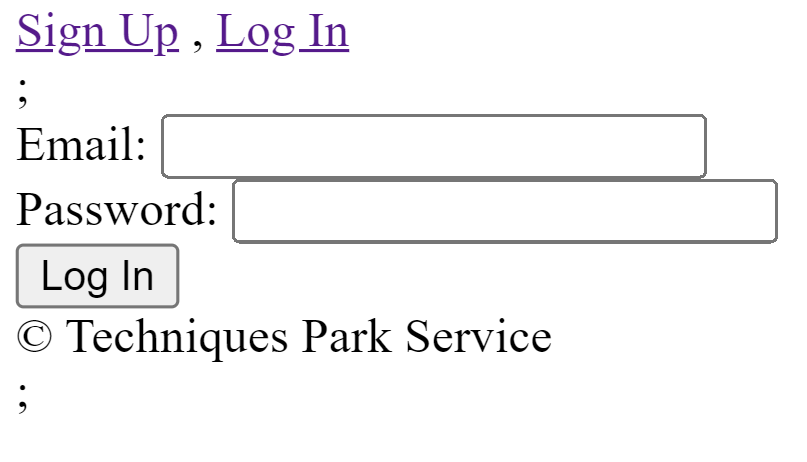


Рисунок 4.1.1 – Страницы аутентификации

На рисунке 4.2 показана UML-диаграмма, отображающая возможности приложения с точки зрения авторизованного как владелец техники пользователя.

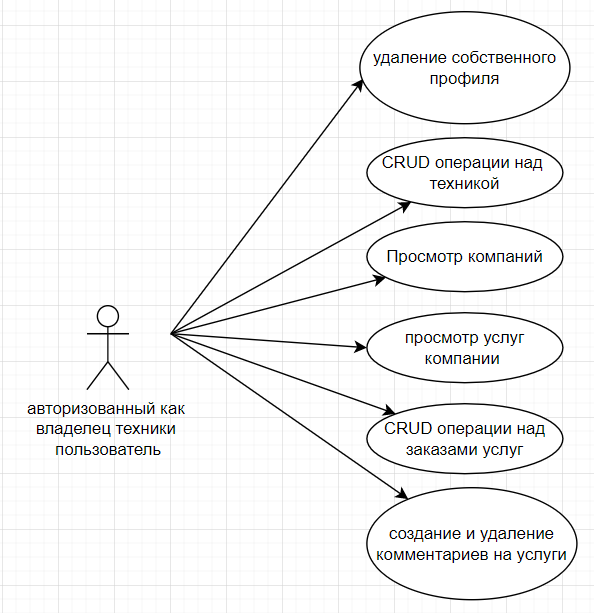


Рисунок 4.2 – UML-диаграмма вариантов использования приложения для авторизованного как владелец техники пользователя

Авторизованному пользователю добавляется дополнительные функции: работа с техникой, возможность работать со своими заказами и т.д. Также пользователь может удалить собственный профиль.

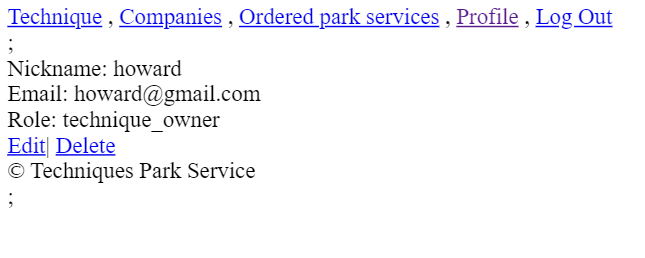


Рисунок 4.2.1 – Страница профиля

На рисунке 4.2.2 отображена страница просмотра заказанных услуг

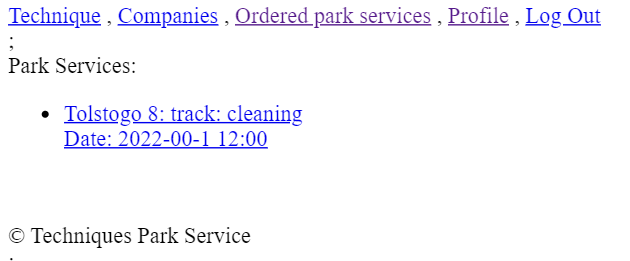


Рисунок 4.2.1 – Страница профиля

На рисунке 4.3 показана UML-диаграмма, отображающая возможности приложения с точки зрения авторизованного как владелец компании пользователя.

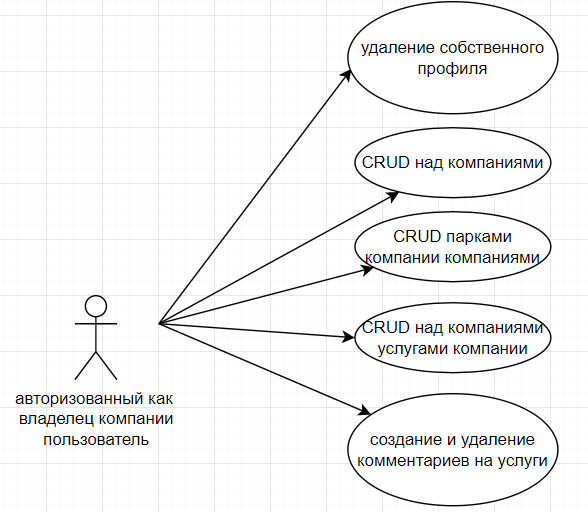


Рисунок 4.3 – UML-диаграмма вариантов использования приложения для авторизованного как владелец компании пользователя

Пользователь авторизованный как владелец компании может производить CRUD операции над компаниями, парками.

На рисунке 4.3.1 отображена страница просмотра компаний зарегистрированных в приложении.

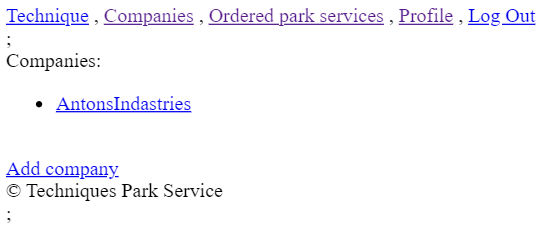


Рисунок 4.3.1 – Страница профиля

На странице просмотра компании доступны ссылки для просмотра заказанных услуг в данной компании, принадлежащие компании парки, предоставляемые компанией услуги и ссылка для заказа новой услуги, рисунок 4.3.2.

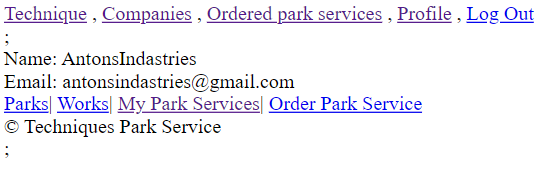


Рисунок 4.3.1 – Страница профиля

Пользователи с ролью владельца компании не имеют возможности добавлять себе технику, создавать заказы на услуги, поскольку такие пользователи их предоставляют через свои компании. Они могут создать компанию, создать в ней парк, указать предоставляемые услуги.

1. Тестирование работоспособности
   1. Тестирование приложения

Данное программное средство тестировалось вручную, были выполнены все технические моменты, предусмотренные реализацией, а также были предприняты попытки нарушить работу приложения.

При переходе на страницу регистрации и вводе значений пользователь переходит на страницу входа. На рисунке 5.1 изображено окно регистрации.

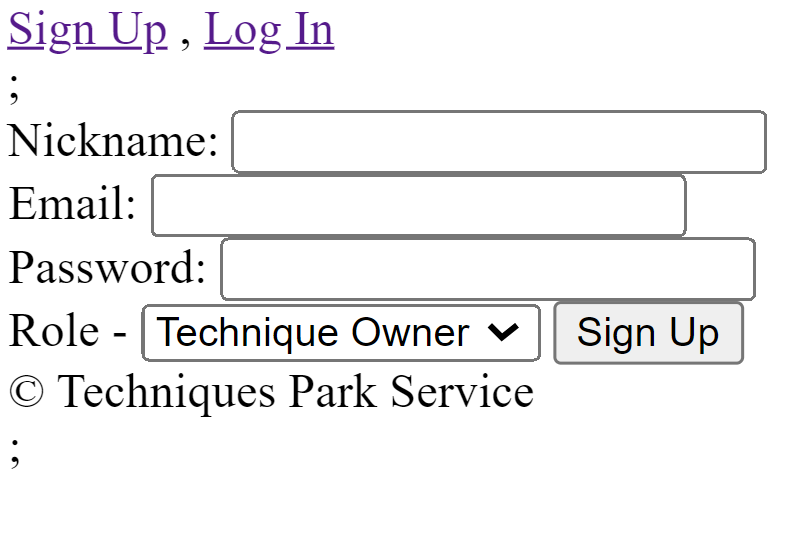


Рисунок 5.1 – Окно регистрации

При попытке перехода на странице указывающие на других пользователей, запрос перенаправляется на последнюю допустимую для пользователя страницу.

На уровне базы данных прописаны ограничения целостности на поля моделей. В них указаны такие ограничения как: отсутствие нулевых или пустых значений, ограничения по внешнему ключу, ограничения на максимальные значения полей.

Заключение

Данный курсовой проект был направлен на развитие навыков построения кроссплатформенных веб-приложений и администрирования базы данных. В результате чего было создано веб-приложения «Парк технического обслуживания». Удалось обеспечить большую гибкость базы данных за счет исключения избыточности, несогласованности зависимостей и использования полиморфических связей. Разработанный проект состоит из веб-сервера, базы данных и клиентской части. Веб-сервер реализован на платформе NodeJS, благодаря этому сервер является кроссплатформенным.

В ходе выполнения работы были достигнуты следующие результаты:

* исследованы приложения аналоги;
* сформулирована модель данных PostgreSQL и приведено ее графическое представление;
* были созданы функции для взаимодействия приложения с базой данных;
* была применена технология web socket;
* были проведены операции для улучшения быстродействия базы данных путем добавления индексов в таблицы;
* было разработано веб-приложения для демонстрации возможностей базы данных.

Также были выполнены следующие функциональные задачи:

* Авторизация и регистрация пользователя;
* Создание записей на обслуживание;
* Cоздание компаний, предоставляющих обслуживание;
* Пользователи имеют возможность оставлять комментарии по обслуживанию;
* Удаление пользователей.

Тестирование программы показало, что она работает корректно и выполняет все свои функции.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список используемых источников

1. Sequelize документация [Электронный ресурс] – Режим доступа: https://sequelize.org/master/manual/model-basics.html – Дата доступа: 10.04.2022;
2. EJS документация [Электронный ресурс] – Режим доступа: https://ejs.co/ – Дата доступа: 04.05.2022.
3. Socket.io документация [Электронный ресурс] – Режим доступа: <https://socket.io/docs/v4/> – Дата доступа: 04.05.2022;
4. Типы индексов PostgreSQL [Электронный ресурс] – Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/indexes-types> – Дата доступа: 11.05.2022;
5. Node.js документация [Электронный ресурс] – Режим доступа: <https://nodejsdev.ru/> – Дата доступа: 11.05.2022.

ПРИЛОЖЕНИЕ А

const { Park, User, Company } = require('../models/associate');

const { authenticate } = require('../services/authentication\_service');

const { authorize } = require('../abilities/parks\_ability');

const parksRouter = require('express').Router();

parksRouter.get(

'/parks/:companyId/index',

authenticate(),

authorize('read\_all'),

async (req, res) => {

const { params } = req;

const viewBag = { authenticated: req.isAuthenticated(), showUserPath: `/users/${req.user.id}/show` }

viewBag.parks = await Park.findAll({

where: { companyId: parseInt(params.companyId) },

include: 'company',

});

viewBag.user = await User.findByPk(req.user.id);

viewBag.userOwnCompany = await viewBag.user.hasCompany(await Company.findByPk(params.companyId));

viewBag.newParkPath = `/parks/${params.companyId}/new`;

res.render('./parks/index', viewBag);

},

);

parksRouter.get(

'/parks/:companyId/new',

authenticate(),

authorize('create'),

async (req, res) => {

const { params } = req;

const viewBag = { authenticated: req.isAuthenticated(), showUserPath: `/users/${req.user.id}/show` }

viewBag.path = `/parks/${params.companyId}/create`;

viewBag.buttonLabel = 'Create';

viewBag.park = {};

viewBag.company = { id: parseInt(params.companyId) };

res.render('./parks/new', viewBag);

},

);

parksRouter.post(

'/parks/:companyId/create',

authenticate(),

authorize('create'),

async (req, res) => {

const { params, body } = req;

const park = await Park.create({

address: body.address,

capacity: body.capacity,

companyId: body.companyId,

});

res.redirect(`/parks/${params.companyId}/index`);

},

);

parksRouter.get(

'/parks/:companyId/show/:id',

authenticate(),

authorize('read'),

async (req, res) => {

const { params } = req;

const viewBag = { authenticated: req.isAuthenticated(), showUserPath: `/users/${req.user.id}/show` }

viewBag.user = await User.findByPk(req.user.id);

viewBag.userIsCompanyOwner = await viewBag.user.isCompanyOwner();

viewBag.park = await Park.findOne({

where: { id: parseInt(params.id), companyId: parseInt(params.companyId) },

include: 'company',

});

viewBag.editParkPath = `/parks/${params.companyId}/edit/${params.id}`;

viewBag.deleteParkPath = `/parks/${params.companyId}/delete/${params.id}?\_method=DELETE`;

viewBag.placesIndexPath = `/places/${params.id}/index`;

res.render('./parks/show', viewBag);

},

);

parksRouter.get(

'/parks/:companyId/edit/:id',

authenticate(),

authorize('update'),

async (req, res) => {

const { params } = req;

const viewBag = { authenticated: req.isAuthenticated(), showUserPath: `/users/${req.user.id}/show` }

viewBag.park = await Park.findOne({

where: { id: parseInt(params.id), companyId: parseInt(params.companyId) },

include: 'company',

});

viewBag.company = viewBag.park.company;

viewBag.path = `/parks/${params.companyId}/update/${params.id}?\_method=PUT`;

viewBag.buttonLabel = 'Update';

res.render('./parks/edit', viewBag);

},

);

parksRouter.put(

'/parks/:companyId/update/:id',

authenticate(),

authorize('update'),

async (req, res) => {

const { params, body } = req;

const park = await Park.update(

{

address: body.address,

capacity: body.capacity,

},

{ where: { id: parseInt(params.id), companyId: parseInt(params.companyId) } },

);

res.redirect(`/parks/${params.companyId}/show/${params.id}`);

},

);

parksRouter.delete(

'/parks/:companyId/delete/:id',

authenticate(),

authorize('delete'),

async (req, res) => {

const { params } = req;

const park = await Park.destroy({ where: { id: parseInt(params.id), companyId: parseInt(params.companyId) } });

res.redirect(`/parks/${params.companyId}/index`);

},

);

module.exports = parksRouter;

ПРИЛОЖЕНИЕ Б

const {

Model,

} = require('sequelize');

const moment = require('moment')

module.exports = (sequelize, DataTypes) => {

class Schedule extends Model {

static associate(models) {

if (models.parkService) {

Schedule.belongsTo(models.parkService, {

foreignKey: 'schedulableId',

onDelete: 'CASCADE',

constraints: false,

});

}

}

formFormat() {

return moment(this.date).format('YYYY-MM-DDTHH:MM')

}

showFormat() {

return moment(this.date).format('yyyy-mm-D hh:mm')

}

}

Schedule.init({

schedulableId: DataTypes.INTEGER,

schedulableType: DataTypes.STRING,

date: DataTypes.DATE

}, {

sequelize,

modelName: 'Schedule',

});

Schedule.addHook('afterFind', (findResult) => {

if (!Array.isArray(findResult)) findResult = [findResult];

for (const instance of findResult) {

if (instance.commentableType === 'ParkService' && instance.ParkService !== undefined) {

instance.commentable = instance.image;

}

// To prevent mistakes:

delete instance.ParkService;

delete instance.dataValues.ParkService;

}

});

return Schedule;

};

ПРИЛОЖЕНИЕ В

const { subject } = require('@casl/ability');

const { AbilityBuilder, Ability } = require('@casl/ability');

const \_ = require('underscore');

const { Park, User } = require('../models/associate');

function authorize(abilityName, failureRedirect = 'back') {

return async (req, res, next) => {

if (await parkAuthorize(req, abilityName)) {

return next();

} else {

res.redirect(failureRedirect);

}

};

}

async function parkAuthorize(req, abilityName) {

const authUser = await User.findOne({ where: { id: req.user.id }, include: 'companies' });

const authUserCompaniesIds = \_.map(authUser.companies, (company) => company.id);

let park = subject('Park', { companyId: authUser.id });

if (req.params.id) {

park = await Park.findOne({ where: { id: req.params.id } });

}

const ability = await abilities(authUser, authUserCompaniesIds);

return ability.can(abilityName, park);

}

let abilities = async (user, authUserCompaniesIds) => {

const { can, rules } = new AbilityBuilder(Ability);

if (await user.isTechniqueOwner()) {

can('read\_all', 'Park');

can('read', 'Park');

} else if (user.isCompanyOwner()) {

can('manage', 'Park', { companyId: { $in: authUserCompaniesIds } });

if (authUserCompaniesIds == 0) { can('manage', 'Park'); }

}

return new Ability(rules);

};

module.exports = {

abilities, parkAuthorize, subject, authorize,

};