**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Разработка приложений баз данных для информационных систем»

на тему: «**РАЗРАБОТКА *WEB*-ПРИЛОЖЕНИЯ БАЗ ДАННЫХ «оЦЕНКА ЭФФЕКТИВНОСТИ ДЕЯТЕЛЬНОСТИ ПОДРАЗДЕЛЕНИЙ И СОТРУДНИКОВ»**

Исполнитель: студент гр. ИТП–31

Солодков М.А

Руководитель: доцент

Асенчик О.Д.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2019

**СОДЕРЖАНИЕ**

[Введение …………………………………………………………………………...5](#_Toc27103113)

[1 Логическая и физическая структура базы данных 7](#_Toc27103114)

[1.1 Информационно-логическая модель информационной системы 7](#_Toc27103115)

[1.2 Физическая модель базы данных 15](#_Toc27103116)

[1.3 Файловая структура базы данных 17](#_Toc27103117)

[2 Аппаратное и программное обеспечение информа- ционной системы 18](#_Toc27103118)

[2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных 18](#_Toc27103119)

[2.2 Требования к системному и прикладному программному   
обеспечению на стороне *web*-сервера 18](#_Toc27103120)

[2.3 Требования к системному и прикладному программному   
обеспечению на стороне клиента 18](#_Toc27103121)

[2.4 Настройка и развёртывание приложения на сервере 19](#_Toc27103122)

[2.5 Описание контроллеров 21](#_Toc27103123)

[2.6 Описание представлений 23](#_Toc27103124)

[3 Руководство пользователя 25](#_Toc27103125)

[3.1 Назначение, условие применения и функционал 25](#_Toc27103126)

[3.2 Подготовка к работе 25](#_Toc27103127)

[3.3 Описание операции по обработки данных 26](#_Toc27103128)

[4 Руководство программиста 30](#_Toc27103129)

[4.1 Назначения и условия применения программы 30](#_Toc27103130)

[4.2 Характеристики программы 30](#_Toc27103131)

[4.3 Сопровождение программного комплекса 30](#_Toc27103132)

[4.4 Входные и выходные данные 31](#_Toc27103133)

[4.5 Сообщения в ходе работы приложения 31](#_Toc27103134)

[Заключение 32](#_Toc27103135)

[Список используемых источников 33](#_Toc27103136)

[Приложение А Код программы 34](#_Toc27103137)

[Приложение Б Чертёж структуры web-приложения 82](#_Toc27103138)

# ВВЕДЕНИЕ

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

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

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

Для данной задачи были определены следующие исходные данные:

– подразделения (наименование, количество сотрудников);

– фактические показатели подразделений (квартал, год, достигнутые показатели за год, достигнутые показатели за квартал);

– плановые показатели подразделений (квартал, год, достигнутые показатели за год, достигнутые показатели за квартал);

– сотрудники (ФИО, зарплата, возраст, рейтинг);

– фактические показатели сотрудников (квартал, год, достигнутые показатели за год, достигнутые показатели за квартал);

– плановые показатели сотрудников (квартал, год, достигнутые показатели за год, достигнутые показатели за квартал).

– прогресс сотрудников (прогресс).

– перечень показателей оценки деятельности подразделений (квартал, год, оценка дяетельности подразделений за год, оценка деятельности подразделений за квартал).

– перечень показателей оценки деятельности сотрудников (квартал, год, оценка дяетельности сотрудников за год, оценка деятельности сотрудников за квартал).

Также прилагаются следующие дополнительные требования к отображению данных:

– рейтинг всех сотрудников заданного подразделения за заданный период (квартал, год);

– рейтинг подразделения за текущий период (квартал, год) и аналогичный период прошлого года;

– рейтинг сотрудника за текущий период (квартал, год) и аналогичный период прошлого года.

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 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ**

## **1.1 Информационно-логическая модель информационной системы**

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

– «Подразделение»;

– «Фактические показатели подразделения»;

– «Плановые показатели подразделения»;

– «Сотрудник»;

– «Фактические показатели сотрудника»;

– «Плановые показатели сотрудника»;

– «Прогресс сотрудника»;

– «Перечень показателей оценки деятельности подразделений»;

– «Перечень плказателей оценки деятельности сотрудников»;

Для сущности «Подразделение» было создано отношение (таблица) с атрибутами: «Код подразделения», «Наименование», «Количество сотрудников». Подробное описание отношения и атрибутов приведено в таблице 1.1. Данное отношение находится в первой нормальной форме.

Таблица 1.1 – **Отношение описывающие сущность «Подразделения»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код подразделения | Уникальный инкрементируемый идентификатор для каждого подразделения. Является первичным ключом. | Целое число |
| Наименования | Содержит наименования подразделений, находящихся в компании | Строка |
| Количество сотрудников | Содержит количество сотрудников в определенном подразделении компании | Целое число |

Отношение для сущности «Фактические показатели подразделений», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «Код фактических показателей», «Квартал», «Год», «Фактические квартальные показатели», «Фактические годовые показатели», «Наименование подразделения». Данное отношение следует привести к третей нормальной форме, заменив атрибут «Наименование подразделения» на атрибут «Код подразделения» связав отношение «Подразделения» с отношением «Фактические показатели подразделений». Тем самым будет организованна связь один ко многим, между данными отношениями.

Таблица 1.2 – **Отношение описывающие сущность «Фактические показатели подразделений»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код фактических показателей | Уникальный инкрементируемый идентификатор для каждого показателя. Является первичным ключом. | Целое число |
| Квартал | Содержит неуникальные наименования кварталов. | Целое число |
| Год | Содержит неуникальные наименование годов. | Целое число |
| Фактические годовые показатели | Содержит числовые значения с плавающей точкой, которые являются фактическими годовыми показателями. | Число с плавающей точкой |
| Фактические плановые показатели | Содержит числовые значения с плавающей точкой, которые являются фактическими квартальными показателями. | Число с плавающей точкой |
| Код подразделения | Содержит ссылки на подразделения. Является внешним ключом для связи с отношением «Подразделения». | Целое число |

Отношение для «Плановых показателей подразделений», таблица 1.3, состоит из атрибутов: «Код вида плановых показателей», «Квартал», «Год», «Плановые годовые показатели подразделений», «Плановые квартальные показатели подразделений», «Код подразделения». Связано отношением «Один ко многим» с отношением, относящимся к сущностям «Подразделения».

Таблица 1.3 – **Отношение описывающие сущность «Плановые показатели подразделений»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код плановых показателей | Уникальный инкрементируемый идентификатор для каждого показателя. Является первичным ключом. | Целое число |
| Квартал | Является неуникальным идентификатором квартала. | Целое число |
| Год | Является неуникальным идентификатором года. | Целое число |
| Плановые годовые показатели | Содержит неуникальные значения, которые описывают годовые показатели. | Число с плавающей точкой |
| Плановые квартальные показатели | Содержит неуникальные значения, которые описывают квартальные показатели. | Число с плавающей точкой |
| Код подразделения | Содержит код подразделения. Является внешним ключом для связи с отношением «Подразделения». | Целое число |

Отношение для «Перечень показателей оценки подразделений», таблица 1.4, состоит из атрибутов: «Код перечня показателей подразделения», «Квартал», «Год», «Годовая оценка перечня показателей подразделения», «Квартальная оценка перечня показателей подразделения», «Код подразделения». Связано отношением «Один ко многим» с отношением, относящимся к сущностям «Подразделения».

Таблица 1.4 – **Отношение описывающие сущность «Перечень показателей оценки подразделений»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код перечня показателей подразделения | Уникальный инкрементируемый идентификатор для каждого перечня показателей. Является первичным ключом. | Целое число |
| Квартал | Является неуникальным идентификатором квартала для каждого подразделения. | Целое число |
| Год | Является неуникальным идентификатором года для каждого подразделения. | Целое число |
| Годовая оценка перечня показателей подразделения | Является неуникальным идентификатором годовой оценки определенного перечня показателей для каждого подразделения. | Число с плавающей точкой |
| Квартальная оценка перечня показателей подразделения | Является неуникальным идентификатором квартальной оценки определенного перечня показателей для каждого подразделения. | Число с плавающей точкой |
| Код подразделения | Является уникальным идентификатором подразделения, который хранит ссылку к отношению «Подразделения». | Целое число |

Отношение для «Сотрудники», таблица 1.5, состоит из атрибутов: «Код Сотрудника», «ФИО сотрудника», «Зароботная плата», «Возраст сотрудника», «Рейтинг сотрудника», «Код подразделения». Связано отношением «Один ко многим» с отношением, относящимся к сущностям «Подразделения».

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

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код сотрудника | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| ФИО сотрудника | Содержит ФИО сотрудника. | Строка |
| Заработная плата | Содержит заработную плату для сотрудников. | Число с плавающей точкой |
| Возраст | Содержит возраст для сотрудников. | Целое число |
| Рейтинг | Содержит рейтинг сотрудников. | Число с плавающей точкой |
| Код подразделения | Содержит уникальный идентификатор подразделения. Является внешним ключом для связи с отношением «Подразделения». | Целое число |

Отношение для «Фактические показатели сотрудников», таблица 1.6, состоит из атрибутов: «Код фактических показателей», «Квартал», «Год», «Фактические годовые показатели эффективности сотрудника», «Фактические квартальные показатели эффективности сотрудника», «Код сотрудника». Связано отношением «Один ко многим» с отношением, относящимся к сущности «Сотрудники».

Таблица 1.6 – **Отношение описывающие сущность «Фактические показатели сотрудников»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код фактического показателя | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| Квартал | Содержит значение квартала для каждого показателя. | Строка |
| Год | Содержит значение года для каждого показателя | Целое число |
| Фактические квартальные показатели сотрудников | Содержит значение квартального показателя для каждого сотрудника. | Число с плавающей точкой |
| Фактические годовые показатели сотрудников | Содержит значение говодового показателя для каждого сотрудника. | Число с плавающей точкой |
| Код сотрудника | Содержит уникальный идетификатор сотрудника. Является внешним ключом для связи с отношением «Сотрудники». | Целое число |

Отношение для «Плановые показатели сотрудников», таблица 1.7, состоит из атрибутов: «Код плановых показателей», «Квартал», «Год», «Плановые годовые показатели эффективности сотрудника», «Плановые квартальные показатели эффективности сотрудника», «Код сотрудника». Связано отношением «Один ко многим» с отношением, относящимся к сущности «Сотрудники».

Таблица 1.7 – **Отношение описывающие сущность «Плановые показатели сотрудников»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код планового показателя | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| Квартал | Содержит значение квартала для каждого показателя. | Целое число |
| Год | Содержит значение года для каждого показателя. | Целое число |
| Плановые квартальные показатели сотрудников | Содержит значение квартального показателя для каждого сотрудника. | Число с плавающей точкой |
| Плановые годовые показатели сотрудников | Содержит значение годового показателя для каждого сотрудника. | Число с плавающей точкой |
| Код сотрудника | Содержит уникальный идентификатор сотрудника. Является внешним ключом для связи с отношением «Сотрудники». | Целое число |

Отношение для «Перечень показателей оценки деятельности сотрудников», таблица 1.8, состоит из атрибутов: «Код показателей оценки сотрудников», «Квартал», «Год», «Годовая оценка деятельности сотрудника», «Квартальная оценка деятельности сотрудника», «Код сотрудника». Связано отношением «Один ко многим» с отношением, относящимся к сущности «Сотрудники».

Таблица 1.8 – **Отношение описывающие сущность «Перечень показателей оценки деятельности сотрудников»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код показателей оценки сотрудников | Уникальный инкрементируемый идентификатор для каждого показателя. Является первичным ключом. | Целое число |
| Квартал | Содержит значение квартала для каждого показателя. | Целое число |
| Год | Содержит значение года для каждого показателя. | Целое число |
| Годовая оценка деятельности сотрудника | Содержит неуникальное значение квартальной оценки деятельности для каждого сотрудника | Число с плавающей точкой |
| Квартальная оценка деятельности сотрудника | Содержит неуникальное значение квартальной оценки деятельности для каждого сотрудника. | Число с плавающей точкой |
| Код сотрудника | Содержит уникальный идентификатор сотрудника. Является внешним ключом для связи с отношением «Сотрудники». | Целое число |

Отношение для «Прогресса сотрудников», таблица 1.9, состоит из атрибутов: «Код прогресса сотрудника», «Наименование прогресса», «Код сотрудника». Связано отношением «Один ко многим» с отношением, относящимся к сущности «Сотрудники».

Таблица 1.8 – **Отношение описывающие сущность «Прогресс сотрудников»**

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Описание домена** | **Тип данных** |
| Код прогресса | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| Прогресс сотрудника | Содержит неуникальное значение прогресса для каждого сотрудника | Строка |
| Код сотрудника | Содержит уникальный идентификатор сотрудника. Является внешним ключом для связи с отношением «Сотрудники». | Целое число |

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

## **1.2 Физическая модель базы данных**

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

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

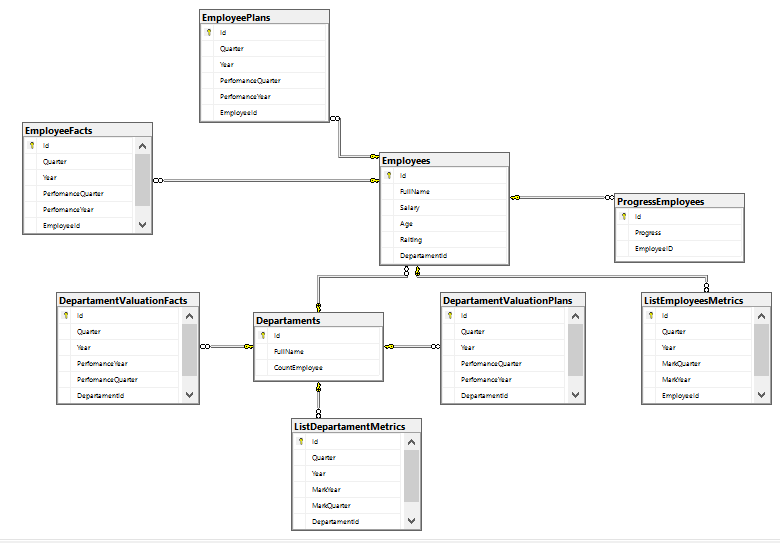


Рисунок 1.1 – Диаграмма базы данных

Для процесса 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.

## **1.3 Файловая структура базы данных**

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

Для каждого отношения были получены следующие таблицы: *Departament*, *DepartamentValuationFact*, *DepartamentValuationPlan*, *Employee*, *EmployeeFact*, *EmployeePlan*, *ProgressEmployee, ListDepartamentMetrics, ListEmployeesMetrics*.

Таблица *Departaments* находится в отношении «один» и описывают сущность «Подразделения». Были подобраны физические типы данных для соответствующих столбцов, а также установлены первичные ключи.

Таблицы *Employees*, *EmployeePlans*, *EmloyeeFacts, DepartamentValuationFacts, DepartamentValuationPlan, ProgressEmployees*, *ListDepartamentMetrics* и *ListEmployeesMetrics* находятся в отношении «многие», описывают сущности «Сотрудники», «Плановые значения показателей эффективности сотрудников», «Фактические значения показателей эффективности сотрудников», «Фактические значения показателей эффективности подразделений», «Плановые значения показателей эффективности подразделений», «Прогресс сотрудников», «Перечень показателей оценки деятельности подразделений», «Перечень показателей оценки деятельности сотрудников». 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.

# 2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМЦИОННОЙ СИСТЕМЫ

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

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

­­– установленный *MS SQL Server*;

– для работы *MS SQL Server* 2016 и выше, требуется *.NET Framework 4.6*;

– сетевое программное обеспечение;

– требуется как минимум 7 ГБ свободного места на диске (при увеличении размера базы данных, может потребоваться свободного места);

– минимальный объем оперативной памяти 1 ГБ;

– процессор *x*64 с тактовой частотой 1,4 ГГц;

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

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

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

­­ – процессор *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.

## **2.3 Требования к системному и прикладному программному обеспечению на стороне клиента**

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

## **2.4 Настройка и развёртывание приложения на сервере**

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

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

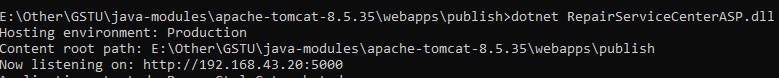


Рисунок 2.1 – Запуск веб-приложения на веб-сервере

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

**3 СТРУКТУРА ПРИЛОЖЕНИЯ**

**3.1 Описание общей структуры веб-приложения**

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

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

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

Контроллер представляет центральный компонент архитектуры *MVC* для управления взаимодействием с пользователем, работы с моделью и выбора представления для отображения. Контроллер обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки 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, EmployeeFact, EmployeePlan, ProgressEmployee, Departament*, *DepartamentValuationFact, DepartamentValuationPlan, ListDepartamentMetric* и *ListEmployeeMetrict* описывают таблицы *Employees, EmployeeFacts, EmployeePlans, ProgressEmployees, Departaments*, *DepartamentValuationFacts, DepartamentValuationPlans, ListDepartamentMetrics* и *ListEmployeeMetricts* 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;

– *DepartamentsController –* отвечает за работу с таблицей *Departaments;*

*– DepartamentValuationFactsController –* отвечает за работу с таблицей *DepartamentValuationFacts;*

*– DepartamentValuationPlansController –* отвечает за работу с таблицей *DepartamentValuationPlans;*

*– EmployeeFactsController –* отвечает за работу с таблицей *EmployeeFacts;*

*– EmployeePlansController –* отвечает за работу с таблицей *EmployeePlans;*

*– EmployeesController –* отвечает за работу с таблицей *Employees;*

*– ProgressEmployeesController–* отвечает за работу с таблицей *ProgressEmployees;*

*– ListDepartamentMetricsController–* отвечает за работу с таблицей *ListDepartamentMetrics;*

*– ListEmployeesMetricsController–* отвечает за работу с таблицей *ListEmployeesMetrics.*

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 и номер страницы. При фильтрации данных учитыается слелующий факт: если фильтры новые (т.е. они не применялись для фильтрации данных), то происходит выборка данных из базы данных, фильтрация с использованием входных значений фильтров. 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;

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

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

– *ProgressEmployee –* содержит представления для работы с таблицей «Прогресс сотрудников», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

– *Departament –* содержит представления для работы с таблицей «Подразделения», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

– *DepartamentValuationFact –* содержит представления для работы с данными из таблицы «Фактические показатели эффективности подразделений», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

*–DepartamentValuationPlan –* содержит представления для работы с данными из таблицы «Плановые показатели эффективности подразделений», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

– *ListDepartamentMetric –* содержит представления для работы с данными из таблицы «Перечень показателей оценки деятельности подразделений», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

–*ListEmployeeMetric –* содержит представления для работы с данными из таблицы «Перечень показателей оценки деятельности сотрудников», для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению;

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 и *html* страничка).

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**4.1 Введение**

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

## **4.2 Назначение, условие применения и функционал**

*Web*-приложение предназначено для оценки эффективности деятельности подразделений и сотрудников компании.

Основные функции приложения:

– добавление, просмотр и редактирования различных показателей эффективности подразделений и сотрудников;

– выборка, сортировка и фильтрация сотрудников и различных показателей эффективности подразделений и сотрудников по заданным критериям;

– добавление, просмотр и редактирования данных о сотрудниках;

– выборка рейтинга всех сотрудников заданного подразделения за указанный период (квартал, год)

– выборка подразделения за текущий период (квартал, год) и аналогичный период прошлого года;

– выборка рейтинга сотрудников за текущий период (квартал, год) и аналогичный период прошлого года.

## **4.3 Подготовка к работе**

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

## **4.4 Описание операции по обработки данных**

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

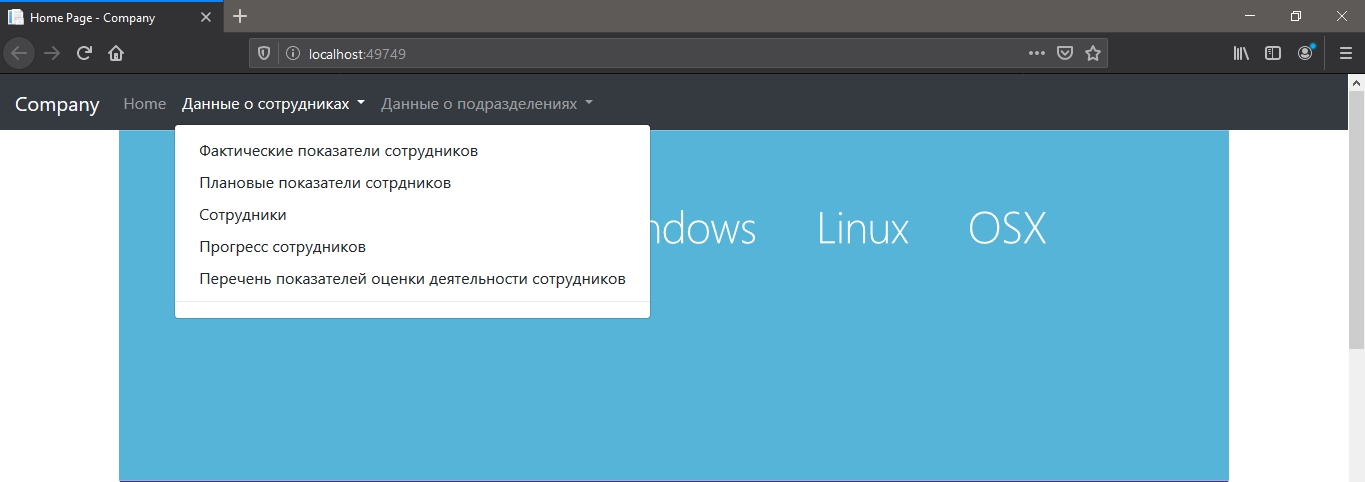


Рисунок 4.1 – Выбор таблицы, для просмотра интересующих данных

Затем можно выбрать любую из открывшихся вкладок. Например, для просмотра фактических показателей сотрудников выбираем вкладку «Фактических показателей сотрудников» и загрузится новое окно со списком фактических показателей сотрудников, рисунок 4.2.

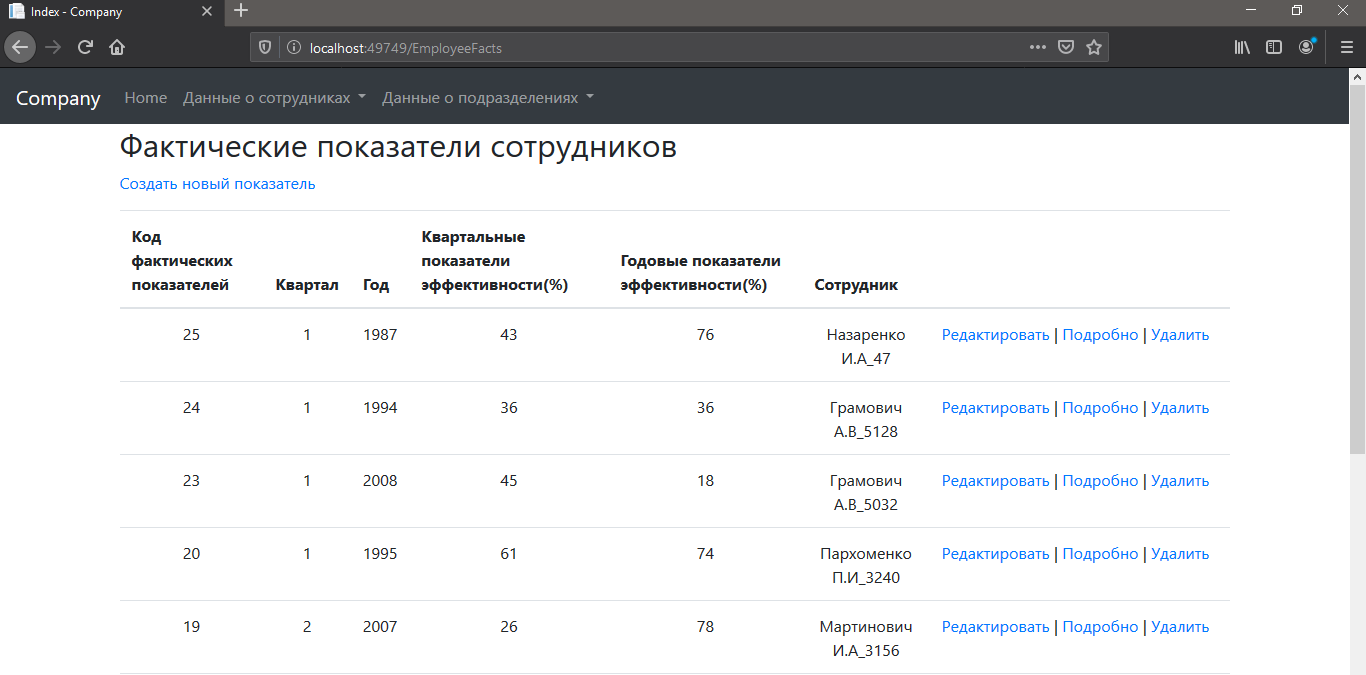


Рисунок 4.2 – Список фактических показателей сотрудников за определенный период (квартал и год)

При нажатии на название столбцов в шапке таблицы, будет произведена сортировка по возрастанию, при повторном нажатии сортироваться будет по убыванию.

Чтобы создать новый показатель нужно нажать на ссылку, выделенную голубым цветом, под названием вкладки (в данном случаи – «Создать новый показатель»), затем загрузится новая страница, рисунок 4.3 с формами для создания нового показателя (все поля формы имеют проверку на корректность введённых данных). Чтобы окончательно создать новый показатель нужно нажать на кнопку «Создать».

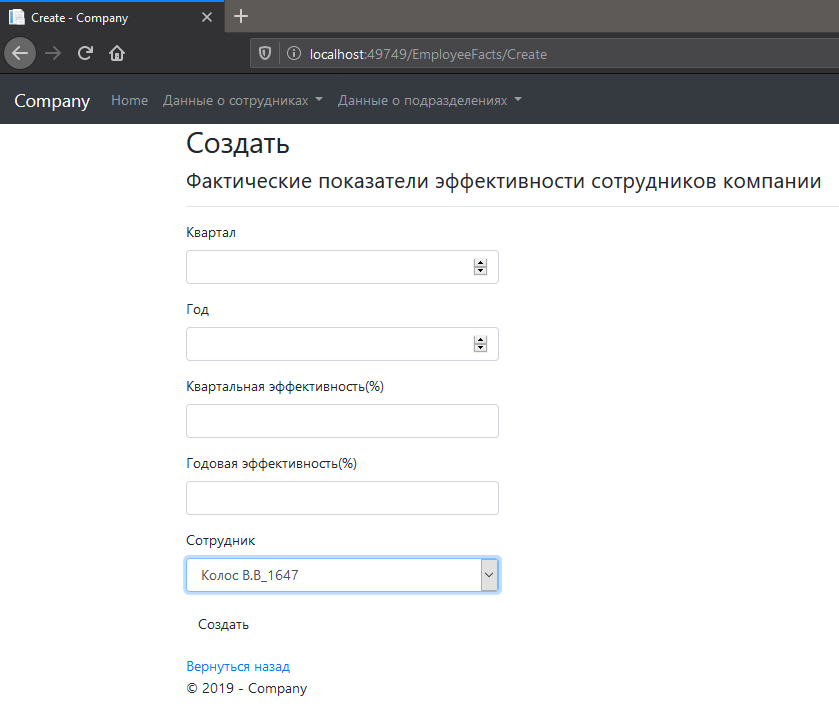


Рисунок 4.3 – Форма для добавления нового заказа

После добавления показателя пользователь будет перенаправлен на страницу с выборкой.

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

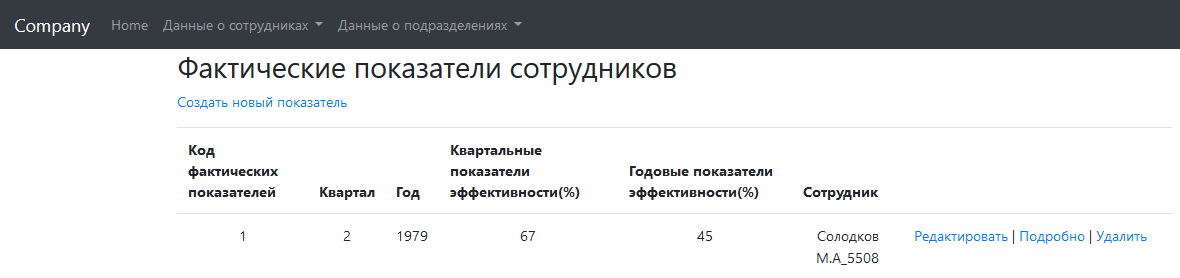


Рисунок 4.4 – Пункты для редактирования, удаления и просмотра заказов

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».

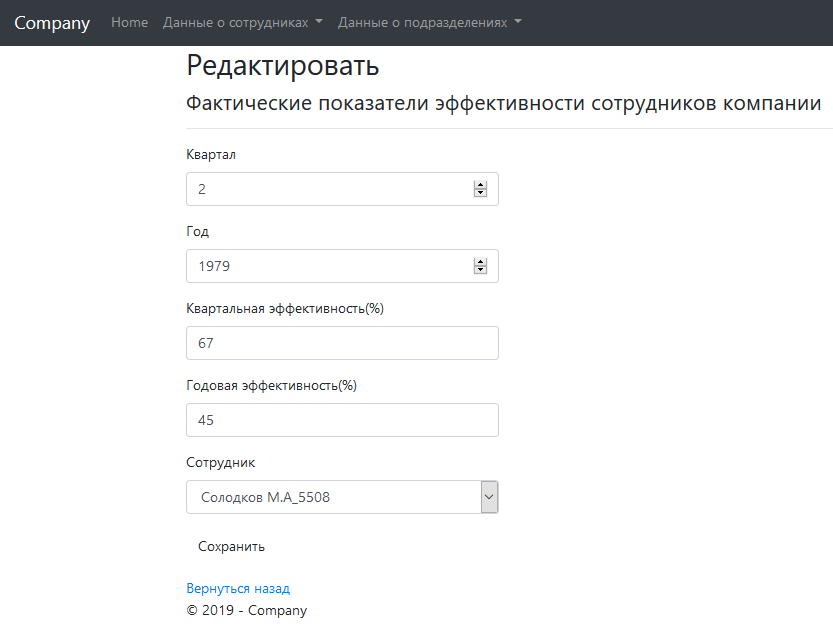


Рисунок 4.5 – Окно для редактирования данных выбранного показателя сотрудника

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

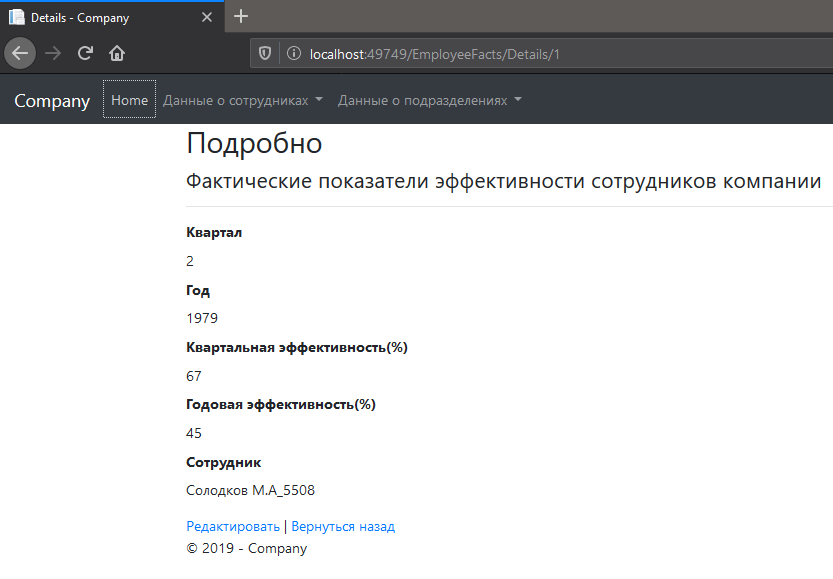


Рисунок 4.6 – Подробности показателя выбранного сотрудника

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

# 

# 5 РУКОВОДСТВО ПРОГРАММИСТА

## **5.1 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.

## **5.2 Характеристики программы**

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

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

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

## **5.3 Сопровождение программного комплекса**

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

## **5.4 Входные и выходные данные**

Входными данными для веб-приложения является:

– веб-сервер, на котором разворачивается приложение;

– сгенерированная база данных с помощью возможностей *Entity Framework*;

– тестовый набор для отладки приложения генерируемый компонентом Middleware, листинг приведён в приложении А.

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

## **5.5 Сообщения в ходе работы приложения**

При работе программа может оповещать пользователя о следующих неполадках:

– некорректно 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.

# ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта было реализовано 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 могут облегчить различным предприятиям и компаниям вести оценку эффективности деятельности своих сотрудников и подразделений.

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.

# Список используемых источников

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

# ПРИЛОЖЕНИЕ А

(обязательное)

**Код программы**

appsettings.json

{

"ConnectionStrings": {

"DefaultConnection": "Server=.\\sqlexpress;Database=CompanyASP;Trusted\_Connection=True;MultipleActiveResultSets=true",

},

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*"

}

Employee.cs

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; }

public double Salary { get; set; }

public int Age { get; set; }

public double Raiting { get; set; }

public int DepartamentId { get; set; }

public Departament Departament { get; set; }

public ICollection<EmployeeFact> EmployeeFacts { get; set; }

public ICollection<EmployeePlan> EmployeePlans { get; set; }

public ICollection<ListEmployeesMetrics> ListEmployeesMetrics { get; set; }

public ICollection<ProgressEmployee> ProgressEmployees { get; set; }

public Employee()

{

ProgressEmployees = new List<ProgressEmployee>();

EmployeeFacts = new List<EmployeeFact>();

EmployeePlans = new List<EmployeePlan>();

ListEmployeesMetrics = new List<ListEmployeesMetrics>();

}

}

}

EmployeeFact.cs

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class EmployeeFact

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public double PerfomanceQuarter { get; set; }

public double PerfomanceYear { get; set; }

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

}

}

EmployeePlan.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class EmployeePlan

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public double PerfomanceQuarter { get; set; }

public double PerfomanceYear { get; set; }

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

}

}

ProgressEmployee.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class ProgressEmployee

{

public int Id { get; set; }

public string Progress { get; set; }

public int EmployeeID { get; set; }

public Employee Employee { get; set; }

}

}

Departament.cs

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class Departament

{

/// <summary>

/// поля класса

/// </summary>

public int Id { get; set; }

public string FullName { get; set; }

public int CountEmployee { get; set; }

public ICollection<Employee> Employees { get; set; } //навигационное свойство

public ICollection<DepartamentValuationFact> DepartamentValuationFacts { get; set; }

public ICollection<DepartamentValuationPlan> DepartamentValuationPlans { get; set; }

public Departament()

{

Employees = new List<Employee>();

DepartamentValuationFacts = new List<DepartamentValuationFact>();

DepartamentValuationPlans = new List<DepartamentValuationPlan>();

}

}

}

DepartamentValuationFact.cs

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class DepartamentValuationFact

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public double PerfomanceYear { get; set; } //эффективность в процентах от 1 до 100

public double PerfomanceQuarter { get; set; } //

public int DepartamentId { get; set; }

public Departament Departament { get; set; } //навигационное свойство

}

}

DepartamentValuationPlan.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models

{

public class DepartamentValuationPlan

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public double PerfomanceQuarter { get; set; } //эффективность в процентах от 1 до 100

public double PerfomanceYear { get; set; }

public int DepartamentId { get; set; }

public Departament Departament { get; set; } //навигационное свойство

}

}

ListDepartamentMetrics.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models.Indicators

{

public class ListDepartamentMetrics

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public int MarkYear { get; set; }

public int MarkQuarter { get; set; }

public int DepartamentId { get; set; }

public Departament Departament { get; set; }

}

}

ListEmployeeMetric.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Models.Indicators

{

public class ListEmployeesMetrics

{

public int Id { get; set; }

public int Quarter { get; set; }

public int Year { get; set; }

public int MarkQuarter { get; set; }

public int MarkYear { get; set; }

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

}

}

CompanyContext.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using CompanyASP.Models;

using CompanyASP.Models.Indicators;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.SqlServer;

namespace CompanyASP.Data

{

public class CompanyContext : DbContext

{

public CompanyContext(DbContextOptions<CompanyContext> options) : base(options)

{

}

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

public DbSet<EmployeeFact> EmployeeFacts { get; set; }

public DbSet<EmployeePlan> EmployeePlans { get; set; }

public DbSet<ProgressEmployee> ProgressEmployees { get; set; }

public DbSet<Departament> Departaments { get; set; }

public DbSet<DepartamentValuationFact> DepartamentValuationFacts { get; set; }

public DbSet<DepartamentValuationPlan> DepartamentValuationPlans { get; set; }

public DbSet<ListEmployeesMetrics> ListEmployeesMetrics { get; set; }

public DbSet<ListDepartamentMetrics> ListDepartamentMetrics { get; set; }

}

}

DbInitalizer.cs

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Data

{

public class DbInitializer

{

private readonly RequestDelegate next;

private static int employeeFactsCount = 10000;

private static int employeePlansCount = 10000;

private static int employeesCount = 10000;

private static int progressEmployeeCount = 10000;

private static int departamentCount = 10000;

private static int departamentValuationFactsCount = 10000;

private static int departamentValuationPlansCount = 10000;

private static int listDepartamentMetrics = 10000;

private static int listEmployeesMetrics = 10000;

private static Random randObj;

public DbInitializer(RequestDelegate next)

{

this.next = next;

}

public static void Initialize(CompanyContext db)

{

InitializeDepartaments(db);

InitializeEmployee(db);

InitializeDepartamentValuationFacts(db);

InitializeDepartamentValuationPlans(db);

InitializeListDepartamentMetrics(db);

InitializeEmployeeFacts(db);

InitializeEmployeePlans(db);

InitializeListEmployeesMetrics(db);

InitializeProgressEmployee(db);

}

private static void InitializeDepartaments(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.Departaments.Any())

{

return;

}

else

{

string departamentName;

int countEmployee;

Random randObj = new Random(1);

string[] departament\_voc = { "администрация", "отдел кадров", "отдел маркетинга", "отдел продаж", "отдел финансов", "отдел логистики", "отдел IT", "отдел закупок", "отдел исследования и развития" };

int count\_departament\_voc = departament\_voc.GetLength(0);

for (int departamentId = 1; departamentId < departamentCount; departamentId++)

{

departamentName = departament\_voc[randObj.Next(count\_departament\_voc)] + "\_" + departamentId.ToString();

countEmployee = (int)randObj.Next(50, 150);

db.Departaments.Add(new Models.Departament { FullName = departamentName, CountEmployee = countEmployee });

}

db.SaveChanges();

}

}

private static void InitializeDepartamentValuationFacts(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.DepartamentValuationFacts.Any())

{

return;

}

else

{

int quarter;

int year;

double perfomanceQuarter; //эффективность

double perfomanceYear;

int departamentId;

randObj = new Random(1);

for (var departamentValuationFactId = 1; departamentValuationFactId < departamentValuationFactsCount; departamentValuationFactId++)

{

quarter = (int)randObj.Next(1, 3);

perfomanceQuarter = (double)randObj.Next(10, 100);

perfomanceYear = (double)randObj.Next(10, 100);

departamentId = (int)randObj.Next(1, 10000);

year = (int)randObj.Next(1970, 2020);

db.DepartamentValuationFacts.Add(new Models.DepartamentValuationFact { Quarter = quarter, Year = year, PerfomanceYear = perfomanceYear, PerfomanceQuarter = perfomanceQuarter, DepartamentId = departamentId });

}

db.SaveChanges();

}

}

private static void InitializeDepartamentValuationPlans(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.DepartamentValuationPlans.Any())

{

return;

}

else

{

int quarter;

int year;

double perfomanceQuarter; //эффективность

double perfomanceYear;

int departamentId;

randObj = new Random(1);

for (var departamentValuationPlanId = 1; departamentValuationPlanId < departamentValuationPlansCount; departamentValuationPlanId++)

{

quarter = (int)randObj.Next(1, 3);

perfomanceQuarter = (double)randObj.Next(10, 100);

perfomanceYear = (double)randObj.Next(10, 100);

departamentId = (int)randObj.Next(1, 10000);

year = (int)randObj.Next(1970, 2020);

db.DepartamentValuationPlans.Add(new Models.DepartamentValuationPlan { Quarter = quarter, Year = year, PerfomanceYear = perfomanceYear, PerfomanceQuarter = perfomanceQuarter, DepartamentId = departamentId });

}

db.SaveChanges();

}

}

private static void InitializeEmployeeFacts(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.EmployeeFacts.Any())

{

return;

}

else

{

int quarter;

int year;

double perfomanceQuarter;

double perfomanceYear;

int employeeId;

randObj = new Random(1);

for (int employeeFactId = 1; employeeFactId < employeeFactsCount; employeeFactId++)

{

quarter = (int)randObj.Next(1, 3);

year = (int)randObj.Next(1970, 2020);

perfomanceQuarter = (double)randObj.Next(10, 100);

perfomanceYear = (double)randObj.Next(10, 100);

employeeId = (int)randObj.Next(1, 10000);

db.EmployeeFacts.Add(new Models.EmployeeFact { Quarter = quarter, Year = year, PerfomanceQuarter = perfomanceQuarter, PerfomanceYear = perfomanceYear, EmployeeId = employeeId});

}

db.SaveChanges();

}

}

private static void InitializeEmployeePlans(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.EmployeePlans.Any())

{

return;

}

else

{

int quarter;

int year;

double perfomanceQuarter;

double perfomanceYear;

int employeePlanId;

randObj = new Random(1);

for (int count = 1; count < employeePlansCount; count++)

{

quarter = (int)randObj.Next(1, 3);

year = (int)randObj.Next(1970, 2020);

perfomanceQuarter = (double)randObj.Next(10, 100);

perfomanceYear = (double)randObj.Next(10, 100);

employeePlanId = (int)randObj.Next(1, 10000);

db.EmployeePlans.Add(new Models.EmployeePlan { Quarter = quarter, Year = year, PerfomanceQuarter = perfomanceQuarter, PerfomanceYear = perfomanceYear, EmployeeId = employeePlanId });

}

db.SaveChanges();

}

}

private static void InitializeEmployee(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.Employees.Any())

{

return;

}

else

{

string fullName;

double salary;

int departamentId;

int age;

double raiting;

randObj = new Random(1);

string[] employee\_voc = { "Солодков М.А", "Трофимов Е.В", "Мартинович И.А", "Назаренко И.А", "Грамович А.В", "Брусенцова Ю.В", "Пархоменко П.И", "Колос В.В" };

int count\_employee\_voc = employee\_voc.GetLength(0);

for (int count = 1; count < employeesCount; count++)

{

fullName = employee\_voc[randObj.Next(count\_employee\_voc)] + "\_" + count.ToString();

salary = (double)randObj.Next(1000, 2000);

age = (int)randObj.Next(20, 55);

departamentId = (int)randObj.Next(1, 10000);

raiting = randObj.Next(1, 10);

db.Employees.Add(new Models.Employee { FullName = fullName, Salary = salary, Age = age, DepartamentId = departamentId, Raiting = raiting });

}

db.SaveChanges();

}

}

private static void InitializeProgressEmployee(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.ProgressEmployees.Any())

{

return;

}

else

{

string fullName;

string progress;

int employeeId;

randObj = new Random(1);

string[] progress\_voc = { "выполнил работу вовремя", "просрочил дедлайн", "разлил кофе на клавиатуру", "повесился", "получил прибавку к заралпте", "повысился в должности" };

int count\_progress\_voc = progress\_voc.GetLength(0);

for (int progressEmployeeId = 1; progressEmployeeId < progressEmployeeCount; progressEmployeeId++)

{

progress = progress\_voc[randObj.Next(count\_progress\_voc)];

employeeId = (int)randObj.Next(1, 10000);

db.ProgressEmployees.Add(new Models.ProgressEmployee {Progress = progress, EmployeeID = employeeId });

}

db.SaveChanges();

}

}

private static void InitializeListEmployeesMetrics(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.ListEmployeesMetrics.Any())

{

return;

}

else

{

int quarter;

int year;

int markYear;

int markQuarter;

int employeeId;

randObj = new Random(1);

for (int count = 1; count < listEmployeesMetrics; count++)

{

quarter = (int)randObj.Next(1, 3);

year = (int)randObj.Next(1970, 2020);

markQuarter = (int)randObj.Next(1, 10);

markYear = (int)randObj.Next(1, 10);

employeeId = (int)randObj.Next(1, 10000);

db.ListEmployeesMetrics.Add(new Models.Indicators.ListEmployeesMetrics { Quarter = quarter, Year = year, MarkQuarter = markQuarter, MarkYear = markYear, EmployeeId = employeeId });

}

}

}

private static void InitializeListDepartamentMetrics(CompanyContext db)

{

db.Database.EnsureCreated();

if (db.ListEmployeesMetrics.Any())

{

return;

}

else

{

int quarter;

int year;

int markYear;

int markQuarter;

int departamentId;

randObj = new Random(1);

for (int count = 1; count < listDepartamentMetrics; count++)

{

quarter = (int)randObj.Next(1, 3);

year = (int)randObj.Next(1970, 2020);

markQuarter = (int)randObj.Next(1, 10);

markYear = (int)randObj.Next(1, 10);

departamentId = (int)randObj.Next(1, 10000);

db.ListDepartamentMetrics.Add(new Models.Indicators.ListDepartamentMetrics { Quarter = quarter, Year = year, MarkQuarter = markQuarter, MarkYear = markYear, DepartamentId = departamentId });

}

}

}

}

}

DepartamentsController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class DepartamentsController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public DepartamentsController(CompanyContext context)

{

\_context = context;

}

// GET: Departaments

public IActionResult Index(int page = 1)

{

IQueryable<Departament> companyContext = \_context.Departaments;

//разбиение на страницы

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

DepartamentViewModel departament = new DepartamentViewModel

{

Departaments = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(departament);

}

// GET: Departaments/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var departament = await \_context.Departaments

.FirstOrDefaultAsync(m => m.Id == id);

if (departament == null)

{

return NotFound();

}

return View(departament);

}

// GET: Departaments/Create

public IActionResult Create()

{

return View();

}

// POST: Departaments/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("Id,FullName,CountEmployee")] Departament departament)

{

if (ModelState.IsValid)

{

\_context.Add(departament);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(departament);

}

// GET: Departaments/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var departament = await \_context.Departaments.FindAsync(id);

if (departament == null)

{

return NotFound();

}

return View(departament);

}

// POST: Departaments/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("Id,FullName,CountEmployee")] Departament departament)

{

if (id != departament.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(departament);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DepartamentExists(departament.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(departament);

}

// GET: Departaments/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var departament = await \_context.Departaments

.FirstOrDefaultAsync(m => m.Id == id);

if (departament == null)

{

return NotFound();

}

return View(departament);

}

// POST: Departaments/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var departament = await \_context.Departaments.FindAsync(id);

\_context.Departaments.Remove(departament);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DepartamentExists(int id)

{

return \_context.Departaments.Any(e => e.Id == id);

}

}

}

DepartamentValuationFactsController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class DepartamentValuationFactsController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public DepartamentValuationFactsController(CompanyContext context)

{

\_context = context;

}

// GET: DepartamentValuationFacts

public IActionResult Index(int page = 1)

{

IQueryable<DepartamentValuationFact> companyContext = \_context.DepartamentValuationFacts

.Include(d => d.Departament);

//разбиение на страницы

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

DepartamentValuationFactViewModel departament = new DepartamentValuationFactViewModel

{

DepartamentValuationFacts = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(departament);

}

// GET: DepartamentValuationFacts/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationFact = await \_context.DepartamentValuationFacts

.Include(d => d.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (departamentValuationFact == null)

{

return NotFound();

}

return View(departamentValuationFact);

}

// GET: DepartamentValuationFacts/Create

public IActionResult Create()

{

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName");

return View();

}

// POST: DepartamentValuationFacts/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("Id,Quarter,Year,PerfomanceYear,PerfomanceQuarter,DepartamentId")] DepartamentValuationFact departamentValuationFact)

{

if (ModelState.IsValid)

{

\_context.Add(departamentValuationFact);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", departamentValuationFact.DepartamentId);

return View(departamentValuationFact);

}

// GET: DepartamentValuationFacts/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationFact = await \_context.DepartamentValuationFacts.FindAsync(id);

if (departamentValuationFact == null)

{

return NotFound();

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", departamentValuationFact.DepartamentId);

return View(departamentValuationFact);

}

// POST: DepartamentValuationFacts/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("Id,Quarter,Year,PerfomanceYear,PerfomanceQuarter,DepartamentId")] DepartamentValuationFact departamentValuationFact)

{

if (id != departamentValuationFact.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(departamentValuationFact);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DepartamentValuationFactExists(departamentValuationFact.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FulName", departamentValuationFact.DepartamentId);

return View(departamentValuationFact);

}

// GET: DepartamentValuationFacts/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationFact = await \_context.DepartamentValuationFacts

.Include(d => d.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (departamentValuationFact == null)

{

return NotFound();

}

return View(departamentValuationFact);

}

// POST: DepartamentValuationFacts/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var departamentValuationFact = await \_context.DepartamentValuationFacts.FindAsync(id);

\_context.DepartamentValuationFacts.Remove(departamentValuationFact);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DepartamentValuationFactExists(int id)

{

return \_context.DepartamentValuationFacts.Any(e => e.Id == id);

}

}

}

DepartamentValuationPlansController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class DepartamentValuationPlansController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public DepartamentValuationPlansController(CompanyContext context)

{

\_context = context;

}

// GET: DepartamentValuationPlans

public IActionResult Index(int page = 1)

{

IQueryable<DepartamentValuationPlan> companyContext = \_context.DepartamentValuationPlans

.Include(d => d.Departament);

//разбиение на страницы

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

DepartamentValuationPlanViewModel departament = new DepartamentValuationPlanViewModel

{

DepartamentValuationPlans = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(departament);

}

// GET: DepartamentValuationPlans/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationPlan = await \_context.DepartamentValuationPlans

.Include(d => d.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (departamentValuationPlan == null)

{

return NotFound();

}

return View(departamentValuationPlan);

}

// GET: DepartamentValuationPlans/Create

public IActionResult Create()

{

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName");

return View();

}

// POST: DepartamentValuationPlans/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,DepartamentId")] DepartamentValuationPlan departamentValuationPlan)

{

if (ModelState.IsValid)

{

\_context.Add(departamentValuationPlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", departamentValuationPlan.DepartamentId);

return View(departamentValuationPlan);

}

// GET: DepartamentValuationPlans/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationPlan = await \_context.DepartamentValuationPlans.FindAsync(id);

if (departamentValuationPlan == null)

{

return NotFound();

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", departamentValuationPlan.DepartamentId);

return View(departamentValuationPlan);

}

// POST: DepartamentValuationPlans/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,DepartamentId")] DepartamentValuationPlan departamentValuationPlan)

{

if (id != departamentValuationPlan.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(departamentValuationPlan);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!DepartamentValuationPlanExists(departamentValuationPlan.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", departamentValuationPlan.DepartamentId);

return View(departamentValuationPlan);

}

// GET: DepartamentValuationPlans/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var departamentValuationPlan = await \_context.DepartamentValuationPlans

.Include(d => d.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (departamentValuationPlan == null)

{

return NotFound();

}

return View(departamentValuationPlan);

}

// POST: DepartamentValuationPlans/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var departamentValuationPlan = await \_context.DepartamentValuationPlans.FindAsync(id);

\_context.DepartamentValuationPlans.Remove(departamentValuationPlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool DepartamentValuationPlanExists(int id)

{

return \_context.DepartamentValuationPlans.Any(e => e.Id == id);

}

}

}

EmployeeFactsController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class EmployeeFactsController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public EmployeeFactsController(CompanyContext context)

{

\_context = context;

}

// GET: EmployeeFacts

public IActionResult Index(int page = 1)

{

IQueryable<EmployeeFact> companyContext = \_context.EmployeeFacts

.Include(d => d.Employee);

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

EmployeeFactViewModel employee = new EmployeeFactViewModel

{

EmployeeFacts = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(employee);

}

// GET: EmployeeFacts/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var employeeFact = await \_context.EmployeeFacts

.Include(e => e.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (employeeFact == null)

{

return NotFound();

}

return View(employeeFact);

}

// GET: EmployeeFacts/Create

public IActionResult Create()

{

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName");

return View();

}

// POST: EmployeeFacts/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,EmployeeId")] EmployeeFact employeeFact)

{

if (ModelState.IsValid)

{

\_context.Add(employeeFact);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeeFact.EmployeeId);

return View(employeeFact);

}

// GET: EmployeeFacts/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var employeeFact = await \_context.EmployeeFacts.FindAsync(id);

if (employeeFact == null)

{

return NotFound();

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeeFact.EmployeeId);

return View(employeeFact);

}

// POST: EmployeeFacts/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,EmployeeId")] EmployeeFact employeeFact)

{

if (id != employeeFact.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employeeFact);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeFactExists(employeeFact.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeeFact.EmployeeId);

return View(employeeFact);

}

// GET: EmployeeFacts/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var employeeFact = await \_context.EmployeeFacts

.Include(e => e.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (employeeFact == null)

{

return NotFound();

}

return View(employeeFact);

}

// POST: EmployeeFacts/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var employeeFact = await \_context.EmployeeFacts.FindAsync(id);

\_context.EmployeeFacts.Remove(employeeFact);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EmployeeFactExists(int id)

{

return \_context.EmployeeFacts.Any(e => e.Id == id);

}

}

}

EmployeePlansController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class EmployeePlansController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public EmployeePlansController(CompanyContext context)

{

\_context = context;

}

// GET: EmployeePlans

public IActionResult Index(int page = 1)

{

IQueryable<EmployeePlan> companyContext = \_context.EmployeePlans

.Include(d => d.Employee);

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

EmployeePlanViewModel employee = new EmployeePlanViewModel

{

EmployeePlans = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(employee);

}

// GET: EmployeePlans/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var employeePlan = await \_context.EmployeePlans

.Include(e => e.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (employeePlan == null)

{

return NotFound();

}

return View(employeePlan);

}

// GET: EmployeePlans/Create

public IActionResult Create()

{

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName");

return View();

}

// POST: EmployeePlans/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,EmployeeId")] EmployeePlan employeePlan)

{

if (ModelState.IsValid)

{

\_context.Add(employeePlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeePlan.EmployeeId);

return View(employeePlan);

}

// GET: EmployeePlans/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var employeePlan = await \_context.EmployeePlans.FindAsync(id);

if (employeePlan == null)

{

return NotFound();

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeePlan.EmployeeId);

return View(employeePlan);

}

// POST: EmployeePlans/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("Id,Quarter,Year,PerfomanceQuarter,PerfomanceYear,EmployeeId")] EmployeePlan employeePlan)

{

if (id != employeePlan.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employeePlan);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeePlanExists(employeePlan.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", employeePlan.EmployeeId);

return View(employeePlan);

}

// GET: EmployeePlans/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var employeePlan = await \_context.EmployeePlans

.Include(e => e.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (employeePlan == null)

{

return NotFound();

}

return View(employeePlan);

}

// POST: EmployeePlans/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var employeePlan = await \_context.EmployeePlans.FindAsync(id);

\_context.EmployeePlans.Remove(employeePlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EmployeePlanExists(int id)

{

return \_context.EmployeePlans.Any(e => e.Id == id);

}

}

}

EmployeesController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

using CompanyASP.ViewModel.SortedEmployee;

namespace CompanyASP.Controllers

{

public class EmployeesController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public EmployeesController(CompanyContext context)

{

\_context = context;

}

// GET: Employees

public async Task<IActionResult> Index(string fullName, int? age, double? salary, double? raiting, int? employee,

string nameDepartament,int page = 1, SortStateEmployee employeeSort = SortStateEmployee.FullNameAsc)

{

//фильтрация

IQueryable<Employee> companyContext = \_context.Employees

.Include(d => d.Departament);

if (!string.IsNullOrEmpty(fullName))

{

companyContext = companyContext.Where(d => d.FullName.Contains(fullName));

}

if (age != null && age.Value != 0)

{

companyContext = companyContext.Where(d => d.Age == age);

}

if (salary != null && salary.Value != 0)

{

companyContext = companyContext.Where(d => d.Salary == salary);

}

if (raiting != null && raiting.Value != 0)

{

companyContext = companyContext.Where(d => d.Raiting == raiting);

}

if (!string.IsNullOrEmpty(nameDepartament))

{

companyContext = companyContext.Where(d => d.Departament.FullName.Contains(nameDepartament));

}

//сортировка

switch (employeeSort)

{

case SortStateEmployee.AgeAsc:

companyContext = companyContext.OrderBy(d => d.Age);

break;

case SortStateEmployee.AgeDesc:

companyContext = companyContext.OrderByDescending(d => d.Age);

break;

case SortStateEmployee.FullNameAsc:

companyContext = companyContext.OrderBy(d => d.FullName);

break;

case SortStateEmployee.FullNameDesc:

companyContext = companyContext.OrderByDescending(d => d.FullName);

break;

case SortStateEmployee.SalaryAsc:

companyContext = companyContext.OrderBy(d => d.Salary);

break;

case SortStateEmployee.SalaryDesc:

companyContext = companyContext.OrderByDescending(d => d.Salary);

break;

case SortStateEmployee.RaitingAsc:

companyContext = companyContext.OrderBy(d => d.Raiting);

break;

case SortStateEmployee.RaitingDesc:

companyContext = companyContext.OrderByDescending(d => d.Raiting);

break;

case SortStateEmployee.DepartamentAsc:

companyContext = companyContext.OrderBy(d => d.Departament.FullName);

break;

case SortStateEmployee.DepartamentDesc:

companyContext = companyContext.OrderByDescending(d => d.Departament.FullName);

break;

default:

companyContext = companyContext.OrderBy(d => d.FullName);

break;

}

//разбиение на страницы

var count = await companyContext.CountAsync();

var companys = await companyContext.Skip((page - 1) \* pageSize).Take(pageSize).ToListAsync();

//формирование представления

IndexEmployeeViewModel employeesView = new IndexEmployeeViewModel

{

Employees = companys,

PageViewModel = new PageViewModel(count, page, pageSize),

SortEmployeeViewModel = new SortEmployeeViewModel(employeeSort),

FilterEmployeeViewModel = new FilterEmployeeViewModel(companyContext.ToList(), fullName, age, salary, raiting, employee)

};

return View(employeesView);

}

// GET: Employees/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees

.Include(e => e.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// GET: Employees/Create

public IActionResult Create()

{

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName");

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("Id,FullName,Salary,Age,Raiting,DepartamentId")] Employee employee)

{

if (ModelState.IsValid)

{

\_context.Add(employee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", employee.DepartamentId);

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["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", employee.DepartamentId);

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("Id,FullName,Salary,Age,Raiting,DepartamentId")] Employee employee)

{

if (id != employee.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employee);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeExists(employee.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", employee.DepartamentId);

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.Departament)

.FirstOrDefaultAsync(m => m.Id == 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.Id == id);

}

}

}

HomeController.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Company.Models;

namespace Company.Controllers

{

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

public IActionResult About()

{

ViewData["Message"] = "Your application description page.";

return View();

}

public IActionResult Contact()

{

ViewData["Message"] = "Your contact page.";

return View();

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

ListDepartamentMetricsController.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 CompanyASP.Data;

using CompanyASP.Models.Indicators;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class ListDepartamentMetricsController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public ListDepartamentMetricsController(CompanyContext context)

{

\_context = context;

}

// GET: ListDepartamentMetrics

public IActionResult Index(int page = 1)

{

IQueryable<ListDepartamentMetrics> companyContext = \_context.ListDepartamentMetrics

.Include(d => d.Departament);

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

ListDepartamentMetricsViewModel listDepartament = new ListDepartamentMetricsViewModel

{

ListDepartamentMetrics = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(listDepartament);

}

// GET: ListDepartamentMetrics/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var listDepartamentMetrics = await \_context.ListDepartamentMetrics

.Include(l => l.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (listDepartamentMetrics == null)

{

return NotFound();

}

return View(listDepartamentMetrics);

}

// GET: ListDepartamentMetrics/Create

public IActionResult Create()

{

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName");

return View();

}

// POST: ListDepartamentMetrics/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("Id,Quarter,Year,MarkYear,MarkQuarter,DepartamentId")] ListDepartamentMetrics listDepartamentMetrics)

{

if (ModelState.IsValid)

{

\_context.Add(listDepartamentMetrics);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "fullName", listDepartamentMetrics.DepartamentId);

return View(listDepartamentMetrics);

}

// GET: ListDepartamentMetrics/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var listDepartamentMetrics = await \_context.ListDepartamentMetrics.FindAsync(id);

if (listDepartamentMetrics == null)

{

return NotFound();

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", listDepartamentMetrics.DepartamentId);

return View(listDepartamentMetrics);

}

// POST: ListDepartamentMetrics/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("Id,Quarter,Year,MarkYear,MarkQuarter,DepartamentId")] ListDepartamentMetrics listDepartamentMetrics)

{

if (id != listDepartamentMetrics.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(listDepartamentMetrics);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ListDepartamentMetricsExists(listDepartamentMetrics.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["DepartamentId"] = new SelectList(\_context.Departaments, "Id", "FullName", listDepartamentMetrics.DepartamentId);

return View(listDepartamentMetrics);

}

// GET: ListDepartamentMetrics/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var listDepartamentMetrics = await \_context.ListDepartamentMetrics

.Include(l => l.Departament)

.FirstOrDefaultAsync(m => m.Id == id);

if (listDepartamentMetrics == null)

{

return NotFound();

}

return View(listDepartamentMetrics);

}

// POST: ListDepartamentMetrics/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var listDepartamentMetrics = await \_context.ListDepartamentMetrics.FindAsync(id);

\_context.ListDepartamentMetrics.Remove(listDepartamentMetrics);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ListDepartamentMetricsExists(int id)

{

return \_context.ListDepartamentMetrics.Any(e => e.Id == id);

}

}

}

ListEmployeesMetricsController.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 CompanyASP.Data;

using CompanyASP.Models.Indicators;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class ListEmployeesMetricsController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public ListEmployeesMetricsController(CompanyContext context)

{

\_context = context;

}

// GET: ListEmployeesMetrics

public IActionResult Index(int page = 1)

{

IQueryable<ListEmployeesMetrics> companyContext = \_context.ListEmployeesMetrics

.Include(d => d.Employee);

//разбиение на страницы

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

ListEmployeesMetrcisViewModel listEmployees = new ListEmployeesMetrcisViewModel

{

ListEmployeesMetrics = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(listEmployees);

}

// GET: ListEmployeesMetrics/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var listEmployeesMetrics = await \_context.ListEmployeesMetrics

.Include(l => l.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (listEmployeesMetrics == null)

{

return NotFound();

}

return View(listEmployeesMetrics);

}

// GET: ListEmployeesMetrics/Create

public IActionResult Create()

{

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName");

return View();

}

// POST: ListEmployeesMetrics/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("Id,Quarter,Year,MarkQuarter,MarkYear,EmployeeId")] ListEmployeesMetrics listEmployeesMetrics)

{

if (ModelState.IsValid)

{

\_context.Add(listEmployeesMetrics);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", listEmployeesMetrics.EmployeeId);

return View(listEmployeesMetrics);

}

// GET: ListEmployeesMetrics/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var listEmployeesMetrics = await \_context.ListEmployeesMetrics.FindAsync(id);

if (listEmployeesMetrics == null)

{

return NotFound();

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", listEmployeesMetrics.EmployeeId);

return View(listEmployeesMetrics);

}

// POST: ListEmployeesMetrics/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("Id,Quarter,Year,MarkQuarter,MarkYear,EmployeeId")] ListEmployeesMetrics listEmployeesMetrics)

{

if (id != listEmployeesMetrics.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(listEmployeesMetrics);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ListEmployeesMetricsExists(listEmployeesMetrics.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName", listEmployeesMetrics.EmployeeId);

return View(listEmployeesMetrics);

}

// GET: ListEmployeesMetrics/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var listEmployeesMetrics = await \_context.ListEmployeesMetrics

.Include(l => l.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (listEmployeesMetrics == null)

{

return NotFound();

}

return View(listEmployeesMetrics);

}

// POST: ListEmployeesMetrics/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var listEmployeesMetrics = await \_context.ListEmployeesMetrics.FindAsync(id);

\_context.ListEmployeesMetrics.Remove(listEmployeesMetrics);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ListEmployeesMetricsExists(int id)

{

return \_context.ListEmployeesMetrics.Any(e => e.Id == id);

}

}

}

ProgressEmployeesController.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 CompanyASP.Data;

using CompanyASP.Models;

using CompanyASP.ViewModel;

namespace CompanyASP.Controllers

{

public class ProgressEmployeesController : Controller

{

private readonly CompanyContext \_context;

private int pageSize = 10;

public ProgressEmployeesController(CompanyContext context)

{

\_context = context;

}

// GET: ProgressEmployees

public IActionResult Index(int page = 1)

{

IQueryable<ProgressEmployee> companyContext = \_context.ProgressEmployees

.Include(d => d.Employee);

//разбиение на страницы

var count = companyContext.Count();

companyContext = companyContext.Skip((page - 1) \* pageSize).Take(pageSize);

//формирование представления

ProgressEmployeeViewModel progress = new ProgressEmployeeViewModel

{

ProgressEmployees = companyContext,

PageViewModel = new PageViewModel(count, page, pageSize),

};

return View(progress);

}

// GET: ProgressEmployees/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var progressEmployee = await \_context.ProgressEmployees

.Include(p => p.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (progressEmployee == null)

{

return NotFound();

}

return View(progressEmployee);

}

// GET: ProgressEmployees/Create

public IActionResult Create()

{

ViewData["EmployeeID"] = new SelectList(\_context.Employees, "Id", "FullName");

return View();

}

// POST: ProgressEmployees/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("Id,Progress,EmployeeID")] ProgressEmployee progressEmployee)

{

if (ModelState.IsValid)

{

\_context.Add(progressEmployee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeID"] = new SelectList(\_context.Employees, "Id", "FullName", progressEmployee.EmployeeID);

return View(progressEmployee);

}

// GET: ProgressEmployees/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var progressEmployee = await \_context.ProgressEmployees.FindAsync(id);

if (progressEmployee == null)

{

return NotFound();

}

ViewData["EmployeeID"] = new SelectList(\_context.Employees, "Id", "FullName", progressEmployee.EmployeeID);

return View(progressEmployee);

}

// POST: ProgressEmployees/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("Id,Progress,EmployeeID")] ProgressEmployee progressEmployee)

{

if (id != progressEmployee.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(progressEmployee);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ProgressEmployeeExists(progressEmployee.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["EmployeeID"] = new SelectList(\_context.Employees, "Id", "FullName", progressEmployee.EmployeeID);

return View(progressEmployee);

}

// GET: ProgressEmployees/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var progressEmployee = await \_context.ProgressEmployees

.Include(p => p.Employee)

.FirstOrDefaultAsync(m => m.Id == id);

if (progressEmployee == null)

{

return NotFound();

}

return View(progressEmployee);

}

// POST: ProgressEmployees/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var progressEmployee = await \_context.ProgressEmployees.FindAsync(id);

\_context.ProgressEmployees.Remove(progressEmployee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ProgressEmployeeExists(int id)

{

return \_context.ProgressEmployees.Any(e => e.Id == id);

}

}

}

DbInitializerExtensions.cs

using Microsoft.AspNetCore.Builder;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Middleware

{

public static class DbInitializerExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

DbInitializerMiddleware.cs

using CompanyASP.Data;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate next;

public DbInitializerMiddleware(RequestDelegate next)

{

this.next = next;

}

public Task Invoke(HttpContext context, IServiceProvider serviceProvider, CompanyContext dbContext)

{

if (!(context.Session.Keys.Contains("starting")))

{

DbInitializer.Initialize(dbContext);

context.Session.SetString("starting", "Yes");

}

return this.next.Invoke(context);

}

}

}

Sturtup.cs

using CompanyASP.Data;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using CompanyASP.Middleware;

namespace Company

{

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<CompanyContext>(options => options.UseSqlServer(connection));

services.AddDistributedMemoryCache();

services.AddSession();

services.AddResponseCaching();

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();

}

else

{

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

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthentication();

app.UseSession();

app.UseResponseCaching();

app.UseCookiePolicy();

app.UseDbInitializer();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Logging;

namespace Company

{

public class Program

{

public static void Main(string[] args)

{

CreateWebHostBuilder(args).Build().Run();

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>();

}

}

PageViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class PageViewModel

{

public int PageNumber { get; private set; }

public int TotalPages { get; private set; }

public PageViewModel(int count, int pageNumber = 1, int pageSize = 20)

{

PageNumber = pageNumber;

TotalPages = (int)Math.Ceiling(count / (double)pageSize);

}

public bool HasPreviousPage

{

get

{

return (PageNumber > 1);

}

}

public bool HasNextPage

{

get

{

return (PageNumber < TotalPages);

}

}

}

}

DepartamentValuationFactViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class DepartamentValuationFactViewModel

{

public IEnumerable<DepartamentValuationFact> DepartamentValuationFacts { get; set; }

[Display(Name = "Код фактических показателей")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Годовые показатели(%)")]

public double PerfomanceYear { get; set; }

[Display(Name = "Квартальные показатели(%)")]

public double PerfomanceQuarter { get; set; }

[Display(Name = "Отдел")]

public int DepartamentId { get; set; }

public Departament Departament { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

DepartamentValuationPlanViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class DepartamentValuationPlanViewModel

{

public IEnumerable<DepartamentValuationPlan> DepartamentValuationPlans { get; set; }

[Display(Name = "Код фактических показателей")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Годовые показатели(%)")]

public double PerfomanceYear { get; set; }

[Display(Name = "Квартальные показатели(%)")]

public double PerfomanceQuarter { get; set; }

[Display(Name = "Отдел")]

public int DepartamentId { get; set; }

public Departament Departament { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

DepartamentViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class DepartamentViewModel

{

public IEnumerable<Departament> Departaments { get; set; }

[Display(Name = "Код отдела")]

public int Id { get; set; }

[Display(Name = "Наименование отдела")]

public string FullName { get; set; }

[Display(Name = "Кол-во сотрудников")]

public int CountEmployee { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

EmployeeFactViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class EmployeeFactViewModel

{

public IEnumerable<EmployeeFact> EmployeeFacts { get; set; }

[Display(Name = "Код фактических показателей")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Квартальные показатели эффективности(%)")]

public double PerfomanceQuarter { get; set; }

[Display(Name = "Годовые показатели эффективности(%)")]

public double PerfomanceYear { get; set; }

[Display(Name = "Сотрудник")]

public double EmployeeId { get; set; }

public Employee Employee { get; set; }

public PageViewModel PageViewModel { get; set; } //навигационное свойство для страничной навигации

}

}

EmployeePlanViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class EmployeePlanViewModel

{

public IEnumerable<EmployeePlan> EmployeePlans { get; set; }

[Display(Name = "Код фактических показателей")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Квартальные показатели эффективности(%)")]

public double PerfomanceQuarter { get; set; }

[Display(Name = "Годовые показатели эффективности(%)")]

public double PerfomanceYear { get; set; }

[Display(Name = "Сотрудник")]

public double EmployeeId { get; set; }

public Employee Employee { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

EmployeeViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class EmployeeViewModel

{

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

[Display(Name = "Код сотрудника")]

public int Id { get; set; }

[Display(Name = "ФИО")]

public string FullName { get; set; }

[Display(Name = "Зарплата")]

public double Salary { get; set; }

[Display(Name = "Возраст")]

public int Age { get; set; }

[Display(Name ="Отдел")]

public int DepartamentId { get; set; }

[Display(Name ="Рейтинг")]

public int Raiting { get; set; }

public Departament Departament { get; set; }

}

}

ListDepartamentMetricsViewModel.cs

using CompanyASP.Models;

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class ListDepartamentMetricsViewModel

{

public IEnumerable<ListDepartamentMetrics> ListDepartamentMetrics { get; set; }

[Display(Name = "Код")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Годовая оценка деятельности")]

public int MarkYear { get; set; }

[Display(Name = "Квартальная оценка деятельности")]

public int MarkQuarter { get; set; }

[Display(Name = "Отдел")]

public int DepartamentId { get; set; }

public Departament Departament { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

ListEmployeesMetrcisViewModel.cs

using CompanyASP.Models;

using CompanyASP.Models.Indicators;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class ListEmployeesMetrcisViewModel

{

public IEnumerable<ListEmployeesMetrics> ListEmployeesMetrics { get; set; }

[Display(Name = "Код")]

public int Id { get; set; }

[Display(Name = "Квартал")]

public int Quarter { get; set; }

[Display(Name = "Год")]

public int Year { get; set; }

[Display(Name = "Годовая оценка деятельности")]

public int MarkYear { get; set; }

[Display(Name = "Квартальная оценка деятельности")]

public int MarkQuarter { get; set; }

[Display(Name = "Сотрудник")]

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

ProgressEmployeeViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel

{

public class ProgressEmployeeViewModel

{

public IEnumerable<ProgressEmployee> ProgressEmployees { get; set; }

[Display(Name = "Код прогресса")]

public int Id { get; set; }

[Display(Name = "Прогресс")]

public string Progress { get; set; }

[Display(Name = "Сотрудник")]

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

FilterEmployeeViewModel.cs

using CompanyASP.Models;

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel.SortedEmployee

{

public class FilterEmployeeViewModel

{

public SelectList Employees { get; private set; }

public string SelectFullName { get; private set; }

public int? SelectAge { get; private set; }

public double? SelectSalary { get; private set; }

public double? SelectRaiting { get; private set; }

public Departament Departament { get; private set; }

public FilterEmployeeViewModel(List<Employee> employees, string fullName, int? age, double? salary,

double? raiting, int? employee)

{

employees.Insert(0, new Employee() { FullName = "Все", Id = 0 });

Employees = new SelectList(employees, "Id", "FullName", employee);

SelectFullName = fullName;

SelectAge = age;

SelectSalary = salary;

SelectRaiting = raiting;

}

}

}

IndexEmployeeViewModel.cs

using CompanyASP.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel.SortedEmployee

{

public class IndexEmployeeViewModel

{

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

public PageViewModel PageViewModel { get; set; }

public SortEmployeeViewModel SortEmployeeViewModel { get; set; }

public FilterEmployeeViewModel FilterEmployeeViewModel { get; set; }

}

}

SortEmployeeViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CompanyASP.ViewModel.SortedEmployee

{

public enum SortStateEmployee

{

FullNameAsc,

FullNameDesc,

SalaryAsc,

SalaryDesc,

AgeAsc,

AgeDesc,

RaitingAsc,

RaitingDesc,

DepartamentAsc,

DepartamentDesc

}

public class SortEmployeeViewModel

{

public SortStateEmployee FullNameSort { get; set; }

public SortStateEmployee SalarySort { get; set; }

public SortStateEmployee AgeSort { get; set; }

public SortStateEmployee RaitingSort { get; set; }

public SortStateEmployee DepartamentSort { get; set; }

public SortStateEmployee CurrentState { get; set; }

public SortEmployeeViewModel(SortStateEmployee SortStateEmployee)

{

FullNameSort = SortStateEmployee == SortStateEmployee.FullNameAsc ? SortStateEmployee.FullNameDesc : SortStateEmployee.FullNameAsc;

SalarySort = SortStateEmployee == SortStateEmployee.SalaryAsc ? SortStateEmployee.SalaryDesc : SortStateEmployee.SalaryAsc;

AgeSort = SortStateEmployee == SortStateEmployee.AgeAsc ? SortStateEmployee.AgeDesc : SortStateEmployee.AgeAsc;

RaitingSort = SortStateEmployee == SortStateEmployee.RaitingAsc ? SortStateEmployee.RaitingDesc : SortStateEmployee.RaitingAsc;

DepartamentSort = SortStateEmployee == SortStateEmployee.DepartamentAsc ? SortStateEmployee.DepartamentDesc : SortStateEmployee.DepartamentAsc;

CurrentState = SortStateEmployee;

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Чертёж структуры *web*-приложения**