**МИНОБРНАУКИ РОССИИ**

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ОТЧЕТ**

**по учебной практике УП.02.01 (по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/3 группы

Мирошниченко Данил Егорович

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, Приморский пр. 63

(наименование и адрес организации)

Период прохождения практики

с «9» декабря 2024 г. по «21» декабря 2024 г.

Руководитель практики Шаврова Л. С.

(подпись) (расшифровка подписи)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на учебную практику (по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/3 группы

Мирошниченко Данил Егорович

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, Приморский пр. 63

(наименование и адрес организации)

Период прохождения практики

с «9» декабря 2024 г. по «21» декабря 2024 г.

**Виды работ, обязательные для выполнения** *(переносится из программы, соответствующего ПМ):*

* Участие в выработке требований к программному обеспечению;
* Стадии проектирования программного обеспечения;
* Разработка модулей программного обеспечения;
* Тестирование программных модулей и их интеграции;
* Разработка программной документации и стандарты кодирования.

**Индивидуальное задание: ВАРИАНТ 3**

Задание выдал «7» октября 2024 г. \_\_\_\_\_\_\_\_\_\_ Шаврова Л.С.

(подпись) (Ф.И.О.)

Задание получил «7» октября 2024 г. \_\_\_\_\_\_\_\_\_\_ Мирошниченко Д.Е.(подпись) (Ф.И.О.)

**МИНОБРНАУКИ РОССИИ**

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПбПУ»)

**Институт среднего профессионального образования**

**ДНЕВНИК**

**прохождения учебной практики УП.02.01**

**(по профилю специальности)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/3 группы

Мирошниченко Данил Егорович

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, Приморский пр. 63

(наименование и адрес организации)

Период прохождения практики

с «9» декабря 2024 г. по «21» декабря 2024 г.

Руководитель практики Шаврова Л.С.

(подпись) (расшифровка подписи)

Итоговая оценка по практике

Санкт-Петербург

2024

**Содержание дневника**

|  |  |  |
| --- | --- | --- |
| **Дата** | **Виды выполненных работ и заданий по программе практики** | **Подпись руководителя практики** |
| **1** | **2** | **3** |
| 09.12 | Анализ предметной области. Выявление требований к программе. |  |
| 10.12 | Разработка технического задания. UML. Проектирование диаграммы вариантов использования. |  |
| 11.12 | UML. Проектирование диаграммы последовательности. UML. Проектирование диаграммы активности. |  |
| 12.12 | Моделирование структуры ПО. Проектирование инфологической и даталогической модели данных. |  |
| 13.12 | Проектирование интерфейса пользователя. Создание Wireframe эскизов. Разработка дизайна программы в соответствии с руководством по стилю. |  |
| 14.12 | Разработка базы данных. Разработка словаря данных. |  |
| 15.12 | Создание приложения. Форма авторизации. Создание приложения. Форма заказов. |  |
| 17.12 | Создание приложения. Основные формы приложения. Разработка библиотеки классов. |  |
| 18.12 | Подготовка отчетов и выгрузка документов для печати. Отладка программных модулей. |  |
| 19.12 | Модульное тестирование. Создание тестовых случаев. |  |
| 20.12 | Интеграционное тестирование. Разработка самодокументирующегося кода. |  |
| 21.12 | Инспекция кода на соответствие стандартам кодирования. Подготовка отчета и размещение результатов в репозитории контроля версий. |  |

**Содержание**

[ВВЕДЕНИЕ 7](#_Toc185373917)

[1 Общая часть 8](#_Toc185373918)

[1.1 Проектирование системы 8](#_Toc185373919)

[1.1.1 Краткая спецификация 8](#_Toc185373920)

[1.1.2 Диаграмма вариантов использования 11](#_Toc185373921)

[1.1.3 Диаграмма активности 12](#_Toc185373922)

[1.1.4 Диаграмма последовательности 13](#_Toc185373923)

[1.1.5 Общий алгоритм 14](#_Toc185373924)

[1.1.6 Алгоритм одной из функций 15](#_Toc185373925)

[1.1.7 Разработка макетов интерфейса системы 16](#_Toc185373926)

[1.2 Создание базы данных и заполнение таблиц данными 21](#_Toc185373927)

[1.2.1 ER-диаграмма 21](#_Toc185373928)

[1.2.2 Словарь данных 22](#_Toc185373929)

[1.2.3 Заполненные данными таблицы 25](#_Toc185373930)

[1.2.4 Резервное копирование 27](#_Toc185373931)

[1.3 Разработка библиотеки и подключение её к проекту 28](#_Toc185373932)

[1.4 Разработка приложения 28](#_Toc185373933)

[1.5 Тестирование приложения 45](#_Toc185373934)

[1.5.1 Создание тестовых случаев 45](#_Toc185373935)

[1.5.2 Модульное тестирование 45](#_Toc185373936)

[1.6 Выгрузка готового проекта в репозиторий GIT 47](#_Toc185373937)

[1.7 Качественные характеристики кода 48](#_Toc185373938)

[1.7.1 Полнота обработки ошибочных данных 48](#_Toc185373939)

[1.7.2 Наличие тестов для проверки допустимых значений входных данных 49](#_Toc185373940)

[1.7.3 Наличие средств контроля корректности входных данных 50](#_Toc185373941)

[1.7.4 Наличие средств восстановления при сбоях оборудования 50](#_Toc185373942)

[1.7.5 Наличие комментариев 50](#_Toc185373943)

[1.7.6 Наличие проверки корректности передаваемых данных 50](#_Toc185373944)

[1.7.7 Наличие описания основных функций 51](#_Toc185373945)

[ЗАКЛЮЧЕНИЕ 53](#_Toc185373946)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 54](#_Toc185373947)

[Приложение А 55](#_Toc185373948)

[Приложение Б 57](#_Toc185373949)

[Приложение В 59](#_Toc185373950)

[Приложение Г 99](#_Toc185373951)

[Приложение Д 105](#_Toc185373952)

# **ВВЕДЕНИЕ**

Активное применение компьютерных технологий для хранения и передачи информации на станциях технического обслуживания способствует повышению эффективности учета заявок, сокращению времени их обработки, оптимизации использования ресурсов и улучшению удовлетворенности клиентов.

Разработка программного модуля для управления заявками на ремонт автомобилей включает создание программного обеспечения, которое позволит автосервисам более эффективно отслеживать и управлять заявками.

Основная цель создания данного модуля заключается в автоматизации учета и обработки заявок на ремонт, что обеспечит сотрудникам сервисов удобное и прозрачное управление процессом, контроль статусов ремонта и улучшение организации работы.

Задачи по разработке программного модуля включают:

* Анализ предметной области и разработка технического задания.
* Проектирование диаграмм вариантов использования, последовательности и активности.
* Моделирование структуры программного обеспечения.
* Проектирование пользовательского интерфейса.
* Разработка базы данных и словаря данных.
* Создание приложения, включая формы авторизации, заказов и основные формы приложения, а также разработка библиотеки классов.
* Отладка программных модулей.
* Модульное тестирование и создание тестовых случаев.
* Инспекция кода на соответствие стандартам кодирования.
* Подготовка отчета и размещение результатов в репозитории Git.

# **Общая часть**

## **Проектирование системы**

Проектирование системы играет важную роль в создании детального плана, который помогает эффективно организовать процесс разработки, определить ключевые этапы и установить сроки выполнения. Это также позволяет учесть правовые и технические аспекты проекта. Грамотно спланированная архитектура системы дает возможность команде заранее оценить объем работ и возможные сложности, что помогает избежать лишних исправлений и длительных согласований. Это способствует улучшению координации внутри команды и повышению общей удовлетворенности как со стороны заказчика, так и исполнителей, минимизируя риски на этапе реализации.

### **Краткая спецификация**

1. Выполнить при разработке:

* Функцию регистрации и входа: позволяет разграничить роли пользователей в приложении. Вход в систему осуществляется с помощью логина и пароля.
* Функцию редактирования заявки для оператора и клиента: оператор может изменять статус заявки и назначать механика для ее выполнения, а клиент может создавать, редактировать и удалять свои заявки.
* Функцию создания заявки: создание заявки доступно клиенту, при этом в таблицу Requests вносятся вид автомобиля, модель, проблема (выбирается из выпадающего списка), ФИО и телефон клиента.
* Функцию обработки заявки: обработка заявки выполняется оператором, который назначает автомеханика для ее выполнения.
* Функцию изменения статуса заявки оператором и автомехаником.
* Функцию составления отчетности: автомеханик может составлять отчет по выбранной заявке в формате .txt.
* Функцию расчета среднего времени ремонта: эта функция позволяет отслеживать эффективность выполнения заявок, рассчитывая среднее время, необходимое для ремонта или обслуживания каждого типа заявок.

Необходимо обеспечить:

* Источники бесперебойного питания.
* Отказоустойчивое оборудование.
* Дублирование носителей информации.
* Целостность данных.

1. Составить техническую документацию и отчетность:

* Разрабатываемые программные модули,
* «Руководство системного программиста».

В таблице 1 представлено определение входных и выходных данных.

Таблица 1 – Определение входных и выходных данных

| **Входные данные** | **Ожидаемый результат** | **Фактический результат/Выходные данные** |
| --- | --- | --- |
| Логин “login4”, пароль “pass4” | Открытие окна оператора | Открытие окна оператора |
| Логин “login4”, пароль “11111” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |
| Логин “login2”, пароль “pass2” | Открытие окна автомеханика | Открытие окна автомеханика |
| Логин “login2”, пароль “11111” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |
| Логин “login11”, пароль “pass11” | Открытие страницы клиента | Открытие страницы клиента |
| Логин “login11”, пароль “11111” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |
| Логин “11111”, пароль “pass11” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |
| Логин “11111”, пароль “pass2” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |
| Логин “11111”, пароль “pass4” | Сообщение об ошибке “Неверный логин или пароль” | Сообщение об ошибке “Неверный логин или пароль” |

### **Диаграмма вариантов использования**

На рисунке 1 представлена диаграмма вариантов использования, которая отражает основной функционал пользователей программного обеспечения по ремонту автомобилей.



Рисунок 1 – Диаграмма вариантов использования

### **Диаграмма активности**

На рисунке 2 представлена диаграмма активности, которая отображает динамические аспекты поведения системы. На диаграмме представлен процесс создания клиентом заявки на ремонт автомобиля.



Рисунок 2 – Диаграмма активности

### **Диаграмма последовательности**

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования. На рисунке 3 представлена диаграмма, которая отражает процесс редактирования заявки заказчиком.

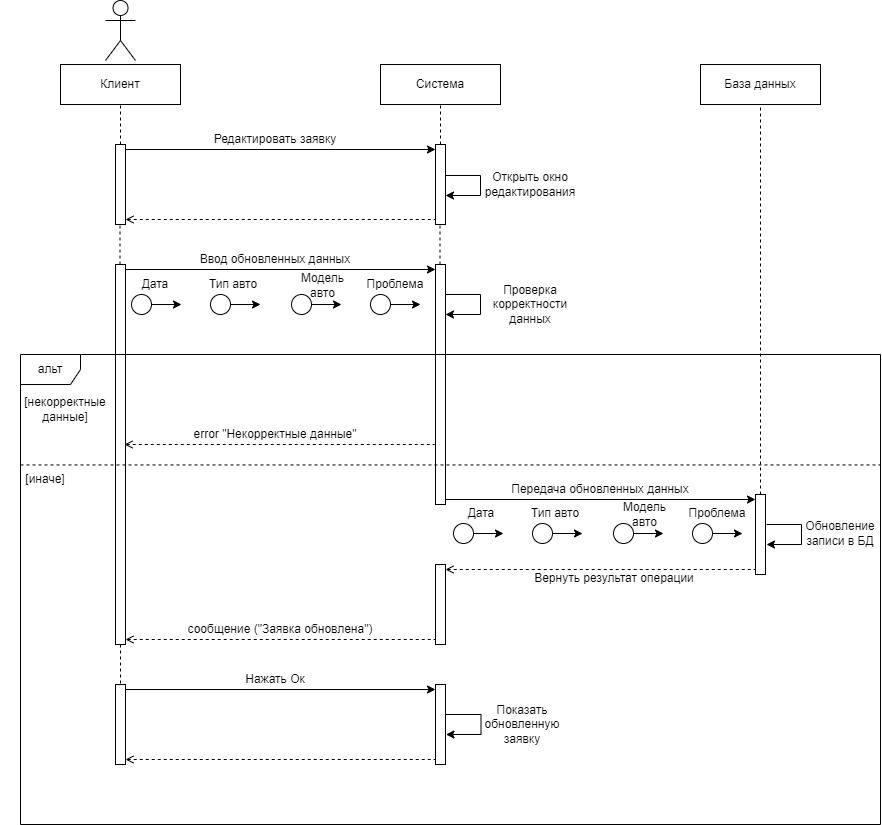


Рисунок 3 – Диаграмма последовательности

### **Общий алгоритм**

Общий алгоритм описывает те действия, которые обязательно произойдут при запуске программы. Общий алгоритм данной системы представлен на рисунке 4.

Изображение выглядит как текст, снимок экрана, Шрифт, графический дизайн

Автоматически созданное описание

Рисунок 4 – Общий алгоритм

### **Алгоритм одной из функций**

На рисунке 5 представлен подробный алгоритм одной из функций данного приложения, а именно расчета среднего времени ремонта.



Рисунок 5 – Алгоритм функции расчета среднего времени ремонта

### **Разработка макетов интерфейса системы**

Создание макетов — это процесс, в котором графические дизайнеры, верстальщики и программисты разрабатывают элементы, необходимые для корректного функционирования графических интерфейсов. Обычно работа начинается с создания макета, который представляет собой статичное изображение или набор элементов, планируемых для использования в приложении.

Грамотно разработанные макеты помогают ускорить процесс разработки интерфейса, обеспечивая прозрачное взаимодействие между командой разработчиков и заказчиком. Хорошо спроектированный макет позволяет визуализировать будущий вид приложения ещё до начала его реализации.

Макеты и мокапы интерфейса, выполненные в графических редакторах, представлены на рисунках 6–15.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 6 – Макет формы авторизации

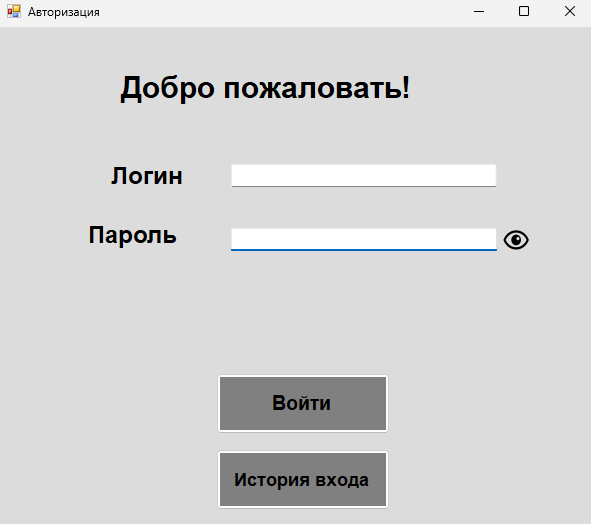


Рисунок 7 – Мокап формы авторизации

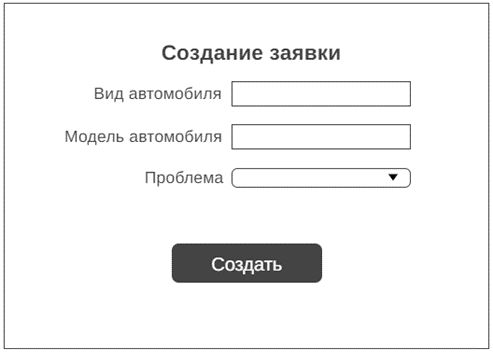


Рисунок 8 – Макет формы создания заявки

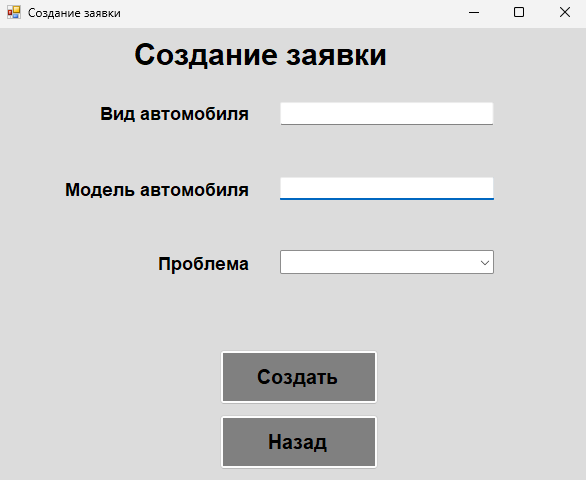


Рисунок 9 – Мокап формы создания заявки

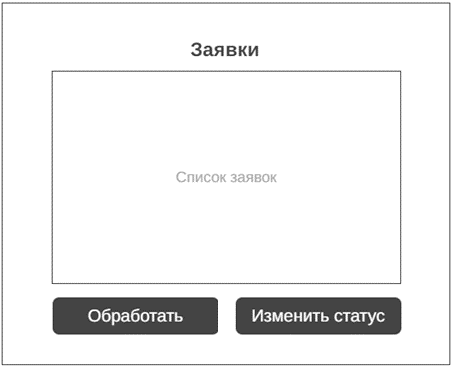


Рисунок 10 – Макет формы просмотра заявок оператором

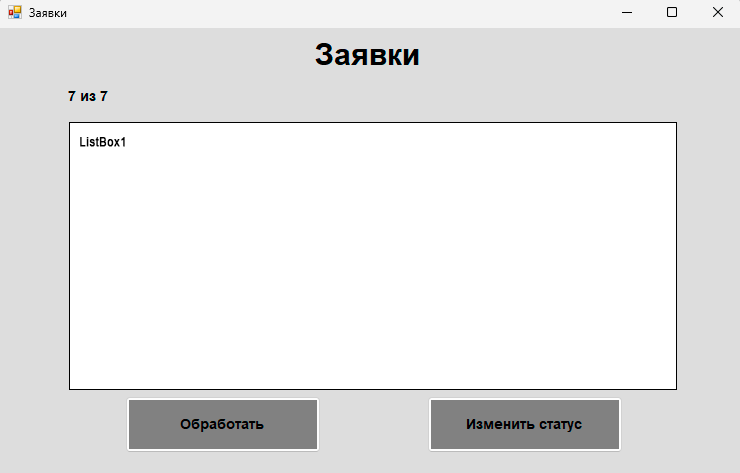


Рисунок 11 – Мокап формы просмотра заявок оператором

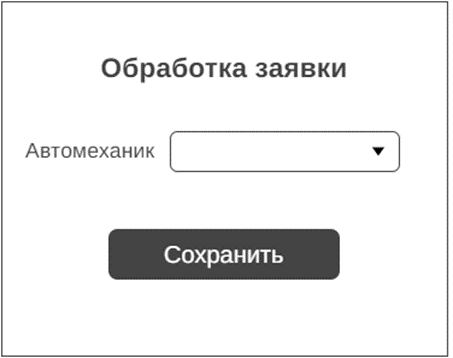


Рисунок 12 – Макет формы обработки заявки

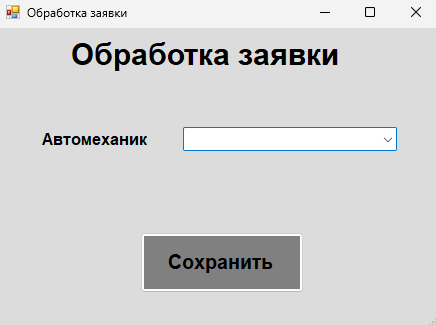


Рисунок 13 – Мокап формы обработки заявки

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 14 – Макет формы изменения статуса заявки

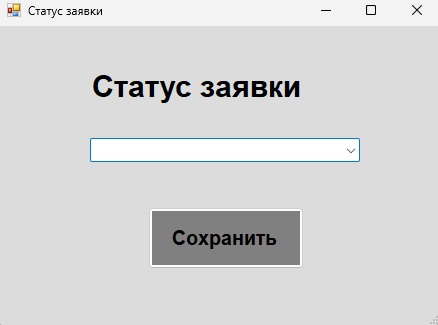


Рисунок 15 – Мокап формы изменения статуса заявки

## **Создание базы данных и заполнение таблиц данными**

База данных – это совокупность структурированных и взаимосвязанных данных и методов, обеспечивающих добавление выборку и отображение данных.

Скрипт таблиц базы данных представлен в приложении А.

### **ER-диаграмма**

Перед разработкой ПО нужно определить, с какими данными предстоит работать и как они связаны между собой. Для этого системные аналитики строят модели данных и создают ER-диаграммы. Собрав все сущности будущего проекта, было выяснено, как они связаны между собой, и была составлена ER-модель. ER-модель данной программы представлена на рисунке 16.

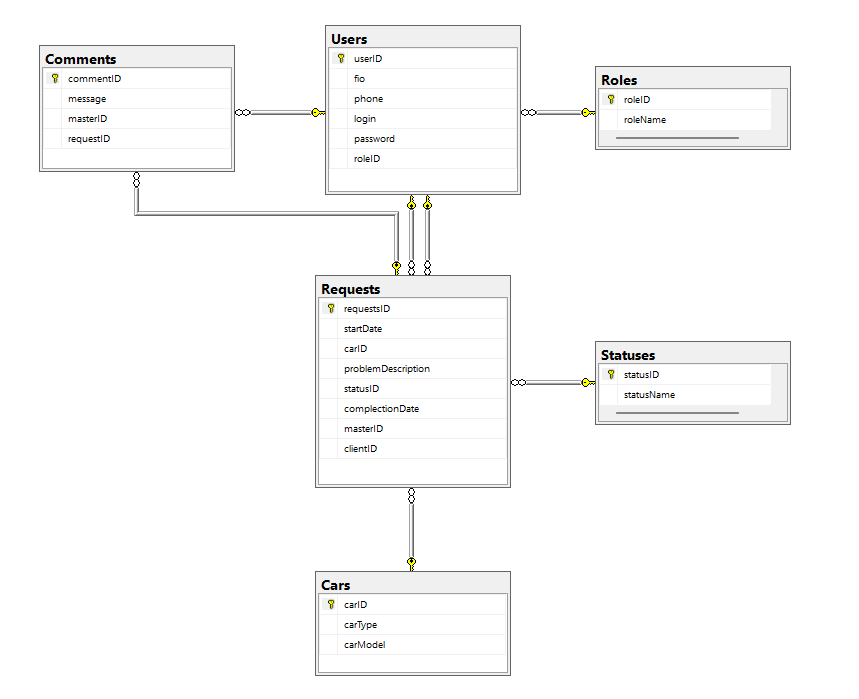


Рисунок 16 – ER-диаграмма

### **Словарь данных**

Словарь данных — это набор реляционных таблиц и представлений, которые содержат информацию о таблицах или метаданные. Словарь данных содержит информацию об источниках, форматах и взаимосвязях между данными, их описания, сведения о характере использования и распределении ответственности. Словарь данных можно рассматривать как вспомогательную базу данных, в которой хранится информация об основной базе данных.

В таблице 2 приведен словарь данных для отношения базы данных «Roles».

Таблица 2 – Словарь данных таблицы «Roles»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| PK | roleID | INT | Y | Auto Increment |
|  | roleName | NVARCHAR(50) | Y |  |

В таблице 3 приведен словарь данных для отношения базы данных «Users».

Таблица 3 – Словарь данных таблицы «Users»

| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| --- | --- | --- | --- | --- |
| PK | userID | INT | Y | Auto Increment |
|  | fio | NVARCHAR(50) | Y |  |
|  | phone | NVARCHAR(12) | Y |  |
|  | login | NVARCHAR(50) | Y |  |
|  | password | NVARCHAR(50) | Y |  |
| FK(Roles) | roleID | INT | N |  |

В таблице 4 приведен словарь данных для отношения базы данных «Cars».

Таблица 4 – Словарь данных таблицы «Cars»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| PK | CarID | INT | Y | Auto Increment |
|  | carType | VARCHAR(50) | Y |  |
|  | carModel | VARCHAR(100) |  |  |

В таблице 5 приведен словарь данных для отношения базы данных «Reguests».

Таблица 5 – Словарь данных таблицы «Requests»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| PK | requestsID | INT | Y | Auto Increment |
|  | startDate | DATE | Y |  |
| FK(Cars) | carID | INT | N |  |
|  | problemDescription | TEXT | Y |  |
| FK(Status) | statusID | INT | N |  |
|  | completionDate | DATE | N |  |
| FK(Users) | masterID | INT | N |  |
| FK(Users) | clientID | INT | N |  |

В таблице 6 приведен словарь данных для отношения базы данных «Comments».

Таблица 6 – Словарь данных таблицы «Comments»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| PK | commentID | INT | Y | Auto Increment |
|  | message | TEXT | Y |  |
| FK(Users) | masterID | INT | N |  |
| FK(Requests) | requestsID | INT | N |  |

В таблице 7 приведен словарь данных для отношения базы данных «Statuses».

Таблица 7 – Словарь данных таблицы «Statuses»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KEY** | **Имя поля** | **Тип данных / Размер** | **Необходимо?** | **Примечания** |
| PK | statusID | INT | Y | Auto Increment |
|  | statusName | VARCHAR(50) | Y |  |

### **Заполненные данными таблицы**

Операция вставки данных в реляционных БД — одна из наиболее востребованных, без которой невозможна нормальная работа с таблицами. В данном приложении таблицы были заполнены средствами реляционных СУБД, работающих с языком SQL.

Со скриптом заполнения базы данных можно ознакомиться в приложении А.

Результаты заполнения таблиц представлены на рисунках 17-22.

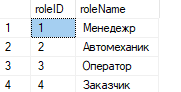


Рисунок 17 – Таблица «Roles»

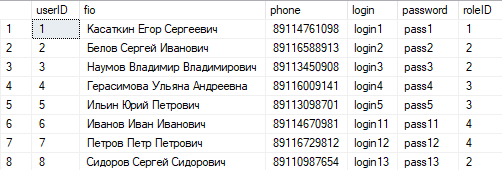


Рисунок 18 – Таблица «Users»

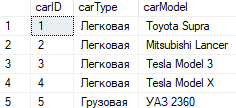


Рисунок 19 – Таблица «Cars»

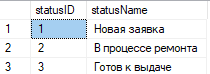


Рисунок 20 – Таблица «Statuses»

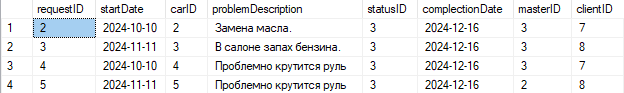


Рисунок 21 – Таблица «Requests»

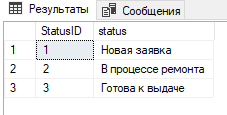


Рисунок 22 – Таблица «Comments»

### **Резервное копирование**

Резервным копированием называется сохранение копии данных где-то вне основного места их хранения. Главное назначение резервного копирования – восстановление данных после их потери.

Создание резервной копии базы данных и восстановление базы данных представлено на рисунках 23-24.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 23 – Создание резервной копии



Рисунок 24 – Файл резервной копии

## **Разработка библиотеки и подключение её к проекту**

Динамическая библиотека — это отдельный исполняемый файл на машинных кодах. Полностью термин звучит как «библиотека подпрограмм». Функции, объекты и другие сущности, которые входят в библиотеку, можно использовать в коде и тем самым облегчать задачу разработки. Например, не писать с нуля сложный алгоритм, а вызвать функцию из библиотеки, где он уже реализован.

Код библиотеки представлен в приложении Б.

## **Разработка приложения**

После установки программы на компьютер и ее запуска на экране появляется окно авторизации пользователя с предложением ввести имя пользователя и его пароль. Внешний вид экранной формы авторизации показан на рисунке 25.

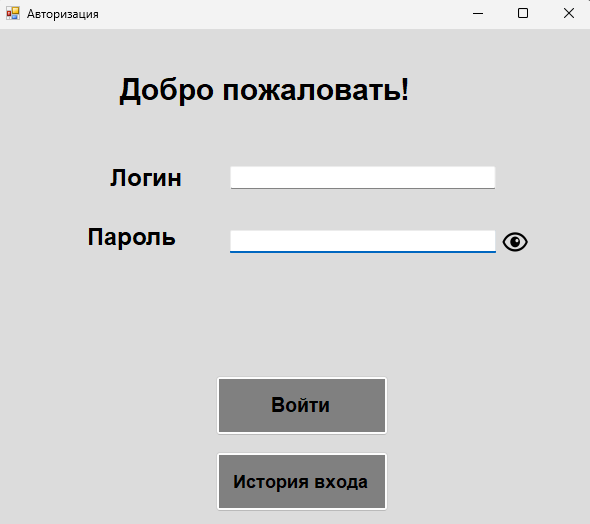


Рисунок 25 – Окно авторизации

Окно авторизации содержит следующие элементы:

1. Текстовое поле для ввода логина – предназначено для ввода уникального идентификатора пользователя.
2. Текстовое поле для ввода пароля – ввод пароля осуществляется в скрытом режиме (символы заменяются точками для защиты информации).
3. Кнопка «Войти» – проверяет введенные данные на корректность. Если логин и пароль введены правильно, система предоставляет доступ к интерфейсу пользователя в зависимости от его роли. В случае некорректных данных выводится сообщение об ошибке.
4. Кнопка «История входа» – открывает экранное окно с таблицей, в которой отображена история входов.
5. Кнопка «Отмена» – закрывает окно авторизации и завершает работу программы.
6. Капча (если активна) – в случае нескольких неудачных попыток авторизации появляется капча для предотвращения автоматизированных попыток входа. Пользователь должен ввести сгенерированный системой код из капчи для подтверждения, что он не является ботом.
7. Счетчик попыток – после нескольких неудачных попыток (например, двух) программа блокирует учетную запись на определенное время (например, 3 минуты). Система выводит сообщение о блокировке и информирует, когда можно будет повторить попытку входа.
8. Кнопка «Показать/скрыть пароль» – позволяет пользователю временно отобразить или скрыть введенный пароль, чтобы убедиться в его правильности.

Проверка модуля авторизации включает:

1. Вход с корректными данными – при вводе верного логина и пароля система должна перенаправить пользователя на соответствующее рабочее окно (например, окно клиента, оператора или мастера).
2. Вход с некорректными данными – при вводе неверного логина или пароля система должна вывести сообщение об ошибке и увеличить счетчик неудачных попыток. При достижении предела попыток должна включаться защита (например, капча или блокировка на время).
3. Работа капчи – после первой неудачной попытки входа появляется капча, которую пользователь должен ввести для подтверждения, что он не бот. Система должна проверять корректность ввода капчи и при необходимости генерировать новый код.
4. Блокировка учетной записи – после нескольких неудачных попыток (например, двух) система блокирует возможность входа на заданный период времени. Проверяется правильность отображения сообщения о блокировке и возобновление работы после истечения времени блокировки.

После успешной/неуспешной авторизации данные об этой попытке сохраняются в базу данных, и пользователь может просмотреть их через окно истории авторизаций, предварительно нажав на соответствующую кнопку на форме авторизации. На экране истории отображаются данные о логине пользователя, времени попытки и ее статусе (успешно или ошибка). Внешний вид окна представлен на рисунке 26.

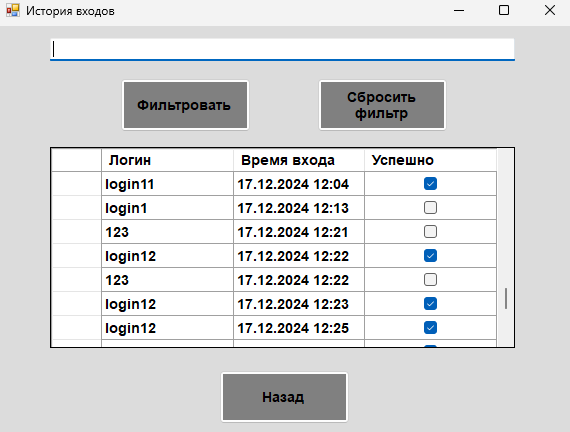


Рисунок 26 – окно истории входов

На экране истории входов представлены следующие элементы: поле фильтрации по логину, где можно ввести часть или полный логин для быстрого поиска конкретного пользователя, и таблица с логами авторизаций, содержащая столбцы:

1. Логин – отображает логин пользователя, который пытался войти.
2. Дата и время – показывает дату и время каждой попытки входа.
3. Статус – указывает, была ли попытка успешной или неуспешной.

Проверка этого модуля заключается в том, чтобы убедиться, что данные обо всех попытках входа корректно записываются в таблицу и отображаются в окне истории, а фильтрация по логину работает корректно.

После авторизации с помощью данных заказчика открывается окно пользователя. Экранный вид формы показан на рисунке 27.



Рисунок 27 – Окно заказчика

Окно клиента содержит следующие элементы:

1. Кнопка «Создать заявку» – при нажатии открывается форма для добавления новой заявки клиента. В форму передается идентификатор клиента для корректной работы с заявками. Текущая форма остаётся открытой.
2. Кнопка «Просмотреть заявки» – при нажатии открывается форма для просмотра существующих заявок клиента. Идентификатор клиента передается для отображения заявок именно этого клиента. Текущая форма закрывается.
3. Кнопка «Назад» – предназначена для выхода из учетной записи клиента и открытия формы авторизации. При нажатии закрывается текущая форма и открывается форма авторизации.

Проверка модуля формы клиента включает:

1. Добавление заявки – при нажатии кнопки «Добавить заявку» система должна корректно открыть форму для создания новой заявки и передать в неё идентификатор клиента. Проверяется корректность передачи данных и отображение формы без ошибок.
2. Просмотр заявок – при нажатии кнопки «Просмотреть заявки» система должна закрыть текущую форму и открыть форму для просмотра заявок клиента. Проверяется корректность передачи идентификатора клиента и отображение списка заявок этого клиента.
3. Выход из учетной записи – при нажатии кнопки «Выйти» система должна закрыть текущую форму и открыть форму авторизации. Проверяется правильность завершения работы текущей формы и открытия формы авторизации без задержек и ошибок.
4. Проверка корректности передачи данных – проверяется, что при каждом действии (добавление, просмотр заявок) идентификатор клиента передаётся в новые формы без изменений, что обеспечивает работу с правильными данными.
5. Закрытие текущей формы – проверяется, что при открытии форм для просмотра заявок и выхода из учетной записи, текущая форма закрывается без зависаний или других проблем.

После нажатия на главной форме заказчика кнопки «Создать заявку» открывается форма для создания заказчиком заявки. Экранное окно формы представлено на рисунке 28.

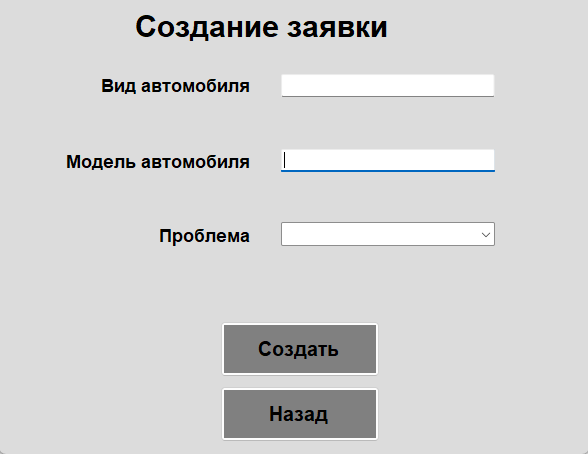


Рисунок 28 – Окно создания заявки

Форма добавления заявки содержит следующие элементы:

1. Текстовое поле для ввода вида автомобиля – предназначено для ввода типа автомобиля, который требует ремонта. Пользователь должен ввести значение вручную.
2. Текстовое поле для ввода модели автомобиля – используется для ввода модели автомобиля. Вводится вручную.
3. Выпадающий список для выбора проблемы – предоставляет пользователю список распространённых проблем с автомобилем. Пользователь выбирает одну из предложенных опций, таких как «Отказали тормоза», «Проблемы с электроникой» и др.
4. Текстовое поле для ввода ФИО клиента – необходимо для ввода полного имени клиента, чтобы связать заявку с конкретным человеком.
5. Текстовое поле для ввода номера телефона клиента – предназначено для ввода номера телефона клиента, который должен быть проверен на корректность (например, минимальная длина – 10 символов).
6. Кнопка «Создать» – сохраняет новую заявку. При нажатии кнопки происходит проверка корректности введённых данных и сохранение заявки в базу данных.
7. Кнопка «Назад» – закрывает форму добавления заявки без сохранения введённых данных.

Проверка модуля формы добавления заявки включает:

1. Проверка валидации данных – при нажатии кнопки "Добавить" система должна убедиться, что все поля заполнены корректно. Проверяется, что текстовые поля не пустые, выбран пункт в выпадающем списке, и номер телефона введён правильно.
2. Проверка списка проблем – проверяется, что выпадающий список содержит все предустановленные проблемы и пользователь может выбрать любую из них.
3. Проверка добавления автомобиля – система должна проверить, существует ли уже автомобиль с введёнными типом и моделью в базе данных. Если нет, автомобиль должен быть добавлен.
4. Проверка добавления заявки – при успешной валидации данных заявка должна быть добавлена в таблицу заявок. Проверяется корректность записи даты создания заявки, описания проблемы, статуса, идентификатора клиента и автомобиля.
5. Сообщение о результате – при успешном добавлении заявки система должна уведомить пользователя сообщением об успешной передаче заявки оператору. В случае ошибки отображается соответствующее сообщение с текстом ошибки.
6. Корректное закрытие формы – после успешного добавления заявки форма должна быть закрыта. Проверяется, что форма закрывается корректно без зависаний или ошибок.
7. Проверка корректного соединения с базой данных – проверяется, что соединение с базой данных устанавливается успешно, а операции вставки выполняются корректно.

После нажатия на главной форме заказчика кнопки «Просмотр заявок» открывается форма просмотра заявок заказчика. Экранное окно формы представлено на рисунке 29.

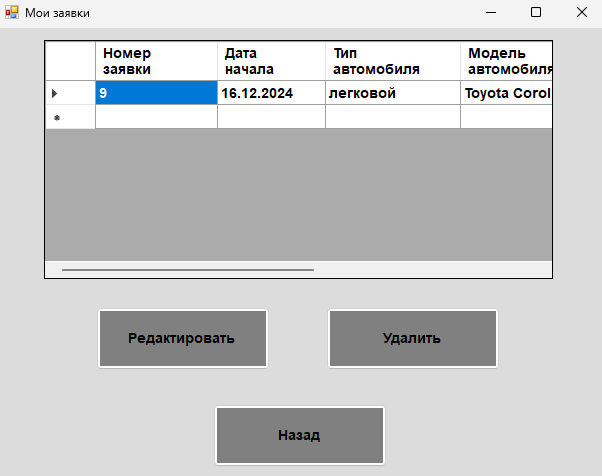


Рисунок 29 – Окно просмотра заявок заказчика

Форма просмотра заявок клиента содержит следующие элементы:

1. Таблица для отображения списка заявок клиента. Каждая строка содержит информацию о заявке, такую как дата создания, тип и модель автомобиля, описание проблемы, и статус заявки.
2. Кнопка «Редактировать» — открывает форму для редактирования выбранной заявки. Если заявка не выбрана, выводится предупреждение.
3. Кнопка «Удалить» — удаляет выбранную заявку после подтверждения. Если заявка не выбрана, выводится предупреждение.
4. Кнопка «Назад» — закрывает текущую форму и возвращает пользователя к форме клиента.

Проверка этого модуля заключается в том, чтобы убедиться, что данные обо всех заявках записываются в таблицу и отображаются корректно, открытии окна редактирования при нажатии на кнопку редактирования и выборе записи, а также производится удаление выбранной заявки с предшествующим уведомлением об удалении.

При нажатии на кнопку «Редактировать» в окне просмотра заявок заказчиком, открывается экранная форма редактирования заявки, представленная на рисунке 30.

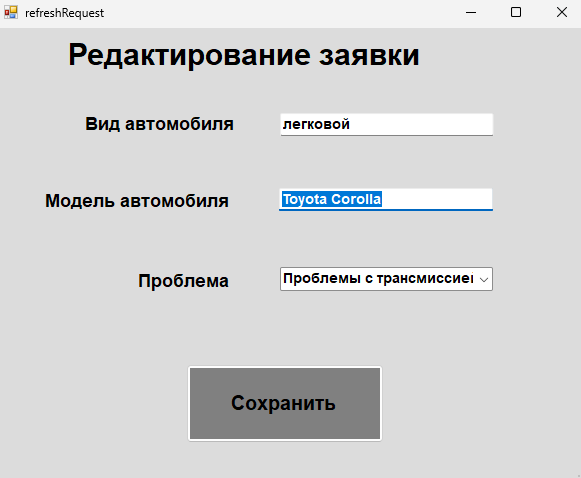


Рисунок 30 – Окно редактирования заявки

Элементами формы являются те же поля, что и на форме добавления заявки, и кнопка «Сохранить» которая обновляет выбранную заявку и закрывает форму редактирования.

Проверка этого модуля заключается в том, чтобы удостовериться в корректном обновлении записей.

После авторизации с помощью данных оператора открывается окно оператора. Экранный вид формы показан на рисунке 31.

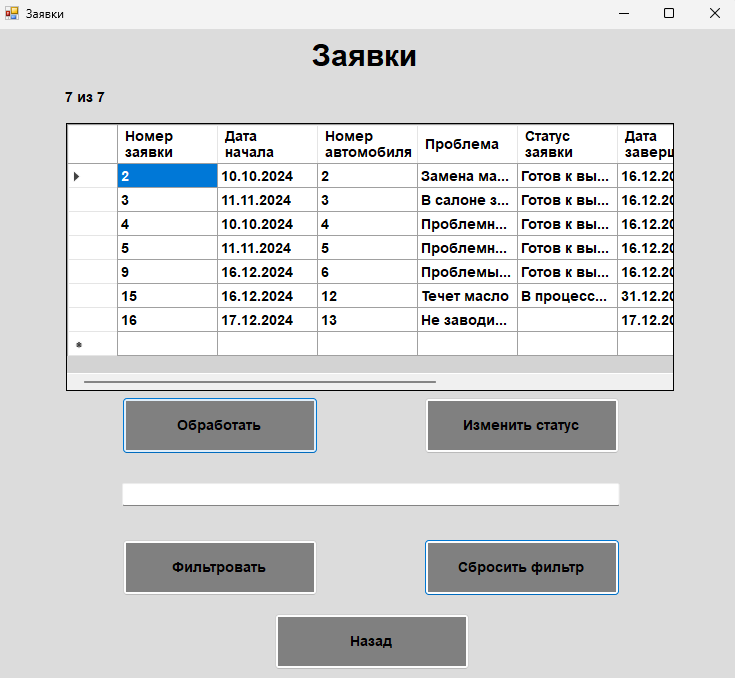


Рисунок 31 – Окно управления заявками оператора

Форма оператора содержит следующие элементы:

1. Таблица для отображения списка заявок. Каждая строка содержит информацию о заявке, такую как дата создания, тип и модель автомобиля, описание проблемы, и статус заявки.
2. Кнопка «Обработать» — открывает форму для обработки выбранной заявки. Если заявка не выбрана, выводится предупреждение.
3. Кнопка «Изменить статус» — открывает форму для изменения статуса выбранной заявки. Если заявка не выбрана, выводится предупреждение.
4. Текстовое поле для ввода фильтров.
5. Кнопки «Фильтровать» и «Сбросить фильтр» – первая отображает записи по заданному фильтру, вторая – сбрасывает фильтр и показывает все заявки.
6. Кнопка «Назад» — закрывает текущую форму и возвращает пользователя к форме клиента.

Проверка этого модуля заключается в том, чтобы убедиться, что данные при обработке и изменении статуса заявок отображаются корректно, а также корректной фильтрации и сортировке данных.

Экранная форма для обработки заявки оператором, которая открывается при выборе заявки и нажатии на соответствующую кнопку, представлена на рисунке 32.

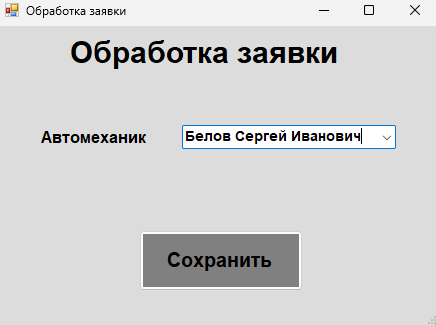


Рисунок 32 – Окно обработки заявок

Форма обработки заявки содержит следующие элементы:

1) Выпадающий список для назначения механика.

2) Кнопка «Сохранить» — сохраняет в таблице измененные данные о назначенном автомеханике и статусе заявки, закрывает форму обработки.

Проверка этого модуля заключается в том, чтобы убедиться, что после нажатия кнопки для заявки назначен автомеханик и статус заявки изменен на «В процессе».

Экранная форма для изменения статуса заявки оператором, которая открывается при выборе заявки и нажатии на соответствующую кнопку, представлена на рисунке 33.

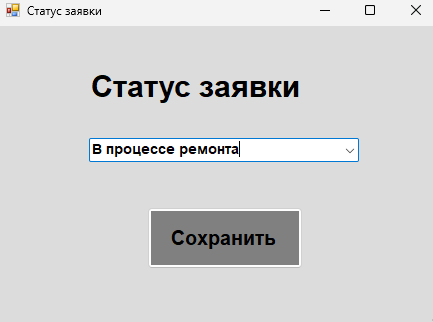


Рисунок 33 – Окно изменения статуса

Форма обработки заявки содержит следующие элементы:

1. Выпадающий список для выбора статуса.
2. Кнопка «Сохранить» — сохраняет в таблице измененные данные о статусе заявки.

Проверка этого модуля заключается в том, чтобы убедиться, что после нажатия кнопки статус заявки изменён на выбранный.

После авторизации с помощью данных автомеханика открывается окно автомеханика. Экранный вид формы показан на рисунке 34.

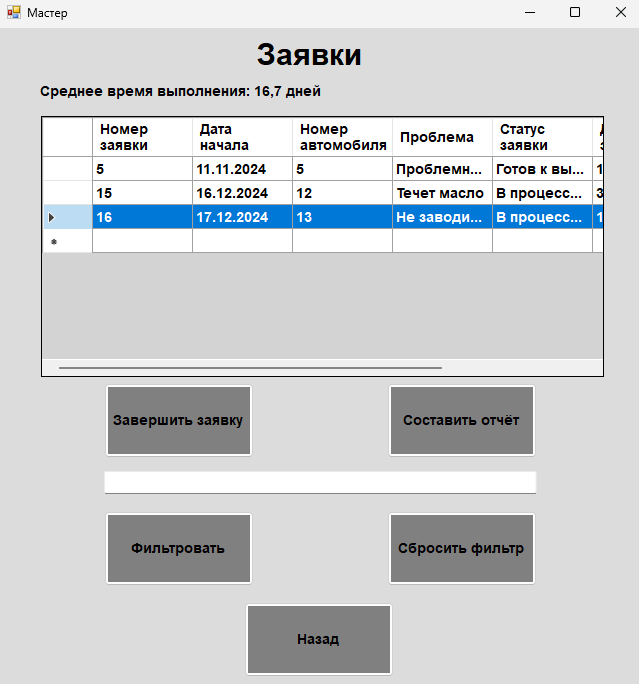


Рисунок 34 – Окно заявок автомеханика

Форма автомеханика содержит следующие элементы:

1. Таблица для отображения списка заявок автомеханика. Каждая строка содержит информацию о заявке, такую как дата создания, тип и модель автомобиля, описание проблемы, и статус заявки.
2. Кнопка «Завершить заявку» — изменяет статус выбранной заявки на «Готов к выдаче» и устанавливает дату выполнения заявки. Если заявка не выбрана, выводится предупреждение.
3. Кнопка «Составить отчет» — создает отчет по выбранной заявке в формате .txt. Если заявка не выбрана, выводится предупреждение.
4. Текстовое поле для ввода фильтров.
5. Кнопки «Фильтровать» и «Сбросить фильтр» – первая отображает записи по заданному фильтру, вторая – сбрасывает фильтр и показывает все заявки.
6. Кнопка «Назад» — закрывает текущую форму и возвращает пользователя к форме клиента.

Проверка этого модуля заключается в том, чтобы убедиться, что данные при завершении заявки отображаются корректно, составляется отчет по выбранной заявке, а также корректной фильтрации и сортировке данных.

После авторизации с помощью данных менеджера открывается окно менеджера. Экранный вид формы показан на рисунке 35.

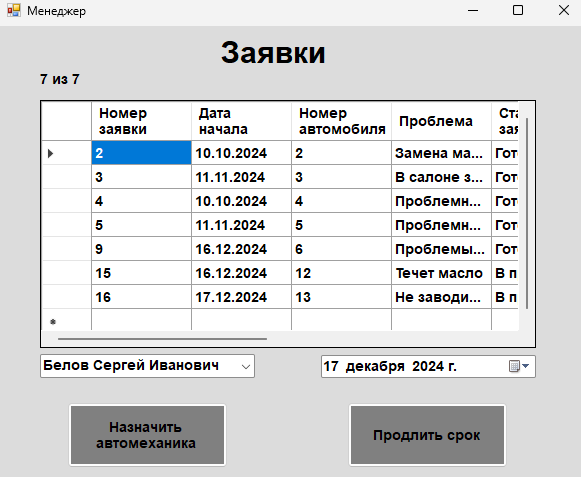


Рисунок 35 – Окно менеджера

Форма Менеджера содержит следующие элементы:

1. Таблица для отображения списка всех заявок. Каждая строка содержит информацию о заявке, такую как дата создания, тип и модель автомобиля, описание проблемы, и статус заявки.
2. Кнопка «Назначить автомеханика» — изменяет назначенного автомеханика выбранной заявки на выбранного из выпадающего списка. Если заявка не выбрана, выводится предупреждение.
3. Кнопка «Продлить срок» — изменяет срок завершения заявки на выбранную дату в календаре, изменяет статус заявки на «В процессе». Если заявка не выбрана, выводится предупреждение.
4. Выпадающий список для выбора назначаемого автомеханика.
5. DateTimePicker для продления даты завершения заявки.

Проверка этого модуля заключается в том, чтобы убедиться, что данные при назначении автомеханика и продления срока выполнения заявки отображаются корректно.

При разработке приложения были применены такие средства, как отладка при помощи точек останова.

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки. Чтобы понять, где возникла ошибка, приходится узнавать текущие значения переменных и выяснять, по какому пути выполнялась программа. Точки останова — это один из важнейших инструментов разработчика для выполнения отладки. Точки останова устанавливаются везде, где нужно приостановить выполнение отладчика. Например, вы хотите просмотреть состояние переменных кода или стек вызовов в определенной точке останова.

Код программы представлен в приложении В.

Отладка представлена на рисунках 36–37.

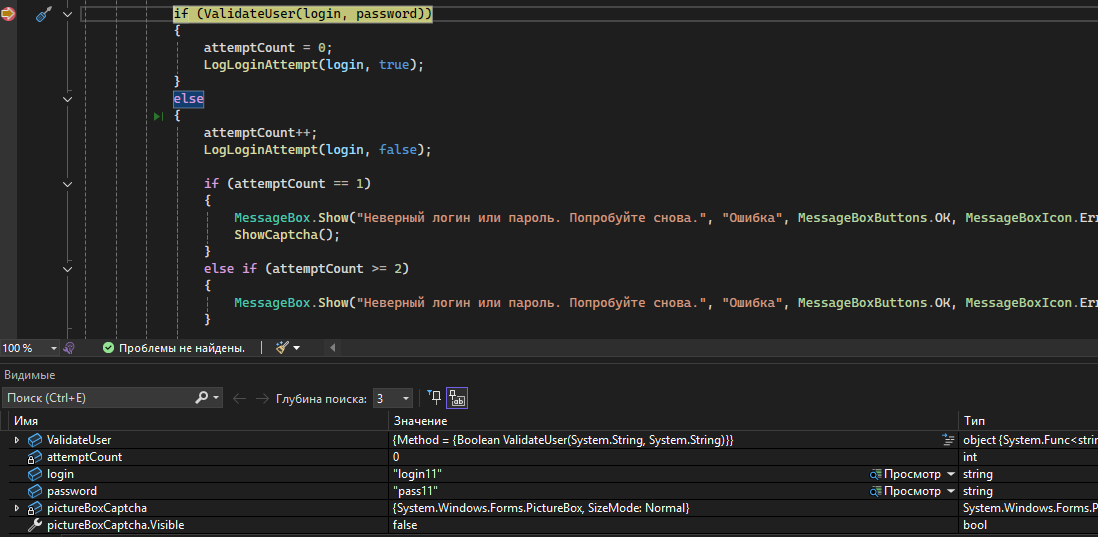


Рисунок 36 – Отладка логина и пароля

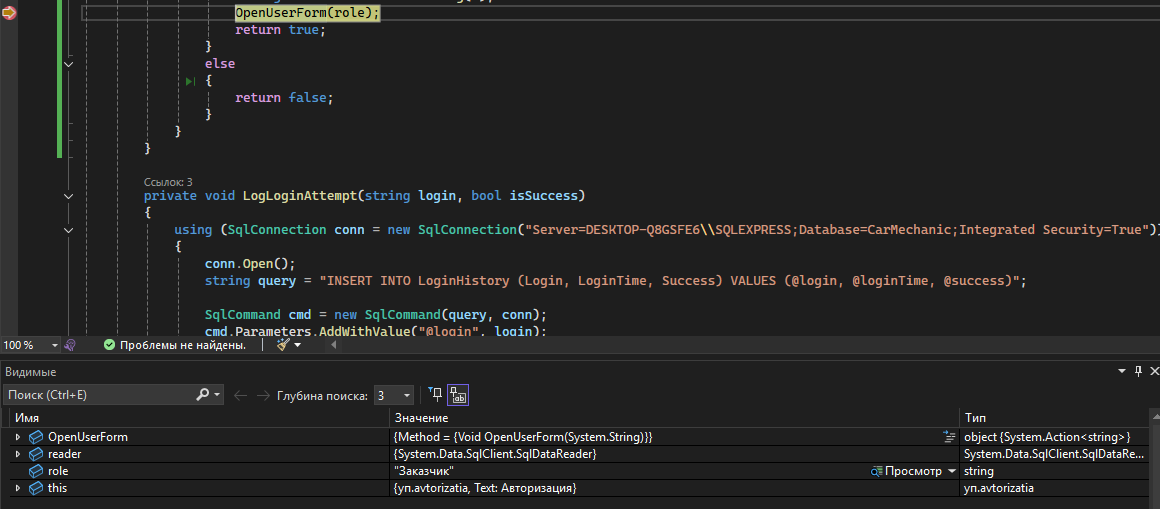


Рисунок 37 – Отладка определения роли со входом

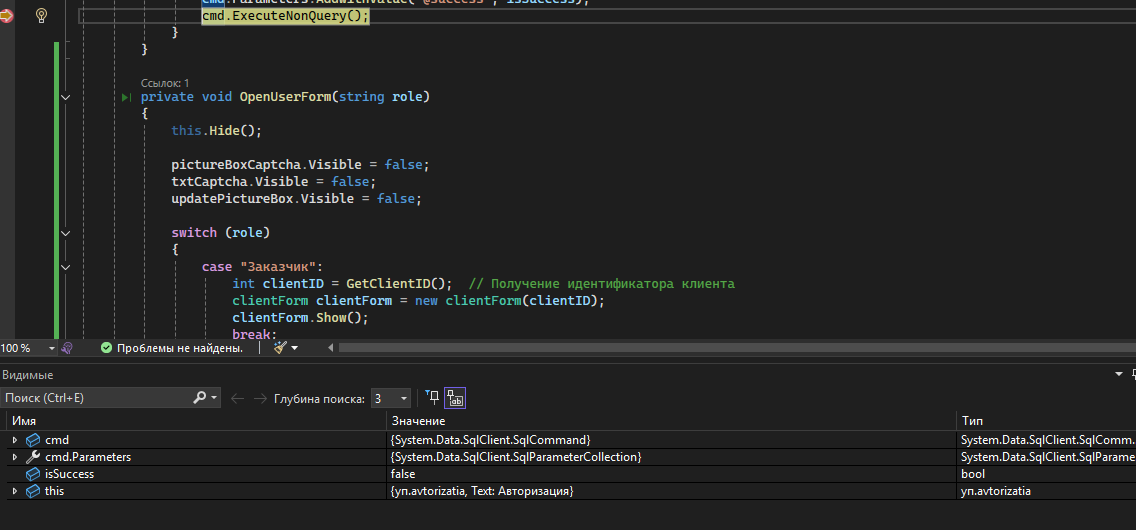


Рисунок 38 – Отладка запроса на получение данных для входа

## **Тестирование приложения**

Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.

Цель тестирования — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

### **Создание тестовых случаев**

Тестовый случай (тест-кейс) — это профессиональная документация тестировщика, последовательность действий, направленная на проверку какого-либо функционала, описывающая, как прийти к фактическому результату. Тест-кейсы помогают провести проверку продукта без ознакомления со всей документацией.

Тестовые случаи представлены в приложении Г.

### **Модульное тестирование**

Модульное тестирование, иногда блочное тестирование или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Также, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирование выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность.

Код модульного тестирования представлен в приложении Д.

Результат модульного тестирования библиотеки проекта представлен на рисунке 39.

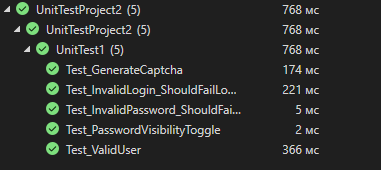


Рисунок 39 – Успешно пройденные юнит-тесты

## **Выгрузка готового проекта в репозиторий GIT**

Git — распределённая система управления версиями. Репозиторий в Git — это место, в котором хранится весь код и вся история его изменений. Он может быть локальным (на компьютере пользователя) или удалённым (на сервере или в облачном хранилище).

Для данного проекта был создан удаленный репозиторий на сайте Git Hub. Вид проекта, загруженного на сайт, предложен на рисунке 40.

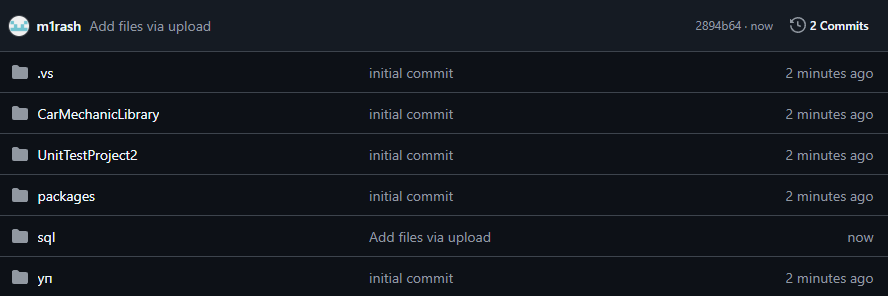


Рисунок 40 – Создание локального репозитория

## **Качественные характеристики кода**

Качественные характеристики кода — это набор признаков и аспектов, которые определяют, насколько хорошо написан программный код с точки зрения его эффективности, читаемости, удобства сопровождения и других факторов. Качественный код облегчает его поддержку, улучшает производительность и снижает количество ошибок в дальнейшем.

### **Полнота обработки ошибочных данных**

В коде программы предусмотрена обработка возможных ошибок при взаимодействии с базой данных. В целом, этот код отвечает за создание заявки в базе данных и обработку возможных ошибок, которые могут возникнуть в процессе выполнения этой операции.

try

{

requestCmd.ExecuteNonQuery();

MessageBox.Show("Заявка успешно создана и передана оператору.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Close(); // Закрываем форму после успешного создания заявки

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при создании заявки: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

### **Наличие тестов для проверки допустимых значений входных данных**

В коде реализована проверка допустимых значений при вводе данных в форме авторизации. Например, при вводе логина и пароля проверяется их наличие в бд через метод ValidateUser.

private bool ValidateUser(string login, string password)

{

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@login", login);

cmd.Parameters.AddWithValue("@password", password);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

string role = reader.GetString(0);

OpenUserForm(role);

return true;

}

else

{

return false;

}

}

}

Также, в форме авторизации при множественных неудачных попытках входа (3 раза) система блокирует доступ и сообщает об этом пользователю.

private int blockTime = 180000; // 3 минуты блокировки

if (attemptCount > 2)

{

BlockUser();

}

if (isBlocked)

{

MessageBox.Show("Вход заблокирован. Подождите 3 минуты.", "Блокировка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

### **Наличие средств контроля корректности входных данных**

В коде присутствует множество проверок корректности входных данных, например, пустых полей при создании пользователем заявки и вывода сообщения об этом.

if (string.IsNullOrWhiteSpace(textBoxCarType.Text))

{

MessageBox.Show("Введите вид автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

### **Наличие средств восстановления при сбоях оборудования**

Код не содержит явных механизмов восстановления после сбоев оборудования. Тем не менее, использование обработки исключений позволяет в определенной степени реагировать на проблемы с оборудованием (например, сбои при подключении к базе данных). В случае сбоя система выводит сообщение об ошибке и даёт пользователю информацию о том, что необходимо предпринять.

### **Наличие комментариев**

В коде присутствуют комментарии для объяснения неоднозначных моментов, например некоторых методов и функций.

### **Наличие проверки корректности передаваемых данных**

Код тщательно проверяет корректность вводимых пользователем данных, таких как логин, пароль и капча. Проверка происходит на уровне ввода данных с последующей валидацией с использованием запросов к базе данных.

if (ValidateUser(login, password))

{

attemptCount = 0;

LogLoginAttempt(login, true);

return;

}

else

{

attemptCount++;

LogLoginAttempt(login, false);

if (attemptCount == 1)

{

MessageBox.Show("Неверный логин или пароль. Попробуйте снова.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

ShowCaptcha();

}

### **Наличие описания основных функций**

Каждый метод выполняет свою отдельную задачу, что соответствует принципам структурированного и читаемого кода.

Основные методы программы:

* button1\_Click(object sender, EventArgs e) - Обработчик события для кнопки "Отправить", выполняет валидацию данных, добавляет информацию о автомобиле и запросе в базу данных, и отображает сообщения об успехе или ошибках.
* button1\_Click - Обрабатывает нажатие кнопки для входа в систему, проверяет, заблокирован ли пользователь, валидирует введенные данные, управляет попытками входа и отображает CAPTCHA при необходимости.
* button2\_Click - Обработчик события нажатия кнопки для удаления заявки. Проверяет, выбрана ли хотя бы одна заявка в DataGridView. Если выбрана, запрашивает подтверждение удаления. При положительном ответе вызывает метод удаления заявки и обновляет данные в таблице. Если ни одна заявка не выбрана, выводит предупреждающее сообщение.
* UpdateRequest - Метод для обновления данных заявки. Проверяет корректность введенных данных с помощью ValidateForm(). Если данные валидны, устанавливает соединение с базой данных и выполняет три обновляющих запроса: обновляет информацию о автомобиле, данные клиента и описание проблемы в заявке. После успешного выполнения запросов выводит сообщение об успешном обновлении, обновляет данные в родительской форме и закрывает текущую форму.

# **ЗАКЛЮЧЕНИЕ**

В результате учебной практики был создан программный продукта, который позволяет сервисам по ремонту автомобилей и эффективно отслеживать учет заявок на ремонт.

Была достигнута основная цель разработки программного продукта.

Были выполнены задачи:

* Анализ предметной области. Разработка технического задания.
* Проектирование диаграмм вариантов использования, последовательности, активности.
* Моделирование структуры ПО. Проектирование интерфейса пользователя.
* Разработка базы данных и словаря данных.
* Создание приложения: форма авторизации, истории входов, основных форм приложения с заявками. Разработка библиотеки классов.
* Отладка программных модулей.
* Модульное тестирование. Создание тестовых случаев.
* Инспекция кода на соответствие стандартам кодирования.
* Подготовка отчета и размещение результатов в репозитории контроля версий.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Стандарты Единой Системы Программной Документации:

ГОСТ 19.105-78 Общие требования к программным документам

ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

ГОСТ 19.301-78 Программа и методика испытаний. Требования к содержанию и оформлению

ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению

ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению

ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

1. Культин Н., C++ в задачах и примерах, 5-е издание, СПб: БХВ, 2020 — 656 с.
2. Моллер Й., Уэллингер С., C++: Полное руководство, 2-е издание, М: Вильямс, 2019 — 880 с.
3. Харрисон Л., Qt 5: Графика и создание интерфейсов, М: ДМК Пресс, 2018 — 560 с.
4. Документация по Qt [Электронный ресурс] — URL: https://doc.qt.io/qt-5/gettingstarted.html (дата обращения: 04.10.2024)

# Приложение А

(справочное)

**Скрипт БД**

Create table Users(

userID int identity(1,1) PRIMARY KEY,

fio VARCHAR(100),

phone VARCHAR(20),

login varchar(50) unique not null,

password varchar(50) not null,

roleID int,

foreign key (roleID) references Roles(roleID)

);

insert into Users (fio, phone, login, password, roleID) values

('Касаткин Егор Сергеевич','89114761098','login1','pass1',1),

('Белов Сергей Иванович','89116588913','login2','pass2',2),

('Наумов Владимир Владимирович','89113450908','login3','pass3',2),

('Герасимова Ульяна Андреевна','89116009141','login4','pass4',3),

('Ильин Юрий Петрович','89113098701','login5','pass5',3),

('Иванов Иван Иванович','89114670981','login11','pass11',4),

('Петров Петр Петрович','89116729812','login12','pass12',4),

('Сидоров Сергей Сидорович','89110987654','login13','pass13',2);

Create table Roles(

roleID int identity(1,1) primary key,

roleName VARCHAR(50) not null

);

insert into Roles (roleName) values

('Менеджер'),

('Автомеханик'),

('Оператор'),

('Заказчик');

create table Cars(

carID int identity(1,1) primary key,

carType varchar(50) not null,

carModel varchar(50) not null

);

insert into Cars (carType, carModel) values

('Легковая','Toyota Supra'),

('Легковая','Mitsubishi Lancer'),

('Легковая','Tesla Model 3'),

('Легковая','Tesla Model X'),

('Грузовая','УАЗ 2360');

create table Statuses(

statusID int identity(1,1) primary key,

statusName varchar (50) not null

);

insert into Statuses (statusName) values

('Новая заявка'),

('В процессе ремонта'),

('Готов к выдаче');

create table Requests (

requestsID int identity(1,1) primary key,

startDate date not null,

carID int,

problemDescription text not null,

statusID int,

complectionDate date,

masterID int,

clientID int,

foreign key (carID) references Cars(carID),

foreign key (statusID) references Statuses(statusID),

foreign key (masterID) references Users(userID),

foreign key (clientID) references Users(userID)

);

insert into Requests (startDate, carID, problemDescription, statusID, complectionDate, masterID, clientID) values

('2024-09-09',1,'Отказали тормоза.',2,NULL,2,6),

('2024-10-10',2,'Замена масла.',2,NULL,3,7),

('2024-11-11',3,'В салоне запах бензина.',3,'2024-11-12',3,8),

('2024-10-10',4,'Проблемно крутится руль',1,NULL,NULL,7),

('2024-11-11',5,'Проблемно крутится руль',1,NULL,NULL,8);

create table Comments(

commentID int identity(1,1) primary key,

message text not null,

masterID int,

requestID int,

foreign key (masterID) references Users(userID),

foreign key (requestID) references Requests(requestsID)

);

insert into Comments (message, masterID, requestID) values

('Очень странно.',2,1),

('Будем разбираться!',3,2),

('Будем разбираться!',3,3);

# Приложение Б

(справочное)

**Библиотека классов**

Файл «Class1.cs»:

using System;

namespace CarMechanicLibrary

{

public static class ServiceCalculator

{

public static double CalculateServiceCost(double laborCost, double partsCost, double markupPercentage)

{

double totalCost = laborCost + partsCost;

double markup = totalCost \* (markupPercentage / 100);

return totalCost + markup;

}

public static TimeSpan CalculateServiceTime(int numberOfTasks, TimeSpan timePerTask)

{

return TimeSpan.FromTicks(numberOfTasks \* timePerTask.Ticks);

}

public static double CalculateDiscountedPrice(double originalPrice, double discountPercentage)

{

return originalPrice - (originalPrice \* (discountPercentage / 100));

}

}}

# Приложение В

(справочное)

**Исходный код**

Файл «addRequests.cs»:

using System;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class addRequest : Form

{

private int clientID;

public addRequest(int clientID)

{

InitializeComponent();

LoadProblems();

this.clientID = clientID;

}

// Метод для загрузки списка проблем в ComboBox

private void LoadProblems()

{

comboBox1.Items.AddRange(new object[]

{

"Отказали тормоза", "В салоне пахнет бензином", "Руль плохо крутится",

"Двигатель стучит", "Не заводится", "Проблемы с электроникой",

"Проблемы с трансмиссией", "Шум в колесах", "Течет масло",

"Проблемы с подвеской"

});

}

// Метод для валидации введённых данных

private bool ValidateForm()

{

if (string.IsNullOrWhiteSpace(textBoxCarType.Text))

{

MessageBox.Show("Введите вид автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (string.IsNullOrWhiteSpace(textBoxCarModel.Text))

{

MessageBox.Show("Введите модель автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (comboBox1.SelectedIndex == -1)

{

MessageBox.Show("Выберите проблему автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

return true;

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close(); // Закрываем текущую форму

}

private void button1\_Click(object sender, EventArgs e)

{

if (ValidateForm())

{

string carType = textBoxCarType.Text.Trim();

string carModel = textBoxCarModel.Text.Trim();

string problem = comboBox1.SelectedItem.ToString();

DateTime startDate = DateTime.Now; // Дата создания заявки

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

// Подключаемся к базе данных и сохраняем заявку

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// Сначала проверим, существует ли уже автомобиль

int carID;

string carCheckQuery = "SELECT carID FROM Cars WHERE carType = @carType AND carModel = @carModel";

using (SqlCommand carCheckCmd = new SqlCommand(carCheckQuery, conn))

{

carCheckCmd.Parameters.AddWithValue("@carType", carType);

carCheckCmd.Parameters.AddWithValue("@carModel", carModel);

object result = carCheckCmd.ExecuteScalar();

if (result != null) // Если автомобиль уже существует

{

carID = (int)result;

}

else // Если нет, добавим новый автомобиль

{

string insertCarQuery = "INSERT INTO Cars (carType, carModel) OUTPUT INSERTED.carID VALUES (@carType, @carModel)";

using (SqlCommand insertCarCmd = new SqlCommand(insertCarQuery, conn))

{

insertCarCmd.Parameters.AddWithValue("@carType", carType);

insertCarCmd.Parameters.AddWithValue("@carModel", carModel);

carID = (int)insertCarCmd.ExecuteScalar();

}

}

}

string requestQuery = "INSERT INTO Requests (startDate, carID, problemDescription, statusID, clientID) " +

"VALUES (@startDate, @carID, @problemDescription, 1, @clientID)";

using (SqlCommand requestCmd = new SqlCommand(requestQuery, conn))

{

requestCmd.Parameters.AddWithValue("@startDate", startDate);

requestCmd.Parameters.AddWithValue("@carID", carID);

requestCmd.Parameters.AddWithValue("@problemDescription", problem);

requestCmd.Parameters.AddWithValue("@clientID", clientID); // Используем переданный clientID

try

{

requestCmd.ExecuteNonQuery();

MessageBox.Show("Заявка успешно создана и передана оператору.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Close(); // Закрываем форму после успешного создания заявки

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при создании заявки: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

// Метод для открытия формы просмотра заявок

public void OpenViewingForm()

{

// Открываем форму для просмотра, передавая clientID

showRequestClient viewingForm = new showRequestClient(clientID); // Передаем clientID

viewingForm.Show(); // Используем Show() вместо ShowDialog()

}

}

}

Файл «avtorizatia.cs»:

using System;

using System.Data.SqlClient;

using System.Drawing;

using System.Timers;

using System.Windows.Forms;

using CarMechanicLibrary;

namespace уп

{

public partial class avtorizatia : Form

{

public int attemptCount = 0;

public bool isBlocked = false;

private System.Timers.Timer blockTimer;

private int blockTime = 180000;

private string captchaValue;

private bool isPasswordVisible = false;

public avtorizatia()

{

InitializeComponent();

ServiceCalculator.CalculateDiscountedPrice(1000, 80);

pictureBoxCaptcha.Visible = false;

txtCaptcha.Visible = false;

updatePictureBox.Visible = false;

txtPassword.UseSystemPasswordChar = true;

pictureBox1.Click += new EventHandler(TogglePasswordVisibility);

blockTimer = new System.Timers.Timer(blockTime);

blockTimer.Elapsed += UnblockUser;

blockTimer.AutoReset = false;

}

public static class CaptchaGenerator

{

private static Random random = new Random();

public static string Generate()

{

const string chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

char[] captcha = new char[7];

for (int i = 0; i < captcha.Length; i++)

{

captcha[i] = chars[random.Next(chars.Length)];

}

return new string(captcha);

}

public static Bitmap RenderCaptchaImage(string captchaText, Size pictureBoxSize)

{

Bitmap bitmap = new Bitmap(pictureBoxSize.Width, pictureBoxSize.Height);

using (Graphics g = Graphics.FromImage(bitmap))

{

g.Clear(Color.Gray);

int fontSize = Math.Min(pictureBoxSize.Width / captchaText.Length, pictureBoxSize.Height / 2);

Font font = new Font("Arial", fontSize, FontStyle.Bold);

for (int i = 0; i < captchaText.Length; i++)

{

int xOffset = i \* fontSize + random.Next(-5, 5);

int yOffset = random.Next(0, pictureBoxSize.Height / 3);

PointF position = new PointF(xOffset, yOffset);

g.DrawString(captchaText[i].ToString(), font, Brushes.White, position);

}

for (int i = 0; i < 100; i++)

{

int x = random.Next(bitmap.Width);

int y = random.Next(bitmap.Height);

bitmap.SetPixel(x, y, Color.Black);

}

}

return bitmap;

}

}

public void GenerateCaptcha()

{

captchaValue = CaptchaGenerator.Generate();

pictureBoxCaptcha.Image = CaptchaGenerator.RenderCaptchaImage(captchaValue, pictureBoxCaptcha.Size);

}

private void button1\_Click(object sender, EventArgs e)

{

if (isBlocked)

{

MessageBox.Show("Вход заблокирован. Подождите 3 минуты.", "Блокировка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

string login = txtLogin.Text;

string password = txtPassword.Text;

if (attemptCount >= 1 && pictureBoxCaptcha.Visible)

{

if (txtCaptcha.Text != captchaValue)

{

attemptCount++;

LogLoginAttempt(login, false);

MessageBox.Show("Неправильная CAPTCHA. Попробуйте снова.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

if (attemptCount > 2)

{

BlockUser();

return;

}

return;

}

}

if (ValidateUser(login, password))

{

attemptCount = 0;

LogLoginAttempt(login, true);

}

else

{

attemptCount++;

LogLoginAttempt(login, false);

if (attemptCount == 1)

{

MessageBox.Show("Неверный логин или пароль. Попробуйте снова.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

ShowCaptcha();

}

else if (attemptCount >= 2)

{

MessageBox.Show("Неверный логин или пароль. Попробуйте снова.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

if (attemptCount > 2)

{

BlockUser();

}

}

}

public bool ValidateUser(string login, string password)

{

using (SqlConnection conn = new SqlConnection("Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True"))

{

conn.Open();

string query = @"

SELECT r.RoleName

FROM Users u

INNER JOIN Roles r ON u.RoleID = r.RoleID

WHERE u.Login = @login AND u.Password = @password";

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@login", login);

cmd.Parameters.AddWithValue("@password", password);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

string role = reader.GetString(0);

OpenUserForm(role);

return true;

}

else

{

return false;

}

}

}

private void LogLoginAttempt(string login, bool isSuccess)

{

using (SqlConnection conn = new SqlConnection("Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True"))

{

conn.Open();

string query = "INSERT INTO LoginHistory (Login, LoginTime, Success) VALUES (@login, @loginTime, @success)";

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@login", login);

cmd.Parameters.AddWithValue("@loginTime", DateTime.Now);

cmd.Parameters.AddWithValue("@success", isSuccess);

cmd.ExecuteNonQuery();

}

}

private void OpenUserForm(string role)

{

this.Hide();

pictureBoxCaptcha.Visible = false;

txtCaptcha.Visible = false;

updatePictureBox.Visible = false;

switch (role)

{

case "Заказчик":

int clientID = GetClientID();

clientForm clientForm = new clientForm(clientID);

clientForm.Show();

break;

case "Оператор":

showRequestAdmin adminForm = new showRequestAdmin();

adminForm.Show();

break;

case "Автомеханик":

int mechanicID = GetMechanicID();

requestFormMaster masterForm = new requestFormMaster(mechanicID);

masterForm.Show();

break;

case "Менедежр":

managerForm managerForm = new managerForm();

managerForm.Show();

break;

default:

MessageBox.Show("Неизвестная роль пользователя.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

this.Show();

break;

}

}

private int GetClientID()

{

using (SqlConnection conn = new SqlConnection("Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True"))

{

conn.Open();

string query = "SELECT UserID FROM Users WHERE Login = @login";

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@login", txtLogin.Text);

object result = cmd.ExecuteScalar();

if (result != null)

{

return Convert.ToInt32(result);

}

else

{

throw new Exception("Не удалось получить clientID");

}

}

}

private int GetMechanicID()

{

using (SqlConnection conn = new SqlConnection("Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True"))

{

conn.Open();

string query = "SELECT UserID FROM Users WHERE Login = @login";

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@login", txtLogin.Text);

object result = cmd.ExecuteScalar();

if (result != null)

{

return Convert.ToInt32(result);

}

else

{

throw new Exception("Не удалось получить идентификатор автомеханика");

}

}

}

private void ShowCaptcha()

{

pictureBoxCaptcha.Visible = true;

txtCaptcha.Visible = true;

updatePictureBox.Visible = true;

GenerateCaptcha();

}

private void BlockUser()

{

isBlocked = true;

MessageBox.Show("Доступ заблокирован на 3 минуты из-за слишком большого количества неудачных попыток.", "Блокировка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

blockTimer.Start();

}

public void UnblockUser(Object source, ElapsedEventArgs e)

{

isBlocked = false;

attemptCount = 0;

blockTimer.Stop();

}

public void TogglePasswordVisibility(object sender, EventArgs e)

{

txtPassword.UseSystemPasswordChar = !isPasswordVisible;

isPasswordVisible = !isPasswordVisible;

}

public void TogglePasswordVisibility()

{

txtPassword.UseSystemPasswordChar = !isPasswordVisible;

isPasswordVisible = !isPasswordVisible;

}

private void updatePictureBox\_Click(object sender, EventArgs e)

{

GenerateCaptcha();

}

private void button2\_Click(object sender, EventArgs e)

{

historyLoginForm historyForm = new historyLoginForm();

historyForm.Show();

}

}

}

Файл «changeStatus»:

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class changeStatus : Form

{

private SqlConnection connection;

private int requestId; // ID заявки, которую мы редактируем

public event EventHandler StatusChanged; // Событие для уведомления об успешном изменении статуса

public changeStatus(int requestId)

{

InitializeComponent();

connection = new SqlConnection(@"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=true;"); // Замените на свою строку подключения

this.requestId = requestId;

LoadStatuses(); // Загружаем статусы в comboBox1

}

private void LoadStatuses()

{

try

{

connection.Open();

string query = "SELECT statusID, statusName FROM Statuses";

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

DataTable statusesTable = new DataTable();

adapter.Fill(statusesTable);

comboBox1.DataSource = statusesTable;

comboBox1.DisplayMember = "statusName"; // Отображаем название статуса

comboBox1.ValueMember = "statusID"; // Сохраняем statusID статусов

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке статусов: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedValue != null)

{

int selectedStatusId = (int)comboBox1.SelectedValue;

try

{

connection.Open();

// Обновляем заявку с новым статусом

string query = "UPDATE Requests SET statusID = @statusID WHERE requestID = @requestID";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@statusID", selectedStatusId);

command.Parameters.AddWithValue("@requestID", requestId);

command.ExecuteNonQuery();

}

MessageBox.Show("Статус заявки успешно изменен!");

StatusChanged?.Invoke(this, EventArgs.Empty); // Уведомляем об успешном изменении статуса

this.Close(); // Закрываем форму изменения статуса

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}");

}

finally

{

connection.Close();

}

}

else

{

MessageBox.Show("Пожалуйста, выберите статус.");

}

}

}

}

Файл «clientForm»:

using QRCoder;

using System;

using System.Drawing;

using System.Windows.Forms;

namespace уп

{

public partial class clientForm : Form

{

private int clientID;

public clientForm(int clientID)

{

InitializeComponent();

this.clientID = clientID;

}

private void button1\_Click(object sender, EventArgs e)

{

addRequest addRequestForm = new addRequest(clientID);

addRequestForm.Show();

}

private void button2\_Click(object sender, EventArgs e)

{

showRequestClient showRequestClientForm = new showRequestClient(clientID);

showRequestClientForm.Show();

this.Close();

}

private void button3\_Click(object sender, EventArgs e)

{

avtorizatia avtorizatiaForm = new avtorizatia();

avtorizatiaForm.Show();

this.Close();

}

private void GenerateQRCode(string url)

{

QRCodeGenerator qrGenerator = new QRCodeGenerator();

QRCodeData qrCodeData = qrGenerator.CreateQrCode(url, QRCodeGenerator.ECCLevel.Q);

QRCode qrCode = new QRCode(qrCodeData);

int qrSize = Math.Min(pictureBox1.Width, pictureBox1.Height);

Bitmap qrCodeImage = qrCode.GetGraphic(20);

Bitmap resizedQRCode = new Bitmap(qrCodeImage, new Size(pictureBox1.Width, pictureBox1.Height));

pictureBox1.Image = resizedQRCode;

}

private void button4\_Click\_1(object sender, EventArgs e)

{

string url = "https://docs.google.com/forms/d/e/1FAIpQLScJWRRmSbUQdjT7-4LrYjtTCt-kIilPvcoe35lytzqLzN3EkQ/viewform?usp=header";

GenerateQRCode(url);

}

}

}

Файл «historyLoginForm»:

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class historyLoginForm : Form

{

public historyLoginForm()

{

InitializeComponent();

button1.Click += button1\_Click;

button2.Click += button2\_Click;

}

private void LoadLoginHistory(string filter = "")

{

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

string query = "SELECT Login, LoginTime, Success FROM LoginHistory";

if (!string.IsNullOrEmpty(filter))

{

query += " WHERE Login LIKE @Filter";

}

using (SqlConnection conn = new SqlConnection(connectionString))

using (SqlCommand cmd = new SqlCommand(query, conn))

{

if (!string.IsNullOrEmpty(filter))

{

cmd.Parameters.AddWithValue("@Filter", "%" + filter + "%");

}

try

{

conn.Open();

SqlDataAdapter da = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

da.Fill(dt);

dataGridView1.DataSource = dt;

dataGridView1.Columns[0].HeaderText = "Логин";

dataGridView1.Columns[1].HeaderText = "Время входа";

dataGridView1.Columns[2].HeaderText = "Успешно";

foreach (DataGridViewColumn column in dataGridView1.Columns)

{

column.AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;

}

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при загрузке истории входов: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

string filter = textBox1.Text.Trim();

LoadLoginHistory(filter);

}

private void button2\_Click(object sender, EventArgs e)

{

textBox1.Clear();

LoadLoginHistory();

}

private void historyLoginForm\_Load(object sender, EventArgs e)

{

LoadLoginHistory();

}

private void backButton\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

Файл «managerForm»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Microsoft.AspNet.SignalR.Infrastructure;

namespace уп

{

public partial class managerForm : Form

{

private SqlConnection connection;

private DataTable requestsTable;

private int totalRecords;

private string connectionString = @"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=True";

private const int InProgressStatusId = 2;

public managerForm()

{

InitializeComponent();

LoadRequests();

LoadMechanics();

}

private void LoadRequests()

{

try

{

connection = new SqlConnection(connectionString);

connection.Open();

string query = @"

SELECT

r.requestID,

r.startDate,

c.carID,

r.problemDescription,

s.statusName,

r.complectionDate,

u.userID AS masterID,

u.fio AS masterName,

uc.fio AS clientName

FROM Requests r

LEFT JOIN Cars c ON r.carID = c.carID

LEFT JOIN Statuses s ON r.statusID = s.statusID

LEFT JOIN Users u ON r.masterID = u.userID

LEFT JOIN Users uc ON r.clientID = uc.userID";

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

requestsTable = new DataTable();

adapter.Fill(requestsTable);

dataGridView1.DataSource = requestsTable;

dataGridView1.Columns[0].HeaderText = "Номер заявки";

dataGridView1.Columns[1].HeaderText = "Дата начала";

dataGridView1.Columns[2].HeaderText = "Номер автомобиля";

dataGridView1.Columns[3].HeaderText = "Проблема";

dataGridView1.Columns[4].HeaderText = "Статус заявки";

dataGridView1.Columns[5].HeaderText = "Дата завершения";

dataGridView1.Columns[6].HeaderText = "Номер мастера";

dataGridView1.Columns[7].HeaderText = "ФИО Мастера";

dataGridView1.Columns[8].HeaderText = "ФИО Клиента";

totalRecords = requestsTable.Rows.Count;

UpdateRecordCount();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке заявок: {ex.Message}");

}

finally

{

if (connection != null && connection.State == ConnectionState.Open)

{

connection.Close();

}

}

}

private void LoadMechanics()

{

string query = @"SELECT userID, fio FROM Users WHERE roleID = (SELECT roleID FROM Roles WHERE roleID = 2)";

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

SqlDataAdapter da = new SqlDataAdapter(query, conn);

DataTable dt = new DataTable();

da.Fill(dt);

comboBox1.DisplayMember = "fio";

comboBox1.ValueMember = "userID";

comboBox1.DataSource = dt;

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0 && comboBox1.SelectedValue != null)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

int selectedMasterId = Convert.ToInt32(comboBox1.SelectedValue);

AssignMechanicToRequest(selectedRequestId, selectedMasterId);

MessageBox.Show("Механик успешно назначен на заявку!");

LoadRequests();

}

else

{

MessageBox.Show("Выберите заявку и автомеханика.");

}

}

private void AssignMechanicToRequest(int requestId, int mechanicId)

{

string query = @"UPDATE Requests

SET masterID = @masterID

WHERE requestID = @requestID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

using (SqlCommand cmd = new SqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@masterID", mechanicId);

cmd.Parameters.AddWithValue("@requestID", requestId);

cmd.ExecuteNonQuery();

}

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

DateTime selectedDate = dateTimePicker1.Value;

UpdateRequestComplectionDateAndStatus(selectedRequestId, selectedDate);

MessageBox.Show("Дата завершения заявки успешно обновлена и статус изменен на 'В процессе'!");

LoadRequests();

}

else

{

MessageBox.Show("Выберите заявку.");

}

}

private void UpdateRequestComplectionDateAndStatus(int requestId, DateTime complectionDate)

{

string query = @"UPDATE Requests

SET complectionDate = @complectionDate, statusID = @statusID

WHERE requestID = @requestID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

using (SqlCommand cmd = new SqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@complectionDate", complectionDate);

cmd.Parameters.AddWithValue("@statusID", InProgressStatusId);

cmd.Parameters.AddWithValue("@requestID", requestId);

cmd.ExecuteNonQuery();

}

}

}

private void UpdateRecordCount()

{

int filteredRecords = (dataGridView1.DataSource as DataTable)?.DefaultView.Count ?? 0;

label2.Text = $"{filteredRecords} из {totalRecords}";

}

}

}

Файл «processingRequests»:

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class processingRequests : Form

{

private SqlConnection connection;

private int requestId;

public event EventHandler RequestAssigned;

public processingRequests(int requestId)

{

InitializeComponent();

connection = new SqlConnection(@"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=true;");

this.requestId = requestId;

LoadMechanics();

}

private void LoadMechanics()

{

try

{

connection.Open();

string query = "SELECT userID, fio FROM Users WHERE roleID = (SELECT roleID FROM Roles WHERE roleID = 2)";

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

DataTable mechanicsTable = new DataTable();

adapter.Fill(mechanicsTable);

if (mechanicsTable.Rows.Count > 0)

{

comboBox1.DataSource = mechanicsTable;

comboBox1.DisplayMember = "fio";

comboBox1.ValueMember = "userID";

}

else

{

MessageBox.Show("Нет доступных механиков для назначения.");

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке механиков: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedValue != null)

{

int selectedMechanicId = (int)comboBox1.SelectedValue;

try

{

connection.Open();

string query = "UPDATE Requests SET masterID = @masterID WHERE requestID = @requestID";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@masterID", selectedMechanicId);

command.Parameters.AddWithValue("@requestID", requestId);

command.ExecuteNonQuery();

}

MessageBox.Show("Механик успешно назначен!");

RequestAssigned?.Invoke(this, EventArgs.Empty);

this.Close();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}");

}

finally

{

connection.Close();

}

}

else

{

MessageBox.Show("Пожалуйста, выберите механика.");

}

}

}

}

Файл «refreshRequests»:

using System;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class refreshRequest : Form

{

private int requestID;

private int clientID;

private showRequestClient parentForm;

public refreshRequest(int requestID, int clientID, showRequestClient parentForm)

{

InitializeComponent();

this.requestID = requestID;

this.clientID = clientID;

this.parentForm = parentForm;

LoadProblems();

LoadRequestData();

}

private void LoadProblems()

{

comboBox1.Items.AddRange(new object[]

{

"Отказали тормоза", "В салоне пахнет бензином", "Руль плохо крутится",

"Двигатель стучит", "Не заводится", "Проблемы с электроникой",

"Проблемы с трансмиссией", "Шум в колесах", "Течет масло",

"Проблемы с подвеской"

});

}

private void LoadRequestData()

{

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

string query = @"SELECT Cars.carType, Cars.carModel, Requests.problemDescription, Users.fio, Users.phone

FROM Requests

JOIN Cars ON Requests.carID = Cars.carID

JOIN Users ON Requests.clientID = Users.userID -- Обратите внимание на правильное имя столбца

WHERE Requests.requestID = @requestID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

using (SqlCommand cmd = new SqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@requestID", requestID);

conn.Open();

using (SqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

textBox1.Text = reader["carType"].ToString();

textBox2.Text = reader["carModel"].ToString();

comboBox1.SelectedItem = reader["problemDescription"].ToString();

}

}

}

}

}

private bool ValidateForm()

{

if (string.IsNullOrWhiteSpace(textBox1.Text))

{

MessageBox.Show("Введите вид автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (string.IsNullOrWhiteSpace(textBox2.Text))

{

MessageBox.Show("Введите модель автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (comboBox1.SelectedIndex == -1)

{

MessageBox.Show("Выберите проблему автомобиля.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

return true;

}

private void UpdateRequest()

{

if (ValidateForm())

{

string carType = textBox1.Text.Trim();

string carModel = textBox2.Text.Trim();

string problem = comboBox1.SelectedItem.ToString();

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

string updateCarQuery = "UPDATE Cars SET carType = @carType, carModel = @carModel WHERE carID = (SELECT carID FROM Requests WHERE requestID = @requestID)";

using (SqlCommand updateCarCmd = new SqlCommand(updateCarQuery, conn))

{

updateCarCmd.Parameters.AddWithValue("@carType", carType);

updateCarCmd.Parameters.AddWithValue("@carModel", carModel);

updateCarCmd.Parameters.AddWithValue("@requestID", requestID);

updateCarCmd.ExecuteNonQuery();

}

string updateRequestQuery = "UPDATE Requests SET problemDescription = @problemDescription WHERE requestID = @requestID";

using (SqlCommand updateRequestCmd = new SqlCommand(updateRequestQuery, conn))

{

updateRequestCmd.Parameters.AddWithValue("@problemDescription", problem);

updateRequestCmd.Parameters.AddWithValue("@requestID", requestID);

updateRequestCmd.ExecuteNonQuery();

}

MessageBox.Show("Заявка успешно обновлена.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

parentForm.RefreshDataGridView();

this.Close();

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

UpdateRequest();

}

}

}

Файл «requestFormMaster»:

using System;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Windows.Forms;

using System.IO;

namespace уп

{

public partial class requestFormMaster : Form

{

private SqlConnection connection;

private DataTable requestsTable;

private int mechanicID;

private int totalRecords;

// Добавляем параметр mechanicID в конструктор

public requestFormMaster(int loggedInMechanicID)

{

InitializeComponent();

connection = new SqlConnection(@"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=true;");

mechanicID = loggedInMechanicID;

LoadRequests();

}

private void LoadRequests()

{

try

{

connection.Open();

string query = @"

SELECT

r.requestID,

r.startDate,

c.carID,

r.problemDescription,

s.statusName,

r.complectionDate,

u.fio AS clientName

FROM Requests r

LEFT JOIN Cars c ON r.carID = c.carID

LEFT JOIN Statuses s ON r.statusID = s.statusID

LEFT JOIN Users u ON r.clientID = u.userID

WHERE r.masterID = @mechanicID";

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

adapter.SelectCommand.Parameters.AddWithValue("@mechanicID", mechanicID);

requestsTable = new DataTable();

adapter.Fill(requestsTable);

dataGridView1.DataSource = requestsTable;

dataGridView1.Columns[0].HeaderText = "Номер заявки";

dataGridView1.Columns[1].HeaderText = "Дата начала";

dataGridView1.Columns[2].HeaderText = "Номер автомобиля";

dataGridView1.Columns[3].HeaderText = "Проблема";

dataGridView1.Columns[4].HeaderText = "Статус заявки";

dataGridView1.Columns[5].HeaderText = "Дата завершения";

dataGridView1.Columns[6].HeaderText = "ФИО Клиента";

totalRecords = requestsTable.Rows.Count;

UpdateAverageComplectionTime();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке заявок: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void UpdateAverageComplectionTime()

{

var completedRows = requestsTable.Select("complectionDate IS NOT NULL");

if (completedRows.Length > 0)

{

TimeSpan totalDuration = TimeSpan.Zero;

foreach (DataRow row in completedRows)

{

DateTime startDate = Convert.ToDateTime(row["startDate"]);

DateTime complectionDate = Convert.ToDateTime(row["complectionDate"]);

totalDuration += (complectionDate - startDate);

}

double avgDuration = totalDuration.TotalDays / completedRows.Length;

label2.Text = $"Среднее время выполнения: {avgDuration:F1} дней";

}

else

{

label2.Text = "Среднее время выполнения: нет завершенных заявок";

}

}

private void button3\_Click(object sender, EventArgs e)

{

avtorizatia avtorizatiaForm = new avtorizatia();

avtorizatiaForm.Show();

this.Close();

}

private void button4\_Click(object sender, EventArgs e)

{

string filter = $"clientName LIKE '%{textBox1.Text}%' OR " +

$"problemDescription LIKE '%{textBox1.Text}%' OR " +

$"statusName LIKE '%{textBox1.Text}%'";

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = filter;

}

private void button5\_Click(object sender, EventArgs e)

{

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

textBox1.Clear();

LoadRequests();

}

private void dataGridView1\_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)

{

string columnName = dataGridView1.Columns[e.ColumnIndex].Name;

if (dataGridView1.SortOrder == System.Windows.Forms.SortOrder.Ascending)

{

dataGridView1.Sort(dataGridView1.Columns[columnName], ListSortDirection.Descending);

}

else

{

dataGridView1.Sort(dataGridView1.Columns[columnName], ListSortDirection.Ascending);

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

CompleteRequest(selectedRequestId);

LoadRequests();

}

else

{

MessageBox.Show("Пожалуйста, выберите заявку для завершения.");

}

}

private void CompleteRequest(int requestId)

{

try

{

connection.Open();

string query = @"

UPDATE Requests

SET statusID = (SELECT statusID FROM Statuses WHERE statusName = 'Готова к выдаче'),

complectionDate = @complectionDate

WHERE requestID = @requestID";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@complectionDate", DateTime.Now);

command.Parameters.AddWithValue("@requestID", requestId);

command.ExecuteNonQuery();

}

MessageBox.Show("Заявка успешно завершена!");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при завершении заявки: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

var orderData = GetOrderData(selectedRequestId);

string report = GenerateReport(selectedRequestId, orderData);

SaveReportToFile(report);

MessageBox.Show("Отчет успешно создан!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Пожалуйста, выберите заявку для создания отчета.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

private (string ClientName, string OrderDescription, DateTime StartTime, DateTime EndTime) GetOrderData(int orderId)

{

using (SqlConnection conn = new SqlConnection(@"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=True"))

{

conn.Open();

string query = @"

SELECT c.fio AS ClientName, r.problemDescription, r.startDate, r.complectionDate

FROM Requests r

INNER JOIN Users c ON r.clientID = c.userID

WHERE r.requestID = @orderId";

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@orderId", orderId);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

string clientName = reader.GetString(0);

string description = reader.GetString(1);

DateTime startTime = reader.GetDateTime(2);

DateTime endTime = reader.IsDBNull(3) ? DateTime.Now : reader.GetDateTime(3);

return (clientName, description, startTime, endTime);

}

else

{

throw new Exception("Заявка не найдена.");

}

}

}

private string GenerateReport(int requestId, (string ClientName, string OrderDescription, DateTime StartTime, DateTime EndTime) orderData)

{

TimeSpan duration = orderData.EndTime - orderData.StartTime;

double durationInDays = duration.TotalDays;

string report = $"Отчет по заявке №{requestId}\n\n" +

$"Клиент: {orderData.ClientName}\n" +

$"Описание заказа: {orderData.OrderDescription}\n" +

$"Время начала выполнения: {orderData.StartTime}\n" +

$"Время завершения выполнения: {orderData.EndTime}\n" +

$"Общее время выполнения: {durationInDays:F1} дней\n";

return report;

}

private void SaveReportToFile(string report)

{

using (SaveFileDialog saveFileDialog = new SaveFileDialog())

{

saveFileDialog.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

saveFileDialog.DefaultExt = "txt";

saveFileDialog.AddExtension = true;

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(saveFileDialog.FileName, report);

}

}

}

}

}

Файл «showRequestAdmin»:

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class showRequestAdmin : Form

{

private SqlConnection connection;

private DataTable requestsTable;

private int totalRecords;

public showRequestAdmin()

{

InitializeComponent();

connection = new SqlConnection(@"Server=DESKTOP-Q8GSFE6\SQLEXPRESS;Database=CarMechanic;Integrated Security=true;");

LoadRequests();

}

private void LoadRequests()

{

try

{

connection.Open();

string query = @"

SELECT

r.requestID,

r.startDate,

c.carID,

r.problemDescription,

s.statusName,

r.complectionDate,

u.userID AS masterID,

u.fio AS masterName,

uc.fio AS clientName

FROM Requests r

LEFT JOIN Cars c ON r.carID = c.carID

LEFT JOIN Statuses s ON r.statusID = s.statusID

LEFT JOIN Users u ON r.masterID = u.userID

LEFT JOIN Users uc ON r.clientID = uc.userID";

SqlDataAdapter adapter = new SqlDataAdapter(query, connection);

requestsTable = new DataTable();

adapter.Fill(requestsTable);

dataGridView1.DataSource = requestsTable;

dataGridView1.Columns[0].HeaderText = "Номер заявки";

dataGridView1.Columns[1].HeaderText = "Дата начала";

dataGridView1.Columns[2].HeaderText = "Номер автомобиля";

dataGridView1.Columns[3].HeaderText = "Проблема";

dataGridView1.Columns[4].HeaderText = "Статус заявки";

dataGridView1.Columns[5].HeaderText = "Дата завершения";

dataGridView1.Columns[6].HeaderText = "Номер мастера";

dataGridView1.Columns[7].HeaderText = "ФИО Мастера";

dataGridView1.Columns[8].HeaderText = "ФИО Клиента";

totalRecords = requestsTable.Rows.Count;

UpdateRecordCount();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке заявок: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void UpdateRecordCount()

{

int filteredRecords = (dataGridView1.DataSource as DataTable)?.DefaultView.Count ?? 0;

label2.Text = $"{filteredRecords} из {totalRecords}";

}

private void button1\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

processingRequests processingRequestsForm = new processingRequests(selectedRequestId);

processingRequestsForm.RequestAssigned += (s, args) =>

{

UpdateRequestStatusToInProgress(selectedRequestId);

LoadRequests();

};

processingRequestsForm.ShowDialog();

}

else

{

MessageBox.Show("Пожалуйста, выберите заявку для редактирования.");

}

}

private void UpdateRequestStatusToInProgress(int requestId)

{

try

{

connection.Open();

string query = "UPDATE Requests SET statusID = (SELECT statusID FROM Statuses WHERE statusName = 'В процессе ремонта') WHERE requestID = @requestID";

using (SqlCommand command = new SqlCommand(query, connection))

{

command.Parameters.AddWithValue("@requestID", requestId);

command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при обновлении статуса заявки: {ex.Message}");

}

finally

{

connection.Close();

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestId = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

changeStatus changeStatusForm = new changeStatus(selectedRequestId);

changeStatusForm.ShowDialog();

LoadRequests();

}

else

{

MessageBox.Show("Пожалуйста, выберите заявку для изменения статуса.");

}

}

private void button3\_Click(object sender, EventArgs e)

{

string filter = $"masterName LIKE '%{textBox1.Text}%' OR " +

$"clientName LIKE '%{textBox1.Text}%' OR " +

$"problemDescription LIKE '%{textBox1.Text}%' OR " +

$"statusName LIKE '%{textBox1.Text}%'";

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = filter;

UpdateRecordCount();

}

private void button4\_Click(object sender, EventArgs e)

{

(dataGridView1.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

LoadRequests();

textBox1.Clear();

}

private void dataGridView1\_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)

{

string columnName = dataGridView1.Columns[e.ColumnIndex].Name;

if (dataGridView1.SortOrder == System.Windows.Forms.SortOrder.Ascending)

{

dataGridView1.Sort(dataGridView1.Columns[columnName], System.ComponentModel.ListSortDirection.Descending);

}

else

{

dataGridView1.Sort(dataGridView1.Columns[columnName], System.ComponentModel.ListSortDirection.Ascending);

}

}

private void button5\_Click(object sender, EventArgs e)

{

avtorizatia avtorizatiaForm = new avtorizatia();

avtorizatiaForm.Show();

this.Close();

}

}

}

Файл «showRequestClient»:

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace уп

{

public partial class showRequestClient : Form

{

private int clientID;

public showRequestClient(int clientID)

{

InitializeComponent();

this.clientID = clientID;

RefreshDataGridView();

}

private void showRequestClient\_Load(object sender, EventArgs e)

{

RefreshDataGridView();

}

public void OpenAddRequestForm()

{

addRequest addRequestForm = new addRequest(clientID);

addRequestForm.FormClosed += (s, args) => RefreshDataGridView();

addRequestForm.ShowDialog();

}

public void RefreshDataGridView()

{

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

string query = "SELECT r.requestID, r.startDate, c.carType, c.carModel, r.problemDescription, s.statusName " +

"FROM Requests r " +

"JOIN Cars c ON r.carID = c.carID " +

"JOIN Statuses s ON r.statusID = s.statusID " +

"WHERE r.clientID = @clientID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

using (SqlCommand cmd = new SqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@clientID", clientID);

try

{

conn.Open();

DataTable dataTable = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

adapter.Fill(dataTable);

dataGridView1.DataSource = dataTable;

ResizeColumns();

dataGridView1.Columns[0].HeaderText = "Номер заявки";

dataGridView1.Columns[1].HeaderText = "Дата начала";

dataGridView1.Columns[2].HeaderText = "Тип автомобиля";

dataGridView1.Columns[3].HeaderText = "Модель автомобиля";

dataGridView1.Columns[4].HeaderText = "Проблема";

dataGridView1.Columns[5].HeaderText = "Статус заявки";

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при загрузке заявок: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void ResizeColumns()

{

dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

}

private void button1\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestID = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

refreshRequest refreshForm = new refreshRequest(selectedRequestID, clientID, this);

refreshForm.FormClosed += (s, args) =>

{

RefreshDataGridView();

MessageBox.Show("Заявка успешно обновлена.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

};

refreshForm.ShowDialog();

}

else

{

MessageBox.Show("Выберите заявку для редактирования.", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count > 0)

{

int selectedRequestID = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells["requestID"].Value);

DialogResult dialogResult = MessageBox.Show("Вы уверены, что хотите удалить эту заявку?", "Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (dialogResult == DialogResult.Yes)

{

DeleteRequest(selectedRequestID);

RefreshDataGridView();

}

}

else

{

MessageBox.Show("Выберите заявку для удаления.", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

private void DeleteRequest(int requestID)

{

string connectionString = "Server=DESKTOP-Q8GSFE6\\SQLEXPRESS;Database=CarMechanic;Integrated Security=True;";

string query = "DELETE FROM Requests WHERE requestID = @requestID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

using (SqlCommand cmd = new SqlCommand(query, conn))

{

cmd.Parameters.AddWithValue("@requestID", requestID);

try

{

conn.Open();

cmd.ExecuteNonQuery();

MessageBox.Show("Заявка успешно удалена.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при удалении заявки: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void button3\_Click(object sender, EventArgs e)

{

this.Close();

clientForm clientForm = new clientForm(clientID);

clientForm.Show();

}

}

}

# Приложение Г

(справочное)

**Данные тестирования**

В таблице Г.1 представлен заголовок тестовых сценариев.

Таблица Г.1 – Заголовок тестовых сценариев

|  |  |
| --- | --- |
| **Название проекта** | Информационная система управления заявками ремонта автомобилей |
| **Рабочая версия** | 0.1 |
| **Имя тестирующего** | Мирошниченко Данил Егорович |
| **Дата(ы) теста** | 17.12.2024 |

В таблицах Г.2 – Г.6 представлены тестовые сценарии.

Таблица Г.2 – Тест кейс 1

|  |  |
| --- | --- |
| **Тест кейс #** | 1 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка авторизации с корректными данными |
| **Краткое изложение теста** | Тест проверяет успешный вход пользователя в систему с корректными логином и паролем |
| **Этапы теста** | 1. Запустить приложение 2. Ввести корректные логин и пароль 3. Нажать кнопку "Войти" |
| **Тестовые данные** | Логин: login3  Пароль: pass3 |
| **Ожидаемый результат** | Появляется сообщение об успешной авторизации и открывается окно автомеханика. |
| **Фактический результат** | Появляется сообщение об успешной авторизации и открывается окно автомеханика. |
| **Статус** | Зачет |
| **Предварительное условие** | Логин и пароль пользователя внесены в базу данных |

Продолжение таблицы Г.2

|  |  |
| --- | --- |
| **Постусловие** | Пользователю доступен функционал роли автомеханика |
| **Примечания/комментарии** | Нет |

Таблица Г.3 – Тест кейс 2

|  |  |
| --- | --- |
| **Тест кейс #** | 2 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка авторизации с некорректными данными |
| **Краткое изложение теста** | Тест проверяет реакцию системы при вводе неверного логина или пароля |
| **Этапы теста** | 1. Запустить приложение 2. Ввести некорректный логин или пароль 3. Нажать кнопку "Войти" |
| **Тестовые данные** | Логин: login3  Пароль: 123 |
| **Ожидаемый результат** | Появляется сообщение об ошибке "Неверный логин или пароль" и капча. |
| **Фактический результат** | Появляется сообщение об ошибке "Неверный логин или пароль" и капча. |
| **Статус** | Зачет |
| **Предварительное условие** | Логин и пароль пользователя не внесены в базу данных |
| **Постусловие** | Авторизация не выполнена, сообщение об ошибке показано |
| **Примечания/комментарии** | Нет |

Таблица Г.4 – Тест кейс 3

|  |  |
| --- | --- |
| **Тест кейс #** | 3 |
| **Приоритет тестирования** | Средний |
| **Заголовок/название теста** | Проверка авторизации после блокировки аккаунта |

Продолжение таблицы Г.4

|  |  |
| --- | --- |
| **Краткое изложение теста** | Тест проверяет возможность входа после блокировки пользователя из-за многократных неудачных попыток авторизации |
| **Этапы теста** | 1. Запустить приложение 2. Ввести данные 3. Нажать кнопку "Войти" 4. Проверить блокировку аккаунта |
| **Тестовые данные** | Логин: login3  Пароль: 123  Верная Captcha |
| **Ожидаемый результат** | Появляется сообщение об ошибке "Неверный логин или пароль" и сообщение о блокировки системы на 3 минуты. |
| **Фактический результат** | Появляется сообщение об ошибке "Неверный логин или пароль" и сообщение о блокировки системы на 3 минуты. |
| **Статус** | Зачет |
| **Предварительное условие** | Логин и пароль пользователя не внесены в базу данных, 2 неудачные попытки входа |
| **Постусловие** |  |
| **Примечания/комментарии** | Нет |

Таблица Г.5 – Тест кейс 4

|  |  |
| --- | --- |
| **Тест кейс #** | 4 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка функциональности создания заявки |
| **Краткое изложение теста** | Тест проверяет корректность работы формы создания заявки |
| **Этапы теста** | 1. Открыть вкладку «Добавление заявки» 2. Заполнить данные заявки 3. Нажать кнопку «Создать» |
| **Тестовые данные** | Вид автомобиля: Легковой  Модель автомобиля: Mazda 3  Проблема: Замена масла. |

Продолжение таблицы Г.5

|  |  |
| --- | --- |
| **Ожидаемый результат** | Заявка успешно создана, сообщение о добавлении заявки, обновление списка заявок |
| **Фактический результат** | Заявка успешно создана, сообщение о добавлении заявки, обновление списка заявок |
| **Статус** | Зачет |
| **Предварительное условие** | Вход был осуществлен под пользователем “Заказчик” |
| **Постусловие** | Заявка будет доступна для работы другим пользователям |
| **Примечания/комментарии** | Нет |

Таблица Г.6 – Тест кейс 5

|  |  |
| --- | --- |
| **Тест кейс #** | 5 |
| **Приоритет тестирования** | Средний |
| **Заголовок/название теста** | Проверка удаления заявки заказчиком |
| **Краткое изложение теста** | Тест проверяет возможность удаления заявки заказчиком |
| **Этапы теста** | 1. Открыть список заявок 2. Выбрать заявку 3. Нажать кнопку "Удалить" |
| **Тестовые данные** | ID заявки 17 |
| **Ожидаемый результат** | Заявка успешно удалена, сообщение о завершении удаления, обновление списка заявок |
| **Фактический результат** | Заявка успешно удалена, сообщение о завершении удаления, обновление списка заявок |
| **Статус** | Зачет |
| **Предварительное условие** | Вход в систему под ролью заказчика, в системе должна присутствовать заявка с ID 17 |
| **Постусловие** |  |
| **Примечания/комментарии** | Заявка удалена |

# Приложение Д

(справочное)

**Код unit-тестов**

Файл «UnitTest1»:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using уп;

namespace UnitTestProject2

{

[TestClass]

public class UnitTest1

{

private avtorizatia form;

[TestInitialize]

public void Setup()

{

form = new avtorizatia();

}

[TestMethod]

public void Test\_InvalidPassword\_ShouldFailLogin()

{

string login = "login1";

string invalidPassword = "wrongpass";

bool result = form.ValidateUser(login, invalidPassword);

Assert.IsFalse(result, "Вход должен завершиться неудачей с неверным паролем.");

}

[TestMethod]

public void Test\_InvalidLogin\_ShouldFailLogin()

{

string invalidLogin = "invalidLogin";

string password = "pass1";

bool result = form.ValidateUser(invalidLogin, password);

Assert.IsFalse(result, "Вход должен завершиться неудачей с неверным логином.");

}

[TestMethod]

public void Test\_ValidUser()

{

string login = "login1";

string password = "pass1";

bool result = form.ValidateUser(login, password);

string userRole = "Менеджер";

Assert.IsTrue(result, "Пользователь должен успешно войти.");

Assert.AreEqual("Менеджер", userRole, "Роль должна быть 'Менеджер'.");

}

[TestMethod]

public void Test\_GenerateCaptcha()

{

form.GenerateCaptcha();

Assert.IsNotNull(form.pictureBoxCaptcha.Image, "Изображение CAPTCHA должно быть сгенерировано и не быть нулевым после вызова GenerateCaptcha.");

}

[TestMethod]

public void Test\_PasswordVisibilityToggle()

{

bool initialVisibility = form.txtPassword.UseSystemPasswordChar;

form.TogglePasswordVisibility();

Assert.AreNotEqual(initialVisibility, !form.txtPassword.UseSystemPasswordChar, "Видимость пароля должна переключаться.");

}

}

}

**АТТЕСТАЦИОННЫЙ ЛИСТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ УП.02.01 (ПО ПРОФИЛЮ СПЕЦИАЛЬНОСТИ)**

по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей»

(код и наименование)

Специальность09.02.07 Информационные системы и программирование

(код и наименование специальности)

Студент(ка) 4 курса 42919/3 группы

Мирошниченко Данил Егорович

(Фамилия, имя, отчество)

Место прохождения практики: УВЦ, Приморский пр. 63

(наименование и адрес организации)

Период прохождения практики

с «9» декабря 2024 г. по «21» декабря 2024 г.

**Виды и качество выполнения работ**

|  |  |  |
| --- | --- | --- |
| **Виды выполненных работ обучающимся**  **во время практики** | **Объем работ, час.** | **Качество выполнения работ (оценка по пятибалльной системе)** |
| Участие в выработке требований к программному  обеспечению | 9 |  |
| Стадии проектирования программного обеспечения | 21 |  |
| Разработка модулей программного обеспечения | 24 |  |
| Тестирование программных модулей и их интеграции | 9 |  |
| Разработка программной документации и стандарты кодирования | 9 |  |

**Характеристика учебной/профессиональной деятельности обучающегося во время учебной практики (по профилю специальности):**

Общие и профессиональные компетенции, предусмотренные программой практики, освоены **/** не освоены.

(нужное подчеркнуть)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики Шаврова Л.С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.) (подпись)

Дата «21» декабря 2024 г.