МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Автосервис»

Исполнитель

студент (ка) 2 курса группы 6 Завадский Роман Витальевич

(Ф.И.О.)

Руководитель работы ассистент Радиванович Д.А.

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

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2019

**Содержание**

[Введение 3](#_Toc9714706)

[1 Аналитический обзор литературы 4](#_Toc9714707)

[2 Анализ требований к программному средству и разработка функциональных требований 7](#_Toc9714708)

[3 Проектирование программного средства 8](#_Toc9714709)

[4 Создание (реализация) программного средства 14](#_Toc9714710)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 19](#_Toc9714711)

[6 Руководство по установке и использованию 23](#_Toc9714712)

[ЗАКЛЮЧЕНИЕ 28](#_Toc9714713)

[СПИСОК ЛИТЕРАТУРЫ 29](#_Toc9714714)

[Приложение А 30](#_Toc9714715)

[Приложение Б 37](#_Toc9714716)

Введение

Программное средство «Автосервис» – это программа на ПК, позволяющая вести учёт работ и материалов в небольшом автосервисе.  Приложение позволяет управлять персоналом, отслеживать движение финансовых средств, вести детальный учёт работ и запчастей в заказ-нарядах, создавать единую базу клиентов и многое другое. Раньше данный учёт занимал кучу времени и требовал ведения огромного количества журналов, которые заполнялись от руки и далеко не каждый сотрудник, а уж тем более клиент имел доступ к интересующей его в данной момент информации. Требовалось некое лицо-посредник, чтобы получить интересующие данные.

Современные программы позволяют отслеживать и получать актуальную информацию относительно заказов, хода работы. С их помощью можно оформлять, распределять ресурсы и управлять процессом удалённо. Как правило, работа программного средства заключается в обращении к общей базе данных и работе с ней. Но многие программы в данной области не позволяют пользоваться ей кругам лиц. То есть доступ к приложению есть лишь у одного пользователя (оператора), что затрудняет обратную связь как с сотрудниками, так и с клиентами.

Поэтому основной целью моего курсового проекта является разработка программного средства не только для управления СТО, но также и наличие доступа к определённым данным и информации для сотрудников и клиентов с соответствующими правами, полномочиями и уровнями доступа.

При использовании данного программного средства, клиенты смогут получать информацию относительно статуса заказа, сотрудники – отчитываться о проделанной работе, а также получать указания от администратора, который в основном и занят управлением программы. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП. Также будут использоваться технологии Windows Presentation Foundation(WPF) и MS SQL.

Аналитический обзор литературы

Основой для учета, контроля и планирования служат всевозможные картотеки, регистрационные журналы, списки и т.д. Они постепенно накапливаются и обновляются. При большом объеме информации поиск и обобщение необходимых сведений, осуществляемых вручную, представляют собой довольно трудоемкий процесс.

Сейчас же, в век компьютерных технологий и больших прогрессов в этой сфере, на смену картотекам пришли очень удобные, надёжные, быстродействующие компьютерные системы. Для решения задач с использованием компьютерных систем используется специальное программное обеспечение, часто объединенное в крупные информационно-справочные системы.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач, например, при автоматизации бухгалтерского учета на предприятии. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). СУБД взаимодействуют не с локальными, а взаимосвязанными по информации массивами, называемыми базами данных. С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

На сегодняшний день существует достаточное количество программ и онлайн-сервисов, позволяющих управлять работой автосервиса. Одной из таковых является бесплатная версия программы “АвтоСервис Express Edition”.

Рассмотрим в качестве примера интерфейс данной программы на рисунке 1.1.

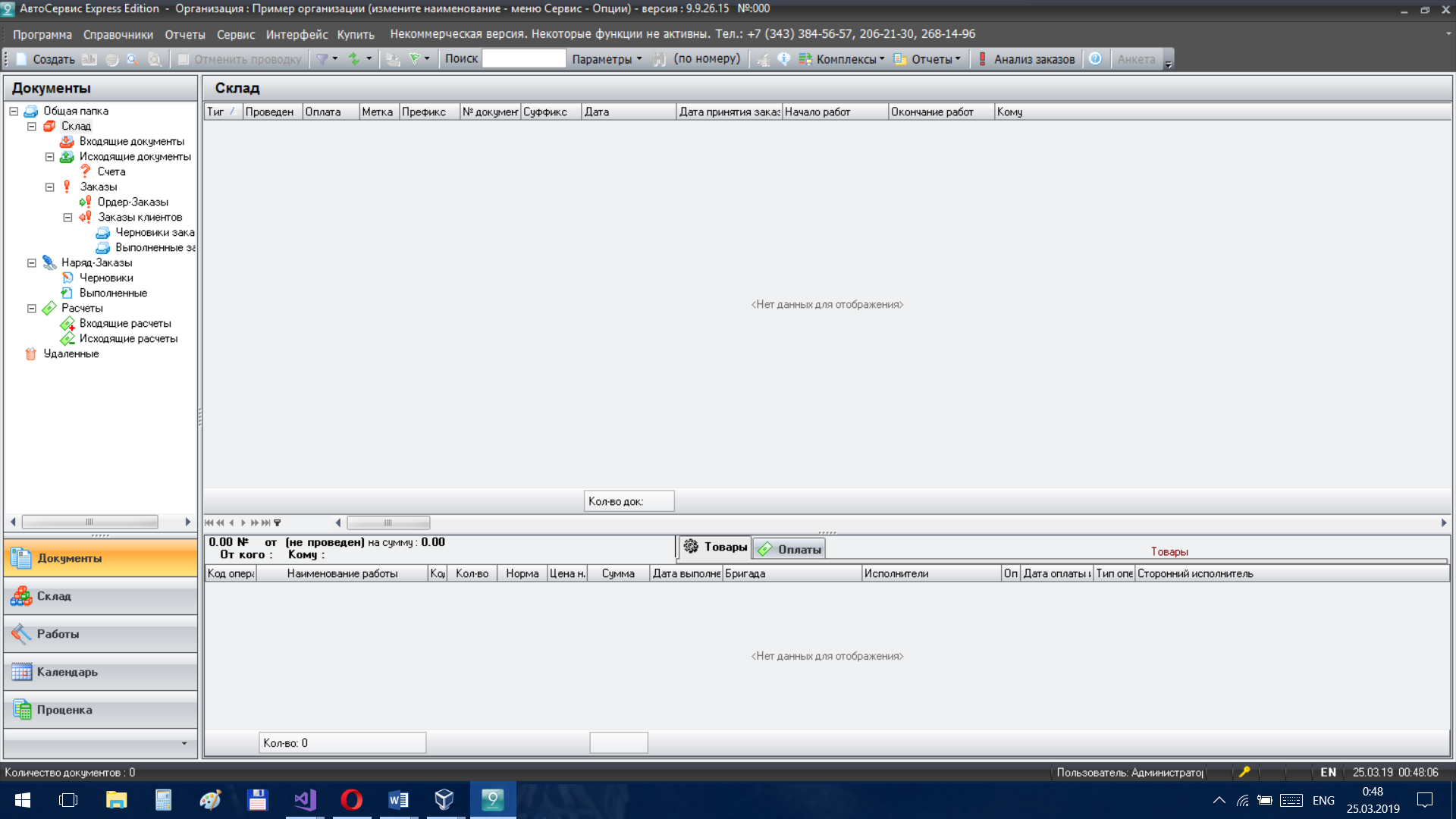


Рисунок 1.1 – интерфейс «АвтоСервис Express Edition»

Вкратце об основных возможностях бесплатной программы для автосервиса «АвтоСервис Express Edition»:

* Ведение учета и автоматизации всего документооборота предприятия любой формы собственности, от ИП до ООО;
* Ведение склада предприятия на всех производственных уровнях;
* Оперативный прогноз расхода материалов/деталей на будущий период;
* Удобная система поддержки заказов, закупок, хранения и продажи запасных частей;
* Четкий, полный и грамотный учёт всех ремонтных операций по автомобилю;
* Наличие в программе складских и товарных отчётов, анализа продаж и кассовых операций, мониторинг финансов и др., позволит Вам быть в курсе всех дел предприятия;
* Наличие и ведение в программе различных справочников (структура склада, комплексные работы, бригады, транспортные средствам, валюты и т.д.);
* Модуль календарь позволит Вам оптимизировать рабочее время в организации и вести планирование рабочего времени, запись клиентов на ремонт, мойку, обслуживание и т.д.

Или же программное средство от разработчика Yukosoft «AutoService», интерфейс которого представлен на рисунке 1.2.

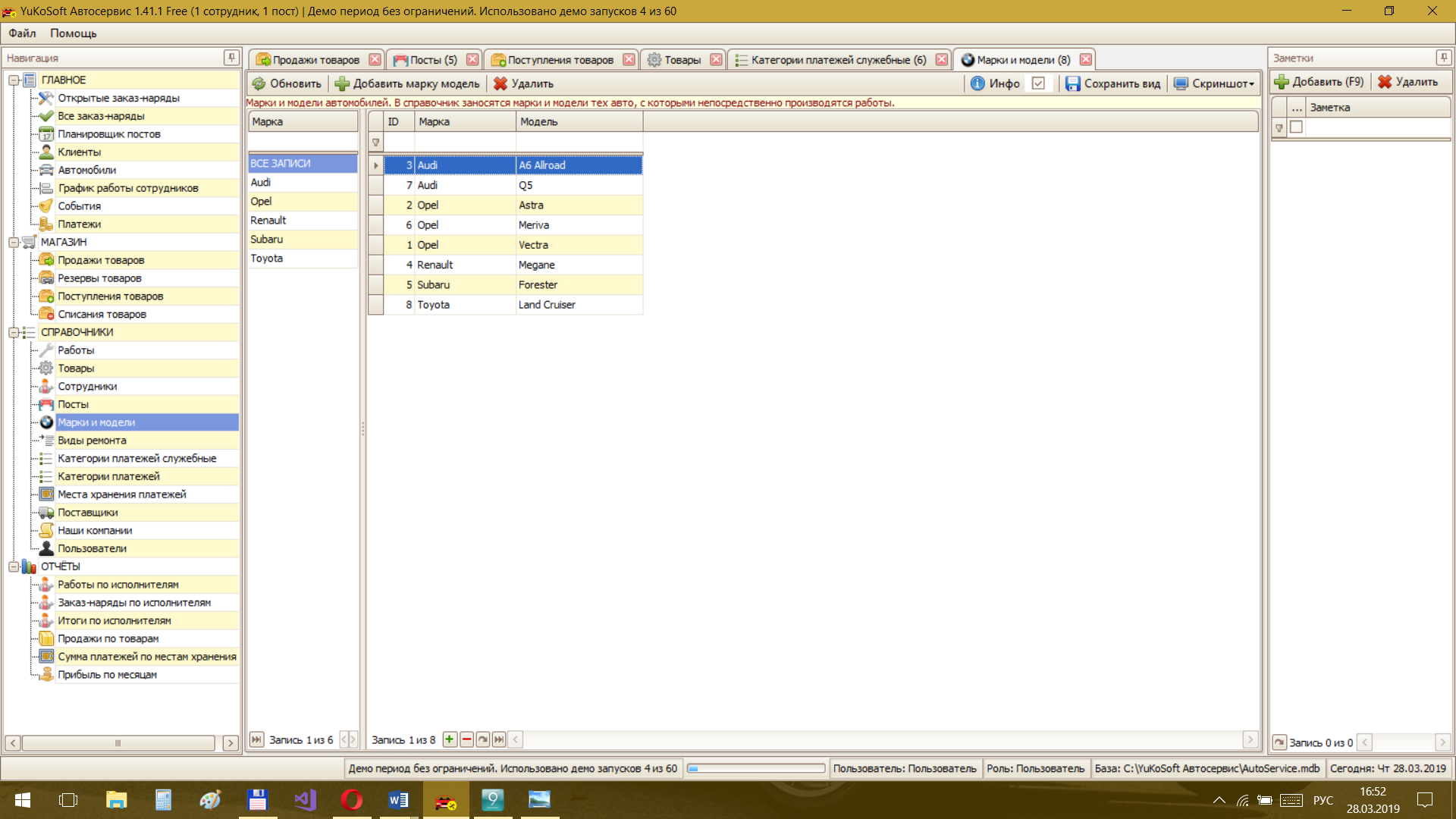


Рисунок 1.2 – интерфейс “Yukosoft AutoService”

И его основные особенности:

* Автоматизация небольших и средних автосервисов и СТО;
* Поддержка работы как с локальной offline базой так и с online базой через интернет;
* Графическое отображение расписания работ по заказ-нарядам;
* Быстрый поиск клиентов и автомобилей;
* Учет клиентов физических и юридических лиц;
* К работам в заказ-наряде можно прикреплять несколько исполнителей с разным процентом участия;
* Удобный справочник работ и товаров для заказ-наряда;
* Стоимость работ может рассчитываться из нормо-часов или фиксированной цены;
* Проставление скидок для работ и запчастей;
* Приём оплат с возможностью разделения на несколько платежей;
* Изменение структуры таблиц. Добавление, переименование и удаление колонок таблиц;
* Редактор дизайна форм. Перетаскивание полей в любое место, создание своего вида формы;
* Возможность хранения изображений, отсканированных копий документов и файлов напрямую в базе;
* Редактор печатных форм. Создание своих печатных форм;
* Экспорт данных в Excel;
* Разграничение прав пользователей программы;
* Получение изображения с WEB камеры.

Проанализировав интерфейсы данных и некоторых других программ, я выявил следующие сходства и особенности:

* Главное поле для отображения документов;
* Поле с документами;
* Различные формы для разных документов;
* Инструменты и функции для работы с документами;
* Схожие интерфейсы.

Исходя из приведённых данных были выдвинуты требования к программному средству, о которых говорится в пункте 2.

Анализ требований к программному средству и разработка функциональных требований

Работа администратора автосервиса благодаря грамотно разработанному программному средству может быть в разы ускорена. Для этого важно определить требования и функции к нашему проекту. Чем сильнее будет автоматизирован процесс работы с документами – тем эффективнее окажется окончательный результат, а это непосредственно положительно сказывается на прибыли компании. В каждом автосервисе важно, как можно детальнее вести учет клиентов и связанных с ними автомобилей, а также хранить полную историю работ по заказ-нарядам клиентов. К каждому клиенту можно привязывать автомобили и смотреть статистику по заказ-нарядам. Программа для СТО позволяет заносить любые необходимые данные и сведения о клиенте: его имя, паспортные данные, контактную информацию, и т.д. Учет в автосервисе умеет прикреплять изображения и файлы. Это может быть, например, фото повреждения авто, скан техпаспорта на авто и т.д. Программа для автосервиса позволяет вести детальный учёт работ и запчастей в заказ-нарядах. К работе прикрепляется один исполнитель.

Управление персоналом в автосервисе так же реализовано в достаточной степени. Например, для сотрудников можно вести отдельный график работы по сменам. Администратором распределяются по сотрудникам заказы клиентов.

Система управления автосервисом позволит взять под контроль все расходы компании: арендная плата, выплаты сотрудникам, текущие расходы. Программа для СТО позволяет добавлять статьи расходов. При таком подходе автоматизированная система для автосервиса предоставляет возможность посредством встроенных отчетов показать, на что в основном тратятся деньги и где можно оптимизировать бюджет автосервиса. Система управления автосервисом позволит повысить уровень сервиса и качества предоставляемых услуг, что в свою очередь обязательно привлечёт новых клиентов, а вместе с ними и прибыль. Схема приведена на рисунке 2.1.



Рисунок 2.1 – Схема взаимодействия функций

Проектирование программного средства

При проектировании нашего программного средства мы использовали технологию Entity Framework. Данная технология представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Entity Framework представляет собой высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

В нашем проекте я использовал подход Code first, который подразумевает, для начала, создание класса модели данных, которые будут храниться в базе данных. После этого Entity Framework по этой модели генерирует базу данных и её таблицы.

На рисунке 3.1 представлена диаграмма классов.



Рисунок 3.1 – Диаграмма классов

Классы с приставкой Add- используются для добавления каких – либо значений в базу данных. Как например класс AddCar позволяет добавить нам автомобиль и необходимую о нём информацию. Класс представлен на рисунке 3.2. Весь класс представлен в приложении А.

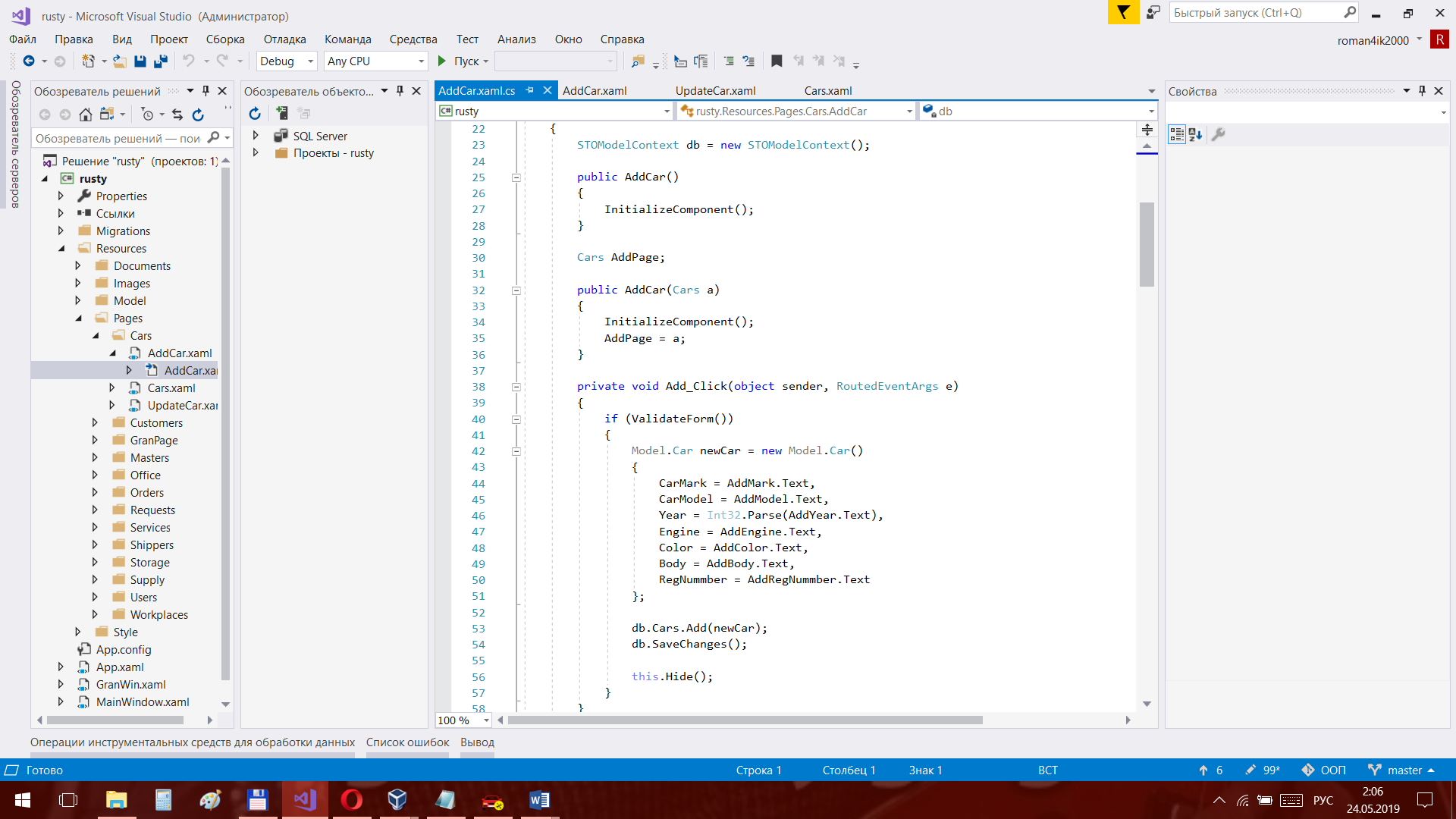


Рисунок 3.2 – Класс AddCar

Классы с приставкой Update- предназначены для изменения соответствующих

данных в таблицах. Как например класс UpdateCar позволяет нам изменять информацию о выбранном автомобиле. Класс представлен на рисунке 3.3. Весь класс представлен в приложении А.

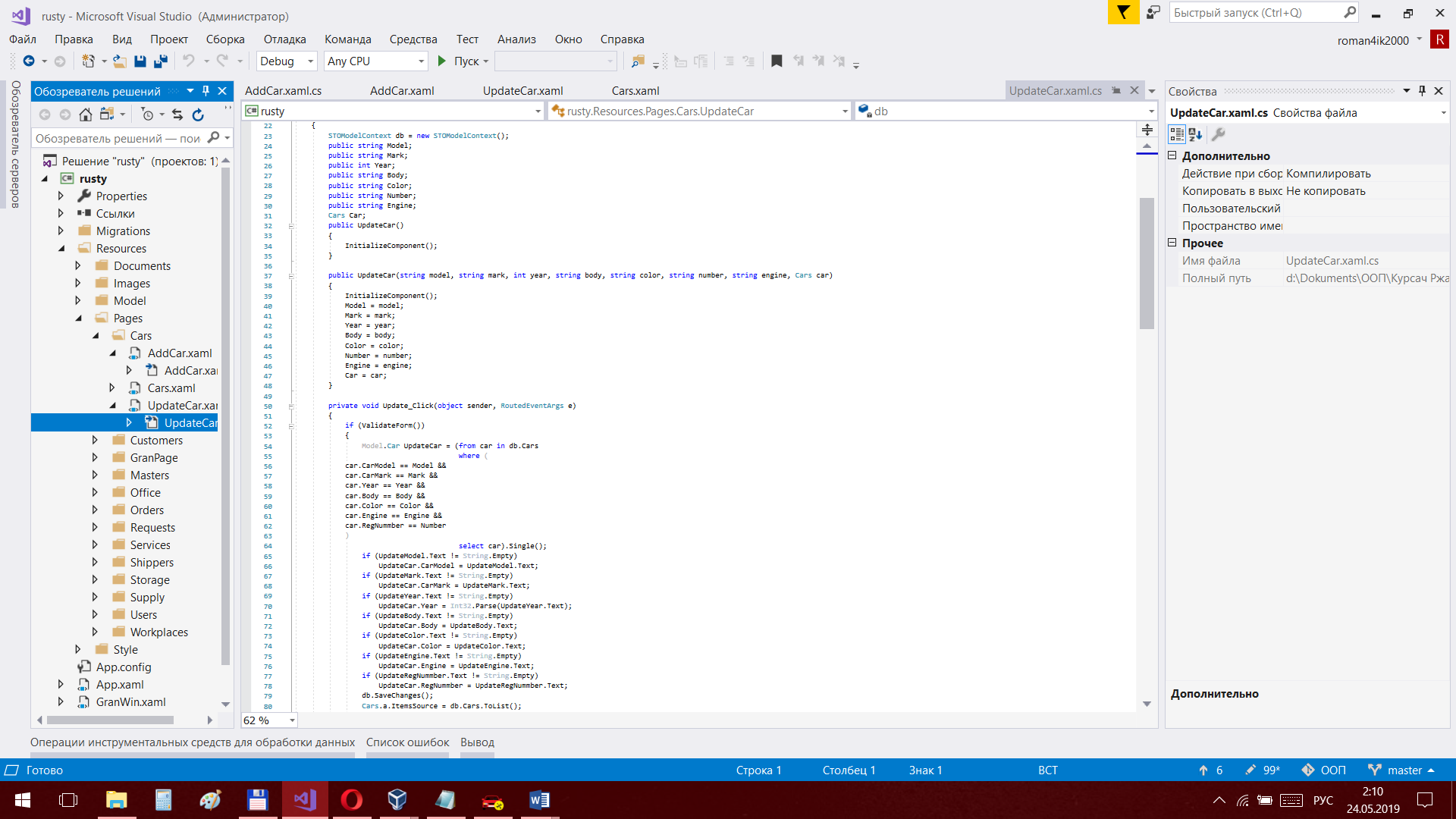


Рисунок 3.3 – Класс UpdateCar

Классы Customers, Users, Storages, Supplies, Requests, Cars, Shippers, Services,

Orders, Masters, Workplace, которые соответствуют названиям таблиц базы данных, являются связующими.

Классы Office- представляю собой личный кабинет для соответствующим им пользователям.

Модель базы данных представлена на рисунке 3.4.

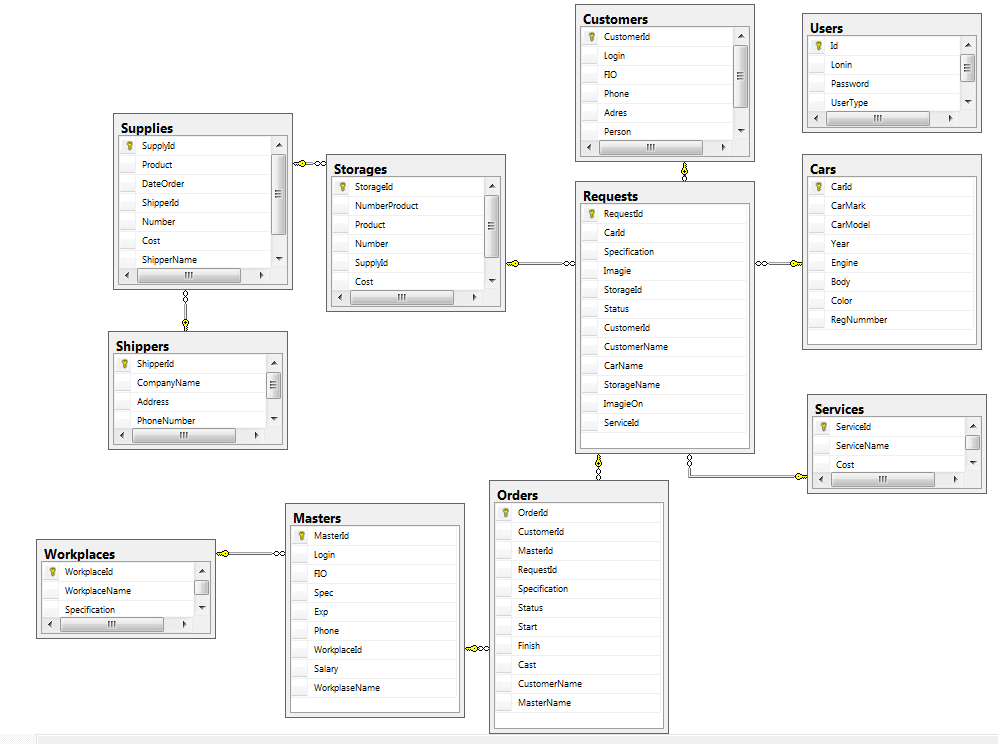


Рисунок 3.4 – Модель базы данных

Всего в базе данных 11 таблиц. В таблице Users хранятся зарегистрированные в приложении пользователи.

Таблица Customers содержит информацию о клиентах. Requests – включает в себя всю информацию о заявке, оставленной клиентом. Cars – данные о различных автомобилях. Storages представляет из себя склад деталей и соответствующим описанием. Supplies – информация о поставках на склад. Shippers – непосредственно сами поставщики. Services – список услуг с соответствующими ценами. Orders – таблица, хранящая в себе данные относительно заказов клиентов и ходе работы над ними. Masters – таблица с сотрудниками автосервиса. Workplaces – рабочие места и их описание.

Таблица Users состоит из столбцов:

* Id;
* Login – логин пользователя;
* Password – пароль пользователя;
* UserType – тип пользователя.

Таблица Customers состоит из столбцов:

* CustomerId;
* Login – логин клиента;
* FIO – ФИО клиента;
* Phone – телефон клиента;
* Address – адрес клиента;
* Person – юридическое либо физическое лицо.

Таблица Requests состоит из столбцов:

* RequestId;
* CarId;
* Specification – описание заявки;
* Image – изображение;
* StorageId;
* Status – статус заявки;
* CustomerId;
* CustomerName – имя клиента;
* CarName – автомобиль;
* StorageName – заменяемая деталь;
* ImageOn – наличие изображения;
* ServiceId – услуга.

Таблица Cars состоит из столбцов:

* CarId;
* CarMark – марка авто;
* CarModel – модель авто;
* Year – год выпуска авто;
* Engine – двигатель авто;
* Body – тип кузова;
* Color – цвет авто;
* RegNummber – регистрационный номер.

Таблица Services состоит из столбцов:

* ServiceId;
* ServiceName – вид услуги;
* Cost – стоимость услуги.

Таблица Storages состоит из столбцов:

* StorageId;
* NumberProduct – код детали;
* Product – название детали;
* Number – количество деталей;
* SupplyId – номер поставки;
* Cost – стоимость детали.

Таблица Supplies состоит из столбцов:

* SupplyId;
* Product – название товара;
* DateOrder – дата заказа;
* ShipperId – поставщик;
* Number – количество;
* Cost – стоимость товара за единицу;
* StorageName.

Таблица Shippers состоит из столбцов:

* ShipperId;
* CompanyName – название поставщика;
* Address – адрес поставщика;
* Phone – номер поставщика.

Таблица Orders состоит из столбцов:

* OrderId;
* CustomerId;
* MasterId;
* RequestId;
* Specification – описание заказа;
* Status – статус заказа;
* Start – дата принятия заказа;
* Finish – дата закрытия заказа;
* Cost – стоимость работы;
* CustomerName – клиент;
* MasterName – мастер.

Таблица Masters состоит из столбцов:

* MasterId;
* Login – логин мастера;
* FIO – ФИО мастера;
* Spec – специальность мастера;
* Exp – стаж мастера;
* Phone – номер мастера;
* WorkplaceId;
* Salary – зарплата мастера;
* WorkplaceName – рабочее место мастера.

Таблица Workplaces состоит из столбцов:

* WorkplaceId;
* WorkplaceName – название рабочего место;
* Specification – описание рабочего места.

Создание (реализация) программного средства

Файлы программы распределены по папкам. Разделение проекта на логические модули представлено на рисунке 4.1.

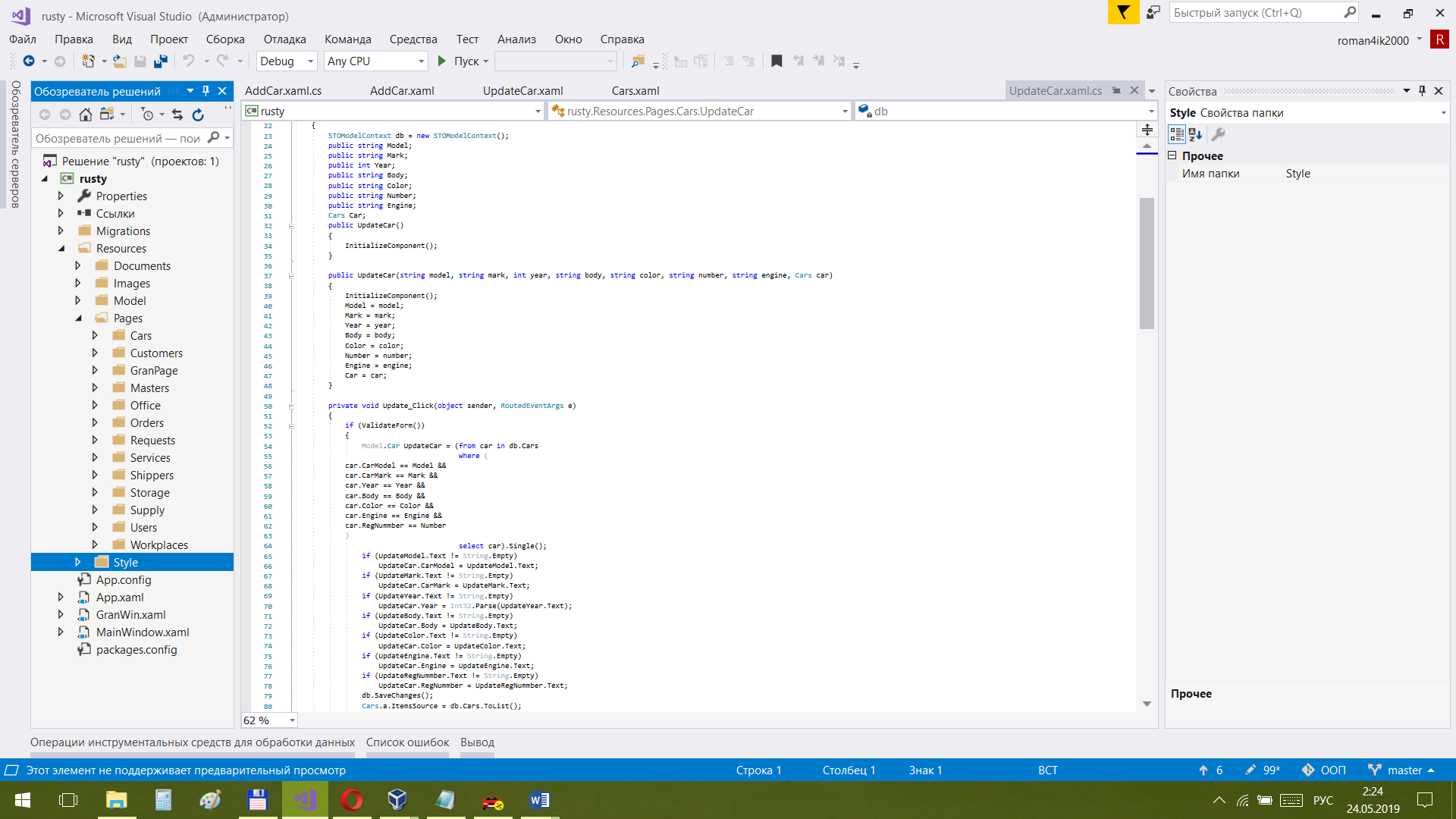


Рисунок 4.1 – логические модули проекта

В папке Migrations находятся миграции. Миграции отвечают за изменение в базе данных при изменениях моделей и контекста данных. Мы можем использовать контекст данных для работы с базой данных (добавление, удаление, изменение). Если мы попытаемся изменить модель и при этом не выполним миграцию, то мы получим ошибку. Сами миграции хранят в себе соответствующие изменения моделей. Пример миграции приведён на рисунке 4.2.

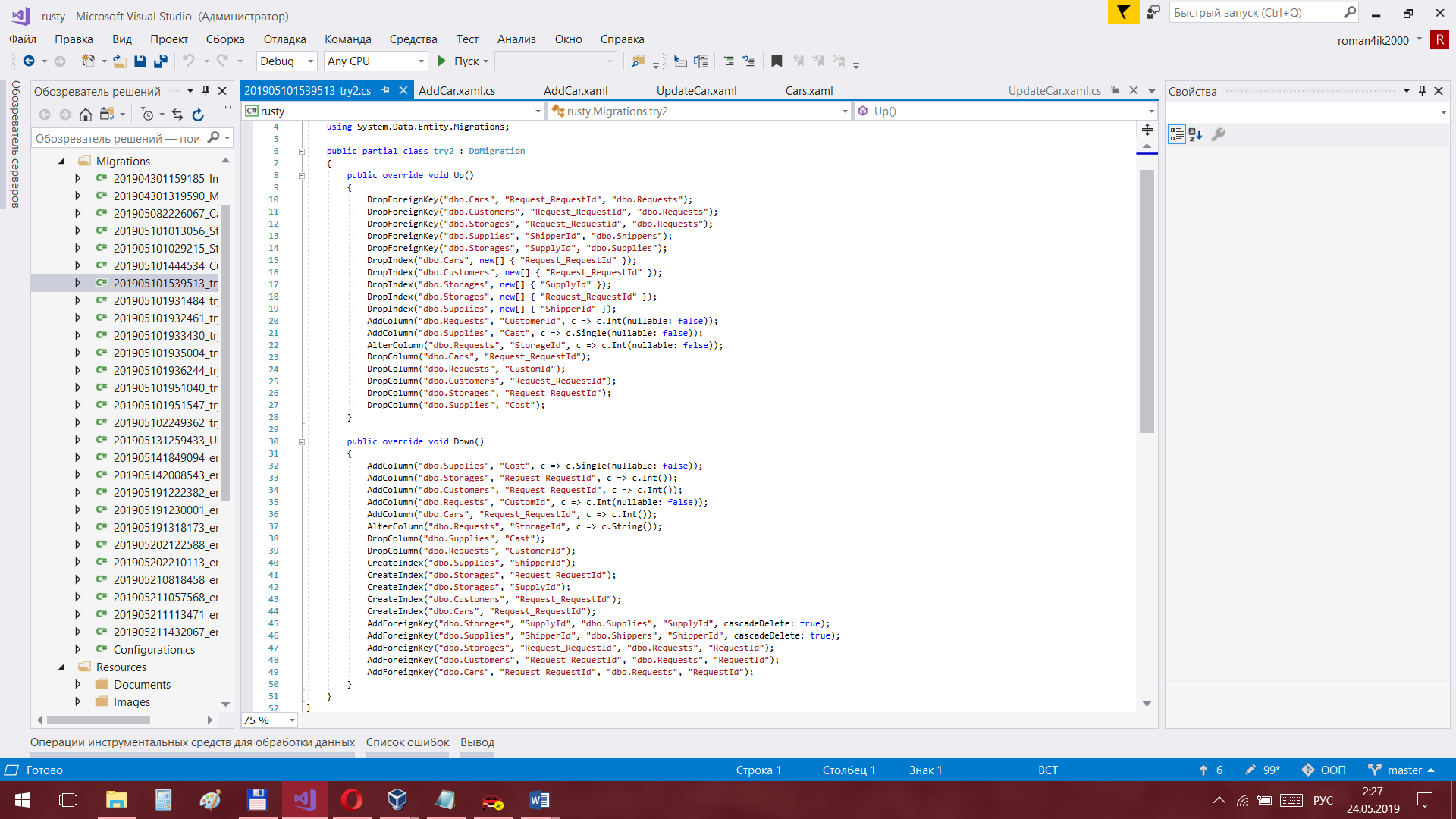


Рисунок 4.2 – пример миграции

В папке Models хранятся модели нашего проекта. Сама модель представляет из себя некоторый класс с набором автосвойств. Каждое свойство соспоставляется с отдельным столбцом из таблицы в нашей базе данных. Здесь же мы накладываем ограничения на наши строки а также прописываем связи с другими таблицами. Пример модели представлен на рисунке 4.3.

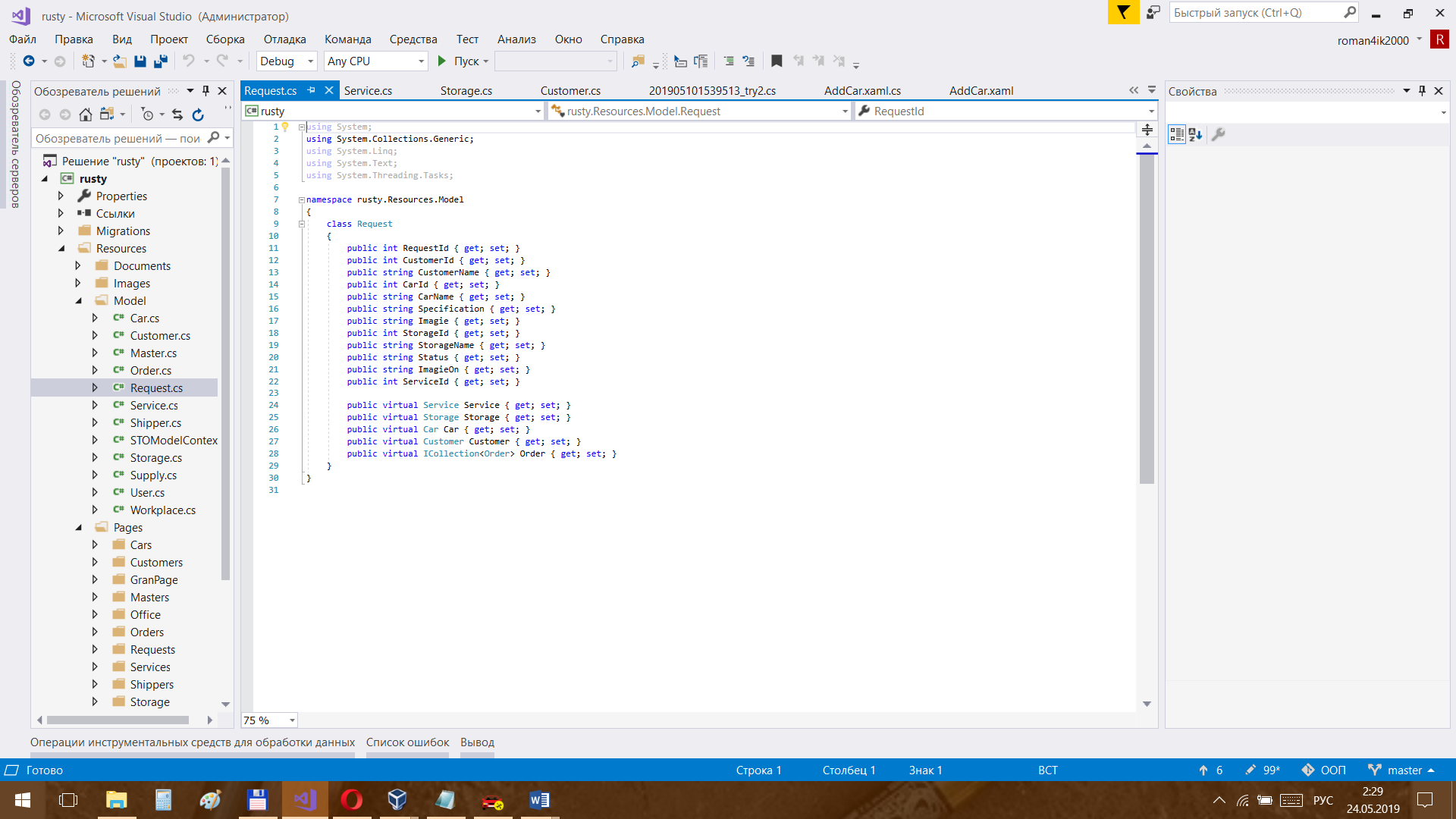


Рисунок 4.3 – пример модели

В папке Images содержатся все изображения, которые используются в курсовом проекте в качестве иконок, либо фона.

В папке Pages находятся папки, соответствующие названиям таблицам нашей бд, а именно Customers, Users, Storages, Supplies, Requests, Cars, Shippers, Services, Orders, Masters, Workplace. В каждой из этих папок находится одна страница, соответствующая названию папки, а также 2 окна, с приставками Add- и Update-, которые позволяют нам соответственно добавлять и изменять данные. Логическая модель представлена на рисунке 4.4, а сигнатуры методов представлены в приложении Б.

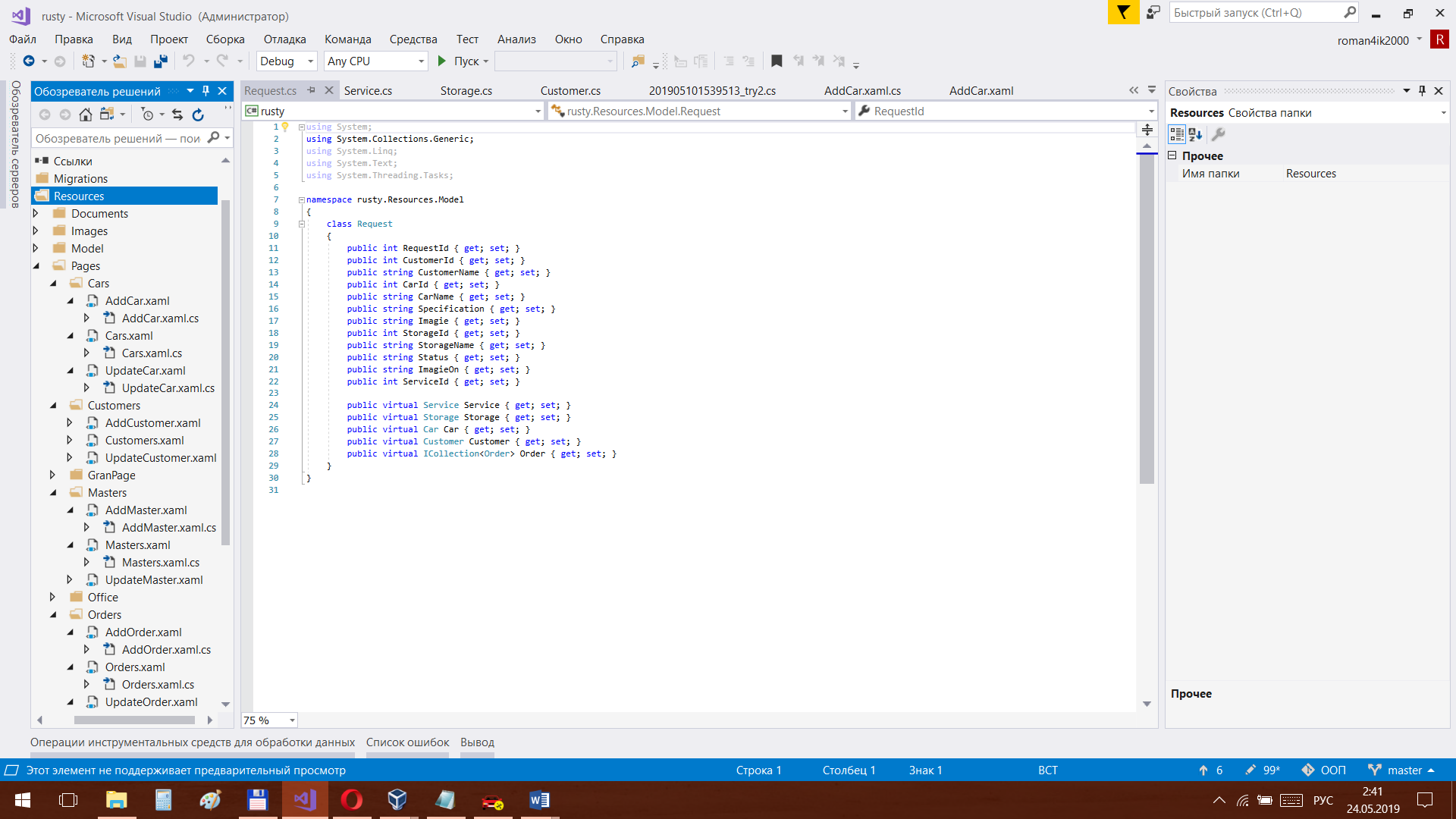


Рисунок 4.4 – Основные модули проекта

Сигнатуры методов моделей и их логические модули представлены на рисунках 4.5 и 4.6 соответственно.

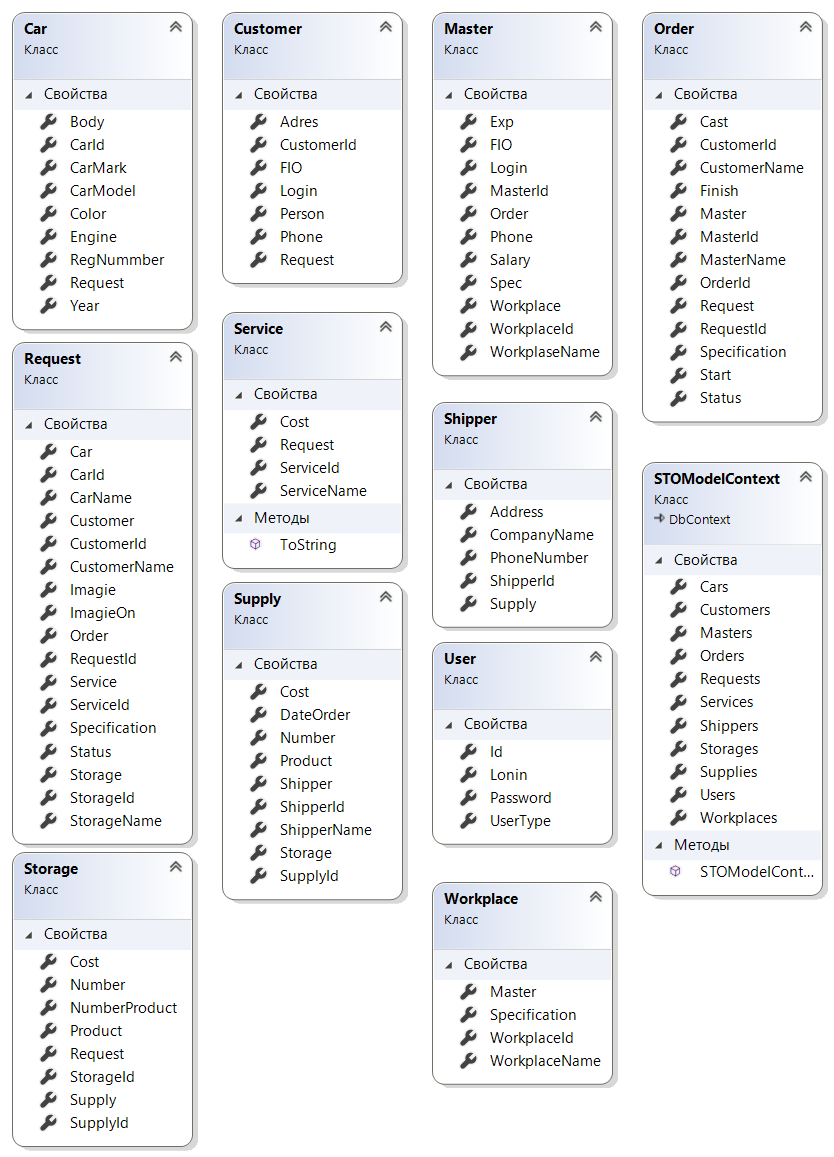


Рисунок 4.5 – Сигнатуры методов моделей

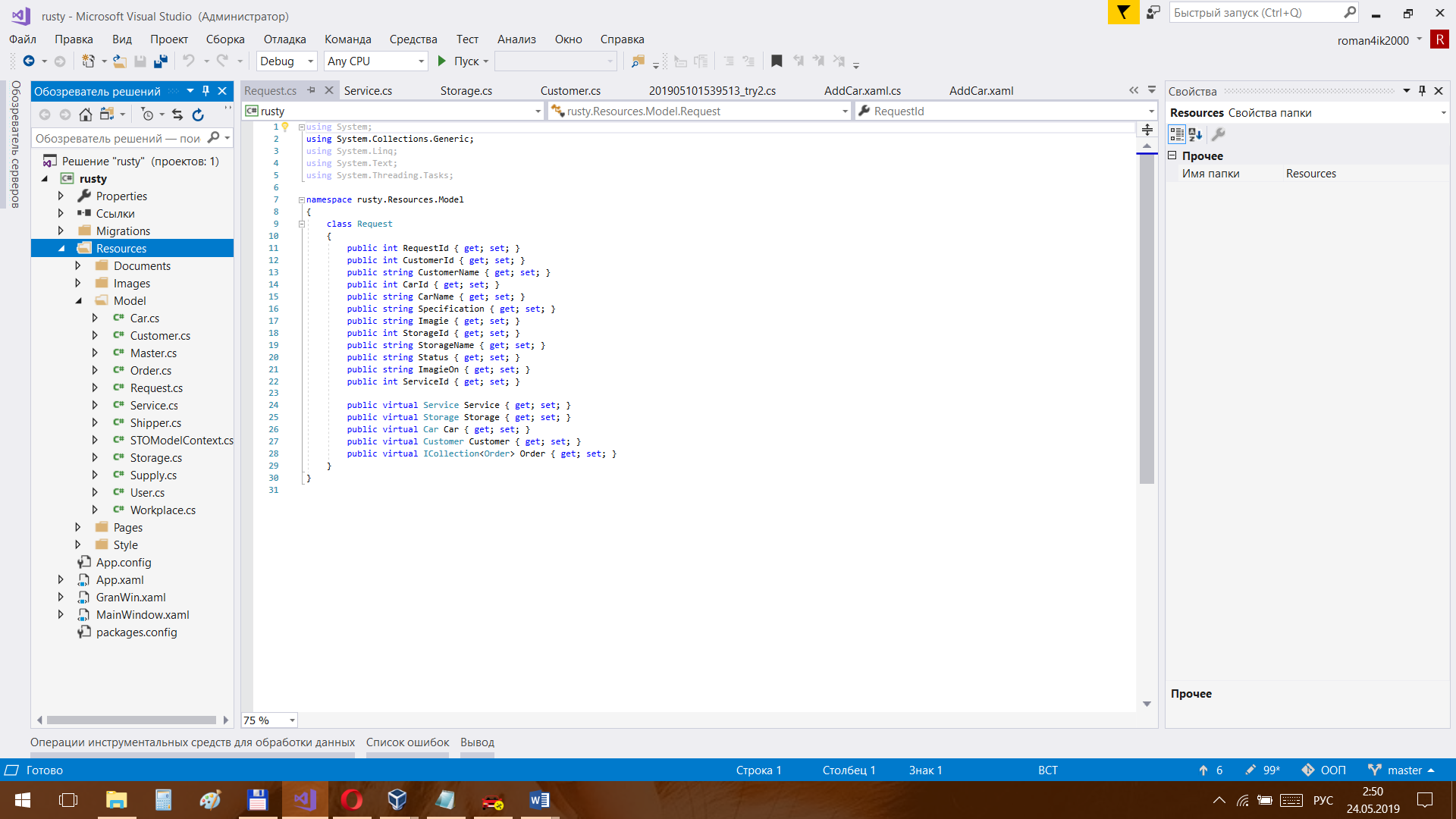


Рисунок 4.6 – логические модули моделей

В папке Office находятся страницы личного кабинета для разных пользователей. Страница GranPage является главной страницей в приложении.

Папка Style содержит подключаемые стили для некоторых элементов управления и окон, что представлено на рисунке 4.7.

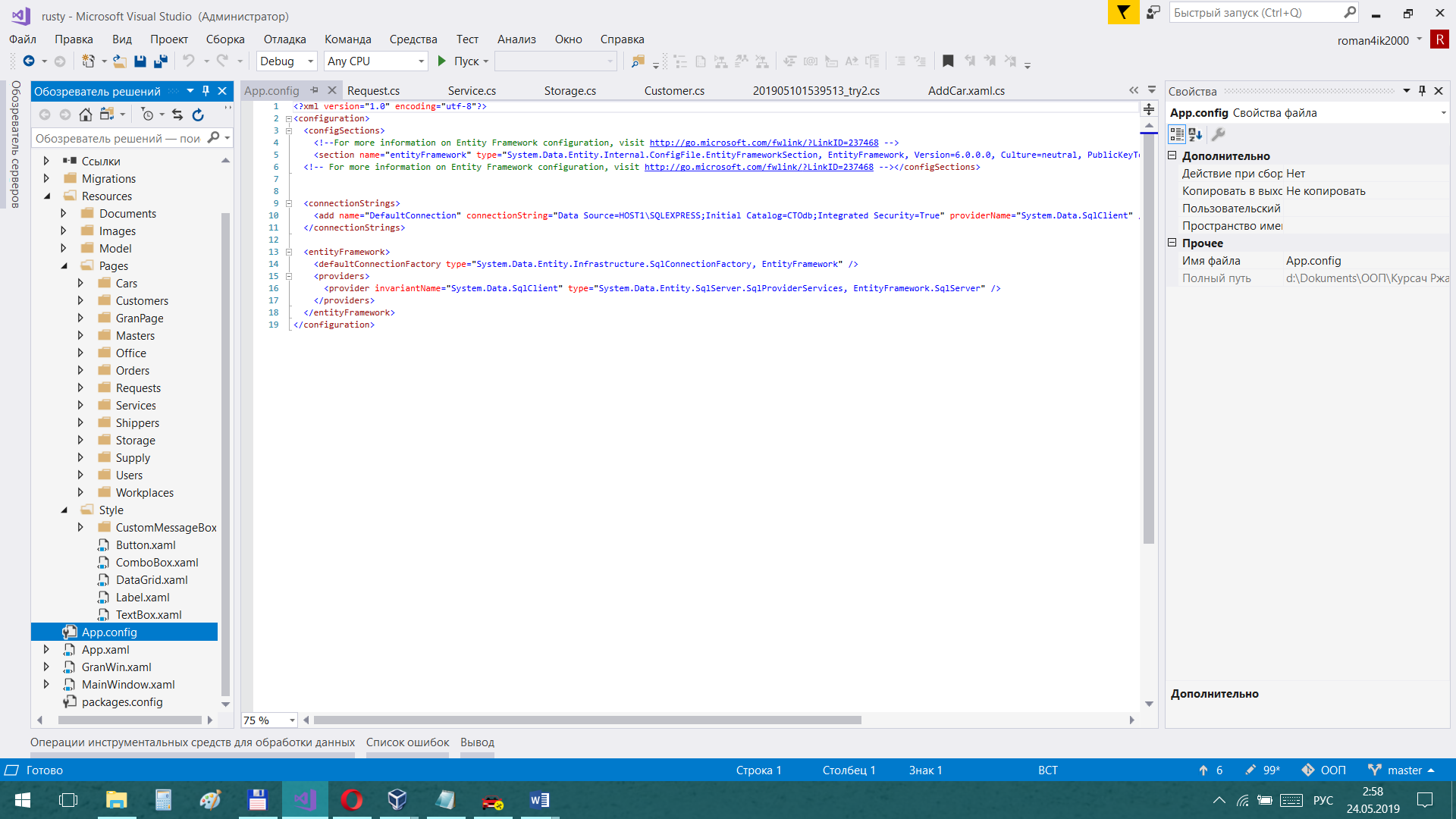


Рисунок 4.7 – Стили

Окно MainWindow является окном регистрации и авторизации, а окно GranWin – основное окно, в котором и осуществляется работа с приложением.

Тестирование, проверка работоспособности и анализ полученных результатов

В ходе тестирования моего приложения были проведены тесты на:

* валидацию авторизации и регистрации;
* добавление, удаление, изменение данных;
* доступ разных типов пользователей в различные уголки программы;
* тесты на отсутствие противолгасий различных типов;
* тесты на взаимодействие классов.

Тестируем для начала окно авторизации и регистрации.

Введём для начала неверный пароль. Обработка продемонстрирована на рисунке 5.1.



Рисунок 5.1 – Тестирование авторизации

Протестируем регистрацию. Попробуем зарегистрировать уже имеющегося в приложении пользователя. Обработка исключения представлена на рисунке 5.2.

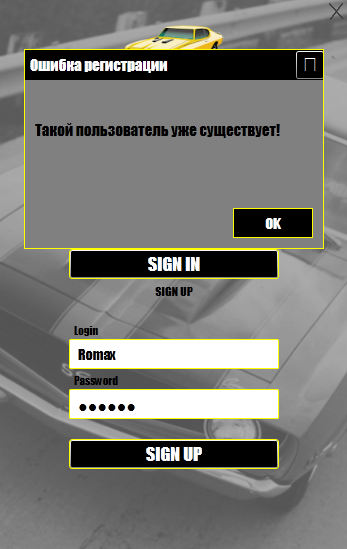


Рисунок 5.2 – Тестирование регистрации

Попробуем ввести при регистрации пустые значения логина и пароля. Обработка представлена на рисунке 5.3.

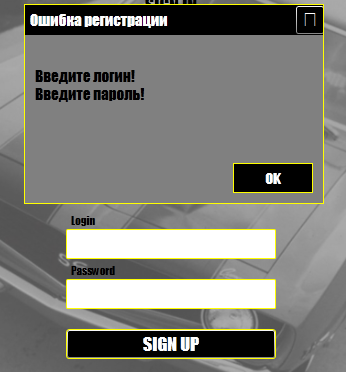


Рисунок 5.3 – Тестирование регистрации на пустое значение

Перейдём к тестированию работоспособности приложения. Для начала протестируем ограничение функционала приложения при разных уровнях доступа. Демонстрация представлена на рисунке 5.4.

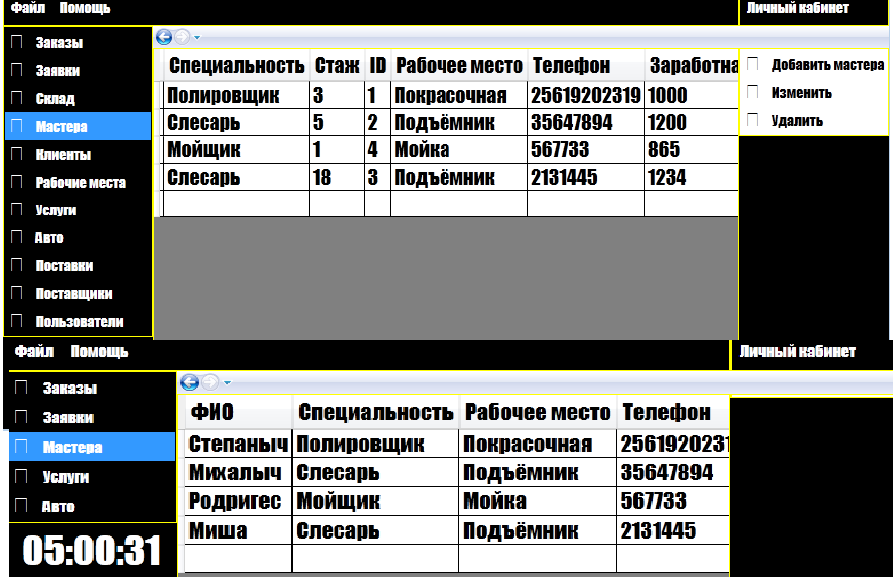


Рисунок 5.4 – Доступные функционалы разных пользователей в одной из страниц

На данном рисунке сверху представлен функционал для пользователя с доступом администратора, а внизу – для пользователя клиента. Для администратора доступны функции как добавление мастера, его изменение, а также удаление. К тому же администратор видит больше столбцов в таблице, как например зарплата рабочего.

Следующая проверка будет на попытку удаления пустой строки. Для этого выделим строку и нажмём «удалить». Демонстрация исключения предоставлена на рисунке 5.5.

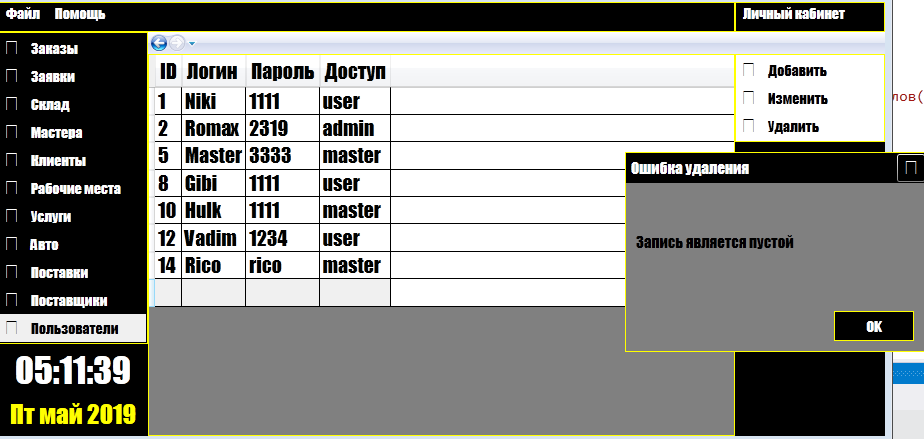


Рисунок 5.5 – Обработка попытки удаления пустой строки

Ну и наконец попробуем ввести в нашу таблицу некорректные значения и данные. Для этого нажмём кнопку «добавить». Появится окно для заполнения необходимых полей. Обработка этого исключения показана на рисунке 5.6.

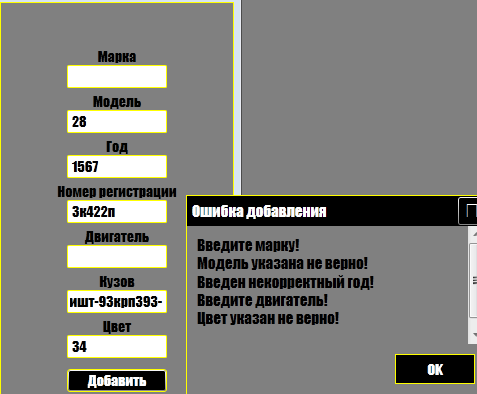


Рисунок 5.6 – Обработка некорректных данных при добавлении в таблицу

Аналогично обрабатываются исключения и при изменении данных, однако если оставить поле или поля незаполненными, это будет означать, что конкретно эти поля мы не хотим вносить изменения и их значения останутся прежними. Так же в некоторых таблицах имеются поля не обязательные для заполнения, данные в которых могут отсутствовать.

Руководство по установке и использованию

Для начала программу необходимо установить с помощью инсталлятора на персональный компьютер.

Запускаем файл Setup.exe и ждём загрузки, как на рисунке 6.1

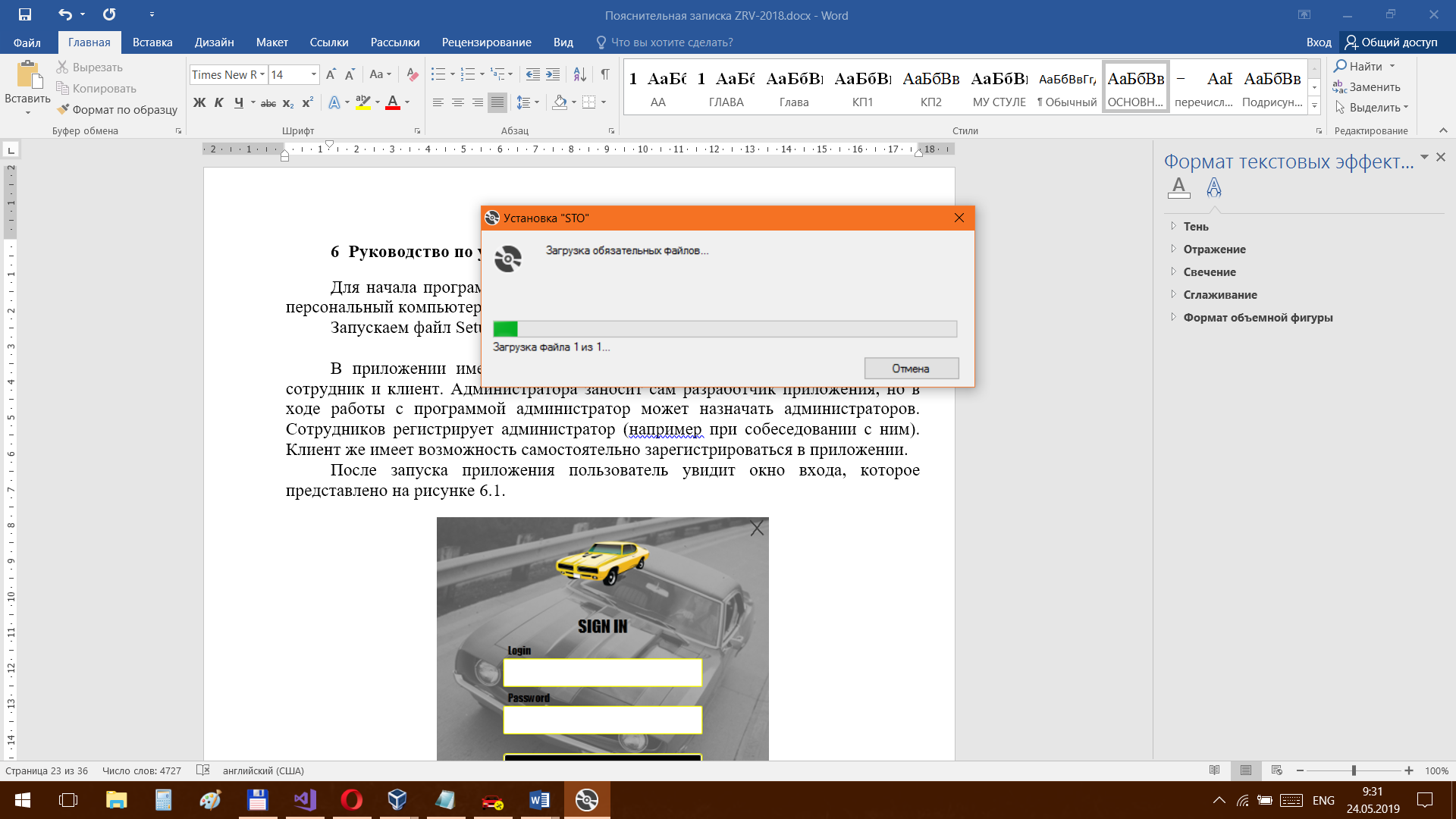


Рисунок 6.1 – Загрузка файлов

Далее следуем указаниям установщика, как на рисунках 6.2 и 6.3.

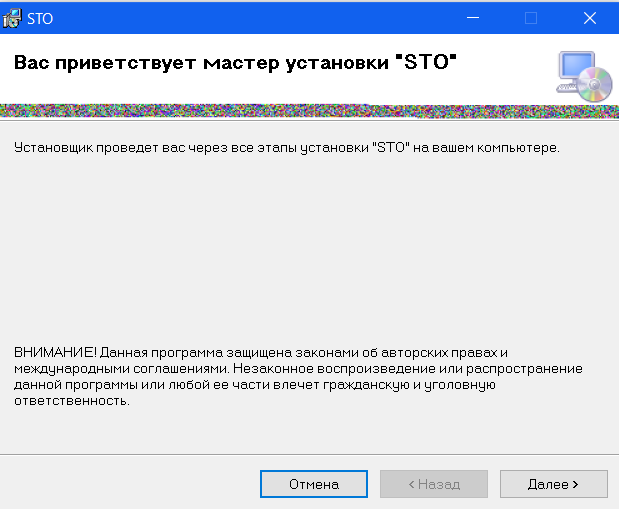


Рисунок 6.2 – Процесс установки



Рисунок 6.3 – Выбор пути установки

В приложении имеется 3 вида пользователей, а именно: администратор, сотрудник и клиент. Администратора заносит сам разработчик приложения, но в ходе работы с программой администратор может назначать администраторов. Сотрудников регистрирует администратор (например, при собеседовании с ним). Клиент же имеет возможность самостоятельно зарегистрироваться в приложении.

После запуска приложения пользователь увидит окно входа, которое представлено на рисунке 6.4.

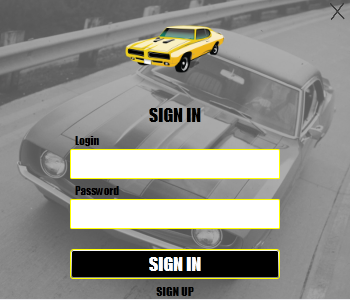


Рисунок 6.4 – Окно входа

Если пользователь не имеет аккаунта в приложении, то он должен пройти регистрацию, нажав на «SIGN UP», указав логин и пароль, а также некоторые личные данные о себе. Окно представлено на рисунке 6.5.

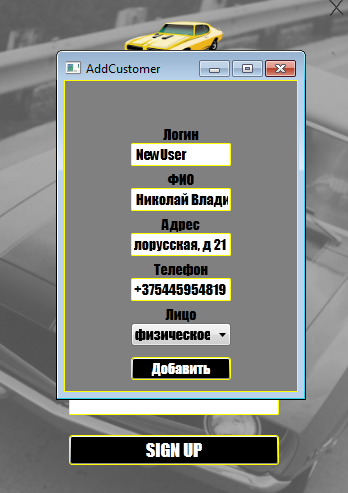


Рисунок 6.5 – Процесс регистрации

Зарегистрировавшись, пользователю необходимо ввести свои логин и пароль.

После авторизации он попадает на главную страничку приложения, которая представлена на рисунке 6.6.

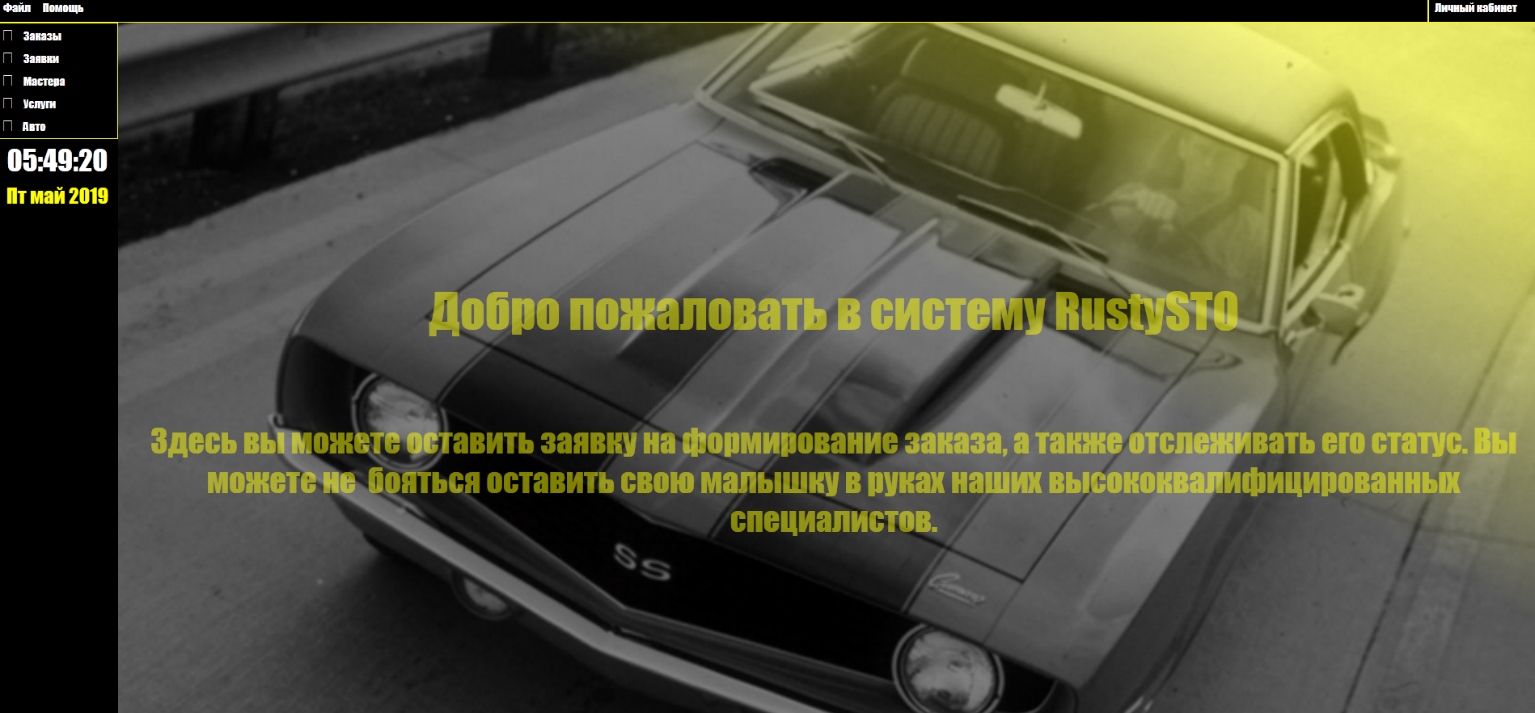


Рисунок 6.6 – Главная страница приложения

Слева от пользователя находится колонка с доступными ему таблицами. В зависимости от того, какой пользователь в данный момент пользуется приложением, он имеет разные уровни доступа к функционалу приложения. Виды колонки для разных видов пользователей указаны на рисунке 6.7.



Рисунок 6.7 – Таблицы для разных типов пользователей

В верхнем правом углу находится вкладка личный кабинет. При нажатии на «Войти в кабинет» перед нами откроется личный кабинет с информацией о пользователе, который представлен на рисунке 6.8.

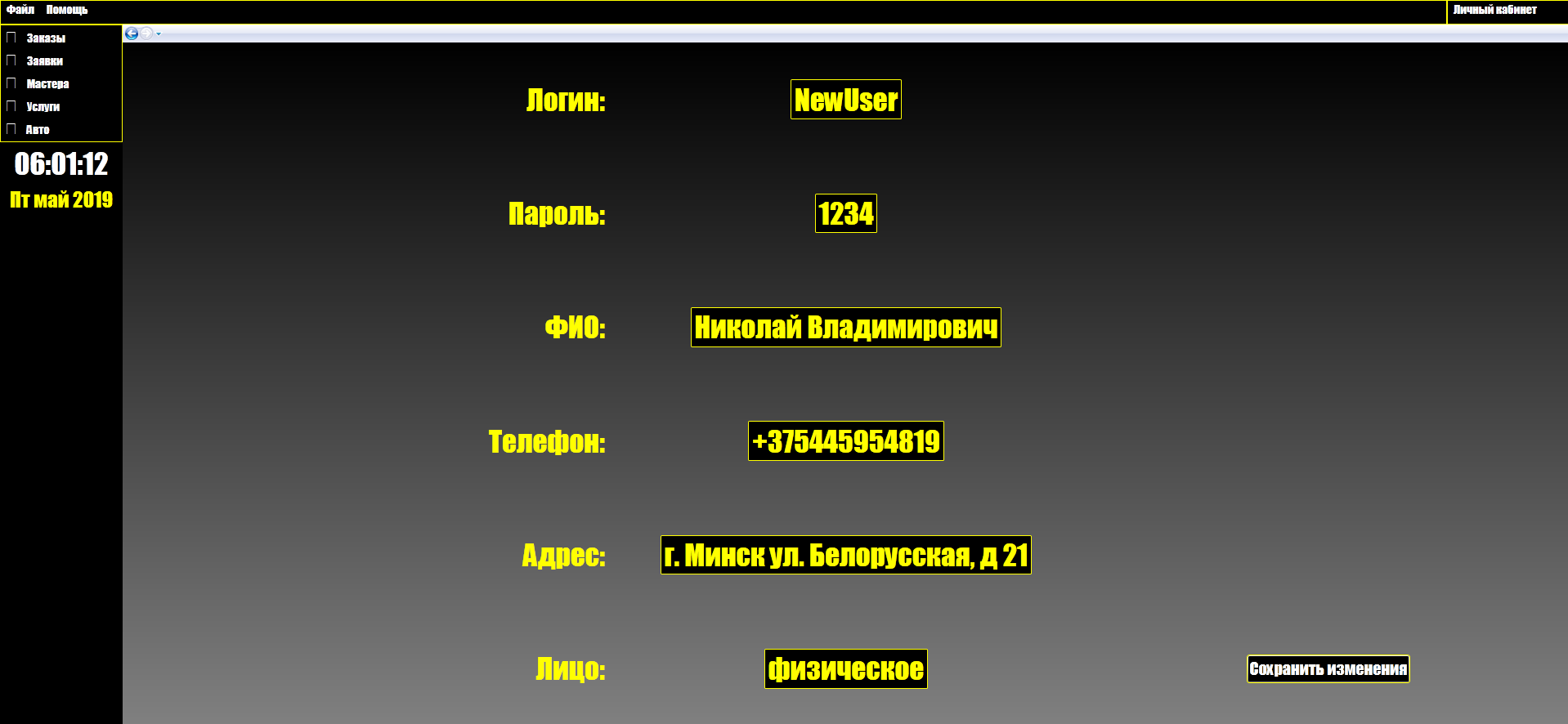


Рисунок 6.8 – Личный кабинет

Здесь мы можем изменить наши личные данные.

При нажатии кнопки «Выйти из аккаунта», пользователь выйдет из своего аккаунта и окажется у окошка входа.

В верхнем левом углу находятся две вкладки: файл и помощь.

При нажатии кнопки «На главную», мы окажемся на главной странице приложения. При нажатии «Выход», приложение завершит работу.

Если же нас интересует какая - либо информация о приложении, то мы обратимся к ней при нажатии «О программе», а для открытия руководства, нажмём на «Руководство», в результате чего откроется текстовый документ с руководством.

И немного о таблицах. Администратору доступны все таблицы и весь функционал. Мастер имеет отношения непосредственно к рабочим особенностям пользования таблицами. Пользователю же в свою очередь и оставляет свои заявки.

Таблица «заказы» позволяет формировать заказы на основе оставленных заявок. Автоматически устанавливает время принятия заказа и высчитывает его стоимость на основе заменяемых деталей и услуг. Видна всем пользователям. Мастер имеет возможность закрывать заказ по его выполнению, при этом дата окончания фиксируется автоматически.

Таблица «заявки». Именно здесь пользователь и оставляет заявку на необходимый ему сервис, выбрав услугу, заменяемую деталь. Так же для удобства, если пользователь оставляет заявку впервые, то он может прямо отсюда заполнить информацию о его автомобиле, а затем указать свой авто в заявке. Так же имеется возможность прикрепления изображения. Саму заявку рассматривает администратор и он принимает решение о принятии или отказе. Мастер же при выполнении заказа может обратиться к таблице и узнать информацию о предстоящей работе.

Таблица «склад» является хранилищем с деталями, которые используются при ремонте авто. К таблице имеет доступ администратор и мастер. Администратор формирует содержимое склада относительно поставок на склад, указывая при этом необходимые данные, а также устанавливая наценку на детали. Мастер же, при выполнении заказа, обращается на склад и уменьшает количество деталей щелчком на единицу.

Таблица «мастера» содержит в себе информацию о сотрудниках автосервиса. Таблица доступна всем пользователям, однако не все столбцы. Именно к этой таблице обращается администратор, когда формирует заказ, указывая при этом мастера. За каждым мастером закреплено своё рабочее место. Пользователь же, исходя из заказа, может найти мастера, который выполняет ремонт его автомобиля и, в случае необходимости, связаться лично с ним. Администратор также заполняет эту таблицу о своём работнике на собеседовании с ним и принятием на работу.

Таблица «клиенты» содержит информацию о клиентах автосервиса. Таблица доступна мастерам для того, чтобы осуществлять обратную связь с клиентом в случае возникновения вопросов. Таблица формируется при регистрации новых пользователей.

Таблица «рабочие места» хранит информацию о рабочих местах на СТО. У каждого мастера своё рабочее место.

Таблица «услуги» доступна всем пользователям. Содержит перечень услуг и цены за интересующие вас услуги.

Таблица «авто» хранит информацию о автомобилях. Здесь можно быстро найти интересующие вас данные об авто. Пользователь может заполнять эту таблицу.

Таблица «поставки» является связующим звеном между таблицей «склад» и «поставщики». Доступна только администратору. После обсуждения заказа с поставщиком данные о нем заносятся сюда.

Таблица «поставщики» доступна только администратору. Здесь имеется информация о поставщиках деталей и прочего на вашу СТО.

Таблица «пользователи» доступна только администратору. Здесь хранятся персональные данные со всех зарегистрированных в приложении аккаунтах. Через неё администратор регистрирует своих сотрудников и может назначать новых администраторов.

ЗАКЛЮЧЕНИЕ

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Автосервис», которое бы позволить автоматизировать процесс управления автосервисом и сделать его работу эффективнее. Основной целью курсового проекта стало формирование навыков разработки приложений с интеграцией базы данных. При разработке выполнены следующие пункты:

* Реализация работы с базой данных;
* Регистрация и авторизация пользователей;
* Разделение функционала приложения в зависимости от использующего его в данный момент пользователя;
* Отслеживание статуса заказа;
* Возможность прикрепить к заявке изображение;
* Автоматический подсчет цены заказа.

В курсовом проекте были реализованы следующие задачи:

* Конфигурирование программного обеспечения;
* Создание базы данных;
* Создание интерфейса;
* Реализации функций работы приложения;
* Тестирование программного продукта.

Основной целью моего курсового проекта являлось разработка программного средства не только для управления СТО, но также и наличие доступа к определённым данным и информации для сотрудников и клиентов с соответствующими правами, полномочиями и уровнями доступа.

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

СПИСОК ЛИТЕРАТУРЫ

1. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.

2. Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с.

3. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> Дата доступа: 29.04.2019

4. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com> Дата доступа: 07.05.2019

5. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru> Дата доступа: 13.05.2019

6. STO-Soft.ru [Электронный ресурс] / Режим доступа: https:// <http://sto-soft.ru> Дата доступа: 17.04.2019

7. Quaru Site [Электронный ресурс] / Режим доступа: https:// qaru.site Дата доступа: 20.05.2019

Приложение А

Листинг класса STOModelContext

|  |
| --- |
| class STOModelContext : DbContext  {  public STOModelContext() : base("DefaultConnection")  {  }  public DbSet<Shipper> Shippers { get; set; }  public DbSet<Supply> Supplies { get; set; }  public DbSet<Car> Cars { get; set; }  public DbSet<Storage> Storages { get; set; }  public DbSet<Request> Requests { get; set; }  public DbSet<Customer> Customers { get; set; }  public DbSet<Order> Orders { get; set; }  public DbSet<Master> Masters { get; set; }  public DbSet<Workplace> Workplaces { get; set; }  public DbSet<User> Users { get; set; }  public DbSet<Service> Services { get; set; }  } |

Листинг класса Cars

|  |
| --- |
| public partial class Cars : Page  {  STOModelContext db;  public static DataGrid a;  public MainWindow mainWindow;  public Cars()  {  InitializeComponent();  db = new STOModelContext();  carGrid.ItemsSource = db.Cars.ToList();  a = carGrid;  if (MainWindow.current\_user == "user")  {  Update.Visibility = Visibility.Collapsed;  Delete.Visibility = Visibility.Collapsed;  CarId.Visibility = Visibility.Collapsed;  }  if (MainWindow.current\_user == "master")  {  Update.Visibility = Visibility.Collapsed;  Delete.Visibility = Visibility.Collapsed;    }  }    private void Add\_PreviewMouseDown(object sender, MouseButtonEventArgs e)  {  AddCar addCar = new AddCar(this);  addCar.Show();  }  private void Update\_PreviewMouseDown(object sender, MouseButtonEventArgs e)  {  try {  string Model;  string Mark;  string Body;  int Year;  string Color;  string Engine;  string Number;  if (carGrid.SelectedItems.Count > 0)  {  for (int i = 0; i < carGrid.SelectedItems.Count; i++)  {  Model.Car car = carGrid.SelectedItems[i] as Model.Car;  Model = car.CarModel;  Mark = car.CarMark;  Year = car.Year;  Color = car.Color;  Body = car.Body;  Number = car.RegNummber;  Engine = car.Engine;  UpdateCar update = new UpdateCar(Model,Mark,Year,Body,Color,Number,Engine,this);  update.Show();  }  }  }  catch (NullReferenceException)  {  rusty.Resources.Style.CustomMessageBox.CustomMessageBox.Show("Ошибка изменения", "Запись является пустой", MessageBoxButton.OK);  }  }  private void Delete\_PreviewMouseDown(object sender, MouseButtonEventArgs e)  {  try  {  int? Id = (carGrid.SelectedItem as Model.Car).CarId;  var deleteCar = db.Cars.Where(db => db.CarId == Id).FirstOrDefault();  db.Cars.Remove(deleteCar);  db.SaveChanges();  carGrid.ItemsSource = db.Cars.ToList();  }  catch (NullReferenceException)  {  rusty.Resources.Style.CustomMessageBox.CustomMessageBox.Show("Ошибка удаления", "Запись является пустой", MessageBoxButton.OK);  }  }  }  } |

Листинг класса AddCar

|  |
| --- |
| public partial class AddCar : Window  {  STOModelContext db = new STOModelContext();  public AddCar()  {  InitializeComponent();  }  Cars AddPage;  public AddCar(Cars a)  {  InitializeComponent();  AddPage = a;  }  private void Add\_Click(object sender, RoutedEventArgs e)  {  if (ValidateForm())  {  Model.Car newCar = new Model.Car()  {  CarMark = AddMark.Text,  CarModel = AddModel.Text,  Year = Int32.Parse(AddYear.Text),  Engine = AddEngine.Text,  Color = AddColor.Text,  Body = AddBody.Text,  RegNummber = AddRegNummber.Text  };  db.Cars.Add(newCar);  db.SaveChanges();  this.Hide();  }  }  private bool ValidateForm()  {  string msgerror = "";  bool error = false;  if (AddMark.Text == String.Empty)  {  error = true;  msgerror += "Введите марку!\n";  }    else if (AddMark.Text.Length < 3)  {  error = true;  msgerror += "Марка указана не верно!\n";  }  if (AddMark.Text.Length > 100)  {  error = true;  msgerror += "Марка превышает максимальное количество символов(100)!\n";  }  if (AddModel.Text == String.Empty)  {  error = true;  msgerror += "Введите модель!\n";  }    else if (AddModel.Text.Length < 3)  {  error = true;  msgerror += "Модель указана не верно!\n";  }  if (AddModel.Text.Length > 100)  {  error = true;  msgerror += "Модель превышает максимальное количество символов(100)!\n";  }  if (AddRegNummber.Text == String.Empty)  {  error = true;  msgerror += "Введите регистрационный номер!\n";  }  else if (AddRegNummber.Text.Length < 3)  {  error = true;  msgerror += "Регистрационный номер указан не верно!\n";  }  if (AddRegNummber.Text.Length > 20)  {  error = true;  msgerror += "Регистрационный номер превышает максимальное количество символов(20)!\n";  }  if (AddYear.Text == String.Empty)  {  error = true;  msgerror += "Введите год!\n";  }  else if (!AddYear.Text.All(char.IsDigit))  {  error = true;  msgerror += "Введен некорректный год!\n";  }  else if (Single.Parse(AddYear.Text) >= 3000 || Single.Parse(AddYear.Text) < 1800)  {  error = true;  msgerror += "Введен некорректный год!\n";  }  if (AddEngine.Text == String.Empty)  {  error = true;  msgerror += "Введите двигатель!\n";  }  else if (AddEngine.Text.Length < 2)  {  error = true;  msgerror += "Двигатель введён не верно!\n";  }  if (AddEngine.Text.Length > 100)  {  error = true;  msgerror += "Двигатель введён не верно!\n";  }  if (AddBody.Text == String.Empty)  {  error = true;  msgerror += "Введите кузов!\n";  }  else if (AddBody.Text.Length < 2)  {  error = true;  msgerror += "Кузов введён не верно!\n";  }  if (AddBody.Text.Length > 100)  {  error = true;  msgerror += "Кузов введён неверно!\n";  }  if (AddColor.Text == String.Empty)  {  error = true;  msgerror += "Введите цвет!\n";  }    else if (AddColor.Text.Length < 3)  {  error = true;  msgerror += "Цвет указан не верно!\n";  }  if (AddColor.Text.Length > 100)  {  error = true;  msgerror += "Цвет превышает максимальное количество символов(100)!\n";  }  if (error)  {  rusty.Resources.Style.CustomMessageBox.CustomMessageBox.Show("Ошибка добавления", msgerror, MessageBoxButton.OK);  return false;  }  return true;  }  } |

Листинг класса UpdateCar

|  |
| --- |
| public partial class UpdateCar : Window  {  STOModelContext db = new STOModelContext();  public string Model;  public string Mark;  public int Year;  public string Body;  public string Color;  public string Number;  public string Engine;  Cars Car;  public UpdateCar()  {  InitializeComponent();  }  public UpdateCar(string model, string mark, int year, string body, string color, string number, string engine, Cars car)  {  InitializeComponent();  Model = model;  Mark = mark;  Year = year;  Body = body;  Color = color;  Number = number;  Engine = engine;  Car = car;  }  private void Update\_Click(object sender, RoutedEventArgs e)  {  if (ValidateForm())  {  Model.Car UpdateCar = (from car in db.Cars  where (  car.CarModel == Model &&  car.CarMark == Mark &&  car.Year == Year &&  car.Body == Body &&  car.Color == Color &&  car.Engine == Engine &&  car.RegNummber == Number  )  select car).Single();  if (UpdateModel.Text != String.Empty)  UpdateCar.CarModel = UpdateModel.Text;  if (UpdateMark.Text != String.Empty)  UpdateCar.CarMark = UpdateMark.Text;  if (UpdateYear.Text != String.Empty)  UpdateCar.Year = Int32.Parse(UpdateYear.Text);  if (UpdateBody.Text != String.Empty)  UpdateCar.Body = UpdateBody.Text;  if (UpdateColor.Text != String.Empty)  UpdateCar.Color = UpdateColor.Text;  if (UpdateEngine.Text != String.Empty)  UpdateCar.Engine = UpdateEngine.Text;  if (UpdateRegNummber.Text != String.Empty)  UpdateCar.RegNummber = UpdateRegNummber.Text;  db.SaveChanges();  Cars.a.ItemsSource = db.Cars.ToList();  this.Hide();  }  }  private bool ValidateForm()  {  string msgerror = "";  bool error = false;  if (UpdateMark.Text.Length < 3 && UpdateMark.Text != String.Empty)  {  error = true;  msgerror += "Марка указана не верно!\n";  }  if (UpdateMark.Text.Length > 100)  {  error = true;  msgerror += "Марка превышает максимальное количество символов(100)!\n";  }  if (UpdateModel.Text.Length < 3 && UpdateModel.Text != String.Empty)  {  error = true;  msgerror += "Модель указана не верно!\n";  }  if (UpdateModel.Text.Length > 100)  {  error = true;  msgerror += "Модель превышает максимальное количество символов(100)!\n";  }  if (UpdateRegNummber.Text.Length < 3 && UpdateRegNummber.Text != String.Empty)  {  error = true;  msgerror += "Регистрационный номер указан не верно!\n";  }  if (UpdateRegNummber.Text.Length > 20)  {  error = true;  msgerror += "Регистрационный номер превышает максимальное количество символов(20)!\n";  }  if (!UpdateYear.Text.All(char.IsDigit))  {  error = true;  msgerror += "Введен некорректный год!\n";  }  else if (UpdateYear.Text != String.Empty)  {  if (Single.Parse(UpdateYear.Text) >= 3000 || Single.Parse(UpdateYear.Text) < 1800)  {  error = true;  msgerror += "Введен некорректный год!\n";  }  }    else if (UpdateEngine.Text.Length < 2 && UpdateEngine.Text != String.Empty)  {  error = true;  msgerror += "Двигатель введён не верно!\n";  }  if (UpdateEngine.Text.Length > 100)  {  error = true;  msgerror += "Двигатель введён не верно!\n";  }    if (UpdateBody.Text.Length < 2 && UpdateBody.Text != String.Empty)  {  error = true;  msgerror += "Кузов введён не верно!\n";  }  if (UpdateBody.Text.Length > 100)  {  error = true;  msgerror += "Кузов введён неверно!\n";  }      if (UpdateColor.Text.Length < 3 && UpdateColor.Text != String.Empty)  {  error = true;  msgerror += "Цвет указан не верно!\n";  }  if (UpdateColor.Text.Length > 100)  {  error = true;  msgerror += "Цвет превышает максимальное количество символов(100)!\n";  }  if (error)  {  rusty.Resources.Style.CustomMessageBox.CustomMessageBox.Show("Ошибка обновления", msgerror, MessageBoxButton.OK);  return false;  }  return true;  }  } |

Приложение Б

Сигнатуры методов классов Add- и Update-

