

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**  
**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ**  
**Государственное образовательное учреждение**  
**высшего профессионального образования**  
**КЫРГЫЗСКО-РОССИЙСКИЙ СЛАВЯНСКИЙ УНИВЕРСИТЕТ**  
**имени Первого Президента Российской Федерации Б.Н. Ельцина**  
**ЕСТЕСТВЕННО – ТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ**  
**Кафедра информационных и вычислительных технологий**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**на тему:**

**Разработка CRM – системы для автотехцентра «Арсенал»**

**Выполнила студентка группы ЕПИ-2-19**

**Качкынбаева Меруерт**



**Руководитель**

**к.т.н., доцент. Осмонов М.С.**



**Работа к защите допущена**

**заведующей кафедрой ИВТ**

**д.т.н., проф. Лыченко Н.М.**



**БИШКЕК 2023**

## **Аннотация**

Выпускная квалификационная работа содержит информацию об объектно-ориентированном проекте программного приложения CRM – системы для автотехцентра «Арсенал». Приложение обладает следующим функционалом:

- Авторизация пользователя
- Учет заказов
- Отслеживание заказов
- Учет сотрудников центра

и адресовано для использования в автотехцентре.

Приложение может быть использовано как базовое для дальнейшей модификации с добавлением функционала.

Средства разработки используемые для реализации системы: Microsoft Visual Studio 2022, PgAdmin 4.

Язык программирования: C#, JavaScript.

Доступ к данным: ORM Entity Framework Core 7.0.

База данных: Postgres

Контроль версий: GitHub.

Объем пояснительной записки: 74 страницы.

Количество рисунков: 15 рисунков.

Количество таблиц: 10 таблиц.

## **Annotation**

The graduate qualification work contains information about the object-oriented project of the software application CRM-system for the auto service center "Arsenal". The application has the following functionality:

- User authorization
- Order bookkeeping
- Tracking of orders
- Center employees accounting

and is intended for use in an auto service center.

The application can be used as a basic one for further modification with addition of functionality.

The development tools used to implement the system: Microsoft Visual Studio 2022, PgAdmin 4.

The programming language: C#, JavaScript.

Data access: ORM Entity Framework Core 7.0.

Database: Postgres

Version control: GitHub.

Scope of the explanatory note: 74 pages.

Number of images: 15 figures.

Number of tables: 10 tables.

## **Аннотация**

Бүтүрүүчү квалификациялык иш "Арсенал"автотехборбору үчүн ЕЭС программалык тиркемесинин объектиге багытталган долбоору жөнүндө маалыматты камтыйт. Колдонмо төмөнкү функцияга ээ:

- Автколдонуучунун иризациясы
- Заказдарды эсепке алуу
- Заказдарды көзөмөлдөө
- Борбордун кызматкерлеринин эсеби

жана автотехборбордо колдонууга багытталган.

Колдонмону функционалды кошуу менен андан ары өзгөртүү үчүн негизги тиркеме катары колдонсо болот.

Системаны ишке ашыруу үчүн колдонулган иштеп чыгуу куралдары: Microsoft Visual Studio 2022, PgAdmin 4.

Программалоо тили: C#, JavaScript.

Маалыматтарга жетүү: Entity Framework Core 7.0

Маалыматтар базасы: Postgres

Версияны башкаруу: GitHib.

Түшүндүрмө каттын көлөмү: 74-беттер.

Чиймелердин саны: 15 сүрөт.

Таблицалардын саны: 10 таблица.

Введение .....	7
Глава 1. Видение программного продукта.....	9
1.1. Введение .....	9
1.2. Позиционирование программного продукта.....	9
1.3. Описание пользователей .....	12
1.4. Краткий обзор программного продукта .....	14
1.5. Возможности продукта .....	15
1.6. Ограничения .....	15
1.7. Показатели качества .....	16
1.8. Старшинство и приоритеты .....	16
1.9. Другие требования к программному продукту .....	17
Глава 2. Спецификация требований к программному продукту .....	18
2.1. Введение .....	18
2.2. Общее описание .....	20
2.3. Специфические требования .....	22
Глава 3. Проектирование и конструирование ПО .....	32
3.1. Введение .....	32
3.2. Разработка диаграммы классов .....	32
3.3. Разработка диаграммы компонентов .....	35
Глава 4. Разработка тестов и тестирование ПО .....	39
4.1. Разработка плана тестирования.....	39
4.2. Модульное тестирование .....	43
Глава 5. Руководство пользователя .....	44
5.1. Назначение системы .....	44
5.2. Условия применения системы .....	44
5.3. Подготовка системы к работе.....	44
5.4. Описание операция .....	45
5.5. Аварийные ситуации .....	51

Заключение.....	53
Список использованных источников.....	54
Приложение 1. Глоссарий.....	55
Приложение 2. Листинг .....	56

## Введение

В современном мире эффективное управление заказами и учет сотрудников играют важную роль в различных организациях, включая автотехцентры. Вместе с тем, традиционные методы управления заказами и ведения учета часто не обеспечивают достаточной эффективности и точности. В связи с этим возникает потребность в разработке CRM-системы, специально адаптированной для автотехцентров, таких как "Арсенал", с целью оптимизации процессов управления заказами и учета сотрудников.

Объектом исследования данной выпускной квалификационной работы является автотехцентр "Арсенал", который занимается ремонтом и обслуживанием автомобилей. Предметом исследования является разработка CRM-системы, которая будет интегрироваться в рабочий процесс автотехцентра и предоставлять функционал для управления заказами и учета сотрудников.

На сегодняшний день существует ряд CRM-систем, предназначенных для управления заказами и учета сотрудников в различных сферах бизнеса. Однако большинство из них не учитывает специфические потребности автотехцентров. Существующие системы также могут быть дорогостоящими или неудобными в использовании. В связи с этим, разработка специализированной CRM-системы для автотехцентра "Арсенал" представляет собой актуальную задачу.

Целью данной работы является разработка CRM-системы, которая будет предоставлять автотехцентру "Арсенал" эффективный инструмент для управления заказами и учета сотрудников. Для достижения данной цели поставлены следующие задачи:

- Изучение потребностей и требований автотехцентра "Арсенал" в отношении управления заказами и учета сотрудников.
- Анализ существующих систем-аналогов и выявление их преимуществ и недостатков.
- Проектирование и разработка функционального и удобного в использовании веб-интерфейса CRM-системы.
- Разработка модуля авторизации и безопасности для обеспечения конфиденциальности данных.
- Разработка компонентов управления заказами и учета сотрудников, включая создание заказов, добавление информации о заказах, просмотр и обработку заказов, управление списком сотрудников и другие операции.
- Тестирование и отладка разработанной CRM-системы для обеспечения ее корректной работы.

- Подготовка руководства пользователя для ознакомления с функционалом и использованием системы.

Данная выпускная квалификационная работа состоит из следующих глав:

Глава 1. Видение программного продукта: в данной главе будет представлено общее видение разрабатываемой CRM-системы, ее основные цели и потенциальные преимущества для автотехцентра "Арсенал".

Глава 2. Спецификация требований к программному продукту: в этой главе будут подробно описаны требования к функциональности, интерфейсу и безопасности CRM-системы.

Глава 3. Проектирование и конструирование ПО: в данной главе будет представлен процесс проектирования и разработки CRM-системы, включая выбор технологий, архитектурных решений и создание веб-интерфейса.

Глава 4. Тестирование ПО: в этой главе будет описан процесс тестирования разработанной CRM-системы для обеспечения ее надежности и соответствия требованиям.

Глава 5. Руководство пользователя: в данной главе будет представлено подробное руководство по использованию CRM-системы для администраторов и работников автотехцентра "Арсенал".

Заключение: в заключительной главе подведены итоги работы, сделаны выводы и предложения для дальнейшего развития системы.

Список использованной литературы содержит все использованные источники информации, научные работы и статьи, которые служили основой для разработки CRM-системы.

Приложение 1. Глоссарий: в приложении представлены определения основных терминов и понятий, используемых в работе.

Приложение 2. Листинг: в приложении представлены исходный код и структура разработанной CRM-системы.

В результате выполнения данной работы реализована функциональная и удобная в использовании CRM-система, специально адаптированная для автотехцентра "Арсенал". Эта система позволит оптимизировать управление заказами и учет сотрудников, улучшить процессы работы и повысить общую эффективность автотехцентра.



## **Глава 1. Видение программного продукта**

### **1.1. Введение**

#### **1.1.1. Цель**

Целью данной главы является представление общего видения разрабатываемого программного продукта, а именно CRM-системы для автотехцентра "Арсенал". В этой главе будут определены основные цели, потенциальные преимущества и ожидаемые результаты от внедрения разработанной системы.

Данный документ является достаточным для разработки спецификации требований к программному продукту.

#### **1.1.2. Контекст**

Данный документ разрабатывается в рамках выполнения выпускной квалификационной работы.

#### **1.1.3. Определения, акронимы и сокращения**

Основные определения приведены в документе «Приложение 1. Глоссарий проекта».

#### **1.1.4. Ссылки**

При создании документа использовались следующие ссылки, вынесенные в раздел «Список используемой литературы»: [1] – [5].

#### **1.1.5. Краткое содержание**

Данный документ содержит описание общих требований и функциональности для разрабатываемой CRM-системы для автотехцентра «Арсенал». В нем указываются основные бизнес-преимущества предлагаемого решения, формулируются ключевые проблемы и предлагаемые способы их решения, а также представлены характеристики пользователей, возможности, ограничения, критерии качества и другие требования к системе.

### **1.2. Позиционирование программного продукта**

#### **1.2.1. Деловые преимущества**

Разрабатываемая CRM-система предлагает ряд деловых преимуществ, которые делают ее ценным инструментом для управления и эффективной работы автотехцентра. Ниже перечислены основные деловые преимущества предлагаемого решения:

- Централизованное управление заказами.
- Учет сотрудников и назначение задач.

- Улучшенная отслеживаемость и контроль заказов.
- Повышение эффективности и удобство использования.

Описанные деловые преимущества делают разработанную CRM-систему для автотехцентра "Арсенал" востребованной и важной составляющей для оптимизации работы и управления заказами автотехцентра.

### 1.2.2. Определение проблемы

В таблице 1.1 – 1.3 представлены определения проблем.

Таблица 1.1. Определение проблемы

Проблема	Неэффективное управление заказами.
Затрагивает	Сотрудников автотехцентра.
Ее следствием является	Использование бумажных документов, электронных таблиц или отдельных программ, не обеспечивают полной автоматизации и интеграции процессов, что может привести к ошибкам, задержкам и неоптимальной работе.
Успешное решение	Разработка CRM-системы.

Таблица 1.2. Определение проблемы

Проблема	Отсутствие системы учета сотрудников и назначения задач, что затрудняет планирование и распределение работ, а также контроль за их выполнение.
Затрагивает	Администратора.
Ее следствием является	Без централизованной системы учета сотрудников и связанных с ними заказов возникает риск пропуска или дублирования работ, а также затруднения в мониторинге и оценке производительности персонала.
Успешное решение	Реализация модуля по отслеживанию статуса заказа.

Таблица 1.3. Определение проблемы

Проблема	Дублирование информации.
Затрагивает	Работники.
Ее следствием является	Негативно сказывается на уровне обслуживания клиентов и репутации автотехцентра.
Успешное решение	Реализация модуля хранения данных.

В целом, отсутствие у автотехцентра эффективной CRM-системы приводит к проблемам в управлении заказами, распределении задач, контроле за выполнением работ и общей эффективности работы. Поэтому разработка CRM-системы для автотехцентра "Арсенал" позволит решить данные проблемы и повысить уровень сервиса, удовлетворение клиентов и эффективность работы автотехцентра.

### **1.2.3. Определение позиции программного продукта**

В таблице 1.4 представлено определение проблемы.

*Таблица 1.4. Определение позиции изделия*

Для	Сотрудников автотехцентра
Которому	Требуется вести учет заказов
Название продукта	CRM-система для автотехцентра «Арсенал»
Который	Предлагает полный набор функций и возможностей, необходимых для эффективного управления заказами и сотрудниками в автотехцентре. Она обеспечивает централизованное управление заказами, учет сотрудников, назначение задач, отслеживание статуса заказов и другие важные функции, что делает ее комплексным решением для автотехцентров.
В отличие от	Существующих аналогов
Наш продукт	Разрабатывается с учетом специфики и требований автотехцентра "Арсенал". Она позволяет настраивать процессы, сопоставлять их с уникальными потребностями и бизнес-процессами центра.

	Гибкость и настраиваемость системы позволяют ей быть адаптированной под конкретные потребности автотехцентра.
--	---

### **1.3. Описание пользователей**

#### **1.3.1. Сведения о пользователях**

Разработанная CRM-система для автотехцентра "Арсенал" предназначена для использования двумя основными категориями пользователей: администратором и инженерами по ремонту автомобилей.

##### **Администратор:**

Администратор является ключевым пользователем системы и имеет полный доступ ко всем функциям и возможностям CRM-системы. Они ответственны за управление заказами, добавление данных о заказе, прикрепление сотрудников к заказу, отслеживание статуса заказа и закрытие заказа. Также администраторы осуществляют учет сотрудников, включая добавление и удаление сотрудников, а также просмотр списка заказов.

##### **Инженеры по ремонту автомобилей:**

Инженеры автотехцентра, такие как мастера, механики и другие специалисты, являются второй категорией пользователей системы. Они имеют ограниченный доступ к функциональности CRM-системы, позволяющий им просматривать заказы и обрабатывать их. Работники могут принимать заказы, добавлять использованные запчасти, изменять статус заказа и вносить другие необходимые данные.

Система предоставляет разные уровни доступа для администраторов и работников, что обеспечивает конфиденциальность и безопасность данных. Администратор имеет полный доступ и контроль над системой, в то время как работники имеют доступ только к информации, необходимой для выполнения своих рабочих обязанностей.

Учетные записи пользователей создаются администратором системы и предоставляются каждому сотруднику автотехцентра в соответствии с их ролями и обязанностями. Это позволяет эффективно организовать рабочий процесс и обеспечить правильное распределение задач между сотрудниками.

Информация о пользователях и их ролях в CRM-системе "Арсенал" позволяет обеспечить правильное функционирование системы и оптимизировать рабочий процесс автотехцентра.

### **1.3.2. Пользовательская среда**

Система будет использоваться на любом компьютере с выходом в интернет и установленным веб-браузером. В связи с этим пользователь не должен иметь особенных навыков работы с компьютером.

### **1.3.3. Профили пользователей**

В таблицах 1.5 и 1.6 представлены пользователи системы.

*Таблица 1.5. Профиль пользователя «Администратор»*

Типичный представитель	Администратор
Описание	Зарегистрированный пользователь системы, наделенный правами на создание, обработку заказа, а также добавления новых сотрудников
Тип	Пользователь
Ответственности	Отслеживание заказа, учет сотрудников
Критерий успеха	Эффективное управления заказами и сотрудниками

*Таблица 1.6. Профиль пользователя «Инженер по ремонту автомобилей»*

Типичный представитель	Инженер по ремонту автомобилей
Описание	Зарегистрированный пользователь системы, наделенный правами на просмотр заказов, а также их обработку
Тип	Пользователь
Ответственности	Добавление использованных запчастей в заказ, изменение статуса заказа
Критерий успеха	Соблюдение сроков заказов

### **1.3.4. Ключевые потребности пользователей**

- Удобное управление заказами;
- Централизованный доступ к данным;
- Эффективное распределение задач;

- Гибкий и настраиваемый функционал;
- Безопасность данных.

## 1.4. Краткий обзор программного продукта

### 1.4.1. Контекст использования системы

CRM-система для автотехцентра "Арсенал" предназначена для улучшения управления заказами и обеспечения эффективного взаимодействия между администраторами и сотрудниками. Она предоставляет инструменты для автоматизации процессов, связанных с приемом, обработкой и отслеживанием заказов на ремонт и обслуживание автомобилей.

Система будет использоваться внутри автотехцентра "Арсенал" персоналом, включая администратора и сотрудников. Администратор будет использовать систему для управления заказами, добавления информации о заказах, привязки сотрудников к заказам и отслеживания статуса выполнения работ. Сотрудники, такие как механики, автоэлектрики, мотористы и т.д. будут использовать систему для просмотра заказов, принятия заказов на выполнение, добавления информации о использованных запчастях и изменения статуса заказа по мере выполнения работ.

### 1.4.2. Сводка возможностей

В таблице 1.7 представлена сводка возможностей.

Таблица 1.7. Сводка возможностей

Выгоды заказчика	Поддерживающие возможности
Управление заказами	Система позволяет администраторам создавать новые заказы, добавлять информацию о заказах (номер автомобиля, описание проблемы и т.д.), привязывать сотрудников к заказу и отслеживать статус выполнения работ.
Просмотр и обработка заказов	Сотрудники могут просматривать список заказов, отфильтровывать их по различным параметрам (статус, дата и т.д.), принимать заказы на выполнение, добавлять информацию о использованных запчастях и изменять статус заказа по мере выполнения работ.
Учет сотрудников	Администраторы могут управлять списком сотрудников, добавлять новых сотрудников, удалять их из системы и просматривать информацию о каждом сотруднике.

Гибкая настройка системы	Система предоставляет возможность настройки статусов заказов, категорий запчастей, отчетов и других параметров в соответствии с особенностями автотехцентра "Арсенал".
--------------------------------	--

### **1.4.3. Предложения и зависимости**

Система возможно будет интегрирована с базой данных автотехцентра. Также необходимо предусмотреть возможность добавления новых функций, модулей и интеграцию с внешними системами.

## **1.5. Возможности продукта**

Управление заказами: Система позволяет администраторам создавать новые заказы, добавлять информацию о заказах, включая номер автомобиля, описание работ и требуемые запчасти. Администраторы также могут привязывать сотрудников к заказам, отслеживать статус выполнения работ и закрывать заказы по их завершению.

Учет сотрудников: Система обеспечивает возможность добавления и удаления сотрудников автотехцентра. Администраторы могут просматривать список сотрудников, их контактные данные, квалификацию и принадлежность к определенным заказам.

Просмотр и обработка заказов сотрудниками: Сотрудники могут получать доступ к системе с использованием своих учетных записей. Они могут просматривать список доступных заказов, принимать заказы на выполнение, добавлять информацию об использованных запчастях и изменять статус заказа по мере выполнения работ.

Более подробные возможности продукта будут представлены во второй главе «Спецификация требований к программному продукту».

## **1.6. Ограничения**

- 1) Стабильное подключение к интернету
- 2) Один из веб-браузеров:
  - Google Chrome. Версия 36.9 и выше;
  - Mozilla Firefox. Версия 39.0 и выше;
  - Opera. Версия 32.0 и выше;
  - Яндекс Браузер. Версия 11.7 и выше;
  - Safari. Версия 14.1.2.

## **1.7. Показатели качества**

### **1.7.1. Применимость**

- Время необходимое для обучения пользователей – 1 час. Для продвинутых пользователей – 30 минут;
- Время отклика для типичных задач – не более 5 секунд, для сложных задач – не более 20 секунд.

### **1.7.2. Надежность**

- Доступность – время, затрачиваемое на обслуживание системы не должно превышать 2% от общего времени работы.
- Максимальная норма ошибок и дефектов – 1 ошибка на тысячу строк кода.

## **1.8. Старшинство и приоритеты**

1. Критичность функциональности: Некоторые функции и требования могут быть критическими для успешного функционирования автотехцентра. Например, функции, связанные с созданием заказов, учетом сотрудников и отслеживанием статуса заказов, могут иметь высокий приоритет, поскольку они непосредственно влияют на работу центра.
2. Бизнес-приоритеты: Необходимо учитывать бизнес-потребности и цели автотехцентра. Функциональность, которая прямо поддерживает основные бизнес-процессы и помогает повысить эффективность и качество обслуживания клиентов, должна иметь высокий приоритет.
3. Затраты и ресурсы: Оценка затрат и доступных ресурсов также влияет на определение приоритетов. Функциональность, которая требует значительных временных, финансовых или человеческих ресурсов, может быть определена как менее приоритетная или может быть разделена на фазы или итерации разработки.
4. Обратная связь пользователей: Важно учитывать мнение и потребности пользователей автотехцентра. Их обратная связь и предпочтения могут помочь определить приоритеты и внести коррективы в разработку системы.
5. Возможность расширения и развития: Приоритет следует отдавать функциональности, которая обеспечивает гибкость, масштабируемость и возможность дальнейшего развития системы. Это позволит адаптировать систему к будущим потребностям и изменениям в автотехцентре.

Определение старшинства и приоритетов в разработке CRM-системы поможет обеспечить эффективное и целенаправленное развитие программного продукта, соответствующего потребностям автотехцентра "Арсенал".



## **1.9. Другие требования к программному продукту**

### **1.9.1. Применяемые стандарты**

- ISO 9001:2015 "Системы менеджмента качества - Требования" [1];
- ISO/IEC 12207 "Жизненный цикл программного обеспечения и его процессы" [2];
- ISO/IEC 15504 "Оценка процессов жизненного цикла программного обеспечения (SPICE)" [3];
- ISO/IEC 27001 "Информационная технология. Методы обеспечения информационной безопасности. Системы менеджмента информационной безопасности. Требования" [4];
- OWASP Top 10 "Десять наиболее значимых уязвимостей веб-приложений" [5].

## **Глава 2. Спецификация требований к программному продукту**

### **2.1. Введение**

#### **2.1.1. Назначение**

Назначение документа «Спецификация требований к CRM-системе для автотехцентра "Арсенал"» заключается в определении и описании требований к разрабатываемой CRM-системе и их согласовании с заказчиком. Данный документ играет важную роль в процессе разработки программного продукта, так как он служит основой для всех последующих этапов разработки, включая проектирование, реализацию и тестирование системы.

#### **2.1.2. Цели создания системы, решаемые задачи и область применения**

##### **2.1.2.1. Краткое описание деятельности заказчика и существующих технологий**

Автотехцентр "Арсенал" является специализированным сервисным центром, предоставляющим широкий спектр услуг по обслуживанию и ремонту автомобилей. Он осуществляет диагностику, ремонт и техническое обслуживание автомобилей различных марок и моделей.

Деятельность автотехцентра включает следующие основные процессы:

- Прием заказов: Администраторы принимают заказы от клиентов, в которых указываются проблемы с автомобилем, запланированные работы и ожидаемое время выполнения. Заказы могут быть как запланированными, так и срочными.
- Управление заказами: Администраторы отслеживают состояние заказов, назначают и распределяют работников для выполнения работ, контролируют сроки выполнения и обновляют статусы заказов.
- Работа: Инженеры выполняют работы по диагностике, ремонту и обслуживанию автомобилей. Они могут использовать различные инструменты, запчасти и техническую документацию для выполнения задач.

В настоящее время автотехцентр "Арсенал" использует ручные и несистематизированные методы учета и управления заказами и работниками. Взаимодействие с клиентами и ведение учета осуществляются посредством бумажных документов и таблиц Excel. Это приводит к недостаточной эффективности работы, потере информации и затрудняет контроль над процессами.

### ***2.1.2.2. Цели создания системы. Эффекты от ее внедрения***

Разработка CRM-системы для автотехцентра "Арсенал" позволит автоматизировать основные бизнес-процессы, улучшить учет и управление заказами, оптимизировать распределение ресурсов и повысить удовлетворенность клиентов. Это позволит автотехцентру более эффективно функционировать, повысить качество обслуживания и увеличить свою конкурентоспособность на рынке.

### ***2.1.2.3. Назначение системы и область применения***

Назначение разработанной CRM-системы для автотехцентра "Арсенал" заключается в автоматизации и оптимизации процессов учета, управления заказами и обслуживания клиентов. Она предоставляет инструменты и функционал для эффективной работы администраторов и сотрудников, снижает вероятность ошибок и повышает производительность труда.

CRM-система для автотехцентра "Арсенал" может быть применена в других сервисных центрах и организациях, осуществляющих аналогичную деятельность по обслуживанию автомобилей. Она может быть настроена и адаптирована под конкретные потребности и требования каждого автосервиса, обеспечивая эффективное управление клиентской базой, заказами и ресурсами.

### ***2.1.3. Определения, акронимы и сокращения***

Определения приведены в документе «Приложение 1. Глоссарий проекта».

### ***2.1.4. Ссылки***

При создании документа использован источник [6].

### ***2.1.5. Краткое содержание***

- **Определение функциональности.** Документ «Спецификация требований к CRM-системе» содержит детальное описание функциональности и возможностей, которые должны быть реализованы в системе. Он определяет основные функции, взаимодействие с пользователями, обработку данных и другие аспекты, необходимые для достижения поставленных целей.
- **Формализация требований.** Спецификация требований формализует и конкретизирует требования, выдвигаемые к системе. Она перечисляет все необходимые характеристики, ограничения, интерфейсы, алгоритмы и другие элементы системы, которые должны быть учтены при ее разработке.

- Описание объектов системы. Документ также содержит описание объектов и компонентов системы, их взаимосвязь и функциональные взаимодействия. Это позволяет разработчикам лучше понять структуру системы и обеспечить ее эффективное функционирование.
- Согласование с заказчиком. Одной из основных задач спецификации требований является достижение согласия с заказчиком относительно требований к системе. Документ служит основой для обсуждения и уточнения требований, а также для установления взаимопонимания между заказчиком и командой разработки.

## **2.2. Общее описание**

### **2.2.1. Позиционирование программного продукта**

CRM-система для автотехцентра "Арсенал" позиционируется как инновационное решение, направленное на оптимизацию и автоматизацию управления процессами в автосервисе. Программный продукт создан с учетом специфических требований и потребностей автотехнического бизнеса, с целью повышения эффективности работы, улучшения качества обслуживания клиентов и оптимизации управленческих решений.

Программная система обладает следующими ключевыми характеристиками и преимуществами, которые определяют ее позиционирование на рынке:

- Интегрированный подход: CRM-система объединяет в себе различные функциональные модули, включая учет заказов, управление сотрудниками, учет запчастей и материалов, а также взаимодействие с клиентами. Интеграция всех этих модулей в единую систему позволяет автотехцентру эффективно управлять всеми аспектами своей деятельности и обеспечить единое информационное пространство.
- Гибкость и настраиваемость: CRM-система предоставляет возможность настройки под конкретные требования и особенности автотехцентра "Арсенал". Пользователи могут настроить рабочие процессы, формы и отчеты, а также определить права доступа к информации, обеспечивая гибкость и индивидуальный подход к управлению.

Позиционирование CRM-системы "Арсенал" в качестве интегрированного, удобного в использовании и гибкого инструмента для автотехцентра позволяет ей выделиться на рынке и эффективно решать задачи управления и обслуживания клиентов.

### ***2.2.2. Функциональность продукта***

CRM-система обладает широким функционалом, который позволяет эффективно управлять клиентской базой, заказами и ресурсами. Вот основные функции, предоставляемые программным продуктом:

- Учет клиентов
- Управление заказами
- Учет сотрудников
- Интерфейс для сотрудников
- Авторизация и безопасность

CRM-система для автотехцентра "Арсенал" обладает мощным и гибким функционалом, способным удовлетворить потребности и требования как администраторов, так и сотрудников, и сделать работу с клиентами более эффективной и удобной. Более подробное описание прописано в пункте «2.3.2. Функциональные требования».

### ***2.2.3. Характеристики пользователей***

Администратор: это пользователь с высоким уровнем доступа и полными правами на управление системой. Администратор имеют полный контроль над функциональностью системы и осуществляют руководство и контроль над работой автотехцентра.

Инженеры по ремонту автомобилей автотехцентра: включает механиков, автомехаников и других специалистов, работающих в автотехцентре. Они используют CRM-систему для просмотра назначенных заказов, обновления статусов заказов, добавления информации о проделанной работе и использованных запчастях. Сотрудники автотехцентра полагаются на систему для упорядочения своей работы и эффективного выполнения заказов.

### ***2.2.4. Ограничения***

- Безопасность данных
- Масштабируемость
- Удобство использования
- Бюджетные ограничения
- Время разработки

### ***2.2.5. Предложения и зависимости***

- Доступ в интернет
- Компьютер/ноутбук

### ***2.2.6. Распределение требований***

Требования к CRM-системе для автотехцентра "Арсенал" могут быть распределены по следующим основным компонентам:

Модуль управления заказами: Этот модуль должен обеспечивать функциональность, связанную с обработкой заказов. Он включает в себя возможности создания заказа, добавления данных о заказе, прикрепления сотрудников к заказу, отслеживания статуса заказа и закрытия заказа.

Модуль учета сотрудников: Этот модуль предназначен для учета сотрудников автотехцентра. Он позволяет добавлять и удалять сотрудников, а также просматривать список заказов, связанных с каждым сотрудником.

Модуль авторизации и аутентификации: Этот модуль обеспечивает безопасность системы путем аутентификации пользователей и контроля доступа. Он позволяет администратору и сотрудникам проходить авторизацию в системе и получать доступ к соответствующим функциям в зависимости от их роли.

Модуль управления запчастями: Этот модуль позволяет сотрудникам автотехцентра добавлять информацию о использованных запчастях в рамках каждого заказа. Он также предоставляет функциональность для изменения статуса заказа после обработки запчастей.

Модуль интерфейса пользователя: Этот модуль отвечает за удобный и интуитивно понятный интерфейс системы. Он должен быть дружелюбным к пользователю, обеспечивать легкую навигацию и удобный доступ ко всем функциям системы.

Распределение требований по указанным модулям поможет структурировать разработку CRM-системы и обеспечить удовлетворение потребностей пользователей автотехцентра "Арсенал".

## **2.3. Специфические требования**

### ***2.3.1. Требования к внешним интерфейсам***

- Интуитивно понятный дизайн
- Графическое отображение данных
- Интерактивные элементы
- Текстовый материал должен быть читабельным

### 2.3.2. Функциональные требования

Функциональное требование — это заявление о том, как должна вести себя система. Он определяет, что система должна делать, чтобы удовлетворить потребности или ожидания пользователя [6].

В таблице 2.1 представлены актеры системы.

Таблица 2.1. Актеры системы

<i>Актёр</i>	<i>Краткое описание</i>
Администратор	Зарегистрированный пользователь системы, обладающий широким спектром возможностей, обладающий правом на создание и последующем отслеживании статуса заказа, а также имеющий возможность контроля над учетом работников.
Инженер по ремонту автомобилей	Зарегистрированный пользователь системы, имеющий ограниченные права. Пользователь имеет право на принятие заказа, добавление запчастей, а также на завершении заказа.

В таблице 2.2 представлена спецификация вариантов использования.

Таблица 2.2. Спецификация вариантов использования

<i>Основной актер</i>	<i>КОД</i>	<i>Наименование</i>	<i>Формулировка</i>
Администратор	A1	Авторизация	Пользователь вводит email и пароль.
	A2	Просмотр заказов	Пользователь просматривает все заказы.
	A3	Обработка заказа	Пользователь обрабатывает заказ (то есть выполняет какие-либо действия из A3 - A7).

	A4	Создание заказа	Пользователь создает новый заказ.
	A5	Добавление данных о заказе	Пользователь вводит данные заказа.
	A6	Прикрепляет работников	Пользователь прикрепляет работников к заказу.
	A7	Отслеживание статуса заказа	Пользователь просматривает статус заказа.
	A8	Просмотр заказов	Пользователь просматривает все активные заказы.
	A9	Учет работников	Пользователь ведет учет работников (то есть выполняет какие-либо действия из A10 – A31).
	A10	Добавление нового работников	Пользователь заполняет данные сотрудника, вводит его email и пароль.
	A11	Удаление работника	Пользователь удаляет работника, отбирая у него права доступа к системе
	A12	Просмотр списка работников	Пользователь просматривает список всех работников.
	A13	Сохранение в БД	Пользователь сохраняет данные в базе данных.
	A14	Закрытие заказа	Пользователь меняет статуса заказа на «Закрыто».
Инженер	И1	Авторизация	Пользователь вводит email и пароль.
	И2	Просмотр заказов	Пользователь просматривает все заказы.
	И3	Обработка заказов	Пользователь обрабатывает заказ (то есть выполняет какие-либо действия из P4 – P7).



	И4	Принятие заказа	Пользователь принимает заказ в работу.
	И5	Добавление запчастей	Пользователь добавляет список запчастей в заказ.
	И6	Изменение статуса заказа	Пользователь изменяет статус заказа на завершен.
	И7	Сохранение в БД	Пользователь сохраняет данные в базе данных.

На рисунке 2.1 представлена USE-CASE диаграмма вариантов использования системы. На рисунке 2.2 представлена развернутая USE-CASE диаграмма варианта использования «Обработка заказа», т.к. анализ сформулированных вариантов использования показал, что с точки зрения потенциальных рисков и архитектурной значимости наиболее существенными являются прецеденты, связанные с обработкой заказа.

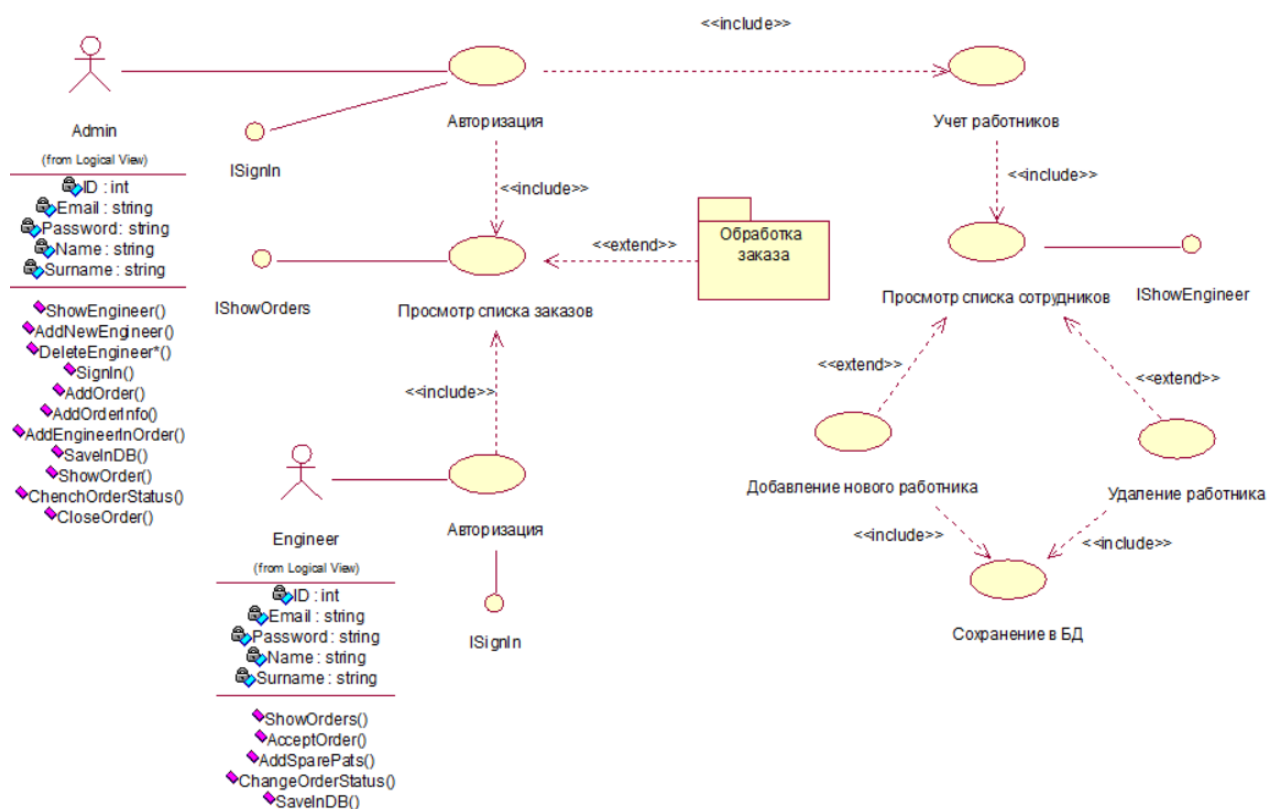


Рисунок 2.1. USE-CASE диаграмма системы.

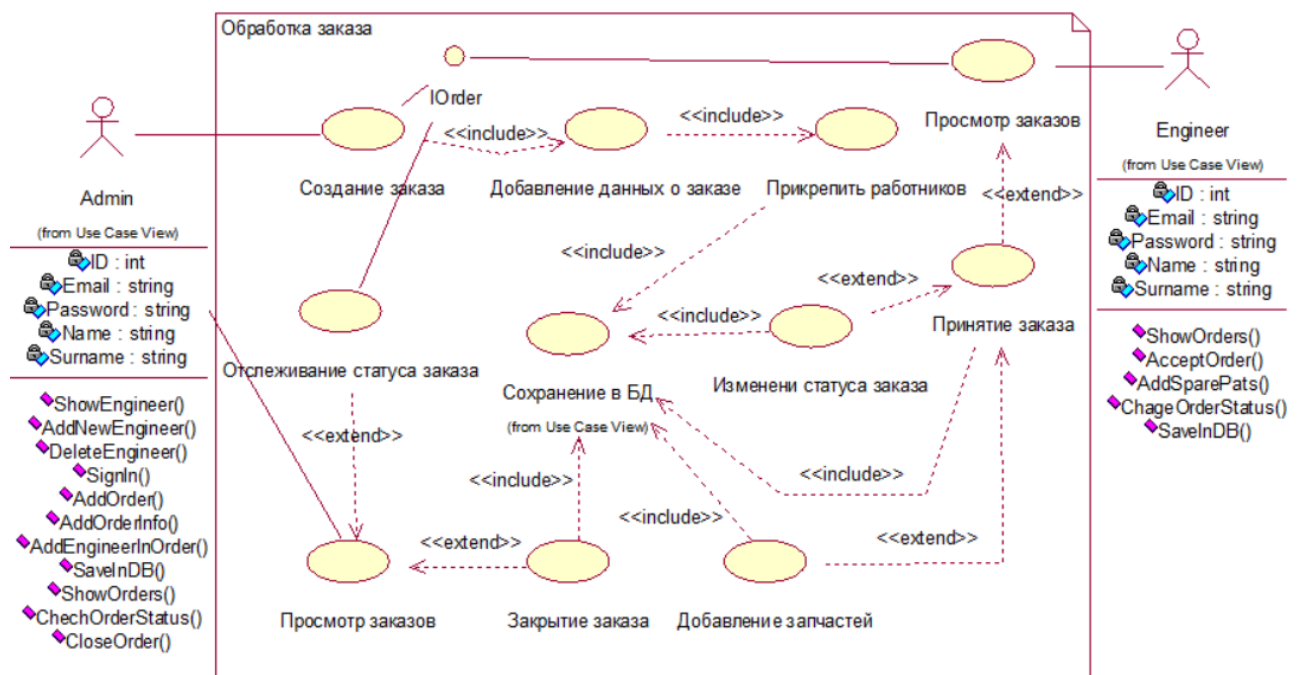


Рисунок 2.2. USE-CASE диаграмма пакета «Обработка заказа»

Выявлено 2 актера системы: Администратор, Инженер.

*Основные функции Администратора:*

- Обработка заказа: создание заказа, добавление данных о заказе, прикрепление работников к заказу, отслеживание статуса заказа, закрытие заказа.
- Учет сотрудников: добавление/удаление сотрудников, просмотр списка сотрудников.

*Основные функции Инженера:*

- Просмотр активных заказов
- Обработка заказа: принятие заказа, добавление использованных запчастей к заказу, изменение статуса заказа.

Для дальнейшей детализации выбраны 2 прецедента:

- А3. Обработка заказа;
- Р3. Обработка заказа.

### Прецедент А3. Обработка заказа пользователя «Администратор»

Обработка заказа

*Краткое описание:*

Пользователь создает заказ, добавляет информацию о заказе, прикрепляет сотрудников, отслеживает статус заказа и при выдаче автомобиля клиенту завершает заказ.

#### *Поток событий*

Прецедент начинается, когда пользователь проходит авторизацию в системе и просматривает список заказов.

#### Базовый поток – Обработка заказа

1. Пользователь выбирает «Создать заказ».
2. Пользователь вводит данные заказа в поля.
3. Система сохраняет данные.
4. Пользователь вводит работников, которые будут принимать участие в ремонте автомобиля.
5. Система сохраняет данные.
6. Пользователь возвращается в «Заказы».
7. Пользователь просматривает статус нужного ему заказа.
8. Пользователь выбирает «Заккрыть» заказ.
9. Система сохраняет данные.

На рисунке 2.3 показана диаграмма деятельности данного прецедента.

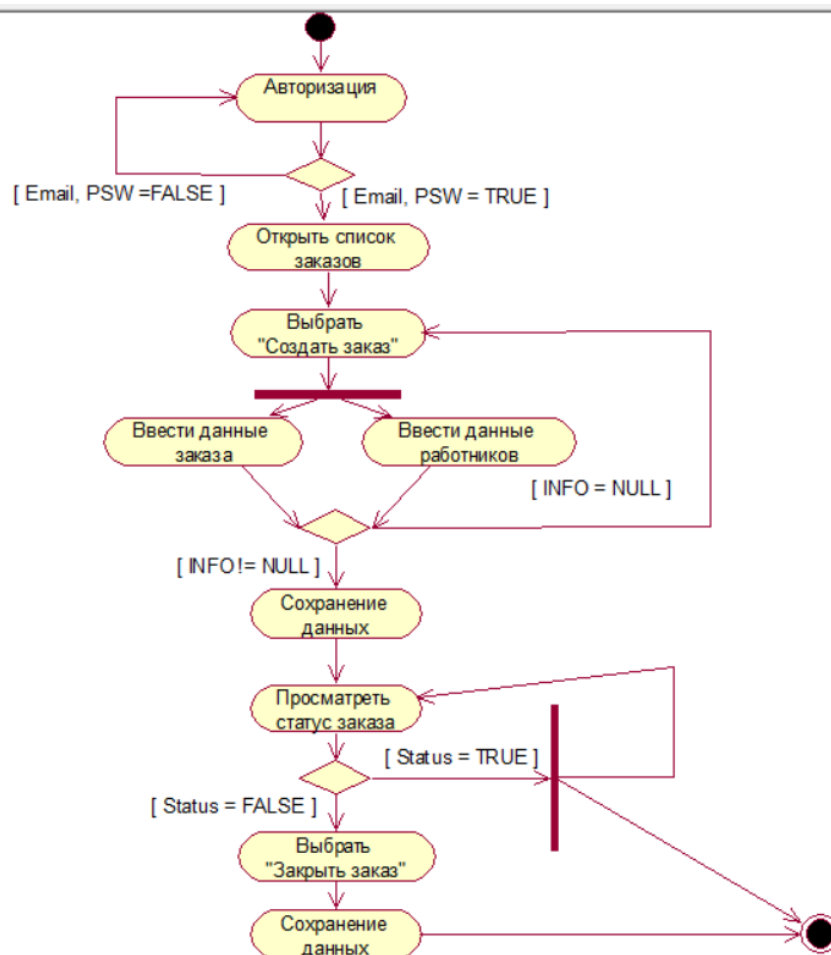


Рисунок 2.3. Диаграмма деятельности прецедента «Обработка заказа» пользователя «Администратор»

### Альтернативные потоки

Если не все поля были заполнены, система просит ввести данные повторно.

Если заказ не был выполнен, система не позволит «Закрыть» заказ.

### Предусловия

Перед тем как начинается этот прецедент, клиент должен быть зарегистрирован в системе.

### Постусловия

При успешном окончании прецедента пользователь может видеть заказ в выполненных заказах.

### Прецедент P3. Обработка заказа пользователя «Инженер»

Обработка заказа

*Краткое описание:*

Пользователь просматривает список активных заказов, принимает заказ, добавляет использованные запчасти, изменяет статус заказа на «Выполнено», при завершении ремонта автомобиля.

*Поток событий*

Прецедент начинается, когда пользователь проходит авторизацию в системе.

Базовый поток – Обработка заказа

1. Пользователь выбирает «Принять заказ».
2. Пользователь вводит данные запчастей в поле ввода.
3. Система сохраняет данные.
4. Пользователь меняет статус заказа на «Выполнено».
5. Система сохраняет данные.
6. Пользователь возвращается в «Заказы».

На рисунке 2.4 показана диаграмма деятельность данного прецедента.

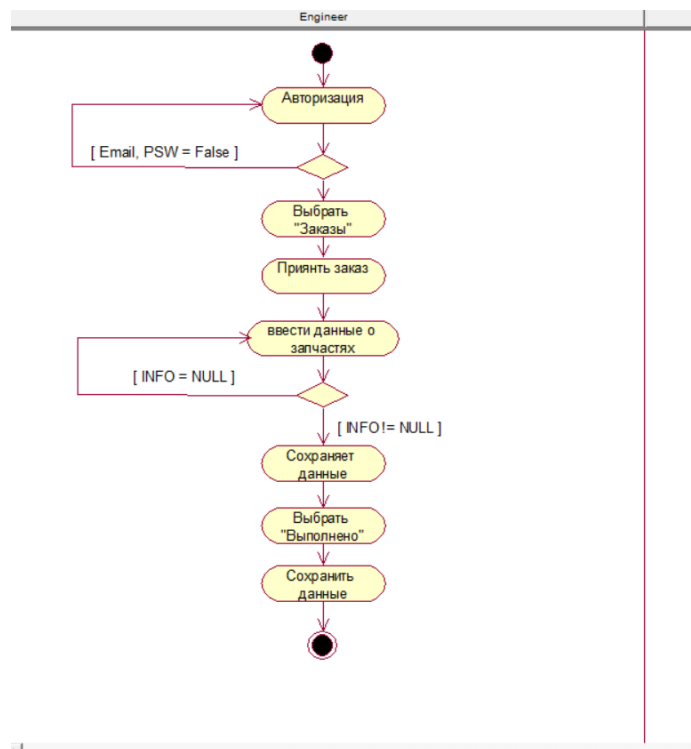


Рисунок 2.4. Диаграмма деятельности прецедента «Обработка заказа» пользователя «Работник»

## **Альтернативные потоки**

Если не все поля были заполнены, система просит ввести данные повторно.

Если заказ не был принят, система не позволит «Выполнить» заказ.

## **Предусловия**

Перед тем как начинается этот прецедент, клиент должен быть зарегистрирован в системе.

## **Постусловия**

При успешном окончании прецедента пользователь может видеть заказ в выполненных заказах.

### ***2.3.3. Требования к производительности***

- **Отклик системы:** Система должна обеспечивать мгновенный отклик на пользовательские действия, такие как открытие страниц, выполнение запросов и обновление данных. Задержки должны быть минимальными, чтобы пользователи могли эффективно работать с системой.
- **Обработка больших объемов данных:** Система должна быть способна обрабатывать и хранить большие объемы данных, таких как информация о клиентах, заказах, запчастях и т.д. Это позволит эффективно управлять данными и обеспечить быстрый доступ к необходимой информации.
- **Масштабируемость:** CRM-система должна быть масштабируемой и способной справиться с ростом числа пользователей и объема данных. Она должна поддерживать горизонтальное и вертикальное масштабирование, чтобы обеспечить стабильную работу системы при увеличении нагрузки.

### ***2.3.4. Логические требования к базе данных***

- **Оптимизация запросов:** Система должна быть оптимизирована для быстрой обработки запросов к базе данных и выполнения операций, таких как поиск, фильтрация и сортировка данных. Это позволит пользователям быстро находить необходимую информацию и выполнять операции с высокой производительностью.
- **Безопасность и защита данных:** Производительность системы должна быть обеспечена при соблюдении высоких стандартов безопасности и защиты данных. Система должна обеспечивать защиту данных от несанкционированного доступа, обеспечивать резервное копирование данных и предотвращать потерю информации.

### **2.3.5. Ограничения проектирования**

Формулировки условий, модифицирующих требования и интерфейс программы должен быть представлен на русском языке.

### **2.3.6. Атрибуты программной системы**

#### **2.3.6.1. Надежность**

- Программный продукт должен отвечать современным требованиям разработки в целях достижения стабильности работы.
- Возможность отката изменений и восстановления проекта.

#### **2.3.6.2. Доступность**

- Среднее время выполнения запроса: 10 секунд;
- Максимальное время выполнения: 20 секунд.

#### **2.3.6.3. Безопасность**

- В случае ошибок системы, скачиваемые данные не подвергаются изменениям.
- Хранимые данные как пользовательские, так и результаты мониторинга защищены
- от вмешательства на несанкционированное изменение и удаление из базы данных.

#### **2.3.6.3. Поддерживаемость**

Все модульные части системы разбиты на классы. Поддержка может обеспечиваться путем добавления новых классов, методов, интерфейсов и представлений.

## **Глава 3. Проектирование и конструирование ПО**

### **3.1. Введение**

Для эффективной работы организаций важно решать проблему управления данными в контексте системы управления базами данных (СУБД). Автоматизация и эффективное применение технологий и программных продуктов, связанных с СУБД, являются неотъемлемой частью функционирования современных предприятий.

Принципы объектно-ориентированного подхода: "разделяй и властвуй" (разбиение сложных проблем на множество лёгких для понимания задач) и иерархическое упорядочение (организация составных частей системы в древовидные структуры с добавлением новых деталей на каждом уровне) – сохраняются так же и при структурном подходе. Однако при этом подходе анализируются не объекты и их взаимодействие, а алгоритмы выполняемых системой функций и отношения между данными при выполнении этих функций [7].

В данном контексте, целью данной главы "Проектирование и конструирование ПО" является построения и применения технологий, инструментов и программных продуктов, связанных с управлением базами данных. Будут рассмотрены вопросы выбора и конфигурирования СУБД, автоматизации процессов проектирования баз данных с использованием CASE-систем, администрирования и мониторинга баз данных, а также других связанных аспектов [8].

Результатом выполнения данной главы будет разработанные диаграмма классов, компонентов, а также диаграммы деятельности.

### **3.2. Разработка диаграммы классов**

Диаграмма классов на унифицированном языке моделирования (UML) — это диаграмма статической структуры, которая описывает структуру системы, показывая ее классы, их атрибуты, операции (или методы) и отношения между объектами. Диаграмма классов — это план системы или подсистемы. Диаграммы классов можно использовать для моделирования объектов, составляющих систему, для демонстрации отношений между объектами и для описания ролей этих объектов и предоставляемых ими услуг [9].

В системе 4 класса представляющие модели:

- User (родительски класс);
- Admin;
- Engineer;
- Order;



Также следующие интерфейсы:

- UserController;
- AdminController;
- EngineerController;
- OrderController.

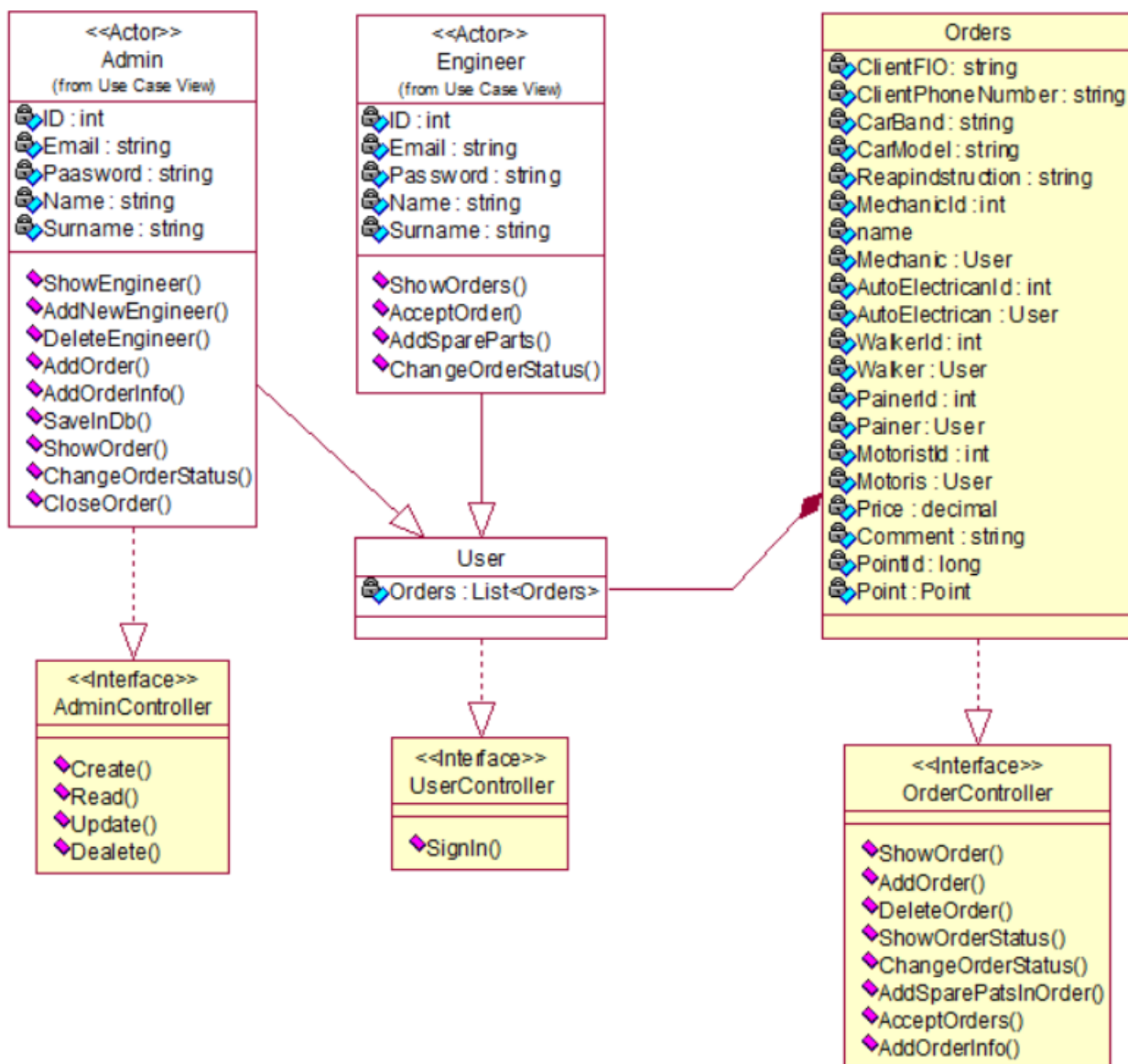


Рисунок 3.1. Диаграмма классов

### 3.2.1. Разработка класса User

User - это родительский класс, от которого наследуются все классы типов пользователей. Класс отвечает за реализацию авторизации.

Атрибуты класса User:

- ID: int – Уникальный номер в таблице;
- Email: string – Адрес электронной почты;
- Password – Пароль;
- Name: string – Имя пользователя;
- Surname: string – Фамилия пользователя;

Методы интерфейса UserController класса User:

- SignIn() – Авторизация пользователя.

### ***3.2.1. Разработка класса Admin***

Admin – это класс модели для пользователя Администратор. Класс отвечает за реализацию отчетов.

Атрибуты класса Admin

- ID: int – Уникальный номер в таблице;
- Email: string – Адрес электронной почты;
- Password – Пароль;
- Name: string – Имя пользователя;
- Surname: string – Фамилия пользователя;

Методы интерфейса AdminController класса Admin:

- SaveInDB() – Сохранения в БД;
- Show otceht() – Показать отчет.

### ***3.2.1. Разработка класса Engineer***

Engineer – это класс модели для пользователя Работник. Класс отвечает за реализацию добавления, удаления, просмотра работников.

Атрибуты класса Engineer

- ID: int – Уникальный номер в таблице;
- Email: string – Адрес электронной почты;
- Password – Пароль;
- Name: string – Имя пользователя;
- Surname: string – Фамилия пользователя;

Методы интерфейса EngineerController класса Engineer:

- SaveInDB() – Сохранения в БД;
- ShowAllEnginner() – Показать всех работников;
- AddNewEngineer() – Добавить нового работника;
- DeleteEngineer() – Удалить работника.

### **3.2.1. Разработка класса Order**

Order – это класс модели для Заказа. Класс отвечает за реализацию обработки заказа.

Атрибуты класса Order:

- ID: int – Уникальный номер в таблице;
- Info: string – Информация о заказе;
- Engineers: List<Engineer> - Сотрудники участвующие в ремонте;
- Status: string;

Методы интерфейса Order Controller класса Order:

- SaveInDB() – Сохранения в БД;
- ShowOrders() – Показать заказы;
- AddOrder() – Добавить заказ;
- DeleteOrder() – Удалить заказ;
- ShowOrderStatus() – Показать статус заказа;
- ChangeOrderStatus() – Поменять статус заказа;
- AddSparePatsInOrder() – Добавить запчасти;
- AcceptOrder() – Принять заказ;
- AddOrderInfo() – Добавить информацию о заказе;

### **3.3. Разработка диаграммы компонентов**

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними [10].

При разработке диаграммы компонентов выявлены следующие программные компоненты:

Data Base – База данных;

Entity Framework Core – ORM, инструмент для взаимодействия с базой данных;

User – Презентационный слой.

Web-Server – Промежуточный слой системной поддержки.

Server – Слой управления ресурсами.

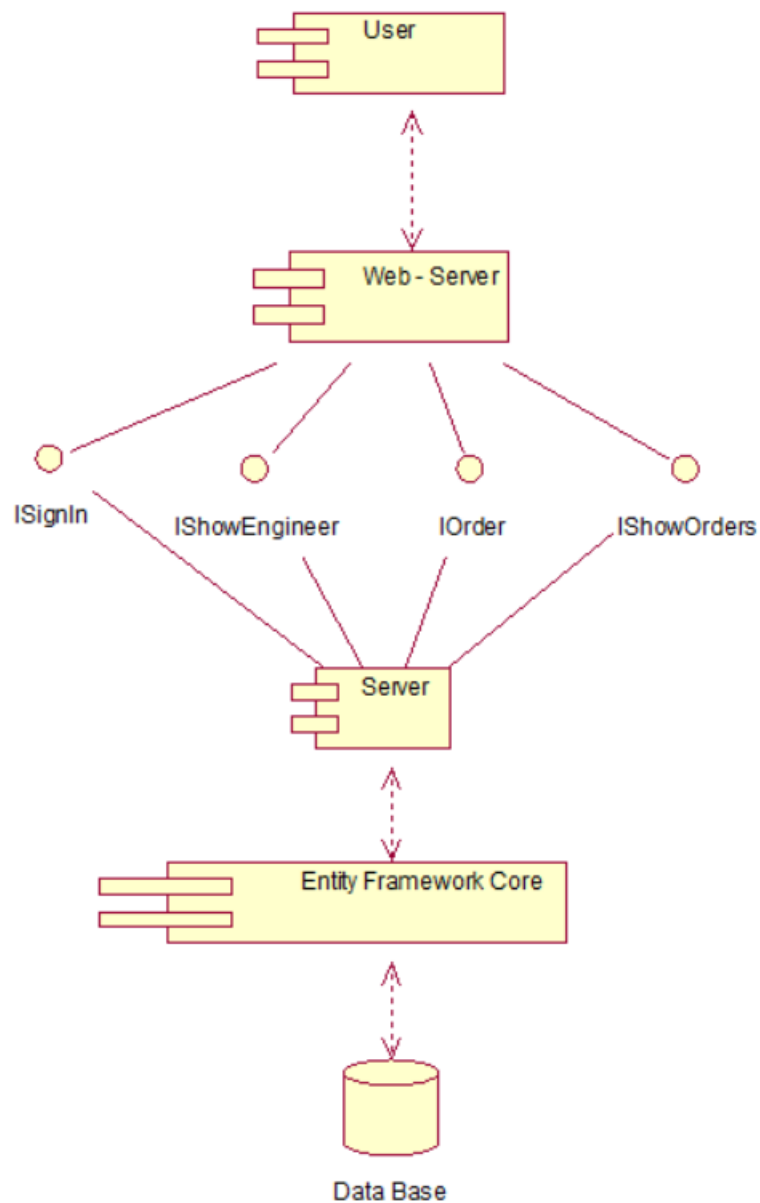


Рисунок 3.2. Диаграмма компонентов

Исходя из диаграммы компонентов (рисунок 3.2.) можно сделать вывод, что в системе лучше всего использовать трехъярусную архитектуру

Трехъярусные архитектуры сложнее и разнообразнее архитектур клиент/сервер. Все слои в них четко разделены. Все слои в них четко разделены. Презентационный слой размещается в клиенте, как в двухъярусной архитектуре. Прикладная логика размещается в среднем ярусе и называется слоем системной поддержки или промежуточным слоем программного обеспечения (middleware). Слой управления ресурсами располагается на третьем ярусе и состоит из всех серверов, которые интегрируются в архитектурное решение. С точки зрения подсистемы управления ресурсами программы, работающие в слое прикладной логики, это просто клиенты [11].

### **3.4. Разработка диаграммы последовательности**

В рамках разработки диаграммы последовательности будет рассмотрен основной прецедент «Обработка заказа».

Для того, чтобы создать заказ, администратор должен пройти авторизацию в системе. Далее администратор вводит данные заказа и прикрепляет к нему инженеров по ремонту автомобилей участвующие в заказе. Система сохраняет данные в БД. У инженеров появляется заказ. Инженер принимает заказ и приступает к ремонту. Система меняет статус заказа и сохраняет его в БД. После того, как инженер закончил работу, заполняет поле «Запчасти» названиями запчастей, что были использованы для ремонта. Система сохраняет данные в БД. Далее инженер меняет статус заказа на «Завершен», если в заказе есть еще работы у других инженеров, система отправляет заказ следующему сотруднику, если же нет, система меняет статус заказа на «Завершен».

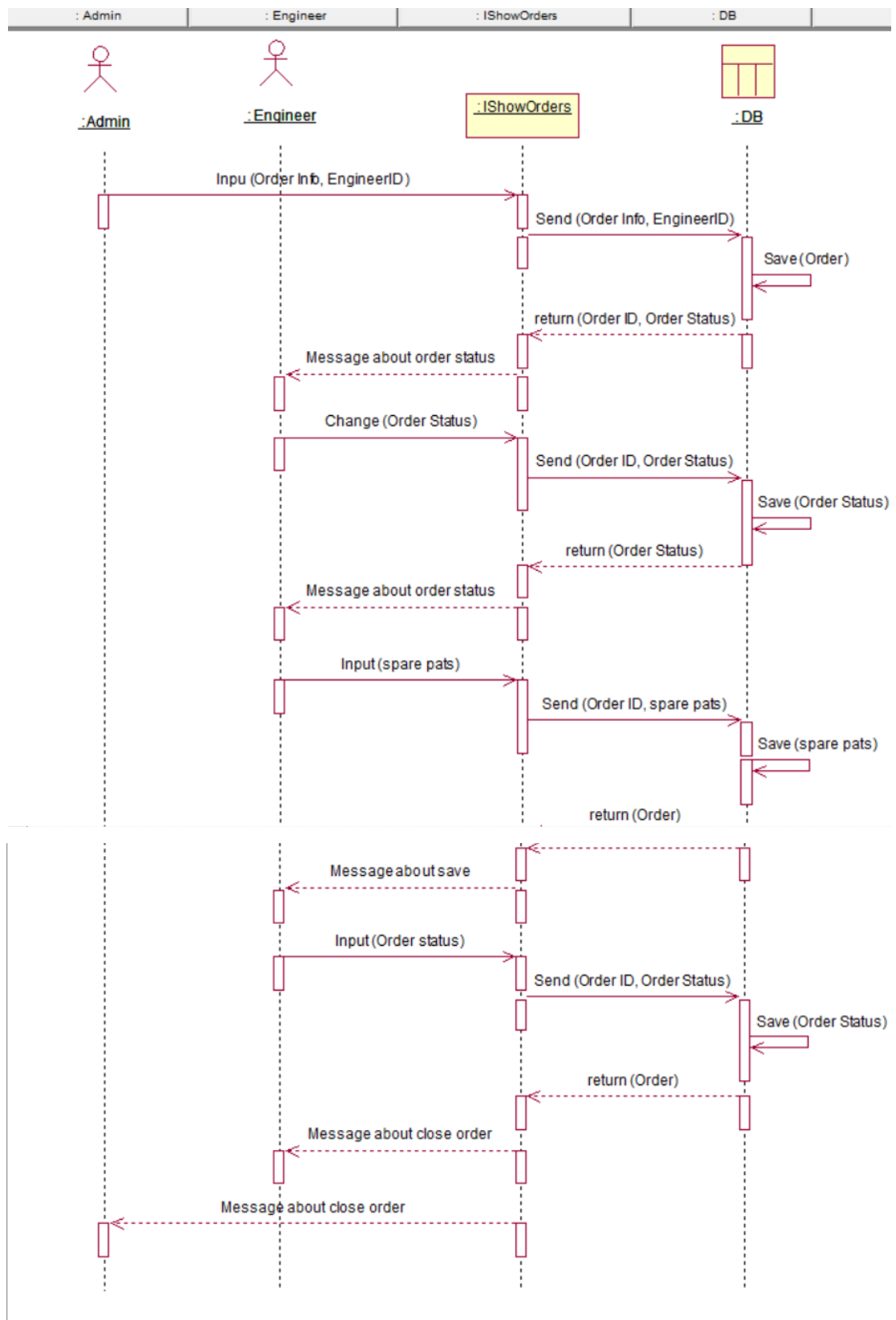


Рисунок 3.3. Диаграмма последовательности сценария «Обработка заказа»

## **Глава 4. Разработка тестов и тестирование ПО**

### **4.1. Разработка плана тестирования**

#### **4.1.1. Введение**

Разработка плана тестирования является важным этапом в процессе разработки программного обеспечения. Цель данной главы "Разработка тестов и тестирование ПО" заключается в определении стратегии и методов тестирования, а также создании подробного плана тестирования для обеспечения качества разрабатываемой системы.

В данном введении будет рассмотрена необходимость проведения тестирования, его роль в процессе разработки и важность создания плана тестирования. Также будут представлены основные задачи, которые ставятся перед командой тестировщиков, а также принципы, на которых основывается разработка плана тестирования.

План тестирования включает в себя описание общей стратегии тестирования, а также подробное описание тестовых сценариев, включая их цели, шаги и ожидаемые результаты. В плане также определяются ресурсы, необходимые для проведения тестирования, расписание и ответственные лица.

Разработка плана тестирования позволяет обеспечить полное и систематическое покрытие функциональности и требований разрабатываемого программного обеспечения. Это помогает выявить ошибки и дефекты в системе, а также улучшить ее качество и надежность.

Результатом выполнения данной главы будет подготовленный план тестирования, который будет использоваться для проведения тестов и проверки работоспособности программного обеспечения. Этот план будет служить основой для успешного завершения этапа тестирования и обеспечения высокого уровня качества разрабатываемой системы.

#### **4.1.2. Техническое задание**

Результатом выполнения данной главы будет подготовленный план тестирования, который будет использоваться для проведения тестов и проверки работоспособности программного обеспечения. Этот план будет служить основой для успешного завершения этапа тестирования и обеспечения высокого уровня качества разрабатываемой системы.

##### **4.1.2.1. Введение**

Данное техническое задание предназначено для CRM-системы для автотехцентра «Арсенал».

#### **4.1.3. Основание для разработки**

- CRM-система разрабатывается в рамках выпускной-квалификационной работы.
- Наименование работы: CRM-система для автотехцентра «Арсенал».
- Исполнитель: Качкынбаева Меруерт ЕПИ-2-19.

##### **4.1.3.1. Назначение**

Целью данной системы является автоматизация бизнес-процессов автотехцентра.

##### **4.1.3.2. Требование к программе**

Функции, предоставляемые программным продуктом:

- Авторизация пользователей
- Создание заказа
- Добавление данных о заказе
- Прикрепить сотрудников к заказу
- Отслеживание статуса заказа
- Добавление работника
- Удаление работника
- Просмотр списка сотрудников
- Просмотр заказов
- Принятие заказа
- Добавление использованных запчастей
- Изменение статуса заказа
- Закрытие заказа

Исходные данные:

- Действия администратора/работников сервиса.

Организация входных и выходных данных:

- Входные данные поступают через действия пользователей/администраторов при использовании клавиатуры и мыши.
- Выходные данные отображаются на экране монитора.

Требования к программной совместимости:

- Программа должна работать на всех современных браузерах.



#### **4.1.4. План тестирования**

##### **4.1.4.1. Введение**

В данном введении будет рассмотрена необходимость разработки плана тестирования, его роль в обеспечении качества программного продукта и основные принципы, которые следует учитывать при его создании.

##### **4.1.4.2. Стратегия тестирования**

Стратегия тестирования определяет общий подход и методы, которые будут использоваться для проведения тестирования программного продукта. Она учитывает характеристики системы, требования к качеству и риски, связанные с ее функционированием.

В рамках разработки плана тестирования для CRM системы автотехцентра, следующие аспекты могут быть включены в стратегию тестирования:

Функциональное тестирование: Основной фокус здесь будет на проверке функциональности системы, включая корректность и полноту реализованных функций. Тестирование будет проводиться на различных сценариях использования системы, включая создание и управление клиентскими профилями, планирование и отслеживание заказов, обработку платежей и прочие функциональные области системы.

Тестирование производительности: Важным аспектом CRM системы для автотехцентра является ее способность обрабатывать большие объемы данных и обеспечивать отзывчивость в режиме реального времени. Тестирование производительности будет направлено на оценку времени отклика системы, скорость выполнения операций и обработку параллельных запросов.

Тестирование безопасности: Учитывая, что CRM система будет хранить и обрабатывать конфиденциальные данные клиентов и информацию о заказах, важно провести тестирование безопасности для обнаружения уязвимостей и защиты от несанкционированного доступа или взлома системы. Это может включать проверку механизмов аутентификации, контроля доступа, шифрования данных и других мер безопасности.

##### **4.1.4.3. Область тестирования**

Цель тестирования заключается в определении подхода к тестированию, выборе соответствующих методов и инструментов, а также организации ресурсов для проведения тестов. Он также устанавливает критерии успешного прохождения тестов и определяет ответственные лица за выполнение каждого этапа тестирования.

#### **4.1.4.4. Начальные условия**

- Рабочая CRM система: Перед началом тестирования необходимо убедиться, что CRM система установлена и настроена правильно согласно требованиям и спецификациям.
- Загруженные тестовые данные: Для проведения тестов необходимо иметь загруженные тестовые данные, которые включают разнообразные сценарии использования, типовые операции и предполагаемые ситуации.
- Подготовленная тестовая среда: Тестирование требует наличия подготовленной тестовой среды, включающей необходимые серверы, базы данных, сетевые настройки и другие компоненты, которые обеспечивают функционирование системы.
- Определение критериев приемки: Перед началом тестирования должны быть определены критерии приемки, которые позволят оценить успешность выполнения тестов и готовность системы к применению.

#### **4.1.4.5. Методы тестирования**

Модульное тестирование: Этот метод фокусируется на тестировании отдельных модулей или компонентов системы. Здесь проводятся тесты на уровне исходного кода для проверки их корректной работы и соответствия требованиям.

#### **4.1.4.6. Среда тестирования**

Конфигурация системы во время тестирования:

- Процессор: Intel Core i5-7300hq.
- Оперативная память: 8 GB DDR4.
- Ethernet: 80 Мбит/с.

#### **4.1.4.7. Результаты**

- Отчет о прохождении тестов: В данном отчете представлены подробные результаты проведенных тестов, включая успешно пройденные тестовые сценарии, выявленные ошибки и их описания, а также действия для воспроизведения проблем.
- Записи журнала тестирования: В журнале тестирования фиксируются все проведенные тесты, включая информацию о дате, времени, продолжительности тестирования, использованных тестовых данных и промежуточных результатов.
- Сводная таблица результатов: В этой таблице суммируются результаты всех проведенных тестов, включая количество успешных и неуспешных тестов, процент покрытия функциональности системы, общее время выполнения тестов и другие метрики.

## 4.2. Модульное тестирование

Модульное тестирование (unit-тестирование) - процесс тестирования, который позволяет проверить отдельные модули программного обеспечения на предмет их правильности работы. Это один из самых распространенных методов тестирования и является неотъемлемой частью процесса разработки программного обеспечения [12].

### Тестовые сценарии

Описание: Тесты, проверяющие основные функции системы.

Исходные данные: методы без параметров.

Таблица 4.1. Тестовые сценарии

№	Входные данные	Что проверяется	Ожидаемый результат	Результат работы программы
1	Email: <a href="mailto:admin@example.com">admin@example.com</a> Password: string123	Корректность данных	Вход в систему	Вход в систему
2	Email: <a href="mailto:admin@example.com">admin@example.com</a> Password:38743289089fhuhdj	Корректность данных	Сообщение: Логин или пароль введены неверно	Сообщение: Логин или пароль введены неверно
3	Фамилия: Иванов Имя: Иван Должность: Механик	Корректность данных	Сотрудник добавлен	Сотрудник добавлен
4	Фамилия: Петров Имя: Петр Должность: Петрович	Не является должностью	Сообщение: Такой должности нет	Сообщение: Такой должности нет
5	Данные заказа:	Пустые данные	Сообщение: поля должны быть заполнены	Сообщение: поля должны быть заполнены

## **Глава 5. Руководство пользователя**

### **5.1. Назначение системы**

CRM-система разработана с целью эффективного управления заказами в автотехцентре. Основная цель системы состоит в обеспечении удобного хранения и обработки заказов сотрудниками центра.

Основные задачи, которые затрагиваются перед нашей CRM-системой, включают:

Централизованное хранение и управление клиентской информацией: Система позволяет хранить и структурировать данные о клиентах, их автомобилях, истории обслуживания и других обсуждениях.

Планирование и учет работ: Система предоставляет инструменты для планирования работ, распределения задач между инженерами и контролем их выполнения. Это позволяет более эффективно заниматься рабочим процессом и сократить время выполнения работ.

CRM-система разработана с учетом спецификаций работы автотехцентров и включает в себя более простые и профессиональные функции.

### **5.2. Условия применения системы**

Требования к аппаратному обеспечению:

- Конфигурация компьютера пользователя: система требует наличия компьютера или ноутбука с процессором Intel Core I3 или выше, оперативной памятью 4 гб. или больше.
- Программное обеспечение: Windows/Linux/macOS.
- Сетевое обеспечение: стабильное подключение в интернет.

Квалификация пользователя:

Навыки работы с компьютером: Пользователи должны обладать базовыми навыками работы с компьютером, включая умение запускать программы, запускать навигацию в интерфейсе системы и выполнять основные операции.

### **5.3. Подготовка системы к работе**

Для подготовки системы к работе необходимо учесть следующие компоненты:

- Операционная система: Windows/Linux/macOS
- Веб-браузер: Google Chrome/Mozilla Firefox/Microsoft Edge

Для запуска системы необходимо:

- Открыть веб-браузер
- Перейти по адресной строке «http://localhost:3000»
- Ввести данные пользователя для прохождения авторизации

## 5.4. Описание операция

### 5.4.1. Операция «Авторизация»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.
- Ввод корректных учетных данных пользователя.
- Активность учетной записи пользователя.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Получение учетных данных

Основные действия, требуемые в последующем:

- Ввод логина пользователя в соответствующее поле.
- Ввод пароля пользователя в поле.
- Проверка соответствия введенного логина и заданного значения, хранящегося в системе.

Заключительные действия:

- При активации системы учета данных система предоставляет доступ к функциональности, доступной авторизованному использованию.

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук
- Учетные данные

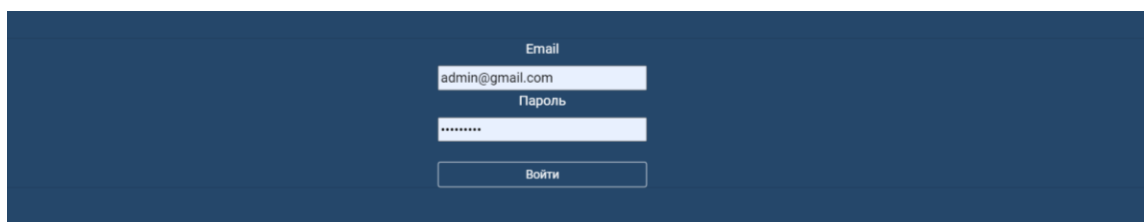


Рисунок 5.1. Операция «Авторизация»

### 5.4.2. Операция «Добавить пользователя»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.
- Данные нового пользователя (ФИО).
- Уникальный адрес электронной почты (email).

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация админа.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Добавить пользователя»
- Заполнить поля Email, Фамилия, Имя, Отчество, Пароль
- Выбрать роль
- Нажать кнопку «Добавить»

Заключительные действия:

- При успешном добавлении выводится сообщение.

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук
- Данные сотрудника

The screenshot shows a web application interface for adding a new user. On the left is a sidebar menu with the following items: 'АДМИН ПАНЕЛЬ', 'Пользователи', 'Добавить пользователя' (highlighted with a blue square), and 'МОДУЛИ'. Under 'МОДУЛИ' are 'Заказы', 'Все', 'В обработке', 'В процессе', and 'Завершен'. The main content area is titled 'Новый пользователь'. It contains several input fields: 'Email' (with 'ivan@gmail.com'), 'Фамилия' (with 'Иванов'), 'Имя' (with 'Иван'), 'Отчество' (with 'Иванович'), 'Пароль' (with masked characters '\*\*\*\*\*'), and 'Роль' (a dropdown menu with 'Моторист' selected). At the bottom of the form is a green button labeled 'Добавить'. In the top right corner of the interface, there is a user profile indicator showing 'admin@gmail.com' and a small user icon.

Рисунок 5.2. Операция «Добавить пользователя»

### 5.4.3. Операция «Удалить пользователя»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация админа.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Список пользователей»
- Выбрать пользователя
- Нажать кнопку «Удалить»
- Подтвердить удаление

Заключительные действия:

- При успешном удалении выводится сообщение.

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук
- У пользователя не должно быть не завершенных заказов

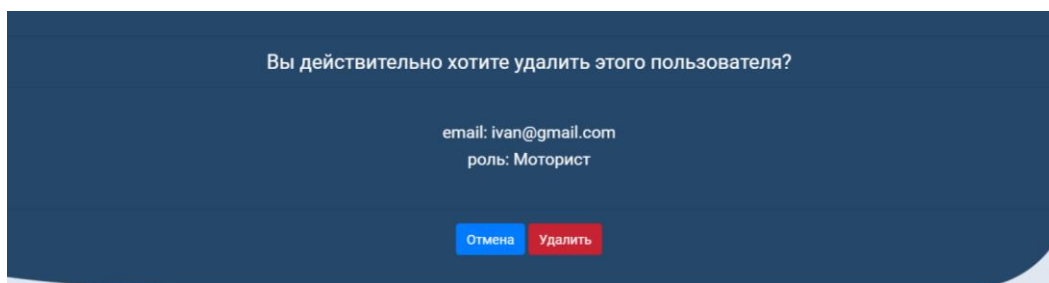


Рисунок 5.3. Операция «Удалить пользователя»

### 5.4.4. Операция «Просмотреть все заказы»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация админа.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Заказы» → «Все»

Заключительные действия:

- При успешном выполнении отображается окно со всеми заказами

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук

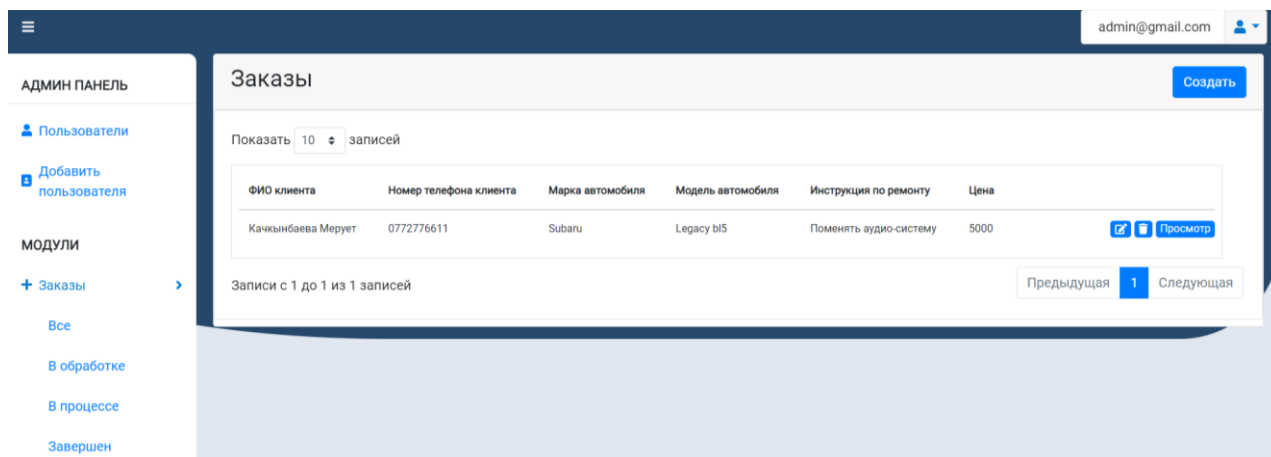


Рисунок 5.4. Операция «Просмотреть все заказы»

#### 5.4.5. Операция «Принять заказ»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация под пользователем.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Заказы» → «В обработке»
- В поле «Действия» выбрать «Принять заказ»
- Подтвердить выбор

Заклучительные действия:

- При успешном выполнении возвращается в окно со всеми заказами

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук



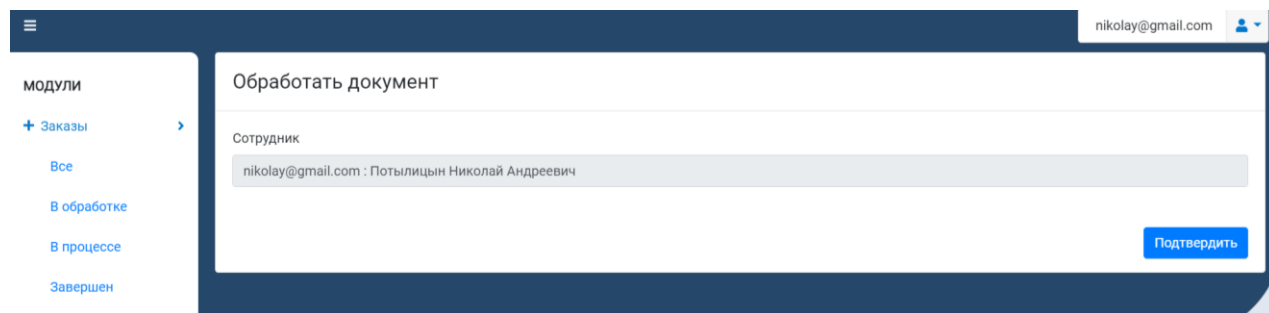


Рисунок 5.5. Операция «Принять заказ»

#### 5.4.6. Операция «Добавить запчасти»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация под пользователем.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Заказы» → «В процессе»
- В поле «Действия» выбрать «Добавить запчасти»
- Заполнить поле «Запчасти»
- Подтвердить выбор

Заключительные действия:

- При успешном выполнении возвращается в окно со всеми заказами

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук

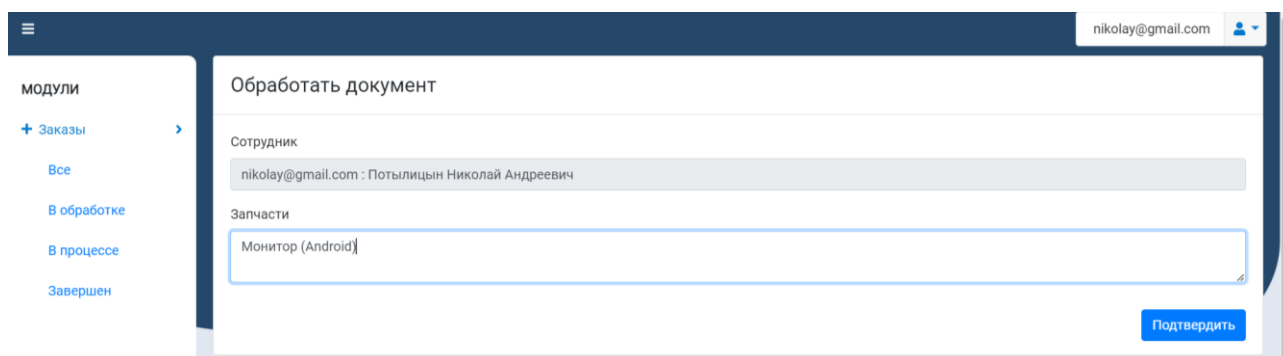


Рисунок 5.6. Операция «Добавить запчасти»

### 5.4.7. Операция «Завершить заказ»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация под пользователем.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Заказы» → «В процессе»
- В поле «Действия» выбрать «Завершить»
- Подтвердить выбор
  - Заключительные действия:
- При успешном выполнении возвращается в окно со всеми заказами

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук



Рисунок 5.7. Операция «Завершить заказ»

### 5.4.8. Операция «Создать заказ»

Условия, при соблюдении которых возможно выполнение операции:

- Наличие активного интернет-соединения.

Подготовительные действия:

- Загрузка веб-страниц или запуск приложений.
- Авторизация под администратором.

Основные действия, требуемые в последующем:

- Выбрать в меню поле «Заказы»
- Нажать на кнопку «Создать заказ»

- Заполнить все поля
- Выбрать инженеров, которые будут принимать участие в заказе
- Нажать на кнопку «Сохранить»

Заключительные действия:

- При успешном выполнении возвращается в окно со всеми заказами

Ресурсы, расходуемые на операцию:

- Веб-браузер
- Компьютер или ноутбук

The screenshot shows a web application interface for creating an order. The title bar at the top says 'Заказы' (Orders). In the top right corner, there is a user profile 'admin@gmail.com' and a blue 'Сохранить' (Save) button. The form is divided into several sections:

- Client Information:** 'ФИО клиента' (Client Full Name) with the value 'Качкынбаева Мерует' and 'Номер телефона клиента' (Client Phone Number) with the value '0772776611'.
- Vehicle Information:** 'Марка автомобиля' (Car Brand) with 'Subaru' and 'Модель автомобиля' (Car Model) with 'Legacy bl5'.
- Repair Instructions:** 'Инструкция по ремонту' (Repair Instructions) with the text 'Поменять Андроид - монитор'.
- Cost and Technician Selection:**
  - 'Цена' (Price) is set to '10000'.
  - 'Механик' (Mechanic) is a dropdown menu with 'Выбрать' (Select) shown.
  - 'Автоэлектрик' (Auto Electrician) is a dropdown menu with 'nikolay@gmail.com : Потылицын Николай Анд' (Nikolay Potylicyn Nikolay And) shown.
  - 'Ходовщик' (Chassis) is a dropdown menu with 'Выбрать' (Select) shown.
  - 'Малляр' (Painter) is a dropdown menu with 'Выбрать' (Select) shown.
  - 'Моторист' (Engineer) is a dropdown menu with 'Выбрать' (Select) shown.
- Comments:** 'Комментарии' (Comments) with the text 'сделать до 20.06.2023'.

Рисунок 5.8. Операция «Создать заказ»

## 5.5. Аварийные ситуации

Сбой в работе системы:

- В случае возникновения проблемы с CRM-системой, сначала попробуйте перезапустить приложение.
- Если перезапуск не решает проблему, возникает техническая поддержка или администратором системы для получения помощи.

Потеря данных:

- В случае потери данных в CRM-системе, необходимо срочно обратиться к администратору системы или ответственным за резервное копирование данных.

Потеря доступа к системе:

- Если у вас возникнут проблемы с доступом к CRM-системе, проверьте правильность введенных учетных данных (логин и пароль).
- Если вы забыли пароль или столкнулись с рядом проблем доступа, обратитесь к администратору системы для восстановления доступа.

При возникновении аварийных ситуаций всегда обращаются за помощью к технической поддержке или администратору системы. Быстрое возникновение и решение проблемы устранения проблемы восстановления работоспособности CRM-системы.

## **Заключение**

В ходе данной выпускной квалификационной работы была разработана и реализована CRM-система для автотехцентра «Арсенал», что позволило автоматизировать процессы управления заказов и повысило эффективность работы.

В результате разработки CRM-системы для управления и учета заказов были реализованы следующие сервисы:

- Авторизация
- Аутентификация
- Создание заказа
- Редактирование заказа
- Удаление заказа
- Регистрация новых сотрудников
- Удаление сотрудников
- Контроль за исполнением заказа
- Учет сотрудников

Также проведено тестирование системы, что подтверждает полную готовность к внедрению.

В дальнейшем планируется добавить фильтр поиска клиентов и отчетность у администратора.

## Список использованных источников

1. Системы менеджмента качества — URL: <https://kurskmed.com/upload/files/GOST%20R%20IS%D0%9E%209001-2015.pdf> (дата обращения: 14.04.2023)
2. Software life cycle processes — URL: [http://sewiki.ru/ISO/IEC\\_12207](http://sewiki.ru/ISO/IEC_12207) (дата обращения: 14.04.2023)
3. Оценка процессов — URL: <https://docs.cntd.ru/document/1200076921> (дата обращения: 14.04.2023)
4. Информационные технологии. Методы защиты. — URL: [https://pqm-online.com/assets/files/pubs/translations/std/iso-mek-27001-2013\(rus\).pdf](https://pqm-online.com/assets/files/pubs/translations/std/iso-mek-27001-2013(rus).pdf) (дата обращения: 14.04.2023)
5. Критические угрозы безопасности — URL: [https://wiki.owasp.org/images/9/96/OWASP\\_Top\\_10-2017-ru.pdf](https://wiki.owasp.org/images/9/96/OWASP_Top_10-2017-ru.pdf) (дата обращения: 14.04.2023)
6. Функциональные требования — URL: <https://visuresolutions.com/ru/blog/functional-requirements/> (дата обращения: 14.04.2023)
7. Объектно-ориентированные case-средства анализа и проектирования — URL: <https://studfile.net/preview/7736309/page:49/> (дата обращения: 20.05.2023)
8. С.А. Орлов «Технологии разработки программного обеспечения. Разработка сложных программных систем» - СПб: Питер, 2002. — 464 с.: ил. (ISBN 5-94723-145-X);
9. Руководство по диаграмме классов — URL: <https://www.cybermedian.com/ru/a-comprehensive-guide-to-uml-class-diagram/> (дата обращения: 20.05.2023)
10. Диаграмма компонентов — URL: [http://imlearning.ru/netcat\\_files/file/FSIS/%D0%A4%D0%A1%D0%98%D0%A1%D1%81%D0%B5%D0%BC%D0%B8%D0%BD%D0%B0%D1%80-9%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2.pdf](http://imlearning.ru/netcat_files/file/FSIS/%D0%A4%D0%A1%D0%98%D0%A1%D1%81%D0%B5%D0%BC%D0%B8%D0%BD%D0%B0%D1%80-9%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2.pdf) (дата обращения: 20.05.2023)
11. Л.Е. Карпов «Архитектура распределенных систем программного обеспечения» - Москва, МАКС Пресс, 2007. — 129 с.: ил. (ISBN 978-5-84907-304-0);
12. Модульное тестирование — URL: <https://qaevolution.ru/testirovanie-po/urovni-testirovaniya-po/module-testing/> (дата обращения: 24.05.2023)

## **Приложение 1. Глоссарий**

Авторизация – предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аутентификация – процедура проверки подлинности, например проверка подлинности пользователя путем сравнения введенного им пароля с паролем, сохраненным в базе данных.

Инженер – сотрудник автотехцентра, имеющий техническое образование и занимающийся ремонтом автомобилей.

## Приложение 2. Листинг

### Файл User.cs

```
namespace digitization.Models
{
    public class User: IdentityUser
    {
        [Display(Name = "Фамилия")]
        public string Surname { get; set; }
        [Display(Name = "Имя")]
        public string Name { get; set; }
        [Display(Name = "Отчество")]
        public string? Patronymic { get; set; }
        public List<Orders> Orders { get; set; }
    }
}
```

### Файл Orders.cs

```
public class Orders : BaseEntity
{
    public string ClientFIO { get; set; }
    public string ClientPhoneNumber { get; set; }
    public string CarBrand { get; set; }
    public string I { get; set; }
    public string RepairInstructions { get; set; }
    public string? MechanicId { get; set; }
    public User? Mechanic { get; set; }
    public string? AutoElectricianId { get; set; }
    public User? AutoElectrician { get; set; }
    public string? WalkerId { get; set; }
    public User? Walker { get; set; }
    public string? PainterId { get; set; }
    public User? Painter { get; set; }
    public string? MotoristId { get; set; }
    public User? Motorist { get; set; }
    public decimal Price { get; set; }
    public string? Comment { get; set; }
    public string? SpareParts { get; set; }
    public long PointId { get; set; } = 101;
    public Point Point { get; set; }
}
}
```

### Файл OrdersPointStatus.cs

```
namespace digitization.Models
{
    public class OrdersPointStatus
    {
        [Key, Column(Order = 0)]
        public long OrdersId { get; set; }
        [Key, Column(Order = 1)]
        public string ExecutorId { get; set; }
        public long PointId { get; set; } = 101;
        public byte Order { get; set; }
    }
}
```

### Файл BaseEntity.cs

```
public class BaseEntity
{
}
```



```

public long Id { get; set; }
[Display(Name = "Кем создан")]
public string? CreatedBy { get; set; }
[Display(Name = "Когда создан")]
public DateTime CreatedAt { get; set; }
[Display(Name = "Кем обновлен")]
public string? UpdatedBy { get; set; }
[Display(Name = "Когда обновлен")]
public DateTime? UpdatedAt { get; set; }
}

```

#### Файл DtParameters.cs

```

public class DtParameters
{
    public int draw { get; set; }
    public List<Column> columns { get; set; }
    public List<Order> order { get; set; }
    public int start { get; set; }
    public int length { get; set; }
    public Search Parameters { get; set; }
}
public class Search
{
    public string value { get; set; }
    public bool regex { get; set; }
}

```

#### Файл Point.cs

```

public class Point: BaseEntity
{
    [Display(Name = "Наименование")]
    public string Name { get; set; }
}

```

#### Файл Reason.cs

```

public class Reason: BaseEntity
{
    [Display(Name = "Наименование")]
    public string Name { get; set; }
}

```

#### Файл RoadMap.cs

```

public class RoadMap: BaseEntity
{
    public long SourcePointId { get; set; }
    public Point? SourcePoint { get; set; }
    public long TargetPointId { get; set; }
    public Point? TargetPoint { get; set; }
    public long ReasonId { get; set; }
    public Reason RoadMap's { get; set; }
}

```

#### Файл RoleInitializer.cs

```

public class RoleInitializer {
    public static async Task InitializeAsync(UserManager<User> userManager, RoleManager<IdentityRole>
roleManager) {
        string adminEmail = "admin@gmail.com";
        string password = "_Aa123456";
    }
}

```

```

        if (await roleManager.FindByNameAsync("Администратор") == null) {
            await roleManager.CreateAsync(new IdentityRole("Администратор"));
        }
        if (await roleManager.FindByNameAsync("Механик") == null) {
            await roleManager.CreateAsync(new IdentityRole("Механик"));
        }
        if (await roleManager.FindByNameAsync("Автоэлектрик") == null) {
            await roleManager.CreateAsync(new IdentityRole("Автоэлектрик"));
        }
        if (await roleManager.FindByNameAsync("Ходовщик") == null) {
            await roleManager.CreateAsync(new IdentityRole("Ходовщик"));
        }
        if (await roleManager.FindByNameAsync("Маляр") == null) {
            await roleManager.CreateAsync(new IdentityRole("Маляр"));
        }
        if (await roleManager.FindByNameAsync("Моторист") == null) {
            await roleManager.CreateAsync(new IdentityRole("Моторист"));
        }
        if (await userManager.FindByNameAsync(adminEmail) == null) {
            User admin = new User { Email = adminEmail, UserName = adminEmail, Surname = "Admin", Name =
"Admin" };
            IdentityResult result = await userManager.CreateAsync(admin, password);
            if (result.Succeeded) {
                await userManager.AddToRoleAsync(admin, "Администратор");
                await userManager.AddToRoleAsync(admin, "Механик");
                await userManager.AddToRoleAsync(admin, "Автоэлектрик");
                await userManager.AddToRoleAsync(admin, "Ходовщик");
                await userManager.AddToRoleAsync(admin, "Маляр");
                await userManager.AddToRoleAsync(admin, "Моторист");
            }
        }
    }
}

```

#### Файл ApplicationDbContext.cs

```

public class ApplicationDbContext : IdentityDbContext<User, IdentityRole, string>
{
    public DbSet<Point> Points { get; set; }
    public DbSet<Reason> Reasons { get; set; }
    public DbSet<RoadMap> RoadMaps { get; set; }
    public DbSet<Orders> Orders { get; set; }
    public DbSet<OrdersPointStatus> OrdersPointStatuses { get; set; }
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<OrdersPointStatus>()
            .HasKey(a => new { a.OrdersId, a.ExecutorId });

        modelBuilder.Entity<Orders>()
            .HasOne(o => o.Mechanic)
            .WithMany()
            .HasForeignKey(o => o.MechanicId)
            .OnDelete(DeleteBehavior.SetNull);

        modelBuilder.Entity<Orders>()
            .HasOne(o => o.AutoElectrician)

```

```

        .WithMany()
        .HasForeignKey(o => o.AutoElectricianId)
        .OnDelete(DeleteBehavior.SetNull);

modelBuilder.Entity<Orders>()
    .HasOne(o => o.Walker)
    .WithMany()
    .HasForeignKey(o => o.WalkerId)
    .OnDelete(DeleteBehavior.SetNull);

modelBuilder.Entity<Orders>()
    .HasOne(o => o.Painter)
    .WithMany()
    .HasForeignKey(o => o.PainterId)
    .OnDelete(DeleteBehavior.SetNull);

modelBuilder.Entity<Orders>()
    .HasOne(o => o.Motorist)
    .WithMany()
    .HasForeignKey(o => o.MotoristId)
    .OnDelete(DeleteBehavior.SetNull);

var points = new List<Point>();

points.Add(new Point
{
    Id = 101,
    Name = "в обработке",
    CreatedAt = DateTime.UtcNow,
});
points.Add(new Point
{
    Id = 102,
    Name = "в процессе",
    CreatedAt = DateTime.UtcNow,
});
points.Add(new Point
{
    Id = 103,
    Name = "завершен",
    CreatedAt = DateTime.UtcNow,
});

modelBuilder.Entity<Point>().HasData(points);

var reasons = new List<Reason>();

reasons.Add(new Reason
{
    Id = 1001,
    Name = "принять заказ",
    CreatedAt = DateTime.UtcNow,
});
reasons.Add(new Reason
{
    Id = 1002,
    Name = "добавить запчасти",
    CreatedAt = DateTime.UtcNow,
});
reasons.Add(new Reason
{
    Id = 1003,

```

```

        Name = "завершить заказ",
        CreatedAt = DateTime.UtcNow,
    });

modelBuilder.Entity<Reason>().HasData(reasons);

var roadMaps = new List<RoadMap>();

roadMaps.Add(new RoadMap
{
    Id = 1,
    SourcePointId = 101,
    TargetPointId = 102,
    ReasonId = 1001,
    CreatedAt = DateTime.UtcNow,
});
roadMaps.Add(new RoadMap
{
    Id = 2,
    SourcePointId = 102,
    TargetPointId = 102,
    ReasonId = 1002,
    CreatedAt = DateTime.UtcNow,
});
roadMaps.Add(new RoadMap
{
    Id = 3,
    SourcePointId = 102,
    TargetPointId = 103,
    ReasonId = 1003,
    CreatedAt = DateTime.UtcNow,
});

modelBuilder.Entity<RoadMap>().HasData(roadMaps);
}

```

#### Файл AccountController.cs

```

public class AccountController : Controller {

    private readonly SignInManager<User> signInManager;
    private readonly UserManager<User> userManager;
    private readonly ApplicationDbContext context;

    public AccountController(SignInManager<User> signInManager, UserManager<User> userManager,
        ApplicationDbContext context) {
        this.signInManager = signInManager;
        this.userManager = userManager;
        this.context = context;
    }

    [HttpGet]
    public IActionResult Login(string returnUrl = null) {
        return View(new LoginViewModel { ReturnUrl = returnUrl });
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model) {
        if (ModelState.IsValid) {
            var result =
                await signInManager.PasswordSignInAsync(model.Email, model.Password, true, false);
            if (result.Succeeded) {

```

```

        var user = userManager.FindByEmailAsync(model.Email).Result;
        if (!string.IsNullOrEmpty(model.ReturnUrl) && Url.IsLocalUrl(model.ReturnUrl)) {
            return Redirect(model.ReturnUrl);
        } else {
            return RedirectToAction("Index", "Orders");
        }
    } else {
        ModelState.AddModelError("", "Неправильный логин и (или) пароль");
    }
}
return View(model);
}

[HttpPost]
public async Task<IActionResult> Logout() {
    var user = userManager.GetUserAsync(User).Result;
    await signInManager.SignOutAsync();
    return RedirectToAction("Index", "Orders");
}

public IActionResult AccessDenied() {
    return View();
}

public async Task<IActionResult> ChangePassword() {
    var currentUserId = context.Users.Where(u => u.UserName == User.Identity.Name).FirstOrDefault();
    if (currentUserId == null)
        return NotFound();
    User model. ReturnUrl = await userManager.FindByIdAsync(currentUserId.Id);
    if (user == null) {
        return NotFound();
    }
    ChangePasswordViewModel model = new ChangePasswordViewModel { Id = user.Id, Email = user.Email };
    return View(model);
}

[HttpPost]
public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model) {
    if (ModelState.IsValid) {
        User user. Email. Email. Email. Email. Email. Email. Email. Email = await
userManager.FindByIdAsync(model.Id);
        if (user != null) {
            var _passwordValidator =
HttpContext.RequestServices.GetService(typeof(IPasswordValidator<User>)) as
IPasswordValidator<User>;
            var _passwordHasher =
HttpContext.RequestServices.GetService(typeof(IPasswordHasher<User>)) as IPasswordHasher<User>;
            IdentityResult result =
                await _passwordValidator.ValidateAsync(userManager, user, model.Password);
            if (result.Succeeded) {
                user.PasswordHash = _passwordHasher.HashPassword(user, model.Password);
                await userManager.UpdateAsync(user);
                return RedirectToAction("Index", "Orders");
            } else {
                foreach (var error in result.Errors) {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }
        } else {
            ModelState.AddModelError(string.Empty, "Пользователь не найден");
        }
    }
}

```

```

        return RedirectToAction("Index", "Orders");
    }
}

```

#### Файл AdminController.cs

```

[Authorize(Roles = RoleService.AdminRole)]
public class AdminController : Controller {
    private readonly UserManager<User> userManager;
    private readonly ApplicationDbContext context;
    private readonly RoleService roleService;

    public AdminController(UserManager<User> userManager, ApplicationDbContext context, RoleService
roleService) {
        this.userManager = userManager;
        this.context = context;
        this.roleService = roleService;
    }

    public async Task<IActionResult> Index() {
        var currentUserId = context.Users.Where(u => u.UserName == User.Identity.Name).FirstOrDefault();
        ViewBag.CurrentUserId = currentUserId.Id;
        IEnumerable<UserViewModel> users = context.Users.ToList().Select(u => new UserViewModel() {
            Role = roleService.GetRole(u.Id),
            Id = u.Id,
            Email = u.Email
        });

        return View(users);
    }

    [HttpGet]
    public IActionResult Create() {
        ViewData["Roles"] = new SelectList(context.Roles, "Name");
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(UserCreateViewModel user) {
        if (ModelState.IsValid) {
            if (await userManager.FindByNameAsync(user.Email) == null) {
                User identityUser = new User { Email = user.Email, UserName = user.Email, Surname = user.Surname,
Name = user.Name, Patronymic = user.Patronymic };
                IdentityResult result = await userManager.CreateAsync(identityUser, user.Password);

                if (result.Succeeded) {
                    await roleService.AddRolesAsync(identityUser, user.Role);
                    return RedirectToAction(nameof(Index));
                } else foreach (var error in result.Errors)
                    ModelState.AddModelError(string.Empty, error.Description);

            } else ModelState.AddModelError(string.Empty, "Этот пользователь уже существует");

            } else ModelState.AddModelError(string.Empty, "Ошибка валидации страницы");
        ViewData["Roles"] = new SelectList(context.Roles, "Name");
        return View(user);
    }

    public async Task<IActionResult> Delete(string? id) {
        if (id == null)

```

```

        return NotFound();

        User user = await userManager.FindByIdAsync(id);

        if (user == null)
            return NotFound();

        return View(new DeleteUserViewModel { Id = user.Id, Email = user.Email, Role =
roleService.GetRole(user.Id) });
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(string id) {
        User user = await userManager.FindByIdAsync(id);
        if (user != null) {
            try
            {
                var userOrdersPointStatuses = context.OrdersPointStatuses.Where(a => a.ExecutorId == id && a.PointId
!= 103)
                    .ToList();

                if(userOrdersPointStatuses.Count > 0)
                {
                    ModelState.AddModelError("", "Вы не можете удалить этого пользователя, так как у этого
пользователя есть не завершенные заказы !!!");
                    return View("Delete", new DeleteUserViewModel { Id = user.Id, Email = user.Email, Role =
roleService.GetRole(user.Id) });
                }

                IdentityResult result = await userManager.DeleteAsync(user);
            }
            catch
            {
            }
        }
        return RedirectToAction(nameof(Index));
    }
}

```

#### Файл OrdersController.cs

```

[Authorize(Roles = $"{RoleService.MechanicRole},{RoleService.AutoElectricianRole}," +
    $"{RoleService.WalkerRole},{RoleService.PainterRole},{RoleService.MotoristRole}")]
public class OrdersController : Controller
{
    private readonly ApplicationDbContext _context;
    private readonly UserManager<User> _userManager;
    private readonly RoleService _roleService;

    public OrdersController(ApplicationDbContext context, UserManager<User> userManager, RoleService
roleService)
    {
        _context = context;
        _userManager = userManager;
        _roleService = roleService;
    }

    // GET: Orders
    public IActionResult Index(long? pointId = null)

```

```

{
    Helper.SessionExtensions.Set(HttpContext.Session, "pointId", pointId);

    ViewBag.pointId = pointId;

    return View();
}

[HttpPost]
public IActionResult LoadTable([FromBody] DtParameters dtParameters)
{
    int dtdraw = dtParameters.draw;
    int startRec = dtParameters.start;
    int pageSize = dtParameters.length;

    var entities = _context.Orders.Select(a => a);

    var userId = _userManager.GetUserId(User);
    var userRole = _roleService.GetRole(userId);

    if(userRole != RoleService.AdminRole)
    {
        var ordersIds = _context.OrdersPointStatuses.Where(a => a.ExecutorId == userId).Select(a => a.OrdersId);
        entities = entities.Where(a => ordersIds.Contains(a.Id));
    }

    var totalResultsCount = entities.Count();

    var pointId = Helper.SessionExtensions.Get<long?>(HttpContext.Session, "pointId");

    if (pointId != null && userRole == RoleService.AdminRole)
    {
        entities = entities.Where(a => a.PointId == pointId);
    }
    else if(pointId != null)
    {
        var ordersIds = _context.OrdersPointStatuses.Where(a => a.ExecutorId == userId && a.PointId ==
pointId).Select(a => a.OrdersId);
        entities = entities.Where(a => ordersIds.Contains(a.Id));
    }

    var data = entities
        .OrderByDescending(a => a.CreatedAt);

    var filteredResultsCount = data.Count();
    var _data = data.Skip(startRec).Take(pageSize)
        .ToList()
        .Select(a => new OrdersViewModel
        {
            Id = a.Id,
            ClientFIO = a.ClientFIO,
            ClientPhoneNumber = a.ClientPhoneNumber,
            CarBrand = a.CarBrand,
            CarModel = a.CarModel,
            RepairInstructions = a.RepairInstructions,
            Price = a.Price,
            RoadMaps = (userRole == RoleService.AdminRole) ? GetRoadMapsForAdmin(a.PointId, a.Id) :
GetRoadMaps(a.Id, userId)
        });

    return Json(new
    {

```



```

        draw = dt.draw,
        recordsTotal = totalResultsCount,
        recordsFiltered = filteredResultsCount,
        data = _data
    });
}

[HttpGet("Orders/Movement")]
public async Task<IActionResult> Movement(long id, long roadMapId)
{
    var movement = new MovementViewModel();

    var roadMap = _context.RoadMaps.Find(roadMapId);

    if (roadMap == null)
    {
        return NotFound(roadMapId);
    }

    movement.EntityId = id;
    movement.RoadMapId = roadMapId;

    var user = await _userManager.GetUserAsync(User);
    var userRole = _roleService.GetRole(user.Id);

    if (userRole == RoleService.AdminRole)
    {
        movement.IsAdmin = true;

        var userIds = _context.OrdersPointStatuses.Where(a => a.OrdersId == id && a.PointId ==
roadMap.SourcePointId)
            .Select(a => a.ExecutorId)
            .ToList();

        var users = _userManager.Users.Where(a => userIds.Contains(a.Id)).ToList()
            .Select(a => new { Id = a.Id, FullName = $"{a.Email} : {a.Surname} {a.Name} {a.Patronymic}", Role =
_userManager.GetRolesAsync(a).Result.First() })
            .ToList();

        ViewData["ExecutorsId"] = new SelectList(users, "Id", "FullName");
    }
    else
    {
        movement.UserId = user.Id;
        movement.User = $"{user.Email} : {user.Surname} {user.Name} {user.Patronymic}";
    }

    if (roadMap.ReasonId == 1002) //добавить запчасти
    {
        movement.IsAddSpareParts = true;
        movement.SpareParts = _context.Orders.Find(id).SpareParts;
    }

    return View(movement);
}

[HttpPost("Orders/Movement")]
public async Task<IActionResult> Movement(MovementViewModel movement)
{
    var roadMap = await _context.RoadMaps.FindAsync(movement.RoadMapId);

    var orders = await _context.Orders.FindAsync(movement.EntityId);

```

```

var ordersPointStatus = await _context.OrdersPointStatuses.FindAsync(movement.EntityId, movement.UserId);

ordersPointStatus.PointId = roadMap.TargetPointId;

await _context.SaveChangesAsync();

var ordersPoint = _context.OrdersPointStatuses
    .Where(a => a.OrdersId == movement.EntityId)
    .OrderByDescending(a => a.Order)
    .Select(a => a.PointId)
    .First();

orders.PointId = ordersPoint;

if (movement.IsAddSpareParts)
{
    orders.SpareParts = movement.SpareParts;
}

await _context.SaveChangesAsync();

return RedirectToAction("Index");
}

// GET: Orders/Details/5
public async Task<IActionResult> Details(long? id)
{
    if (id == null || _context.Orders == null)
    {
        return NotFound();
    }

    var orders = await _context.Orders
        .Include(a => a.Mechanic)
        .Include(a => a.AutoElectrician)
        .Include(a => a.Walker)
        .Include(a => a.Painter)
        .Include(a => a.Motorist)
        .FirstOrDefaultAsync(m => m.Id == id);

    if (orders == null)
    {
        return NotFound();
    }

    var res = new OrdersDetailViewModel
    {
        ClientFIO = orders.ClientFIO,
        ClientPhoneNumber = orders.ClientPhoneNumber,
        RepairInstructions = orders.RepairInstructions,
        Price = orders.Price,
        Comment = orders.Comment,
        SpareParts = orders.SpareParts,
        CreatedAt = orders.CreatedAt,
        Mechanic = $"{orders.Mechanic?.Email} : {orders.Mechanic?.Surname} {orders.Mechanic?.Name} {orders.Mechanic?.Patronymic}",
        AutoElectrician = $"{orders.AutoElectrician?.Email} : {orders.AutoElectrician?.Surname} {orders.AutoElectrician?.Name} {orders.AutoElectrician?.Patronymic}",
        Walker = $"{orders.Walker?.Email} : {orders.Walker?.Surname} {orders.Walker?.Name} {orders.Walker?.Patronymic}",
        Painter = $"{orders.Painter?.Email} : {orders.Painter?.Surname} {orders.Painter?.Name} {orders.Painter?.Patronymic}",
    };
}

```

```

        Motorist = $"{orders.Motorist?.Email} : {orders.Motorist?.Surname} {orders.Motorist?.Name}
{orders.Motorist?.Patronymic}",
    };

    return View(res);
}

// GET: Orders/Create
[Authorize(Roles = RoleService.AdminRole)]
public IActionResult Create()
{
    var users = _userManager.Users
        .ToList()
        .Where(a => !_userManager.GetRolesAsync(a).Result.Any(r => r.Equals(RoleService.AdminRole)))
        .Select(a => new { Id = a.Id, FullName = $"{a.Email} : {a.Surname} {a.Name} {a.Patronymic}", Role
= _userManager.GetRolesAsync(a).Result.First()});

    var mechanics = users.Where(a => a.Role == RoleService.MechanicRole).ToList();
    var autoElectricians = users.Where(a => a.Role == RoleService.AutoElectricianRole).ToList();
    var walkers = users.Where(a => a.Role == RoleService.WalkerRole).ToList();
    var painters = users.Where(a => a.Role == RoleService.PainterRole).ToList();
    var motorists = users.Where(a => a.Role == RoleService.MotoristRole).ToList();

    mechanics.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    autoElectricians.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    walkers.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    painters.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    motorists.Add(new { Id = "", FullName = "Выбрать", Role = "" });

    ViewData["MechanicId"] = new SelectList(mechanics.OrderBy(a => a.Id), "Id", "FullName");
    ViewData["AutoElectricianId"] = new SelectList(autoElectricians.OrderBy(a => a.Id), "Id", "FullName");
    ViewData["WalkerId"] = new SelectList(walkers.OrderBy(a => a.Id), "Id", "FullName");
    ViewData["PainterId"] = new SelectList(painters.OrderBy(a => a.Id), "Id", "FullName");
    ViewData["MotoristId"] = new SelectList(motorists.OrderBy(a => a.Id), "Id", "FullName");

    return View();
}

// POST: Orders/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[Authorize(Roles = RoleService.AdminRole)]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(OrdersModifyViewModel ordersViewModel)
{
    if (ordersViewModel.MechanicId == null && ordersViewModel.AutoElectricianId == null &&
        ordersViewModel.WalkerId == null && ordersViewModel.PainterId == null &&
ordersViewModel.MotoristId == null)
    {
        ModelState.AddModelError("IsExecutorSelected", "Выберите хотя бы одного сотрудника");
    }
    if (ModelState.IsValid)
    {
        var orders = new Orders
        {
            ClientFIO = ordersViewModel.ClientFIO,
            ClientPhoneNumber = ordersViewModel.ClientPhoneNumber,
            CarBrand = ordersViewModel.CarBrand,
            I = ordersViewModel.CarModel,
            RepairInstructions = ordersViewModel.RepairInstructions,
            Price = ordersViewModel.Price,
            Comment = ordersViewModel.Comment,

```

```

        MechanicId = ordersViewModel.MechanicId,
        AutoElectricianId = ordersViewModel.AutoElectricianId,
        WalkerId = ordersViewModel.WalkerId,
        PainterId = ordersViewModel.PainterId,
        MotoristId = ordersViewModel.MotoristId,
        CreatedAt = DateTime.UtcNow,
        CreatedBy = _userManager.GetUserId(User)
    };

    _context.Add(orders);
    await _context.SaveChangesAsync();

    byte order = 1;
    if(ordersViewModel.MechanicId != null)
    {
        var ordersPointStatus = new OrdersPointStatus
        {
            OrdersId = orders.Id,
            ExecutorId = ordersViewModel.MechanicId,
            Order = order
        };

        _context.OrdersPointStatuses.Add(ordersPointStatus);
        order++;
    }
    if(ordersViewModel.AutoElectricianId != null)
    {
        var ordersPointStatus = new OrdersPointStatus
        {
            OrdersId = orders.Id,
            ExecutorId = ordersViewModel.AutoElectricianId,
            Order = order
        };

        _context.OrdersPointStatuses.Add(ordersPointStatus);
        order++;
    }
    if(ordersViewModel.WalkerId != null)
    {
        var ordersPointStatus = new OrdersPointStatus
        {
            OrdersId = orders.Id,
            ExecutorId = ordersViewModel.WalkerId,
            Order = order
        };

        _context.OrdersPointStatuses.Add(ordersPointStatus);
        order++;
    }
    if(ordersViewModel.PainterId != null)
    {
        var ordersPointStatus = new OrdersPointStatus
        {
            OrdersId = orders.Id,
            ExecutorId = ordersViewModel.PainterId,
            Order = order
        };

        _context.OrdersPointStatuses.Add(ordersPointStatus);
        order++;
    }
    if(ordersViewModel.MotoristId != null)

```

```

    {
        var ordersPointStatus = new OrdersPointStatus
        {
            OrdersId = orders.Id,
            ExecutorId = ordersViewModel.MotoristId,
            Order = order
        };

        _context.OrdersPointStatuses.Add(ordersPointStatus);
        order++;
    }
    await _context.SaveChangesAsync();

    return RedirectToAction(nameof(Index));
}
var users = _userManager.Users
    .ToList()
    .Where(a => !_userManager.GetRolesAsync(a).Result.Any(r => r.Equals(RoleService.AdminRole)))
    .Select(a => new { Id = a.Id, FullName = $"{a.Email} : {a.Surname} {a.Name} {a.Patronymic}",
Role = _userManager.GetRolesAsync(a).Result.First() });

var mechanics = users.Where(a => a.Role == RoleService.MechanicRole).ToList();
var autoElectricians = users.Where(a => a.Role == RoleService.AutoElectricianRole).ToList();
var walkers = users.Where(a => a.Role == RoleService.WalkerRole).ToList();
var painters = users.Where(a => a.Role == RoleService.PainterRole).ToList();
var motorists = users.Where(a => a.Role == RoleService.MotoristRole).ToList();

mechanics.Add(new { Id = "", FullName = "Выбрать", Role = "" });
autoElectricians.Add(new { Id = "", FullName = "Выбрать", Role = "" });
walkers.Add(new { Id = "", FullName = "Выбрать", Role = "" });
painters.Add(new { Id = "", FullName = "Выбрать", Role = "" });
motorists.Add(new { Id = "", FullName = "Выбрать", Role = "" });

ViewData["MechanicId"] = new SelectList(mechanics.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.MechanicId);
ViewData["AutoElectricianId"] = new SelectList(autoElectricians.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.AutoElectricianId);
ViewData["WalkerId"] = new SelectList(walkers.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.WalkerId);
ViewData["PainterId"] = new SelectList(painters.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.PainterId);
ViewData["MotoristId"] = new SelectList(motorists.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.MotoristId);

return View(ordersViewModel);
}

// GET: Orders/Edit/5
[Authorize(Roles = RoleService.AdminRole)]
public async Task<IActionResult> Edit(long? id)
{
    if (id == null || _context.Orders == null)
    {
        return NotFound();
    }

    var orders = await _context.Orders.FirstOrDefaultAsync(a => a.Id == id);
    if (orders == null)
    {
        return NotFound();
    }

    var res = new OrdersModifyViewModel

```

```

    {
        ClientFIO = orders.ClientFIO,
        ClientPhoneNumber = orders.ClientPhoneNumber,
        CarBrand = I,
        I = I,
        RepairInstructions = orders.RepairInstructions,
        Price = orders.Price,
        Comment = orders.Comment,
        MechanicId = orders.MechanicId,
        AutoElectricianId = orders.AutoElectricianId,
        WalkerId = orders.WalkerId,
        PainterId = orders.PainterId,
        MotoristId = orders.MotoristId,
    };

    var users = _userManager.Users
        .ToList()
        .Where(a => !_userManager.GetRolesAsync(a).Result.Any(r => r.Equals(RoleService.AdminRole)))
        .Select(a => new { Id = a.Id, FullName = $"{a.Email} : {a.Surname} {a.Name} {a.Patronymic}" },
    Role = _userManager.GetRolesAsync(a).Result.First() );

    var mechanics = users.Where(a => a.Role == RoleService.MechanicRole).ToList();
    var autoElectricians = users.Where(a => a.Role == RoleService.AutoElectricianRole).ToList();
    var walkers = users.Where(a => a.Role == RoleService.WalkerRole).ToList();
    var painters = users.Where(a => a.Role == RoleService.PainterRole).ToList();
    var motorists = users.Where(a => a.Role == RoleService.MotoristRole).ToList();

    mechanics.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    autoElectricians.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    walkers.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    painters.Add(new { Id = "", FullName = "Выбрать", Role = "" });
    motorists.Add(new { Id = "", FullName = "Выбрать", Role = "" });

    ViewData["MechanicId"] = new SelectList(mechanics.OrderBy(a => a.Id), "Id", "FullName",
orders.MechanicId);
    ViewData["AutoElectricianId"] = new SelectList(autoElectricians.OrderBy(a => a.Id), "Id", "FullName",
orders.AutoElectricianId);
    ViewData["WalkerId"] = new SelectList(walkers.OrderBy(a => a.Id), "Id", "FullName", orders.WalkerId);
    ViewData["PainterId"] = new SelectList(painters.OrderBy(a => a.Id), "Id", "FullName", orders.PainterId);
    ViewData["MotoristId"] = new SelectList(motorists.OrderBy(a => a.Id), "Id", "FullName", orders.MotoristId);
    ViewBag.Id = orders.Id;

    return View(res);
}

// POST: Orders/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[Authorize(Roles = RoleService.AdminRole)]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Edit(long id, OrdersModifyViewModel ordersViewModel)
{
    if (id != ordersViewModel.Id)
    {
        return NotFound();
    }

    if (ordersViewModel.MechanicId == null && ordersViewModel.AutoElectricianId == null &&
        ordersViewModel.WalkerId == null && ordersViewModel.PainterId == null &&
ordersViewModel.MotoristId == null)
    {
        ModelState.AddModelError("IsExecutorSelected", "Выберите хотя бы одного сотрудника");
    }
}

```

```

    }
    if (ModelState.IsValid)
    {
        try
        {
            var orders = await _context.Orders.FirstAsync(a => a.Id == id);

            orders.ClientFIO = ordersViewModel.ClientFIO;
            orders.ClientPhoneNumber = ordersViewModel.ClientPhoneNumber;
            I = ordersViewModel.CarBrand;
            I = ordersViewModel.CarModel;
            orders.RepairInstructions = ordersViewModel.RepairInstructions;
            orders.Price = ordersViewModel.Price;
            orders.Comment = ordersViewModel.Comment;
            orders.MechanicId = ordersViewModel.MechanicId;
            orders.AutoElectricianId = ordersViewModel.AutoElectricianId;
            orders.WalkerId = ordersViewModel.WalkerId;
            orders.PainterId = ordersViewModel.PainterId;
            orders.MotoristId = ordersViewModel.MotoristId;

            _context.Orders.Update(orders);

            var ordersPointStatuses = _context.OrdersPointStatuses.Where(a => a.OrdersId == id);
            _context.OrdersPointStatuses.RemoveRange(ordersPointStatuses);

            byte order = 1;
            if (ordersViewModel.MechanicId != null)
            {
                var ordersPointStatus = new OrdersPointStatus
                {
                    OrdersId = orders.Id,
                    ExecutorId = ordersViewModel.MechanicId,
                    Order = order
                };

                _context.OrdersPointStatuses.Add(ordersPointStatus);
                order++;
            }
            if (ordersViewModel.AutoElectricianId != null)
            {
                var ordersPointStatus = new OrdersPointStatus
                {
                    OrdersId = orders.Id,
                    ExecutorId = ordersViewModel.AutoElectricianId,
                    Order = order
                };

                _context.OrdersPointStatuses.Add(ordersPointStatus);
                order++;
            }
            if (ordersViewModel.WalkerId != null)
            {
                var ordersPointStatus = new OrdersPointStatus
                {
                    OrdersId = orders.Id,
                    ExecutorId = ordersViewModel.WalkerId,
                    Order = order
                };

                _context.OrdersPointStatuses.Add(ordersPointStatus);
                order++;
            }
        }
    }
}

```

```

        if (ordersViewModel.PainterId != null)
        {
            var ordersPointStatus = new OrdersPointStatus
            {
                OrdersId = orders.Id,
                ExecutorId = ordersViewModel.PainterId,
                Order = order
            };

            _context.OrdersPointStatuses.Add(ordersPointStatus);
            order++;
        }
        if (ordersViewModel.MotoristId != null)
        {
            var ordersPointStatus = new OrdersPointStatus
            {
                OrdersId = orders.Id,
                ExecutorId = ordersViewModel.MotoristId,
                Order = order
            };

            _context.OrdersPointStatuses.Add(ordersPointStatus);
            order++;
        }

        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!OrdersExists(ordersViewModel.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return RedirectToAction(nameof(Index));
}
var users = _userManager.Users
    .ToList()
    .Where(a => !_userManager.GetRolesAsync(a).Result.Any(r => r.Equals(RoleService.AdminRole)))
    .Select(a => new { Id = a.Id, FullName = $"{a.Email} : {a.Surname} {a.Name} {a.Patronymic}",
        Role = _userManager.GetRolesAsync(a).Result.First() });

var mechanics = users.Where(a => a.Role == RoleService.MechanicRole).ToList();
var autoElectricians = users.Where(a => a.Role == RoleService.AutoElectricianRole).ToList();
var walkers = users.Where(a => a.Role == RoleService.WalkerRole).ToList();
var painters = users.Where(a => a.Role == RoleService.PainterRole).ToList();
var motorists = users.Where(a => a.Role == RoleService.MotoristRole).ToList();

mechanics.Add(new { Id = "", FullName = "Выбрать", Role = "" });
autoElectricians.Add(new { Id = "", FullName = "Выбрать", Role = "" });
walkers.Add(new { Id = "", FullName = "Выбрать", Role = "" });
painters.Add(new { Id = "", FullName = "Выбрать", Role = "" });
motorists.Add(new { Id = "", FullName = "Выбрать", Role = "" });

ViewData["MechanicId"] = new SelectList(mechanics.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.MechanicId);

```



```

        ViewData["AutoElectricianId"] = new SelectList(autoElectricians.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.AutoElectricianId);
        ViewData["WalkerId"] = new SelectList(walkers.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.WalkerId);
        ViewData["PainterId"] = new SelectList(painters.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.PainterId);
        ViewData["MotoristId"] = new SelectList(motorists.OrderBy(a => a.Id), "Id", "FullName",
ordersViewModel.MotoristId);
        ViewBag.Id = id;

        return View(ordersViewModel);
    }

// Delete: Orders/Delete?Id=5
[Authorize(Roles = RoleService.AdminRole)]
[HttpDelete]
public async Task Delete(long id)
{
    try
    {
        var order = await _context.Orders.FindAsync(id);
        _context.Orders.Remove(order);

        var ordersPointStatuses = _context.OrdersPointStatuses.Where(a => a.OrdersId == id);
        _context.OrdersPointStatuses.RemoveRange(ordersPointStatuses);

        await _context.SaveChangesAsync();
    }
    catch
    {
        throw;
    }
}

private bool OrdersExists(long id)
{
    return (_context.Orders?.Any(e => e.Id == id)).GetValueOrDefault();
}

private List<RoadMapViewModel> GetRoadMaps(long entityId, string userId)
{
    var res = new List<RoadMapViewModel>();

    var ordersPointStatus = _context.OrdersPointStatuses.Find(entityId, userId);

    if (ordersPointStatus.Order != 1)
    {
        var previousOrdersPointStatus = _context.OrdersPointStatuses.First(a => a.OrdersId == entityId && a.Order
== (ordersPointStatus.Order - 1));
        if (previousOrdersPointStatus.PointId == 103)
        {
            res = _context.RoadMaps.Where(a => a.SourcePointId == ordersPointStatus.PointId)
.Include(a => a.Reason)
.Select(a => new RoadMapViewModel
{
    Id = a.Id,
    EntityId = entityId,
    Reason = a.Reason
})
.ToList();
        }
    }
}

```

```

else
{
    res = _context.RoadMaps.Where(a => a.SourcePointId == ordersPointStatus.PointId)
        .Include(a => a.Reason)
        .Select(a => new RoadMapViewModel
        {
            Id = a.Id,
            EntityId = entityId,
            Reason = a.Reason
        })
        .ToList();
}

return res;
}
private List<RoadMapViewModel> GetRoadMapsForAdmin(long? pointId, long entityId)
{
    var res = new List<RoadMapViewModel>();

    if (pointId.HasValue)
    {
        var ordersPointStatus = from ops in _context.OrdersPointStatuses
                                where ops.OrdersId == entityId
                                group ops by ops.PointId into g
                                select g.Key;

        res = _context.RoadMaps.Where(a => ordersPointStatus.Contains(a.SourcePointId))
            .Include(a => a.Reason)
            .Select(a => new RoadMapViewModel
            {
                Id = a.Id,
                EntityId = entityId,
                Reason = a.Reason
            })
            .ToList();
    }

    return res;
}
}

```