

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»
Специальность 1-40 05 01-01 Информационные системы и технологии (в
проектировании и производстве)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Разработка приложений баз данных для информационных
систем»

на тему: **«СЕРВИСНЫЙ ЦЕНТР ПО РЕМОНТУ ОБОРУДОВАНИЯ»**

Исполнитель: студент гр. ИТП–31
Трофимов Е. В.

Руководитель: старший преподаватель
Асенчик О.Д.

Дата проверки: _____

Дата допуска к защите: _____

Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2019

Установа адукацыі «Гомельскі дзяржаўны
тэхнічны ўніверсітэт імя П.В. Сухого»
Факультэт аўтаматызаваных і інфармацыйных сістэм

Кафедра _____

РЭЦЭНЗИЯ
на курсавы праект (работу)

па дысцыпліне _____ ,

выканана студэнтам _____

групы _____

I. Пералік заўваг па тэксту курсавога праекта (работы)

II. Агульная характарыстыка работы

СОДЕРЖАНИЕ

| | |
|---|----|
| Введение | 4 |
| 1 Логическая и физическая структура базы данных | 6 |
| 1.1 Информационно-логическая модель информационной системы | 6 |
| 1.2 Физическая модель базы данных | 10 |
| 1.3 Файловая структура базы данных | 11 |
| 2 Аппаратное и программное обеспечение информа- ционной системы | 13 |
| 2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных | 13 |
| 2.2 Требования к системному и прикладному программному обеспечению на стороне <i>web</i> -сервера | 13 |
| 2.3 Требования к системному и прикладному программному обеспечению на стороне клиента..... | 13 |
| 2.4 Настройка и развёртывание приложения на сервере | 14 |
| 3.3 Описание контроллеров | 16 |
| 3.4 Описание представлений..... | 18 |
| 4 Руководство пользователя..... | 19 |
| 4.2 Назначение, условие применения и функционал | 19 |
| 4.3 Подготовка к работе | 19 |
| 4.4 Описание операции по обработки данных | 20 |
| 5 Руководство программиста | 23 |
| 5.1 Назначения и условия применения программы..... | 23 |
| 5.2 Характеристики программы | 23 |
| 5.3 Сопровождение программного комплекса..... | 23 |
| 5.4 Входные и выходные данные | 24 |
| 5.5 Сообщения в ходе работы приложения | 24 |
| Заключение | 25 |
| Список используемых источников..... | 26 |
| Приложение А – Код программы | 27 |
| Приложение Б – Чертёж структуры <i>web</i> -приложения | 74 |

ВВЕДЕНИЕ

Данная курсовой проект посвящён разработке *web*-приложения баз данных сервисного центра по ремонту оборудования, создание интерфейса в виде набора *web*-страниц, обеспечивающих отображение и редактирование информации из базы данных, для автоматизации работы со структурированной информацией сервисного центра.

Наиболее используемым типом информационной системы является клиент-серверная система. Данный тип системы представляет собой взаимодействие структурных компонентов, где структурными компонентами являются сервер и узлы-поставщики определённых сервисов, а также клиенты, которые пользуются данным сервисом. Данный тип системы наиболее часто используется в создании корпоративных баз данных, в которой база данных является главным элементом, а все необходимые операции с базой выполняются сервером. Запросы на получение и изменение информации из базы данных отправляют клиенты. Сервер обрабатывает запросы и возвращает ответ клиенту. Преимуществом такой системы является её достаточно высокий уровень производительности за счёт распределения вычислительной нагрузки между клиентом и сервером, а также непротиворечивость данных за счёт централизованной обработки.

Задачей курсового проекта является проектирование и создание базы данных в выбранной СУБД и разработка веб-приложения, которое обеспечивает отображение, редактирование и обработку информации из разработанной базы данных. Структура базы данных должна быть нормализована – таблицы базы данных должны удовлетворять требованиям третьей нормальной формы. База данных должна содержать тестовый набор данных (не менее 100 записей у таблицы на стороне отношения «один» и не менее 10000 записей у таблицы на стороне отношения «многие»).

Для данной задачи определены следующие исходные данные:

- запчасти (наименование, функции, цена, модель оборудования);
- ремонтируемые модели (наименование, тип, производитель, технические характеристики, особенности);
- виды неисправностей (модель оборудования, наименование, методы ремонта, перечень использованных запчастей, цена работы);
- обслуживаемые магазины (наименование, адрес, телефон);
- заказы (дата заказа, серийный номер и дата возврата отремонтированного оборудования, ФИО заказчика, наименование неисправности, магазин, отметка о гарантии, срок гарантии ремонта, цена, сотрудник);
- виды неисправностей (модель оборудования, наименование, методы ремонта, перечень использованных запчастей, цена работы).

Также прилагаются следующие дополнительные требования к отображению данных:

- отдел кадров (данные обо всех сотрудниках и их должностях, отображение сведений о сотрудниках с определенными: ФИО, стажем работы);
- список неисправностей (отображение полных сведений по всем неисправностям, отображение сведений по неисправностям с определенными: моделью оборудования, наименованием, методом ремонта, клиентом);
- список заказов (отображение полных сведений по всем заказам; отображение сведений по заказам с определенными: датой, ФИО заказчика, сотрудником, отображение гарантийных и не гарантийных заказов).

Для решения поставленной задачи в качестве СУБД используется *MS SQL Server*. Данная СУБД обеспечивает поддержку баз данных очень большого объема и обработку сложных запросов, а также имеет эффективные алгоритмы для работы с памятью и автоматизированным контролем размера файлов баз данных. В качестве технологии для разработки веб-приложения используется платформа *ASP.NET Core MVC* [3, с. 105]. Данная платформа является многофункциональной платформой для создания веб-приложений с помощью шаблона проектирования *Model-View-Controller* (модель-контроллер-представление). Структура *MVC* предполагает разделение приложения на три основных компонента: модель, представление и контроллер [6]. Каждый компонент решает свои задачи и взаимодействует с другими компонентами. Т.е. данный шаблон проектирования позволяет разделить задачи для каждого компонента, позволяет разрабатывать проект в команде, разделяя задачи между участниками и обеспечивает дальнейшую масштабируемость проекта. Благодаря такой схеме связей и распределения обязанностей между компонентами процесс масштабирования приложения становится проще, т.к. облегчается процесс написания кода, выполнения отладки и тестирования компонентов. Для доступа к данным используется технология *Entity Framework Core*. Данная технология является *ORM (object-relational mapping* – отображение данных на реальные объекты) инструментом, т.е. она позволяет работать с реляционными данными, используя классы и их иерархии. Также основным преимуществом данной технологии является использование универсального интерфейса для работы с данными, что позволяет легко и быстро сменить СУБД.

1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

1.1 Информационно-логическая модель информационной системы

Для решения задачи была сформирована структура и логика приложения. В первую очередь из исходных данных были выделены следующие сущности:

- «Ремонтируемое оборудование»;
- «Запчасти»;
- «Типы повреждения»;
- «Обслуживаемые магазины»;
- «Сотрудник»;
- «Должности»;
- «Заказы».

Для сущности «Ремонтируемая модель» было создано отношение (таблица) с атрибутами: «Код модели», «Наименование», «Тип», «Производитель», «Технические характеристики», «Особенности». Подробное описание отношения и атрибутов приведено в таблице 1.1. Данное отношение находится в первой нормальной форме.

Таблица 1.1 – Отношение описывающие сущность «Ремонтируемая модель»

| Атрибуты | Описание домена | Тип данных |
|----------------------------|---|-------------|
| Код модели | Уникальный инкрементируемый идентификатор для каждой ремонтируемой модели. Является первичным ключом. | Целое число |
| Наименования | Содержит наименований ремонтируемых моделей. | Строка |
| Тип | Содержит название типов ремонтируемых моделей. | Строка |
| Технические характеристики | Содержит технические характеристики ремонтируемых моделей. | Строка |
| Особенности | Содержит описание особенностей ремонтируемых моделей. | Строка |

Отношение для сущности «Запчасти», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «Код запчасти», «Наименование», «Функции», «Наименование ремонтируемой модели». Данное отношение следует привести к третьей нормальной форме, заменив атрибут «Наименование ремонтируемой модели» на атрибут «Код ремонтируемой модели» связав отношение «Запчасти» с отношением «Ремонтируемая модель». Так как по условию

задачи сущность «Виды неисправностей», описание сущности и атрибутов приведено в таблице 1.3, должна иметь список запчастей. В сущность «Запчасти» следует ещё добавить атрибут «Код вида неисправности», который будет являться внешним ключом. Тем самым будет организована связь один ко многим, между данными отношениями.

Таблица 1.2 – **Отношение описывающие сущность «Запчасти»**

| Атрибуты | Описание домена | Тип данных |
|--------------------------|---|--------------------------|
| Код запчасти | Уникальный инкрементируемый идентификатор для каждой запчасти. Является первичным ключом. | Целое число |
| Наименования | Содержит неуникальные наименования ремонтируемых моделей. | Строка |
| Функции | Содержит описание функции запчастей. | Строка |
| Цена | Содержит числовые значения с плавающей точкой стоимость запчастей. | Число с плавающей точкой |
| Код ремонтируемой модели | Содержит ссылки на код ремонтируемой модели. Является внешним ключом для связи с отношением «Ремонтируемые модели». | Целое число |
| Код вида неисправности | Содержит ссылки на код вида неисправности. Является внешним ключом для связи с отношением «Виды неисправностей». | Целое число |

Отношение для «Видов неисправностей», таблица 1.3, состоит из атрибутов: «Код вида неисправности», «Код модели оборудования», «Код модели оборудования», «Наименование», «Метод ремонта», «Цена работы», «Код неисправности». Связано отношением «Один ко многим» с отношениями относящимся к сущностям «Виды неисправностей» и «Ремонтируемые модели».

Таблица 1.3 – Отношение описывающие сущность «Виды неисправностей»

| Атрибуты | Описание домена | Тип данных |
|-------------------------|---|--------------------------|
| Код вида неисправности | Уникальный инкрементируемый идентификатор для каждой неисправности. Является первичным ключом. | Целое число |
| Код модели оборудования | Содержит код ремонтируемой модели. Является внешним ключом для связи с отношением «Ремонтируемые модели». | Целое число |
| Наименование | Содержит наименование ремонтируемых моделей. | Строка |
| Методы ремонта | Содержит описание методов ремонта неисправностей. | Строка |
| Цена работы | Содержит стоимость работы для устранения неисправности. Вычисляется относительно стоимости запчастей. | Число с плавающей точкой |
| Код вида неисправности | Содержит код вида неисправности. Является внешним ключом для связи с отношением «Виды неисправностей». | Целое число |

Сущность «Отдел кадров» была разбита на два отношения: «Сотрудники» и «Должности» и создана связь «Один ко многим» между отношениями. Описание отношений для сущностей «Обслуживаемые магазины», «Должности» и «Сотрудники» приведены в таблицах 1.4 – 1.6.

Таблица 1.4 – Отношение описывающие сущность «Обслуживаемые магазины»

| Атрибуты | Описание домена | Тип данных |
|-----------------|--|-------------------|
| Код магазина | Уникальный инкрементируемый идентификатор для каждого магазина. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование магазина. | Строка |
| Номер телефон | Содержит номер телефона обслуживаемого магазина. | Строка |

Таблица 1.5 – **Отношение описывающие сущность «Должности»**

| Атрибуты | Описание домена | Тип данных |
|-----------------|--|--------------------------|
| Код должности | Уникальный инкрементируемый идентификатор для каждой должности. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование должности. | Строка |
| Зарботная плата | Содержит заработную плату для сотрудников с указанной должностью. | Число с плавающей точкой |

Таблица 1.6 – **Отношение описывающие сущность «Сотрудники»**

| Атрибуты | Описание домена | Тип данных |
|-----------------|---|-------------------|
| Код сотрудника | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| ФИО | Содержит ФИО сотрудника. | Строка |
| Опыт работы | Содержит опыт работы сотрудника. | Целое число |
| Код должности | Содержит код вида должности сотрудника. Является внешним ключом для связи с отношением «Должности». | Целое число |

Сущность «Заказы» было реализовано отношением, таблица 1.7, связанное связью «Один ко многим» с отношениями «Обслуживаемые магазины», «Виды неисправностей» и «Сотрудники».

Таблица 1.6 – **Отношение описывающие сущность «Заказы»**

| Атрибуты | Описание домена | Тип данных |
|-----------------|--|-------------------|
| Код заказа | Уникальный инкрементируемый идентификатор для каждого заказа. Является первичным ключом. | Целое число |
| Дата заказа | Содержит дату заказа услуги по ремонту оборудования. | Дата |
| Дата возврата | Содержит дату возврата отремонтируемого оборудования. | Целое число |

Продолжение таблицы 1.6

| Атрибуты | Описание домена | Тип данных |
|------------------------|--|--------------------------|
| ФИО заказчика | Содержит ФИО заказчика услуги. | Строка |
| Код вида неисправности | Содержит код вида неисправности. Является внешним ключом для связи с отношением «Виды неисправностей». | Целое число |
| Дата магазина | Содержит код обслуживаемого магазина. Является внешним ключом для связи с отношением «Обслуживаемые магазины». | Целое число |
| Отметка о гарантии | Содержит отметку о гарантии. | Логическое значение |
| Срок гарантии | Содержит количество дней, в течении которого услуга подлежит гарантии. | Целое число |
| Цена | Содержит конечную стоимость услуги, вычисленную относительно стоимости работы. | Число с плавающей точкой |
| Код сотрудника | Содержит код сотрудника. Является внешним ключом для связи с отношением «Сотрудники». | Целое число |

После определения всех отношений и атрибутов, тем самым была составлена информационно-логическая модель информационной системы.

1.2 Физическая модель базы данных

По созданной информационно-логической модели была создана иерархия класса и контекст данных, приложение Б, которая описывает ранее созданные отношения атрибуты и домены, для каждого отношения был создан свой соответствующий класс и определены реляционные отношения между ими. Далее по подходу *Code First* с помощью средств *Entity Framework*, была сгенерирована база данных в СУБД *MS SQL Server*. После преобразования логической модели в физическую, в физической модели были получены таблицы со связями соответствующие каждой из ранее определённых отношений, диаграмма базы данных и связи между сгенерированными таблицами представлены на рисунке 1.1.

На рисунке 1.1 изображена сгенерированная база данных с помощью *Entity Framework*.

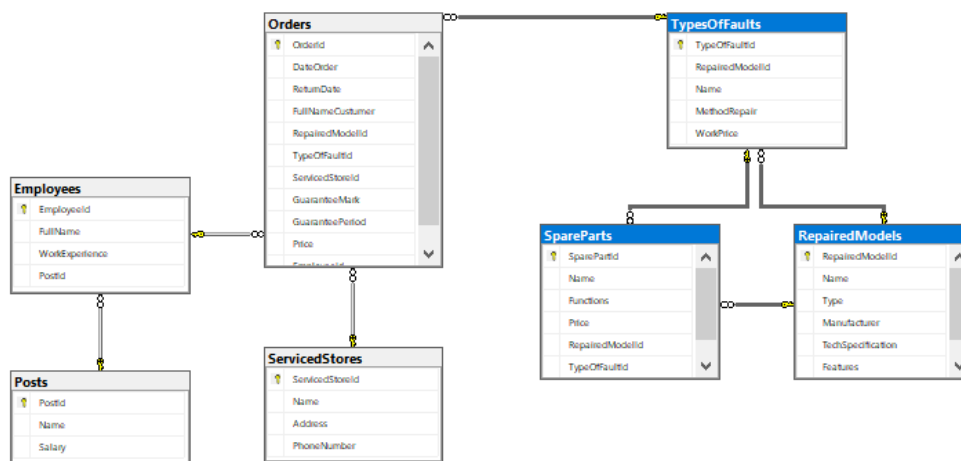


Рисунок 1.1 – Диаграмма базы данных

Для процесса преобразования логической модели в физическую существует несколько правил:

- сущности становятся таблицами в физической базе данных;
- атрибуты становятся столбцами в физической базе данных. Также для каждого столбца необходимо определить подходящий тип данных;
- уникальные идентификаторы становятся столбцами, не допускающими значение *NULL*, т.е. первичными ключами. Также значение идентификатора делается автоинкрементным для обеспечения уникальности;
- все отношения моделируются в виде внешних ключей.

1.3 Файловая структура базы данных

Все базы данных *MS SQL Server* имеют два основных рабочих системных файла: файл данных и файл журнала. Файлы данных содержат данные и объекты, такие как таблицы, индексы, хранимые процедуры и представления. Файлы журнала содержат сведения, необходимые для восстановления всех транзакций в базе данных.

Для каждого отношения были получены следующие таблицы: *Employees*, *Orders*, *Post*, *RepairedModels*, *ServicedStores*, *SpareParts*, *TypesOfFaults*.

Таблицы *Posts*, *RepairedModels* и *ServicedStores* находятся в отношении «один» и описывают сущности «Должности», «Ремонтируемые модели» и «Обслуживаемые магазины» и подобраны физические тип данных для соответствующих столбцов, установлены первичные ключи.

Таблицы *Employees*, *Orders*, *TypeOfFaults* и *SpareParts* находятся в отношении «многие», описывают сущности «Сотрудники», «Заказы», «Типы повреждения» и «Запчасти». Имеют автоинкрементируемый первичный ключ и внешние ключи для связи с таблицами в отношении «Один».

С помощью библиотеки *Entity Framework* было осуществлено взаимодействие языка программирования *C#* с физической моделью данных, который произвёл соотношения классов и таблиц, был создан контекст данных, с помощью которого можно осуществлять доступ непосредственно в коде приложения.

2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных

Для корректной работы аппаратного и программного обеспечения на стороне сервера хранилища данных, требуется соблюдения следующих условий:

- установленный *MS SQL Server*;
- для работы *MS SQL Server 2016* и выше, требуется *.NET Framework 4.6*;
- сетевое программное обеспечение;
- требуется как минимум 7 ГБ свободного места на диске (при увеличении размера базы данных, может потребоваться свободного места);
- минимальный объем оперативной памяти 1 ГБ;
- процессор x64 с тактовой частотой 1,4 ГГц;

Требования перечисленные выше являются минимальными и могут меняться относительно размера базы данных и требуемых задач.

2.2 Требования к системному и прикладному программному обеспечению на стороне web-сервера

Минимальные требования к аппаратному и программному обеспечению и корректной работы на нём, необходимо соблюдение следующих условий:

- процессор x86/x64 с тактовой частотой 1 ГГц;
- минимальный объем оперативной памяти 512 МБ;
- требуется как минимум 4,5 ГБ свободного места на диске;
- операционные системы *Windows 7, 8, 10, Linux, Mac OS*.

Так приложение разработана на платформе *.NET Core*, оно является кроссплатформенным и может быть запущено на любой поддерживаемой операционной системе. Для организации связи с СУБД требуется настроить подключение к нему. Так как СУБД может быть установлено на удалённом компьютере возможно потребуются подключение к интернету, либо к локальной сети, в которой находится сервер хранилища данных. Так же системные требования могут изменяться относительно масштаба приложения.

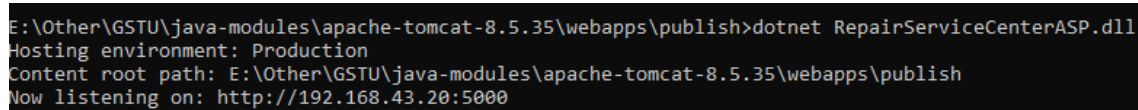
2.3 Требования к системному и прикладному программному обеспечению на стороне клиента

Чтобы приложение корректно работало на стороне клиента требуется браузера с поддержкой «*Bootstrap*» и наличие клиента и web-сервера в одной сети (локальной, глобальной).

2.4 Настройка и развёртывание приложения на сервере

Данное приложение может быть развёрнуто на серверах: *Apache Tomcat*, *Kestel*, *IIS*, *GlassFish* и др. Чтобы развернуть приложение, нужно перейти в папку с проектом и открыть командную строку и выполнить команду «*dotnet publish RepairServiceCenter -c Release*». После выполнении команды выходные данные приложения публикуется в папку «*./bin/Release/netcoreapp2.1/publish*» относительно директории проекта.

Для запуска приложения веб-приложение нужно скопировать папку «*publish*» в директорию с установленным веб-сервером (в случае *Tomcat* «*./webapp*») и выполнить команду «*dotnet RepairServiceCenterASP.dll*» с командной строки, рисунок 2.1, после этого веб-приложение будет запущено на сервере. Чтобы пользователь мог использовать веб-приложение, он должен находиться в одной сети с веб-сервером.



```
E:\Other\GSTU\java-modules\apache-tomcat-8.5.35\webapps\publish>dotnet RepairServiceCenterASP.dll
Hosting environment: Production
Content root path: E:\Other\GSTU\java-modules\apache-tomcat-8.5.35\webapps\publish
Now listening on: http://192.168.43.20:5000
```

Рисунок 2.1 – Запуск веб-приложения на веб-сервере

Чтобы подключиться к базе данных, требуется сконфигурировать подключение к ней. Для этого требуется отредактировать конфигурационный файл приложения «*appsetting.json*» и изменить строку подключение. Для того чтобы веб-приложению удалось установить соединение с базой данных, СУБД и веб-приложение должны находиться в одной сети [4].

3 СТРУКТУРА ПРИЛОЖЕНИЯ

3.1 Описание общей структуры веб-приложения

В состав данного веб-приложения входят три основных компонента: модель, представление и контроллер.

Модель представляет состояние приложения и бизнес-логику, непосредственно связанную с данными. Как правило, объекты моделей хранятся в базе данных. В архитектуре *MVC* модели представлены двумя основными типами: модели представлений, которые используются представлениями для отображения данных на веб-странице, и модели домена, описывающие логику управления данными. Модель содержит данные и хранит логику обработки этих данных, но не содержит логику взаимодействия с пользователем, т.е. с представлением.

Представление является графическим веб-интерфейсом, через который пользователь может взаимодействовать с приложением напрямую. Данный компонент содержит минимальную логику, которая связана с представлением данных.

Контроллер представляет центральный компонент архитектуры *MVC* для управления взаимодействием с пользователем, работы с моделью и выбора представления для отображения. Контроллер обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки введенных пользователем данных и логику формирования ответа пользователю. Контроллер является начальной отправной точкой в приложении и отвечает за выбор рабочих типов моделей и отображаемых представлений.

3.2 Описание классов для доступа к данным

Для работы с таблицами базы данных в приложении необходимы классы, которые описывают каждую таблицу. В данных классах описываются поля таблиц в виде свойств и связи между таблицами в виде связей между классами.

Классы *Employee*, *Order*, *Post*, *RepairedModel*, *ServicedStore*, *SparePart* и *TypeOfFault* описывают таблицы *Employees*, *Orders*, *Posts*, *RepairedModels*, *ServicedStores*, *SpareParts* и *TypeOfFaults* соответственно. Код данных классов представлен в приложении Б.

Свойства в каждом классе описывают столбцы соответствующей таблицы. В классах, описывающих таблицы, которые находятся на стороне отношения «многие», содержат ссылку на объект класса, моделирующего таблицу, связанную внешним ключом.

Также в данных классах используются аннотации – специальные атрибуты, которые определяют различные правила для отображения свойств модели. Для

задания параметров отображения свойства используется атрибут *Display*. Данный атрибут устанавливает заголовок свойства, который используется при отображении названия свойства в представлении. Для предоставления среде выполнения информации о типе свойства используется атрибут *DataType*. Также для проверки значений свойств применяются специальные атрибуты валидации – *Required*, *RegularExpression* и *Range*. Атрибут *Required* помечает, что свойство должно быть обязательно установлено. С помощью свойства *ErrorMessage* этого атрибута задаётся выводимое при валидации сообщение. Атрибут *RegularExpression* помечает, что значение свойства должно соответствовать указанному в этом атрибуте регулярному выражению. Атрибут *Range* определяет минимальное и максимальное ограничение для свойств с числовым типом данных. Аналогично атрибут *StringLength* определяет ограничения для свойств строкового типа.

3.3 Описание контроллеров

Контроллер представляет обычный класс, который наследуется от абстрактного базового класса *Microsoft.AspNetCore.Mvc.Controller*. Именование контроллеров строго предопределено, т.е. имя контроллера обязательно должно иметь суффикс «*Controller*», а остальная часть считается названием контроллера.

Адрес, который обрабатывается контроллерами, представлен в виде паттерна *{controller=[ControllerName]}/{action=[MethodName]}*, где *[ControllerName]* – название контроллера, *[MethodName]* – название метода контроллера.

Для работы с созданными моделями разработаны следующие контроллеры:

- *HomeController* – отвечает за вывод начальной страницы;
- *EmployeesController* – отвечает за работу с таблицей *Employees*;
- *TypeOfFaultControllers* – отвечает за работу с таблицей *TypesOfFaults*;
- *SparePartController* – отвечает за работу с таблицей *SpareParts*;
- *ServicedStoreController* – отвечает за работу с таблицей *ServicedStores*;
- *OrdersController* – отвечает за работу с таблицей *Orders*;
- *PostsControllers* – отвечает за работу с таблицей *Posts*.

Контроллеры, отвечающие за работу с таблицами, имеют следующие методы:

- *Index*;
- *Details[GET]*;
- *Create[GET]*;
- *Create[POST]*;
- *Edit[GET]*;
- *Edit[POST]*;
- *Delete[GET]*;

Метод *Index* в качестве входных параметров принимает значения, по которым производится фильтрация данных, флаг фильтра и номер страницы. Флаг фильтра указывает, являются ли входные значения фильтров новыми или нет. Если фильтры новые (т.е. они не применялись для фильтрации данных), то происходит выборка данных из базы данных, фильтрация с использованием входных значений фильтров, формирование ключа кеша и запись данных в кеш. Если входные фильтры использовались, то происходит формирование ключа кеша и получение данных из кеша по ключу. Сформированный ключ добавляется в список с ключами, а применяемые фильтры сохраняются в сессию. Данный метод возвращает объект класса *IndexViewModel<T>*, который содержит отфильтрованные данные, значения фильтров и объект класса *PageViewModel*, содержащий свойства и методы, необходимые для работы страничной навигации.

Метод *Details[GET]* принимает идентификатор записи, производит выборку нужной записи из определённой таблицы базы данных и возвращает объект, моделирующий эту таблицу и содержащий все данные из таблицы.

Метод *Create[GET]* возвращает одноимённое представление с полями для добавления записи в таблицу базы данных. Для таблиц, стоящих на стороне «многие» данный метод формирует словари *ViewData*, в которые добавляются необходимые данные из таблиц, стоящих на стороне отношения «один».

Метод *Create[POST]* вызывается при отправке результата формы создания записи. Данный метод принимает объект, таблицу которого он моделирует и содержит данные, которые необходимо записать в базу данных. Перед записью производится валидация данных. Если данные неверны, то формируется ошибка, которая выводится в представлении. Если данные верны, то происходит запись данных в базу и переход в метод *Index* текущего контроллера.

Метод *Edit[GET]* принимает идентификатор записи и производит выборку нужной записи из определённой таблицы базы данных. Если запись найдена, то происходит добавление необходимых данных из других таблиц в словари *ViewData* и возврат представления с формой редактирования записи. Если запись не найдена, то метод возвращает стандартное сообщение об ошибке.

Метод *Edit[POST]* вызывается при отправке результата формы редактирования записи. Данный метод в качестве входных параметров принимает идентификатор записи и объект, содержащий данные об этой записи. Если входной идентификатор и идентификатор объекта не совпадают, то метод возвращает стандартное сообщение об ошибке. Иначе метод выполняет валидацию входных данных и если данные верны, то производится обновление данных в базе. Если операция обновления прошла успешно, то происходит переход в метод *Index* текущего контроллера. В случае возникновения ошибки метод возвращает стандартное сообщение об ошибке.

3.4 Описание представлений

Представления – это файлы в формате *cshtml*, в которых используется язык разметки *HTML* и язык программирования *C#* в разметке *Razor*. Все представления объединяются в папки с именами, соответствующими названиям контроллеров. Все эти папки находятся в папке *Views* в корне приложения.

Для существующих контроллеров разработаны представления, которые содержатся следующих в папках:

- *Employees* – содержит представления для работы с данными о сотрудниках, для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

- *Posts* – содержит представления для работы с данными о должностях сотрудников;

- *RepairedModels* – содержит представления для работы с данными о ремонтируемых моделях;

- *ServicedStores* – содержит представления для работы с таблицей «Обслуживаемые магазины»;

- *SpareParts* – содержит представления для работы с таблицей «Запчасти»;

- *Home* – содержит представления для домашних веб-страниц;

- *Orders* – содержит представления для работы с заказами;

- *TypeOfFaults* – содержит представления для работы с типами повреждения.

Для каждого представления с выборкой данных был разработан класс «модель-представление» (*ViewModel*), данный класс нужен для создание постраничной навигации. Так же эти классы содержат объекты для дополнительной манипуляции с данными (фильтрации и сортировки). Так же некоторые данные выборки, которые редко редактируются и добавляются, кешируются в кеше браузера с помощью атрибута *ResponseCache* (кешируются *css* стили, *html* страничка) и с помощью интерфейса *IMemoryCache* (кешируются данные выборки). Для кеширования с помощью *IMemoryCache* были реализованы дополнительные классы, которые подключаются как «сервисы» и благодаря технологии «*Dependency Injection*» (внедрение зависимости) неявно передаются классом контроллерам [5].

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Введение

Данное *web*-приложение производит автоматизацию процесса ремонта оборудования и обслуживания различных предприятий, пользующихся услугами по ремонту оборудования.

4.2 Назначение, условие применения и функционал

Web-приложение предназначено для управления и учёта данных о заказах на обслуживание оборудования и управление данными сотрудников предприятия.

Основные функции приложения:

- добавление, просмотр и редактирования заказов;
- выборка, сортировка и фильтрация заказов по заданным критериям;
- добавление, просмотр и редактирования данных о сотрудниках;
- фильтрация, сортировка и выборка данных о сотрудниках и их должностях по заданным критериям;
- добавление, просмотр и редактирования данных о видах повреждения оборудования список требуемых запчастей для замены;
- фильтрация, сортировка и выборка видов повреждения по заданным критериям;
- добавление, просмотр и редактирования данных обслуживаемых магазинов;
- добавление, просмотр и редактирования данных о должностях сотрудников;
- автоматизация рабочего процесса;
- добавление, просмотр и редактирования данных о запчастях и их стоимости;

4.3 Подготовка к работе

Для использования приложение требуется веб-браузер (*Mozilla Firefox*, *Chrome*, *Opera*, *Microsoft Edge* и пр.) в адресной строке веб-браузера ввести *URL*-адрес выданный системным-администратором и нахождение устройства в той же локальной сети, где находится *web*-сервер (если сервер находится в глобальной сети, то подключение к интернету).

4.4 Описание операции по обработки данных

Для операции просмотра данных о заказах, требуется выбрать вкладку «Сервисы» вверху окна браузера, рисунок 4.1.

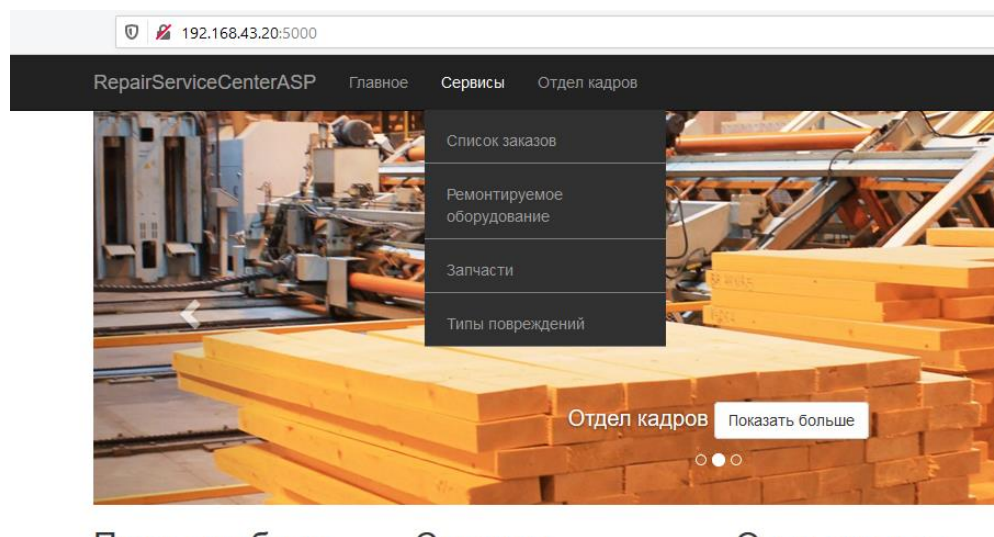


Рисунок 4.1 – Выбор сервиса

Затем выбрать вкладку «Список заказов» и загрузится новое окно со списком заказов, рисунок 4.2.

Заказы

[Добавить новый заказ](#)

Дата заказа:

Имя заказчика:

Сотрудник:

Гарантия: ☐

| Дата заказа | Дата возврата | Полное имя заказчика | Гарантия | Срок гарантии | Стоимость заказа | Ремонтируемое оборудование | Тип повреждения | Обслуживаемый магазин | Сотрудник |
|-------------|---------------|----------------------|-------------|---------------|------------------|----------------------------|------------------------|-----------------------|--|
| 12.12.2019 | 12.12.2019 | Башаримов Ю.И. | Имеется | 22 | 12,89 | S-21 | Поврежден дисплей | Zeon 72 | Солодков Е.В. Ред. Детали Удалить |
| 12.12.2019 | 12.12.2019 | Каркозов В.В. | Отсутствует | 21 | 7,25 | AM-0 | Повреждена проводка | Бай-Бак 24 | Липский Д.Ю. Ред. Детали Удалить |
| 12.12.2019 | 12.12.2019 | Степаненко Ю.А. | Отсутствует | 59 | 6,12 | S-28 | Повреждена проводка | Бай-Бак 13 | Козаченко М.А. Ред. Детали Удалить |
| 12.12.2019 | 12.12.2019 | Ястребов А.С. | Имеется | 16 | 5,94 | PES-6 | Визуальные повреждения | RentalCars 1 | Липский Д.Ю. Ред. Детали Удалить |

Рисунок 4.2 – Список заказов по ремонту оборудования

Так же можно осуществлять фильтрацию заказов по дате заказа, имени заказчика, сотруднику выполняющий данный заказ и по наличию гарантии. При нажатии на название столбцов в шапке таблицы, будет произведена сортировка по возрастанию, при повторном нажатии сортироваться будет по убыванию.

Чтобы добавить новый заказ, нужно нажать на ссылку, выделенную голубым цветом, под названием вкладки (в данном случае – «Заказы»), затем загрузится новая страница, рисунок 4.3 с формами для оформления заказа (все поля формы имеют проверку на корректность введенных данных). Чтобы окончательно оформить заказ, нужно нажать на кнопку «Добавить».

RepairServiceCenterASP | Главное | Сервисы | Отдел кадров

Создание Заказа

Дата заказа:

Дата возврата:

Полное имя заказчика:

Ремонтируемая модель:

Тип повреждения:

Магазин:

Гарантия: ☒

Срок гарантии:

Сотрудник:

[Вернуться назад](#)

© 2019 - RepairServiceCenterASP

Рисунок 4.3 – Форма для добавления нового заказа

После добавления заказа пользователь будет перенаправлен на страницу с выборкой.

Для удаления заказа, нужно выбрать нужный заказ в таблице и в самой правой части данной таблицы выбрать пункт «Удалить», рисунок 4.4, после чего заказ будет удалён из базы данных.

| № | Тип повреждения | Обслуживаемый магазин | Сотрудник | |
|---|---------------------|-----------------------|---------------|---|
| | Повреждён дисплей | Zeon 72 | Солодков Е.В. | Ред. Детали Удалить |
| | Повреждена проводка | Бай-Бак 24 | Липский Д.Ю. | Ред. Детали Удалить |
| | Повреждена | Бай-Бак 13 | Козаченко | Ред. |

Рисунок 4.4 – Пункты для редактирования, удаления и просмотра заказов

Для редактирования требуется выбрать пункт «Редактировать», затем пользователь будет перенаправлен, на страницу с формами для редактирования данных, рисунок 4.5. Чтобы сохранить изменения, требуется нажать на кнопку сохранить.

RepairServiceCenterASP | Главное | Сервисы | Отдел кадров

Редактирование

Заказ 12 декабря 2019 г.

Дата заказа
12.12.2019

Дата возврата
26.12.2019

Полное имя заказчика
Башаримов Ю.И.

Ремонтируемая модель
S-21

Тип повреждения
Повреждён дисплей

Магазин
Zeon 72

Гарантия
☐

Срок гарантии
22

Сотрудник
Солодков Е.В.

[Сохранить](#)

[Вернуться назад](#)

Рисунок 4.5 – Окно для редактирования данных заказе

Так же можно посмотреть подробности заказа, нажав на кнопку «Детали», рисунок 4.4. Далее загружается окно с подробным описанием заказа, рисунок 4.6.

RepairServiceCenterASP | Главное | Сервисы | Отдел кадров

Подробности

Заказ 12 декабря 2019 г.

| | |
|-----------------------------|--------------------|
| Дата заказа | 12 декабря 2019 г. |
| Дата возврата | 31 декабря 2019 г. |
| Полное имя заказчика | Башаримов Ю.И. |
| Гарантия | Имеется |
| Срок гарантии | 22 |
| Стоимость заказа | 12,8869652032771 |
| Ремонтируемая мод... | S-21 |
| Тип повреждения | Повреждён дисплей |
| Обслуживаемый ма... | Zeon 72 |
| Сотрудник | Солодков Е.В. |

[Редактировать](#) | [Вернуться назад](#)

© 2019 - RepairServiceCenterASP

Рисунок 4.6 – Подробности заказа

По той же аналогии, описанной выше, можно производить аналогичные манипуляции с данными других сервисов.

5 РУКОВОДСТВО ПРОГРАММИСТА

5.1 Назначения и условия применения программы

Приложение предназначена для предоставления информации из базы данных предприятия по ремонту оборудования, чтобы автоматизировать учёт заказов по их ремонту и прочие манипуляции с данными из базы.

Основные функции приложения:

- производить различные манипуляции с данными из базы данных;
- предоставления данных в удобном виде пользователям для их просмотра;
- управление данными отдела кадров;
- редактирования, добавление и изменения данных из базы с помощью веб-интерфейса.

Для запуска приложения на сервере должна быть установлена платформа *.NET Core*. Для соединения с базой данных, требуется предварительная конфигурация параметров для соединения с ней.

5.2 Характеристики программы

Разработанное приложение написано на языке программирования *C#* в среде разработки *Visual Studio 2019*.

Для хранения данных используется база данных *MS SQL Server*. Работа с ней осуществляется с помощью библиотеки *Entity Framework*, работающая на основании стандартных драйверов для подключения *ADO*.

Серверная часть представляет собой *ASP.NET* приложение, к которому происходят запросы по протоколу *HTTP*, которые он обрабатывает и возвращает клиенту требуемую информацию. При работе используются следующие виды *HTTP*-глаголов: *GET*, *POST*.

5.3 Сопровождение программного комплекса

Для дополнения программного обеспечения новым функционалом можно использовать любую среду разработки на языке программирования *C#*. Приложение реализовано с помощью паттерна *MVC (Model-View-Controller)*, который позволяет в свою очередь разделить модель данных, бизнес-логику приложения и представления, на три части, что позволит разрабатывать новый функционал и поддерживать приложения в команде из нескольких разработчиков. Так же использование данного паттерна сделала приложение легко масштабируемым и поддерживаемым.

При необходимости можно заменить источник данных с *MS SQL Server* на другую базу данных, благодаря интерфейсу источник данных.

5.4 Входные и выходные данные

Входными данными для веб-приложения является:

- веб-сервер, на котором разворачивается приложение;
- сгенерированная база данных с помощью возможностей *Entity Framework*;
- тестовый набор для отладки приложения генерируемый компонентом *Middleware*, листинг приведён в приложении А.

Выходными данным для приложения является получение и предоставление данных с базы пользователю, их сортировка и выборка по критериям.

5.5 Сообщения в ходе работы приложения

При работе программа может оповещать пользователя о следующих неполадках:

- некорректно введенные данных при добавлении и редактировании записей;
- некорректный *URL*-адрес, страница не найдена;
- ошибка при добавлении записей, запись с введенными значениями уже существуют в базе.

Данные сообщения передаются в специальном виде ошибки с описанием проблемы.

ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта было реализовано веб-приложение, которое производит автоматизацию оформления заказов по ремонту оборудования. Приложение является простым и удобным благодаря адаптивному и понятному интерфейсу. Критериями удобства является в первую очередь наличие навигационного меню, что позволяет пользователю всю необходимую информацию, а также улучшает навигацию между страницами, не производя при этом никаких лишних действий.

Функционал приложения является вполне достаточным для выполнения основных задач, и структура спроектирована таким образом, что его дальнейшее расширение не приведёт ни к каким трудностям: изменению структуры или переписыванию логики. Все вышеперечисленные преимущества, поможет мелким сервисам по ремонту оборудования автоматизировать свой производственный процесс и учёт заказов.

В результате разработки курсового проекта, была изучена технология *ASP.NET Core MVC*. Технология позволяет использовать шаблоны, которые выполняют конкретные задачи. Так же благодаря платформе *.NET Core* приложение не зависит от операционной системы, или веб-сервера и является кроссплатформенной.

MVC описывает простой способ создания основной структуры приложения, что позволяет легко ориентироваться в коде, т.к. он разбит на блоки, а также серьёзно упрощает отладочный процесс. <https://docs.microsoft.com/ru-ru/ASPNET/core/host-and-deploy/?view=aspnetcore-2.1>

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Практическое руководство к курсовому проектированию по курсу «Информатика» для студентов технических специальностей дневной и заочной форм обучения – Гомель: ГГТУ им. П.О. Сухого, 2019. – 32 с.
2. Шилдт Герберт. С# 4.0: полное руководство: учебное пособие – ООО «И.Д. Вильямс», 2011. – 1056 с.
3. Чамберс Д., Пэккетт Д., Тиммс С., ASP.NET Core. Разработка приложений. – Спб.: Питер, 2018. – 464 с.
4. Размещение и развёртывания ASP.NET Core приложения, – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/ASPNET/core/host-and-deploy/?view=aspnetcore-2.1>. – Дата доступа: 12.12.2019.
5. ASP.NET Core. Dependency Injection, Электрон. данные. – Режим доступа: <https://metanit.com/sharp/aspnet5/6.1.php>. Дата доступа: 13.12.2019.
6. ASP.NET Core. Введение в MVC, Электрон. данные. – Режим доступа: <https://metanit.com/sharp/aspnet5/3.1.php>. Дата доступа: 13.12.2019.

ПРИЛОЖЕНИЕ А

(обязательное)

Код программы

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=.\SQLEXPRESS;Database=RepairServiceCenterGenerate;Trusted_Connection=True;MultipleActiveResultSets=true",
    "SqlLiteConnection": "DataSource=.\RepairServiceCenterLite.db"
  },

  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Employee.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Сотрудник")]
    public class Employee
    {
        public enum SortState
        {
            FullNameAsc, FullNameDesc,
            ExperienceAsc, ExperienceDesc,
            PostAsc, PostDesc
        }

        [Display(Name = "Код")]
        public int EmployeeId { get; set; }
        [Display(Name = "ФИО")]
        public string FullName { get; set; }
        [Display(Name = "Опыт работы")]
        public int? Experience { get; set; }
        [Display(Name = "Id должности")]
        public int? PostId { get; set; }

        [Display(Name = "Должность")]
        public Post Post { get; set; }
        public ICollection<Order> Orders { get; set; }

        public Employee()
        {
            Orders = new List<Order>();
        }
    }
}
```

Order.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    public class Order
    {
        public enum SortState
        {
            DateOrderAsc, DateOrderDesc,
            ReturnDateAsc, ReturnDateDesc,
            FullNameCustAsc, FullNameCustDesc,
            RepModelAsc, RepModelDesc,
            TypeOfFaultAsc, TypeOfFaultDesc,
            ServiceStoreAsc, ServiceStoreDesc,
            GuaranteeMarkAsc, GuaranteeMarkDesc,
            GuaranteePeriodAsc, GuaranteePeriodDesc,
            PriceAsc, PriceDesc,
            EmployeeAsc, EmployeeDesc
        }

        [Display(Name = "Код")]
        public int OrderId { get; set; }
        [Display(Name = "Дата заказа")]
        public DateTime DateOrder { get; set; }
        [Display(Name = "Дата возврата")]
        public DateTime ReturnDate { get; set; }
        [Display(Name = "Полное имя заказчика")]
        public string FullNameCustomer { get; set; }
        [Display(Name = "Ремонтируемая модель")]
        public int? RepairedModelId { get; set; }
        [Display(Name = "Тип повреждения")]
        public int? TypeOfFaultId { get; set; }
        [Display(Name = "Магазин")]
        public int? ServicedStoreId { get; set; }
        [Display(Name = "Гарантия")]
        public bool? GuaranteeMark { get; set; }
        [Display(Name = "Срок гарантии")]
        public int GuaranteePeriod { get; set; }
        [Display(Name = "Стоимость заказа")]
        public double Price { get; set; }
        [Display(Name = "Сотрудник")]
        public int? EmployeeId { get; set; }

        public RepairedModel RepairedModel { get; set; }
        public TypeOfFault TypeOfFault { get; set; }
        public ServicedStore ServicedStore { get; set; }
        public Employee Employee { get; set; }
    }
}
```

Post.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Должность")]
}
```

```

public class Post
{
    [Display(Name = "Код")]
    public int PostId { get; set; }
    [Display(Name = "Название")]
    public string Name { get; set; }
    [Display(Name = "Зарплата")]
    public double? Money { get; set; }

    public ICollection<Employee> Employees { get; set; }

    public Post()
    {
        Employees = new List<Employee>();
    }
}

```

RepairedModel.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Ремонтируемая модель")]
    public class RepairedModel
    {
        [Display(Name = "Ремонтируемая модель")]
        public int RepairedModelId { get; set; }
        [Display(Name = "Ремонтируемая модель")]
        public string Name { get; set; }
        [Display(Name = "Тип")]
        public string Type { get; set; }
        [Display(Name = "Производитель")]
        public string Manufacturer { get; set; }
        [Display(Name = "Тех. спецификация")]
        public string TechSpecification { get; set; }
        [Display(Name = "Особенности")]
        public string Features { get; set; }

        public ICollection<SparePart> SpareParts { get; set; }
        public ICollection<Order> Orders { get; set; }

        public RepairedModel()
        {
            SpareParts = new List<SparePart>();
            Orders = new List<Order>();
        }
    }
}

```

ServicedStore.cs

```

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Обслуживаемый магазин")]
    public class ServicedStore

```

```

{
    [Display(Name = "Код")]
    public int ServicedStoreId { get; set; }
    [Display(Name = "Название")]
    public string Name { get; set; }
    [Display(Name = "Адрес")]
    public string Address { get; set; }
    [Display(Name = "Номер телефона")]
    public string PhoneNumber { get; set; }

    public ICollection<Order> Orders { get; set; }

    public ServicedStore()
    {
        Orders = new List<Order>();
    }
}

```

SparePart.cs

```

using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Запчасть")]
    public class SparePart
    {
        [Display(Name = "Код")]
        public int SparePartId { get; set; }
        [Display(Name = "Название")]
        public string Name { get; set; }
        [Display(Name = "Функции")]
        public string Functions { get; set; }
        [Display(Name = "Стоимость")]
        public double? Price { get; set; }
        [Display(Name = "Ремонтируемая модель")]
        public int? RepairedModelId { get; set; }

        [Display(Name = "Ремонтируемая модель")]
        public RepairedModel RepairedModel { get; set; }

        [Display(Name = "Вид неисправности")]
        public int TypeOfFaultId { get; set; }
        [Display(Name = "Вид неисправности")]
        public TypeOfFault TypeOfFault { get; set; }
    }
}

```

TypeOfFault.cs

```

using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace RepairServiceCenterASP.Models
{
    [Display(Name = "Тип повреждения")]
    public class TypeOfFault
    {
        public enum SortState
        {

```

```

        RepairedModelAsc, RepairedModelDesc,
        NameAsc, NameDesc,
        MethodRepairAsc, MethodRepairDesc,
        WorkPriceAsc, WorkPriceDesc
    }

    [Display(Name = "Код")]
    public int TypeOfFaultId { get; set; }
    [Display(Name = "Ремонтируемая модель")]
    public int RepairedModelId { get; set; }
    [Display(Name = "Тип повреждения")]
    public string Name { get; set; }
    [Display(Name = "Метод починки")]
    public string MethodRepair { get; set; }
    [Display(Name = "Цена работы")]
    public double? WorkPrice { get; set; }

    public RepairedModel RepairedModel { get; set; }
    public IEnumerable<SparePart> SpareParts { get; set; }

    public ICollection<Order> Orders { get; set; }

    public TypeOfFault()
    {
        Orders = new List<Order>();
    }
}
}}
ErrorViewModel.cs

using System;

namespace RepairServiceCenterASP.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}

ICachingModel.cs

using System.Collections.Generic;

namespace RepairServiceCenterASP.Services
{
    public interface ICachingModel<T>
    {
        ICollection<T> ReadAllCache(string cacheKey);
        void RefreshCache(string cacheKey);
        bool CreateCache(T entity);
        T ReadCache(string cacheKey, int id);
        bool EditCache(T entity);
    }
}

RepeiredModelService.cs

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using Microsoft.Extensions.Caching.Memory;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Services
{
    public class RepeiredModelService : ICachingModel<RepairedModel>
    {
        private RepairServiceCenterContext db;
        private IMemoryCache cache;
        private int rowsNumber = 50;
        private const int SECONDS = 272;

        public RepeiredModelService(RepairServiceCenterContext db, IMemoryCache memoryCache)
        {
            this.db = db;
            cache = memoryCache;
        }

        public bool EditCache(RepairedModel entity)
        {
            try
            {
                var model = db.RepairedModels.Where(m => m.RepairedModelId == entity.RepairedModelId)
                    .FirstOrDefault();
                if (model != null)
                {
                    model = entity;
                    cache.Set(entity.RepairedModelId, entity, new MemoryCacheEntryOptions
                    {
                        AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(SECONDS)
                    });
                    db.SaveChanges();
                }
                return true;
            }
            catch
            {
                return false;
            }
        }

        public bool CreateCache(RepairedModel entity)
        {
            try
            {
                db.RepairedModels.Add(entity);
                int n = db.SaveChanges();
                if (n > 0)
                {
                    cache.Set(entity.RepairedModelId, entity, new MemoryCacheEntryOptions
                    {
                        AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(SECONDS)
                    });
                }
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}

```



```

    }

    public RepairedModel ReadCache(string cacheKey, int id)
    {
        ICollection<RepairedModel> repairedModels = null;
        if (!cache.TryGetValue(cacheKey, out repairedModels))
        {
            var repairedModel = db.RepairedModels.Where(r => r.RepairedModelId == id).FirstOrDefault();
            cache.Set(repairedModel.RepairedModelId, repairedModel, new MemoryCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(SECONDS)
            });
        }
        return repairedModels.Where(r => r.RepairedModelId == id).FirstOrDefault();
    }

    public ICollection<RepairedModel> ReadAllCache(string cacheKey)
    {
        ICollection<RepairedModel> repairedModels = null;
        if (!cache.TryGetValue(cacheKey, out repairedModels))
        {
            repairedModels = db.RepairedModels.Take(rowsNumber).ToList();
            if (repairedModels != null)
            {
                cache.Set(cacheKey, repairedModels,
                    new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(SECONDS)));
            }
        }
        return repairedModels;
    }

    public void RefreshCache(string cacheKey)
    {
        var repairedModels = db.RepairedModels.Take(rowsNumber).ToList();
        cache.Set(cacheKey, repairedModels,
            new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(SECONDS)));
    }
}

```

RepairServiceCenterContext.cs

```

using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Data
{
    public class RepairServiceCenterContext : DbContext
    {
        public RepairServiceCenterContext(DbContextOptions options) : base(options)
        {
        }

        public DbSet<Post> Posts { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<RepairedModel> RepairedModels { get; set; }
        public DbSet<SparePart> SpareParts { get; set; }
        public DbSet<TypeOfFault> TypeOfFaults { get; set; }
        public DbSet<ServicedStore> ServicedStores { get; set; }
        public DbSet<Order> Orders { get; set; }
    }
}

```

DbInitializer.cs

```
using RepairServiceCenterASP.Models;
using System;
using System.Linq;

namespace RepairServiceCenterASP.Data
{
    public static class DbInitializer
    {
        private static Random randObj = new Random(1);

        public static void Initialize(RepairServiceCenterContext db)
        {
            db.Database.EnsureCreated();

            int repairedModelsNumber = 35;
            int sparePartsNumber = 200;
            int typeOfFaultsNumber = 35;
            int serviceStoreNumber = 40;
            int countOrders = 300;

            RepairedModelsGenerate(repairedModelsNumber, ref db);
            TypeOfFaultGeneration(typeOfFaultsNumber, repairedModelsNumber, ref db);
            SparePartsGeneration(sparePartsNumber, repairedModelsNumber, typeOfFaultsNumber, ref db);
            ServiceStoreGeneration(serviceStoreNumber, ref db);
            PostGenerate(ref db);
            EmployeeGenerate(ref db);
            OrdersGenerate(countOrders, ref db);
        }

        private static void OrdersGenerate(int count, ref RepairServiceCenterContext db)
        {
            if (db.Orders.Any())
            {
                return;
            }

            int countTypeOfFault = db.TypeOfFaults.Count();
            int countServicedStore = db.ServicedStores.Count();
            int countEmployee = db.Employees.Count();
            int countRepairedModel = db.RepairedModels.Count();

            DateTime dateOrder;
            DateTime returnDate;
            string fullNameCust;
            int repairedModelId;
            int typeOfFaultId;
            int serviceStoreId;
            bool guaranteeMark;
            int guaranteePeriod;
```

```

int employeeId;
double price;

string[] namesVoc = { "Жмайлик А.В.", "Сетко А.И.", "Семёнов С.А.", "Давыдчик А.Е.", "Пискун Е.А.",
    "Дракула В.А.", "Ястребов А.В.", "Степаненко Ю.А.", "Башаримов Ю.И.", "Каркозов В.В."
};

for (int i = 0; i < count; i++)
{
    dateOrder = DateTime.Now.AddTicks(-randObj.Next());
    returnDate = dateOrder.AddDays(randObj.Next(1, 40));
    fullNameCust = namesVoc[randObj.Next(namesVoc.GetLength(0))];
    repairedModelId = randObj.Next(1, countRepairedModel - 1);
    typeOfFaultId = randObj.Next(1, countTypeOfFault - 1);
    serviceStoreId = randObj.Next(1, countServicedStore - 1);
    employeeId = randObj.Next(1, countEmployee - 1);
    guaranteeMark = randObj.Next(1000) > 500 ? true : false;
    guaranteePeriod = randObj.Next(120);

    var typeOfFaults = db.TypeOfFaults.Where(t => t.TypeOfFaultId == typeOfFaultId)
        .FirstOrDefault();
    price = (double)typeOfFaults.WorkPrice * 2 * randObj.NextDouble();

    db.Orders.Add(new Order()
    {
        DateOrder = dateOrder,
        ReturnDate = returnDate,
        FullNameCustomer = fullNameCust,
        RepairedModelId = repairedModelId,
        TypeOfFaultId = typeOfFaultId,
        ServicedStoreId = serviceStoreId,
        GuaranteeMark = guaranteeMark,
        GuaranteePeriod = guaranteePeriod,
        EmployeeId = employeeId,
        Price = price
    });
}
db.SaveChanges();
}

private static void PostGenerate(ref RepairServiceCenterContext db)
{
    if (db.Posts.Any())
    {
        return;
    }

    db.Posts.Add(new Post()
    {
        Name = "Директор",
        Money = 5000
    });
}

```

```

db.Posts.Add(new Post()
{
    Name = "Зам. директора",
    Money = 2500
});
db.Posts.Add(new Post()
{
    Name = "Программист",
    Money = 2000
});
db.Posts.Add(new Post()
{
    Name = "Инженер",
    Money = 1500
});
db.Posts.Add(new Post()
{
    Name = "Главный-инженер",
    Money = 2100
});
db.Posts.Add(new Post()
{
    Name = "ИТ-директор",
    Money = 3500
});
db.Posts.Add(new Post()
{
    Name = "ИТ-менеджер",
    Money = 2500
});
db.Posts.Add(new Post() //9
{
    Name = "Сантехник",
    Money = 500
});
db.Posts.Add(new Post() //10
{
    Name = "Уборщик",
    Money = 300
});
db.Posts.Add(new Post() //11
{
    Name = "Мед. сестра",
    Money = 550
});
db.Posts.Add(new Post() //12
{
    Name = "Продавец",
    Money = 400
});
db.SaveChanges();
}

```

```

private static void EmployeeGenerate(ref RepairServiceCenterContext db)
{
    if (db.Employees.Any())
    {
        return;
    }

    db.Employees.Add(new Employee()
    {
        FullName = "Трофимов Е.В.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Директор")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Солодков Е.В.",
        Experience = 8,
        Post = db.Posts.Where(p => p.Name == "Программист")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Ропот И.В.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Инженер")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Липский Д.Ю.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "ИТ-менеджер")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Межейников А.С.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "ИТ-директор")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Михайлов А.С.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Главный-инженер")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {

```

```

        FullName = "Козаченко М.А.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Уборщик")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Главич Д.Ю.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Мед. сестра")
            .FirstOrDefault(),
    });
    db.Employees.Add(new Employee()
    {
        FullName = "Стольный С.В.",
        Experience = 10,
        Post = db.Posts.Where(p => p.Name == "Инженер")
            .FirstOrDefault(),
    });
    db.SaveChanges();
}

private static void ServiceStoreGeneration(int num, ref RepairServiceCenterContext db)
{
    if (db.ServicedStores.Any())
    {
        return;
    }

    string name;
    string address;
    string phoneNumber;

    string[] namesVoc = { "PriceRent", "RentalCars", "ServiceTransportOnline", "PhonesOne", "BestSpendTime",
        "БелГосСтрах", "SAMSUNG STORE", "Бай-Бак", "Zeon" };
    string[] addressVoc = { "пер.Заслонова, ", "ул.Гастело, ", "ул.Полесская, ", "пр.Речецкий, ", "ул,
        Интерноциональная, ",
        "пр.Октября, ", "ул.Бассейная, ", "бул.Юности, " };

    for (int i = 0; i < num; i++)
    {
        name = namesVoc[randObj.Next(namesVoc.GetLength(0))] + " " + randObj.Next(0, num + 50);
        address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(0, 250);
        phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) +
            "-" + randObj.Next(10, 99);
        db.ServicedStores.Add(new ServicedStore()
        {
            Name = name,
            Address = address,
            PhoneNumber = phoneNumber
        });
        db.SaveChanges();
    }
}

```

```

    }
}

private static void TypeOfFaultGeneration(int num, int rModelsNum, ref RepairServiceCenterContext db)
{
    if (db.TypeOfFaults.Any())
    {
        return;
    }

    int repairedModelId;
    string name;
    string methodRepair;
    double workPrice;

    string[] namesVoc = { "Повреждён дисплей", "Повреждение электроники", "Визуальные повреждения",
        "Повреждена проводка" };
    string[] methodRepairVoc = { "Полная замена деталей", "Частичная замена", "Незначительный ремонт" };

    for (int i = 0; i < num; i++)
    {
        repairedModelId = randObj.Next(1, rModelsNum - 1);
        name = namesVoc[randObj.Next(namesVoc.GetLength(0))];
        methodRepair = methodRepairVoc[randObj.Next(methodRepairVoc.GetLength(0))];

        var repairedModel = db.RepairedModels.Where(r => r.RepairedModelId == repairedModelId)
            .FirstOrDefault();
        workPrice = repairedModelId * 2 * randObj.NextDouble();

        db.TypeOfFaults.Add(new TypeOfFault
        {
            RepairedModelId = repairedModelId,
            Name = name,
            MethodRepair = methodRepair,
            WorkPrice = workPrice
        });
    }
    db.SaveChanges();
}

private static void SparePartsGeneration(int num, int rModelsNum, int typeNum, ref RepairServiceCenterContext
db)
{
    if (db.SpareParts.Any())
    {
        return;
    }

    string name;
    string function;
    double price;

```

```

int repairedModelId;
int typeOfFaultId;

string[] namesVoc = { "Дисплей-", "Процессор-", "Проводка-", "Корпус-", "Комплектующие-" };
string[] functionVoc = { "Create", "Read", "Update", "Delete" };

for (int i = 0; i < num; i++)
{
    name = namesVoc[randObj.Next(namesVoc.GetLength(0))] + i.ToString();
    price = 30 * randObj.NextDouble();
    function = functionVoc[randObj.Next(functionVoc.GetLength(0))];
    repairedModelId = randObj.Next(1, rModelsNum - 1);
    typeOfFaultId = randObj.Next(1, typeNum - 1);

    db.SpareParts.Add(new SparePart()
    {
        Name = name,
        Functions = function,
        Price = price,
        RepairedModelId = repairedModelId,
        TypeOfFaultId = typeOfFaultId
    });
}
db.SaveChanges();
}

private static void RepairedModelsGenerate(int num, ref RepairServiceCenterContext db)
{
    if (db.RepairedModels.Any())
    {
        return;
    }

    string name;
    string type;
    string manufacture;
    string techSpecification;
    string features;

    string[] namesVoc = { "FE-", "AM-", "S-", "A-", "PES-", "CAT-", "DOG-", "T-" };
    string[] typesVoc = { "Производственное", "Медицинское", "Строительное", "Военное" };
    string[] manufacturesVoc = { "SAMSUNG", "PHILIPS", "HONDA", "Google", "ASUS", "БелТех",
"МозырьСтрой" };
    string[] featuresVoc = { "Полный ремонт", "Полная функциональность", "Обновление", "Приемлемый
вид" };

    for (int i = 0; i < num; i++)
    {
        name = namesVoc[randObj.Next(namesVoc.GetLength(0))] + i.ToString();
        type = typesVoc[randObj.Next(typesVoc.GetLength(0))];
        manufacture = manufacturesVoc[randObj.Next(typesVoc.GetLength(0))];
        techSpecification = "CF18" + randObj.Next(100000, 999999);
    }
}

```



```

        features = featuresVoc[randObj.Next(featuresVoc.GetLength(0))];

        db.RepairedModels.Add(new RepairedModel()
        {
            Name = name,
            Type = type,
            Manufacturer = manufacture,
            TechSpecification = techSpecification,
            Features = features
        });
    }

    db.SaveChanges();
}
}
}

```

TypeOfFaultsController.cs

```

using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;

namespace RepairServiceCenterASP.Controllers
{
    public class TypeOfFaultsController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public TypeOfFaultsController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: TypeOfFaults
        public async Task<IActionResult> Index(int? model, string name, string methodRepair, string client,
            int page = 1, TypeOfFault.SortState sortOrder = TypeOfFault.SortState.NameAsc)
        {
            int pageSize = 20;

            IQueryable<TypeOfFault> source = _context.TypeOfFaults.Include(t => t.RepairedModel);

            if (model != null)
                source = source.Where(t => t.RepairedModel.RepairedModelId == model.Value);

            if (!String.IsNullOrEmpty(name))
                source = source.Where(t => t.Name.Contains(name));

            if (!String.IsNullOrEmpty(methodRepair))
                source = source.Where(t => t.MethodRepair.Contains(methodRepair));

```

```

if (!String.IsNullOrEmpty(client))
{
    source = _context.Orders.Include(o => o.TypeOfFault)
        .Include(o => o.RepairedModel)
        .Where(o => o.FullNameCustomer.Contains(client))
        .Select(o => new TypeOfFault()
        {
            Name = o.TypeOfFault.Name,
            RepairedModelId = o.RepairedModelId.Value,
            RepairedModel = o.RepairedModel,
            MethodRepair = o.TypeOfFault.MethodRepair
        });
}

source = TypesOfFaultsSort(source, sortOrder);

int count = await source.CountAsync();
var items = await source.Skip((page - 1) * pageSize).Take(pageSize).ToListAsync();
var models = await _context.RepairedModels.ToListAsync();

var viewModel = new TypeOfFaultsViewModel()
{
    TypeOfFaults = items,
    TypesOfFaultsFilter = new TypesOfFaultsFilter(models, model, name, methodRepair, client),
    TypesOfFaultsSort = new TypesOfFaultsSort(sortOrder),
    PageViewModel = new PageViewModel(count, page, pageSize)
};

return View(viewModel);
}

// GET: TypeOfFaults/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var typeOfFault = await _context.TypeOfFaults
        .Include(t => t.RepairedModel)
        .FirstOrDefaultAsync(m => m.TypeOfFaultId == id);
    if (typeOfFault == null)
    {
        return NotFound();
    }

    return View(typeOfFault);
}

// GET: TypeOfFaults/Create
public IActionResult Create()
{
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name");
    return View();
}

// POST: TypeOfFaults/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]

```

```

public async Task<IActionResult> Create([Bind("TypeOfFaultId,RepairedModelId,Name,MethodRepair,Work-
Price")] TypeOfFault typeOfFault)
{
    if (ModelState.IsValid)
    {
        _context.Add(typeOfFault);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
typeOfFault.RepairedModelId);
    return View(typeOfFault);
}

// GET: TypeOfFaults/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var typeOfFault = await _context.TypeOfFaults.FindAsync(id);
    if (typeOfFault == null)
    {
        return NotFound();
    }
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
typeOfFault.RepairedModelId);
    return View(typeOfFault);
}

// POST: TypeOfFaults/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("TypeOfFaultId,RepairedModelId,Name,MethodRe-
pair,WorkPrice")] TypeOfFault typeOfFault)
{
    if (id != typeOfFault.TypeOfFaultId)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(typeOfFault);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!TypeOfFaultExists(typeOfFault.TypeOfFaultId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
    }
}

```

```

        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
typeOfFault.RepairedModelId);
    return View(typeOfFault);
}

// GET: TypeOfFaults/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var typeOfFault = await _context.TypeOfFaults
        .Include(t => t.RepairedModel)
        .FirstOrDefaultAsync(m => m.TypeOfFaultId == id);
    if (typeOfFault == null)
    {
        return NotFound();
    }

    return View(typeOfFault);
}

// POST: TypeOfFaults/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var typeOfFault = await _context.TypeOfFaults.FindAsync(id);
    _context.TypeOfFaults.Remove(typeOfFault);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool TypeOfFaultExists(int id)
{
    return _context.TypeOfFaults.Any(e => e.TypeOfFaultId == id);
}

private IQueryable<TypeOfFault> TypesOfFaultsSort(IQueryable<TypeOfFault> typesOfFaults,
TypeOfFault.SortState sortOrder)
{
    switch (sortOrder)
    {
        case TypeOfFault.SortState.NameAsc:
            return typesOfFaults.OrderBy(t => t.Name);

        case TypeOfFault.SortState.NameDesc:
            return typesOfFaults.OrderByDescending(t => t.Name);

        case TypeOfFault.SortState.RepairedModelAsc:
            return typesOfFaults.OrderBy(t => t.RepairedModel.Name);

        case TypeOfFault.SortState.RepairedModelDesc:
            return typesOfFaults.OrderByDescending(t => t.RepairedModel.Name);

        case TypeOfFault.SortState.MethodRepairAsc:

```

```

        return typesOfFaults.OrderBy(t => t.MethodRepair);

    case TypeOfFault.SortState.MethodRepairDesc:
        return typesOfFaults.OrderByDescending(t => t.MethodRepair);

    case TypeOfFault.SortState.WorkPriceAsc:
        return typesOfFaults.OrderBy(t => t.WorkPrice);

    case TypeOfFault.SortState.WorkPriceDesc:
        return typesOfFaults.OrderByDescending(t => t.WorkPrice);

    default:
        return typesOfFaults.OrderBy(t => t.Name);
    }
}
}
}

```

SparePartsController.cs

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Controllers
{
    public class SparePartsController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public SparePartsController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: SpareParts
        public async Task<IActionResult> Index()
        {
            var repairServiceCenterContext = _context.SpareParts.Include(s => s.RepairedModel).Include(s =>
s.TypeOfFault);
            return View(await repairServiceCenterContext.ToListAsync());
        }

        // GET: SpareParts/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var sparePart = await _context.SpareParts
                .Include(s => s.RepairedModel)
                .Include(s => s.TypeOfFault)
                .FirstOrDefaultAsync(m => m.SparePartId == id);
            if (sparePart == null)
            {
                return NotFound();
            }
        }
    }
}

```

```

    }

    return View(sparePart);
}

// GET: SpareParts/Create
public IActionResult Create()
{
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name");
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name");
    return View();
}

// POST: SpareParts/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("SparePartId,Name,Functions,Price,RepairedModelId,TypeOfFaultId")] SparePart sparePart)
{
    if (ModelState.IsValid)
    {
        _context.Add(sparePart);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
    sparePart.RepairedModelId);
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name",
    sparePart.TypeOfFaultId);
    return View(sparePart);
}

// GET: SpareParts/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var sparePart = await _context.SpareParts.FindAsync(id);
    if (sparePart == null)
    {
        return NotFound();
    }
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
    sparePart.RepairedModelId);
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name",
    sparePart.TypeOfFaultId);
    return View(sparePart);
}

// POST: SpareParts/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("SparePartId,Name,Functions,Price,RepairedModelId,TypeOfFaultId")] SparePart sparePart)
{

```

```

        if (id != sparePart.SparePartId)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(sparePart);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!SparePartExists(sparePart.SparePartId))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name",
        sparePart.RepairedModelId);
        ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name",
        sparePart.TypeOfFaultId);
        return View(sparePart);
    }

    // GET: SpareParts/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var sparePart = await _context.SpareParts
            .Include(s => s.RepairedModel)
            .Include(s => s.TypeOfFault)
            .FirstOrDefaultAsync(m => m.SparePartId == id);
        if (sparePart == null)
        {
            return NotFound();
        }

        return View(sparePart);
    }

    // POST: SpareParts/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var sparePart = await _context.SpareParts.FindAsync(id);
        _context.SpareParts.Remove(sparePart);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
}

```

```

        private bool SparePartExists(int id)
        {
            return _context.SpareParts.Any(e => e.SparePartId == id);
        }
    }
}

```

ServicedStoresController.cs

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Controllers
{
    public class ServicedStoresController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public ServicedStoresController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: ServicedStores
        public async Task<IActionResult> Index()
        {
            return View(await _context.ServicedStores.ToListAsync());
        }

        // GET: ServicedStores/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var servicedStore = await _context.ServicedStores
                .FirstOrDefaultAsync(m => m.ServicedStoreId == id);
            if (servicedStore == null)
            {
                return NotFound();
            }

            return View(servicedStore);
        }

        // GET: ServicedStores/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: ServicedStores/Create
        // To protect from overposting attacks, please enable the specific properties you want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
    }
}

```



```

[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ServicedStoreId,Name,Address,PhoneNumber")] ServicedStore
servicedStore)
{
    if (ModelState.IsValid)
    {
        _context.Add(servicedStore);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(servicedStore);
}

// GET: ServicedStores/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var servicedStore = await _context.ServicedStores.FindAsync(id);
    if (servicedStore == null)
    {
        return NotFound();
    }
    return View(servicedStore);
}

// POST: ServicedStores/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ServicedStoreId,Name,Address,PhoneNumber")] Serviced-
Store servicedStore)
{
    if (id != servicedStore.ServicedStoreId)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(servicedStore);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ServicedStoreExists(servicedStore.ServicedStoreId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
}

```

```

        return View(servicedStore);
    }

    // GET: ServicedStores/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var servicedStore = await _context.ServicedStores
            .FirstOrDefaultAsync(m => m.ServicedStoreId == id);
        if (servicedStore == null)
        {
            return NotFound();
        }

        return View(servicedStore);
    }

    // POST: ServicedStores/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var servicedStore = await _context.ServicedStores.FindAsync(id);
        _context.ServicedStores.Remove(servicedStore);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool ServicedStoreExists(int id)
    {
        return _context.ServicedStores.Any(e => e.ServicedStoreId == id);
    }
}

```

RepairedModelsController.cs

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.Services;

namespace RepairServiceCenterASP.Controllers
{
    public class RepairedModelsController : Controller
    {
        private readonly RepairServiceCenterContext _context;
        private readonly ICachingModel<RepairedModel> _cachingModel;
        private const string KEY_CACHE = "RepairedModel50";
        private const int PAGE_SIZE = 10;

        public RepairedModelsController(RepairServiceCenterContext context,
            ICachingModel<RepairedModel> cachingModel)
        {
            _context = context;

```

```

        _cachingModel = cachingModel;
    }

    // GET: RepairedModels
    [ResponseCache(Location = ResponseCacheLocation.Any, Duration = 300)]
    public IActionResult Index(int page = 1)
    {
        // Разбиение на страницы
        var count = _cachingModel.ReadAllCache(KEY_CACHE).Count();
        var rModels = _cachingModel.ReadAllCache(KEY_CACHE);

        return View(rModels);
    }

    // GET: RepairedModels/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var repairedModel = await _context.RepairedModels
            .FirstOrDefaultAsync(m => m.RepairedModelId == id);
        if (repairedModel == null)
        {
            return NotFound();
        }

        return View(repairedModel);
    }

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

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("RepairedModelId,Name,Type,Manufacturer," +
        "TechSpecification,Features")] RepairedModel repairedModel)
    {
        if (ModelState.IsValid)
        {
            _context.Add(repairedModel);
            await Task.Run(() =>
            {
                _context.SaveChangesAsync();
                _cachingModel.RefreshCache(KEY_CACHE);
            });
            return RedirectToAction(nameof(Index));
        }
        return View(repairedModel);
    }

    // GET: RepairedModels/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }
    }

```

```

var repairedModel = await _context.RepairedModels.FindAsync(id);
if (repairedModel == null)
{
    return NotFound();
}
return View(repairedModel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("RepairedModelId,Name,Type,Manufacturer," +
    "TechSpecification,Features")] RepairedModel repairedModel)
{
    if (id != repairedModel.RepairedModelId)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(repairedModel);
            await Task.Run(() =>
            {
                _context.SaveChangesAsync();
                _cachingModel.RefreshCache(KEY_CACHE);
            });
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!RepairedModelExists(repairedModel.RepairedModelId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(repairedModel);
}

// GET: RepairedModels/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var repairedModel = await _context.RepairedModels
        .FirstOrDefaultAsync(m => m.RepairedModelId == id);
    if (repairedModel == null)
    {
        return NotFound();
    }

    return View(repairedModel);
}

```

```

    }

    // POST: RepairedModels/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var repairedModel = await _context.RepairedModels.FindAsync(id);
        _context.RepairedModels.Remove(repairedModel);
        await Task.Run(() =>
        {
            _context.SaveChangesAsync();
            _cachingModel.RefreshCache(KEY_CACHE);
        });
        return RedirectToAction(nameof(Index));
    }

    private bool RepairedModelExists(int id)
    {
        return _context.RepairedModels.Any(e => e.RepairedModelId == id);
    }
}
}

```

PostsController.cs

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Controllers
{
    public class PostsController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public PostsController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: Posts
        public async Task<IActionResult> Index()
        {
            return View(await _context.Posts.ToListAsync());
        }

        // GET: Posts/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var post = await _context.Posts
                .FirstOrDefaultAsync(m => m.PostId == id);
            if (post == null)
            {

```

```

        return NotFound();
    }

    return View(post);
}

// GET: Posts/Create
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("PostId,Name,Money")] Post post)
{
    if (ModelState.IsValid)
    {
        _context.Add(post);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(post);
}

// GET: Posts/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var post = await _context.Posts.FindAsync(id);
    if (post == null)
    {
        return NotFound();
    }
    return View(post);
}

// POST: Posts/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("PostId,Name,Money")] Post post)
{
    if (id != post.PostId)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(post);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {

```

```

        if (!PostExists(post.PostId))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View(post);
}

// GET: Posts/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var post = await _context.Posts
        .FirstOrDefaultAsync(m => m.PostId == id);
    if (post == null)
    {
        return NotFound();
    }

    return View(post);
}

// POST: Posts/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var post = await _context.Posts.FindAsync(id);
    _context.Posts.Remove(post);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool PostExists(int id)
{
    return _context.Posts.Any(e => e.PostId == id);
}
}
}

```

OrdersController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;

```

```

using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;

namespace RepairServiceCenterASP.Controllers
{
    public class OrdersController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public OrdersController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: Orders
        public async Task<IActionResult> Index(DateTime? dateOrder, string fullNameCust, int? employee,
            bool? guarantee, int page = 1, Order.SortState sortOrder = Order.SortState.DateOrderDesc)
        {
            int pageSize = 20;

            IQueryable<Order> source = _context.Orders.Include(o => o.Employee)
                .Include(o => o.RepairedModel)
                .Include(o => o.ServicedStore)
                .Include(o => o.TypeOfFault);

            if (dateOrder != null)
                source = source.Where(o => o.DateOrder.Year == dateOrder.Value.Year
                    && o.DateOrder.Month == dateOrder.Value.Month
                    && o.DateOrder.Day == dateOrder.Value.Day);

            if (!String.IsNullOrEmpty(fullNameCust))
                source = source.Where(o => o.FullNameCustomer.Contains(fullNameCust));

            if (employee != null && employee != 0)
                source = source.Where(o => o.EmployeeId == employee);

            if (guarantee != null)
                source = source.Where(o => o.GuaranteeMark == guarantee.Value);

            source = OrdersSort(source, sortOrder);

            int count = await source.CountAsync();
            var items = await source.Skip((page - 1) * pageSize).Take(pageSize).ToListAsync();
            var employees = await _context.Employees.ToListAsync();

            OrdersViewModel ordersViewModels = new OrdersViewModel()
            {
                OrdersSort = new OrdersSort(sortOrder),
                OrdersFilter = new OrdersFilter(dateOrder, fullNameCust, employees, employee, guarantee),
                Orders = items,
            }
        }
    }
}

```



```

        PageViewModel = new PageViewModel(count, page, pageSize)
    };
    return View(ordersViewModels);
}

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

    var order = await _context.Orders
        .Include(o => o.Employee)
        .Include(o => o.RepairedModel)
        .Include(o => o.ServicedStore)
        .Include(o => o.TypeOfFault)
        .FirstOrDefaultAsync(m => m.OrderId == id);
    if (order == null)
    {
        return NotFound();
    }

    return View(order);
}

// GET: Orders/Create
public IActionResult Create()
{
    ViewData["EmployeeId"] = new SelectList(_context.Employees, "EmployeeId", "FullName");
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name");
    ViewData["ServicedStoreId"] = new SelectList(_context.ServicedStores, "ServicedStoreId", "Name");
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name");
    return View();
}

// POST: Orders/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("OrderId,DateOrder,ReturnDate,FullNameCustomer,RepairedModelId," +
    "TypeOfFaultId,ServicedStoreId,GuaranteeMark,GuaranteePeriod,EmployeeId")] Order order)
{
    order.Price = (double)_context.TypeOfFaults.Where(t => t.TypeOfFaultId == order.TypeOfFaultId)
        .Select(t => t.WorkPrice)
        .FirstOrDefault();
    if (ModelState.IsValid)
    {
        _context.Add(order);
    }
}

```

```

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["EmployeeId"] = new SelectList(_context.Employees, "EmployeeId", "FullName", order.EmployeeId);
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name", order.RepairedModelId);
    ViewData["ServicedStoreId"] = new SelectList(_context.ServicedStores, "ServicedStoreId", "Name", order.ServicedStoreId);
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name", order.TypeOfFaultId);
    return View(order);
}

// GET: Orders/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var order = await _context.Orders.FindAsync(id);
    if (order == null)
    {
        return NotFound();
    }
    ViewData["EmployeeId"] = new SelectList(_context.Employees, "EmployeeId", "FullName", order.EmployeeId);
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name", order.RepairedModelId);
    ViewData["ServicedStoreId"] = new SelectList(_context.ServicedStores, "ServicedStoreId", "Name", order.ServicedStoreId);
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name", order.TypeOfFaultId);
    return View(order);
}

// POST: Orders/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("OrderId,DateOrder,ReturnDate,FullNameCustomer," +
    "RepairedModelId,TypeOfFaultId,ServicedStoreId,GuaranteeMark,GuaranteePeriod,EmployeeId")] Order order)
{
    if (id != order.OrderId)
    {
        return NotFound();
    }

    if (ModelState.IsValid)

```

```

    {
        try
        {
            order.Price = (double)_context.TypeOfFaults.Where(t => t.TypeOfFaultId == order.TypeOfFaultId)
                .Select(t => t.WorkPrice)
                .FirstOrDefault();
            _context.Update(order);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!OrderExists(order.OrderId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["EmployeeId"] = new SelectList(_context.Employees, "EmployeeId", "FullName", order.EmployeeId);
    ViewData["RepairedModelId"] = new SelectList(_context.RepairedModels, "RepairedModelId", "Name", order.RepairedModelId);
    ViewData["ServicedStoreId"] = new SelectList(_context.ServicedStores, "ServicedStoreId", "Name", order.ServicedStoreId);
    ViewData["TypeOfFaultId"] = new SelectList(_context.TypeOfFaults, "TypeOfFaultId", "Name", order.TypeOfFaultId);
    return View(order);
}

// GET: Orders/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var order = await _context.Orders
        .Include(o => o.Employee)
        .Include(o => o.RepairedModel)
        .Include(o => o.ServicedStore)
        .Include(o => o.TypeOfFault)
        .FirstOrDefaultAsync(m => m.OrderId == id);
    if (order == null)
    {
        return NotFound();
    }

    return View(order);
}

```

```

    }

    // POST: Orders/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var order = await _context.Orders.FindAsync(id);
        _context.Orders.Remove(order);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool OrderExists(int id)
    {
        return _context.Orders.Any(e => e.OrderId == id);
    }

    private IQueryable<Order> OrdersSort(IQueryable<Order> orders, Order.SortState sortOrder)
    {
        switch (sortOrder)
        {
            case Order.SortState.DateOrderAsc:
                return orders.OrderBy(o => o.DateOrder);

            case Order.SortState.DateOrderDesc:
                return orders.OrderByDescending(o => o.DateOrder);

            case Order.SortState.ReturnDateAsc:
                return orders.OrderBy(o => o.ReturnDate);

            case Order.SortState.ReturnDateDesc:
                return orders.OrderByDescending(o => o.ReturnDate);

            case Order.SortState.FullNameCustAsc:
                return orders.OrderBy(o => o.FullNameCustomer);

            case Order.SortState.FullNameCustDesc:
                return orders.OrderByDescending(o => o.FullNameCustomer);

            case Order.SortState.RepModelAsc:
                return orders.OrderBy(o => o.RepairedModel.Name);

            case Order.SortState.RepModelDesc:
                return orders.OrderByDescending(o => o.RepairedModel.Name);

            case Order.SortState.TypeOfFaultAsc:
                return orders.OrderBy(o => o.TypeOfFault.Name);

            case Order.SortState.TypeOfFaultDesc:
                return orders.OrderByDescending(o => o.TypeOfFault.Name);
        }
    }

```

```

        case Order.SortState.GuaranteeMarkAsc:
            return orders.OrderBy(o => o.GuaranteeMark);

        case Order.SortState.GuaranteeMarkDesc:
            return orders.OrderByDescending(o => o.GuaranteeMark);

        case Order.SortState.GuaranteePeriodAsc:
            return orders.OrderBy(o => o.GuaranteePeriod);

        case Order.SortState.GuaranteePeriodDesc:
            return orders.OrderByDescending(o => o.GuaranteePeriod);

        case Order.SortState.PriceAsc:
            return orders.OrderBy(o => o.Price);

        case Order.SortState.PriceDesc:
            return orders.OrderByDescending(o => o.Price);

        case Order.SortState.EmployeeAsc:
            return orders.OrderBy(o => o.Employee.FullName);

        case Order.SortState.EmployeeDesc:
            return orders.OrderByDescending(o => o.Employee.FullName);

        default:
            return orders.OrderBy(o => o.DateOrder);
    }
}
}
}

```

HomeController.cs

```

using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}

```

EmployeesController.cs

```

using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;

namespace RepairServiceCenterASP.Controllers
{
    public class EmployeesController : Controller
    {
        private readonly RepairServiceCenterContext _context;

        public EmployeesController(RepairServiceCenterContext context)
        {
            _context = context;
        }

        // GET: Employees
        public async Task<IActionResult> Index(string fullName, int? experience, int page = 1,
            Employee.SortState sortOrder = Employee.SortState.FullNameAsc)
        {
            int pageSize = 10;

            IQueryable<Employee> source = _context.Employees.Include(e => e.Post);

            if (!String.IsNullOrEmpty(fullName))
                source = source.Where(e => e.FullName.Contains(fullName));

            if (experience != null)
                source = source.Where(e => e.Experience == experience.Value);

            source = EmployeesSort(source, sortOrder);

            int count = await source.CountAsync();
            var items = await source.Skip((page - 1) * pageSize).Take(pageSize).ToListAsync();

            EmployeesViewModel viewModel = new EmployeesViewModel()
            {
                EmployeesSort = new EmployeesSort(sortOrder),
                Employees = items,
                PageViewModel = new PageViewModel(count, page, pageSize),
                EmployeesFilter = new EmployeesFilter(fullName, experience)
            };

            return View(viewModel);
        }

        // GET: Employees/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
        }
    }
}

```

```

var employee = await _context.Employees
    .Include(e => e.Post)
    .FirstOrDefaultAsync(m => m.EmployeeId == id);
if (employee == null)
{
    return NotFound();
}

return View(employee);
}

// GET: Employees/Create
public IActionResult Create()
{
    ViewData["PostId"] = new SelectList(_context.Posts, "PostId", "Name");
    return View();
}

// POST: Employees/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("EmployeeId,FullName,Experience,PostId")] Employee em-
ployee)
{
    if (ModelState.IsValid)
    {
        _context.Add(employee);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["PostId"] = new SelectList(_context.Posts, "PostId", "Name", employee.PostId);
    return View(employee);
}

// GET: Employees/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var employee = await _context.Employees.FindAsync(id);
    if (employee == null)
    {
        return NotFound();
    }
    ViewData["PostId"] = new SelectList(_context.Posts, "PostId", "Name", employee.PostId);
    return View(employee);
}

// POST: Employees/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("EmployeeId,FullName,Experience,PostId")] Employee em-
ployee)
{
    if (id != employee.EmployeeId)

```

```

    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(employee);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EmployeeExists(employee.EmployeeId))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["PostId"] = new SelectList(_context.Posts, "PostId", "Name", employee.PostId);
    return View(employee);
}

// GET: Employees/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var employee = await _context.Employees
        .Include(e => e.Post)
        .FirstOrDefaultAsync(m => m.EmployeeId == id);
    if (employee == null)
    {
        return NotFound();
    }

    return View(employee);
}

// POST: Employees/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var employee = await _context.Employees.FindAsync(id);
    _context.Employees.Remove(employee);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool EmployeeExists(int id)
{
    return _context.Employees.Any(e => e.EmployeeId == id);
}

```



```

private IQueryable<Employee> EmployeesSort(IQueryable<Employee> source, Employee.SortState sortOrder)
{
    switch (sortOrder)
    {
        case Employee.SortState.FullNameAsc:
            return source.OrderBy(e => e.FullName);

        case Employee.SortState.FullNameDesc:
            return source.OrderByDescending(e => e.FullName);

        case Employee.SortState.ExperienceAsc:
            return source.OrderBy(e => e.FullName);

        case Employee.SortState.ExperienceDesc:
            return source.OrderByDescending(e => e.Experience);

        case Employee.SortState.PostAsc:
            return source.OrderBy(e => e.Post.Name);

        case Employee.SortState.PostDesc:
            return source.OrderByDescending(e => e.Post.Name);
        default:
            return source;
    }
}
}
}

```

DbInitializerExtensions.cs

```

using Microsoft.AspNetCore.Builder;

namespace RepairServiceCenterASP.Middleware
{
    public static class DbInitializerExtensions
    {
        public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<DbInitializerMiddleware>();
        }
    }
}

```

DbInitializerMiddleware.cs

```

using Microsoft.AspNetCore.Http;
using System;
using System.Linq;
using System.Threading.Tasks;
using RepairServiceCenterASP.Data;

namespace RepairServiceCenterASP.Middleware
{
    public class DbInitializerMiddleware
    {
        private readonly RequestDelegate _next;

        public DbInitializerMiddleware(RequestDelegate next)
        {
            // инициализация базы данных по университетам

```

```

        _next = next;
    }
    public Task Invoke(HttpContext context, IServiceProvider serviceProvider, RepairServiceCenterContext dbContext)
    {
        if (!context.Session.Keys.Contains("starting"))
        {
            DbInitializer.Initialize(dbContext);
            context.Session.SetString("starting", "Yes");
        }

        // Call the next delegate/middleware in the pipeline
        return _next.Invoke(context);
    }
}

```

Startup.cs

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using RepairServiceCenterASP.Data;
using RepairServiceCenterASP.Middleware;
using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.Services;

namespace RepairServiceCenterASP
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            string connection = Configuration.GetConnectionString("DefaultConnection");
            services.AddDbContext<RepairServiceCenterContext>(options => options.UseSqlServer(connection));
            services.AddMemoryCache();
            services.AddSession();
            services.AddTransient<ICachingModel<RepairedModel>, RepeiredModelService>();
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                //app.UseBrowserLink();
            }
            else
            {

```

```

        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseSession();

    app.UseDbInitializer();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
    }
}

```

Program.cs

```

using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace RepairServiceCenterASP
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseUrls("http://192.168.43.20:5000")
                .UseStartup<Startup>();
    }
}

```

PageViewModel.cs

```

using System;

namespace RepairServiceCenterASP.ViewModels
{
    public class PageViewModel
    {
        public int PageNumber { get; private set; }
        public int TotalPages { get; private set; }

        public PageViewModel(int count, int pageNumber, int pageSize)
        {
            PageNumber = pageNumber;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);
        }

        public bool HasPreviousPage
        {
            get
            {

```

```

        return (PageNumber > 1);
    }
}

public bool HasNextPage
{
    get
    {
        return (PageNumber < TotalPages);
    }
}
}
}

```

OrdersViewModel.cs

```

using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace RepairServiceCenterASP.ViewModels
{
    public class OrdersViewModel
    {
        public OrdersFilter OrdersFilter { get; set; }
        public OrdersSort OrdersSort { get; set; }
        public IEnumerable<Order> Orders { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

EmployeesViewModel.cs

```

using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels
{
    public class EmployeesViewModel
    {
        public EmployeesSort EmployeesSort { get; set; }
        public EmployeesFilter EmployeesFilter { get; set; }
        public IEnumerable<Employee> Employees { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

PostViewModel.cs

```

using RepairServiceCenterASP.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

```

```

namespace RepairServiceCenterASP.ViewModels
{
    public class PostsViewModel
    {
        public IEnumerable<Post> Posts { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

RepeiredViewModels.cs

```

using RepairServiceCenterASP.Models;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels
{
    public class RepeiredViewModels
    {
        public IEnumerable<RepairedModel> RepairedModels { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

ServiceStoresViewModel.cs

```

using RepairServiceCenterASP.Models;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels
{
    public class ServicedStoresViewModel
    {
        public IEnumerable<ServicedStore> ServicedStores { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

SparePartsViewModels.cs

```

using RepairServiceCenterASP.Models;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels
{
    public class SparePartsViewModels
    {
        public IEnumerable<SparePart> SpareParts { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

TypeOfFaultsViewModel.cs

```

using RepairServiceCenterASP.Models;
using RepairServiceCenterASP.ViewModels.Filters;
using RepairServiceCenterASP.ViewModels.Sortings;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels
{
    public class TypeOfFaultsViewModel

```

```

    {
        public TypesOfFaultsFilter TypesOfFaultsFilter { get; set; }
        public TypesOfFaultsSort TypesOfFaultsSort { get; set; }
        public IEnumerable<TypeOfFault> TypeOfFaults { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

TypesOfFaultsSort.cs

```
using RepairServiceCenterASP.Models;
```

```
namespace RepairServiceCenterASP.ViewModels.Sortings
```

```

{
    public class TypesOfFaultsSort
    {
        public TypeOfFault.SortState RepairedModelSort { get; set; }
        public TypeOfFault.SortState NameSort { get; set; }
        public TypeOfFault.SortState MethodRepairSort { get; set; }
        public TypeOfFault.SortState WorkPriceSort { get; set; }
        public TypeOfFault.SortState Current { get; private set; }

        public TypesOfFaultsSort(TypeOfFault.SortState sortOrder)
        {
            RepairedModelSort = sortOrder == TypeOfFault.SortState.RepairedModelAsc ? TypeOfFault.SortState.Re-
repairedModelDesc
                                : TypeOfFault.SortState.RepairedModelAsc;

            NameSort = sortOrder == TypeOfFault.SortState.NameAsc ? TypeOfFault.SortState.NameDesc
                                : TypeOfFault.SortState.NameAsc;

            MethodRepairSort = sortOrder == TypeOfFault.SortState.MethodRepairAsc ? TypeOfFault.Sort-
State.MethodRepairDesc
                                : TypeOfFault.SortState.MethodRepairAsc;

            WorkPriceSort = sortOrder == TypeOfFault.SortState.WorkPriceAsc ? TypeOfFault.SortState.WorkPriceDesc
                                : TypeOfFault.SortState.WorkPriceAsc;

            Current = sortOrder;
        }
    }
}

```

OrdersSort.cs

```
using RepairServiceCenterASP.Models;
```

```
namespace RepairServiceCenterASP.ViewModels.Sortings
```

```

{
    public class OrdersSort
    {
        public Order.SortState DateOrderSort { get; private set; }
        public Order.SortState ReturnDateSort { get; private set; }
        public Order.SortState FullNameCustSort { get; private set; }
        public Order.SortState RepModelSort { get; private set; }
        public Order.SortState TypeOfFaultSort { get; private set; }
        public Order.SortState ServicedStoreSort { get; private set; }
        public Order.SortState GuaranteeMarkSort { get; private set; }
        public Order.SortState GuaranteePeriodSort { get; private set; }
        public Order.SortState PriceSort { get; private set; }
        public Order.SortState EmployeeSort { get; private set; }
    }
}

```

```

public Order.SortState Current { get; private set; }

public OrdersSort(Order.SortState sortOrder)
{
    DateOrderSort = sortOrder == Order.SortState.DateOrderAsc ? Order.SortState.DateOrderDesc
        : Order.SortState.DateOrderAsc;

    ReturnDateSort = sortOrder == Order.SortState.ReturnDateAsc ? Order.SortState.ReturnDateDesc
        : Order.SortState.ReturnDateAsc;

    FullNameCustSort = sortOrder == Order.SortState.FullNameCustAsc ? Order.SortState.FullNameCustDesc
        : Order.SortState.FullNameCustAsc;

    RepModelSort = sortOrder == Order.SortState.RepModelAsc ? Order.SortState.RepModelDesc
        : Order.SortState.RepModelAsc;

    TypeOfFaultSort = sortOrder == Order.SortState.TypeOfFaultAsc ? Order.SortState.TypeOfFaultDesc
        : Order.SortState.TypeOfFaultAsc;

    ServicedStoreSort = sortOrder == Order.SortState.ServiceStoreAsc ? Order.SortState.ServiceStoreDesc
        : Order.SortState.ServiceStoreAsc;

    GuaranteeMarkSort = sortOrder
        == Order.SortState.GuaranteeMarkAsc ? Order.SortState.GuaranteeMarkDesc
        : Order.SortState.GuaranteeMarkAsc;

    GuaranteePeriodSort = sortOrder
        == Order.SortState.GuaranteePeriodAsc ? Order.SortState.GuaranteePeriodDesc
        : Order.SortState.GuaranteePeriodAsc;

    PriceSort = sortOrder == Order.SortState.PriceAsc ? Order.SortState.PriceDesc
        : Order.SortState.PriceAsc;

    EmployeeSort = sortOrder == Order.SortState.EmployeeAsc ? Order.SortState.EmployeeDesc
        : Order.SortState.EmployeeAsc;

    Current = sortOrder;
}
}
}

```

EmployeesSort.cs

```

using RepairServiceCenterASP.Models;

namespace RepairServiceCenterASP.ViewModels.Sortings
{
    public class EmployeesSort
    {
        public Employee.SortState FullNameSort { get; private set; }
        public Employee.SortState ExperienceSort { get; private set; }
        public Employee.SortState PostSort { get; private set; }
        public Employee.SortState Current { get; private set; }

        public EmployeesSort(Employee.SortState sortOrder)
        {
            FullNameSort = sortOrder == Employee.SortState.FullNameAsc ? Employee.SortState.FullNameDesc
                : Employee.SortState.FullNameAsc;

            ExperienceSort = sortOrder == Employee.SortState.ExperienceAsc ? Employee.SortState.ExperienceDesc
                : Employee.SortState.ExperienceAsc;

```

```

        PostSort = sortOrder == Employee.SortState.PostAsc ? Employee.SortState.PostDesc
            : Employee.SortState.PostAsc;

        Current = sortOrder;
    }
}
}

```

TypesOfFaultsFilter.cs

```

using Microsoft.AspNetCore.Mvc.Rendering;
using RepairServiceCenterASP.Models;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels.Filters
{
    public class TypesOfFaultsFilter
    {
        public int? SelectedModel { get; private set; }
        public SelectList Models { get; private set; }
        public string InputName { get; private set; }
        public string InputMethodRepair { get; private set; }
        public string InputClient { get; private set; }

        public TypesOfFaultsFilter(List<RepairedModel> models, int? model, string name,
            string methodRepair, string client)
        {
            models.Insert(0, new RepairedModel() { RepairedModelId = 0, Name = "Bce" });
            Models = new SelectList(models, "RepairedModelId", "Name", model);

            InputName = name;
            InputMethodRepair = methodRepair;
            InputClient = client;
        }
    }
}

```

OrdersFilter.cs

```

using Microsoft.AspNetCore.Mvc.Rendering;
using RepairServiceCenterASP.Models;
using System;
using System.Collections.Generic;

namespace RepairServiceCenterASP.ViewModels.Filters
{
    public class OrdersFilter
    {
        public DateTime? SelectedDateOrder { get; private set; }
        public string InputFullNameCust { get; private set; }
        public int? SelectedEmployee { get; private set; }
        public SelectList Employees { get; private set; }
        public bool? SelectedGuarantee { get; private set; }

        public OrdersFilter(DateTime? dateOrder, string fullName, List<Employee> employees, int? employee,
            bool? guarantee)
        {
            employees.Insert(0, new Employee() { EmployeeId = 0, FullName = "Bce" });
            Employees = new SelectList(employees, "EmployeeId", "FullName", employee);

            SelectedDateOrder = dateOrder;
        }
    }
}

```



```

        InputFullNameCust = fullName;
        SelectedEmployee = employee;
        SelectedGuarantee = guarantee;
    }
}
}

```

EmployeesFilter.cs

```

namespace RepairServiceCenterASP.ViewModels.Filters
{
    public class EmployeesFilter
    {
        public string InputFullName { get; private set; }
        public int? InputExp { get; private set; }

        public EmployeesFilter(string fullName, int? experience)
        {
            InputFullName = fullName;
            InputExp = experience;
        }
    }
}

```

ПРИЛОЖЕНИЕ Б
(обязательное)
Чертёж структуры *web*-приложения