Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Разработка приложения для сбора статистики по дорожно-транспортным происшествиям Российской Федерации»

Студент _____ИУ7-62Б_____ _____ _____Д.А. Куликов_____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ ____Ю.М. Гаврилова____
(Подпись, дата) (И.О.Фамилия)

Москва, 2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ____ИУ7___
(Индекс)
_____ _И.В Рудаков_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Базы данных_____

Студент группы _____ИУ7-62Б___

_____Куликов Дмитрий Алексеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта _Разработка приложения для сбора статистики по дорожно-транспортным происшествиям Российской Федерации_____

Направленность КП_____учебная_____

Источник тематики _____кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к _11_ нед., 100% к _14_ нед.

Задание Спроектировать и реализовать базу данных дорожно-транспортных происшествий. Разработать интерфейс, который позволит работать с данной базой для получения информации и статистики дорожно-транспортных проишествий по регионам Российской Федерации._____

Оформление курсового проекта:

Расчетно-пояснительная записка на __20-30__ листах формата А4.
Перечень графического материала (плакаты, схемы, чертежи и т.п.).
На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс._____
Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ ____Ю.М. Гаврилова__
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Д.А. Куликов _
(Подпись, дата) (И.О.Фамилия)

РЕФЕРАТ

Курсовой проект представляет собой реализацию приложения для сбора статистики по дорожно-транспортным происшествиям Российской Федерации.

Ключевые слова: web-приложение, ДТП, ТС, статистика погибших и раненных, PostreSQL, Angular, Nest JS, Yandex Maps.

Приложение реализуется на языке программирования TypeScript с использованием фреймворков Angular и Nest JS.

Полученное в результате работы ПО является инструментом для составления протоколов дорожно-транспортных происшествий и может быть использовано сотрудниками ГИБДД.

Отчёт содержит 39 страниц, 22 рисунка, 1 таблицу, 8 источников.

Оглавление

Введение 5
1. Аналитическая часть 6
1.1 Постановка задачи 6
1.2 Формализация данных 6
1.3 Анализ моделей баз данных и выбор наиболее подходящей СУБД 8
1.3.1 Основные функции СУБД 8
1.3.2 Классификация СУБД по модели данных 8
1.3.3 Выбор реляционной СУБД 10
1.3.3.1 Microsoft SQL Server 10
1.3.3.2 MySQL 11
1.3.3.3 PostgreSQL 12
1.3.3.4 Oracle 12
2. Конструкторская часть 14
2.1 Функциональная модель 14
2.2 Сценарий использования 14
2.3 Проектирование базы данных 16
2.4 Проектирование архитектуры приложения 19
Вывод 20
3. Технологическая часть 21
3.1 Выбор и обоснование инструментов разработки 21
3.2 Реализация моделей хранения данных 22
3.4 Реализация контроллера 23
3.5 Реализация бизнес логики 25
3.6 Администрирование БД 26
3.7 Интерфейс приложения 31
Вывод 37
Заключение 38
Список литературы 39

Введение

Дорожно-транспортный травматизм в России уверенно снижается – за 5 месяцев 2021 года количество ДТП снизилось почти на 14%, удалось спасти более 300 жизней. Однако безопасность дорожного движения остается одной из серьёзнейших проблем России. Для того чтобы сохранять тенденцию снижения показателя смертности на дорогах необходимо проведение углубленного анализа динамики, структуры и причин совершения дорожно-транспортных происшествий.

Цель курсовой работы – реализовать приложение для сбора статистики по дорожно-транспортным происшествиям Российской Федерации.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) формализовать задание, определить необходимый функционал;

2) провести анализ существующих СУБД;

3) описать структуру базы данных, включая объекты, из которых она состоит;

4) спроектировать приложение для доступа к БД;

5) разработать программное обеспечение, которое позволит пользователю получать информацию об аварийных ситуациях на дорогах.

1. Аналитическая часть

В данном разделе представлены постановка задачи, анализ существующих моделей баз данных и выбор наиболее подходящей СУБД для решения поставленных задач.

1.1 Постановка задачи

Необходимо разработать приложение для сбора статистики по дорожно-транспортным происшествиям Российской Федерации. Пользователь должен иметь возможность просматривать количество погибших и раненых в результате аварийных ситуаций на дорогах. Данное приложение является инструментом для составления протоколов дорожно-транспортных происшествий и может быть использовано сотрудниками ГИБДД.

1.2 Формализация данных

В соответствии с поставленной задачей необходимо разработать клиент-серверное веб-приложение с возможностью аутентификации пользователей.

Для каждого типа пользователя предусмотрен свой набор функций.

Неавторизованный пользователь:

- просмотр количества дорожно-транспортных происшествий в городах и регионах РФ;
- просмотр количества пострадавших водителей, пассажиров и пешеходов.

Сотрудник ГИБДД:

- составление протоколов дорожно-транспортных происшествий;
- регистрирование поврежденных транспортных средств;
- добавление информации об участниках дорожно-транспортных происшествий;

- редактирование дорожно-транспортных происшествий в городах и регионах РФ.

Администратор:

- регистрирование нового пользователя в системе с установлением соответствующей роли;
- редактирование учетной записи сотрудника ГИБДД.

База данных должна хранить информацию о:

- дорожно-транспортных происшествиях;
- пострадавших водителях, пешеходах и пассажирах;
- поврежденных транспортных средств;
- учетной записи пользователя.

В таблице 1.1 приведены категории и сведения о данных.

Таблица 1.1 – категории и сведения о данных

Категория	**Сведения**
ДТП	Дата, время, местоположение, тип(опрокидывание/столкновение/наезд на пешехода/итд), описание
Пострадавшие	ФИО, дата рождения, паспорт (если нет 14 лет, то свидетельство о рождении), водительское удостоверение, состояние здоровье, виновность.
Поврежденные ТС	Марка, модель, тип(легковой автомобиль/грузовой автомобиль/трамвай/итд), регистрационный номер, владелец ТС
Пользователь	Логин, пароль, права доступа

1.3 Анализ моделей баз данных и выбор наиболее подходящей СУБД

Система управления базами данных, сокр. СУБД — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [1].

1.3.1 Основные функции СУБД

Основными функциями СУБД являются:

- управление данными во внешней памяти;
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД.

1.3.2 Классификация СУБД по модели данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных [2].

Существует 3 основных типа моделей организации данных:

- иерархическая;
- сетевая;
- реляционная.

В иерархической модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных

уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

На рисунке 1.1 представлена структура иерархической модели данных.

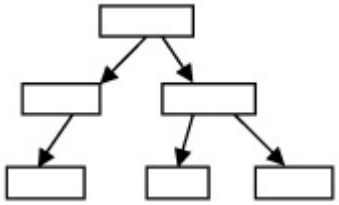


Рис. 1.1 – структура иерархической модели данных

В отличие от иерархической модели данных потомок сетевой может иметь любое число предков. Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями. Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения.

На рисунке 1.2 представлена структура сетевой модели данных.

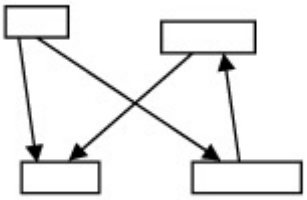


Рис. 1.2 – структура сетевой модели данных

Реляционная модель данных является совокупностью данных и состоит из набора двумерных таблиц. При табличной организации отсутствует

иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов. Реляционная модель является удобной и наиболее широко используемой формой представления данных.

На рисунке 1.3 представлена структура реляционной модели данных.

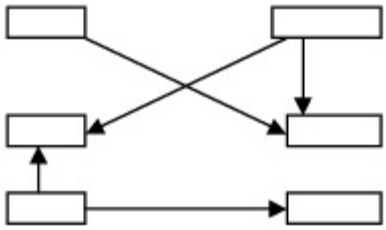


Рис. 1.3 – структура реляционной модели данных

В результате анализа моделей баз данных, в соответствии с поставленной задачей**,** наиболее оптимальным решением является использование реляционной модели базы данных, так как это позволит реализовать поставленные цели, не усложняя программную архитектуру.

1.3.3 Выбор реляционной СУБД

Наиболее популярными РСУБД являются Microsoft SQL Server, MySQL, PostgreSQL, Oracle.

1.3.3.1 Microsoft SQL Server

Microsoft SQL Server — это СУБД, движок которой работает на облачных серверах, а также локальных серверах, причем можно комбинировать типы применяемых серверов одновременно.

Достоинства:

- простота использования;

- текущая версия работает быстро и стабильно;
- движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
- визуализация на мобильных устройствах.

Недостатки:

- высокая стоимость продукта для юридических лиц;
- высокая ресурсоемкость SQL Server.
- возможны проблемы с использованием службы интеграции для импорта файлов.

1.3.3.2 MySQL

MySQL — реляционная СУБД с открытым исходным кодом с моделью клиент-сервер.

Достоинства:

- простота использования;
- поддерживает большую часть функционала SQL;
- поддерживает набор пользовательских интерфейсов;
- может работать с другими базами данных, включая Oracle.

Недостатки:

- ограничения функциональности, так как не полностью реализованы SQL-стандарты;
- некоторые операции реализованы менее надёжно, чем в других РСУБД.

1.3.3.3 PostgreSQL

PostgreSQL — реляционная СУБД, которая отличается от других тем, что обладает объектно-ориентированным функционалом, в том числе полной поддержкой концепта ACID (Atomicity, Consistency, Isolation, Durability).

Достоинства:

- полная SQL-совместимость;
- поддержка сторонними организациями;
- является масштабируемой и способна обрабатывать терабайты данных;
- расширяемость за счёт использования хранимых процедур.

Недостатки:

- скорость работы может уменьшаться во время проведения пакетных операций или выполнения запросов чтения.

1.3.3.4 Oracle

Oracle — объектно-реляционная СУБД.

Достоинства:

- поддержка огромных баз данных и большого числа пользователей;
- быстрая обработка транзакций.

Недостатки:

- высокая стоимость;
- требуются значительные вычислительные ресурсы.

В результате анализа реляционных СУБД , в соответствии с поставленной задачей**,** наиболее оптимальным решением является использование PostgreSQL, так как обеспечивает целостность данных, поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных.

Вывод

В данном разделе была поставлена задача реализации приложения для сбора статистики по дорожно-транспортным происшествиям, проведен анализ моделей баз данных и сравнение наиболее популярных РСУБД. Для реализации поставленной задачи принято использовать PostgreSQL.

2. Конструкторская часть

В данном разделе рассматриваются структура приложения и базы данных.

2.1 Функциональная модель

На рисунке 2.1 изображена функциональная модель, отображающая структуру и функции системы.

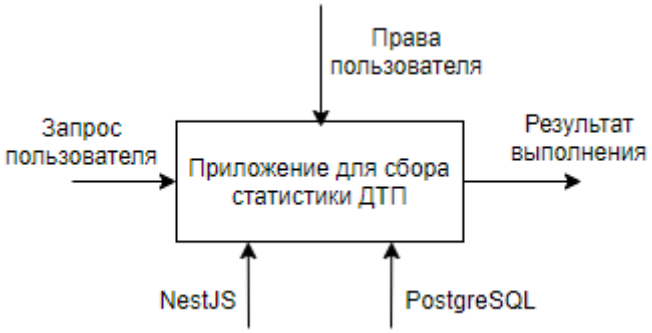


Рис. 2.1 – Функциональная модель приложения для сбора статистики ДТП.

2.2 Сценарий использования

В данном разделе необходимо построить Use Case Diagram (диаграмму прецедентов). Она состоит из графической диаграммы, описывающей действующие лица и прецеденты – конкретные действия, которые выполняет пользователь при работе с системой.

Данная диаграмма предназначена для определения функциональных требований. В системе есть два типа пользователей:

- авторизованные (сотрудник ГИБДД, администратор);
- неавторизованные.

На рисунке 2.2 представлена Use Case Diagram для действий неавторизованного пользователя.

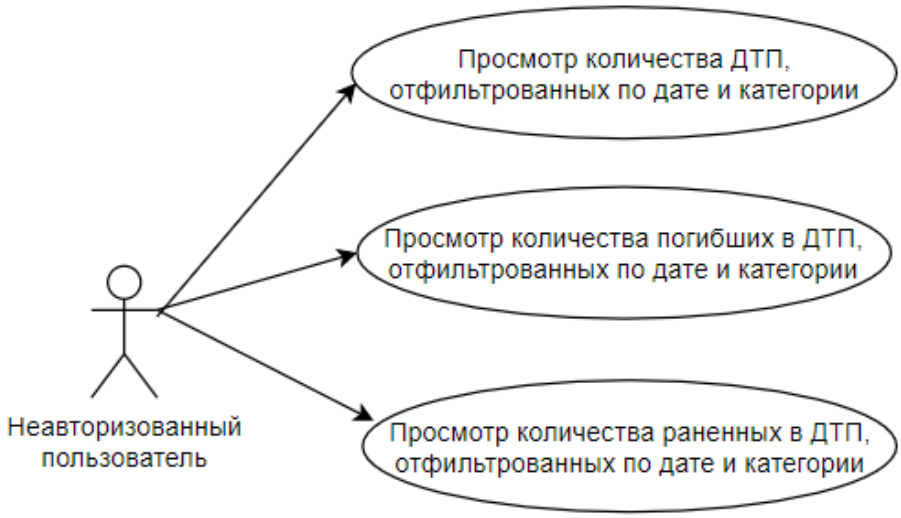


Рис. 2.2 – Use Case Diagram для действий неавторизованного пользователя.

На рисунке 2.3 представлена Use Case Diagram для действий авторизованного пользователя.

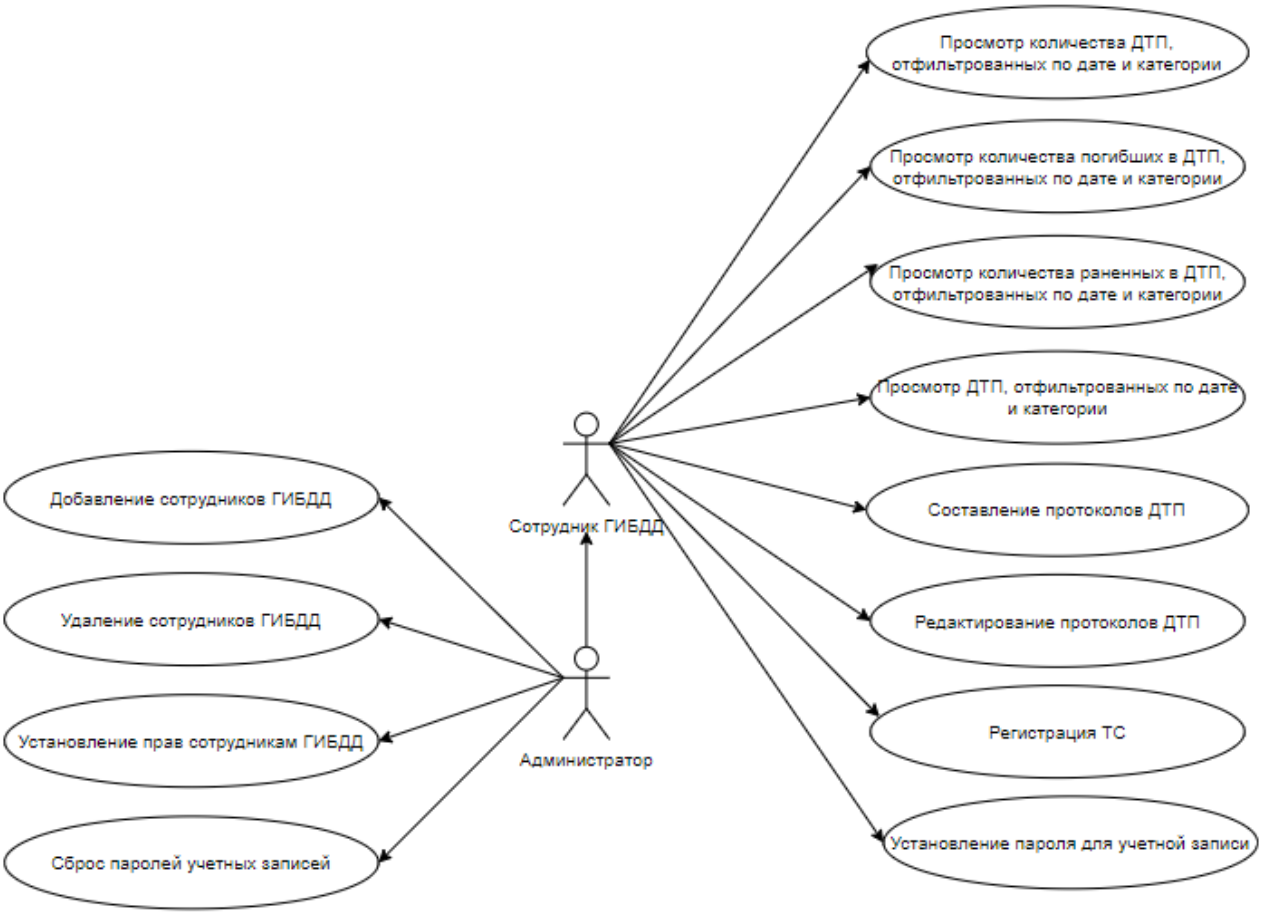


Рис. 2.3 – Use Case Diagram для действий авторизованного пользователя.

2.3 Проектирование базы данных

База данных должна хранить рассмотренные в таблице 1.1 данные. В соответствии с этой таблицей можно выделить следующие таблицы:

- таблица ДТП Dtp;
- таблица типов ДТП TypeDtp;
- таблица пострадавших водителей AffectedDrivers;
- таблица пострадавших пассажиров, пешеходов, кучеров, велосипедистов AffectedOthers;
- таблица транспортных средств TS;
- таблица людей People;
- таблица пользователей Users.

Таблица **Dtp** должна хранить информацию о ДТП:

- id – уникальный идентификатор ДТП, PK, uniqueidentifier;
- dateDtp – дата совершения ДТП, date;
- timeDtp – время совершения ДТП, time;
- regionDtp – регион, в котором произошло ДТП; character varying;
- cityDtp – город, в котором произошло ДТП; character varying;
- descriptionDtp – дополнительное описание ДТП, character varying.

Таблица **TypeDtp** должна хранить информацию о типах ДТП:

- id - уникальный идентификатор типа ДТП, PK, uniqueidentifier;
- description – тип ДТП, character varying.

Таблица **AffectedDrivers** должна хранить информацию о пострадавших водителях:

- id – уникальный идентификатор пострадавшего водителя, PK, uniqueidentifier;

- dtpID – идентификатор ДТП, FK, uniqueidentifier;
- personID – идентификатор человека, FK, uniqueidentifier;
- tsID – идентификатор ТС, FK, uniqueidentifier;
- health – состояние здоровья после ДТП (ранен/погиб/цел), character varying;
- guilt – виновность в совершении ДТП (виновен/невиновен), character varying.

Таблица **AffectedOthers** должна хранить информацию о пострадавших пассажирах, пешеходах, велосипедистах, кучерах:

- id – уникальный идентификатор пострадавшего водителя, PK, uniqueidentifier;
- dtpID – идентификатор ДТП, FK, uniqueidentifier;
- personID – идентификатор человека, FK, uniqueidentifier;
- type – тип пострадавшего (пассажир/пешеход/велосипедист/кучер), character varying;
- health – состояние здоровья после ДТП (ранен/погиб/цел), character varying;
- guilt – виновность в совершении ДТП (виновен/невиновен), character varying.

Таблица **People** должна хранить информацию о людях:

- id – уникальный идентификатор человека, PK, uniqueidentifier;
- surname – фамилия, character varying;
- name – имя, character varying;
- patronymic – отчество, character varying;
- birthdate – дата рождения, character varying;
- sex – пол (муж/жен), character varying

- passport – паспорт (если человеку меньше 14 лет, то указывается свидетельство о рождении), должен быть уникальным, character varying;
- driverlicense – водительское удостоверение, character varying.

Таблица **Ts** должна хранить информацию о транспортных средствах:

- id – уникальный идентификатор транспортного средства, PK, uniqueidentifier;
- brand – марка, character varying;
- model – модель, character varying;
- color – цвет, character varying;
- registernumber – регистрационный номер, должен быть уникальным, character varying;
- ownerId – ID владельца, FK, uniqueidentifier.

Таблица **User** должна хранить информацию о пользователях:

- id - уникальный идентификатор пользователя, PK, uniqueidentifier;
- login – логин пользователя, используется для авторизации, должен быть уникальным, character varying;
- password – пароль пользователя, используется для авторизации, должен хранится в хешированном состоянии, character varying;
- role – права доступа пользователя (администратор/сотрудник), character varying.

На рисунке 2.4 представлена ER – модель базы данных.

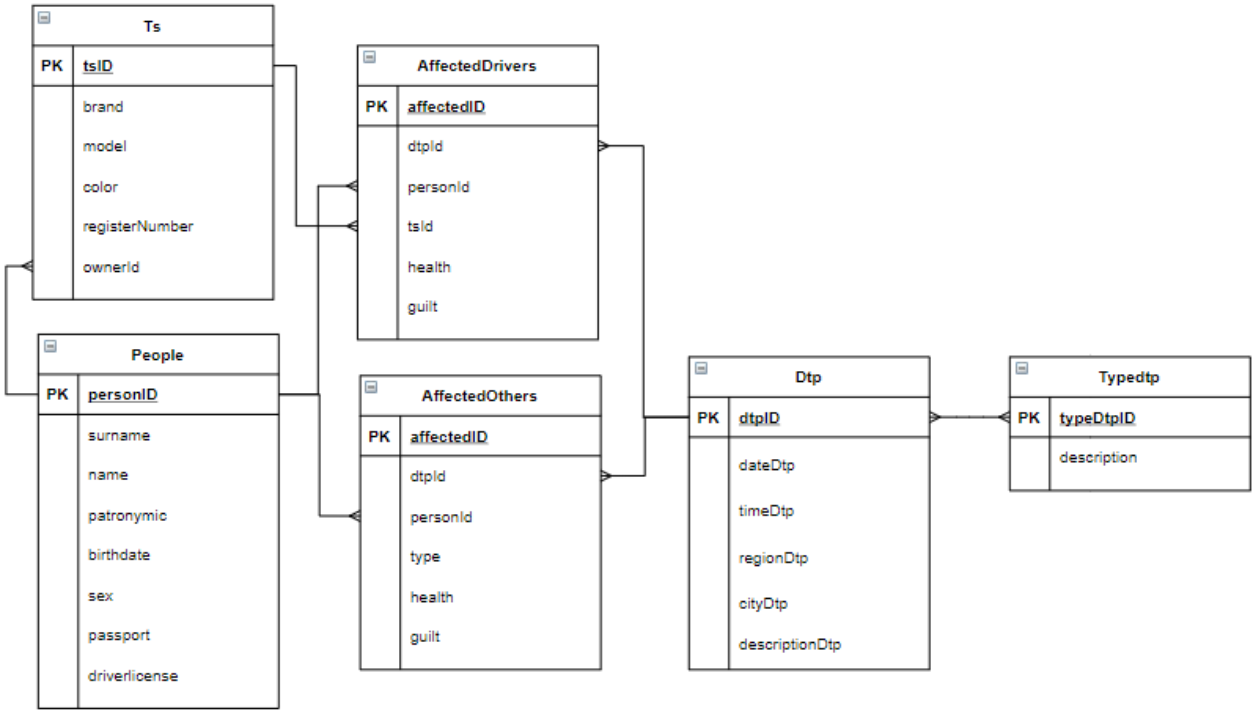


Рис. 2.4 – ER – модель базы данных.

2.4 Проектирование архитектуры приложения

При построении архитектуры приложения был выбран подход к созданию клиент-серверного web-приложения.

В пользовательской части приложения, визуализируется интерактивный и понятный интерфейс, выделяются сервисы, отвечающие за обмен данных с серверной частью приложения.

За логику, работоспособность и правильное функционирование сайта отвечает серверная часть, которая скрыта от пользователя. На серверной части web-приложения, где обрабатываются пользовательские запросы, в рамках архитектуры, располагаются следующие части: модуль общения с базой данных, контроллеры.

На рисунке 2.5 представлена архитектура приложения.

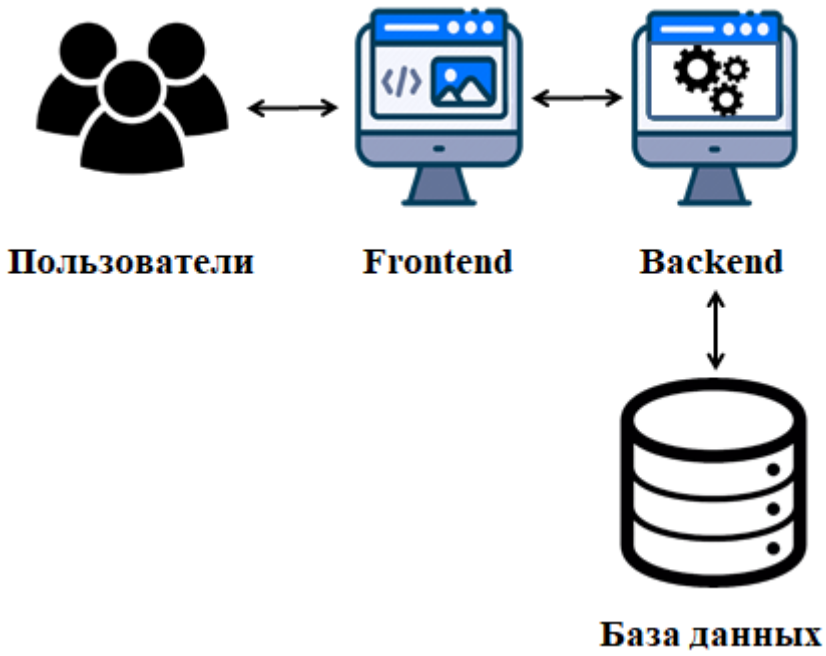


Рис. 2.5 – Архитектура приложения.

Вывод

В данном разделе были спроектированы база данных и архитектура приложения.

3. Технологическая часть

3.1 Выбор и обоснование инструментов разработки

Путём анализа современных трендов web-разработки было выявлено, что наиболее выигрышным является подход к созданию web-приложений, в виде SPA (Single Page Application – одностраничное веб-приложение, которое загружается на одну HTML-страницу).

Целесообразность использования SPA определяется следующими пунктами:

- Одностраничные приложения работают значительно быстрее обычных сайтов. Скорость загрузки у них выше, соответственно, они удобнее пользователям.
- Есть необходимость в многофункциональном, насыщенном пользовательском интерфейсе.
- SPA лучше адаптировано под мультиплатформенность: такое веб-приложение отлично показывает себя на любых устройствах и браузерах.

Динамическое обновление страницы, происходящее благодаря скриптам написанным на языке JavaScript, позволяет пользователю не перезагружать страницу при переходе к другому блоку приложения. В процессе работы пользователю может показаться, что он использует не веб-сайт, а десктопное приложение, так как оно мгновенно реагирует на все его действия, без задержек и «подвисаний». Добиться такого эффекта позволяет использование фреймворков для языка JavaScript, таких как: Angular, React, Vue.

Учитывая, что Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизацию [3], было принято решение использовать именно его.

Чтобы избежать ошибки, с которыми часто сталкиваются разработчики на JavaScript, используется дополнительное расширение JavaScript – TypeScript, которое позволяет работать со статической типизацией [4].

Для реализация серверной части был выбран Nest JS, который является Node JS фреймворком, написанным на TypeScript и внешне похожим на Angular, так как поддерживает микросервисную архитектуру и позволяет создавать большие масштабируемые приложения [5].

Для поддержки аналитической работы с данными необходимо использование визуального анализа, который позволяет представить большие объёмы данных в удобной для восприятия форме. Карта является хорошим инструментом для наглядного изучения. В приложении используется интерактивная карта Yandex Maps [6] для отображения количества пострадавших в ДТП. Преимущества Yandex Maps:

- присутствует республика Крым;
- границы субъектов не наезжают друг на друга;
- Чукотский край не разрезает по 180 меридиану.

Для автодополнения места совершения ДТП, в составлении протоколов используется сервис Dadata [7], так как решение бесплатное и подключается без дополнительных JS библиотек.

3.2 Реализация моделей хранения данных

Используя TypeOrm [8], Nest JS предоставляет функционал для интеграции СУБД приложениями в объектно-ориентированном стиле.

В листинге 1 представлена модель базы данных на примере таблицы ДТП.

Листинг 1 – Модель базы данных на примере таблицы ДТП.

```
import { Entity, PrimaryGeneratedColumn,
Column, OneToMany, ManyToMany, JoinTable } from 'typeorm';
import { AffectedDrivers } from '../entities/affecteddrivers.entity';
import { AffectedOthers } from '../entities/affectedothers.entity';
import { Typedtp } from '../entities/typedtp.entity';

@Entity('dtp')
export class Dtp {
    @PrimaryGeneratedColumn()
    dtpId: number;

    @Column({ type: 'date'})
    dateDtp: String;

    @Column({ type: 'time without time zone' })
    timeDtp: String;

    @Column()
    regionDtp: String;

    @Column()
    cityDtp: String;

    @Column({ nullable: true })
    descriptionDtp: String;

    @OneToMany(type => AffectedDrivers, affecteddrivers => affecteddrivers.dtp)
    affecteddrivers: AffectedDrivers[];

    @OneToMany(type => AffectedOthers, affectedothers => affectedothers.dtp)
    affectedothers: AffectedDrivers[];

    @ManyToMany(type => Typedtp)
    @JoinTable()
    dt: Typedtp[];
}
```

3.4 Реализация контроллера

Контроллеры отвечают за обработку входящих запросов и возврат ответов клиенту. Целью контроллера является получение конкретных запросов на приложение. Механизм маршрутизации контролирует, какой контроллер получает, какие запросы. Часто каждый контроллер имеет более одного маршрута, и различные маршруты могут выполнять различные действия.

Для создания базового контроллера Nest JS использует классы и декораторы. Декораторы связывают классы с требуемыми метаданными и позволяют Nest связать запросы с соответствующими контроллерами.

В листинге 2 представлена часть реализации контроллера для взаимодействия с данными ДТП.

Листинг 2 – Часть реализации контроллера для взаимодействия с данными ДТП.

```
import { Controller, Delete, Get, Param, Post, Put, Request, UseGuards } from '@nes
tjs/common';
import { DtpService } from './dtp.service';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';

@Controller('dtp')
export class DtpController {
    constructor(private dtpService: DtpService) {}

    @Get()
    @UseGuards(JwtAuthGuard)
    async findAll(@Request() req) {;
        return await this.dtpService.find(req.user.role);
    }

    @Post()
    @UseGuards(JwtAuthGuard)
    async create(@Request() req) {
        return await this.dtpService.create(req.body, req.user.role);
    }

    @Put('description/:id')
    @UseGuards(JwtAuthGuard)
    async updateDescription(@Param('id') id: number, @Request() req) {
       return await this.dtpService.updateDescription(id, req.body, req.user.role);
    }

    @Delete(':id')
    @UseGuards(JwtAuthGuard)
    async remove(@Param('id') id: number, @Request() req) {
        return await this.dtpService.remove(id, req.user.role)
    }
}
```

3.5 Реализация бизнес логики

Сервисы являются фундаментальной концепцией в Nest JS, вся бизнес логика реализована в них. Сервис - это обычный класс, используемый в контексте Nest JS для хранения глобального состояния приложения или в качестве поставщика данных. Контроллеры должны обрабатывать запросы HTTP и делегировать задачи сервисам.

В листинге 3 представлена часть реализации сервиса для взаимодействия с данными ДТП на примере метода создания происшествия.

Листинг 3 – Часть реализации сервиса для взаимодействия с данными ДТП на примере метода создания происшествия.

```
import { Injectable } from '@nestjs/common';
import { Dtp } from './entities/dtp.entity';
import { Connection } from "typeorm";
import { CreateDtpDto } from './dto/create-dtp';
import { ConfigService } from 'src/config.service';

@Injectable()
export class DtpService {

    constructor(private configs: ConfigService) {}

    async create(newDtp: CreateDtpDto, role: string): Promise<Dtp> {
        // подключение к БД под определенной ролью
        const connection = await this.configs.getConnection(role);

        let res;
        try {
            const repositoryDtp = connection.getRepository(Dtp);
            const dtp = repositoryDtp.create(newDtp);
            res = await repositoryDtp.save(dtp);
        }
        finally {
            await connection.close();
        }

        return res;
    }
}
```

3.6 Администрирование БД

С использованием pgAdmin (программное обеспечение, предоставляющее графический интерфейс для работы с базой данных) было проверено корректное создание базы данных с необходимыми ключами.

На рисунке 3.1 продемонстрировано существование таблиц в БД.

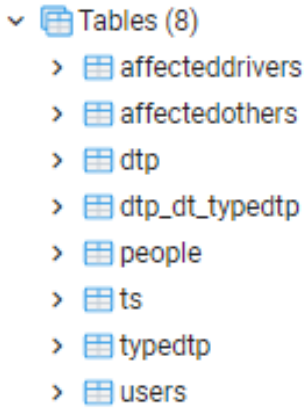


Рис. 3.1 – Таблицы БД.

Для реализации поставленных задач необходимо наличие ролевой модели на уровне базы данных. Роль – это разрешение, предоставляемое группе пользователей для доступа к данным. Было выделено три роли:

- reader – может просматривать данные таблиц;
- editor – может просматривать, редактировать данные таблиц, кроме users;
- admin(postgres) – может добавлять, редактировать данные всех таблиц.

В листинге 4 представлена реализация ролевой модели на уровне базы данных.

Листинг 4 – Реализация ролевой модели на уровне базы данных.

```
create user reader with password 'reader'
create user editor with password 'editor'

create role only_read;

GRANT select on dtp to only_read;
GRANT select on affecteddrivers to only_read;
GRANT select on affectedothers to only_read;
GRANT select on dtp_dt_typedtp to only_read;
GRANT select on people to only_read;
GRANT select on ts to only_read;
GRANT select on typedtp to only_read;

GRANT only_read TO reader;

create role only_editor;

GRANT select on dtp to only_editor;
GRANT select on affecteddrivers to only_editor;
GRANT select on affectedothers to only_editor;
GRANT select on dtp_dt_typedtp to only_editor;
GRANT select on people to only_editor;
GRANT select on ts to only_editor;
GRANT select on typedtp to only_editor;

GRANT insert on dtp to only_editor;
GRANT insert on affecteddrivers to only_editor;
GRANT insert on affectedothers to only_editor;
GRANT insert on dtp_dt_typedtp to only_editor;
GRANT insert on people to only_editor;
GRANT insert on ts to only_editor;

GRANT delete on dtp to only_editor;
GRANT delete on affecteddrivers to only_editor;
GRANT delete on affectedothers to only_editor;
GRANT delete on dtp_dt_typedtp to only_editor;
GRANT delete on people to only_editor;
GRANT delete on ts to only_editor;

GRANT update on dtp to only_editor;
GRANT update on affecteddrivers to only_editor;
GRANT update on affectedothers to only_editor;
GRANT update on dtp_dt_typedtp to only_editor;
GRANT update on people to only_editor;
GRANT update on ts to only_editor;

GRANT only_editor TO editor;
```

Для корректности данных в таблицах БД были реализованы триггеры Before для таблиц (рисунок 3.2):

- people, для добавления или обновления данных человека, необходимо соблюдать условия достижения 16-летнего возраста, если у него есть водительское удостоверение, серия и номер которого должны быть уникальными в системе;
- ts, регистрационный номер транспортного средства должен быть уникальным в системе;
- users, если администратор остался один в системе, то его нельзя удалить или установить ему права сотрудника.

Рис. 3.2 – Триггеры БД.

В листинге 5 представлена реализация триггеров Before для вставки и удаления строки для таблицы people.

Листинг 5 – Реализация триггеров Before для вставки и удаления строки для таблицы people.

```
CREATE FUNCTION trigger_create_person () RETURNS trigger AS
$$
        BEGIN
        if (NEW.driverlicense is NOT NULL) then
                if (SELECT date_part('year',age(NEW.birthdate)) >= 16)  then
                   if ((SELECT count(*) from people where people.driverlicense = NEW.driverlicense) >
0) then
                                return NULL;
                        else
                        return NEW;
                        end if;
                else
                        return NULL;
                end if;
        else
          return NEW;
   end if;
END;
$$ LANGUAGE  plpgsql;

CREATE TRIGGER createPerson
BEFORE INSERT ON people FOR EACH ROW
EXECUTE PROCEDURE trigger_create_person ()

-- Обновление данных о человеке
CREATE FUNCTION trigger_update_person () RETURNS trigger AS
$$
        BEGIN
        if (NEW.driverlicense is NOT NULL) then
                if (SELECT date_part('year',age(NEW.birthdate)) >= 16)  then
                   if ((SELECT count(*) from people where people.driverlicense = NEW.driverlicense
and people.id <> NEW.id) > 0) then
                                return OLD;
                        else
                        return NEW;
                        end if;
                else
                        return OLD;
                end if;
        else
          return NEW;
   end if;
END;
$$ LANGUAGE  plpgsql;


CREATE TRIGGER updatePerson
BEFORE UPDATE ON people FOR EACH ROW
EXECUTE PROCEDURE trigger_update_person ()
```

Для получения статистики количества погибших и раненных в ДТП были реализованы специальные функции (рисунок 3.3).

Рис. 3.3 – Функции БД.

В листинге 6 представлена реализация функции для получения количества ДТП в регионе, отфильтрованных по дате и типу ДТП.

Листинг 6 – Реализация функции для получения количества ДТП в регионе, отфильтрованных по дате и типу ДТП.

```
CREATE OR REPLACE function getCountDtpWithDateAndCategory(minDate date, maxDate date,
category integer)
returns table (
    region character varying,
    count bigint
          )
    as $$
    begin
          return query(
                select dtp."regionDtp" as region, count(*) as countDtp
                from dtp join dtp_dt_typedtp on dtp_dt_typedtp."dtpDtpId"=dtp."dtpId"
                join typedtp on typedtp.id = dtp_dt_typedtp."typedtpId"
                where dtp."dateDtp" >= $1 and dtp."dateDtp" <= $2 and typedtp.id = $3
                group by dtp."regionDtp"
          );
    end;
$$
language 'plpgsql';
```

3.7 Интерфейс приложения

Интерфейс приложения разбит на экраны, На рис.3.4 – рис.3.14 показан интерфейс различных экранов приложения.

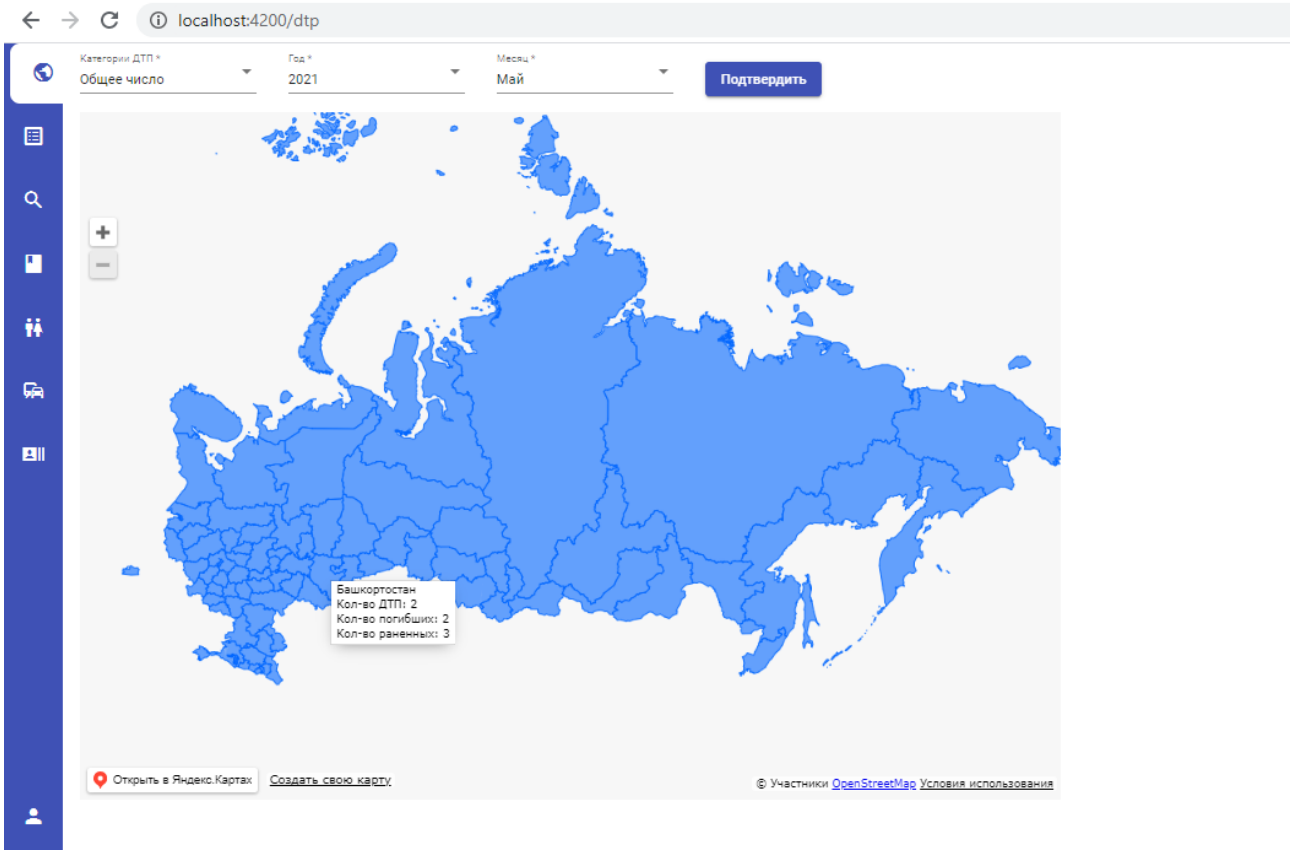


Рис. 3.4 – Интерфейс экрана для сбора статистики погибших и раненных с помощью карты.

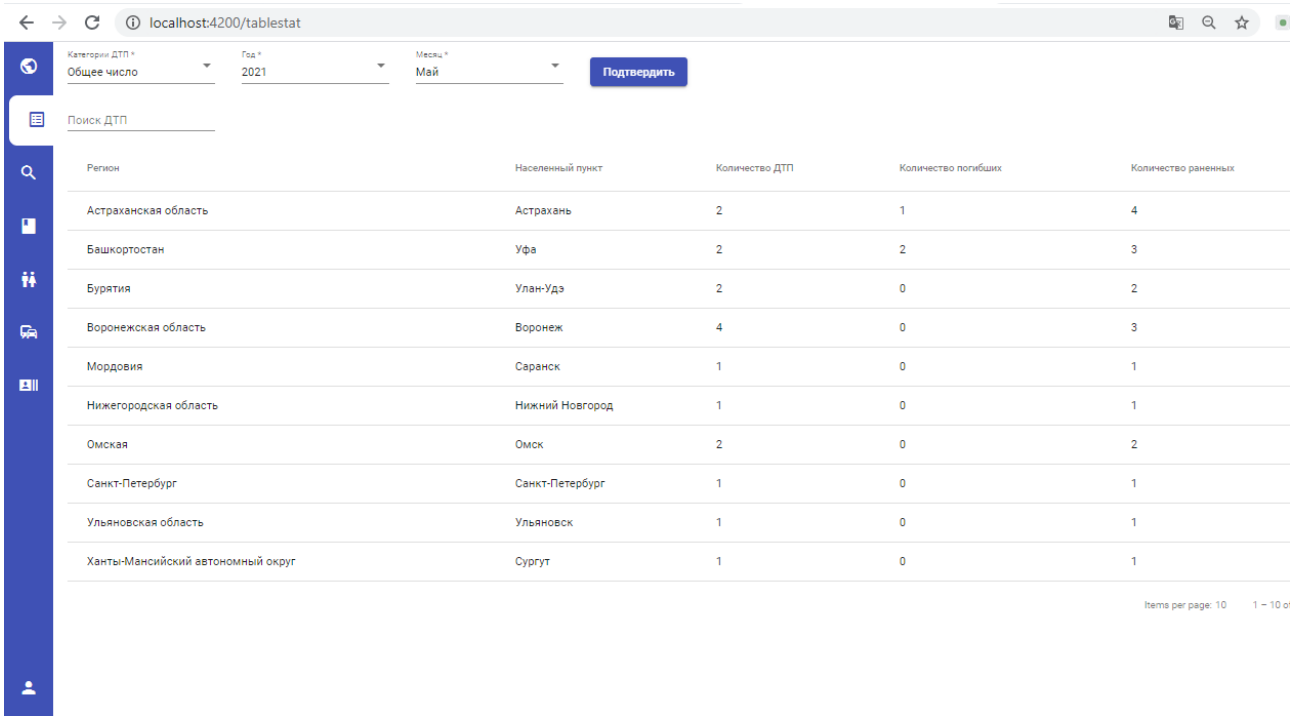


Рис. 3.5 – Интерфейс экрана для сбора статистики погибших и раненных с помощью таблицы.

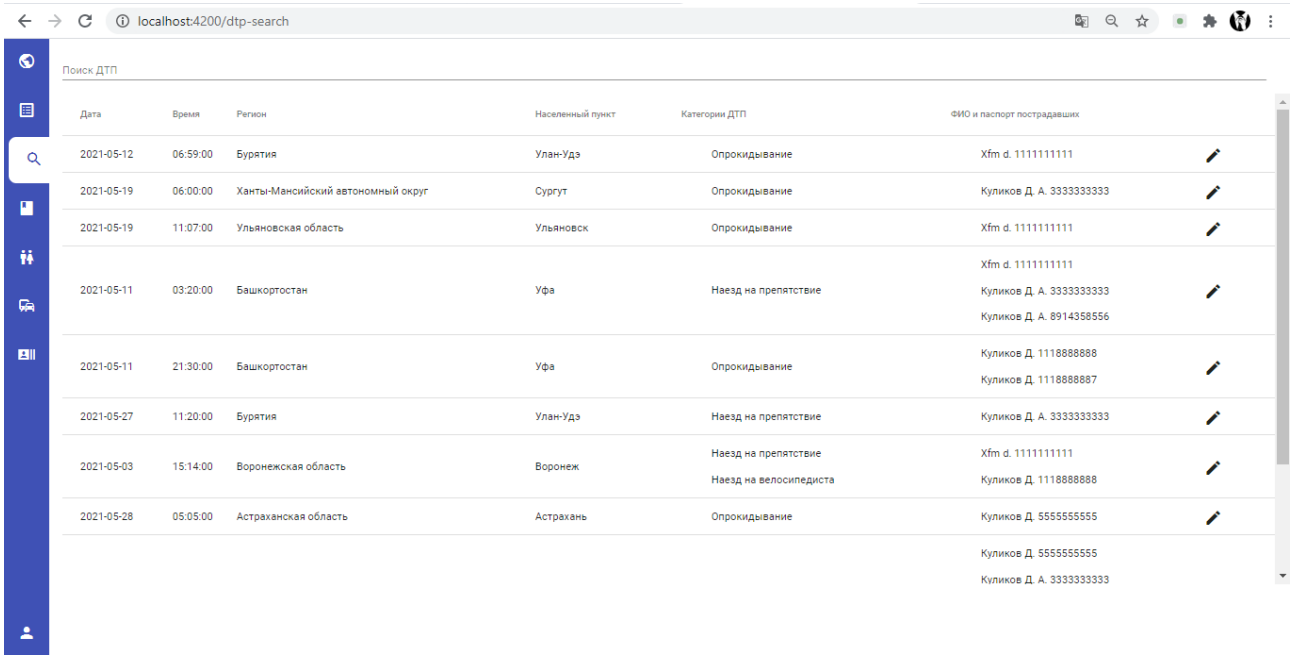
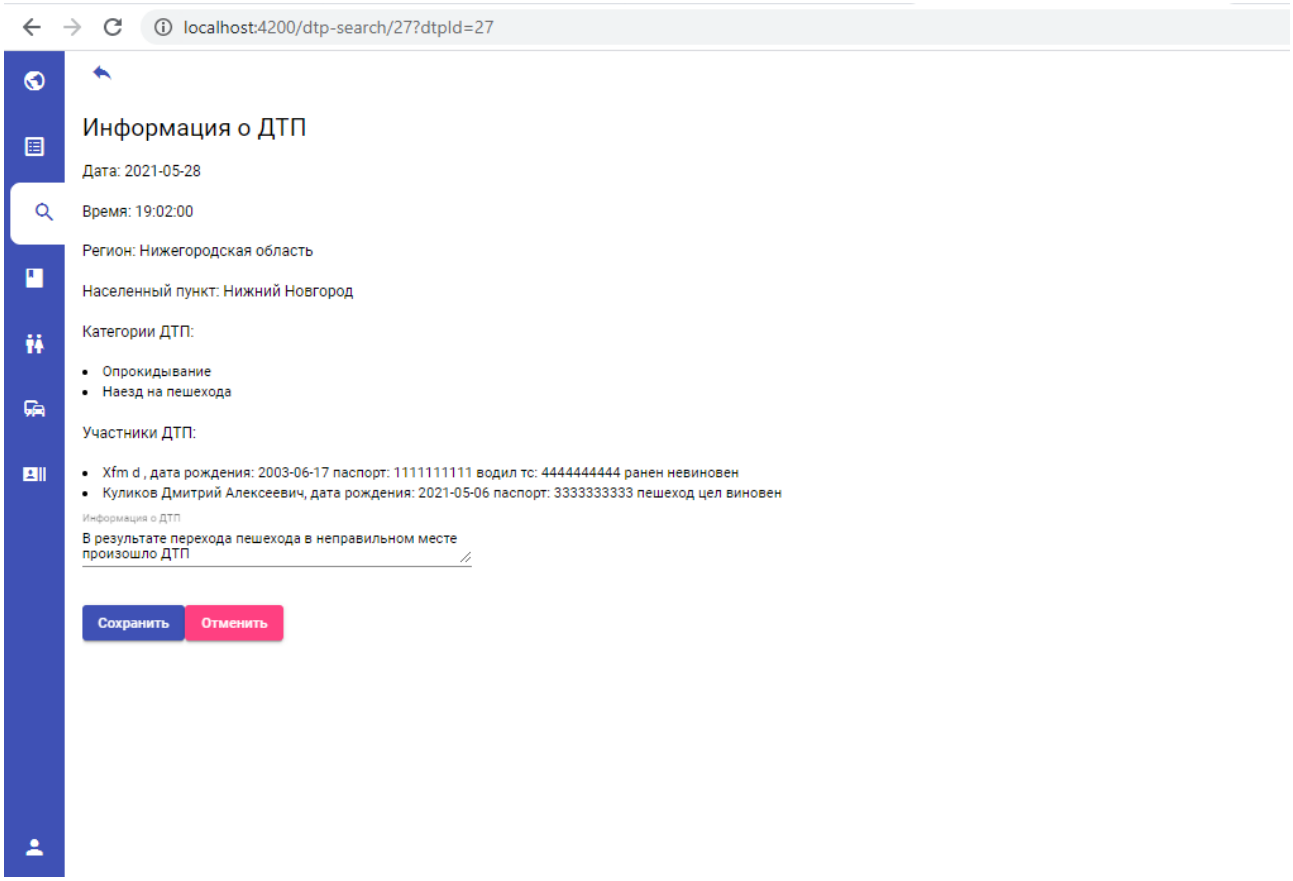
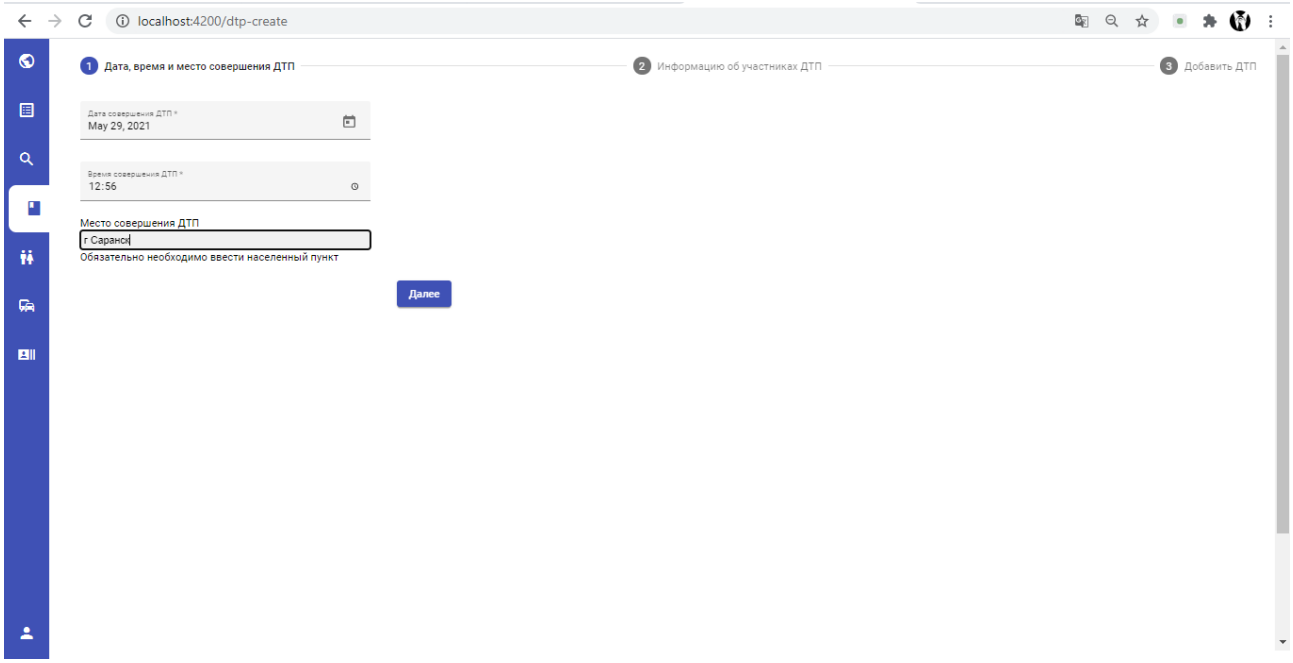


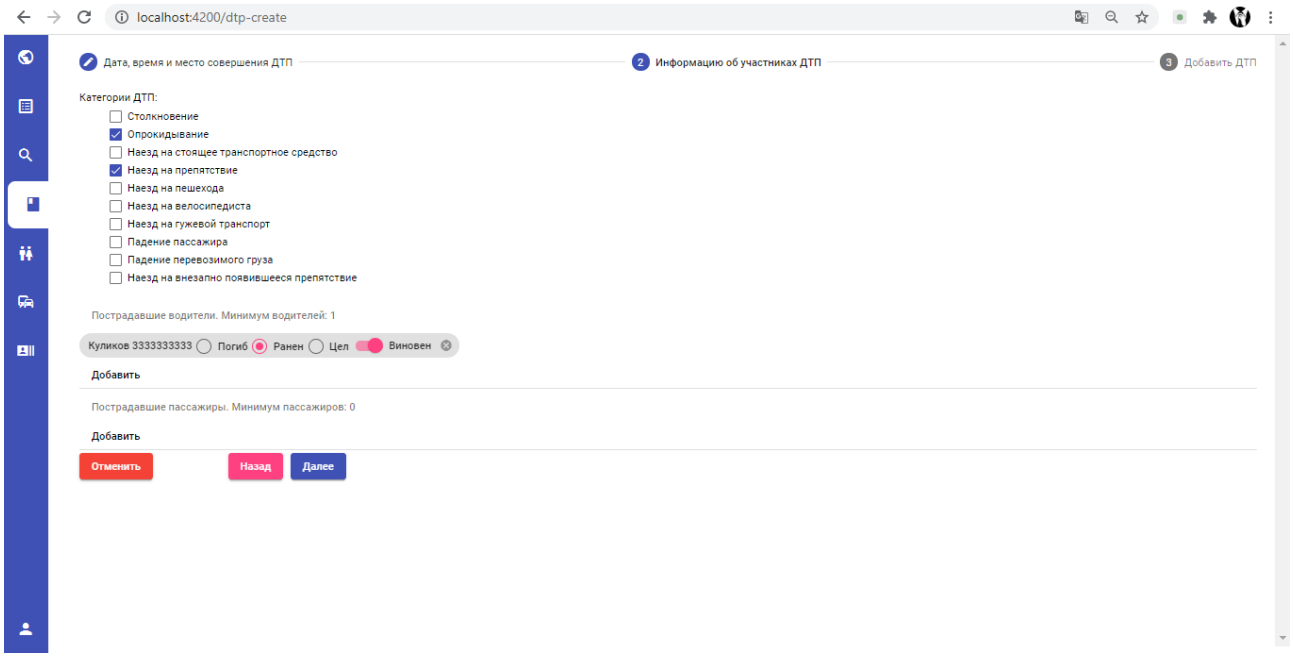
Рис. 3.6 – Интерфейс экрана для поиска нужного ДТП.



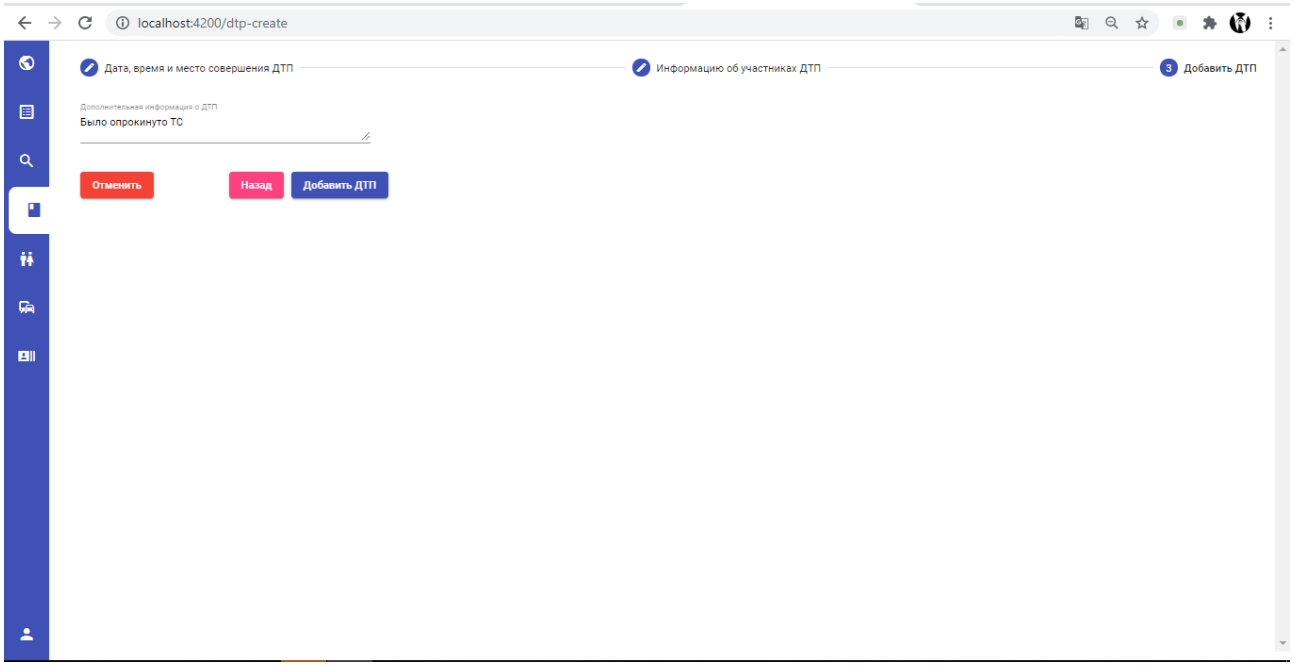
На рис. 3.7 — Интерфейс экрана информации о ДТП.



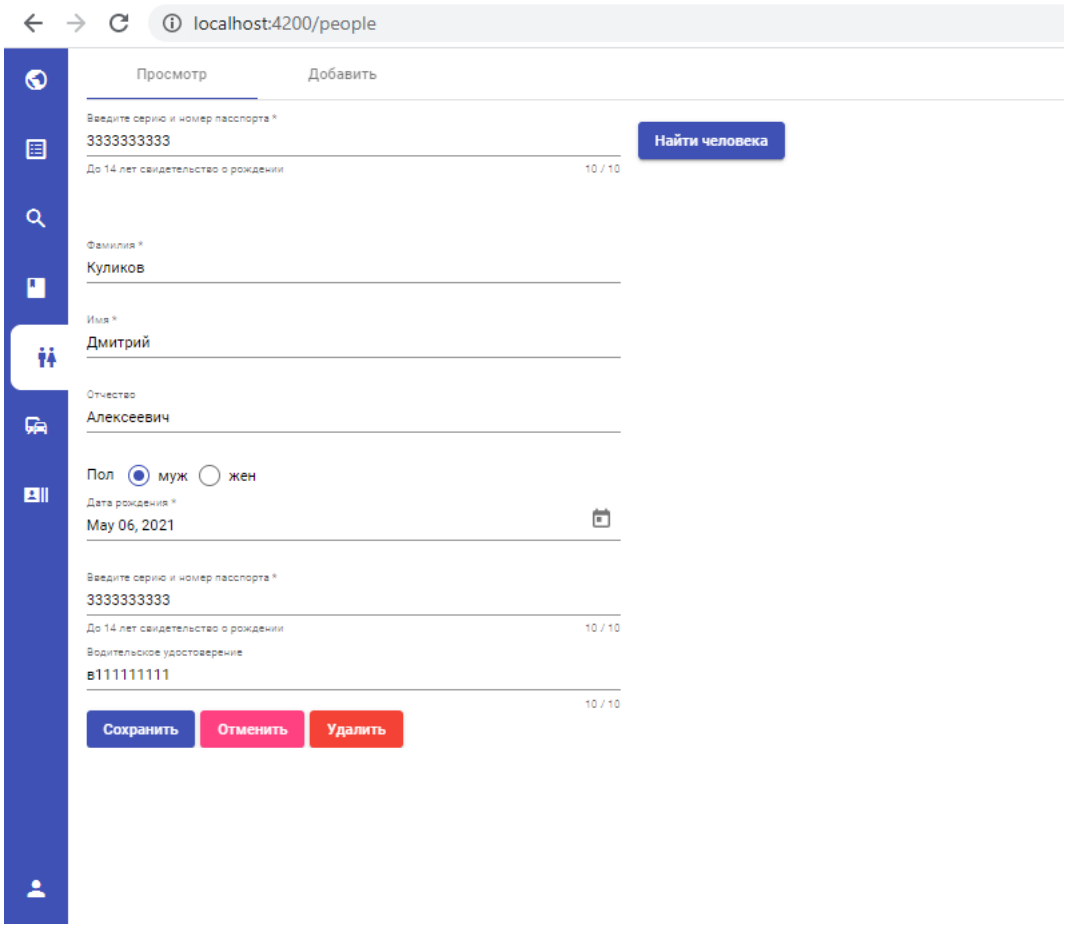
На рис. 3.8 — Интерфейс экрана составление протокола ДТП. Ввод даты, времени и места происшествия.



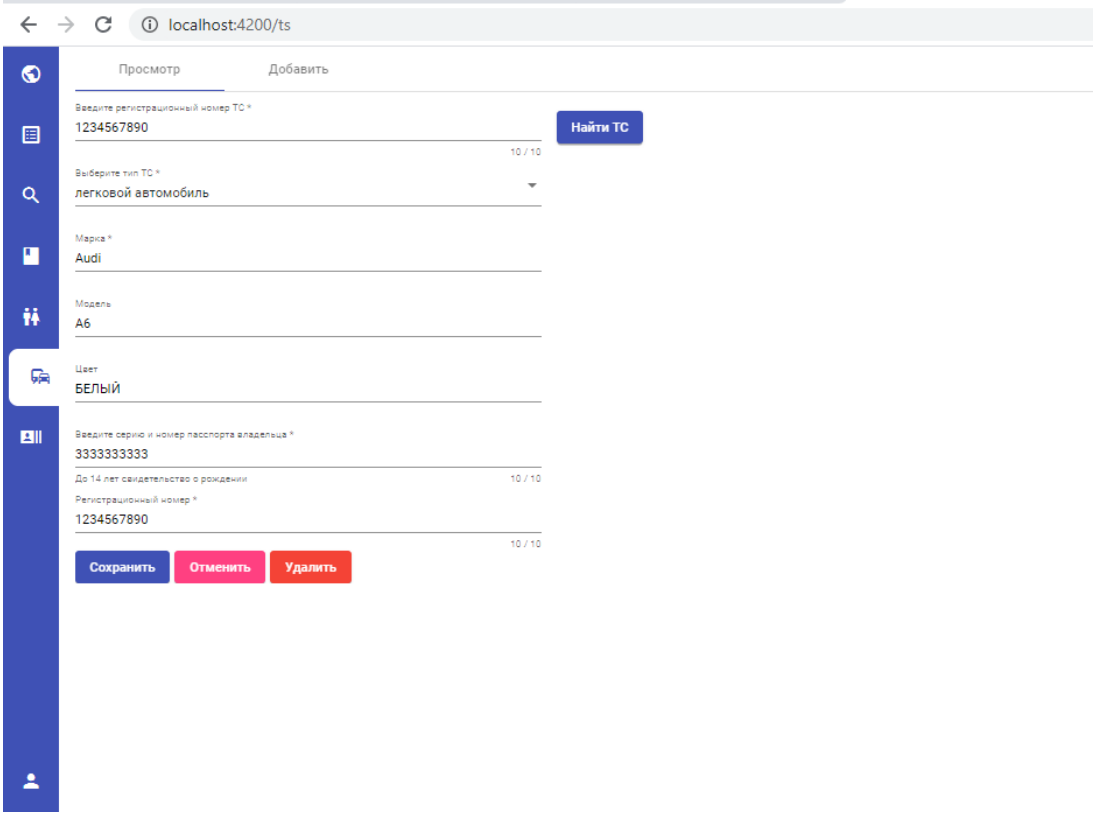
На рис. 3.9 — Интерфейс экрана составление протокола ДТП. Ввод данных об участниках происшествия.



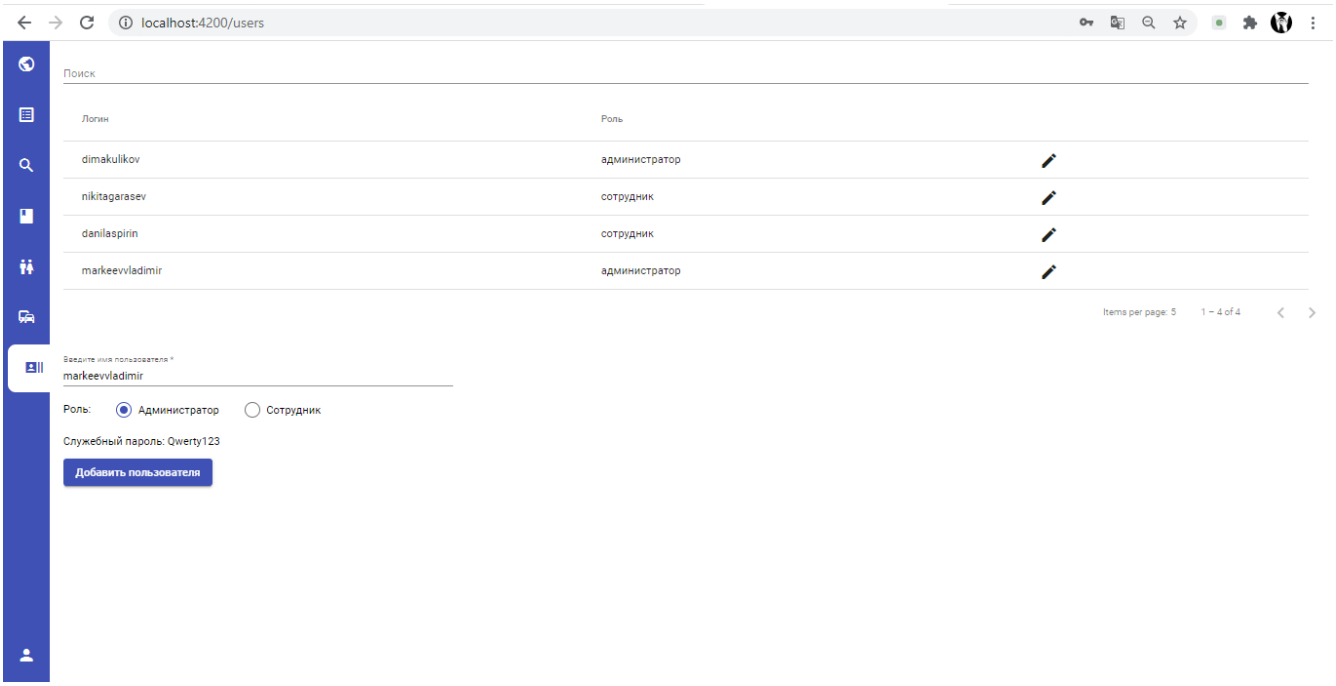
На рис. 3.10 — Интерфейс экрана составление протокола ДТП. Ввод дополнительных сведений происшествия.



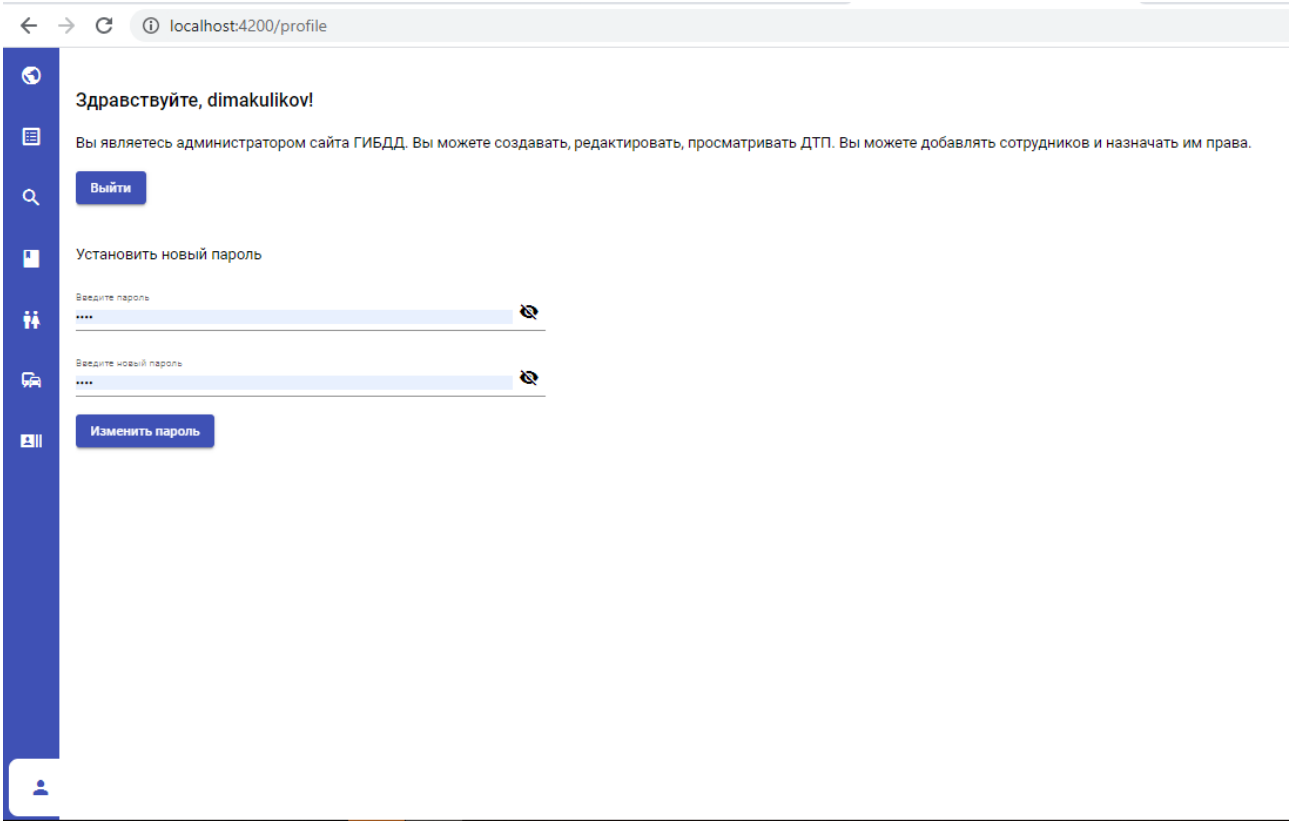
На рис. 3.11 — Интерфейс экрана поиска сведений о человеке.



На рис. 3.12 — Интерфейс экрана поиска сведений о ТС.



На рис. 3.13 — Интерфейс экрана поиска сведений об авторизованных пользователях.



На рис. 3.14 — Интерфейс экрана профиля пользователя.

Вывод

В данном разделе был выбран стек технологий, рассмотрены структуры и состав реализованных классов, предоставлены сведения о модулях приложения и об его интерфейсе.

Заключение

Цель курсовой работы достигнута. Спроектировано и реализовано программное обеспечение для сбора статистики по дорожно-транспортным происшествиям Российской Федерации.

В ходе работы была формализована задача, определен необходимый функционал, рассмотрены существующие виды СУБД, проведен анализ РСУБД, описана структура базы данных и приложения, изучены возможности языка TypeScript, фреймворков Angular и NestJS, получен опыт работы с картами и PostgreSQL.

Список литературы

1. ISO/IEC TR 10032:2003 Information technology — Reference model of data management.
2. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006.
3. Официальный сайт Angular, документация. [Электронный ресурс] – Режим доступа: https://angular.io/docs, свободный – (дата обращения: 20.03.21)
4. Основы TypeScript, необходимые для разработки web-приложений [Электронный ресурс]. – Режим доступа: https://metanit.com/web/typescript/1.1.php, свободный – (дата обращения: 20.03.21)
5. Официальный сайт Nest JS, документация. [Электронный ресурс] – Режим доступа: https://docs.nestjs.com/, свободный – (дата обращения: 02.04.21)
6. Официальный сайт Yandex Maps, документация. [Электронный ресурс] – Режим доступа: https://yandex.ru/dev/maps/, свободный – (дата обращения: 15.04.21)
7. Официальный сайт Dadata, документация. [Электронный ресурс] – Режим доступа: https://dadata.ru/, свободный – (дата обращения: 18.05.21)
8. Официальный сайт TypeOrm, документация. [Электронный ресурс] – Режим доступа: https://typeorm.io/#/, свободный – (дата обращения: 02.04.21)