МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

**КУРСОВАЯ РАБОТА**

по дисциплине «Базы и хранилища данных»

Разработка базы данных для предметной области «Автостоянка “AutoCar”»

Выполнил студент Шапошников Даниил Валерьевич

Группы ИТЗ-201

заочной формы обучения

направления подготовки

02.03.02 Фундаментальная информатика и информационные технологии

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

|  |  |
| --- | --- |
|  | Научный руководитель  Фамилия, имя, отчество\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Должность\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Ученая степень \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Ученое звание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_\_\_г. |

Челябинск

2023

**ЛИСТ ЗАДАНИЯ**

1. **Провести анализ предметной области по следующему описанию:**

В городе существует круглосуточная отапливаемая автостоянка “AutoCar” с установленной автоматизированной системой наблюдения, которая предоставляет гарантии безопасности автомобиля посетителя на своей территории, за счет того, что контролирует все выходы из комплекса автостоянки и сохраняет время автовладельцев на прогрев автомобиля, также в добавок ко всему закрытая автостоянка дает возможность избежать загрязнения от природных погодных условий. Клиент оплачивает место автостоянки ежемесячно.

**Перечень входных (первичных) документов:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ФИО клиента | Марка автомобиля | парковочный номер | Сумма оплаты | Дата оплаты | цена места | дата начисления оплаты | номер телефона | дата рождения | номер автомобиля | год выпуска |
| Иванов Иван Иванович | Mazda | 174 | 500 | 20/04/07 | 400 | 19/04/07 | 8935123456 | 20/04/1993 | А888НА174 | 1965 |
| Петров Алексей Петрович | LADa | 53 | 100 | 25/04/07 | 100 | 20/04/07 | 8241459673 | 23/04/1986 | O812OP74 | 1999 |

**Ограничение предметной области**

* Номер телефона клиента указывается в 10-ом формате;
* Номера автомобиля являются уникальный и имеют вид: Пример(А888НА174);
* Клиенты должны иметь возрастное ограничение 18+;
* Уникальный номер парковочного места имеет числовой формат с ограничением не более 999.
* дата рождения имеет вид (DD/MM/YYYY)
* Год выпуска имеет тип number, к примеру 1965;
* У каждого уникального парковочного места есть своя цена.
* Клиент паркуются только на своем парковочном месте исходя из номера автомобиля;
* У одного клиента может быть несколько автомобилей;
* У одного автомобиля может быть несколько клиентов.

1. **Выполнить концептуальное (инфологическое) проектирование.**
2. **Выполнить даталогическое проектирование для реляционной базы данных.**
3. **Выбрать СУБД (обосновав выбор) и выполнить физическое проектирование.**
4. **Создать базу данных в выбранной СУБД с учетом ограничений предметной области.**
5. **Реализовать следующие отчеты (запросы):**
   1. Выявить какой клиент больше всего задолжал автостоянке и когда у него был выполнен последний платеж.
   2. Посчитать сколько автомобилей имеют более одного хозяина и вывести всех хозяев.
   3. Вывести автомобиль, который имеет самый маленький долг к автостоянке за указанный пользователем период.
   4. Вывести сумму долга по всем клиентам за указанный пользователем период, которая вычисляется из расчета того, что начисление происходит 1 раз в месяц, оплаты могут происходить хоть каждый день. Все суммы начислений в данном поле складываются, а все оплаты идут со знаком минус. Долг может быть отрицательным!
   5. Вывести все номера и владельцев автомобилей, относящиеся к указанной пользователем марки автомобиля.
6. **Выбрать язык программирования и разработать приложение для работы с БД (формы ввода/редактирования данных и отчеты).**
7. **Оформить пояснительную записку.**

ОГЛАВЛЕНИЕ

[АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc128468488)

[КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ БД 6](#_Toc128468489)

[ЛОГИЧЕСКАЯ МОДЕЛЬ БД 10](#_Toc128468490)

[ФИЗИЧЕСКАЯ МОДЕЛЬ БД 13](#_Toc128468491)

[ОПИСАНИЕ ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БД 16](#_Toc128468492)

[ЗАКЛЮЧЕНИЕ 23](#_Toc128468493)

[СПИСОК ЛИТЕРАТУРЫ 24](#_Toc128468494)

[ПРИЛОЖЕНИЕ 1 25](#_Toc128468495)

[ПРИЛОЖЕНИЕ 2 30](#_Toc128468496)

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Основные ограничения предметной области и первичные документы уже даны нам в техническом задании (стр. 2), что упрощает дальнейшую работу.

В результате анализа предметной области по описанию технического задания можно прийти к следующим выводам:

* База данных создаётся для администрирования круглосуточной отапливаемой автостоянки.
* БД должна содержать данные о клиентах компании, автомобилях клиентов, парковочных местах, финансовом состоянии компании, посредством расчета задолженностей клиентов и предоставлять возможность получать разнообразные отчёты.

# КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ БД

Для построения концептуальной модели БД выделим базовые сущности этой предметной области и представим эти сущности в виде таблицы сущностей (таб. 1):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование сущности | Описание сущности | Первичный ключ | Кол-во экземпляров на основе первичных документов |
| Клиент | Посетитель автостоянки, пользующийся услугами автостоянки. | Код клиента | 2 |
| Автомобиль | Собственность клиента, оставляемая на хранение в рамках оказания услуг автостоянки владельцу этого вида собственности. | Номер автомобиля | 2 |
| Парковочное место | Площадь пространства на территории автостоянки, арендуемая клиентом с целью хранения на ней автомобиля, который закрепляется за этой площадью на срок аренды в рамках оказания услуг клиенту автостоянкой. | Парковочный номер | 2 |

Таблица 1

После определения базовых сущностей следующим шагом следует определить атрибуты каждой сущности.

Атрибуты сущности «Клиент» (таб. 2):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип значения | Формат | Диапазон значений | Возможность принимать неопр. значения |
| Код клиента | Первичный ключ | Атомарное | Целое число | ≥0 | Нет |
| Фамилия | - | Атомарное | Строка.  Длина строки – до 100 символов | Допустимые символы “А-я” | Нет |
| Имя | - | Атомарное | Строка.  Длина строки – до 100 символов | Допустимые символы “А-я” | Нет |
| Отчество | - | Атомарное | Строка.  Длина строки – до 100 символов | Допустимые символы “А-я” | Да |
| Дата рождения | - | Множественное | Дата в формате  DD/MM/YYYY | От 01/01/1900 до сегодняшней даты | Нет |
| Номер телефона | - | Атомарное | Строка.  Длина - 11 цифр. | От 0 до 9 | Да |

Таблица 2

Атрибуты сущности «Автомобиль» (таб. 3):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип значения | Формат | Диапазон значений | Возможность принимать неопр. значения |
| Номер автомобиля | Первичный ключ | Атомарное | Строка.  9 символов. | Допустимые символы “А-Я” и “0-9” | Нет |
| Марка автомобиля | - | Атомарное | Строка  127 символов. | Допустимые символы “A-z” и “А-я” и “0-9” | Да |
| Год выпуска | - | Атомарное | Целое число.  4 цифры. | 1900 – текущий год | Нет |

Таблица 3

Атрибуты сущности «Парковочное место» (таб. 4):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип значения | Формат | Диапазон значений | Возможность принимать неопр. значения |
| Номер парковочного места | Первичный ключ | Атомарное | Целое число.  До 3 цифр. | 0…999 | Нет |
| Цена места | - | Атомарное | Число с плавающей точкой. | >0 | Нет |

Таблица 4

Сформировав сущности и их атрибуты закономерным шагом будет определение связей между этими сущностями (таб. 5):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование связи | Тип связи | Хар-ка мощности связи | Перечень атрибутов связи |
| Договор аренды | Тернарная | 3 | Код клиента;  Номер места;  Номер автомобиля;  Дата договора;  Дата начисления;  Дата оплаты;  Сумма оплаты;  Задолженность. |

Таблица 5

После определения связей для каждой связи распишем её атрибуты.

Атрибуты связи «Аренда» (таб. 6):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | | Тип значения | Формат | Диапазон значений | Возможность принимать неопр. значения |
| Код клиента | Составной ключ | Внешний ключ | Атомарное | Целое число | ≥0 | Нет |
| Номер автомобиля | Атомарное | Строка.  9 символов. | Допустимые символы “А-Я” и “0-9” | Нет |
| Дата договора | - | | Атомарное | Дата в формате  DD/MM/YYYY | От 01/01/2000 до сегодняшней даты | Нет |
| Дата начисления | - | | Множественное | Дата в формате  DD/MM/YYYY | От 01/01/2000 до сегодняшней даты | Да |
| Дата оплаты | - | | Множественное | Дата в формате  DD/MM/YYYY | От 01/01/2000 до сегодняшней даты | Да |
| Сумма оплаты | - | | Атомарное | Число с плавающей точкой. | >0 | Да |
| Задолженность | - | | Атомарное | Число с плавающей точкой. | > -∞  < +∞ | Нет |

Таблица 6

На основании анализа предметной области представляется возможность проектирования концептуальной модели БД. (рис.1)

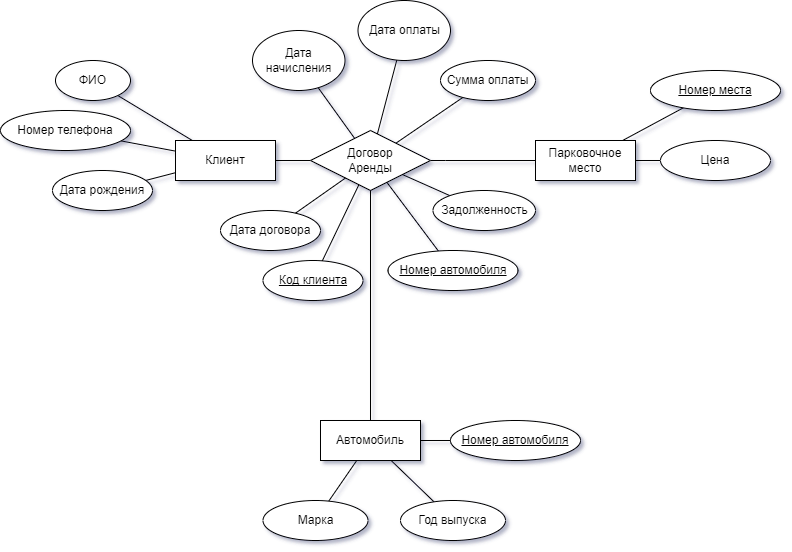


Рисунок 1 - ER диаграмма

Следующим шагом на пути разработки БД будет преобразование концептуальной модели в логическую модель базы данных.

# ЛОГИЧЕСКАЯ МОДЕЛЬ БД

Перед созданием логической модели необходимо определиться с типом модели БД, которая будет использоваться на этапе физического проектирования.

Учитывая наше стремление к тому, чтобы получившаяся БД была достаточно гибкой для дальнейшего развития и расширения, а также, беря во внимание тенденции рынка, предпочтительней будет выбрать реляционную модель БД.

Соответственно следующие шаги по проектированию базы данных будут выполняться с учетом выбранной модели.

Сначала преобразуем сущности концептуальной модели в логическую.

Создание отношения «Клиент» (рис. 2):

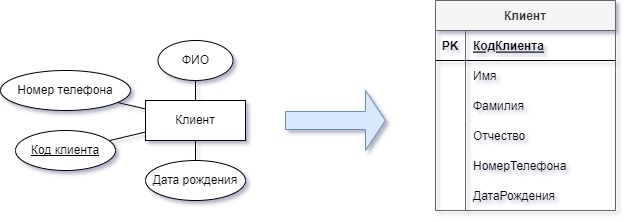


Рисунок 2 – «Клиент»

В рамках 1-й нормальной формы Атрибут «ФИО» преобразован в 3 атомарных атрибута – «Фамилия», «Имя» и «Отчество».

Создание отношения «Автомобиль» (рис. 3):

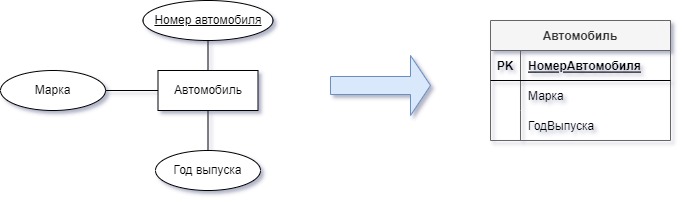


Рисунок 3 – «Автомобиль»

Создание отношения «Парковочное место» (рис. 4):

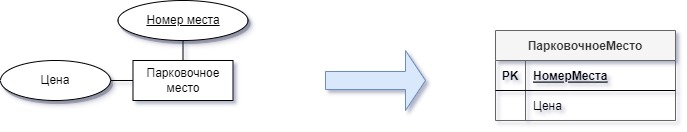


Рисунок 4 – «Парковочное место»

Учитывая то, что связь «Договор аренды» является тернарной и имеет атрибуты, перенести её в логическую модель можно будет только путем создания нового отношения «ДоговорАренды» (рис. 5):

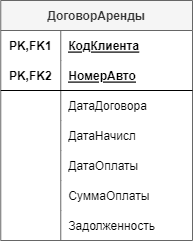


Рисунок 5 – «Договор Аренды»

При этом, учитывая то, что для одного парковочного места в один промежуток времени может быть один активный договор аренды, при этом парковочное место может быть свободным, то связь отношений «ДоговорАренды» и «ПарковочноеМесто» будет необязательной один-к-одному. Таким образом отношение «ПарковочноеМесто» принимает следующий вид (рис. 6):

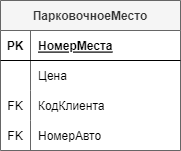


Рисунок 6 – обновленное «Парковочное Место»

В таком случае общая логическая модель БД принимает вид (рис. 7):

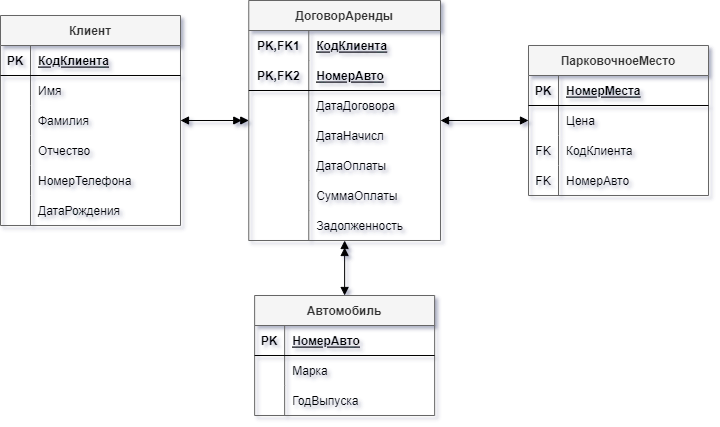


Рисунок 7

После прохождения проверок на 2НФ и 3НФ обнаружилось, что все атрибуты отношений находятся в полной ФЗ, из-за чего полученная модель находится в 3НФ и в дополнительных операциях нормализации не нуждается, в связи с чем имеет смысл приступить к физическому проектированию получившейся БД.

# ФИЗИЧЕСКАЯ МОДЕЛЬ БД

На этапе физического проектирования БД стоит определиться с конкретной СУБД, на которой планируется разработка этой БД, т.к. это влечёт за собой различия в особенностях оптимизации и денормализации для различных СУБД.

В данном случае выбор пал на PostgreSQL по причине наличия мощных инструментов, обширной документации, бесплатной лицензии и популярности в профессиональной среде.

На этапе логического проектирования мы определи конкретные отношения-таблицы, которые будут присутствовать в нашей базе данных. Теперь же стоит рассмотреть размещение данных этих таблиц в рамках выбранной СУБД.

Размещение данных в отношении Client (логическое отношение “Клиент”) (таб. 7):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| ID | PRIMARY KEY | SERIAL | NOT NULL |
| FirstName | - | VARCHAR(100) | NOT NULL |
| LastName | - | VARCHAR(100) | NOT NULL |
| Patronymic | - | VARCHAR(100) | - |
| BirthDate | - | DATE | NOT NULL |
| PhoneNumber | - | VARCHAR(11) | - |

Таблица 7

Размещение данных в отношении Car (логическое отношение “Автомобиль”) (таб. 8):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| Number | PRIMARY KEY | VARCHAR(9) | NOT NULL |
| Brand | - | VARCHAR(100) | - |
| ReleaseYear | - | SMALLINT | NOT NULL |

Таблица 8

Размещение данных в отношении ParkingSeat (логическое отношение “ПарковочноеМесто”) (таб. 9):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| ID | PRIMARY KEY | SMALLINT | NOT NULL |
| Price | - | MONEY | NOT NULL |
| ClientID | FOREIGN KEY | INTEGER | - |
| CarNumber | FOREIGN KEY | VARCHAR(9) | - |

Таблица 9

Размещение данных в отношении Contract (логическое отношение “ДоговорАренды”) (таб. 10):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| ClientID | FOREIGN KEY / PRIMARY KEY | INTEGER | NOT NULL |
| CarNumber | FOREIGN KEY / PRIMARY KEY | VARCHAR(9) | NOT NULL |
| SeatID | FOREIGN KEY | SMALLINT | NOT NULL |
| ContractDate | - | DATE | NOT NULL |
| AccrualDate | - | DATE | - |
| Debt | - | MONEY | NOT NULL |
| PaymentDate | - | DATE | - |
| PaymentAmount | - | MONEY | - |

Таблица 10

После размещения данных стоит рассмотреть возможность денормализации либо оптимизации в рамках СУБД PostgreSQL.

В рамках оптимизации предпочтительно для отношения «ДоговорАренды» было принято решение заменить составной первичный / внешний ключ на искусственный ключ, при этом атрибуты составного ключа сделать обязательными и создать для них индексы.

Таким образом размещение данных в отношении Contract обретает следующий вид (таб. 11):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| ID | PRIMARY KEY | INTEGER | NOT NULL |
| ClientID | FOREIGN KEY | INTEGER | NOT NULL |
| CarNumber | FOREIGN KEY | VARCHAR(9) | NOT NULL |
| SeatID | FOREIGN KEY | SMALLINT | NOT NULL |
| ContractDate | - | DATE | NOT NULL |
| AccrualDate | - | DATE | - |
| Debt | - | MONEY | NOT NULL |
| PaymentDate | - | DATE | - |
| PaymentAmount | - | MONEY | - |

Таблица 11

В свою очередь размещение данных в отношении ParkingSeat обретает следующий вид (таб. 12):

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование атрибута | Признак ключа | Тип данных | Возможность принимать NULL |
| ID | PRIMARY KEY | SMALLINT | NOT NULL |
| Price | - | MONEY | NOT NULL |
| СontractID | FOREIGN KEY | INTEGER | - |

Таблица 12

При этом стоит добавить B-Tree индексы для следующих атрибутов:

* Contract.ClientID
* Contract.CarNumber
* Contract.SeatID
* ParkingSeat.ContractID

Следующим шагом остается реализация физической модели БД в СУБД PostgreSQL и разработка интерфейса для взаимодействия с БД.

# ОПИСАНИЕ ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БД

Приложение для работы с получившейся БД представляет собой Web приложение написанное на ASP.NET Core 7.0 с применением технологии Razor Pages и фронтенд фреймворком Bootstrap.

Приложение имеет следующие формы:

* Форма списка клиентов
* Форма добавления нового клиента
* Форма редактирования данных существующего клиента
* Форма списков договоров, оплаты и начисления
* Форма заключение нового договора
* Форма списка автомобилей
* Форма добавления нового автомобиля
* Форма списка парковочных мест
* Форма заключение нового договора (для выбранного свободного парковочного места)

Далее подробнее по каждой форме.

Форма списка клиентов (рис. 8):

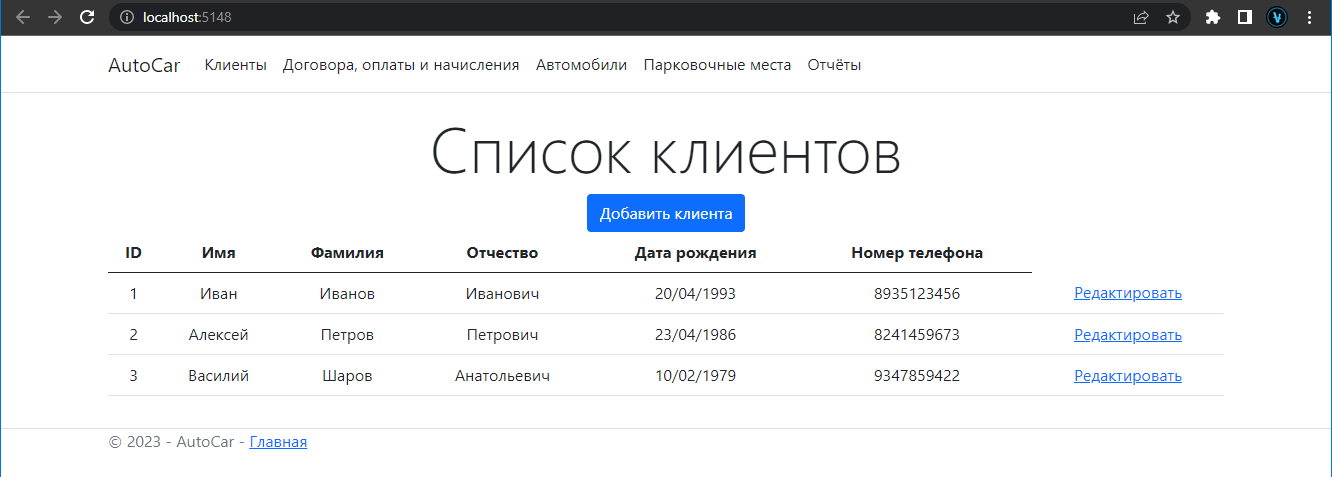


Рисунок 8

Позволяет посмотреть список доступных клиентов, при этом содержит ссылки на форму добавление нового клиента. Так же имеется ссылки на формы редактирования существующих клиентов.

Форма добавления нового клиента (рис. 9):

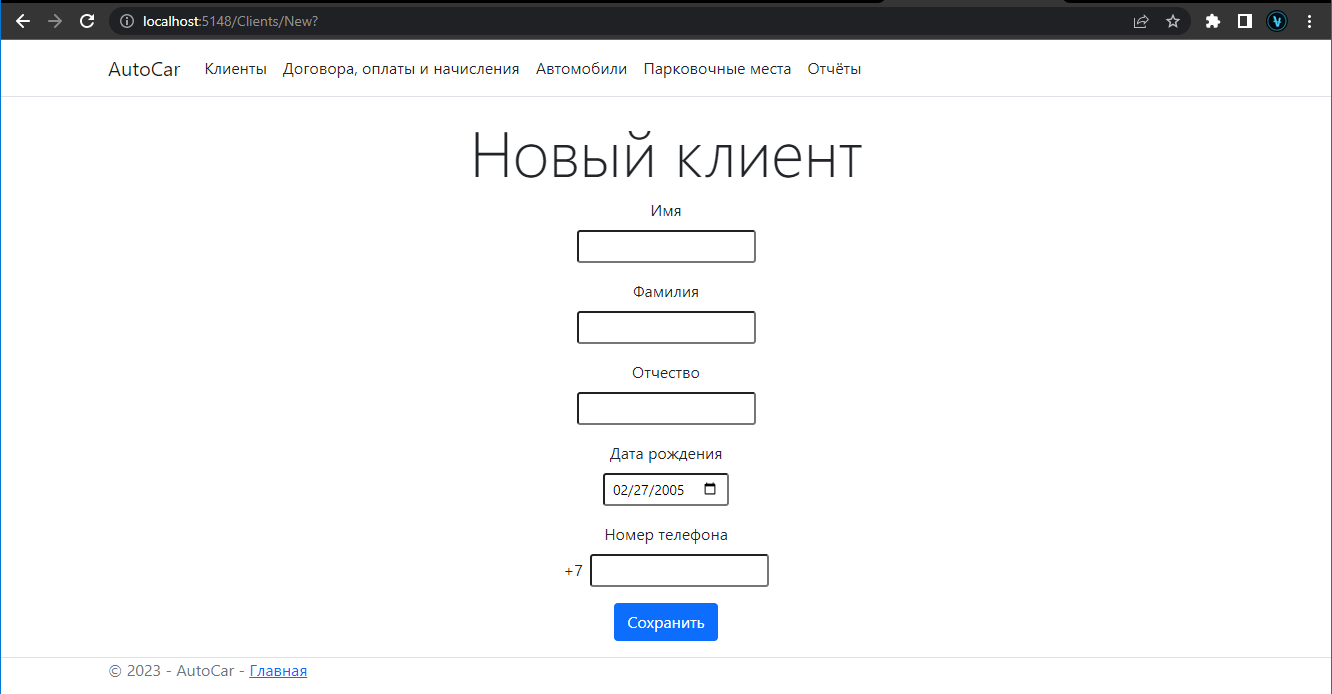


Рисунок 9

Форма позволяет добавить нового клиента, имеет валидацию данных на уровне обработки запроса.

Форма редактирования существующего клиента (рис. 10):

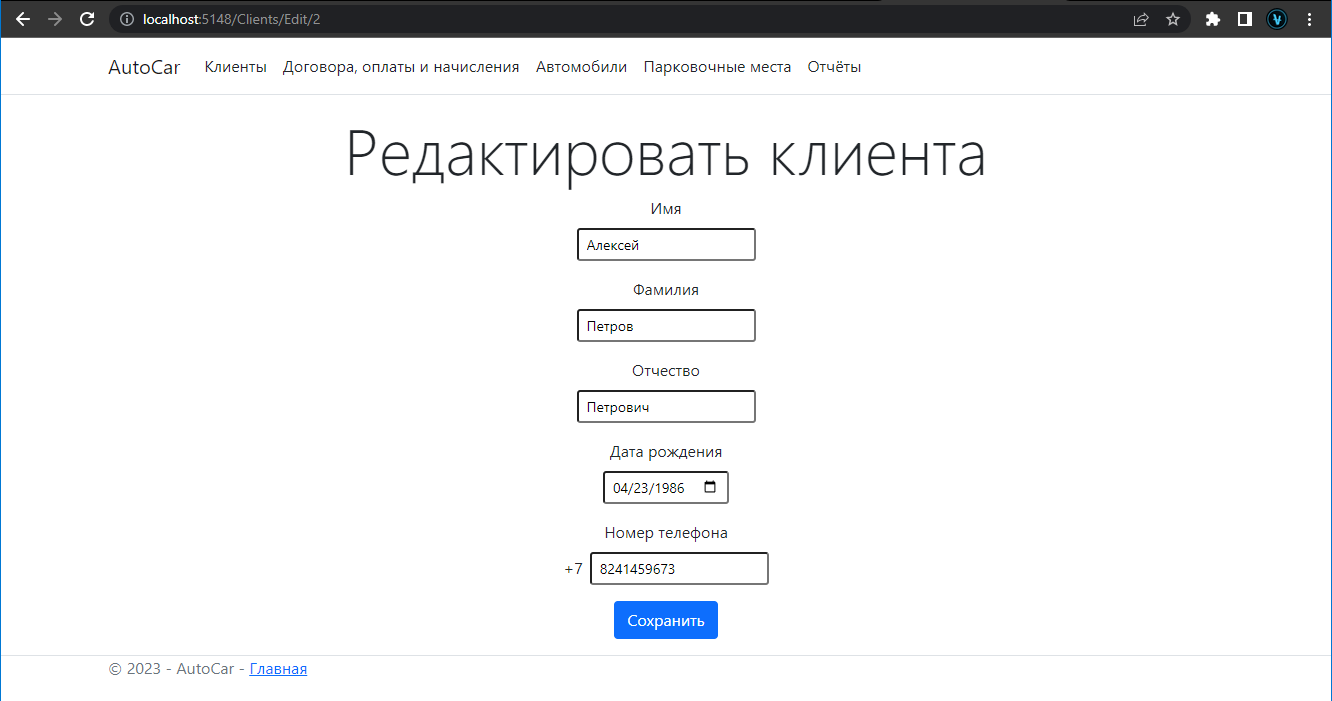


Рисунок 10

Форма позволяет редактировать существующего клиента, имеет валидацию данных на уровне обработки запроса.

Форма списка автомобилей (рис. 11):

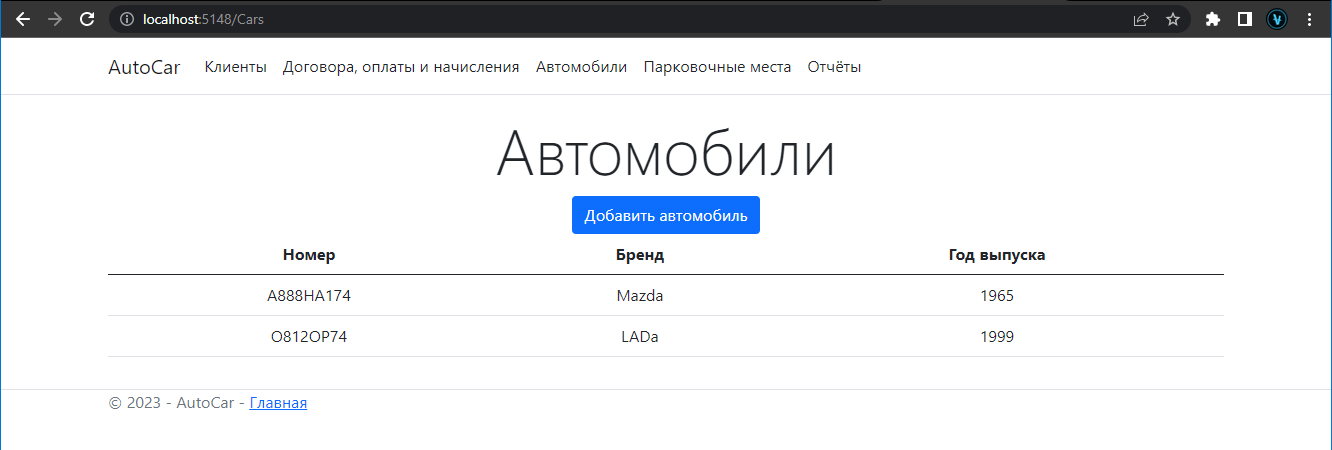


Рисунок 11

Форма позволяет просмотреть список автомобилей. Также имеется ссылка на форму добавления нового автомобиля.

Форма добавления автомобиля (рис. 12):

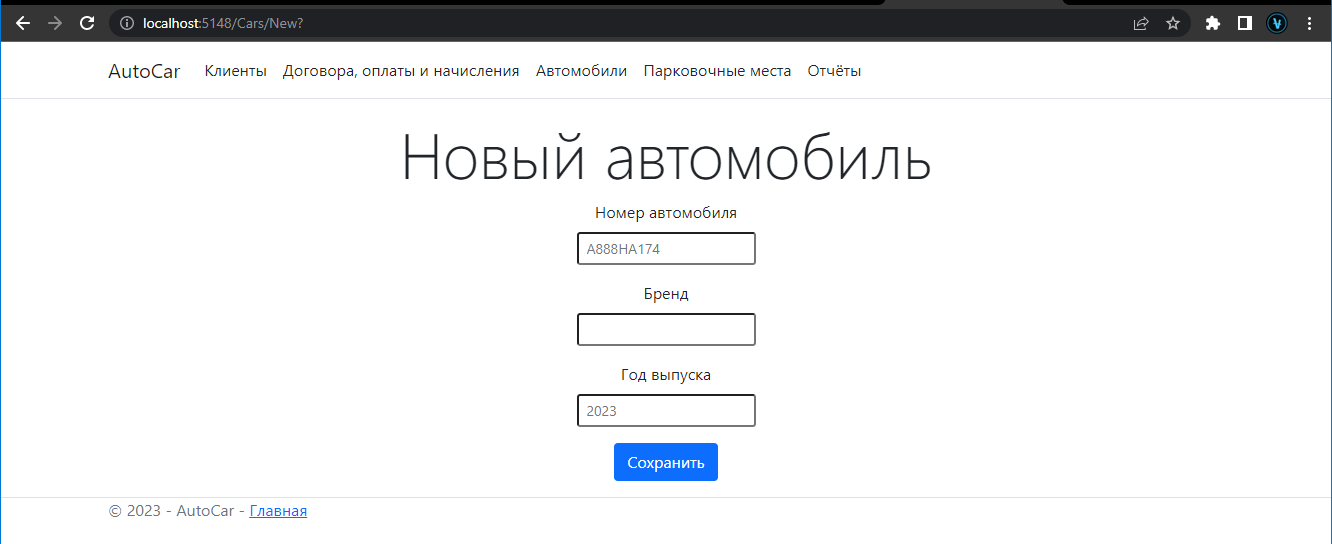


Рисунок 12

Форма позволяет добавить новый автомобиль, имеет валидацию данных на уровне обработки запроса.

Форма списка договоров (рис. 13):

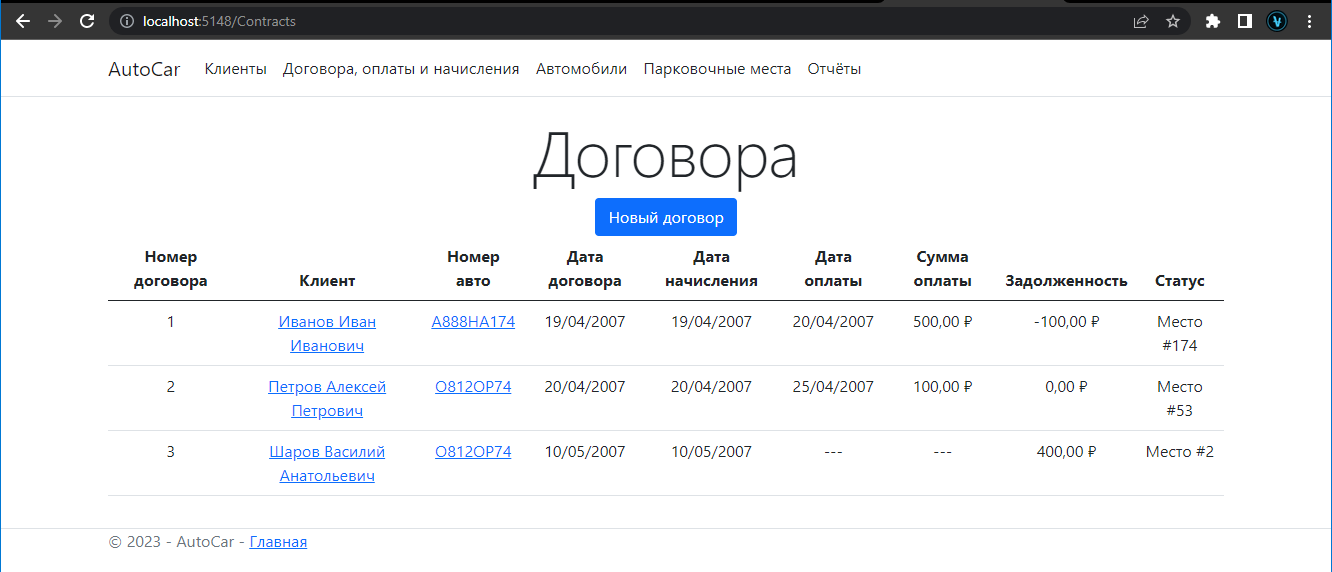


Рисунок 13

Форма позволяет просмотреть список договоров. Также имеется ссылка на форму заключения нового договора.

Форма заключения нового договора (рис. 14):

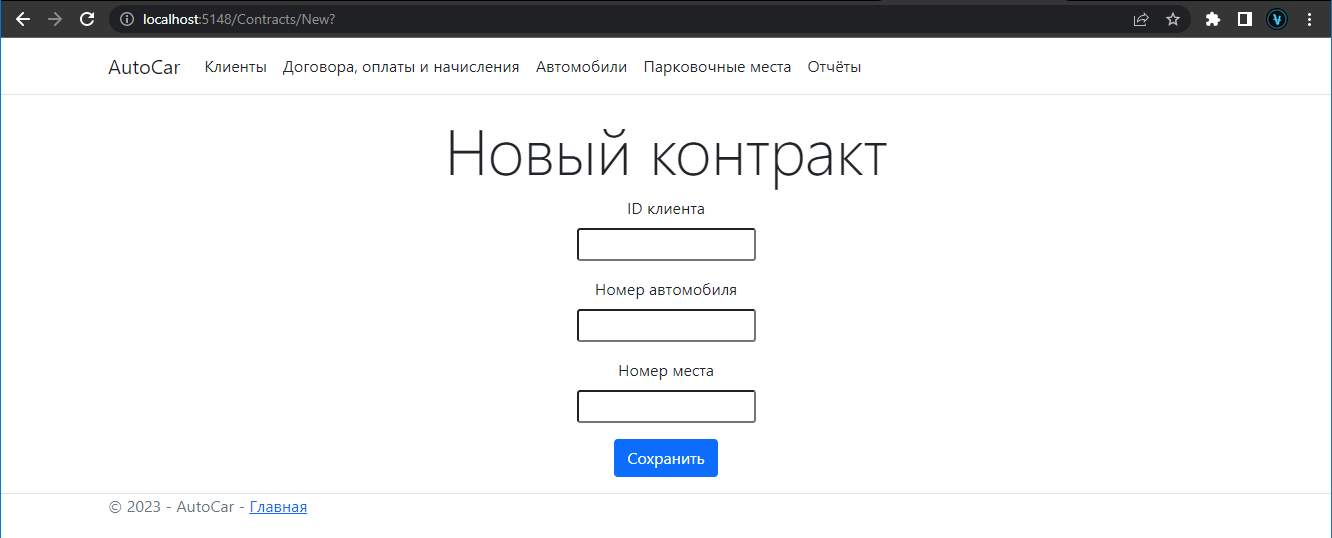


Рисунок 14

Форма позволяет заключить новый договор с конкретным клиентом, арендуя парковочное место для конкретного автомобиля.

Форма списка парковочных мест (рис. 15):

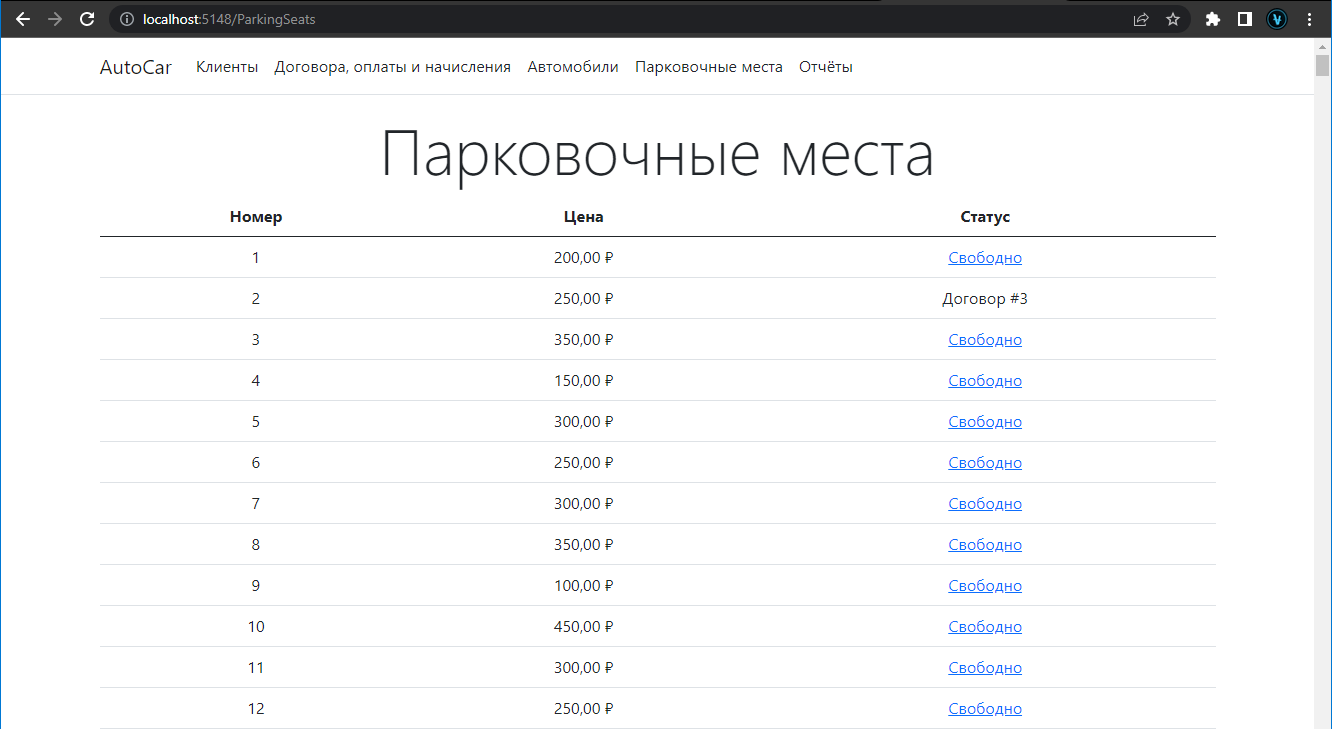


Рисунок 15

Форма позволяет просмотреть список парковочных мест, а также выбрать свободное парковочное место и перейти на форму заключения договора (рис. 14) на аренду этого места.

Форма отчёта #1 (рис. 16):

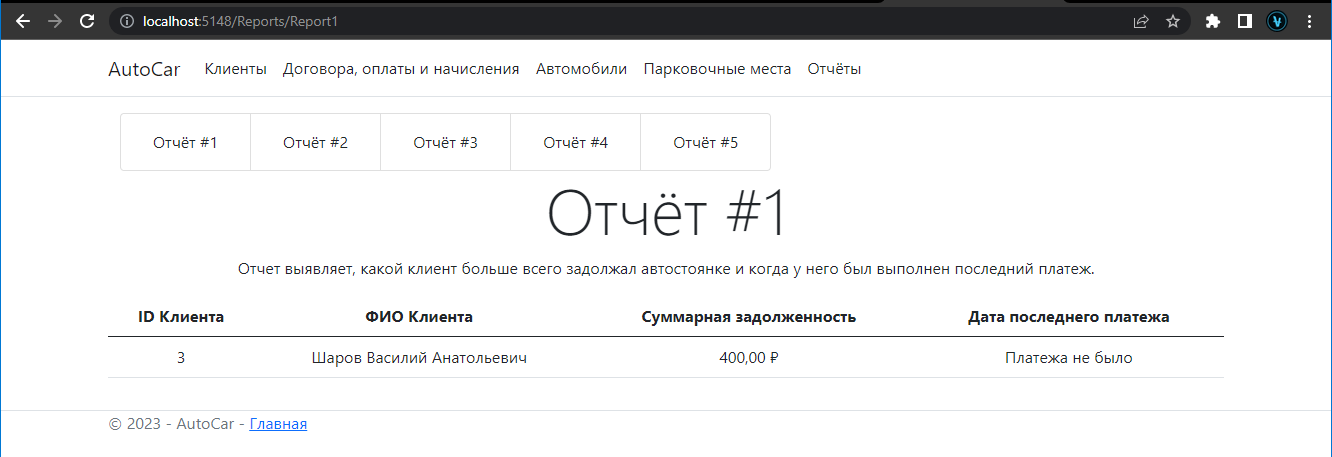


Рисунок 16

Форма позволяет ознакомиться с результатами отчёта указанного в п. 6.1 листа задания.

Также присутствует навигация к другим отчётам.

Отчёт реализован единичным SQL SELECT запросом, возвращаемым в формате отчёта и де-сериализуется в структуру данных отчета на сервере.

Форма отчёта #2 (рис. 17):

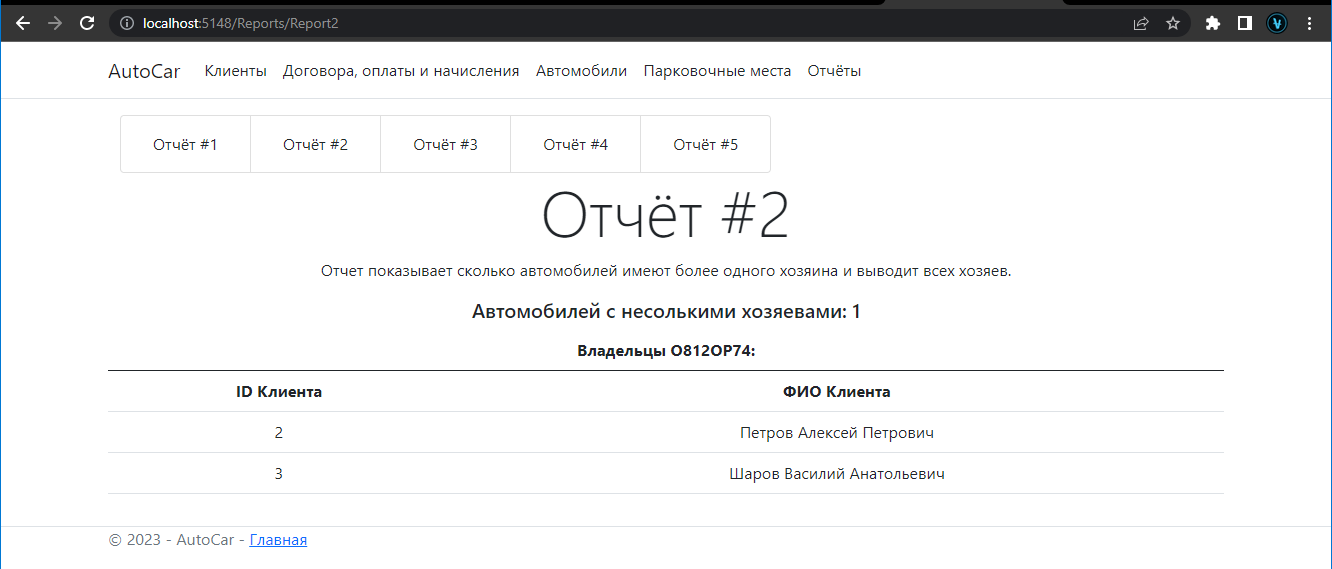
****

Рисунок 17

Форма позволяет ознакомиться с результатами отчёта указанного в п. 6.2 листа задания.

Также присутствует навигация к другим отчётам.

Отчёт реализован несколькими SQL SELECT запросами. Первый запрос возвращает список автомобилей подходящих по запросу, в результате мы имеем номера автомобилей и их количество, посчитанное на сервере Далее по номеру каждого автомобиля выполняем запрос, возвращающий уникальных владельцев авто с этим номером. Все эти данные де-сериализуется в структуру данных отчета на сервере.

Форма отчёта #3 (рис. 18):

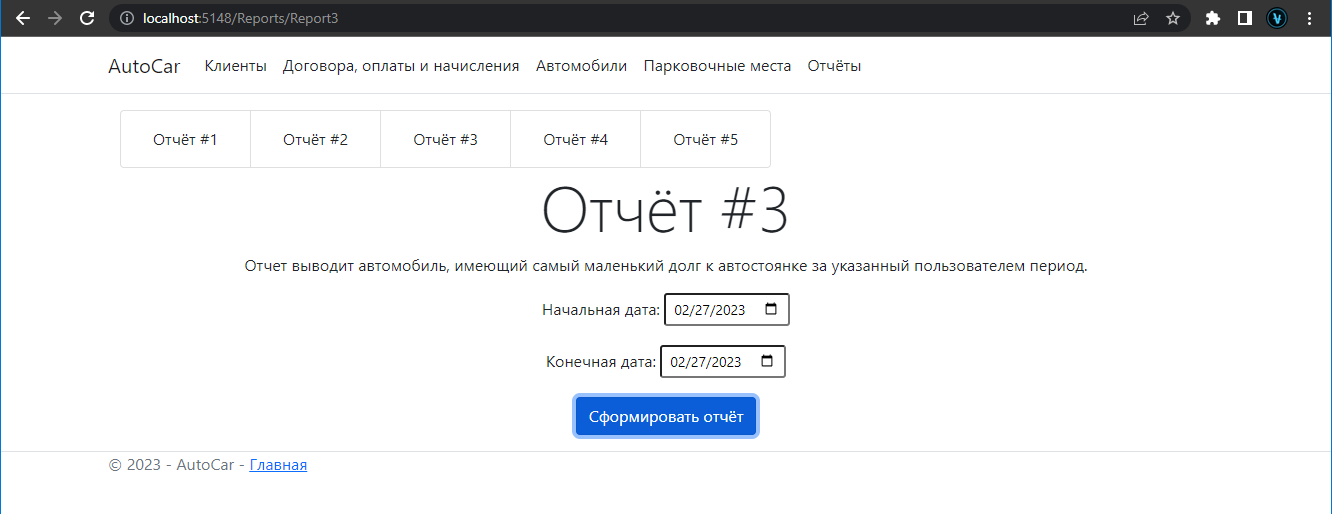


Рисунок 18

Форма позволяет выбрать временной диапазон и сформировать указанный в п. 6.3 листа задания.

Также присутствует навигация к другим отчётам.

Отчёт реализован единичным SQL SELECT запросом, возвращаемым в формате отчёта (номер автомобиля и задолженность) и де-сериализуется в структуру данных отчета на сервере.

Форма отчёта #4 (рис. 19):

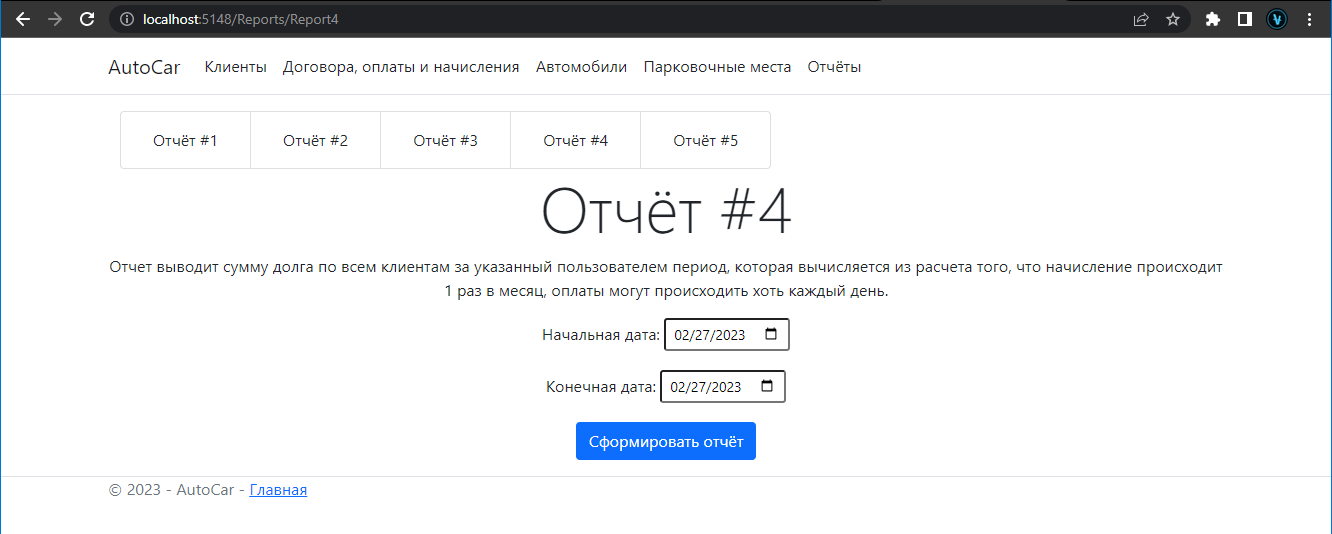


Рисунок 19

Форма позволяет выбрать временной диапазон и сформировать указанный в п. 6.4 листа задания.

Также присутствует навигация к другим отчётам.

Отчёт реализован единичным SQL SELECT запросом, возвращаемым в формате отчёта (общая задолженность и диапазон) и де-сериализуется в структуру данных отчета на сервере.

Форма отчета #5 (рис. 20):

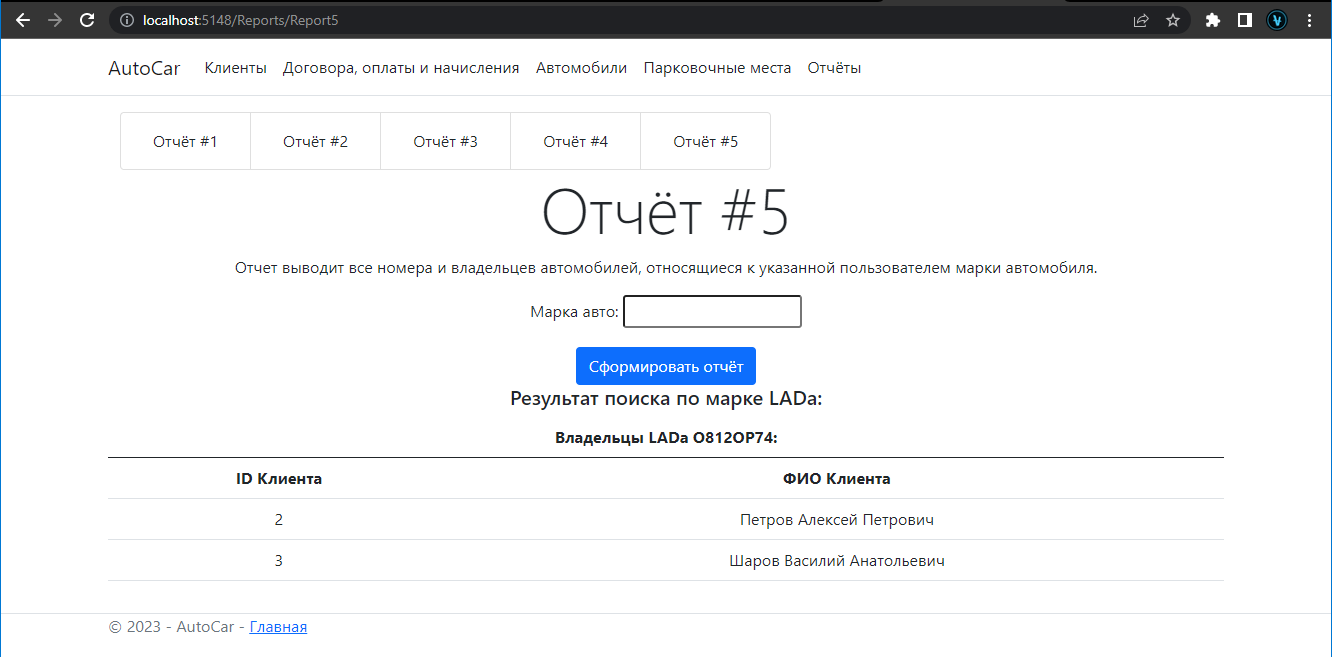


Рисунок 20

Форма позволяет указать марку автомобиля и сформировать указанный в п. 6.5 листа задания.

Также присутствует навигация к другим отчётам.

Отчёт реализован единичным SQL SELECT запросом, возвращаемым в формате отчёта и де-сериализуется в структуру данных отчета на сервере.

# ЗАКЛЮЧЕНИЕ

Подводя итог, в результате анализа предметной области по описанию требований потенциального заказчика, последовательного перехода от одной модели базы данных к другой в процессе проектирования БД в рамках задания и реализации получившейся физической модели БД на СУБД PostgreSQL был получен практический опыт проектирования, разработки и взаимодействия с базами данных, изучены особенности работы конкретной СУБД, а так же имеется работающий экземпляр Web приложения для взаимодействия с разработанной БД.

# СПИСОК ЛИТЕРАТУРЫ

1. И.П. Карпова. Базы данных: Учебное пособие. СПб: Питер, 2013г. – 240с.
2. М.П. Малыхина. Базы данных: основы, проектирование, использование. 2-е изд. СПб: БХВ-Петербург, 2007.
3. В.В. Кирилов, Г.Ю. Громов. Введение в реляционные базы данных. – СПб.:БХВ-Петербург, 2009г. – 464с.
4. Г. Хансен, Дж. Хансен. Базы данных: разработка и управление. М: Бином, 1999г. – 704с.

# ПРИЛОЖЕНИЕ 1

SQL cкрипты для создания БД и таблиц:

CREATE DATABASE autocar\_db  
 WITH  
 OWNER = postgres  
 ENCODING = 'UTF8';  
  
CREATE TABLE "Clients"  
(  
 "Id" **SERIAL** PRIMARY KEY,  
 "FirstName" **VARCHAR**(100) NOT NULL,  
 "LastName" **VARCHAR**(100) NOT NULL,  
 "Patronymic" **VARCHAR**(100),  
 "BirthDate" **DATE** NOT NULL,  
 "PhoneNumber" **VARCHAR**(10)  
);  
  
CREATE TABLE "Cars"  
(  
 "Number" **VARCHAR**(9) PRIMARY KEY,  
 "Brand" **VARCHAR**(100),  
 "ReleaseYear" **SMALLINT** NOT NULL  
);  
  
CREATE TABLE "ParkingSeats"  
(  
 "Id" **SMALLINT** PRIMARY KEY,  
 "Price" **MONEY** NOT NULL,  
 "ContractId" **INTEGER**,  
 FOREIGN KEY ("ContractId")  
 REFERENCES "Contracts" ("Id")  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
);

CREATE TABLE "Contracts"  
(  
 "Id" **INTEGER** PRIMARY KEY,  
 "ClientId" **INTEGER** NOT NULL,  
 "CarNumber" **VARCHAR**(9) NOT NULL,  
 "SeatId" **SMALLINT**,  
 "ContractDate" **DATE** NOT NULL,  
 "AccuralDate" **DATE**,  
 "Debt" **MONEY** NOT NULL,  
 "PaymentDate" **DATE**,  
 "PaymentAmount" **MONEY**,  
 FOREIGN KEY ("CarNumber")  
 REFERENCES "Cars" ("Number")  
 ON UPDATE NO ACTION  
 ON DELETE CASCADE,  
 FOREIGN KEY ("ClientId")  
 REFERENCES "Clients" ("Id")  
 ON UPDATE NO ACTION  
 ON DELETE CASCADE,  
 FOREIGN KEY ("SeatId")  
 REFERENCES "ParkingSeats" ("Id")  
 ON UPDATE NO ACTION  
 ON DELETE CASCADE  
);  
  
CREATE INDEX "IX\_Contracts\_CarNumber"  
 ON "Contracts" USING btree ("CarNumber" ASC);  
  
CREATE INDEX "IX\_Contracts\_ClientId"  
 ON "Contracts" USING btree ("ClientId" ASC);  
  
CREATE UNIQUE INDEX "IX\_Contracts\_SeatId"  
 ON "Contracts" USING btree ("SeatId" ASC);  
  
CREATE UNIQUE INDEX "IX\_ParkingSeats\_ContractId"  
 ON "ParkingSeats" USING btree ("ContractId" ASC);

SQL скрипт для формирования отчета #1:

SELECT *client*."Id", CONCAT(*client*."LastName", ' ', *client*."FirstName", ' ', *client*."Patronymic") AS *"FullName"*,  
 (  
 SELECT COALESCE(sum(*contract*."Debt"), 0.0)  
 FROM "Contracts" AS *contract* WHERE *client*."Id" = *contract*."ClientId"  
 ) AS *"TotalDebt"*,  
 (  
 SELECT *contract*."PaymentDate"  
 FROM "Contracts" AS *contract* WHERE *client*."Id" = *contract*."ClientId"  
 ORDER BY *contract*."PaymentDate" DESC  
 LIMIT 1  
 ) AS *"LastPaymentDate"*FROM "Clients" AS *client*ORDER BY (SELECT COALESCE(sum(*contract*."Debt"), 0.0)  
 FROM "Contracts" AS *contract* WHERE *client*."Id" = *contract*."ClientId")  
 DESC  
 LIMIT 1;

SQL скрипты для формирования отчета #2:

Скрипт для получения списка номеров автомобилей, подходящих по условиям:

SELECT *contract*."CarNumber"  
FROM "Contracts" AS *contract* LEFT JOIN "Cars" *car* ON *car*."Number" = *contract*."CarNumber"  
GROUP BY *contract*."CarNumber"  
HAVING COUNT(DISTINCT *contract*."ClientId") > 1;

Скрипт для получения списка владельца автомобиля (номер автомобиля передается в скрипт параметром):

SELECT DISTINCT *client*."Id",  
 CONCAT(*client*."LastName", ' ', *client*."FirstName", ' ', *client*."Patronymic") AS *"FullName"*FROM "Contracts" AS *contract*JOIN "Clients" AS *client* ON *contract*."ClientId" = *client*."Id"  
WHERE *contract*."CarNumber" = @CarNumber;

SQL скрипт для формирования отчета #3:

SELECT *car*."Number", "Brand", "ReleaseYear",  
((SELECT SUM(*PS*."Price")  
 FROM "Contracts" AS *contract* LEFT JOIN "ParkingSeats" *PS* on *PS*."Id" = *contract*."SeatId"  
WHERE *car*."Number" = *contract*."CarNumber" AND *contract*."AccuralDate" BETWEEN @StartInterval AND @EndInterval  
GROUP BY *contract*."CarNumber") -  
 (SELECT SUM(*contract*."PaymentAmount")  
 FROM "Contracts" AS *contract* WHERE *car*."Number" = *contract*."CarNumber" AND *contract*."PaymentDate" BETWEEN @StartInterval AND @EndInterval  
 GROUP BY *contract*."CarNumber")) AS *"TotalDebt"*FROM "Cars" AS *car*ORDER BY *"TotalDebt"*  
LIMIT 1;

SQL скрипт для формирования отчета #4:

SELECT SUM((SELECT SUM(*PS*."Price")  
 FROM "Contracts" AS *CT* LEFT JOIN "ParkingSeats" *PS* on *PS*."Id" = *CT*."SeatId"  
WHERE *contract*."Id" = *CT*."Id" AND *CT*."AccuralDate" BETWEEN @StartInterval AND @EndInterval  
GROUP BY *CT*."CarNumber") -  
 (SELECT SUM(*CT*."PaymentAmount")  
 FROM "Contracts" AS *CT* WHERE *contract*."Id" = *CT*."Id" AND *CT*."PaymentDate" BETWEEN @StartInterval AND @EndInterval  
 GROUP BY *CT*."CarNumber")) AS *"TotalDebt"*FROM "Contracts" AS *contract*;

SQL скрипты для формирования отчета #5:

Скрипт для получения списка номеров автомобилей, выбранного бренда:

SELECT DISTINCT *contract*."CarNumber"  
FROM "Contracts" AS *contract* LEFT JOIN "Cars" *car* ON *car*."Number" = *contract*."CarNumber"  
WHERE *car*."Brand" = @Brand

Скрипт для получения списка владельца автомобиля (номер автомобиля передается в скрипт параметром):

SELECT DISTINCT *client*."Id",  
 CONCAT(*client*."LastName", ' ', *client*."FirstName", ' ', *client*."Patronymic") AS *"FullName"*FROM "Contracts" AS *contract*JOIN "Clients" AS *client* ON *contract*."ClientId" = *client*."Id"  
WHERE *contract*."CarNumber" = @CarNumber;

# ПРИЛОЖЕНИЕ 2

Разработка приложения проводилась в редакторе JetBrains Rider 2022.2.3 на версии .NET Core SDK 6.0.

Файл Program.cs

using AutoCar.Services;  
  
var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddRazorPages();  
builder.Services.AddTransient<ValidationService, ValidationService>();  
var app = builder.Build();  
  
if (!app.Environment.IsDevelopment())  
{  
 app.UseExceptionHandler("/Error");  
}  
  
app.UseStaticFiles();  
app.UseRouting();  
app.MapRazorPages();  
  
app.Run();

Файл PostgresStorage.cs

using System.Data;  
using AutoCar.Models;  
using Microsoft.EntityFrameworkCore;  
using Npgsql;  
  
namespace AutoCar.Storage;  
  
public class PostgresStorage : DbContext  
{  
 public DbSet<Client> Clients { get; set; }  
 public DbSet<Car> Cars { get; set; }  
 public DbSet<ParkingSeat> ParkingSeats { get; set; }  
 public DbSet<Contract> Contracts { get; set; }  
 public PostgresStorage()  
 {  
 Database.EnsureCreated();  
 }  
  
 public async Task<DataTable> ExecuteReportAsync(string queryPath, IReadOnlyDictionary<string, object> parameters = null)  
 {  
 using var dt = new DataTable();  
 NpgsqlConnection connection = null;  
 try  
 {  
 connection = (NpgsqlConnection)Database.GetDbConnection();  
 await connection.OpenAsync();  
 var reportQuery = await File.ReadAllTextAsync(queryPath);  
 await using var command = new NpgsqlCommand(reportQuery, connection);  
 if (parameters != null)  
 foreach (var pair in parameters)  
 command.Parameters.AddWithValue(pair.Key, pair.Value);  
 using var adapter = new NpgsqlDataAdapter(command);  
 adapter.Fill(dt);  
 }  
 finally  
 {  
 if (connection is { State: not ConnectionState.**Closed** }) await connection.CloseAsync();  
 }  
 return dt;  
 }  
  
 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
 {  
 optionsBuilder.EnableSensitiveDataLogging();  
 optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=autocar\_db;Username=postgres;Password=autocar");  
 optionsBuilder.LogTo(Console.WriteLine);  
 }   
}

Файл DateTimeSerivce.cs:

namespace AutoCar.Services;  
  
public static class DateTimeSerivce  
{  
 public static DateOnly GetMaxBirthDate()  
 {  
 var maxDate = DateOnly.FromDateTime(DateTime.Today.AddYears(-18));  
 return maxDate;  
 }  
 public static DateOnly GetMinBirthDate() => new(1900,01,01);  
 public static DateOnly GetTodayDate() => DateOnly.FromDateTime(DateTime.Today);  
 public static string ToHTMLString(this DateOnly date) => date.ToString("yyyy'-'MM'-'dd");  
 public static string ToFormattedString(this DateOnly date) => date.ToString("dd'/'MM'/'yyyy");  
 public static string ToFormattedString(this DateTime date) => date.ToString("dd'/'MM'/'yyyy");  
}

Файл ValidationService.cs:

using System.Text.RegularExpressions;  
  
namespace AutoCar.Services;  
  
public class ValidationService  
{  
 public readonly static short MIN\_CAR\_RELEASE\_YEAR = 1900;  
 public readonly static short MAX\_CAR\_RELEASE\_YEAR = Convert.ToInt16(DateTime.Today.Year);  
 public readonly static string SAMPLE\_CAR\_NUMBER = "А888НА174";  
 public bool PassedAllValidations { get; private set; }  
 private readonly List<string> \_validationMessages;  
  
 public ValidationService()  
 {  
 PassedAllValidations = true;  
 \_validationMessages = new List<string>();  
 }  
 public void ClearHistory()  
 {  
 PassedAllValidations = true;  
 \_validationMessages.Clear();  
 }  
 public IEnumerable<string> GetValidationMessages() => \_validationMessages;  
 public bool ValidateInitials(string input, string fieldName)  
 {  
 var isValid = IsCyrillicInitials(input);  
 if (!isValid) \_validationMessages.Add($"Поле \"{fieldName}\" должно состоять только " +  
 "из букв кириллицы с заглавной буквы!");  
 PassedAllValidations &= isValid;  
 return isValid;  
 }  
 public bool ValidatePhoneNumber(string input)  
 {  
 var isValid = IsPhoneNumber(input);  
 if (!isValid) \_validationMessages.Add("Номер телефона должен быть в формате +7 XXXXXXXXXX!");  
 PassedAllValidations &= isValid;  
 return isValid;  
 }  
 public bool ValidateAndRetrieveBirthDate(string input, out DateOnly outBirthDate)  
 {  
 var isValid = !string.IsNullOrEmpty(input) && DateOnly.TryParse(input, out outBirthDate);  
 if (!isValid) \_validationMessages.Add($"Поле \"Дата рождения\" не должно выходить за диапазон" +  
 $"{DateTimeSerivce.GetMinBirthDate()} и {DateTimeSerivce.GetMaxBirthDate()}!");  
 PassedAllValidations &= isValid;  
 return isValid;  
 }  
 public bool ValidateCarNumber(string input)  
 {  
 var isValid = IsCarNumber(input);  
 if (!isValid) \_validationMessages.Add($"Номер автомобиля должен быть в формате {SAMPLE\_CAR\_NUMBER}!");  
 PassedAllValidations &= isValid;  
 return isValid;  
 }  
 public bool ValidateCarReleaseYear(int input)  
 {  
 var isValid = input >= MIN\_CAR\_RELEASE\_YEAR && input <= MAX\_CAR\_RELEASE\_YEAR;  
 if (!isValid) \_validationMessages.Add("Год выпуска автомобиля не может быть меньше " +  
 $"{MIN\_CAR\_RELEASE\_YEAR} и больше {MAX\_CAR\_RELEASE\_YEAR}!");  
 PassedAllValidations &= isValid;  
 return isValid;  
 }  
  
 private bool IsCarNumber(string input) => Regex.Match(input, @"^[АВЕКМНОРСТУХ]\d{3}(?<!000)[АВЕКМНОРСТУХ]{2}\d{2,3}$").Success;  
 private bool IsCyrillicInitials(string input) => Regex.Match(input, @"^[А-Я]{1}[а-я]+$").Success;  
 private bool IsPhoneNumber(string input) => Regex.Match(input, @"^[0-9]{10}$").Success;  
}

Далее идут файлы каждой формы. В целях сокращения объема документа далее будут только форма списка пользователей, форма добавления пользователя и форма отчета #5.

С остальными аналогичными формами можно ознакомиться в приложенном архиве.

Файл Pages/Shared/\_MasterLayout.cshtml:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="utf-8"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1.0"/>  
 <title>@ViewData["Title"] - AutoCar</title>  
 <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css"/>  
 <**link** rel="stylesheet" **href**="~/css/site.css" **asp-append-version**="true"/>  
</head>  
<body>  
<header>  
 <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">  
 <div class="container">  
 <**a** class="navbar-brand" **asp-area**="" **asp-page**="/Index">AutoCar</**a**>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"  
 aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">  
 <ul class="navbar-nav flex-grow-1">  
 <li class="nav-item">  
 <**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/Index">Клиенты</**a**>  
 </li>  
 <li class="nav-item">  
 <**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/Contracts/Index">Договора, оплаты и начисления</**a**>  
 </li>  
 <li class="nav-item">  
 <**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/Cars/Index">Автомобили</**a**>  
 </li>  
 <li class="nav-item">  
 <**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/ParkingSeats">Парковочные места</**a**>  
 </li>  
 <li class="nav-item">  
 <**a** class="nav-link text-dark" **asp-area**="" **asp-page**="/Reports/Report1">Отчёты</**a**>  
 </li>  
 </ul>  
 </div>  
 </div>  
 </nav>  
</header>  
<div class="container">  
 <main role="main" class="pb-3">  
 @RenderBody()  
 </main>  
</div>  
  
<footer class="border-top footer text-muted">  
 <div class="container">  
 &copy; 2023 - AutoCar - <**a asp-area**="" **asp-page**="/Index">Главная</**a**>  
 </div>  
</footer>  
  
<script src="~/lib/jquery/dist/jquery.min.js"></script>  
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>  
<**script src**="~/js/site.js" **asp-append-version**="true"></**script**>  
  
@await RenderSectionAsync("Scripts", required: false)  
</body>  
</html>

Файл Pages/Index.cshtml:

@page  
@model Pages.IndexModel  
@{  
 ViewData["Title"] = "Главная";  
}  
  
<div class="text-center">  
 <h2 class="display-3">Список клиентов</h2>  
 <**form method**="get" **asp-page**="/Clients/New">  
 <input class="btn btn-primary" type="submit" value="Добавить клиента">  
 </**form**>  
 <table class="table">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Имя</th>  
 <th>Фамилия</th>  
 <th>Отчество</th>  
 <th>Дата рождения</th>  
 <th>Номер телефона</th>  
 </tr>  
 </thead>  
 <tbody>  
 @foreach (var clientModel in Model.Clients)  
 {  
 <tr id="@clientModel.Id">  
 <td>@clientModel.Id</td>  
 <td>@clientModel.FirstName</td>  
 <td>@clientModel.LastName</td>  
 <td>@clientModel.Patronymic</td>  
 <td>@clientModel.BirthDate.ToFormattedString()</td>  
 <td>@clientModel.PhoneNumber</td>  
 <td>  
 <**a asp-page**="/Clients/Edit" asp-route-id="@clientModel.Id">Редактировать</**a**>  
 </td>  
 </tr>  
 }  
 </tbody>  
 </table>  
</div>

Файл Pages/Index.cshtml.cs:

using AutoCar.Storage;  
using Microsoft.AspNetCore.Mvc.RazorPages;  
  
namespace AutoCar.Pages  
{  
 public class IndexModel : PageModel  
 {  
 public IEnumerable<Models.Client> Clients { get; private set; } = Enumerable.Empty<Models.Client>();  
  
 public void OnGet(int listPage)  
 {  
 using (var storage = new PostgresStorage()) Clients = storage.Clients.ToArray();  
 }  
 }  
}

Файл Pages/Clients/New.cshtml:

@page  
@model NewClientModel  
@{  
 ViewData["Title"] ="Новый клиент";  
 var maxDate = DateTimeSerivce.GetMaxBirthDate().ToHTMLString();  
}  
<div class="text-center">  
 <h2 class="display-3">@ViewData["Title"]</h2>  
 @foreach (var message in Model.ValidationMessages)  
 {  
 <h4 class="alert-danger">@message</h4>   
 }  
 <**form method**="post" **asp-antiforgery**="true">  
 <div class="mb-3">  
 <**label asp-for**="Client.FirstName" class="form-label">Имя</**label**><br/>  
 <**input type**="text" **asp-for**="Client.FirstName" class="form-control-sm"/>  
 </div>  
 <div class="mb-3">  
 <**label asp-for**="Client.LastName" class="form-label">Фамилия</**label**><br/>  
 <**input type**="text" **asp-for**="Client.LastName" class="form-control-sm"/>  
 </div>  
 <div class="mb-3">  
 <**label asp-for**="Client.Patronymic" class="form-label">Отчество</**label**><br/>  
 <**input type**="text" **asp-for**="Client.Patronymic" class="form-control-sm"/>  
 </div>  
 <div class="mb-3">  
 <**label asp-for**="Client.BirthDate" class="form-label">Дата рождения</**label**><br/>  
 <**input type**="date"  
 min="@DateTimeSerivce.GetMinBirthDate().ToHTMLString()"  
 max="@maxDate" **value**="@maxDate" **asp-for**="NewBirthDate" class="form-control-sm"/>  
 </div>  
 <div class="mb-3">  
 <**label asp-for**="Client.PhoneNumber" class="form-label">Номер телефона</**label**><br/>  
 <label style="margin-right: 2pt">+7</label>  
 <**input type**="tel" minlength="10" maxlength="10" **asp-for**="Client.PhoneNumber" class="form-control-sm"/>  
 </div>  
 <input type="submit" class="btn btn-primary" value="Сохранить"/>  
 </**form**>  
</div>

Файл Pages/Clients/New.cshtml.cs:

using AutoCar.Services;  
using AutoCar.Storage;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Mvc.RazorPages;  
  
namespace AutoCar.Pages.Clients;  
  
public class NewClientModel : PageModel  
{  
 [BindProperty] public Models.Client Client { get; set; }  
 [BindProperty] public string NewBirthDate { get; set; }  
 public IEnumerable<string> ValidationMessages { get; private set; } = Enumerable.Empty<string>();  
 public void OnGet()  
 {  
 }  
 public IActionResult OnPost([FromServices] ValidationService service)  
 {  
 service.ValidateInitials(Client.FirstName, "Имя");  
 service.ValidateInitials(Client.LastName, "Фамилия");  
 service.ValidateInitials(Client.Patronymic, "Отчество");  
 service.ValidatePhoneNumber(Client.PhoneNumber);  
 service.ValidateAndRetrieveBirthDate(NewBirthDate,out var newBirthDate);  
 Client.BirthDate = newBirthDate;  
 if (service.PassedAllValidations)  
 {  
 using (var storage = new PostgresStorage())  
 {  
 storage.Clients.Add(Client);  
 storage.SaveChanges();  
 }  
 }  
 ValidationMessages = service.GetValidationMessages();  
 return service.PassedAllValidations ? RedirectToPage("/Index") : Page();  
 }  
   
}

Файл Pages/Reports/Report5.cshtml:

@page  
@model AutoCar.Pages.Reports.Report5  
@{  
 Layout = "\_ReportMasterLayout";  
 ViewData["Title"] = "Отчёт #5";  
}  
<div class="text-center">  
 <h2 class="display-3">@ViewData["Title"]</h2>  
 <p>Отчет выводит все номера и владельцев автомобилей, относящиеся к указанной пользователем марки автомобиля.</p>  
 <**form asp-antiforgery**="true">  
 <div class="mb-3">  
 <label for="brand" class="form-label">Марка авто: </label>  
 <input type="text" name="brand" id="brand" class="form-control-sm">  
 </div>  
 <input class="btn btn-primary" type="submit" value="Сформировать отчёт">  
 </**form**>  
 @if (Model.Report != null)  
 {  
 <h5>Результат поиска по марке @Model.Report.Brand:</h5>  
 @foreach (var car in Model.Report.Cars)  
 {  
 <table class="table">  
 <thead>  
 <tr><th colspan="2">Владельцы @Model.Report.Brand @car.CarNumber:</th></tr>  
 </thead>  
 <tbody>  
 <tr>  
 <th>ID Клиента</th>  
 <th>ФИО Клиента</th>  
 </tr>  
 @foreach (var clientData in car.Clients)  
 {  
 <tr>  
 <td>@clientData.Id</td>  
 <td>@clientData.FullName</td>  
 </tr>  
 }  
 </tbody>  
 </table>   
 }   
 }  
</div>

Файл Pages/Reports/Report5.cshtml.cs:

using System.Data;  
using AutoCar.Storage;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Mvc.RazorPages;  
  
namespace AutoCar.Pages.Reports;  
public class Report5 : PageModel  
{  
 public record ReportData(string Brand, IEnumerable<MultiClientCarData> Cars);  
 public ReportData Report { get; private set; }  
  
 public void OnGet()  
 {  
 }  
 public async Task OnPost([FromServices] IWebHostEnvironment appEnvironment, string brand)  
 {  
 await using var storage = new PostgresStorage();  
 var multiClientCars = new List<MultiClientCarData>();  
 if (string.IsNullOrEmpty(brand))  
 {  
 Report = new ReportData(brand, multiClientCars);  
 return;  
 }  
 using var carsTable = await storage.ExecuteReportAsync(Path.Combine(appEnvironment.WebRootPath, "queries", "report5.sql"),  
 new Dictionary<string, object> {{ "Brand", brand }});  
 var clientQueryParams = new Dictionary<string, object>();  
 foreach (DataRow multiClientCar in carsTable.Rows)  
 {  
 var carNumber = multiClientCar["CarNumber"].ToString();  
 clientQueryParams["CarNumber"] = carNumber;  
 using var clientsTable = await storage.ExecuteReportAsync(Path.Combine(appEnvironment.WebRootPath,  
 "queries", "report2\_2.sql"), clientQueryParams);  
 multiClientCars.Add(new MultiClientCarData(carNumber, ClientData.ExtractClientsDataFromTable(clientsTable)));  
 }  
 Report = new ReportData(brand, multiClientCars);  
 }  
}