**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение…………………………………………………………………………… | 4 |
| 1 Общие сведения об предприятии РУП «Минскэнерго» Филиал «Минские кабельные сети»…………………………………………………………………… | 6 |
| 1.1 История предприятия……………………………………………………. | 6 |
| 1.2 Сфера услуг предприятия……………………………………………….. | 7 |
| 2 Этапы разработки приложения…………………………………………………. | 9 |
| 2.1 Цели и задачи разрабатываемого программного продукта……………. | 9 |
| 2.2 Структура *MVC…………………………………………………………………..* | 9 |
| 2.3 Информационно-логическая модель приложения…………………….. | 11 |
| 2.4 Физическая модель базы данных……………………………………….. | 14 |
| 2.5 Файловая структура базы данных………………………………………. | 15 |
| 3 Структура приложения………………………………………………………….. | 17 |
| 3.1 Описание общей структуры веб-приложения………………………….. | 17 |
| 3.2 Описание классов для доступа к данным………………………………. | 17 |
| 3.3 Описание контроллеров…………………………………………………. | 18 |
| 3.4 Описание представлений………………………………………………... | 20 |
| 4 Аппаратное и программное обеспечение информационной системы……….. | 21 |
| 4.1 Требование к системному и прикладному обеспечению на стороне сервера хранилища данных…………………………………………………. | 21 |
| 4.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера……………………………………… | 21 |
| 4.3 Требования к системному и прикладному программному обеспечению на стороне клиента…………………………………………… | 21 |
| 4.4 Настройка и развёртывание приложения на сервере………………….. | 22 |
| Заключение………………………………………………………………………… | 23 |
| Список используемых источников……………………………………………….. | 24 |
| Приложение А Должностная инструкция инженера-программиста…………… | 25 |
| Приложение Б Правила охраны труда……………………………………………. | 28 |
| Приложение В Руководство пользователя………………………………………. | 31 |
| Приложение Г Руководство программиста……………………………………… | 35 |
| Приложение Д Листинг программы……………………………………………… | 37 |

# ВВЕДЕНИЕ

Важнейшим этапом подготовки будущего специалиста является технологическая практика, позволяющая сформировать необходимые практические навыки на основе полученных теоретических знаний. Основными целями технологической практики являются:

– ознакомление с предприятиями и условиями труда в выбранной профессии;

– получение практических навыков работы в должности инженера-программиста;

– освоение в реальных условиях предприятия принципов организации и управления производством, анализа различных показателей, проводимых мероприятий для повышения качества разрабатываемого ПО при снижении сроков его создания;

– закрепление полученных в процессе обучения знаний и навыков в области разработки ПО и использовании современных информационных систем и технологий в проектировании и производстве;

– проверка возможности самостоятельной работы в условиях конкретной организации.

Во время прохождения практики студенты находятся в условиях реальной работы, что позволяет им получить дополнительные навыки в профессиональной деятельности, получить новые знания и опыт.

Производственная практика содержит в себе перечень следующих задач:

– изучение требований к промышленному программированию;

– получение практических знаний в области программной инженерии;

– получение практических навыков в разработке проектных решений, знакомство с конкретными проектами и особенностями их реализации, сопровождения и эксплуатации;

– знакомство с современным уровнем научно-исследовательских и проектно-конструкторских работ в области информационных технологий.

Теоретическое обучение направлено на получение знаний и развивает в студенте способности самостоятельного обучения и саморазвития.

Во время практической подготовки, обеспечивается подготовка студента профессиональными знаниями и умениями, которые должны охватывать всю будущую деятельности специалиста.

Таким образом, теоретические знания студенты могут приобрести во время лекционных занятий, выполнения заданий на лабораторных, практических работах, а также курсовых проектах. Практическая подготовка студента осуществляется во время производственных практик.

В качестве объекта практики, выступает учебный проект, задачей которого является проектирование и создание базы данных в выбранной СУБД и разработка веб-приложения, которое обеспечивает отображение, редактирование и обработку информации из разработанной базы данных.

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).

Для решения поставленной задачи в качестве СУБД используется *MS SQL Server*. Данная СУБД обеспечивает поддержку баз данных очень большого объёма 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 ОБЩИЕ СВЕДЕНИЯ ОБ ПРЕДПРИЯТИИ РУП «МИНСКЭНЕРГО» ФИЛИАЛ «МИНСКИЕ КАБЕЛЬНЫЕ СЕТИ»**

**1.1 История предприятия**

Начало использования электроэнергии в Минске от стационарных источников, зафиксированное историческими документами, относится к январю 1895 года, когда на электрическо-водопроводной станции – водокачальне “Эльвод”- были введены в эксплуатацию первые источники электроэнергии – паровые машины по 100 и 50 л.с. с динамомашинами 110 Вольт постоянного тока.

В 1895 году была введена в эксплуатацию ГЭС-1 в г. Минске. Реконструкция 1927 и расширение, с пуском трамвайного транспорта, 30-х годов увеличили мощность ГЭС-1 до 6,7 МВт. В марте 1934 года пущена первая очередь Минской ГЭС-2 (впоследствии ТЭЦ-2) мощностью 6,8 МВт. В 1940 году велись работы по пуску второй очереди. Годовая выработка электроэнергии двумя ГЭС составила 82,4 МВт.ч. В то время в городе были РП – 4 шт., ТП –240 шт., кабельные линии 6 кВ – 52 км, воздушных линий 6 кВ –115 км, ВЛ 0,22-0,4 кВ – 63 км.

Сводных данных об энергетике Минской области в предвоенные годы нет. Локомобильные и дизельные установки по 54 –167 кВт в районных центрах с малочисленными низковольтными сетями были в составе местной промышленности. Они снабжали мелкие предприятия и незначительную часть населения.

1945 год – восстановлена и пущена Минская ТЭЦ-2. 1946 год – начато строительство Минской ТЭЦ-3. 1954 год – введена ВЛ 110 кВ Смолевичская ГРЭС –ПС Орша. Сформирована центральная часть республиканской энергосистемы. 1960-1961 годы – введена в эксплуатацию ПС 220 кВ «Колядичи». Начато формирование сети 220 кВ. 1963 год – пущен 8-й агрегат Минской ТЭЦ-3, станция достигла мощности 420 МВт, введена в эксплуатацию первая в республике ВЛ 330 кВ Колядичи-Вильнюс. Энергосистема Беларуси соединена с энергосистемой Северо-Запада СССР.

1983 год – Введен в эксплуатацию уникальный теплофикационный комплекс на базе Минской ТЭЦ-4 и районных котельных г. Минска, в Минске введена единственная в республике кабельная линия 110 кВ.

1987 год – введена в габаритах 750 кВ ВЛ 330 кВ ПСБелорусская –Игналинская АЭС.

1986-1992 годы – введена первая в республике ПС 750 кВ Белорусская с ВЛ 750 кВ на Смоленскую АЭС.

1992 год, октябрь – введен 6-й энергоблок Минской ТЭЦ-4, станция достигла проектной мощности 1030 МВт, 1519 Гкал/ч.

1999 год, август – пущен первый энергоблок Минской ТЭЦ-5 мощностью 330 МВт.

2005 год – на Минской ТЭЦ-4 введена в экспуатацию детандер-генераторная установка суммарной мощностью 5 МВт.

2006-2007 годы –введены 1-й и 2-й пусковые комплексы по реконструкции ПС 330 кВ «Колядичи» с вводом автотрансформаторов №1 и №2 мощностью по 200 МВА. Таким образом, завершены работы по 2-й очереди реконструкции подстанции.

2009 год – на Минской ТЭЦ-3 введён в эксплуатацию крупнейший в республике парогазовый блок мощностью 230 МВт. На Жодинской ТЭЦ введен в эксплуатацию головной образец котлоагрегата Е 60-9,5-510, работающий на местных видах топлива (дробленый торфобрикет, древесная щепа), производительностью 60 тонн пара в час.

За 2019 г. переложено 67,042 км тепловых сетей в однотрубном исчислении при плане 65,0 км

**1.2 Сфера услуг предприятия**

Республиканское унитарное предприятие «Минскэнерго» – государственное предприятие, осуществляющее электроснабжение потребителей Минска и Минской области, теплоснабжение городов: Минск, Молодечно, Жодино, Борисов, Вилейка, Солигорск, Слуцк и поселков Руденск, Дружный, Свислочь. Входит в состав государственного производственного объединения «Белэнерго».

Филиал «Минские кабельные сети» (далее – Филиал «МКС») является частью единого производственно-технологического комплекса по производству, передаче и распределению электрической и тепловой энергии.

Основными целями деятельности филиала «Минские кабельные сети» являются:

–обеспечение надежного энергоснабжения потребителей;

–хозяйственная деятельность, направленная на получение прибыли;

–подключение электроустановок потребителей к электрическим сетям.

Основной вид деятельности Филиала – передача и распределение электроэнергии, техническое обслуживание (эксплуатация и ремонт), строительство, реконструкция и модернизация электрических сетей 0,4-110кВ, энергетического и технологического оборудования.

Филиал «МКС» обслуживает кабельные сети 0,4-110кВ (кроме ВЛ 35-110кВ), расположенные в административных границах города Минска, а также поселках Тростенец, Стиклево, Копище.

Все работы по ремонту и техническому обслуживанию подстанций 35-110кВ осуществляются производственными службами филиала, а по распределительным сетям напряжением 0,38-10кВ – районами электрических сетей (РЭС).

В структуру филиала входят 25 централизованных служб и отделов, 6 электросетевых районов, 2 аккредитованные лаборатории. Для работы с заказчиками в филиале создан центр обслуживания потребителей и отдел проектирования.

Филиал «Минские кабельные сети» осуществляет подключение электроустановок юридических лиц, индивидуальных предпринимателей и граждан к электрическим сетям и другие административные процедуры.

Центр обслуживания потребителей также ведет прием юридических лиц, индивидуальных предпринимателей и граждан по административным процедурам, оказывает услуги по согласованию и рассмотрению Технических условий, выданных энергоснабжающей организацией, проектной документации на отступление от технических условий, нормативов надежности.

Центр обслуживания потребителей осуществляет консультативную поддержку на каждом этапе подключения к электрическим сетям.

Услуги лаборатории для проведения входного контроля и испытания образцов силовых кабелей напряжением до 110 кВ включительно востребованы не только в г. Минске, но и во всей Республике.

Технологическое присоединение электроустановок осуществляется по принципу «одно окно». Потребителю необходимо только написать заявление и предоставить необходимые документы.

В РУП «Минскэнерго» эксплуатируется более 60 тыс. км электрических сетей, осуществляющих электроснабжение потребителей г. Минска и области, прием и передачу энергии в другие регионы.

Численность персонала энергосистемы – более 14 тыс. человек. РУП «Минскэнерго», расположенное в центре Беларуси, интегрирует областные энергосистемы в объединённую энергосистему республики. В его состав также включены основные линии электропередачи, связывающие республику с энергосистемами Российской Федерации и Прибалтийских государств, – ВЛ 750 кВ (подстанция – 750 кВ «Белорусская»).

РУП «Минскэнерго» входит в десятку крупнейших налогоплательщиков Беларуси.

**2 ЭТАПЫ РАЗРАБОТКИ ПРИЛОЖЕНИЯ**

**2.1 Цели и задачи разрабатываемого программного продукта**

На время прохождения практики, были выставлены следующие цели и задачи:

– изучить обязанности инженера-программиста;

– ознакомиться с заданной предметной областью;

– ознакомится с требованиями техники безопасности и охраны труда;

– разработать *web*-приложение 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.

Обязанности инженера-программиста подробно описано в приложении А. Правила техники безопасности и охраны труда подробно расписаны в приложении Б.

**2.2 Структура *MVC***

Шаблон *MVC* расшифровывается как *Model*–*View*–*Controller*. Это принцип построения архитектуры большого приложения, при котором оно разбивается на три части.

Первая часть содержит всю бизнес-логику приложения. Такая часть называется Модель (*Model*). В ней содержится код, который делает все то, для чего приложение создавалось. Эта часть наиболее независимая от остальных.

Вторая часть содержит все, что касается отображения данных пользователю. Такая часть называется Вид (*View*). Именно в ней содержится код, который управляет показом окон, страниц, сообщений и т.д.

Третья часть содержит код, который занимается обработкой действий пользователя. Любые действия пользователя, направленные на изменения модели, должны обрабатываться тут. Такая часть называется *Controller*.

Такой подход позволяет независимо делать три вещи: логику программы (*Model*), механизм показа всех данных программы пользователю (*View*), обрабатывать ввод/действия пользователя (*Controller*).

Модель – это самая независимая часть системы. Она не зависит от *View* и *Controller*. Модель не может использовать классы из разделов *View* и *Controller*.

Основное ограничение *View* – не может вносить изменения в модель. *View* могут обращаться к модели за данными или подписываться на события, но менять модель классы *View* не могут.

Основное ограничение контроллера – он не занимается отображением данных. Контроллер обрабатывает действия пользователя и меняет в соответствии с ними модель.

Использование паттерна *MVC* обусловлено требованием в задании.

Обработка запросов происходит с помощью соответствующих методов, которые вызываются в зависимости от типа поступившего запроса.

Последним компонентом является представление, и оно реализуется с помощью *html* страниц. Общая схема модели *MVC* приведена на рисунке 2.1.

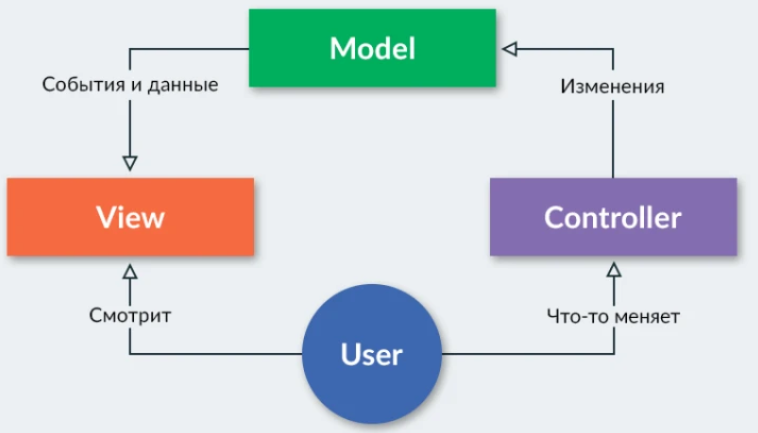


Рисунок 2.1 – Схема модели *MVC*

Для лучшего понимания взаимодействия между компонентами, необходимо рассмотреть структуру более подробно.

На рисунке 2.2, приведена структура типичной обработки запроса, поступившего на контроллер.

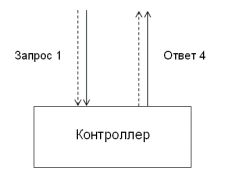


Рисунок 2.2 – Схема модели обработки запроса на контроллере

Контроллер, получивший запрос, анализирует его и, в зависимости от результатов обработки может выдать ответ об ошибке (например, при запросе несуществующей страницы отдать только *HTTP*-заголовок «404 *Not found*») или если запрос признан корректным, то, в зависимости от того, является он запросом на просмотр или на модификацию данных, контроллер вызывает соответствующий метод модели [5].

**2.3 Информационно-логическая модель приложения**

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 2.1. Данное отношение находится в первой нормальной форме.

Таблица 2.1 – Описание атрибутов сущности «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 2.2. Отношение по условию задачи должно содержать атрибуты: «Код запчасти», «Наименование», «Функции», «Наименование ремонтируемой модели». Данное отношение следует привести к третей 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 2.3, должна иметь список запчастей. В сущность «Запчасти» следует ещё добавить атрибут «Код вида неисправности», который будет являться внешним ключом.

Таблица 2.2 – Описание атрибутов сущности «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 2.3, 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 2.3 – Описание атрибутов сущности «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 описывание методов ремонта неисправностей. | Строка |
| Цена работы | Содержит стоимость работы для устранения неисправности. Вычисляется относительно стоимости запчастей. | Число с плавающей точкой |

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 2.4 – 2.6.

Таблица 2.4 – Описание атрибутов сущности «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 2.5 – Описание атрибутов сущности «Должности»

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

Таблица 2.6 – Описание атрибутов сущности «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 2.7, связанное связью «Один ко многим» с отношениями «Обслуживаемые магазины», «Виды неисправностей» и «Сотрудники».

Таблица 2.7 – Описание атрибутов сущности «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 |
| 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.

**2.4 Физическая модель базы данных**

По созданной информационно-логической модели была создана иерархия класса и контекст данных, 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 2.1.

На рисунке 2.1 изображена сгенерированная база данных с помощью *Entity Framework*.

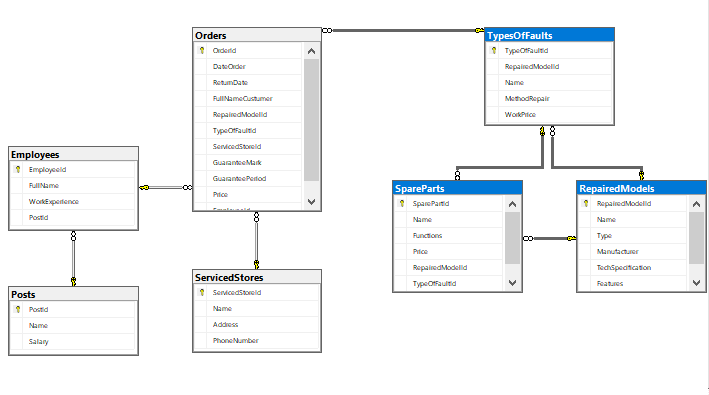


Рисунок 2.1 – Диаграмма базы данных

Для процесса преобразовании 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.

**2.5 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 взаимодействия 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. 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. 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. Также основным преимуществом данной технологии является использование универсального интерфейса для работы с данными, что позволяет легко и быстро сменить СУБД.

# 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 введённых 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 [*ControllerNa-me*] – 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 сотрудников, 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 моделей;

– *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, кешируется 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 котроллерам [5].

**4 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

**4.1 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 *x*64 с тактовой частотой 1,4 ГГц;

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

**4.2 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 *x*86/*x*64 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*, *Max OS*.

Так приложение разработана на платформе *.NET Core*, оно является кроссплатформенным и может быть запущенно 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 потребуется подключение к интернету, либо к локальной сети, в которой находится сервер хранилища данных. Так же системные требования могут изменятся относительно масштаба приложения.

**4.3 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).

**4.4 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 случаи Tomcat «*./webapp*») и выполнить команду «*dotnet RepairServiceCenterASP.dll*» с командной строки, рисунок 4.1, после этого веб-приложение будет запущенно на сервере. Чтобы пользователь мог использовать веб-приложение, он должен находится в одной сети с веб-сервером.

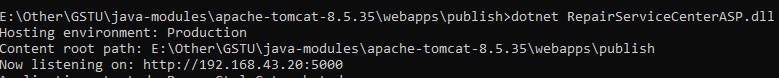


Рисунок 4.1 – Запуск веб-приложения на веб-сервере

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 находится в одной сети [4].

# ЗАКЛЮЧЕНИЕ

Технологическая практика является обязательным этапом всех студентов высших учебных заведений. На данном этапе обучения, студент способен проявить свои лучшие качества и применить навыки в работе.

В ходе производственной практики был получен опыт работы на предприятии. Результатом прохождения практики является веб-приложение, которое производит автоматизацию оформления заказов по ремонту оборудования. Приложение является простым и удобным благодаря адоптивному 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 автоматизироваться 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.

«Минскэнерго» в целях формирования высококвалифицированных кадров, эффективного использования кадрового потенциала особое внимание уделяет подбору, расстановке и профессиональному обучению кадров. Для выпускников учреждений образования проводятся семинары с участием руководителей подразделений предприятия и профсоюзных комитетов. Разработаны мероприятия по закреплению специалистов, создаются условия для их профессионального и карьерного роста. Осуществляются меры по защите социально-экономических и трудовых интересов молодежи, перспективным работникам оказывается содействие в получении первого высшего образования за счет средств предприятия с последующей обязательной отработкой с учетом полученной специальности, а также в продолжении обучения в магистратурах.

# Список используемых источников

1. Минскэнерго [Электронный ресурс]. – Режим доступа: https://minskenergo.by/osnovnye-svedeniya. – Дата доступа: 05.10.2020.

2. Основные этапы разработки web-приложений [Электронный ресурс]. – Режим доступа: http://www.rusnauka.com/Informatica/3\_85389.doc.html. – Дата доступа: 06.10.2020.

3. Гото К., Котлер Э. Веб-редизайн: книга Келли Гото и Эмили Котлер: Пер. с англ. – СПб. : Символ-Плюс, 2007. – 376 с.

4. Лопак Л. Web-дизайн. – 2-е изд.: Пер. с англ. – М. : Издательский дом «Вильямс», 2008. – 304 с.

5. Что такое MVC: базовые концепции и пример приложения [Электронный ресурс]. – Режим доступа: https://skillbox.ru/media/code/chto\_takoe\_mvc\_bazovye\_kontseptsii\_i\_primer\_prilozheniya/. – Дата доступа: 08.10.2020.

6. Тепляков, С. Паттерны проектирования на платформе .NET. / С. Тепляков – СПб. : Питер, 2015. – 320 с.

7. Фиайли, К. SQL. Руководство по изучению языка: Пер. с англ. / К. Фиайли – М. : ДМК Пресс, 2013. – 456 с.

8. Илюшечкин, В. М. Основы использования и проектирования баз данных / В.М. Илюшечкин. – М. : Юрайт, 2015. – 212 c.

9. Винниченко «Автоматизация процессов тестирования». - СПб. : Питер, 2014. – 655 с.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Должностная инструкция инженера-программиста**

Общие положения:

а) инженер-программист относится к категории специалистов, принимается на работу и увольняется с работы приказом директора предприятия;

б) на должность инженера-программиста назначается лицо, имеющее высшее техническое или инженерно-экономическое образование без предъявления требований к стажу работы или среднее специальное образование и стаж работы в должности техника I категории не менее 3 лет либо на других должностях, замещаемых специалистами со средним специальным образованием, не менее 5 лет.

На должность инженера-программиста II категории назначается лицо, имеющее высшее техническое или инженерно-экономическое образование и стаж работы в должности инженера-программиста или на других инженерно-техниче-ских должностях, замещаемых специалистами с высшим образованием, не менее 3 лет.

На должность инженера-программиста I категории назначается лицо, имеющее высшее техническое или инженерно-экономическое образование и стаж работы в должности инженера-программиста II категории не менее 3 лет.

в) в своей деятельности инженер-программист руководствуется:

1) нормативными документами по вопросам выполняемой работы;

2) методическими материалами, касающимися соответствующих вопросов;

3) уставом предприятия;

4) правилами трудового распорядка;

5) приказами и распоряжениями директора предприятия (непосредственного руководителя);

6) настоящей должностной инструкцией;

г) инженер-программист должен знать:

1) постановления, распоряжения, приказы и другие руководящие и нормативные документы, касающиеся методов программирования и использования вычислительной техники при обработке информации;

2) технико-эксплуатационные характеристики, конструктивные особенности, назначение и режимы работы оборудования, правила его технической эксплуатации;

3) технологию механизированной обработки информации;

4) виды технических носителей информации;

5) методы классификации и кодирования информации;

6) формализованные языки программирования;

7) действующие стандарты, системы счислений, шифров и кодов;

8) методы программирования;

9) порядок оформления технической документации;

10) передовой отечественный и зарубежный опыт программирования и использования вычислительной техники;

11) основы экономики, организации труда и организации производства;

12) правила и нормы охраны труда, техники безопасности, производственной санитарии и противопожарной защиты;

д) Во время отсутствия инженера-программиста его обязанности выполняет в установленном порядке назначаемый заместитель, несущий полную ответственность за надлежащее исполнение возложенных на него обязанностей.

На инженера-программиста возлагаются следующие функции:

– разработка программ, направленных на решение экономических и иных задач;

– отладка программ;

– сопровождение внедренных программ и программных средств;

– участие в разработке форм документов, подлежащих машинной обработке.

Для выполнения возложенных на него функций инженер-программист обязан:

– на основе анализа математических моделей и алгоритмов разрабатывать программы, реализующие решение экономических и других задач;

– разрабатывать технологию, этапы и последовательность решения;

– осуществлять выбор языка программирования и перевод на него используемых моделей и алгоритмов задач;

– определять информацию, подлежащую обработке на ЭВМ, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и выдачи информации, методы ее контроля;

– определять объем и содержание данных тестовых примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;

– выполнять работу по подготовке программ к отладке и проводить отладку;

– разрабатывать инструкции по работе с программами, оформлять необходимую техническую документацию;

– определять возможность использования готовых программных средств;

– осуществлять сопровождение внедренных программ и программных средств;

– проводить камеральную проверку программ на основе логического анализа;

– определять совокупность данных, обеспечивающих решение максимального числа условий, включенных в программу, выполнять работу по ее подготовке к отладке;

– проводить отладку разработанных программ, корректировать их в процессе доработки;

– разрабатывать и внедрять методы автоматизации программирования, типовые и стандартные программы, программирующие программы, трансляторы, входные алгоритмические языки;

– выполнять работу по унификации и типизации вычислительных процессов;

– принимать участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектных работах по расширению области применения вычислительной техники.

Инженер-программист имеет право:

– знакомиться с проектами решений руководства предприятия, касающимися его деятельности;

– получать от руководителей структурных подразделений, специалистов информацию и документы, необходимые для выполнения своих должностных обязанностей;

– привлекать специалистов всех структурных подразделений предприятия для решения возложенных на него обязанностей (если это предусмотрено положениями о структурных подразделениях, если нет – с разрешения руководителя предприятия);

– требовать от руководства предприятия оказания содействия в исполнении своих должностных обязанностей и прав.

Инженер-программист несет ответственность:

– за неисполнение (ненадлежащее исполнение) своих должностных обязанностей, предусмотренных настоящей должностной инструкцией, в пределах, определенных действующим трудовым законодательством Республики Беларусь;

– за совершенные в процессе осуществления своей деятельности правонарушения – в пределах, определенных действующим административным, уголовным и граждански законодательством Республики Беларусь;

за причинение материального ущерба – в пределах, определенных действующим трудовым, уголовным и гражданским законодательством Республики Беларусь

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Правила охраны труда**

Охрана труда – это система обеспечения безопасности жизни и здоровья, работающих в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-гигиенические, реабилитационные и иные мероприятия и средства.

Основными принципами государственной политики в области охраны труда являются:

– приоритет жизни и здоровья работников по отношению к результатам производственной деятельности;

– обеспечение гарантий права работников на охрану труда;

– установление обязанностей всех субъектов правоотношений в области охраны труда, полной ответственности работодателей за обеспечение здоровых и безопасных условий труда;

– совершенствование правоотношений и управления в этой сфере, включая внедрение экономического механизма обеспечения охраны труда.

Основными направлениями государственной политики в области охраны труда являются:

– приоритет сохранения жизни и здоровья работающих;

– ответственность работодателя за создание здоровых и безопасных условий труда;

– комплексное решение задач охраны труда на основе республиканских, отраслевых и территориальных целевых программ по улучшению условий и охраны труда с учетом направлений экономической и социальной политики, достижений в области науки и техники;

– социальная защита работающих, возмещение вреда лицам, потерпевшим при несчастных случаях на производстве и (или) получившим профессиональные заболевания;

– установление единых требований по охране труда для всех работодателей; использование экономических методов управления охраной труда, участие государства в финансировании мероприятий по улучшению условий и охраны труда;

– информирование граждан, обучение работающих по вопросам охраны труда;

– взаимодействие республиканских органов государственного управления и иных государственных организаций, подчиненных Правительству Республики Беларусь, местных исполнительных и распорядительных органов, органов, уполномоченных на осуществление контроля (надзора), профессиональных союзов, работодателей;

– сотрудничество между работодателями и работающими.

– использование международного опыта организации работы по улучшению условий и повышению безопасности труда.

*IT*-специалисты, также могут быть подвержены определённому риску, поэтому, они должны изучить правила охраны труда либо в учебном центре, либо в рамках самой организации.

Кроме того, *IT*-специалисты имеют свою группу по электробезопасности:

– группа I – неэлектротехническому персоналу (тем, кто напрямую не связан с электроустановками, но имеют риск поражения электрическим током);

– II и выше – электротехническому и электротехнологическому персоналу (тем, кто непосредственно занят эксплуатацией или наладкой электрооборудования).

Чтобы присвоить работникам группу I по электробезопасности, не нужно их специально обучать – достаточно инструктажа. Провести его должен ответственный за электрохозяйство компании сотрудник с группой по электробезопасности не ниже III. Инструктаж должен повторяться не реже одного раза в год.

Закон обязывает работодателя направлять на предварительный и периодические медосмотры сотрудников, которые проводят за компьютером более 50 процентов рабочего времени. Труд *IT*-специалистов почти непрерывно связан с компьютером. Таким образом, медосмотры им нужны.

Микроклимат офисных помещений должен соответствовать нормам Сан-Пин 2.2.4.548-96. Если работа за компьютером является вспомогательной, то температура воздуха должна быть допустимой (20 – 21,9 °С), если работа является основной – то оптимальной (22 – 24 °С).

В офисе нужно провести замеры температуры сертифицированным прибором. Если она меньше оптимальной или допустимой, необходимо сообщить об этом руководству и потребовать привести условия труда в норму.

Важно создать сбалансированный режим труда и отдыха, предварительно проанализировав загрузку *IT*-специалистов. Сотрудники, работающие за

компьютером, должны делать регулярные перерывы. Помещение, оборудованное компьютерами, нужно проветривать каждый час, а специалистам рекомендуется чередовать работу за компьютером и без него. Если такой возможности нет, лучше делать 10 – 15-минутные перерывы через каждые 45–60 минут работы.

Как правило, IT-специалисты трудятся по пятидневной рабочей неделе с двумя выходными днями. Но если производство непрерывное, такие сотрудники могут трудиться и посменно. В таком случае о графике работы сотрудников оповещают не менее, чем за месяц. Если специалиста нужно привлечь к работе не по графику (например, до начала смены или после, в выходные дни), должен соблюдаться порядок, установленный Трудовым кодексом.

*IT*-специалистам порой приходится сталкиваться с ситуациями, когда требуется задерживаются в офисе в период сдачи срочного проекта. В этом случае нужно оформить переработку и оплатить ее.

Работа будет сверхурочной, только если сотрудник выполняет ее по инициативе руководителя по окончании трудового дня (смены) или сверх нормального числа рабочих часов за учетный период. Переработку компенсируют за:

– первые два часа работы – не менее чем в полуторном размере;

– последующие часы – не менее чем в двойном размере.

По желанию работника денежную компенсацию можно заменить дополнительным отдыхом продолжительностью не менее отработанного времени.

Чтобы привлечь сотрудника к работе в выходной или праздничный день, нужно оформить следующие документы:

– уведомление о работе в нерабочий праздничный день (с обязательным указанием на право сотрудника отказаться от такой работы);

– приказ о привлечении к работе в нерабочий праздничный день и предоставлении дополнительных дней отдыха за работу в нерабочий праздничный день;

– заявление работника о предоставлении другого дня для отдыха.

Трудовой режим работника должен быть четко прописан в его трудовом договоре, должностной инструкции и локальных нормативных актах организации (например, в Правилах трудового распорядка). За их нарушения к работнику могут применяться дисциплинарные взыскания.

**ПРИЛОЖЕНИЕ В**

(обязательное)

# Руководство пользователя

**1 Описание приложения**

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

## **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.

## **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 Описание операции по обработки данных**

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

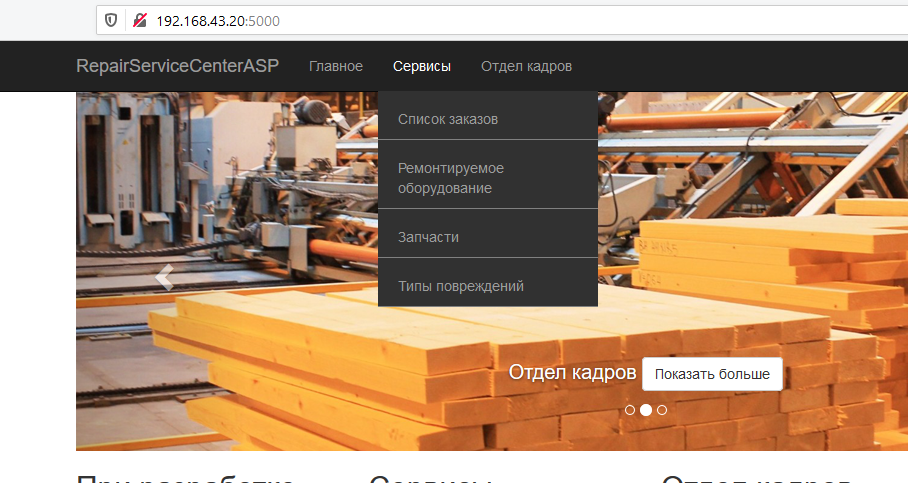


Рисунок 4.1 – Выбор сервиса

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

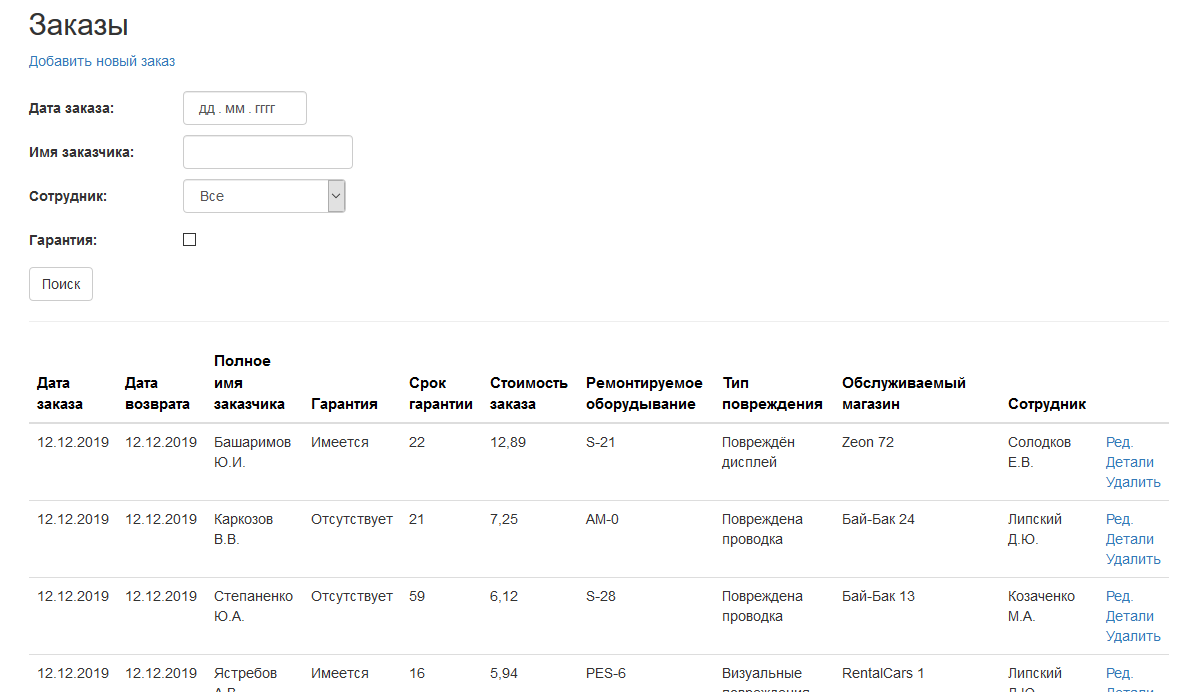


Рисунок 4.2 – Список заказов по ремонту оборудования

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 случаи – «Заказы»), затем загрузится новая страница, рисунок 4.3 с формами для оформления заказа (все поля формы имеют проверку на корректность введённых данных). Чтобы окончательно оформить заказ, нужно нажать на кнопку «Добавить».

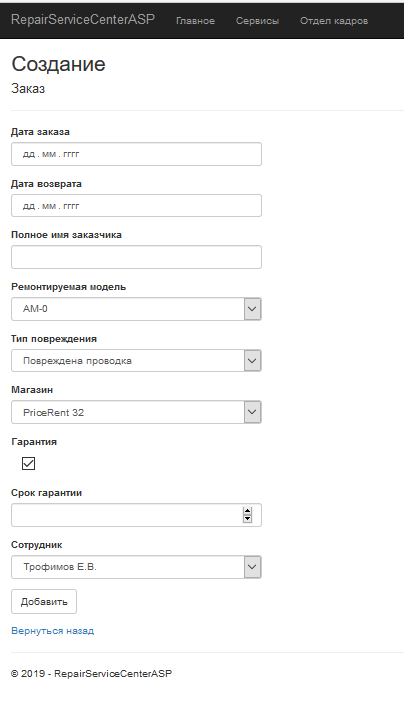


Рисунок 4.3 – Форма для добавления нового заказа

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.



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.

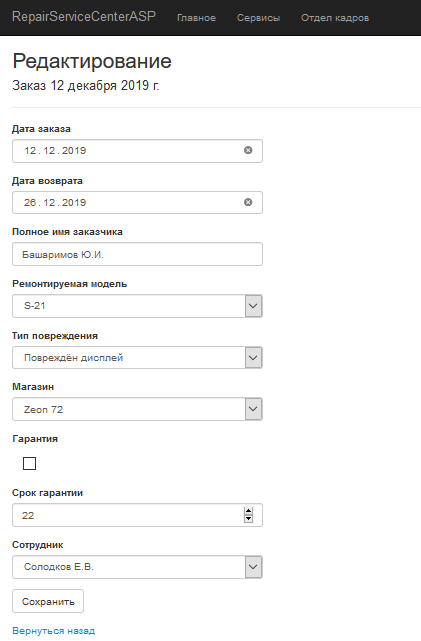


Рисунок 4.5 – Окно для редактирования данных заказе

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

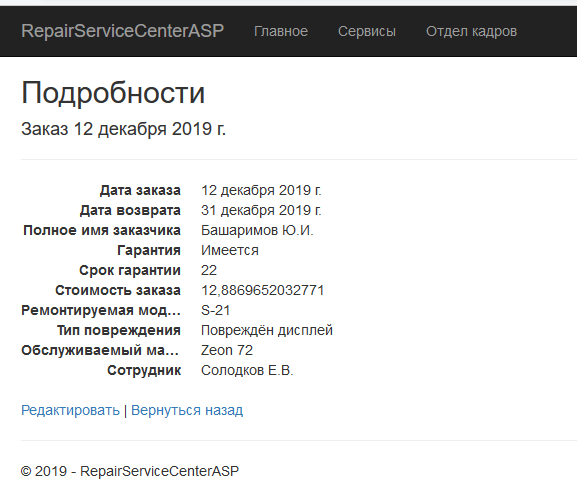


Рисунок 4.6 – Подробности заказа

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

**ПРИЛОЖЕНИЕ Г**

(обязательное)

# Руководство программиста

## **1 Назначения и условия применения программы**

Приложение предназначена для предоставления информации из базы данных предприятиям 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.

## **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*.

## **3 Сопровождение программного комплекса**

Для дополнения программного обеспечения новым функционалом можно использовать любую среду разработки на языке программирования *C#.* Приложения 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.

## **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 Сообщения в ходе работы приложения**

При работе программа может оповещать пользователя о следующих неполадках:

– некорректно введённые данных при добавлении и редактировании записей;

– некорректный *URL*-адрес, страница не найдена;

– ошибка при добавлении записей, запись с введёнными значениями уже существуют в базе.

Данные сообщения передаются в специальном виде ошибки с описанием проблемы.

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**Листинг 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, ServiceStroeDesc,

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 FullNameCustumer { 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);

}

}

**IChachingModel.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; }

}

}

**DbInitalizer.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,

FullNameCustumer = 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 = "Прогроммист",

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",

"БелГосСтах", "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 repeiredModel = 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 manafacture;

string techSpecification;

string features;

string[] namesVoc = { "FE-", "AM-", "S-", "A-", "PES-", "CAT-", "DOG-", "T-" };

string[] typesVoc = { "Производственное", "Медицинское", "Строительное", "Военное" };

string[] manafacturesVoc = { "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))];

manafacture = manafacturesVoc[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 = manafacture,

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.FullNameCustumer.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,WorkPrice")] 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,MethodRepair,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")] ServicedStore 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.FullNameCustumer.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,FullNameCustumer,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,FullNameCustumer," +

"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.FullNameCustumer);

case Order.SortState.FullNameCustDesc:

return orders.OrderByDescending(o => o.FullNameCustumer);

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 employee)

{

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 employee)

{

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);

}

}

}

**Sturtup.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.RepairedModelDesc

: TypeOfFault.SortState.RepairedModelAsc;

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

: TypeOfFault.SortState.NameAsc;

MethodRepairSort = sortOrder == TypeOfFault.SortState.MethodRepairAsc ? TypeOfFault.SortState.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.ServiceStroeDesc

: 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 = "Все" });

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 = "Все" });

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? experiance)

{

InputFullName = fullName;

InputExp = experiance;

}

}}