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

ФАКУЛЬТЕТ ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА ИУ7 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных для сервиса отслеживания маршрутов автомобилей по данным с камер

Студент	**ИУ7-62Б**		**Нисуев**
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			**Тассов К.Л.**
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка к курсовой работе содержит n страниц, 10 иллюстраций, 15 таблицы, 12 источников, 1 приложение.

Целью работы является разработка базы данных для сервиса отслеживания маршрутов автомобилей по данным с камер, а также приложения для доступа к ней.

Ключевые слова: база данных, система управления базами данных, PostgreSQL.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В данной расчетно-пояснительной записке используются следующие сокращения и обозначения.

БД — база данных.

СУБД — система управления базами данных.

ПО — программное обеспечение.

ВУ — водительское удостоверение.

ТС — транспортное средство.

СТС — свидетельство о регистрации транспортного средства.

ПТС — паспорт транспортного средства.

ГРЗ — государственный регистрационный знак.

ВВЕДЕНИЕ

Отслеживание маршрутов автомобилей с использованием данных с камер представляет собой важную задачу, направленную на повышение безопасности дорожного движения, предотвращение краж транспортных средств и других правонарушений. Для эффективного сбора, хранения и анализа информации, поступающей с камер, требуется разработка базы данных и приложения, обеспечивающего доступ к ней. Такая система позволит отслеживать службам безопасности маршруты подозрительных автомобилей, а также даст возможность простым пользователям просматривать маршруты своих автомобилей в случае угона.

Цель работы: разработка базы данных для сервиса отслеживания маршрутов автомобилей по данным с камер, а также приложения для доступа к ней.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать существующие решения и предметную область;
- сформулировать требования к разрабатываемой базе данных и приложению;
- спроектировать сущности и ограничения базы данных;
- выбрать средства реализации базы данных и программного обеспечения;
- реализовать базу данных и программное обеспечения для доступа к ней;
- провести исследования зависимости времени вставки данных в базу данных от метода добавления данных.

1 Аналитическая часть

В этом разделе представлен анализ предметной области, а также сравнительный обзор существующих решений. Формулируются требования к разрабатываемой базе данных и приложению, проводится формализация и описание данных, которые будут храниться в проектируемой БД. Кроме того, анализируются существующие модели баз данных, приводится диаграмма сущность-связь проектируемой системы, описываются типы пользователей и строится диаграмма прецедентов.

1.1 Анализ предметной области

Современные транспортные системы требуют эффективных инструментов для контроля и анализа перемещений автомобилей. Города активно оснащаются разветвленными сетями камер видеонаблюдения, которые в режиме реального времени фиксируют движение транспорта. Сервис, основанный на данных с уличных камер видеонаблюдения, позволяет точно отслеживать маршруты транспортных средств, обеспечивая безопасность дорожного движения и улучшение городского управления.

Технологической основой таких сервисов являются решения для автоматического распознавания номеров (Automatic License Plate Recognition, ALPR), которые позволяют идентифицировать транспортные средства по их государственным регистрационным знакам. Современные ALPR-системы достигают точности распознавания до 96.2% даже в сложных условиях освещения и при различных углах обзора камер [1].

1.2 Анализ существующих решений

В качестве существующих решений были выбраны Flock Safety [2], Plate Recognizer [3] и City Point [4].

Для сравнения существующих решений были выбраны следующие критерии:

— КР1 — история перемещений;

— КР2 — интеграция с городскими камерами;

— КР3 — гибкие фильтры поиска;

— КР4 — визуализация маршрутов на карте;

— КР5 — мобильное приложение;

— КР6 — возможность отслеживания личного автомобиля для пользователей.

Сравнение существующих решений представлено в таблице 1.1.

Таблица 1.1 – Сравнение существующих решений

Решение	**КР1**	**КР2**	**КР3**	**КР4**	**КР5**	**КР6**
Flock Safety	+	+	+	+	-	-
Plate Recognizer	+	+	+	-	-	-
City Point	+	+	+	+	-	+

По результатам сравнения, ни одно из рассмотренных решений не удовлетворяет всем выдвинутым критериям.

1.3 Анализ моделей баз данных

База данных (БД) — совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения иманипулирования данными, независимая от прикладных программ [5].

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Упомянутые объекты позволяют моделировать структуру данных, а операторы — поведение данных [6].

По модели данных базы данных разделяют на следующие категории: дореляционные, реляционные и постреляционные.

1.3.1 Дореляционные базы данных

Являются предшественниками реляционных баз данных. Такие системы не основывались на каких-либо абстрактных моделях, а также такие системы не основывались на каких-либо абстрактных моделях [7].

Основным преимуществом данных моделей является возможность построения вручную эффективных прикладных систем.

Недостатками данных моделей являются необходимость знания о физической организации, а также перегруженность деталями организации доступа.

1.3.2 Реляционные базы данных

Реляционная база данных – это набор данных с заданными взаимосвязями.

Реляционная модель объединяет данные в таблицы, где каждая строка представляет собой отдельную запись, а каждый столбец состоит из атрибутов, содержащих значения [6].

Основными преимуществами данной модели являются:

— поддержка языка запросов SQL;

— отображение информации в наиболее простой для пользователя форме;

— представление сущностей единым образом в виде отношений.

Основным недостатком данной модели является медленный доступ к данным среди всех моделей данных;

1.3.3 Постреляционные базы данных

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся

в записях таблиц. Постреляционная модель допускает многозначные поля — поля значение которых состоит из подзначений. Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля.

Основным достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц в виде одной постреляционной таблицы.

Недостатками данной модели являются отсутствие стандартизированных языков запросов и общепринятой модели, а также сложность обеспечения целостности.

1.4 Выбор модели базы данных

Так как данные обладают строгой структурой, не подвержены частым изменениям, а также предполагается выполнение запросов различной сложности и необходимость обеспечения независимости хранения информации от приложения, было принято решение использовать реляционную модель базы данных.

1.5 Формализация ролей

Исходя из поставленной задачи и особенностей предметной области, следует выделить три основных типа пользователей:

- **пользователь** должен обладать следующими возможностями:
  - регистрация;
  - авторизация;
  - просмотр маршрутов автомобилей;
- **оператор** должен обладать следующими возможностями:
  - авторизация;
  - поиск маршрутов об отслеживании по критериям;

— просмотр маршрутов автомобилей;

— **аудит** должен обладать следующими возможностями:

— авторизация;

— поиск информации об отслеживании по критериям;

— просмотр информации об отслеживании.

На рисунке 1.1 представлена диаграмма прецедентов.

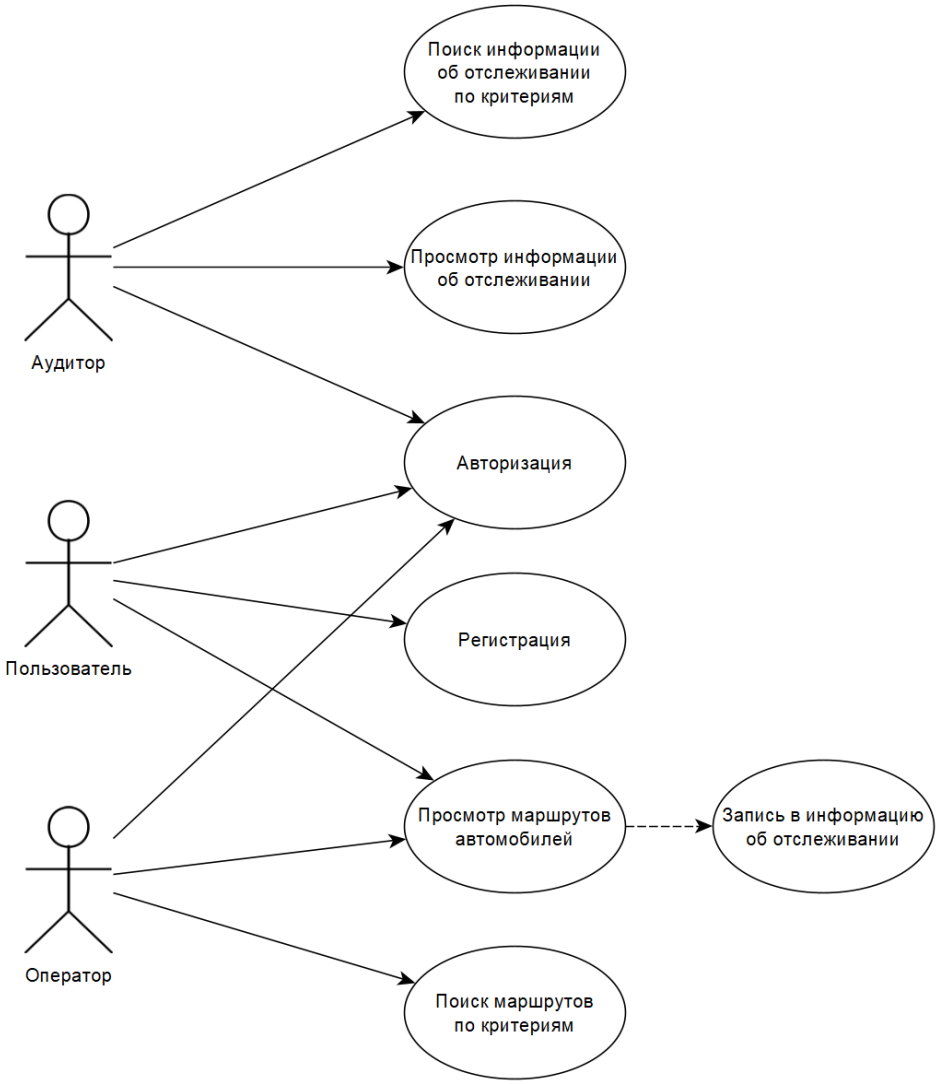

Рисунок 1.1 – Диаграмма прецедентов

1.6 Формализация данных

База данных должна содержать информацию о:

— пользователь;

— информация об отслеживании;

— транспортное средство (ТС);

— свидетельство о регистрации ТС (СТС);

— паспорт ТС (ПТС);

— владелец;

— история владельцев;

— камера;

— снимок.

Сведения о каждой сущности содержится в таблице 1.2.

Таблица 1.2 – Сведения о категориях данных

Сущность	**Информация**
Пользователь	ФИО, паспорт, логин, пароль, роль
Информация об отслеживании	Время, дата, дата маршрута
ТС	Цвет, VIN, марка, модель, тип кузова, пробег
СТС	Масса, тип двигателя, номер лошадиные силы, модель, марка, дата постановки, год выпуска, ГРЗ, VIN, категория ТС
ПТС	страна ввоза, номер
Владелец	Возраст, ФИО, ВУ, паспорт, стаж вождения
История владельцев	Дата постановки на учет, дата снятия с учета, пробег
Камера	Наличие радара, дата установки, ширина, долгота
Снимок	Полоса, время, дата, скрость, ГРЗ

На рисунке 1.2 приведена диаграмма сущность-связь в нотации Чена.

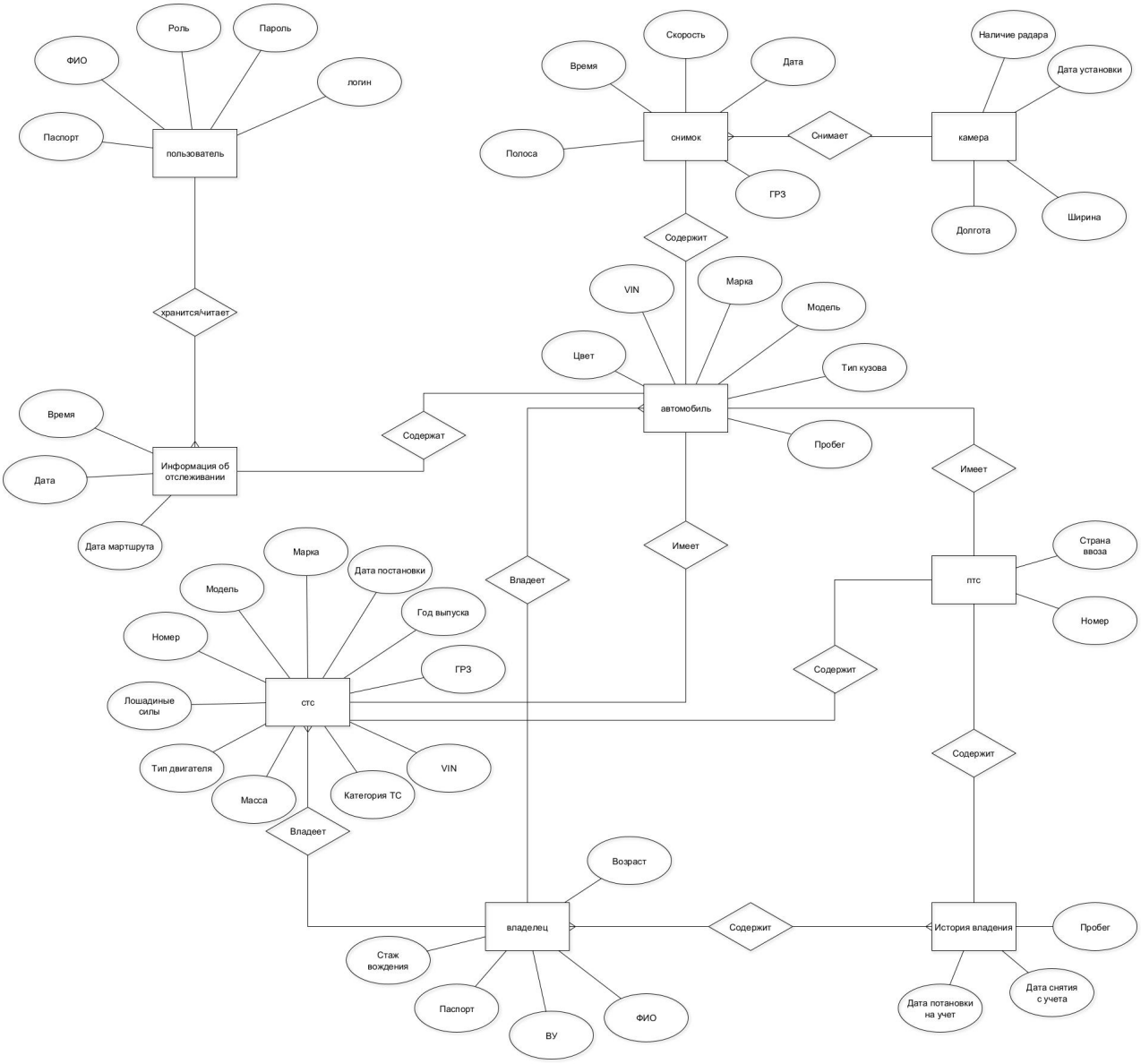

Рисунок 1.2 – Диаграмма сущность-связь

1.7 Формализация задачи

Требуется спроектировать базу данных для хранения сведений о пользователях, автомобилях, владельцах, снимках и камерах. Также необходимо разработать приложение с интерфейсом для просмотра данных, содержащихся в базе. Предусматривается реализация трех ролей с различными уровнями доступа: пользователь, оператор и аудитор.

ВЫВОД

В данном разделе представлен анализ предметной области и проведено сравнение существующих решений. Сформулированы требования к разрабатываемой базе данных и приложению, а также выполнена формализация и описание информации, подлежащей хранению в базе данных. Кроме того, рассмотрены существующие модели баз данных и представлена диаграмма сущность-связь для проектируемой системы. Также описаны типы пользователей и приведена диаграмма прецедентов.

2 Конструкторская часть

В данном разделе представлена диаграмма разрабатываемой базы данных, выполнено описание ее сущностей, определены ограничения целостности, а также описаны проектируемая функция и ролевые модели.

2.1 Описание сущностей и ограничений целостности

В реализуемой базе данных выделены следующие таблицы:

— AppUser — таблица пользователей;

— TrackInfo — таблица отслеживаний маршрутов автомобилей;

— Car — таблица автомобилей;

— CarOwner — таблица владельцев;

— STS — таблица СТС;

— PTS — таблица ПТС;

— OwnerHistory — таблица истории владения транспортными средствами;

— OwnerHistoryOwner — таблица-связь владельцев и историй владения;

— Camera — таблица дорожных камер;

— CarSnapshot — таблица снимков автомобилей.

На рисунке 2.1 представлена диаграмма базы данных.

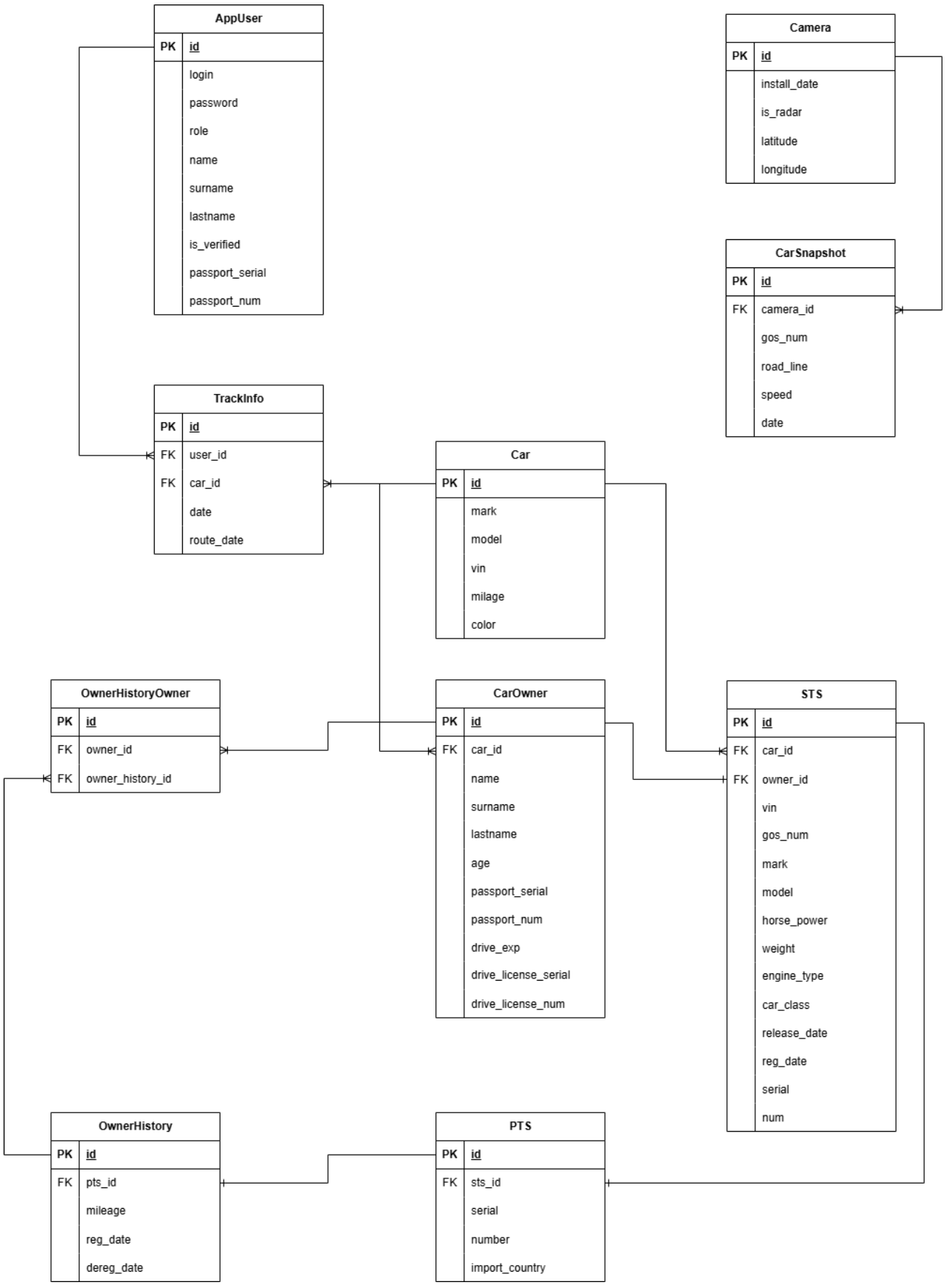

Рисунок 2.1 – Диаграмма базы данных

На таблицах 2.1 — 2.10 приведено описание столбцов таблиц базы данных.

Таблица 2.1 – Информация о столбцах таблицы AppUser

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор пользователя
firstname	Строка	ненулевой	Имя
surname	Строка	ненулевой	Фамилия
lastname	Строка		Отчество
role	Строка	ненулевой	Роль пользователя
login	Строка	ненулевой	Электронная почта, логин
password	Строка	ненулевой	Пароль
is_verified	Логический тип	ненулевой	Верифицирован ли пользователь
passport_serial	Целое число	ненулевой	Серия паспорта
passport_num	Целое число	ненулевой	Номер паспорта

Таблица 2.2 – Информация о столбцах таблицы TrackInfo

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор информации об отслеживании
user_id	Целое число	ненулевой, внешний ключ	id пользователя
car_id	Целое число	ненулевой, внешний ключ	id автомобиля
track_date	Дата	ненулевой	Дата просмотра маршрута
route_date	Дата	ненулевой	Дата просмотренного пути

Таблица 2.3 – Информация о столбцах таблицы Car

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой первичный ключ	Идентификатор автомобиля
mark	Строка	ненулевой	Марка
model	Строка	ненулевой	Модель
vin	Строка	ненулевой	ВИН-номер
milage	Строка	ненулевой положительное число	Роль пользователя
color	Строка	ненулевой	Цвет автомобиля

Таблица 2.4 – Информация о столбцах таблицы PTS

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор ПТС
sts_id	Целое число	ненулевой, внешний ключ	Идентификатор СТС
serial	Целое число	ненулевой	Серия ПТС
number	Целое число	ненулевой	Номер ПТС
import_country	Строка	ненулевой	Страна импорта транспортного средства

Таблица 2.5 – Информация о столбцах таблицы STS

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой первичный ключ	Идентификатор СТС
car_id	Целое число	ненулевой, внешний ключ	Идентификатор автомобиля
owner_id	Целое число	ненулевой, внешний ключ	Идентификатор владельца
vin	Строка	ненулевой	VIN-номер ТС
gos_num	Строка	ненулевой	Государственный решистрационный знак
mark	Строка	ненулевой	Марка ТС
model	Строка	ненулевой	Модель ТС
horse_power	Целое число	ненулевой, положительное число	Мощность (л.с.)
weight	Целое число	ненулевой, положительное число	Масса ТС (кг)
engine_type	Строка	ненулевой	Тип двигателя
car_class	Строка	ненулевой	Класс автомобиля
release_date	Дата	ненулевой	Дата выпуска
reg_date	Дата	ненулевой	Дата регистрации
serial	Целое число	ненулевой	Серия СТС
num	Целое число	ненулевой	Номер СТС

Таблица 2.6 – Информация о столбцах таблицы CarOwner

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой первичный ключ	Идентификатор владельца
car_id	Целое число	ненулевой внешний ключ	Идентификатор автомобиля
name	Строка	ненулевой	Имя
surname	Строка	ненулевой	Фамилия
lastname	Строка		Отчество
age	Целое число	ненулевой, положительное число	Возраст владельца
passport_serial	Целое число	ненулевой	Серия паспорта
passport_num	Целое число	ненулевой	Номер паспорта
drive_exp	Целое число	ненулевой, положительное число	Стаж вождения (в годах)
drive_ license_ serial	Целое число	ненулевой	Серия водительского удостоверения
drive_ license_ num	Целое число	ненулевой	Номер водительского удостоверения

Таблица 2.7 – Информация о столбцах таблицы OwnerHistory

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор истории владельцев
pts_id	Целое число	ненулевой, внешний ключ	Идентификатор ПТС
mileage	Целое число	ненулевой, положительное число	Пробег (км)
reg_date	Дата	ненулевой	Дата регистрации
dereg_date	Дата		Дата снятия с учета

Таблица 2.8 – Информация о столбцах таблицы OwnerHistoryOwner

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор
owner_id	Целое число	ненулевой, внешний ключ	Идентификатор владельца
owner_history_id	Целое число	ненулевой, внешний ключ	Идентификатор записи истории

Таблица 2.9 – Информация о столбцах таблицы Camera (Камеры)

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор камеры
install_date	Дата	ненулевой	Дата установки
is_radar	Логический тип	ненулевой	Есть ли радаром
latitude	Вещественное число	ненулевой	Широта
longitude	Вещественное число	ненулевой	Долгота

Таблица 2.10 – Информация о столбцах таблицы CarSnapshot (Снимки камер)

Столбец	Тип данных	Ограничения	Значение
id	Целое число	ненулевой, первичный ключ	Идентификатор снимка
camera_id	Целое число	ненулевой, внешний ключ	Идентификатор камеры
gos_num	Строка	ненулевой	ГРЗ ТС на снимке
road_line	Целое число	ненулевой, положительное число	Полоса движения
speed	Целое число	ненулевой, положительное число	Скорость (км/ч)
snap_ datetime	Дата и время	ненулевой	Дата и время снимка

2.2 Описание функции базы данных

В рамках базы данных будет присутствовать функция верификации пользователя. В качестве входных параметров функции подаются id пользователя, серия и номер паспорта. В случае успеха функция обновит паспортные

данные у пользователя с заданным id. На рисунке 2.2 представлена схема функции верификация пользователя.

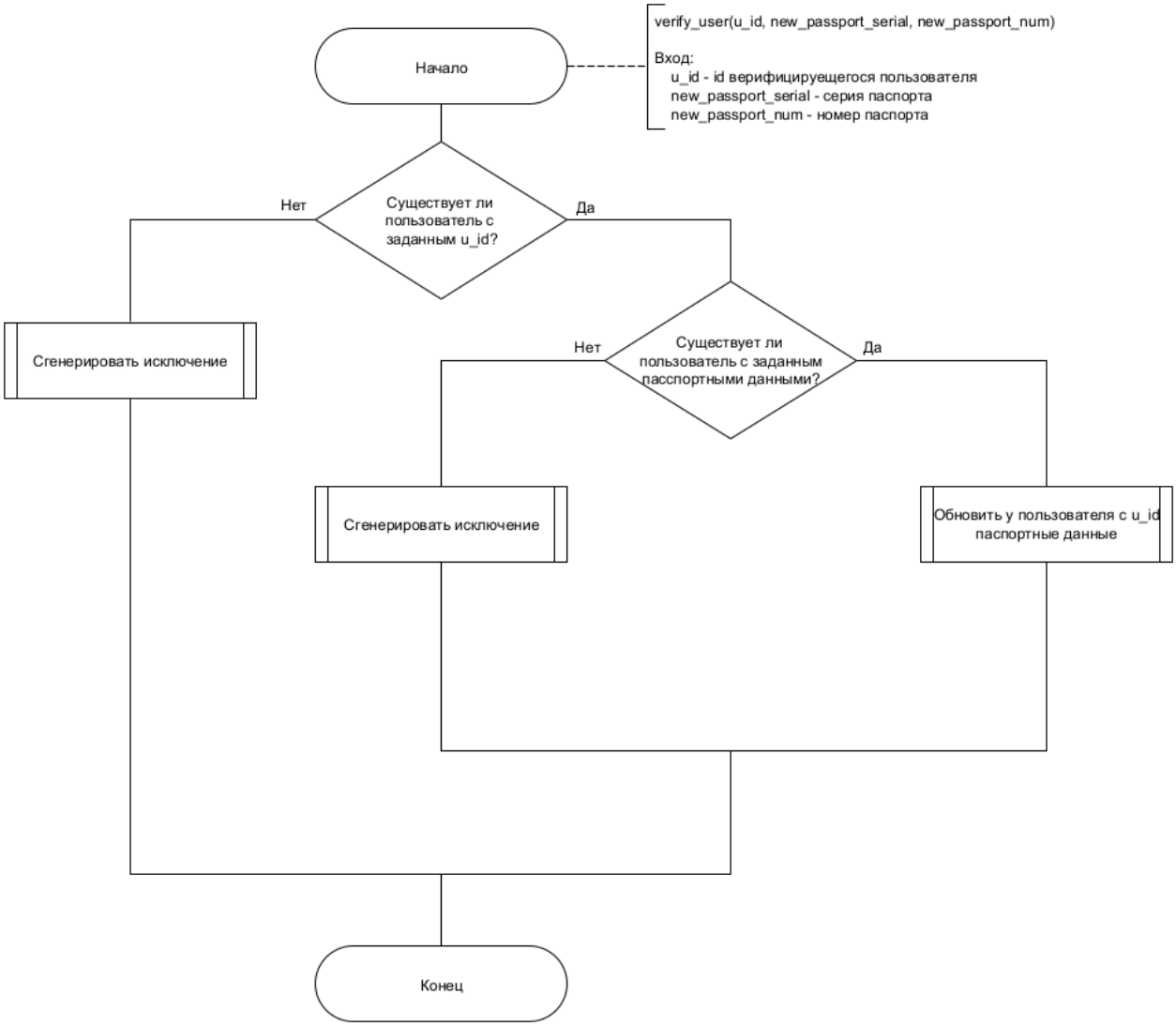

Рисунок 2.2 – Схема функции верификации пользователя

2.3 Описание ролей базы данных

Для обеспечения безопасности сервера баз данных необходимо определить следующие пользовательские роли:

1. Пользователь имеет права для чтения своих данных из таблиц автомобилей и снимков.

2. Оператор имеет права для чтения записей из всех таблиц кроме таблицы отслеживания маршрутов автомобилей.

3. Аудит имеет права для чтения записей из всех таблиц кроме таблиц камер и снимков.

ВЫВОД

В этом разделе представлена диаграмма разрабатываемой базы данных, выполнено описание ее сущностей и ограничений целостности, а также описаны проектируемая процедура и ролевые модели.

3 Технологическая часть

В этом разделе рассматривается процесс выбора инструментов для реализации базы данных и приложения. Описывается создание сущностей, заданных ограничений целостности и пользовательских ролей в разработанной базе данных. Также приведена реализация хранимой процедуры и результаты её тестирования. Дополнительно представлен интерфейс разработанного программного обеспечения.

3.1 Выбор СУБД

Наиболее распространенными СУБД реляционной модели БД являются:

— PostgreSQL [8];

— SQLite [9];

— MySQL [10];

— MSSQL [11].

Для сравнения СУБД были выделены следующие критерии:

— возможность создания ролевой модели;

— соответствие всем SQL стандартам ANSI/ISO [12];

— наличие Row-Level Security (RLS).

В таблице 3.1 представлены результаты сравнения наиболее распространенных реляционных СУБД по выбранным критериям.

Таблица 3.1 – Сравнение распространенных реляционных СУБД

	PostgreSQL	SQLite	MySQL	MSSQL
Создание ролей	+	–	±	+
Соответствие SQL стандартам	+	–	±	±
Row-Level Security	+	–	+	+

В рамках данной работы в качестве СУБД была выбрана PostgreSQL, так как она обладает достаточным функционалом для поставленной цели.

3.2 Средства реализации

Для реализации программного обеспечения выбран язык *Rust* [13]. Выбор был сделан на основании того, что данный язык поддерживает все типы данных, определенные на этапе проектирования, и предоставляет полный набор возможностей, необходимых для реализации поставленной задачи. Для реализации сервера был выбран фреймворк *axum* [14]. Для реализации компонента доступа к данным был выбран фреймворк *sqlx* [15].

Для реализации клиента был выбран язык *Swift* [16], который позволяет разрабатывать приложения под *iOS*. Для реализации интерфейса был выбран фреймворк *UIKit* [17], который включает стандартные компоненты пользовательского интерфейса, поддерживает обработку ввода и жестов, а также обеспечивает анимации и переходы между экранами.

3.3 Реализация

3.3.1 Создание таблиц

В листингах 3.1—3.10 приведены запросы для создания таблиц и ограничений целостности базы данных.

Листинг 3.1 – Создание таблицы AppUser

```
1 CREATE TABLE AppUser (
2     id SERIAL PRIMARY KEY,
3     login TEXT NOT NULL,
4     password TEXT NOT NULL,
5     role TEXT NOT NULL,
6     name TEXT NOT NULL,
7     surname TEXT NOT NULL,
8     lastname TEXT,
9     is_verified BOOLEAN DEFAULT FALSE,
```

```
10      passport_serial INTEGER,
11      passport_num INTEGER
12  );
13
14  ALTER TABLE AppUser
15      ADD CONSTRAINT unique_passport UNIQUE (passport_serial,
            passport_num),
16      ADD CONSTRAINT check_role CHECK (role IN ('user', 'operator',
            'audit')),
17      ADD CONSTRAINT unique_login UNIQUE (login),
18      ADD CONSTRAINT check_login_email CHECK (login ~
            '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
19      ADD CONSTRAINT check_passport_format CHECK (passport_serial
            BETWEEN 1 AND 9999 AND passport_num BETWEEN 1 AND 999999);
```

Листинг 3.2 – Создание таблицы Car

```
1  CREATE TABLE Car (
2      id SERIAL PRIMARY KEY,
3      owner_id INTEGER NOT NULL,
4      mark TEXT NOT NULL,
5      model TEXT NOT NULL,
6      vin TEXT NOT NULL,
7      mileage INTEGER NOT NULL,
8      color TEXT NOT NULL
9  );
10
11 ALTER TABLE Car
12     ADD FOREIGN KEY (owner_id) REFERENCES CarOwner(id) ON DELETE
           CASCADE,
13     ADD CONSTRAINT unique_vin UNIQUE (vin),
14     ADD CONSTRAINT check_mileage CHECK (mileage >= 0),
15     ADD CONSTRAINT check_vin_length CHECK (LENGTH(vin) = 17);
```

Листинг 3.3 – Создание таблицы CarOwner

```
1  CREATE TABLE CarOwner (
2      id SERIAL PRIMARY KEY,
3      name TEXT NOT NULL,
4      surname TEXT NOT NULL,
5      lastname TEXT,
6      age INTEGER NOT NULL,
```

```
7      drive_exp INTEGER NOT NULL,
8      passport_serial INTEGER NOT NULL,
9      passport_num INTEGER NOT NULL,
10     drive_license_serial INTEGER NOT NULL,
11     drive_license_num INTEGER NOT NULL
12 );
13
14 ALTER TABLE CarOwner
15     ADD CONSTRAINT unique_owner_passport UNIQUE (passport_serial,
           passport_num),
16     ADD CONSTRAINT unique_drive_license UNIQUE
           (drive_license_serial, drive_license_num),
17     ADD CONSTRAINT check_age CHECK (age >= 18),
18     ADD CONSTRAINT check_drive_exp CHECK (drive_exp >= 0),
19     ADD CONSTRAINT check_owner_passport_format CHECK
           (passport_serial BETWEEN 1 AND 9999 AND passport_num
           BETWEEN 1 AND 999999),
20     ADD CONSTRAINT check_drive_license_format CHECK
           (drive_license_serial BETWEEN 1 AND 9999 AND
           drive_license_num BETWEEN 1 AND 999999);
```

Листинг 3.4 – Создание таблицы PTS

```
1  CREATE TABLE PTS (
2      id SERIAL PRIMARY KEY,
3      sts_id INTEGER NOT NULL,
4      pts_serial INTEGER NOT NULL,
5      pts_number INTEGER NOT NULL,
6      import_country TEXT NOT NULL
7  );
8
9  ALTER TABLE PTS
10     ADD FOREIGN KEY (sts_id) REFERENCES STS(id) ON DELETE CASCADE,
11     ADD CONSTRAINT unique_pts UNIQUE (pts_serial, pts_number),
12     ADD CONSTRAINT check_pts_format CHECK (pts_serial BETWEEN 1
           AND 9999 AND pts_number BETWEEN 1 AND 999999);
```

Листинг 3.5 – Создание таблицы STS

```
1  CREATE TABLE STS (
2      id SERIAL PRIMARY KEY,
3      car_id INTEGER NOT NULL,
```

```
4       owner_id INTEGER NOT NULL,
5       vin TEXT NOT NULL,
6       gos_num TEXT NOT NULL,
7       mark TEXT NOT NULL,
8       model TEXT NOT NULL,
9       horse_power INTEGER,
10      car_weight INTEGER,
11      sts_serial INTEGER NOT NULL,
12      sts_num INTEGER NOT NULL,
13      engine_type TEXT NOT NULL,
14      car_class TEXT NOT NULL,
15      release_date DATE NOT NULL,
16      reg_date DATE NOT NULL
17  );
18  ALTER TABLE STS
19      ADD FOREIGN KEY (car_id) REFERENCES Car(id) ON DELETE CASCADE,
20      ADD FOREIGN KEY (owner_id) REFERENCES CarOwner(id) ON DELETE
            CASCADE,
21      ADD CONSTRAINT unique_sts UNIQUE (sts_serial, sts_num),
22      ADD CONSTRAINT unique_gos_num UNIQUE (gos_num),
23      ADD CONSTRAINT sts_unique_vin UNIQUE (vin),
24      ADD CONSTRAINT check_horse_power CHECK (horse_power > 0),
25      ADD CONSTRAINT check_weight CHECK (car_weight > 0),
26      ADD CONSTRAINT check_engine_type CHECK (engine_type IN
            ('petrol', 'diesel', 'electric', 'hybrid')),
27      ADD CONSTRAINT check_car_class CHECK (car_class IN ('A', 'B',
            'C', 'D', 'E', 'F', 'S', 'M', 'J')),
28      ADD CONSTRAINT check_release_date CHECK (release_date <=
            CURRENT_DATE),
29      ADD CONSTRAINT check_reg_date CHECK (reg_date >=
            release_date),
30      ADD CONSTRAINT check_sts_format CHECK (sts_serial BETWEEN 1
            AND 9999 AND sts_num BETWEEN 1 AND 999999),
31      ADD CONSTRAINT check_gos_num_format CHECK (gos_num ~
            '^[АВЕКМНОРСТУХ]\d{3}[АВЕКМНОРСТУХ]{2}\d{2,3}\$'),
32      ADD CONSTRAINT check_vin_length CHECK (LENGTH(vin) = 17);
```

Листинг 3.6 – Создание таблицы OwnerHistory

```
1   CREATE TABLE OwnerHistory (
2       id SERIAL PRIMARY KEY,
3       pts_id INTEGER NOT NULL,
```

```
    mileage INTEGER NOT NULL,
    reg_date DATE NOT NULL,
    dereg_date DATE
);

ALTER TABLE OwnerHistory
    ADD FOREIGN KEY (pts_id) REFERENCES PTS(id) ON DELETE CASCADE,
    ADD CONSTRAINT check_mileage CHECK (mileage >= 0),
    ADD CONSTRAINT check_dates CHECK (dereg_date IS NULL OR
        dereg_date >= reg_date);
```

Листинг 3.7 – Создание таблицы OwnerHistoryOwner

```
CREATE TABLE OwnerHistoryOwner (
    id SERIAL PRIMARY KEY,
    owner_id INTEGER NOT NULL,
    owner_history_id INTEGER NOT NULL
);

ALTER TABLE OwnerHistoryOwner
    ADD FOREIGN KEY (owner_id) REFERENCES CarOwner(id) ON DELETE
        CASCADE,
    ADD FOREIGN KEY (owner_history_id) REFERENCES
        OwnerHistory(id) ON DELETE CASCADE;
```

Листинг 3.8 – Создание таблицы Camera

```
CREATE TABLE Camera (
    id SERIAL PRIMARY KEY,
    longitude DOUBLE PRECISION NOT NULL,
    latitude DOUBLE PRECISION NOT NULL,
    install_date DATE NOT NULL,
    is_radar BOOLEAN DEFAULT FALSE
);

ALTER TABLE Camera
    ADD CONSTRAINT check_install_date CHECK (install_date <=
        CURRENT_DATE);
```

Листинг 3.9 – Создание таблицы CarSnapshot

```
CREATE TABLE CarSnapshot (
```

```
2     id SERIAL PRIMARY KEY,
3     camera_id INTEGER NOT NULL,
4     snap_datetime TIMESTAMP NOT NULL,
5     speed INTEGER,
6     gos_num TEXT NOT NULL,
7     road_line INTEGER
8 );
9
10 ALTER TABLE CarSnapshot
11     ADD FOREIGN KEY (camera_id) REFERENCES Camera(id) ON DELETE
          CASCADE,
12     ADD CONSTRAINT check_speed CHECK (speed >= 0),
13     ADD CONSTRAINT check_road_line CHECK (road_line >= 0),
14     ADD CONSTRAINT check_gos_num_format CHECK (gos_num ~
          '^[АВЕКМНОРСТУХ]\d{3}[АВЕКМНОРСТУХ]{2}\d{2,3}$'),
15     ADD CONSTRAINT check_snapshot_date CHECK (snap_datetime <=
          (NOW() AT TIME ZONE 'Europe/Moscow' + INTERVAL '5
          seconds'));
```

Листинг 3.10 – Создание таблицы TrackInfo

```
1 CREATE TABLE TrackInfo (
2     id SERIAL PRIMARY KEY,
3     user_id INTEGER NOT NULL,
4     track_time TIMESTAMP NOT NULL,
5     route_date DATE NOT NULL,
6     car_id INTEGER NOT NULL
7 );
8
9 ALTER TABLE TrackInfo
10     ADD FOREIGN KEY (user_id) REFERENCES AppUser(id) ON DELETE
          CASCADE,
11     ADD FOREIGN KEY (car_id) REFERENCES Car(id) ON DELETE CASCADE,
12     ADD CONSTRAINT check_route_date CHECK (route_date <=
          track_time::DATE);
```

3.3.2 Создание ролей

В листингах 3.11— 3.14 представлено создание ролей базы данных.

Листинг 3.11 – Активация Row-Level Security (RLS)

```
ALTER TABLE Car ENABLE ROW LEVEL SECURITY;
ALTER TABLE CarOwner ENABLE ROW LEVEL SECURITY;
ALTER TABLE STS ENABLE ROW LEVEL SECURITY;
ALTER TABLE PTS ENABLE ROW LEVEL SECURITY;
ALTER TABLE OwnerHistory ENABLE ROW LEVEL SECURITY;
ALTER TABLE OwnerHistoryOwner ENABLE ROW LEVEL SECURITY;
```

Листинг 3.12 – Создание роли оператора

```
CREATE ROLE operator_role WITH
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    NOINHERIT
    NOREPLICATION
    NOBYPASSRLS
    CONNECTION LIMIT -1
    LOGIN
    PASSWORD 'Oper@tor123';

GRANT SELECT ON
    AppUser,
    Car,
    CarSnapshot,
    Camera,
    CarOwner,
    PTS,
    OwnerHistory,
    OwnerHistoryOwner,
    STS
TO operator_role;

CREATE POLICY operator_car_access ON Car
FOR SELECT TO operator_role
USING (true);

CREATE POLICY operator_carowner_access ON CarOwner
FOR SELECT TO operator_role
USING (true);

CREATE POLICY operator_sts_access ON STS
```

```
33 FOR SELECT TO operator_role
34 USING (true);
35
36 CREATE POLICY operator_pts_access ON PTS
37 FOR SELECT TO operator_role
38 USING (true);
39
40 CREATE POLICY operator_ownerhistory_access ON OwnerHistory
41 FOR SELECT TO operator_role
42 USING (true);
43
44 CREATE POLICY operator_ownerhistoryowner_access ON
      OwnerHistoryOwner
45 FOR SELECT TO operator_role
46 USING (true);
```

Листинг 3.13 – Создание роли аудита

```
1  CREATE ROLE audit_role WITH
2      NOSUPERUSER
3      NOCREATEDB
4      NOCREATEROLE
5      NOINHERIT
6      NOREPLICATION
7      NOBYPASSRLS
8      CONNECTION LIMIT -1
9      LOGIN
10     PASSWORD 'Aud1t$ecure';
11
12 GRANT SELECT ON
13     AppUser,
14     Car,
15     TrackInfo,
16     CarOwner,
17     PTS,
18     OwnerHistory,
19     OwnerHistoryOwner,
20     STS
21 TO audit_role;
22
23 CREATE POLICY audit_car_access ON Car
24 FOR SELECT TO audit_role
```

```
25 USING (true);
26
27 CREATE POLICY audit_carowner_access ON CarOwner
28 FOR SELECT TO audit_role
29 USING (true);
30
31 CREATE POLICY audit_sts_access ON STS
32 FOR SELECT TO audit_role
33 USING (true);
34
35 CREATE POLICY audit_pts_access ON PTS
36 FOR SELECT TO audit_role
37 USING (true);
38
39 CREATE POLICY audit_ownerhistory_access ON OwnerHistory
40 FOR SELECT TO audit_role
41 USING (true);
42
43 CREATE POLICY audit_ownerhistoryowner_access ON OwnerHistoryOwner
44 FOR SELECT TO audit_role
45 USING (true);
```

Листинг 3.14 – Создание роли пользователя

```
1  CREATE ROLE user_role WITH
2      NOSUPERUSER
3      NOCREATEDB
4      NOCREATEROLE
5      NOINHERIT
6      NOREPLICATION
7      NOBYPASSRLS
8      CONNECTION LIMIT -1
9      LOGIN
10     PASSWORD 'Us3rP@ssword';
11
12 GRANT SELECT ON
13     Car,
14     PTS,
15     OwnerHistory,
16     OwnerHistoryOwner,
17     STS,
18     CarOwner
```

```
19 TO user_role;
20
21 CREATE POLICY user_carowner_access ON CarOwner
22 FOR SELECT TO user_role
23 USING (
24     passport_serial =
          current_setting('app.passport_serial')::INTEGER AND
25     passport_num = current_setting('app.passport_num')::INTEGER
26 );
27
28 CREATE POLICY user_car_access ON Car
29 FOR SELECT TO user_role
30 USING (
31     id IN (
32         SELECT car_id FROM CarOwner
33         WHERE passport_serial =
              current_setting('app.passport_serial')::INTEGER
34           AND passport_num =
                current_setting('app.passport_num')::INTEGER
35     )
36 );
37
38 CREATE POLICY user_sts_access ON STS
39 FOR SELECT TO user_role
40 USING (
41     car_id IN (
42         SELECT id FROM Car
43         WHERE id IN (
44             SELECT car_id FROM CarOwner
45             WHERE passport_serial =
                  current_setting('app.passport_serial')::INTEGER
46               AND passport_num =
                    current_setting('app.passport_num')::INTEGER
47         )
48     )
49 );
50
51 CREATE POLICY user_pts_access ON PTS
52 FOR SELECT TO user_role
53 USING (
54     sts_id IN (
```

```
55          SELECT id FROM STS
56          WHERE car_id IN (
57              SELECT car_id FROM CarOwner
58              WHERE passport_serial =
                    current_setting('app.passport_serial')::INTEGER
59                AND passport_num =
                      current_setting('app.passport_num')::INTEGER
60          )
61      )
62 );
63
64 CREATE POLICY user_ownerhistory_access ON OwnerHistory
65 FOR SELECT TO user_role
66 USING (
67      pts_id IN (
68          SELECT id FROM PTS
69          WHERE sts_id IN (
70              SELECT id FROM STS
71              WHERE car_id IN (
72                  SELECT car_id FROM CarOwner
73                  WHERE passport_serial =
                        current_setting('app.passport_serial')::INTEGER
74                    AND passport_num =
                          current_setting('app.passport_num')::INTEGER
75              )
76          )
77      )
78 );
79
80 CREATE POLICY user_ownerhistoryowner_access ON OwnerHistoryOwner
81 FOR SELECT TO user_role
82 USING (
83      owner_id IN (
84          SELECT id FROM CarOwner
85          WHERE passport_serial =
                current_setting('app.passport_serial')::INTEGER
86            AND passport_num =
                  current_setting('app.passport_num')::INTEGER
87      )
88 );
```

3.3.3 Создание и тестирование процедуры

Реализация процедуры обновления паспортных данных представлена в листинге 3.15. На вход функции подаются логин пользователя и новые паспортные данные(серия и номер). В случае успеха функция обновит пасспортные данные пользователя.

Листинг 3.15 – Реализация процедуры обовления паспортных данных

```
CREATE OR REPLACE PROCEDURE verify_user(
    u_login TEXT,
    new_passport_serial INTEGER,
    new_passport_num INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF u_login IS NULL THEN
        RAISE EXCEPTION 'Login cannot be NULL';
    END IF;

    IF new_passport_serial IS NULL THEN
        RAISE EXCEPTION 'Passport serial cannot be NULL';
    END IF;

    IF new_passport_num IS NULL THEN
        RAISE EXCEPTION 'Passport number cannot be NULL';
    END IF;

    IF NOT EXISTS (SELECT 1 FROM AppUser WHERE login = u_login)
       THEN
        RAISE EXCEPTION 'User with login % does not exist',
           u_login;
    END IF;

    IF EXISTS (SELECT 1 FROM AppUser WHERE passport_serial =
       new_passport_serial AND passport_num = new_passport_num
       AND login != u_login) THEN
        RAISE EXCEPTION 'Passport data already exists for another
           user';
    END IF;
```

```
28
29      UPDATE AppUser
30      SET is_verified = TRUE,
31          passport_serial = new_passport_serial,
32          passport_num = new_passport_num
33      WHERE login = u_login;
34  END;
35  $$;
```

Тестирование проводилось с помощью расширения pgtap [18].

В таблице 3.2 представлено начальное состояние таблицы AppUser на момент запуска тестирования.

Таблица 3.2 – начальное состояние таблицы AppUser

login	is_verified	passport_serial	passport_num
test_user1@mail.ru	FALSE	NULL	NULL
test_user2@mail.ru	FALSE	NULL	NULL
test_user3@mail.ru	TRUE	1234	567890
test_user4@mail.ru	TRUE	4321	987654

Верификация существующего пользователя с несуществующими паспортными данными

Входные данные: test_user1@mail.ru, 1234, 567891.

Ожидаемый результат: паспортные данные пользователя test_user1@mail.ru обновятся и он станет верифицированным. Ожидаемое состояние таблицы AppUser представлено на таблице 3.3.

Таблица 3.3 – начальное состояние таблицы AppUser

login	is_verified	passport_serial	passport_num
test_user1@mail.ru	TRUE	1234	567891
test_user2@mail.ru	FALSE	NULL	NULL
test_user3@mail.ru	TRUE	1234	567890
test_user4@mail.ru	TRUE	4321	987654

Верификация несуществующего пользователя

Входные данные: non_existent_user@mail.ru, 1111, 222222.

Ожидаемый результат: Должно вернуть исключение 'User with login non_existent_user@mail.ru does not exist'. Таблица AppUser измениться не должна.

Верификация существующего пользователя с существующими паспортными данными

Входные данные: test_user2@mail.ru, 1234, 567890.

Ожидаемый результат: Должно вернуть исключение 'Passport data already exists for another user'. Таблица AppUser измениться не должна.

Проверка входных параметров на NULL

Входные данные: NULL, 1234, 567891.

Ожидаемый результат: Должно вернуть исключение 'Login cannot be NULL'. Таблица AppUser измениться не должна.

Входные данные: test_user2@mail.ru, NULL, 567891.

Ожидаемый результат: Должно вернуть исключение 'Passport serial cannot be NULL'. Таблица AppUser измениться не должна.

Входные данные: test_user2@mail.ru, 1234, NULL.

Ожидаемый результат: Должно вернуть исключение 'Passport number cannot be NULL'. Таблица AppUser измениться не должна.

3.4 Пример работы программы

На рисунках 3.1 — 3.4 представлены экраны клиентского приложения.

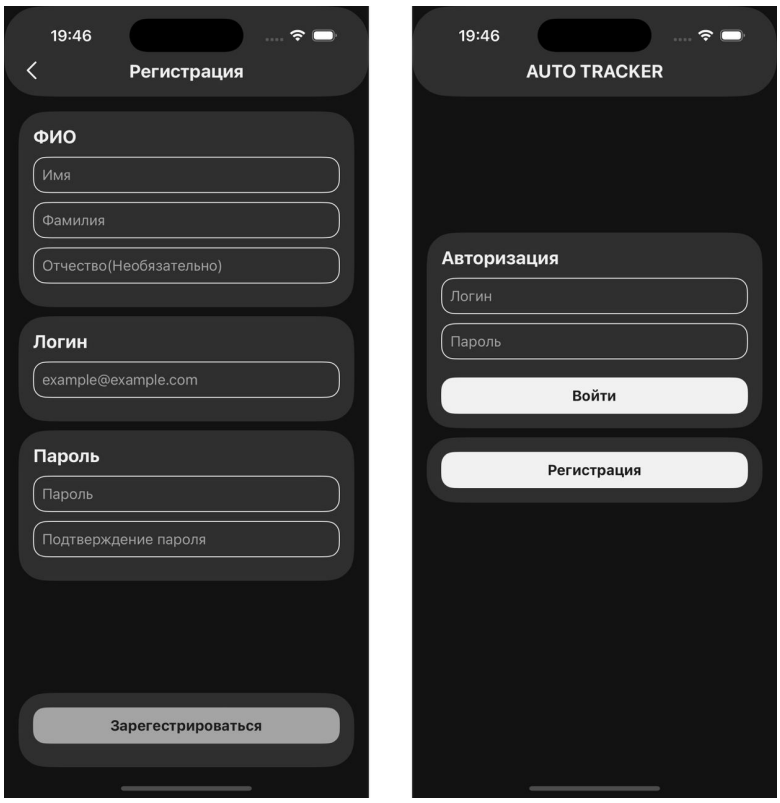

Рисунок 3.1 – Экраны аутентификации

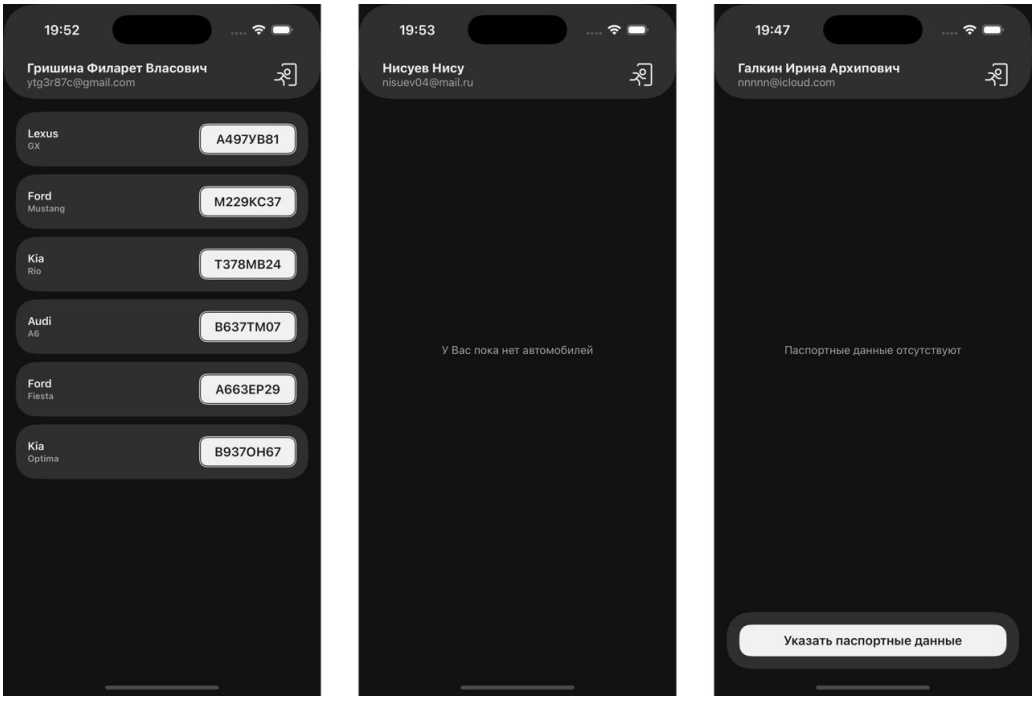

Рисунок 3.2 – Экраны пользователя

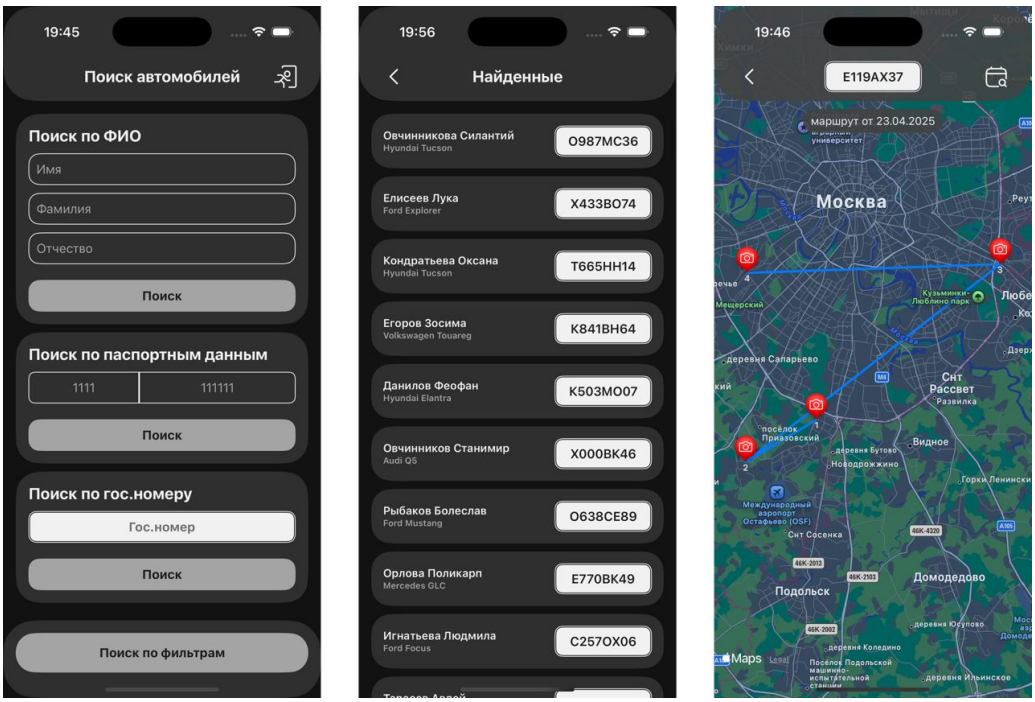

Рисунок 3.3 – Экраны пользователя

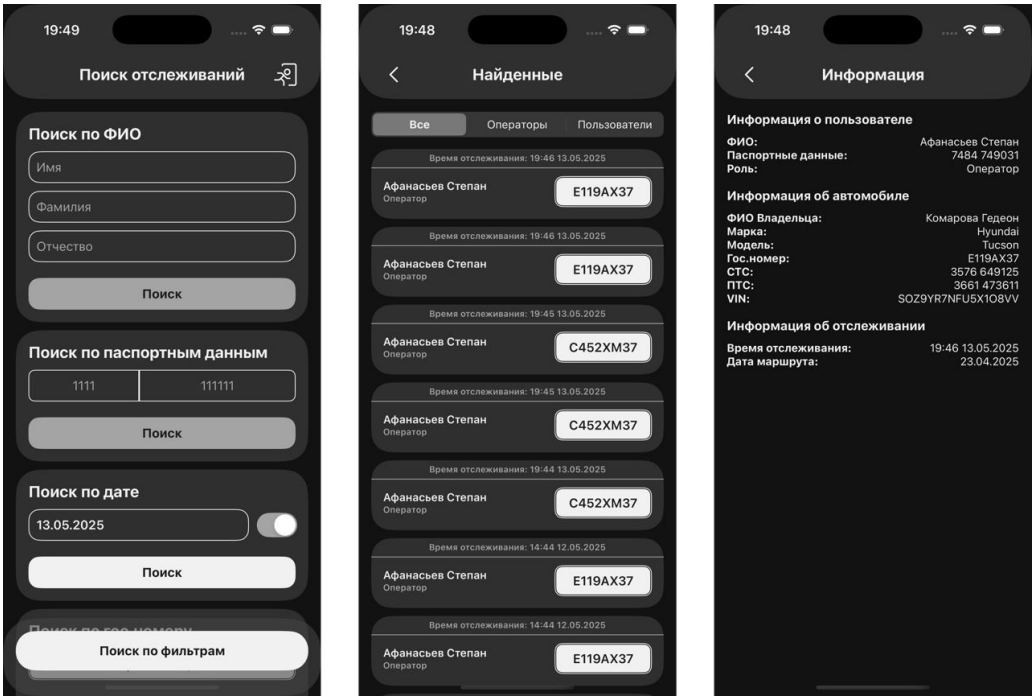

Рисунок 3.4 – Экраны пользователя

Для взаимодействие клиентского приложения с базой данных был разработан API. На рисунке 3.5 представлен Swagger реализованной API.

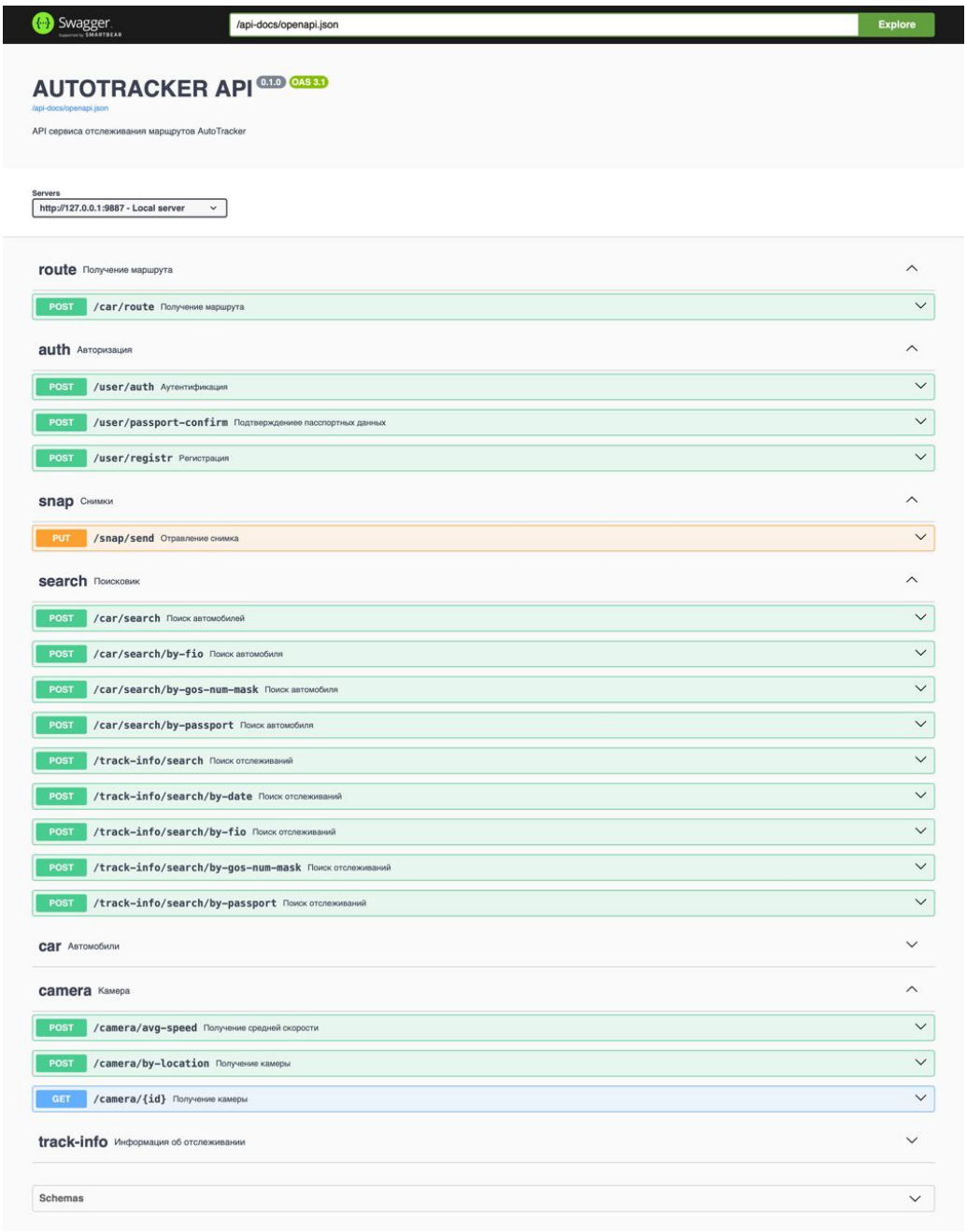

Рисунок 3.5 – Демонстрация Swagger

ВЫВОД

В данном разделе были выбраны средства реализации. Представлена реализация сущностей, ролевой модели и ограничений целостности в созданной базе данных. Также рассмотрена реализация хранимой процедуры и проведено ее тестирование. Дополнительно представлен пользовательский интерфейс программного обеспечения.

4 Исследовательская часть

В данном разделе представлены технические характеристики устройства, использованного для измерения времени работы программного обеспечения, а также приведены результаты проведенных замеров.

4.1 Технические характеристики

Характеристики используемого оборудования:

— операционная система — macOS Sequoia 15.1 [19];

— память — 16 Гб;

— процессор — Apple M1 Pro [20].

4.2 Постановка задачи

Цель данного исследования — установить зависимость времени вставки данных в базу данных от используемого способа добавления данных.

В качестве объекта исследования была выбрана сущность «снимок», а также определены следующие методы вставки данных:

1. Постепенная вставка. Реализация данного запроса представлена в листинге 4.1.

2. Пакетная вставка VALUES. Реализация данного запроса представлена в листинге 4.2.

3. Массовая загрузка через COPY. Реализация данного запроса представлена в листинге 4.3.

Листинг 4.1 – Постепенная вставка

```
1 async fn insert_snap(&self, snap: &Snap) -> Result<(),
    DataAccessError> {
2     let datetime = NaiveDateTime::parse_from_str(
```

```
3           &format!("{} {}", snap.date, snap.time),
4           "%d.%m.%Y %H:%M",
5       )
6       .map_err(|e| {
7           log::error!("Failed to parse datetime: {}", e);
8           DataAccessError::InvalidInput(e.to_string())
9       })?;
10
11      let query = "INSERT INTO CarSnapshot
12                   (camera_id, speed, snap_datetime, gos_num)
13                   VALUES ($1, $2, $3, $4)";
14
15      sqlx::query(query)
16          .bind(snap.camera.id as i32)
17          .bind(snap.speed.map(|s| s as i32))
18          .bind(datetime)
19          .bind(&snap.gos_num)
20          .execute(&self.pool)
21          .await
22          .map_err(|e| {
23              log::error!("Failed to insert snap: {}", e);
24              DataAccessError::PsqlDataBaseError(e)
25          })?;
26
27      Ok(())
28  }
29
30  pub async fn insert_snaps_by_one(&self, snaps: &[Snap]) ->
        Result<Duration, DataAccessError> {
31      let start_time = Instant::now();
32
33      for (i, snap) in snaps.iter().enumerate() {
34          let snap_start = Instant::now();
35          self.insert_snap(snap).await?;
36          log::debug!(
37              "Inserted snap {}/{} in {}ms",
38              i + 1,
39              snaps.len(),
40              snap_start.elapsed().as_millis()
41          );
42      }
```

```
43
44      let total_time = start_time.elapsed();
45      Ok(total_time)
46  }
```

Листинг 4.2 – Пакетная вставка VALUES

```
1  pub async fn insert_snaps_by_values(
2      &self,
3      snaps: &[Snap],
4  ) -> Result<Duration, DataAccessError> {
5      let start_time = std::time::Instant::now();
6
7      for chunk in snaps.chunks(100) {
8          let mut query_builder = QueryBuilder::new(
9              "INSERT INTO CarSnapshot (camera_id, speed,
                   snap_datetime, gos_num) ",
10         );
11
12         query_builder.push_values(chunk, |mut b, snap| {
13             let datetime = NaiveDateTime::parse_from_str(
14                 &format!("{} {}", snap.date, snap.time),
15                 "%d.%m.%Y %H:%M",
16             )
17             .expect("Failed to parse datetime");
18
19             b.push_bind(snap.camera.id as i32)
20                 .push_bind(snap.speed.map(|s| s as i32))
21                 .push_bind(datetime)
22                 .push_bind(&snap.gos_num);
23         });
24
25         let query = query_builder.build();
26
27         query.execute(&self.pool).await.map_err(|e| {
28             log::error!("Failed to execute bulk insert: {}", e);
29             DataAccessError::PsqlDataBaseError(e)
30         })?;
31     }
32
33     let total_time = start_time.elapsed();
34     Ok(total_time)
```

```
35 }
```

Листинг 4.3 – Массовая загрузка через COPY

```
1  pub async fn insert_snaps_by_copy(&self, snaps: &[Snap]) ->
       Result<Duration, DataAccessError> {
2      let start_time = std::time::Instant::now();
3
4      let mut connection = self.pool.acquire().await.map_err(|e| {
5          log::error!("Failed to acquire connection: {}", e);
6          DataAccessError::PsqlDataBaseError(e)
7      })?;
8
9      let pg_conn: &mut sqlx::PgConnection = &mut *connection;
10
11     let mut bytes = Vec::new();
12     for snap in snaps {
13         let datetime = NaiveDateTime::parse_from_str(
14             &format!("{} {}", snap.date, snap.time),
15             "%d.%m.%Y %H:%M",
16         )
17         .map_err(|e| {
18             log::error!("Failed to parse datetime: {}", e);
19             DataAccessError::InvalidInput(e.to_string())
20         })?;
21
22         let line = format!(
23             "{}\t{}\t{}\t{}\n",
24             snap.camera.id,
25             snap.speed
26                 .map(|s| s.to_string())
27                 .unwrap_or("\\N".to_string()),
28             datetime.format("%Y-%m-%d %H:%M:%S"),
29             snap.gos_num
30         );
31         bytes.extend_from_slice(line.as_bytes());
32     }
33
34     {
35         let mut copy = pg_conn
36             .copy_in_raw(
37                 "COPY CarSnapshot (camera_id, speed,
```

```
                    snap_datetime, gos_num) FROM STDIN",
            )
            .await
            .map_err(|e| {
                log::error!("Failed to start COPY: {}", e);
                DataAccessError::PsqlDataBaseError(e)
            })?;

        copy.send(bytes).await.map_err(|e| {
            log::error!("Failed to send COPY data: {}", e);
            DataAccessError::PsqlDataBaseError(e)
        })?;

        copy.finish().await.map_err(|e| {
            log::error!("Failed to finish COPY: {}", e);
            DataAccessError::PsqlDataBaseError(e)
        })?;
    }

    let total_time = start_time.elapsed();
    Ok(total_time)
}
```

4.3 Результаты замеров

Каждое значение получено путем взятия среднего из 10 измерений. Зависимость времени вставки от количества вставляемых строк представлена на рисунке 4.1.

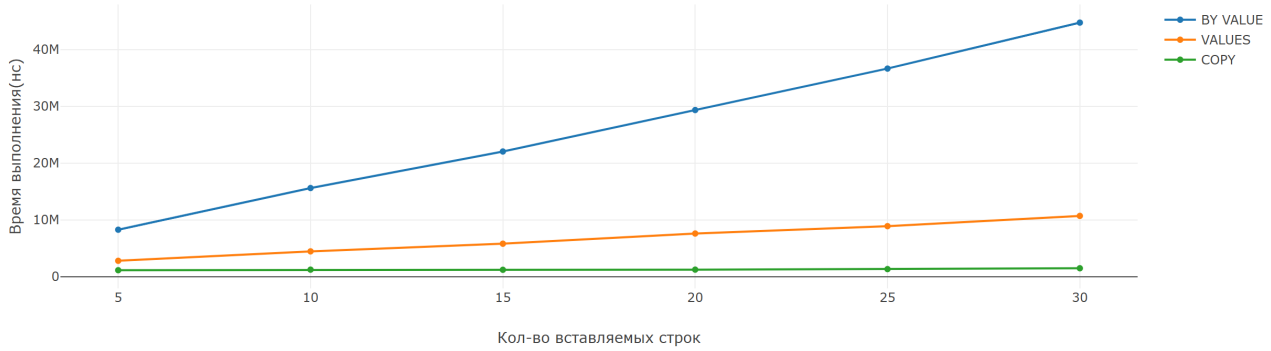

Рисунок 4.1 – Зависимость времени вставки от количества вставляемых строк

4.4 Результат исследования

На основании проведённого исследования можно сделать вывод, что метод вставки 3 является самым быстрым: при большом объёме вставляемых данных он демонстрирует производительность, превышающую метод 1 более чем в 10 раз, а метод 2 — более чем в 4 раза.

ВЫВОД

В данном разделе рассмотрены технические характеристики устройства, на котором осуществлялось измерение времени работы программного обеспечения, а также представлены полученные результаты замеров.

ЗАКЛЮЧЕНИЕ

В результате курсовой работы были разработаны база данных для сервиса отслеживания маршрутов автомобилей по данным с камер, а также приложения для доступа к ней.

В ходе выполнения данной работы были решены следующие задачи:

— проанализированы существующие решения и предметную область;

— сформулированы требования к разрабатываемой базе данных и приложению;

— спроектированы сущности и ограничения базы данных;

— выбраны средства реализации базы данных и программного обеспечения;

— реализованы база данных и программное обеспечения для доступа к ней;

— проведено исследования зависимости времени вставки данных в базу данных от метода добавления данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Automated license plate recognition: a survey on methods and techniques / J. Shashirangana [и др.] // IEEE Access. — 2020. — Т. 9. — С. 11203—11225.
2. Flock Safety [Электронный ресурс]. — Режим доступа: https://www.flocksafety.com/ (дата обращения: 06.03.2025).
3. Plate Recognizer [Электронный ресурс]. — Режим доступа: https://platerecognizer.com/ (дата обращения: 06.03.2025).
4. City Point [Электронный ресурс]. — Режим доступа: https://citypoint.ru/ (дата обращения: 06.03.2025).
5. *Карпова И.* Базы данных: курс лекций и материалы для практических занятий // учеб. пособие для вузов/ИП Карпова.-СПб.: Питер. — 2013.
6. *Дейт К. Д.* Введение в системы баз данных. — 2001.
7. *Бородин Д.*, *Строганов Ю.* К задаче составления запросов к базам данных на естественном языке // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 119—126.
8. PostgreSQL [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/ (дата обращения: 06.03.2025).
9. SQLite [Электронный ресурс]. — Режим доступа: https://www.sqlite.org/docs.html (дата обращения: 06.03.2025).
10. MySQL [Электронный ресурс]. — Режим доступа: https://dev.mysql.com/doc/ (дата обращения: 06.03.2025).
11. MSSQL [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/sql/?view=sql-server-ver17 (дата обращения: 06.03.2025).
12. *Melton J.* Iso/ansi working draft: Database language sql (sql3) // ISO/IEC SQL Revision. New York: American National Standards Institute. — 1992.
13. *Klabnik S.*, *Nichols C.* The Rust programming language. — No Starch Press, 2023.
14. Crate axum [Электронный ресурс]. — Режим доступа: https://docs.rs/axum/0.8.4/axum/ (дата обращения: 12.04.2025).

15. Crate sqlx [Электронный ресурс]. — Режим доступа: https://docs.rs/sqlx/0.8.5/sqlx/ (дата обращения: 12.04.2025).

16. Swift [Электронный ресурс]. — Режим доступа: https://www.swift.org/ (дата обращения: 12.04.2025).

17. UIKit framework [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/uikit (дата обращения: 12.04.2025).

18. pgTAP [Электронный ресурс]. — Режим доступа: https://pgtap.org/ (дата обращения: 12.04.2025).

19. macOS Sequoia [Электронный ресурс]. — Режим доступа: https://www.apple.com/macos/macos-sequoia/ (дата обращения: 12.04.2025).

20. Apple silicon [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/apple-silicon (дата обращения: 12.04.2025).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 12 слайдов.