**Содержание**

Введение……………………………………………………………………………...4

1 Анализ предметной области………………………………………………………7

1.1 Анализ существующих аналогов……………………………………………….7

1.2 Постановка задачи……………………………………………………………….7

2 Математические модели и методы положенные в основу программного средства и теоретические исследования…………………………………………..............8

3 Разработка программного средства………………………………………………9

3.1 Разработка спецификации требований…………………………………………9

3.2 Разработка архитектуры программного средства……………………………..9

3.3 Обоснование выбора языка программирования……………………………...10

3.4 Разработка модели файловой системы………………………………………..13

3.5 Разработка схемы алгоритма…………………………………………………..15

4 Тестирование программного средства………………………………………….17

5 Методика использования программного средства………………………….….19

Заключение………………………………………………………………………....21

Список использованных источников……………………………………………..22

**Введение**

В настоящее время компьютеры прочно обосновались практически во всех сферах деятельности человека. Этот факт вынуждает человека с все большей интенсивностью использовать в своей повседневной жизни компьютер. Если сравнить работу Госавтоинспекции сейчас и, скажем, 10 лет назад, то можно заметить массу изменений в работе.

Автоматическое регулирование светофора, в зависимости от загруженности движения; камеры видео наблюдения на дорогах, у оператора, следящего за камерами, есть возможность проверять информацию, отслеживать ситуацию и оперативно вмешиваться, если необходимо, например, отрегулировать затор; камеры фото фиксации, когда автоматически прибор определяет скорость движения водителей и фиксирует превышение скоростного режима, путем фотографирования; появление системы автоматизированной оплаты дорог; собственный сайт Госавтоинспекции, c возможностями удаленно просматривать свои штрафы-все это говорит о постоянном внедрении компьютеров и гаджетов в нашу жизнь. Без информатики уже немыслима современная человеческая культура и профессиональная деятельность человека. C развитием компьютерных технологий все больше информации распространяется по компьютерным сетям с использованием различных компьютерных систем. Все шире используются информационные ресурсы всемирной сети Internet.

Роль ЭВМ в жизни людей стремительно возрастает. Круг пользователей ими включает не только профессиональных программистов, но и специалистов различных областей деятельности. Инспекторы ГАИ, например, отошли от регистрации транспортных средств, путем записи в журналы регистрации. Использование ЭВМ позволяет значительно упростить инспекторам работу в поиске информации и ее добавлении, а также помогает им не отставать от требований нынешнего века.

В данном курсовом проекте поставлена задача разработать программное средство для автоматизации учета транспортных средств ГАИ, которое наиболее удобно можно организовать с помощью баз данных (БД).

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая не имела бы в качестве основы или важной составляющей базу данных. Концепции и технологии баз данных складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации.

Проектирование программного средства «Автоматизация учета транспортных средств ГАИ» ведется для упрощения работы инспектора ГАИ: отслеживания информации об автомобилях и их регистрации, быстрого поиска информации о марке, технических характеристиках, владельцах и страховках автомобилей.

Данная пояснительная записка является полным описанием прилагаемого разработанного приложения и разделена на разделы и подразделы. Каждый раздел посвящен определенному процессу изучения поставленной задачи и разработки приложения.

Раздел «Постановка задачи» будет содержать описание предметной области, входные и выходные данные, а также требования к программе.

Раздел «Математические модели и теоретические исследования» будет включать описание, обоснование и краткую характеристику выбранных программных средств для разработки курсового проекта; описание разработки алгоритма решения задачи.

Раздел «Разработка программного средства» будет описывать сущность структуры проекта, а так же методы и классы, использованные в программе.

Раздел «Тестирование» будет описывать ход тестирования программы. Также в нём будет содержаться описание ошибок, которые могут быть обнаружены в ходе создания программы, а также описание путей их исправления.

Итогом всей работы станет заключение, анализирующее, как выполнена поставленная задача, степень соответствия проектных решений заданию, описан процесс и результаты разработки программы, а так же возможные пути ее практического применения.

**1.1 Анализ существующих аналогов**

При изучении существующих аналогов были изучены все плюсы и недостатки.

Главной сущностью будет выступать добавление, удаление, а также  изменение информации о клиенте. Личной информацией будут выступать такие данные, как: имя, фамилия, отчество, адрес, номер телефона владельца, номер авто, марка, модель, техосмотр.

Данная программа позволяет упростить работу сотрудника ГАИ.

* 1. **Постановка задачи**

Разработка программного средства для автоматизации работы сотрудника ГАИ. Данное приложение демонстрирует работу в сфере учета транспортных средств.

Анализ предметной области показал, что реализуемое программное средство имеет небольшое количество аналогов, но при этом одинаковые недостатки.

В связи с этим на курсовое проектирование поставлены задачи:

1. Реализовать фильтр по автомобилям и владельцам.
2. Реализовать отображение всей доступной информации.
3. **Математические модели и методы положенные в основу программного средства и теоретические исследования**

Программное средство учета транспортных средств имеет интуитивно-понятный интерфейс, удобною навигацию, оно просто в освоении, в основу проектирования данного ПС положены самые современные технологии и популярные решения, благодаря чему, оно соответствует всем современным требованиям.

Пользователь выбирает действие, после чего согласно алгоритму каждое действие выполняет свои функции.

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Разработка спецификаций требований**

Данный курсовой проект разрабатывался для безотказной и стабильной работы. В основу разработки положены функции фильтрации, добавления информации, удаления информации, создания модулей, пагинации.

Программное средство учета транспортных средств выполняет все установленные требования и удовлетворяет все базовые потребности пользователя.

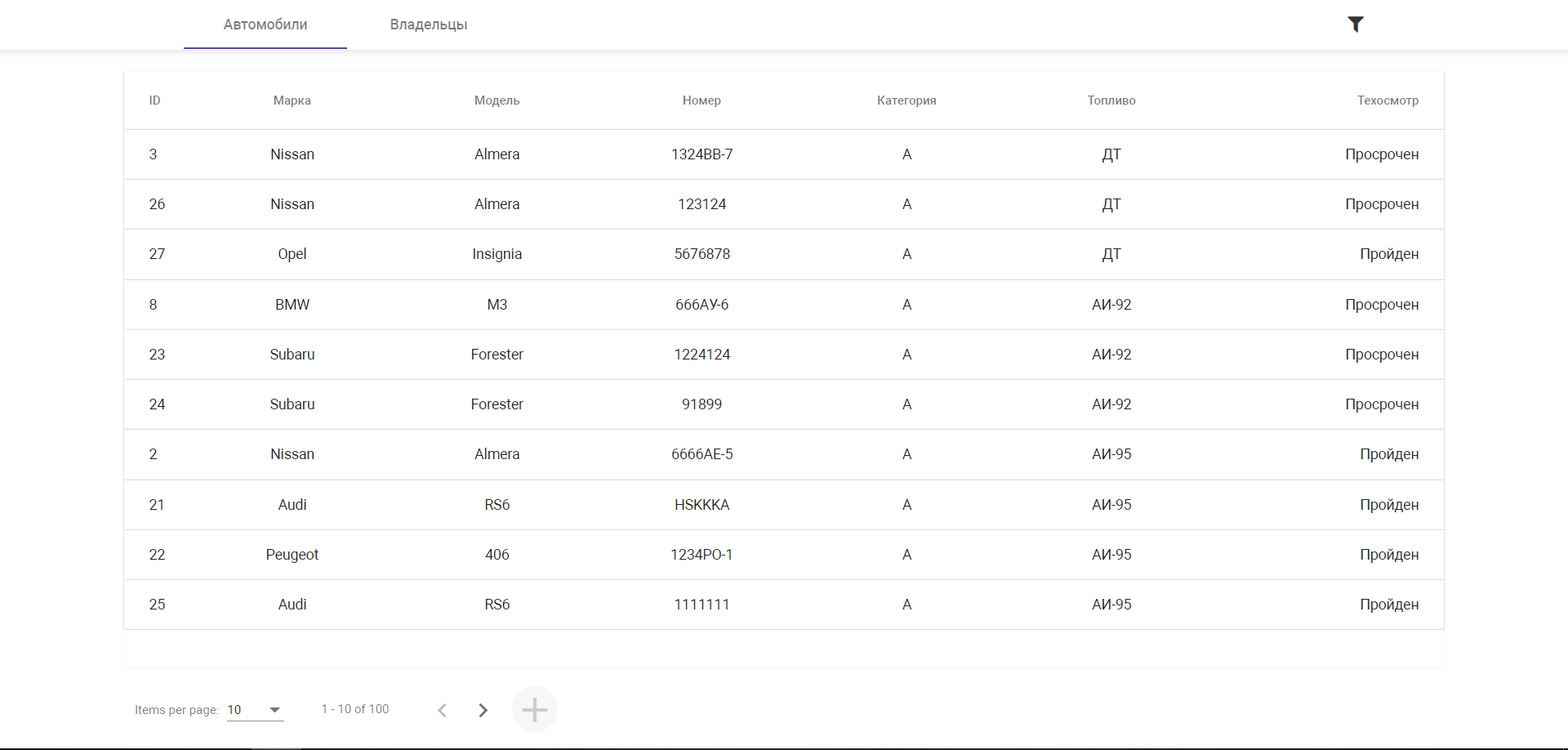


Рисунок 3.1 – Главное окно

**3.2 Разработка архитектуры программного средства**

Интерфейс приложения представляет собой одну форму, на которой и осуществляется весь процесс работы. Схема интерфейса представлена на рисунке 3.2.

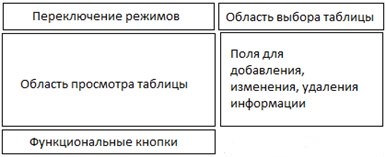


Рисунок 3.2 – Схема интерфейса

На форме присутствуют элементы управления: поля для ввода текста, кнопки, поле для отображения таблиц, радиокнопки.

Данные отображаются в поле для таблицы. Таблица загружается из базы данных SQL и отображается. Выбор таблицы, которая будет загружаться осуществляется вводом названия таблицы.

После ввода названия таблицы надо нажать на кнопку, после которой таблица и отобразится в поле вывода таблицы. После этого с помощью радиокнопок можно изменять выбор действия, который можно совершить над таблицей. Они расположены в верхнем углу.

В зависимости от выбранной радиокнопки на форме становятся видимыми поля и кнопки.

При выборе радиокнопки поиска отображаются поля для ввода значений и кнопка, при нажатии на которую происходит поиск нужных записей. Данные результата поиска выводятся в поле для вывода таблиц в виде таблицы. Такие же элементы выводятся при нажатии на радиокнопку «Добавить».

При просмотре таблице на форме активны кнопки экспорта. Эти кнопки расположены справа и дают возможность экспортировать заданную таблицу в файлы Excel и PDF.

Программа имеет основной класс Form1 , в котором описан весь процесс работы и основные методы.

Для Form1 (рабочее пространство):

* + private void button1\_Click – метод, который выполняет вывод таблицы и подготовку полей для использования основных функций программы;
  + private void button2\_Click – метод, выполняющий добавление записей в таблицу;
  + private void button3\_Click – метод, который выполняет изменение выбранной записи;
  + private void button4\_Click – функция, в которой реализована обработка удаления выбранной записи из таблицы;
  + private void button5\_Click – с помощью данного метода в программе осуществляется поиск записей;

**3.3   Обоснование выбора языка программирования**

Angular 5 – Современный популярный js-фреймворк, разрабатываемый компанией Microsoft является одним из лидеров на рынке front-end разработки, все современные решения и любая современная функциональность – у этого фреймворка есть все, чтобы считаться одним из лучших вариантом разработки клиентской части приложения.

Html - стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык HTML до 5-й версии определялся как приложение SGML (стандартного обобщённого языка разметки по стандарту ISO 8879). Спецификации HTML5 формулируются в терминах DOM (объектной модели документа).

Язык XHTML является более строгим вариантом HTML, он следует синтаксису XML и является приложением языка XML в области разметки гипертекста.

Во всемирной паутине HTML-страницы, как правило, передаются браузерам от сервера по протоколам HTTP или HTTPS, в виде простого текста или с использованием шифрования.

Css - формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL.

Node.js - программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

MySql - свободная реляционная система управления базами данных[6]. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

# Разработка модели файловой системы

Структура данного программного средства состоит из следующих компонентов:

* директория Views;
* директория Models;
* директория Controllers;
* форма Autos;
* форма People;
* форма Filter;
* форма Pagination;
* форма Service;

Каждая из этих форм состоит файла с Typescript кодом и кода html.

Как видно, все описанные выше элементы присутствуют.

Следующим действием будет добавление нового адреса в базу данных. Для этого в списке таблиц требуется выбрать «Авто» и нажать кнопку «Добавить».

Стоит заметить, что программа не содержит все эти формы для каждой таблицы, вместо этого был разработан шаблон генерации форм добавления и обновления, в зависимости от данных, в следствии чего, была достигнута значительная гибкость и расширяемость данного приложения, которые позволяют использовать одну и ту же программу или ее части для различных баз данных и задач. Поэтому в разработке программы требовалось достигнуть некоторой универсальности.

Далее будет продемонстрирована форма редактирования, она очень похожа на форму добавления, разница только в том, что в полях уже есть предыдущие данные, которые вы можете изменять, чтобы вызвать данную форму, вам потребуется нажать на кнопку «редактировать», которая находится в правой крайней колонке на нужной вам записи.

Поиск в базе данных - это отбор записей, удовлетворяющих условиям поиска, заданным в форме фильтра или запроса.

Фильтры и запросы позволяют отбирать записи, которые удовлетворяют условиям поиска. Условия поиска записей создаются с использованием операторов сравнения

Если пользователь закончил работу с приложением и ему больше ничего не надо, то он может выйти из приложения нажатием на стандартную кнопку закрытия окна Windows в правом верхнем углу окна.

Также программное средство имеет функцию выполнения пользовательских запросов если они не предусмотрены в базовом функционале.

# Разработка схемы алгоритма

Понятие алгоритма так же фундаментально для информатики, как и понятие информации. Существует много различных определений алгоритма, так как это понятие достаточно широкое и используется в различных областях науки, техники и повседневной жизни.

Алгоритм – понятная и точная  последовательность действий, описывающая процесс преобразования объекта из начального состояния в конечное.

Программа -это описание алгоритма и данных на некотором языке программирования, предназначенное для последующего автоматического выполнения.

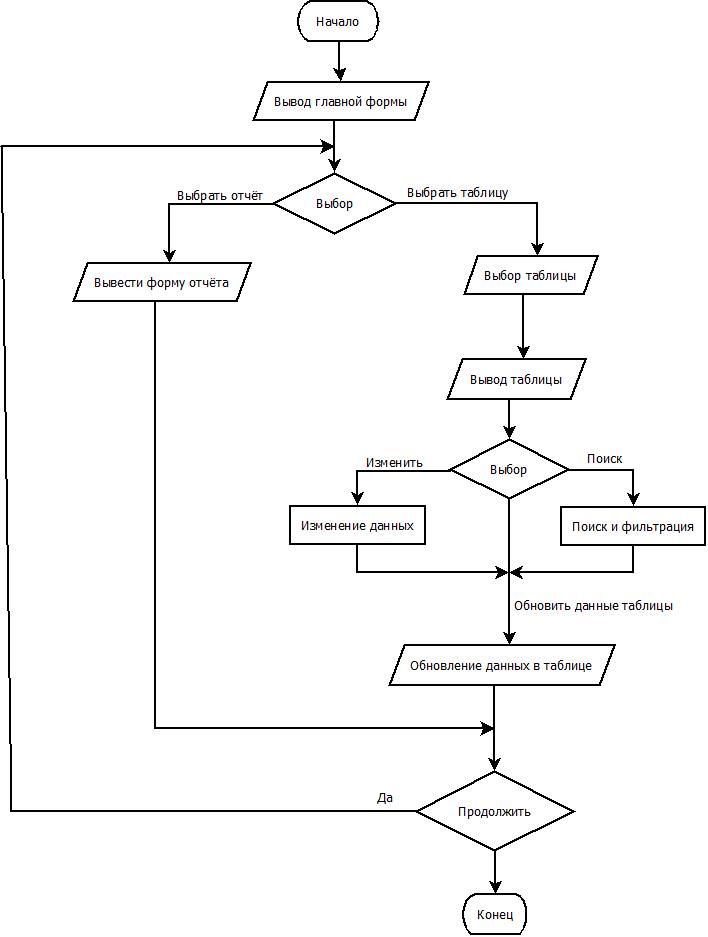


Рисунок 4 – Алгоритм работы программы

На рисунке 5, показан алгоритм работы программы, который отражает последовательность и логику выполнения операций обработки информации. Алгоритм, показанный на рисунке выше, также отображает основные функции программного средства.

**4  Тестирование программного средства**

Тестирование ПО это процесс исследования программного обеспечения с целью выявления ошибок и проверки его качества. Также тестирование ПО можно описать как процесс валидации и верификации того или иного программного продукта, чтобы узнать, на сколько точно он удовлетворяет всем техническим требованиям.

В процессе разработки данного программного средства было проведено тестирование. В таблице 4.1 представлены результаты тестирования.

Таблица 4.1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| Действие | Ожидаемый результат | Фактический результат |
| Переход на главную страницу | Отображение информации об автомобилях | Пройдено |
| Нажать кнопку “владельцы” | Отображение информации о владельцах | Пройдено |
| Нажать на кнопку “Удалить” | Производится удаление выбранной строки | Пройдено |
| Нажать на строку | Отображается форма “Изменение авто” | Пройдено |
| Нажать на кнопку “фильтр ” | Производится фильтр | Пройдено |
| Нажать на кнопку “Выход” | Выход из приложения | Пройдено |

В ходе тестирования выявился ряд ошибок. Большая часть ошибок возникала в связи с некорректным составлением запросов для базы данных. После полной проверки полей для ввода и принципа составления запросов все ошибки были устранены.

**5  Методика использования программного средства**

Программа «Учет транспортных средств» предназначена для упрощенной работы сотрудника ГАИ.

Программа должна реализовывать функции:

* Добавления авто.
* Добавления владельца.
* Изменение информации об авто.
* Изменение информации о владельце.
* Хранение информации об авто.
* Удаление устаревшей информации из БД.
* Фильтр.

Она может быть доступна для любого пользователя. Возрастные ограничения на нее не распространяются.

Минимальные системные требования:

* + OC Windows XP/Vista/7/8/10;
  + ОЗУ 512 мб;
  + место на жестком диске: 20 мб.

При наличии минимальных системных требований на персональном компьютере проблем с использованием программы не возникнет.

Интерфейс программы имеет простой дизайн. Все кнопки и ключевые взаимодействующие элементы расположены в, присущих для них, местах и не разбросаны по форме для удобства использования программы.

Запуск приложения осуществляется двойным нажатием кнопки мыши по исполняемому файлу.

После запуска исполняемого файла, перед пользователем появится главное окно программы (рис. 5.1).

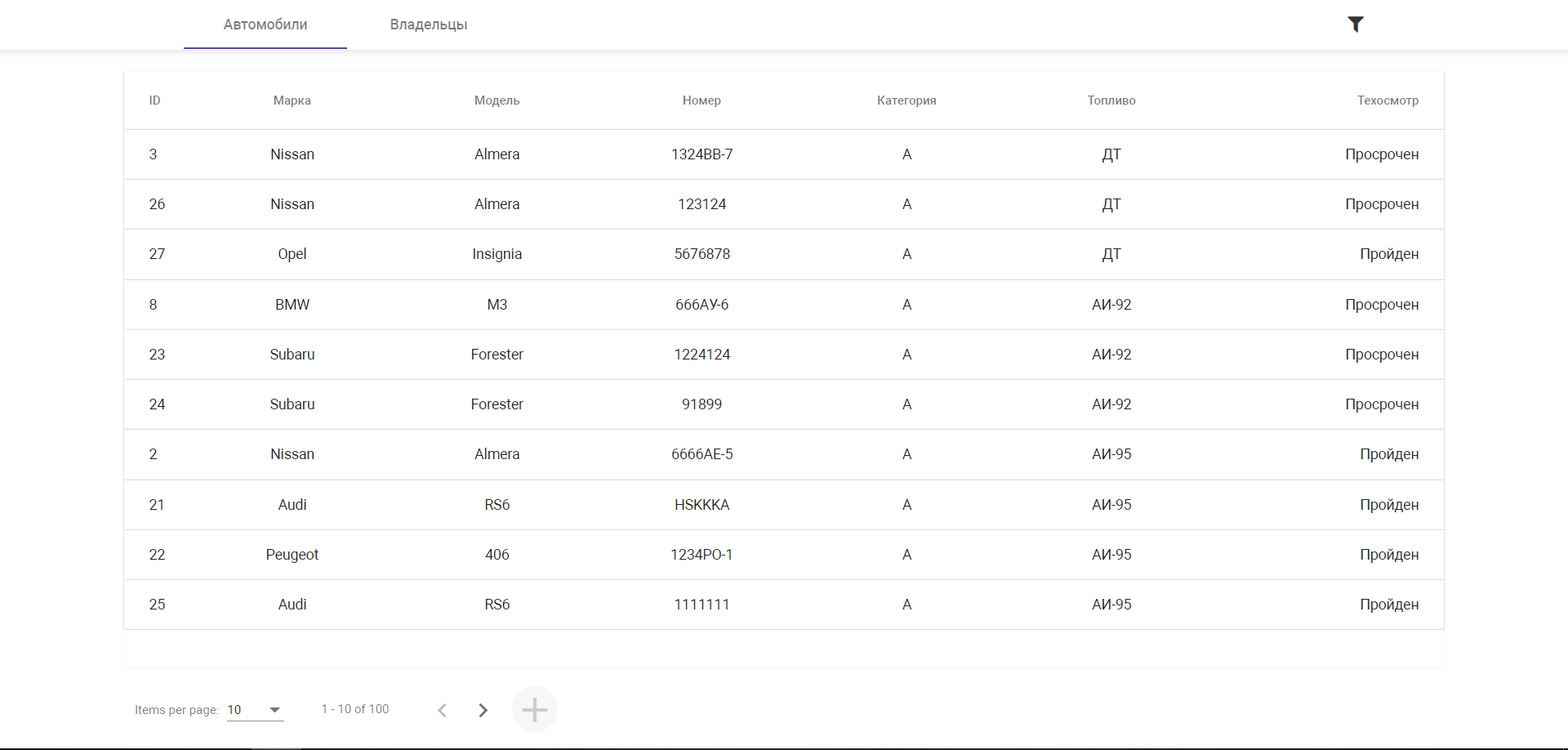


Рисунок 5.1 – Главная форма приложения

Работа с владельцами производиться в окне “Владельцы” (рис. 5.2).

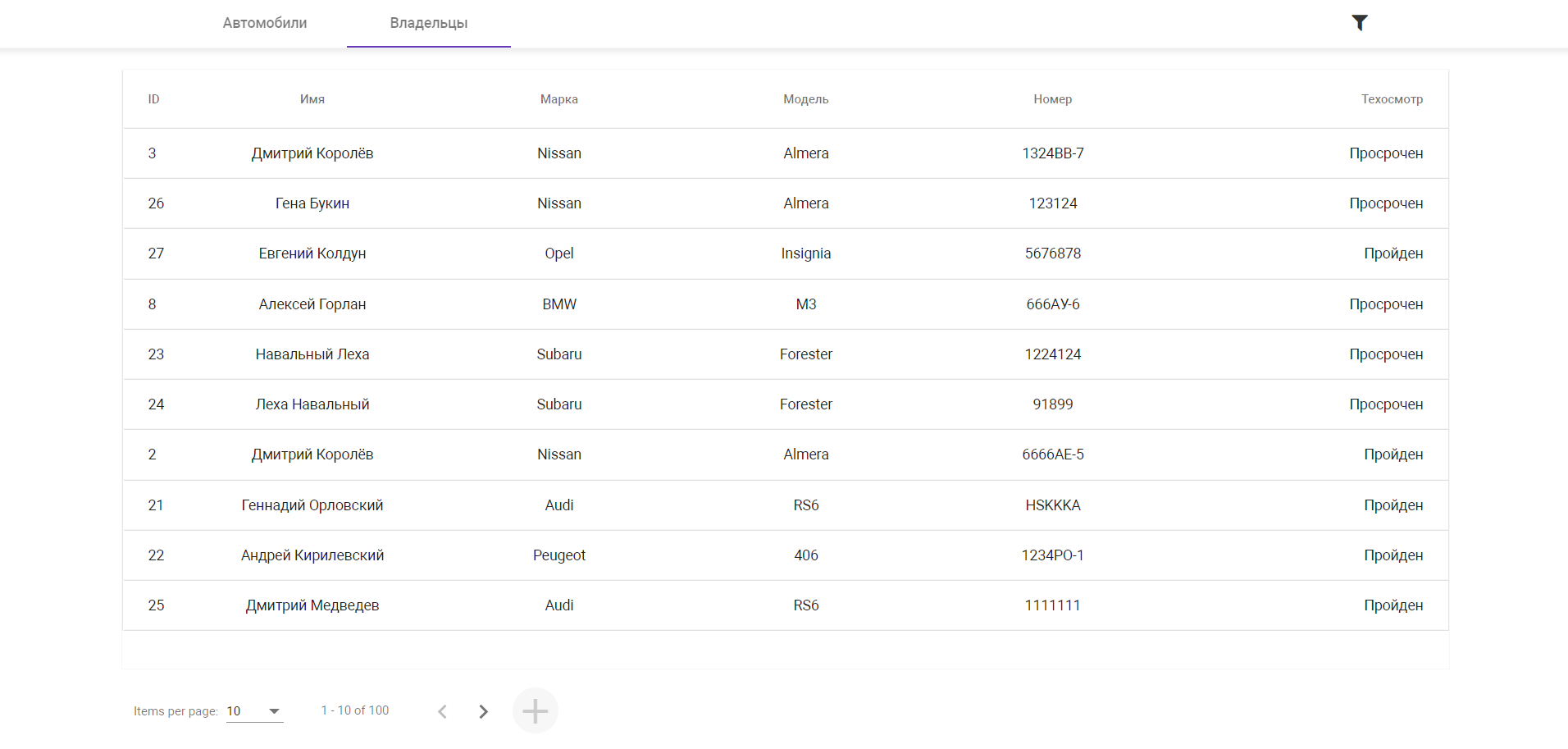


Рис. 5.2 Окно клиенты

Добавление нового авто в базу данных осуществляется путем нажатия кнопки «+». После нажатия отобразится окно добавления информации об авто (рис. 5.3).

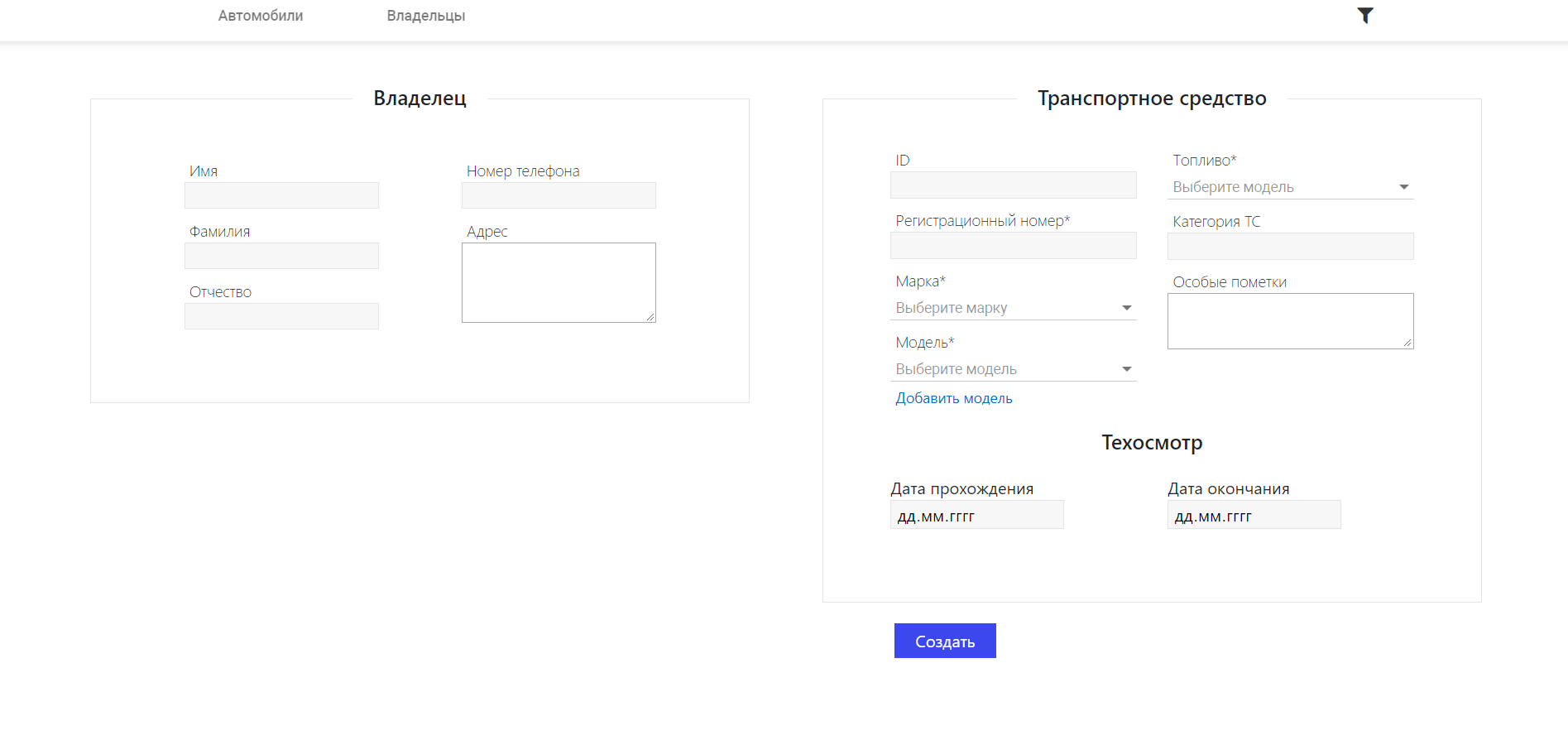


Рис. 5.3. Добавление авто

Для добавления записи необходимо заполнить все поля и нажать кнопку “Создать”. Если не все поля заполнены, то программа выдаст ошибку и добавление записи не произойдет.

Для изменения существующей записи нужно нажать на саму запись. После этого появится окно с информацией о клиенте, где вы сможете изменить необходимые данные. Для сохранения изменений необходимо нажать кнопку “Сохранить”.

Для фильтрации необходимо нажать кнопку фильтров. (Рис. 5.6):

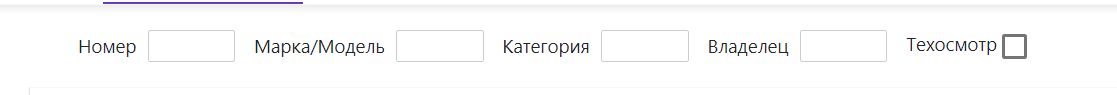


Рис. 5.6 фильтрация

**Заключение**

В деловой или личной сфере часто приходится работать с данными из разных источников, каждый из которых связан с определённым видом деятельности. Для координации всех этих данных необходимы определённые знания и организационные навыки. Microsoft Access объединяет сведения из разных источников в одной реляционной базе данных. Создаваемые формы, запросы и отчёты позволяют быстро и эффективно обновлять данные, получать ответы на вопросы, осуществлять поиск нужных данных, анализировать данные, печатать отчёты, диаграммы и пр.

В результате работы по курсовому проекту была спроектирована информационная система и создан ее прототип. Использование данной системы позволит более эффективно управлять работой турагентства. Снизить случайные ошибки связанных с ручным заполнением форм. Автоматизировать обработку информации в системе. И повысить скорость работы менеджера по продажам при работе с клиентами.

В результате проведенной работы была достигнутая поставленная цель: разработка информационной системы учета транспортных средств. Данное приложение не охватывает всю бизнес-логику этого бизнеса, однако является прототипом, демонстрирующим работу в данной отрасли, и может быть расширена для автоматизации не рассмотренных в рамках данного курсового проекта концепций в предметной области «Учет транспортных средств».

**Список использованных источников**

1. Архангельский А.Я. С++Builder 6. Справочное пособие. Книга 1.Язык С++. –М.: Бином-Пресс, 2002 г. -544 с.: ил.
2. Архангельский А.Я. С++Builder 6. Справочное пособие. Книга 2. Классы и компоненты. М.: Бином-Пресс, 2002 г. -528 с.: ил.
3. Хомоненко А.Д., Ададуров С.Е. Работа с базами данных в С++Builder. –СПб.: БХВ-Петербург, 2006. -496 с: ил.
4. Культин Н.Б. Самоучитель С++Builder. – СПб.: БХВ-Петербург, 2005. – 320с.: ил.
5. Культин Н.Б. С/С++ в задачах и примерах. – СПб.: БХВ-Петербург, 2006. – 288с.: ил.
6. BorlandBuilder C++. Освой самостоятельно. – СПб.: Питер, 2006 – 570с.: ил.
7. Холзнер С. Visual C++ 6. Учебный курс. – СПб.: Питер, 2006. – 570.: ил.
8. Красиков И.В., Баранова И.Е. Алгоритмы. Просто как дважды два. –М.: Эксмо, 2006. -256с.: ил.
9. Карпов Б., Баранова Т. С++. Специальный справочник (2-е издание). – СПб.: Питер, 2005. – 381с.: ил.
10. Франко П. С++: Учебный курс. – СПб.: Питер, 2005. – 522с.: ил.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ**

<div *class=*"table-wrapper">  
 <mat-table *#table [dataSource]=*"dataSource">  
  
 *<!-- Position Column -->* <ng-container *matColumnDef=*"id\_auto">  
 <mat-header-cell *\*matHeaderCellDef*> ID </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.id\_auto}} </mat-cell>  
 </ng-container>*<!--<ng-container matColumnDef="position">  
 <mat-header-cell \*matHeaderCellDef> ID </mat-header-cell>  
 <mat-cell \*matCellDef="let element"> {{element.position}} </mat-cell>  
 </ng-container>-->  
  
 <!-- Name Column -->* <ng-container *matColumnDef=*"year">  
 <mat-header-cell *\*matHeaderCellDef*> Год </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.year}} </mat-cell>  
 </ng-container>  
  
 <ng-container *matColumnDef=*"oil">  
 <mat-header-cell *\*matHeaderCellDef*> Топливо </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.oilName}} </mat-cell>  
 </ng-container>  
  
 <ng-container *matColumnDef=*"checkUp">  
 <mat-header-cell *\*matHeaderCellDef*> Техосмотр </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.id\_checkup?'Пройден':'Просрочен'}} </mat-cell>  
 </ng-container>  
  
 *<!-- Weight Column -->* <ng-container *matColumnDef=*"number">  
 <mat-header-cell *\*matHeaderCellDef*> Номер </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.number}} </mat-cell>  
 </ng-container>  
  
 *<!-- Symbol Column -->* <ng-container *matColumnDef=*"category">  
 <mat-header-cell *\*matHeaderCellDef*> Категория </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.categoryName}} </mat-cell>  
 </ng-container>  
  
 <ng-container *matColumnDef=*"brand">  
 <mat-header-cell *\*matHeaderCellDef*> Марка </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.brandName}} </mat-cell>  
 </ng-container>  
  
 <ng-container *matColumnDef=*"model">  
 <mat-header-cell *\*matHeaderCellDef*> Модель </mat-header-cell>  
 <mat-cell *\*matCellDef=*"let element"> {{element.modelName}} </mat-cell>  
 </ng-container>  
  
 <mat-header-row *\*matHeaderRowDef=*"displayedColumns"></mat-header-row>  
 <mat-row *(click)=*"showAuto(row)" *\*matRowDef=*"let row; columns: displayedColumns;"></mat-row>  
 </mat-table>  
  
</div>

*const* connection = require('./../db').connection;  
*const* Sequilize = require('sequelize');  
  
exports.all = *function* (params,cb) {  
  
 *let* filterObj = JSON.parse(params.filterObj);  
  
 selectAutosQuery = 'SELECT auto.id\_auto, auto.number, p.name `personName`, p.secondName, p.thirdName, \n' +  
 'p.address, ch.id\_checkup, ch.dateEnd, m.id\_model,m.name `modelName`, br.id\_brand,\n' +  
 'br.name `brandName`, oil.id\_oil, oil.name `oilName`, category.id\_category, category.name `categoryName` \n' +  
 'FROM auto\n' +  
 'Inner join person p on p.id\_person = auto.id\_person\n' +  
 'left join checkup ch on ch.id\_checkup = auto.id\_checkup\n' +  
 'inner join model m on m.id\_model = auto.id\_model\n' +  
 'inner join brand br on br.id\_brand = m.id\_brand\n' +  
 'inner join oil on oil.id\_oil = auto.id\_oil\n' +  
 'inner join category on category.id\_category = m.id\_category\n';  
 console.log(params);  
 *if*(filterObj && filterObj.person && filterObj.person.name){  
 selectAutosQuery += 'where (p.name like \'%'+filterObj.person.name+'%\' or ' +  
 'p.secondName like \'%'+filterObj.person.name+'%\') ';  
 }  
 *if*(filterObj && filterObj.number){  
 selectAutosQuery += 'where (auto.number like \'%'+filterObj.number+'%\') ';  
 }  
 *if*(filterObj && filterObj.model && filterObj.model.name){  
 selectAutosQuery += 'where (m.name like \'%'+filterObj.model.name+'%\' or br.name like \'%'+filterObj.model.name+'%\') ';  
 }  
 selectAutosQuery += 'Limit ' + params.limit + ' Offset '+ ((params.offset) \* params.limit) + ' ';  
  
 connection.query(selectAutosQuery, { type: Sequilize.QueryTypes.SELECT }).then( data => {  
 cb(data);  
 });  
};  
  
exports.create = *async function* (auto, cb) {  
  
 *if*(!auto.person.id\_person){  
 *let* addPersonQuery = 'insert into person (`name`,`secondName`,`thirdName`, `address`, `phone`) ' +  
 'values (?,?,?,?,?);';  
 *let* promiseAddPerson = *new* Promise( (res, rej) => {  
 connection.query(addPersonQuery, {replacements:  
 [auto.person.name, auto.person.secondName, auto.person.thirdName, auto.person.address, auto.person.phone],  
 type: Sequilize.QueryTypes.INSERT  
 }).then( () => {  
 res();  
 });  
 });  
 *await* promiseAddPerson;  
 *let* getIdPersonQuery = 'select id\_person from person where person.name =? AND person.secondName=? AND person.thirdName=?';  
 *let* promiseGetIdPerson = *new* Promise( (res, rej) => {  
 connection.query(getIdPersonQuery, {  
 replacements: [auto.person.name, auto.person.secondName, auto.person.thirdName],  
 type: Sequilize.QueryTypes.SELECT  
 }).then( results => {  
 auto.person.id\_person = results[0].id\_person;  
 res();  
 });  
 });  
 *await* promiseGetIdPerson;  
 }  
 *if*(auto.checkup && !auto.checkup.id\_checkup){  
 *let* addCheckUpQuery = 'insert into checkup (`dateStart`,`dateEnd`) values (?,?);';  
 *let* promiseAddCheckUp = *new* Promise( (res, rej) => {  
 connection.query(addCheckUpQuery, {replacements:  
 [auto.checkup.dateStart, auto.checkup.dateEnd],  
 type: Sequilize.QueryTypes.INSERT  
 }).then( () => {  
 res();  
 });  
 });  
 *await* promiseAddCheckUp;  
 *let* getIdCheckUp = 'select id\_checkup from checkup where checkup.dateStart =? AND checkup.dateEnd=?;';  
 *let* promiseGetIdCheckUp = *new* Promise( (res, rej) => {  
 connection.query(getIdCheckUp, {  
 replacements: [auto.checkup.dateStart, auto.checkup.dateEnd],  
 type: Sequilize.QueryTypes.SELECT  
 }).then( results => {  
 auto.checkup.id\_checkup = results[0].id\_checkup;  
 res();  
 });  
 });  
 *await* promiseGetIdCheckUp;  
 }*else* {  
 auto.checkup = {  
 id\_checkup: *null*,  
 dateStart: *null*,  
 dateEnd: *null* }  
 }  
  
 *let* addAutoQuery = 'INSERT INTO auto (`number`, `id\_person`, `id\_checkup`, `id\_model`, `id\_oil`) VALUES (?,?,?, ' +  
 '(select model.id\_model from model inner join brand on brand.id\_brand = model.id\_brand where model.name = ? and brand.name = ?)\n' +  
 ', (select id\_oil from oil where oil.name = ?));';  
  
 *let* promiseAddAuto = *new* Promise( (res, rej) => {  
 connection.query(addAutoQuery,  
 {replacements: [auto.number, auto.person.id\_person, auto.checkup.id\_checkup, auto.model.name, auto.model.brand.name, auto.oil.name]}  
 ).then( ()=>{  
 res();  
 });  
 });  
  
 *await* promiseAddAuto;  
 cb();  
};  
exports.common = *function* (cb) {  
 *let* data = {  
 models:[  
  
 ],  
 oils: []  
 };  
 *let* promises = [];  
 *let* getAllBrandsQuery = 'select model.id\_model, model.name `modelName`,brand.id\_brand, brand.name `brandName`, category.id\_category, category.name `categoryName` ' +  
 'from brand ' +  
 'left outer join model on model.id\_brand = brand.id\_brand ' +  
 'left outer join category on category.id\_category = model.id\_category;';  
 *let* promiseGetModels = *new* Promise( (res, rej) => {  
 connection.query(getAllBrandsQuery, {type: Sequilize.QueryTypes.SELECT}).then( (results) =>{  
 data.models = results;  
 res();  
 });  
 });  
 promises.push(promiseGetModels);  
 *let* getAllOilsQuery = 'select oil.id\_oil, oil.name from oil;';  
 *let* promiseGetAllOils = *new* Promise( (res,rej) => {  
 connection.query(getAllOilsQuery, {type: Sequilize.QueryTypes.SELECT}).then( results => {  
 data.oils = results;  
 res();  
 });  
 });  
 promises.push(promiseGetAllOils);  
  
 Promise.all(promises).then( ()=> {  
 cb(data);  
 });  
};  
  
exports.addBrand = *function* (brand ,cb) {  
 *let* addBrandQuery = 'INSERT INTO brand (`name`) VALUES (?);';  
 *let* promiseAddBrand = *new* Promise( (res, rej) => {  
 connection.query(addBrandQuery, {replacements: [brand.name]}).then(()=>{  
 res();  
 });  
 });  
 promiseAddBrand.then(()=>{  
 cb();  
 });  
};  
exports.addModel = *function* (model, cb) {  
 *let* addModelQuery = 'insert into model (model.id\_brand, model.id\_category, model.name)\n' +  
 'values ((select id\_brand from brand where brand.name = ?),\n' +  
 '(select id\_category from category where category.name = ?), ?)';  
 connection.query(addModelQuery, {replacements: [model.brand.name, model.category.name, model.name], type: Sequilize.QueryTypes.INSERT})  
 .then( () => {  
 cb();  
 });  
};  
  
exports.getById = *function* (id, cb) {  
 console.log('im here and id= ' + id);  
 *let* auto = {  
 model: {}  
 };  
 *let* promises = [];  
 *let* getPersonQuery = 'select person.id\_person, person.name, person.secondName, person.thirdName, person.address, person.phone from person inner join auto on auto.id\_person = person.id\_person ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetPerson = *new* Promise((res,rej)=>{  
 connection.query(getPersonQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 auto.person = results[0];  
 res();  
 })  
 });  
 promises.push(promiseGetPerson);  
  
 *let* getOilQuery = 'select oil.id\_oil, oil.name from oil inner join auto on auto.id\_oil = oil.id\_oil ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetOil = *new* Promise((res,rej)=>{  
 connection.query(getOilQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 auto.oil = results[0];  
 res();  
 })  
 });  
 promises.push(promiseGetOil);  
  
 *let* getCheckUpQuery = 'select checkup.id\_checkup, checkup.dateStart, checkup.dateEnd from checkup inner join auto on auto.id\_checkup = checkup.id\_checkup ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetCheckUp = *new* Promise((res,rej)=>{  
 connection.query(getCheckUpQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 *if*(results[0]){  
 auto.checkup = results[0];  
 }  
 res();  
 })  
 });  
 promises.push(promiseGetCheckUp);  
  
 *let* getModelQuery = 'select model.id\_model, model.name from model inner join auto on auto.id\_model = model.id\_model ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetModel = *new* Promise((res,rej)=>{  
 connection.query(getModelQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 Object.assign(auto.model, results[0]);  
 res();  
 })  
 });  
 promises.push(promiseGetModel);  
  
 *let* getBrandQuery = 'select brand.id\_brand, brand.name from brand ' +  
 'inner join model on model.id\_brand= brand.id\_brand inner join auto on auto.id\_model = model.id\_model ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetBrand = *new* Promise((res,rej)=>{  
 connection.query(getBrandQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 auto.model.brand = results[0];  
 res();  
 })  
 });  
 promises.push(promiseGetBrand);  
  
 *let* getCategoryQuery = 'select category.id\_category, category.name from category ' +  
 'inner join model on model.id\_category= category.id\_category inner join auto on auto.id\_model = model.id\_model ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetCategory = *new* Promise((res,rej)=>{  
 connection.query(getCategoryQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 auto.model.category = results[0];  
 res();  
 })  
 });  
 promises.push(promiseGetCategory);  
  
 *let* getAutoQuery = 'select auto.id\_auto, auto.number from auto ' +  
 'where auto.id\_auto=? ;';  
 *let* promiseGetAuto = *new* Promise((res,rej)=>{  
 connection.query(getAutoQuery, {replacements: [id], type: Sequilize.QueryTypes.SELECT})  
 .then( results=>{  
 Object.assign(auto, results[0]);  
 res();  
 })  
 });  
 promises.push(promiseGetAuto);  
  
 Promise.all(promises).then(()=>{  
 console.log(auto);  
 cb(auto);  
 });  
};  
  
exports.updateAuto = *async function* (newAuto, cb) {  
  
 *let* promises = [];  
 *if*(newAuto.person.name && newAuto.person.secondName){  
 *let* updatePersonQuery = 'UPDATE person ' +  
 'inner join auto on auto.id\_person = person.id\_person ' +  
 'SET `name`=?, `secondName`=?, `thirdName`=?, `address`=?, `phone`=? ' +  
 'WHERE auto.id\_auto=? ;';  
 *let* promiseUpdatePerson = *new* Promise((res,rej)=>{  
 connection.query(updatePersonQuery,  
 {replacements: [  
 newAuto.person.name,  
 newAuto.person.secondName,  
 newAuto.person.thirdName,  
 newAuto.person.address,  
 newAuto.person.phone,  
 newAuto.id\_auto  
 ], type: Sequilize.QueryTypes.UPDATE})  
 .then(()=>{  
 res();  
 });  
 });  
 promises.push(promiseUpdatePerson);  
 }  
  
 *if*(newAuto.checkup && newAuto.checkup.dateStart && newAuto.checkup.dateEnd){  
 *let* addCheckUpQuery = 'INSERT INTO checkup (`dateStart`, `dateEnd`) VALUES (?, ?);';  
 *let* promiseAddCheckUp = *new* Promise((res,rej)=>{  
 connection.query(addCheckUpQuery, {replacements: [newAuto.checkup.dateStart, newAuto.checkup.dateEnd],  
 type: Sequilize.QueryTypes.INSERT}).then(()=>{  
 res();  
 });  
 });  
 *await* promiseAddCheckUp;  
 }*else*{  
 newAuto.checkup = {  
 dateStart: *null*,  
 dateEnd: *null* }  
 };  
 *let* updateModelQuery = 'update auto set auto.id\_model = (select model.id\_model from model \n' +  
 'inner join brand on brand.id\_brand = model.id\_brand where brand.name =? AND model.name =? limit 1), ' +  
 'auto.number =?, ' +  
 'auto.id\_oil = (select oil.id\_oil from oil where oil.name =?), ' +  
 'auto.id\_checkup = (select checkup.id\_checkup from checkup where checkup.dateStart=? AND checkup.dateEnd=? limit 1) ' +  
 'where auto.id\_auto =?';  
 *let* promiseUpdateModel = *new* Promise((res,rej)=>{  
 connection.query(updateModelQuery, {replacements: [  
 newAuto.model.brand.name,  
 newAuto.model.name,  
 newAuto.number,  
 newAuto.oil.name,  
 newAuto.checkup.dateStart,  
 newAuto.checkup.dateEnd,  
 newAuto.id\_auto  
 ], type: Sequilize.QueryTypes.UPDATE }).then(()=>{  
 res();  
 });  
 });  
 promises.push(promiseUpdateModel);  
  
 Promise.all(promises).then(()=>{  
 cb();  
 });  
  
};