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: «*WEB*-ПРИЛОЖЕНИЕ БАЗ ДАННЫХ «СТРОИТЕЛЬНАЯ КОМПАНИЯ»

Исполнитель: студент гр. ЗИТ-31
Нелегач Я.А
Руководитель: доцент
Асенчик О.Д.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____
Оценка работы: _____

Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2021

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

Кафедра _____

РЭЦЭНЗІЯ
на курсавы праект (работу)

па дысцыпліне _____ ,

выканана студэнтам _____

групы _____

I. Пералік заўваг па тэксту курсавога праекта (работы)

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

II. Агульная характарыстыка работы

_____
_____
_____
_____
_____
_____
_____
_____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный курсовой проект посвящён разработке *web*-приложения баз данных строительной компании, создание интерфейса в виде набора *web*-страниц, обеспечивающих отображение и редактирование информации из базы данных, для автоматизации работы со структурированной информацией конторы.

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

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

Строительные компании в большинстве своем представляют собой коммерческие организации по оказанию различных услуг по строительству и проектированию различной сложности объектов. Строительные компании нацелены в первую очередь на получение прибыли от продажи своих услуг, поэтому наличие веб-приложения, позволяющего оптимизировать и отладить процесс оказания услуг и существенно снизить время и денежные ресурсы.

Для курсового проекта было создано *web*-приложение, так как оно дает пользователям возможность вводить, получать и манипулировать данными с помощью взаимодействия. Данное взаимодействие будет характеризоваться возможностью получения данных о заключаемых контрактов, получения актуальной информации об персонале и клиентах.

Для реализации поставленной задачи необходимо определить список технологий и программных средств, позволяющих автоматизировать весь процесс.

Программа будет состоять из источника данных, представляющего из себя базу данных, и *web*-приложения, работающего с конкретной базой данных.

В качестве источника данных предпочтительно использовать СУБД (систему управления базами данных). Среди всех выгодно выделяется *MS SQL Server*. Ее главными преимуществами являются производительность, надежность (можно шифровать данные) и простота. Также эта СУБД разработано компанией *Microsoft*, что говорит о раскрытии высокого потенциала при работе с платформой *.NET Framework, .NET Core* и *Visual Studio* в частности.

Для создания *web*-приложения используется технология *ASP.NET Core*, разработанная компанией *Microsoft* для всех основных операционных систем. Программная модель *ASP.NET* основывается на протоколе *HTTP* и использует его правила взаимодействия между сервером и браузером. Поскольку *ASP.NET Core* основывается на *Common Language Runtime (CLR)*, разработчики могут писать код для *ASP.NET Core*, используя языки программирования, входящие в комплект *.NET* (*C#, Visual Basic.NET, J#* и *JScript .NET*). В курсовом проекте будет использоваться язык *C#* и среда программирования *Visual Studio*.

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

Логическая модель информационной системы отражает логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения. Другими словами, логическая модель отображает логические связи между информационными данными в данной концептуальной модели.

Составление логической модели включает в себя:

– разработку требований к информационной системе,

– предварительное проектирование системы.

Описание требований к системе задается в виде модели и описания системных прецедентов, а предварительное проектирование осуществляется с использованием диаграммы классов с помощью языка моделирования *UML*.

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

– «Сотрудник»;

– «Должность»;

– «Вид работ»;

– «Материал»;

– «Бригада»;

– «Заказ»;

– «Заказчик».

Для сущности «Сотрудник» было создано отношение (таблица) с атрибутами: «Идентификатор», «ФИО», «Возраст», «Пол», «Адрес», «Телефон», «Паспортные данные». Данная сущность находится в ненормализованном виде и имеет связь один ко многим с отношениями «Должность» и «Бригада». Данная сущность находится на стороне связи «один», поэтому после приведения данной сущности к нормализованному виду добавляется следующие атрибуты, «Код Должности» и «Код Бригады», которые связывает данную сущность с сущностями «Должность» и «Бригада» соответственно связью один ко многим. Подробное описание отношения и атрибутов приведено в таблице 1.1.

Таблица 1.1 – Отношение описывающие сущность «Сотрудники»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом.	Целое число
ФИО	Содержит полную ФИО сотрудника строительной компании	Строка

Продолжение таблицы 1.1.

Атрибуты	Описание домена	Тип данных
Возраст	Год рождения сотрудника строительной компании	Целое число
Пол	Полоролевая идентификация сотрудника строительной компании	Строка
Адрес	Содержит адрес проживания сотрудника строительной компании	Строка
Телефон	Содержит мобильный телефон сотрудника строительной компании	Строка
Паспортные данные	Содержит серию и номер паспорта сотрудника строительной компании	Строка
Код должности	Содержит ссылку на идентификатор должности. Является внешним ключом для связи с отношением «Должность».	Целое число
Код бригады	Содержит ссылку на идентификатор должности. Является внешним ключом для связи с отношением «Бригада».	Целое число

Отношение для сущности «Должности», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «Наименование», «Заработная плата», «Обязанности», «Требования». Так как по условию задачи данная сущность находится на стороне отношений «Многие» по отношению к сущности «Сотрудники», то она должна иметь список сотрудников. Тем самым будет организованна связь один ко многим, между данной сущностью и сущностью «Сотрудники».

Таблица 1.2 – Отношение описывающие сущность «Должности»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждой должности. Является первичным ключом.	Целое число
Наименование	Содержит наименование должности.	Строка

Продолжение таблицы 1.2

Атрибуты	Описание домена	Тип данных
Заработная плата	Содержит данные о заработной плате сотрудника рекламного агенства	Вещественное число
Обязанности	Содержит информацию об обязанностях сотрудника рекламного агенства	Строка
Требования	Содержит информацию об необходимых требованиях для принятия на данную должность	Строка

Отношение для «Материал» состоит из атрибутов: «Идентификатор», «Наименование», «Упаковка», «Описание», «Стоимость». Данная сущность находится в ненормализованном виде и имеет связь один ко многим с отношением «Вид работ». Данная сущность находится на стороне связи «один», поэтому после приведения данной сущности к нормализованному виду добавляется следующий атрибут, «Код вида работ», который связывает данную сущность с сущностью «Вид работ» связью один ко многим. Подробное описание отношения и атрибутов приведено в таблице 1.3.

Таблица 1.3 – Отношение описывающие сущность «Материал»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого материала. Является первичным ключом.	Целое число
Наименование	Содержит наименование материала.	Строка
Упаковка	Содержит данные об упаковочном материале	Строка
Описание	Содержит информацию о материале	Строка

Продолжение таблицы 1.3

Атрибуты	Описание домена	Тип данных
Стоимость	Содержит информацию о стоимости материала	Вещественное число

Для сущности «Вид работ» было создано отношение (таблица) с атрибутами: «Идентификатор работ», «Наименование», «Описание», «Стоимость». Данная сущность находится в нормализованном виде и имеет связь один ко многим с отношением «Виды рекламы. Кроме того, данная сущность находится на стороне отношений «многие» с сущностями «Материал» и, «Заказ» поэтому она должна содержать список заказов и список материалов. Подробное описание отношения и атрибутов приведено в таблице 1.4. Данное отношение находится в первой нормальной форме.

Таблица 1.4 – Отношение описывающие сущность «Вид работ»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого места расположения. Является первичным ключом.	Целое число
Наименование	Содержит данные о названии вида работы	Строка
Описание	Содержит информацию, описывающую вид работы	Строка
Стоимость	Содержит данные, которые отражают информацию о стоимости вида работы	Вещественное число

Сущность «Заказчик» была реализовано отношением, которое приведено в таблице 1.5. В данном отношении будут определены следующие атрибуты: «Идентификатор», «Полное имя заказчика (ФИО)», «Адрес заказчика», «Мобильный телефон», «Паспортные данные». Данная сущность связана отношением «один ко многим» с сущностью «Заказы». Сущность «Заказчик» находится на стороне отношений «многие» с сущностью «Заказы», поэтому должна содержать список заказов. Данная сущность находится в нормализованном виде.

Таблица 1.5 – Отношение, описывающие сущность «Заказчик»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого заказчика. Является первичным ключом.	Целое число
ФИО	Содержит фамилию, имя и отчество заказчика	Строка
Адрес заказчика	Содержит информацию об месте проживания заказчика	Строка
Мобильный телефон	Содержит информацию о мобильном телефон заказчика.	Строка
Паспортные данные	Содержит информацию о паспортных данных заказчика	Строка

Для сущности «Заказы» было создано отношение (таблица) с атрибутами: «Идентификатор заказа», «Стоимость», «Дата начала работы», «Дата окончания», «Отметка о выполнении», «Отметка об оплате». Данная сущность находится в ненормализованном виде и имеет связь один ко многим с отношениями «Вид работ», «Бригада» и «Заказчик» соответственно. После приведения данной сущности к нормализованному виду появится следующий атрибут, «Код вида работ», «Код бригады» и «Код заказчика» соответственно. Подробное описание отношения и атрибутов приведено в таблице 1.6. Данное отношение находится в первой нормальной форме.

Таблица 1.6 – Отношение, описывающее сущность «Заказы»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого заказа. Является первичным ключом.	Целое число
Дата начала	Содержит дату начала работы над заказом	Дата

Продолжение таблицы 1.6

Дата окончания	Содержит дату окончания работы после	Дата
Отметка об оплате	Содержит информацию об оплате клиентом заказа	Логическое значение
Отметка о выполнении	Содержит отметку о результате выполнения работы	Логическое значение
Код вида работ	Содержит ссылку на идентификатор вида работ. Является внешним ключом для связи с отношением «Вид работ».	Число
Код бригады	Содержит ссылку на идентификатор бригады. Является внешним ключом для связи с отношением «Бригада».	Число
Код заказчика	Содержит ссылку на идентификатор заказчика. Является внешним ключом для связи с отношением «Заказчик».	Число

Сущность «Бригада» была реализовано отношением, которое приведено в таблице 1.7. В данном отношении будут определены следующие атрибуты: «Идентификатор», «Наименование. Данная сущность связана отношением «один ко многим» с сущностями «Заказы» и «Сотрудники». Сущность «Бригада» находится на стороне отношений «многие» с сущностями «Заказы» и «Сотрудники», поэтому должна содержать список заказов и список сотрудников. Данная сущность находится в нормализованном виде.

Таблица 1.7 – Отношение, описывающее сущность «Бригада»

Атрибуты	Описание домена	Тип данных
Адрес заказчика	Содержит информацию об месте проживания заказчика	Строка
Мобильный телефон	Содержит информацию о мобильном телефон заказчика.	Строка

Продолжение таблицы 1.7

Паспортные данные	Содержит информацию о паспортных данных заказчика	Строка

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

1.2 Физическая модель базы данных

SQL Server одновременно может поддерживать до 32767 баз данных, которые могут иметь связи друг с другом или внешними базами данных. После установки пакета создаются четыре системы базы данных: master – системные хранимые процедуры и системные таблицы, *msdb* – репликация и восстановление баз данных, *tempdb* – временные объекты для пользователей и промежуточные результаты выполнения запросов, *model* – модельная база данных (вновь создаваемые базы данных используют данную базу как шаблон, включая набор объектов и прав), и две пользовательских

Основная единица хранения данных на уровне файла базы данных – страница (*page*). При дисковых операциях чтения-записи страница обрабатывается целиком. В *SQL Server* размер страницы равен 8192 байт. Первые 96 байт отводятся под заголовок, в котором хранится системная информация о типе страницы, объёме свободного места на странице и идентификационном номере объекта базы данных, которому принадлежит эта страница. Базовые типы страниц: *data, index, text/image*. После заголовка идёт область данных, а в конце страницы – таблица смещений строк, в которой указывается начало каждой записи относительно начала страницы. При удалении строки пустое пространство помечается и потом его может занять новая строка, но перемещения строк не происходит.

Для более эффективного управления страницами *SQL Server* использует объединения страниц – экстенты (*extent* – непрерывная область). Каждый экстент содержит 8 страниц и занимает 64 Кбайт. Для управления экстентами используются страницы специального типа *GAM – Global Allocation Map*, каждая из которых может хранить информацию о заполнении 64 000 экстентов. Специальные страницы типа *PFS* (*Page Free Space*) – используются для сбора информации о свободном месте не страницах. Это намного убыстряет процесс записи в базу, так как серверу для поиска оптимального варианта записи достаточно перебрать только страницы *PFS*. Для сбора информации о том, какому объекту принадлежит страница, используются страницы специального типа *IAM – Index Allocation Map*. Для объекта-владельца создаётся собственная страница *IAM*, в которой указываются принадлежащие ему экстенты. Если одной *IAM* не хватает, создаётся цепочка страниц этого типа.

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.

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.

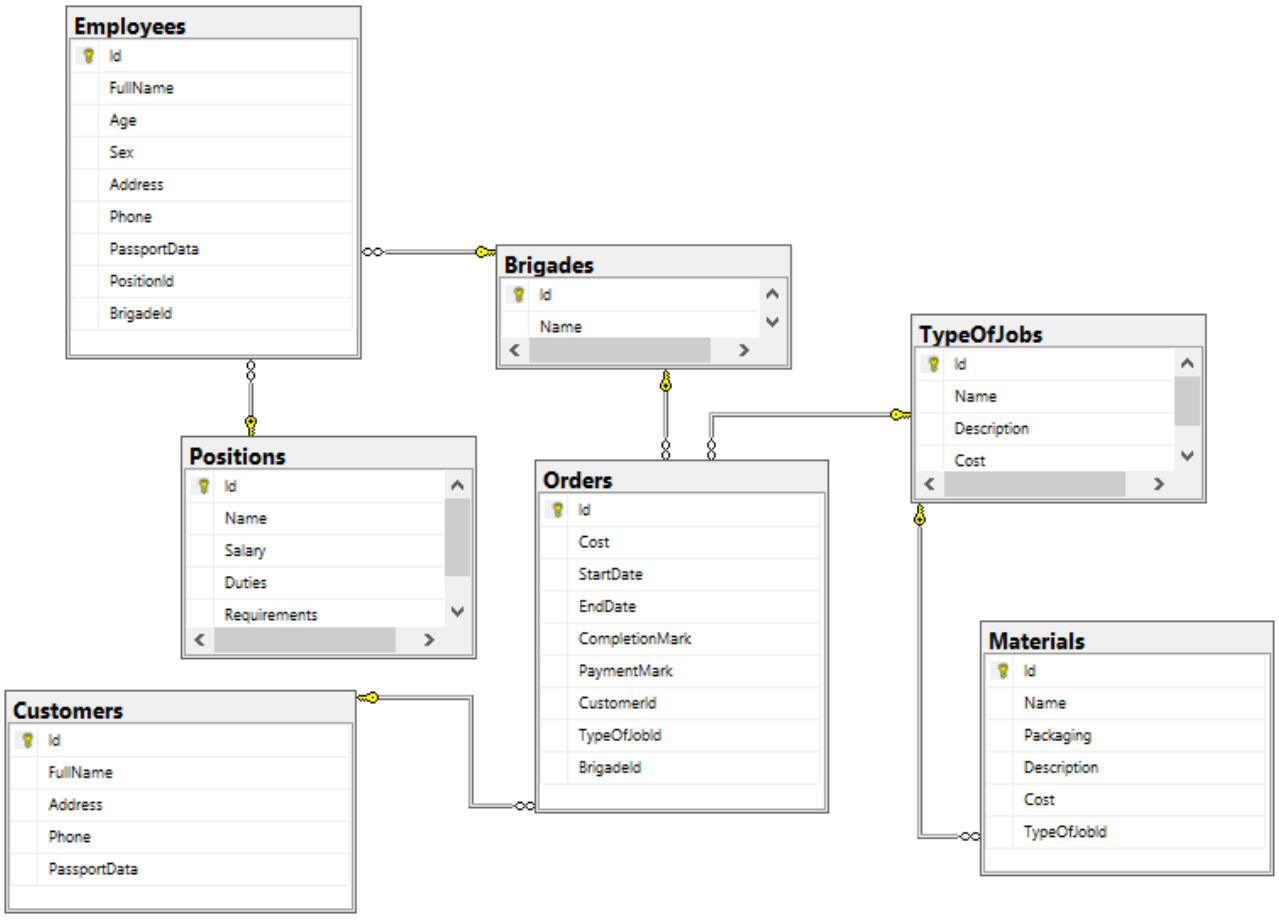

Рисунок 1.1 – Диаграмма базы данных

1.3 Файловая структура базы данных

Каждая база данных *SQL Server* имеет как минимум два 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. Файлы данных могут быть объединены в файловые группы для удобства распределения и администрирования.

По умолчанию и данные, и журналы транзакций помещаются на один и тот же диск и имеют один и тот же путь для обработки однодисковых систем. Для производственных сред это может быть неоптимальным решением. Рекомендуется помещать данные и файлы журнала на разные диски.

Файлы *SQL Server* имеют два типа имен файлов.

logical_file_name: имя, используемое для ссылки на физический файл во всех инструкциях *Transact-SQL*. Логическое имя файла должно соответствовать правилам для идентификаторов *SQL Server* и быть уникальным среди логических имен файлов в соответствующей базе данных.

os_file_name: имя физического файла, включающее путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы.

Файлы *SQL Server* могут автоматически увеличиваться в размерах, превосходя первоначально заданные показатели. При определении файла пользователь может указывать требуемый шаг роста. Каждый раз при заполнении файла его размер увеличивается на указанный шаг роста. Если в файловой группе имеется несколько файлов, их автоматический рост начинается лишь по заполнении всех файлов.

Для каждого отношения были получены следующие таблицы: *Brigades, Customers, Employees, Materials, Orders, Positons, TypeOfJobs*.

Таблицы *Positions, Customers, TypeOfJobs, Brigades* находятся в отношении «один» и описывают сущности «Должность», «Заказчик», «Тип работ» и «Бригада» и подобраны физические тип данных для соответствующих столбцов, установлены первичные ключи.

Таблицы *Employees*, и *Orders* и *Materials* находятся в отношении «многие», описывают сущности «Сотрудник», «Заказ» и «Материал». 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 ИНФОРМАЦИОННОЙ СИСТЕМЫ

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

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

– установленный *MS SQL Server*;

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

– сетевое программное обеспечение;

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

– минимальный объем оперативной памяти 1 ГБ;

– процессор *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*, *Mac 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 ConstructionCompany -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 ConstructionCompany.dll*» с командной строки, после этого веб-приложение будет запущенно на сервере. Чтобы пользователь мог использовать веб-приложение, он должен находится в одной сети с веб-сервером.

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 *Brigade, Customer, Employee, Material, Order, Position, TypeOfJob* описывают таблицы *Brigades, Customers, Employees, Materials, Orders, Positions, TypeOfJobs* 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.

Кроме того, в вышеописанных классах можно использовать аннотации – специальные атрибуты, которые определяют различные правила для отображе-

ния 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 патерна *{controller=[ControllerName]}/{action=[MethodName]}*, где [*ControllerName*] – название контроллера, [*MethodName*] – название метода контроллера.

Для работы с созданными моделями разработаны следующие контроллеры:

– *HomeController* – отвечает за вывод начальной страницы;

– *BrigadesController* – отвечает за работу с таблицей *Brigades;*

– *CustomersController* – отвечает за работу с таблицей *Customers;*

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

– *OrdersController* – отвечает за работу с таблицей *Orders;*

– *MaterialsController* – отвечает за работу с таблицей *Materials;*

– *PositionsController* – отвечает за работу с таблицей *Positions.*

– *TypeOfJobsController* – отвечает за работу с таблицей *TypeOfJobs.*

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

– *Index;*

– *Details[GET];*

– *Create[GET];*

– *Create[POST];*

– *Edit[GET];*

– *Edit[POST];*

– *Delete[GET];*

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

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

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

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

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

Метод *Edit[POST]* вызывается при отправке результата формы редактирования записи. Данный метод в качестве входных параметров принимает идентификатор записи и объект, содержащий данные об этой записи. Если входной идентификатор и идентификатор объекта не совпадают, то метод возвращает стандартное сообщение об ошибке. Иначе метод выполняет валидацию входных данных и если данные верны, то производится обновление данных в базе. Если

операция обновления прошла успешно, то происходит переход в метод *Index* текущего контроллера. В случае возникновения ошибки метод возвращает стандартное сообщение об ошибке.

3.4 Описание представлений

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

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

– *Brigades* – содержит представления для работы с данными о бригадах, для данного представления был создан дополнительный класс «модель-представление»;

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

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

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

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

– *Positions* – содержит представления для работы с данными о должностях сотрудников, для данного представления был создан дополнительный класс «модель-представление», с помощью которого можно передавать несколько объектов представлению.

– *TypeOfJobs* – содержит представления для работы с данными о видах работ, 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, кэшируется в кэше браузера с помощью атрибута *ResponseCache* (кэшируются *css* стили, *html* страничка).

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Введение

Данное *web*-приложение производит автоматизацию процесса предоставления рекламных услуг и учёта данных внутренней экосистемы различных рекламных агенств.

4.2 Назначение, условие применения и функционал

Web-приложение предназначено для управления и учёта данных в строительной компании.

Основные функции приложения:

– просмотр списка бригад с информацией о сотрудниках, которые в них работают;

– список работ, в которых использовалось не менее трех видов материалов;

– отображение информации о сотрудниках, работающих на определенной должности;

– отображение заказаов, сделанных определенным заказчиком;

– отображение количества заказов, выполненых заданной бригадой;

– добавление, просмотр и редактирования данных о сотрудниках;

– добавление, просмотр и редактирования данных о должностях сотрудников;

– добавление, просмотр и редактирования данных о видах работы;

– добавление, просмотр и редактирования данных о материалах.

– добавление, просмотр и редактирования данных о бригадах.

– добавление, просмотр и редактирования данных о заказчиках;

– добавление, просмотр и редактирования данных о заказах;

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 видах рекламы, требуется выбрать вкладку «*Advertising*» в вверху окна браузера, рисунок 4.1.

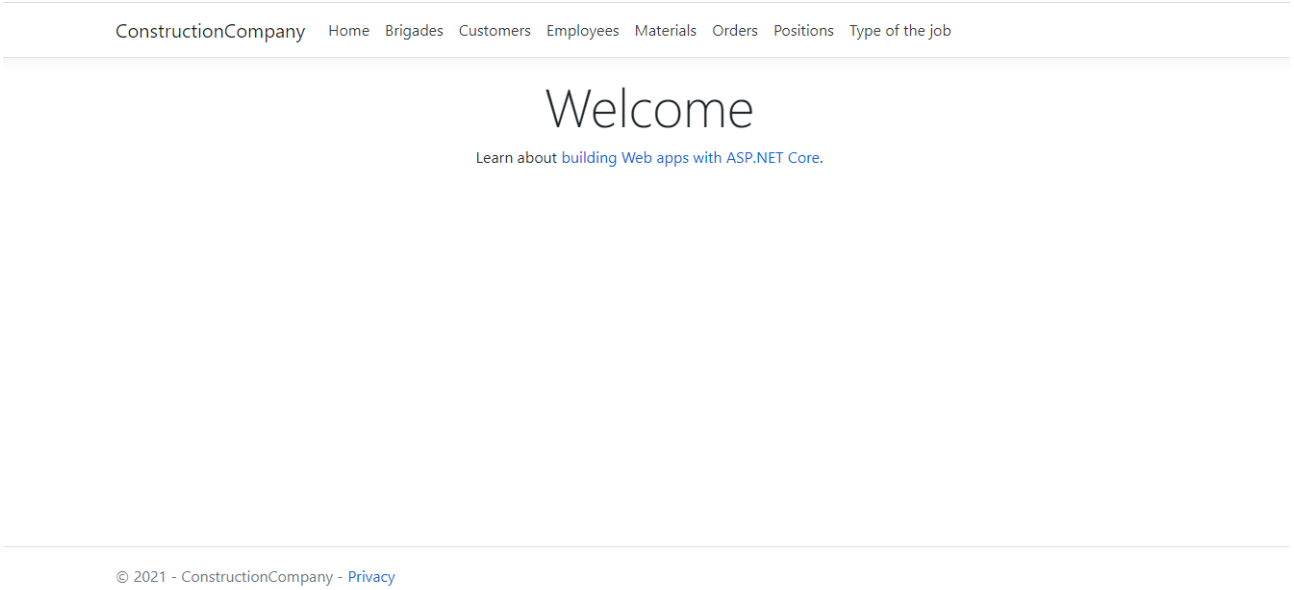

Рисунок 4.1 – Выбор сервиса

Затем загрузится новое окно со списком сотрудников, рисунок 4.2.

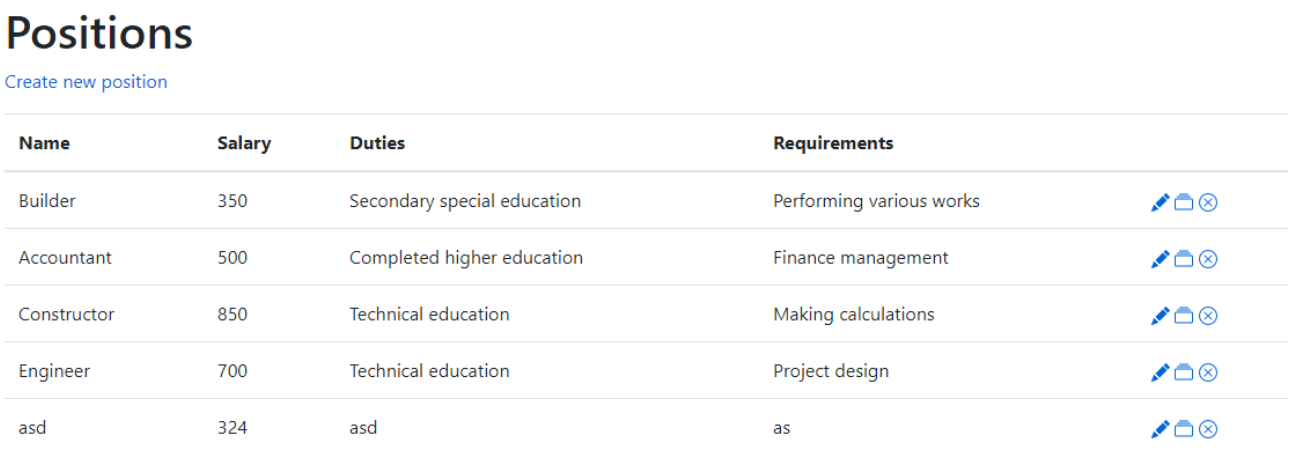

Рисунок 4.2 – Список должностей

Чтобы добавить нового сотрудника, нужно нажать на ссылку «Create new position», под названием вкладки (в данном случае – «*Positions*»), затем загрузится новая страница, рисунок 4.3 с формами для создания нового вида рекламы (все поля формы имеют проверку на корректность введённых данных). Чтобы завершить созданрие нового вида рекламы, нужно нажать на кнопку «*Create*».

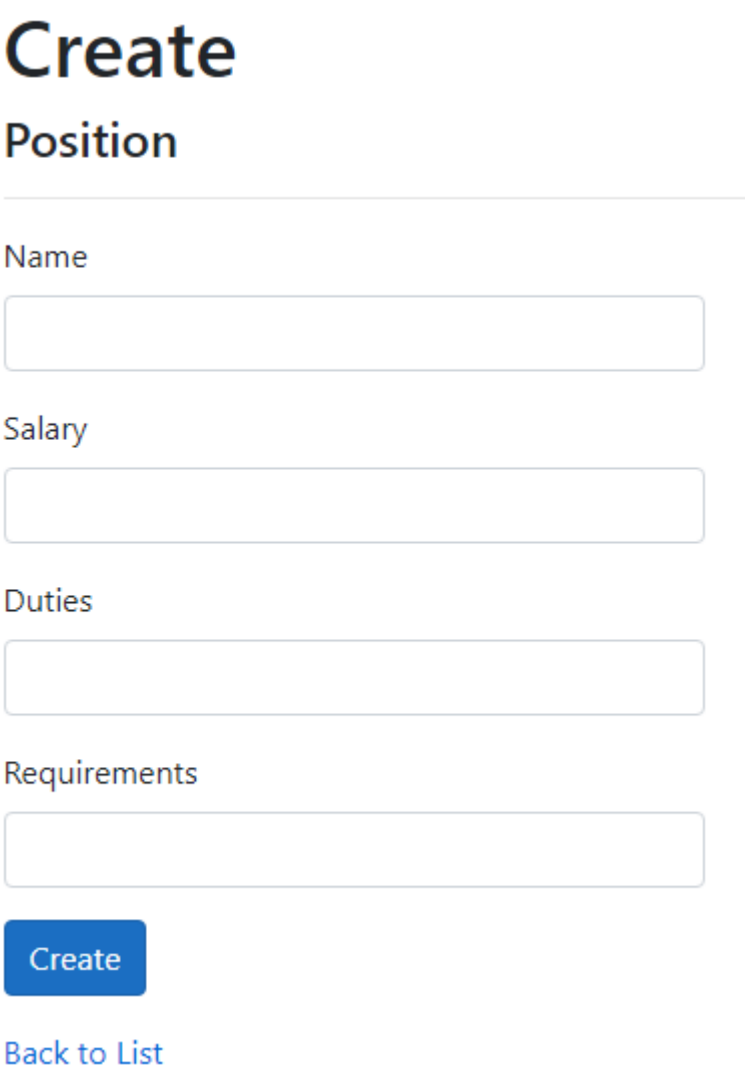

Рисунок 4.3 – Форма для добавления новой должности

После добавления нового вида рекламы пользователь будет перенаправлен на страницу с выборкой.

Для удаления вида рекламы, требуется выбрать необходимый вид рекламы в таблице и в самой правой части данной таблицы выбрать пункт «Удалить», рисунок 4.4, после чего выбранный вид рекламы будет удалён из базы данных.

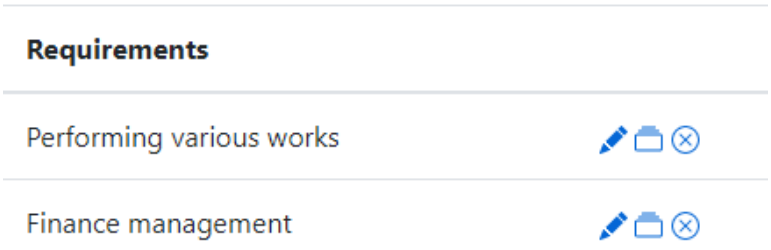

Рисунок 4.4 – Пункты для редактирования, просмотра и удаления должности соответственно

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.

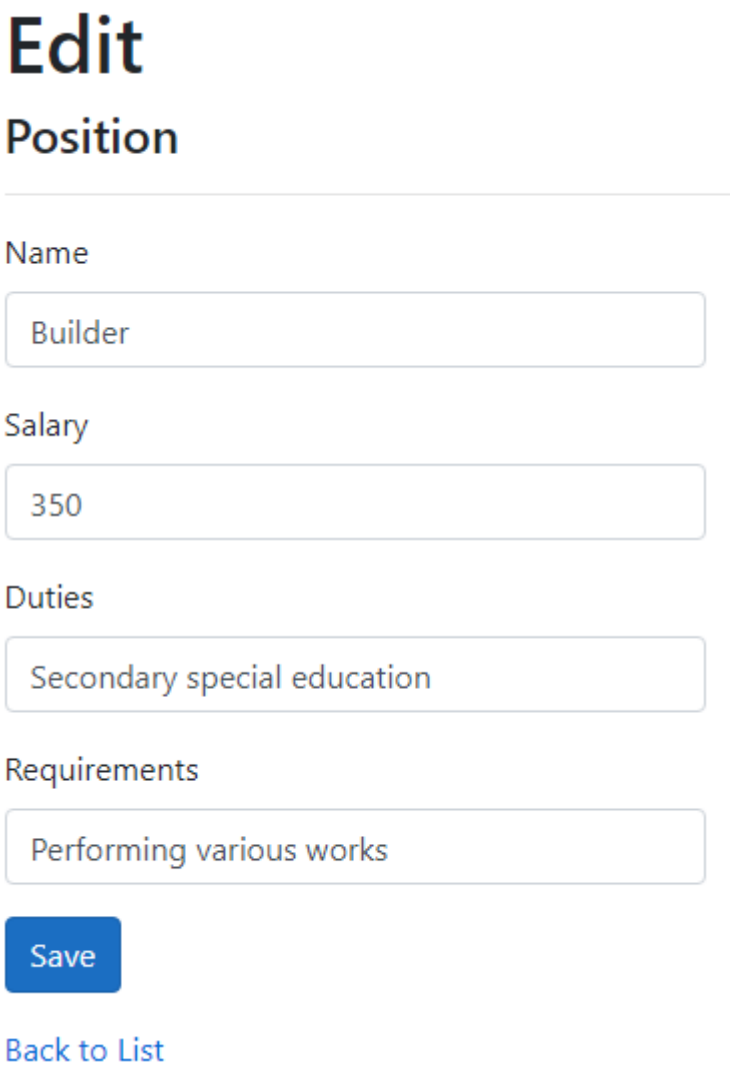

Рисунок 4.5 – Окно для редактирования данных

Так же можно посмотреть подробности о выбранном виде рекламы, нажав на кнопку «Детали», рисунок 4.4. Далее загружается окно с подробным описанием вида рекламы, рисунок 4.6.

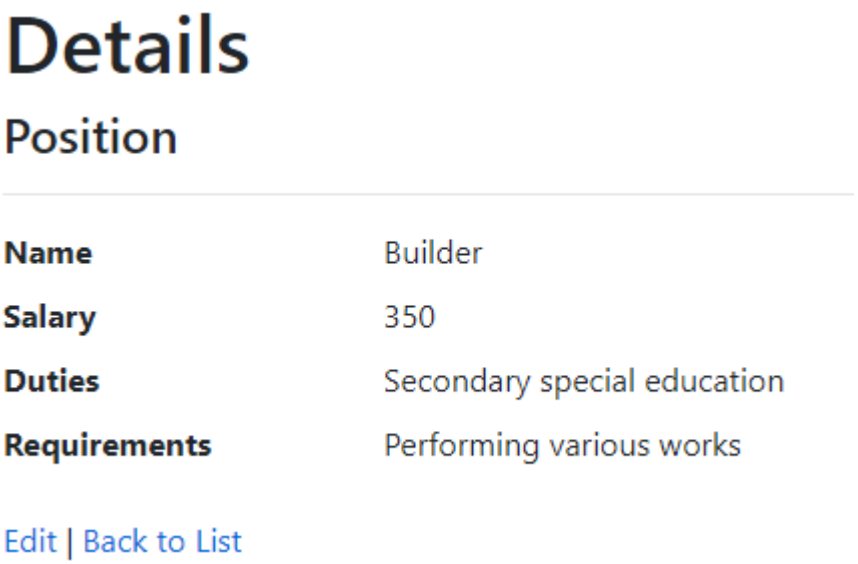

Рисунок 4.6 – Подробности о сотруднике

Для некоторых вкладок реализованы инструменты фильтрации данных, для последующего анализа, рисунок 4.7.

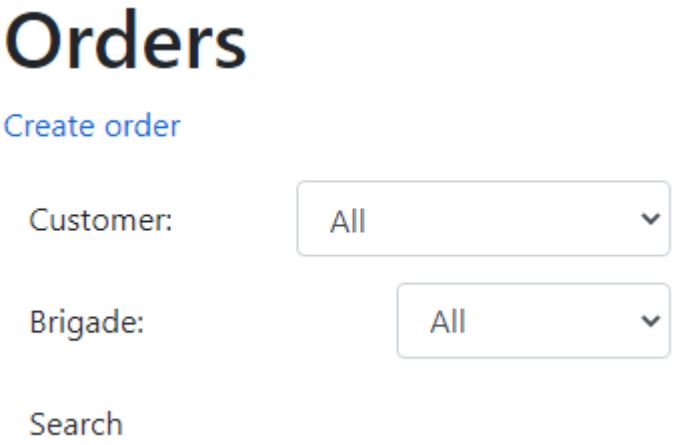

Рисунок 4.7 – Формы для фильтрации данных

Для фильтрации данных требуется ввести соответствующие данные в формы и нажать кнопку «Search».

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

5 РУКОВОДСТВО ПРОГРАММИСТА

5.1 Назначения и условия применения программы

Приложение предназначена для предоставления информации из базы данных предприятия.

Основные функции приложения:

– производить различные манипуляции с данными из базы данных;

– предоставления данных в удобном виде пользователям для их просмотра;

– управление данными о клиентах;

– 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 *HTTPS*, 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 и учет оформления услуг, предоставлямых строительной компанией. Приложение является простым и удобным благодаря адаптивному и понятному интерфейсу. Критериями удобства является, в первую очередь, наличие навигационного меню, что позволяет пользователю всю необходимую информацию, а также улучшает навигацию между страницами, не производя при этом никаких лишних действий.

Функционал приложения является вполне достаточным для выполнения основных задач, и структура спроектирована таким образом, что его дальнейшее расширение не приведёт ни к каким трудностям: изменению структуры или переписыванию логики. Все вышеперечисленные преимущества, поможет мелким предприятиям, предоставляющим услуги проектирования и строительства различных объектов автоматизироват свой производственный процесс и учёт заказов.

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

MVC описывает простой способ создания основной структуры приложения, что позволяет легко ориентироваться в коде, т.к. он разбит на блоки, а также серьёзно упрощает отладочный процесс.

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=ConstructionCompany;Trusted_Connection=True;MultipleActiveResultSets=tru
e"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Brigade.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class Brigade
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public ICollection<Employee> Employees { get; set; }
        public ICollection<Order> Orders { get; set; }

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

Customer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
```

```
    public class Customer
    {
        public int Id { get; set; }
        public string FullName { get; set; }
        public string Address { get; set; }
        public string Phone { get; set; }
        public string PassportData { get; set; }

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

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

Employee.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class Employee
    {
        public int Id { get; set; }
        public string FullName { get; set; }
        public int Age { get; set; }
        public string Sex { get; set; }
        public string Address { get; set; }
        public string Phone { get; set; }
        public string PassportData { get; set; }

        public int PositionId { get; set; }
        public Position Position { get; set; }

        public int BrigadeId { get; set; }
        public Brigade Brigade { get; set; }
    }
}
```

Materials.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class Materials
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Packaging { get; set; } //упаковка
        public string Description { get; set; }
        public double Cost { get; set; }
```

```
        public int TypeOfJobId { get; set; }
        public TypeOfJob TypeOfJob { get; set; }
    }
}
```

Order.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class Order
    {
        public int Id { get; set; }
        public double Cost { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
        public bool CompletionMark { get; set; }
        public bool PaymentMark { get; set; }

        public int CustomerId { get; set; }
        public Customer Customer { get; set; }

        public int TypeOfJobId { get; set; }
        public TypeOfJob TypeOfJobs { get; set; }

        public int BrigadeId { get; set; }
        public Brigade Brigade { get; set; }

    }
}
```

Position.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class Position
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public double Salary { get; set; }
        public string Duties { get; set; } //обязанности
        public string Requirements { get; set; } //требования

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

        public Position()
        {
            Employees = new List<Employee>();
```

```
        }
    }
}
```

TypeOfJob.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Models
{
    public class TypeOfJob
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public double Cost { get; set; }

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

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

ConstructionCompanyContext.cs

```
using ConstructionCompany.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Data
{
    public class ConstructionCompanyContext : DbContext
    {
        public ConstructionCompanyContext(DbContextOptions options) : base(options)
        {
        }

        public DbSet<Brigade> Brigades { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Materials> Materials { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<TypeOfJob> TypeOfJobs { get; set; }
    }
}
```

DbInitializer.cs

```
using ConstructionCompany.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.Data
{
   public class DbInitializer
   {
      private static Random randObj = new Random(1);
      public static void Initialize(ConstructionCompanyContext db)
      {
         db.Database.EnsureCreated();

         int brigadeCount = 50;
         int customerCount = 100;
         int employeeCount = 300;
         int materialsCount = 40;
         int orderCount = 600;
         int typeOfJobCount = 15;

         CustomerGenerate(db, customerCount);
         PositionGenerate(db);
         BrigadeGenerate(db, brigadeCount);
         TypeOfJobGenerate(db, typeOfJobCount);
         OrderGenerate(db, orderCount);
         EmployeeGenerate(db, employeeCount);
         MaterialGenerate(db, materialsCount);
      }

      public static void OrderGenerate(ConstructionCompanyContext db, int orderCount)
      {
         if (db.Orders.Any())
         {
            return;
         }
         int countCustomers = db.Customers.Count();
         int countTypeOfJobs = db.TypeOfJobs.Count();
         int countBrigades = db.Brigades.Count();

         string[] namesVoc = { "Cofe", "Hostel", "Restaurant", "Cinema", "Shop" };
         bool[] markVoc = { true, false };

         for(var i = 0; i < orderCount; i++)
         {
            var cost = randObj.Next(30, 600);
            var startDate = DateTime.Now.AddDays(-randObj.Next(5000, 10000));
            var endDate = startDate.AddDays(randObj.Next(1000, 5000));
            var completionMark = markVoc[randObj.Next(0, 2)];
            var paymentMark = markVoc[randObj.Next(0, 2)];
            var customerId = randObj.Next(1, countCustomers + 1);
            var typeOfJobId = randObj.Next(1, countTypeOfJobs + 1);
            var brigadesId = randObj.Next(1, countBrigades + 1);

            db.Orders.Add(new Order()
            {
               Cost = cost,
               StartDate = startDate,
               EndDate = endDate,
```

```
            CompletionMark = completionMark,
            PaymentMark = paymentMark,
            CustomerId = customerId,
            TypeOfJobId = typeOfJobId,
            BrigadeId = brigadesId
        });
    }
    db.SaveChanges();
}

public static void TypeOfJobGenerate(ConstructionCompanyContext db, int typeOfJobCount)
{
    if (db.TypeOfJobs.Any())
    {
        return;
    }

    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string[] nameVoc = { "Excavation", "Asphalt production", "Technological design", "Technological design",
                    "Architectural design", "Quality control of work"};

    for(int i = 0; i < typeOfJobCount; i++)
    {
        var name = nameVoc[randObj.Next(nameVoc.GetLength(0))] + randObj.Next(typeOfJobCount);
        var description = new string(Enumerable.Repeat(chars, 10)
                    .Select(s => s[randObj.Next(s.Length)]).ToArray());
        var cost = randObj.Next(70, 350);

        db.TypeOfJobs.Add(new TypeOfJob()
        {
            Name = name,
            Description = description,
            Cost = cost
        });
    }
    db.SaveChanges();
}

public static void MaterialGenerate(ConstructionCompanyContext db, int materialCount)
{
    if (db.Materials.Any())
    {
        return;
    }

    int countTypeOfJob = db.TypeOfJobs.Count();
    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    string[] packagingVoc = {"Stretch paper", "Thermoresistant paper", "Corrugated packaging", "Strapping tape",
                        "Air bubble wrap" };
    for(var i = 0; i < materialCount; i++)
    {
        var name = new string(Enumerable.Repeat(chars, 7)
                    .Select(s => s[randObj.Next(s.Length)]).ToArray());
        var packaging = packagingVoc[randObj.Next(packagingVoc.GetLength(0))] + randObj.Next(materialCount);
        var description = new string(Enumerable.Repeat(chars, 20)
                    .Select(s => s[randObj.Next(s.Length)]).ToArray());
        var cost = randObj.Next(10, 73);
        var typeOfJobId = randObj.Next(1, countTypeOfJob + 1);

        db.Materials.Add(new Materials()
        {
```

```
                Name = name,
                Packaging = packaging,
                Description = description,
                Cost = cost,
                TypeOfJobId = typeOfJobId
            });
        }
        db.SaveChanges();
    }

    public static void EmployeeGenerate(ConstructionCompanyContext db, int employeeCount)
    {
        if (db.Employees.Any())
        {
            return;
        }

        int positionCount = db.Positions.Count();
        int brigadesCount = db.Brigades.Count();
        string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        string[] fullNamesVoc = { "Yunetova K.A", "Gramovich A.A", "Semenov D.S.", "Trofimov E.W", "Gerasimenko D.A",
                                  "Kolos W.D", "Bartnovskaya A.A", "Dashkevich D.A", "Karkozov V.V." };

        string[] addressVoc = {"Mozyr, per.Zaslonova, ", "Gomel, st.Gastelo, ", "Minsk, st.Poleskay, ", "Grodno, pr.Rechetski, ", "Vitebsk, st, International, ",
                               "Brest, pr.October, ", "Minsk, st.Basseinaya, ", "Mozyr, boulevard Youth, " };

        string[] sexsVoc = { "Man", "Woman" };

        for (int i = 0; i < employeeCount; i++)
        {
            var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(employeeCount);
            var age = randObj.Next(20, 60);
            var sexEmployee = sexsVoc[randObj.Next(0, 2)];
            var address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(employeeCount);
            var phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) + "-" + randObj.Next(10, 99);
            var passportData = new string(Enumerable.Repeat(chars, 2).Select(s => s[randObj.Next(s.Length)]).ToArray()).ToUpper() + randObj.Next(100000, 999999);
            var positionId = randObj.Next(1, positionCount + 1);
            var brigadeId = randObj.Next(1, brigadesCount + 1);

            db.Employees.Add(new Employee()
            {
                FullName = fullName,
                Age = age,
                Sex = sexEmployee,
                Address = address,
                Phone = phoneNumber,
                PassportData = passportData,
                PositionId = positionId,
                BrigadeId = brigadeId
            });
        }
        db.SaveChanges();
    }

    public static void PositionGenerate(ConstructionCompanyContext db)
    {
```

```
        if (db.Positions.Any())
        {
            return;
        }
        db.Positions.AddRange(new Position[]
        {
            new Position()
            {
                Name = "Engineer",
                Salary = 700,
                Duties = "Technical education",
                Requirements = "Project design"
            },
            new Position()
            {
                Name = "Constructor",
                Salary = 850,
                Duties = "Technical education",
                Requirements = "Making calculations"
            },
            new Position()
            {
                Name = "Accountant",
                Salary = 500,
                Duties = "Completed higher education",
                Requirements = "Finance management"
            },
            new Position()
            {
                Name = "Builder",
                Salary = 350,
                Duties = "Secondary special education",
                Requirements = "Performing various works"
            },
            new Position()
            {
                Name = "HR",
                Salary = 1100,
                Duties = "Higher education, communicativeness",
                Requirements = "Hiring"
            }
        });
        db.SaveChanges();
    }

    public static void CustomerGenerate(ConstructionCompanyContext db, int customerCount)
    {
        if (db.Customers.Any())
        {
            return;
        }

        string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        string[] fullNamesVoc = { "Yunetova K.A", "Gramovich A.A", "Semenov D.S.", "Trofimov E.W", "Gerasi-
menko D.A",
                        "Kolos W.D", "Bartnovskaya A.A", "Dashkevich D.A", "Karkozov V.V." };

        string[] addressVoc = {"Mozyr, per.Zaslonova, ", "Gomel, st.Gastelo, ", "Minsk, st.Poleskay, ", "Grodno, pr.Re-
chetski, ", "Vitebsk, st, International, ",
                          "Brest, pr.October, ", "Minsk, st.Basseinaya, ", "Mozyr, boulevard Youth, " };
        for(var i = 0; i < customerCount; i++)
```

```
            {
                var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(customerCount);
                var address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(customerCount);
                var phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) +
                        "-" + randObj.Next(10, 99);
                var passportData = chars[randObj.Next(chars.Length)].ToString() + chars[ran-
dObj.Next(chars.Length)].ToString() + randObj.Next(100000, 999999);

                db.Customers.Add(new Customer()
                {
                    FullName = fullName,
                    Address = address,
                    Phone = phoneNumber,
                    PassportData = passportData
                });
            }
            db.SaveChanges();
        }

        private static void BrigadeGenerate(ConstructionCompanyContext db, int brigadeCount)
        {
            if (db.Brigades.Any())
            {
                return;
            }

            for (var i = 0; i < brigadeCount; i++)
            {
                var name = "Brigade" + randObj.Next(1, 10000);
                db.Brigades.Add(new Brigade()
                {
                    Name = name
                });
            }
            db.SaveChanges();
        }
    }
}
```

ErrorViewModel.cs

```
using System;

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

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

BrigadesController.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 ConstructionCompany.Data;
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;

namespace ConstructionCompany.Controllers
{
    public class BrigadesController : Controller
    {
        private readonly ConstructionCompanyContext _context;

        public BrigadesController(ConstructionCompanyContext context)
        {
            _context = context;
        }

        // GET: Brigades
        public async Task<IActionResult> Index(int page = 1)
        {
            var pageSize = 5;
            var itemCount = _context.Brigades.Count();

            IQueryable<Brigade> brigades = _context.Brigades;

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

            return View(new BrigadeViewModel()
            {
                Brigades = await brigades.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize)
            });
        }

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

            var brigade = await _context.Brigades
                .FirstOrDefaultAsync(m => m.Id == id);
            if (brigade == null)
            {
                return NotFound();
            }

            return View(brigade);
        }

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

```
// POST: Brigades/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Name")] Brigade brigade)
{
    if (ModelState.IsValid)
    {
        _context.Add(brigade);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(brigade);
}

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

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

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

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(brigade);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!BrigadeExists(brigade.Id))
            {
                return NotFound();
```

```
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(brigade);
    }

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

        var brigade = await _context.Brigades
            .FirstOrDefaultAsync(m => m.Id == id);
        if (brigade == null)
        {
            return NotFound();
        }

        return View(brigade);
    }

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

    private bool BrigadeExists(int id)
    {
        return _context.Brigades.Any(e => e.Id == id);
    }
  }
}
```

CustomersController.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 ConstructionCompany.Data;
```

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;

namespace ConstructionCompany.Controllers
{
    public class CustomersController : Controller
    {
        private readonly ConstructionCompanyContext _context;

        public CustomersController(ConstructionCompanyContext context)
        {
            _context = context;
        }

        // GET: Customers
        public async Task<IActionResult> Index(int page = 1)
        {
            var pageSize = 5;
            var itemCount = _context.Customers.Count();

            IQueryable<Customer> customers = _context.Customers;

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

            return View(new CustomerViewModel()
            {
                Customers = await customers.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize)
            });
        }

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

            var customer = await _context.Customers
                .FirstOrDefaultAsync(m => m.Id == id);
            if (customer == null)
            {
                return NotFound();
            }

            return View(customer);
        }

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

        // POST: Customers/Create
```

```
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Id,FullName,Address,Phone,PassportData")] Customer customer)
        {
            if (ModelState.IsValid)
            {
                _context.Add(customer);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(customer);
        }

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

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

        // POST: Customers/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,Address,Phone,PassportData")] Customer customer)
        {
            if (id != customer.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(customer);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!CustomerExists(customer.Id))
                    {
                        return NotFound();
```

```
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(customer);
    }

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

        var customer = await _context.Customers
            .FirstOrDefaultAsync(m => m.Id == id);
        if (customer == null)
        {
            return NotFound();
        }

        return View(customer);
    }

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

    private bool CustomerExists(int id)
    {
        return _context.Customers.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 ConstructionCompany.Data;
```

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;
using ConstructionCompany.ViewModel.Filter;

namespace ConstructionCompany.Controllers
{
    public class EmployeesController : Controller
    {
        private readonly ConstructionCompanyContext _context;

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

        // GET: Employees
        public async Task<IActionResult> Index(int? selectedBrigadeId, int? selectedPositionId, int page = 1)
        {
            var pageSize = 5;
            var itemCount = _context.Employees.Count();

            IQueryable<Employee> employees = _context.Employees;

            if (selectedBrigadeId.HasValue && selectedBrigadeId != 0)
            {
                employees = employees.Where(employee => employee.BrigadeId == selectedBrigadeId);
            }

            if (selectedPositionId.HasValue && selectedPositionId != 0)
            {
                employees = employees.Where(employee => employee.PositionId == selectedPositionId);
            }

            employees = employees.Skip((page - 1) * pageSize)
                .Include(e => e.Brigade).Include(e => e.Position)
                .Take(pageSize);

            return View(new EmployeeViewModel()
            {
                Employees = await employees.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize),
                EmployeeFilter = new EmployeeFilter(selectedBrigadeId, selectedPositionId,
                                        await _context.Brigades.ToListAsync(), await _context.Positions.ToListAsync())
            });
        }

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

            var employee = await _context.Employees
                .Include(e => e.Brigade)
                .Include(e => e.Position)
                .FirstOrDefaultAsync(m => m.Id == id);
            if (employee == null)
            {
                return NotFound();
            }
```

```
            return View(employee);
        }

        // GET: Employees/Create
        public IActionResult Create()
        {
            ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name");
            ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Name");
            return View();
        }

        // POST: Employees/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Id,FullName,Age,Sex,Address,Phone,PassportData,Posi-
tionId,BrigadeId")] Employee employee)
        {
            if (ModelState.IsValid)
            {
                _context.Add(employee);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", employee.BrigadeId);
            ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Name", employee.PositionId);
            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["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", employee.BrigadeId);
            ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Name", employee.PositionId);
            return View(employee);
        }

        // POST: Employees/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,Age,Sex,Address,Phone,PassportData,Posi-
tionId,BrigadeId")] 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["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", employee.BrigadeId);
            ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Name", employee.PositionId);
            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.Brigade)
                .Include(e => e.Position)
                .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);
        }
    }
}
```

MaterialsController.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 ConstructionCompany.Data;
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;

namespace ConstructionCompany.Controllers
{
    public class MaterialsController : Controller
    {
        private readonly ConstructionCompanyContext _context;

        public MaterialsController(ConstructionCompanyContext context)
        {
            _context = context;
        }

        // GET: Materials
        public async Task<IActionResult> Index(int page = 1)
        {
            var pageSize = 5;
            var itemCount = _context.Materials.Count();

            IQueryable<Materials> materials = _context.Materials;

            materials = materials.Skip((page - 1) * pageSize)
                .Include(m => m.TypeOfJob)
                .Take(pageSize);

            return View(new MaterialViewModel()
            {
                Materials = await materials.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize)
            });
        }

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

            var materials = await _context.Materials
                .Include(m => m.TypeOfJob)
                .FirstOrDefaultAsync(m => m.Id == id);
            if (materials == null)
            {
                return NotFound();
```

```
            }

            return View(materials);
        }

        // GET: Materials/Create
        public IActionResult Create()
        {
            ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name");
            return View();
        }

        // POST: Materials/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Id,Name,Packaging,Description,Cost,TypeOfJobId")] Materials
materials)
        {
            if (ModelState.IsValid)
            {
                _context.Add(materials);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", materials.TypeOfJobId);
            return View(materials);
        }

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

            var materials = await _context.Materials.FindAsync(id);
            if (materials == null)
            {
                return NotFound();
            }
            ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", materials.TypeOfJobId);
            return View(materials);
        }

        // POST: Materials/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Packaging,Description,Cost,TypeOfJobId")] Ma-
terials materials)
        {
            if (id != materials.Id)
            {
```

```
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(materials);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MaterialsExists(materials.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", materials.TypeOfJobId);
        return View(materials);
    }

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

        var materials = await _context.Materials
            .Include(m => m.TypeOfJob)
            .FirstOrDefaultAsync(m => m.Id == id);
        if (materials == null)
        {
            return NotFound();
        }

        return View(materials);
    }

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

```
        private bool MaterialsExists(int id)
        {
            return _context.Materials.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 Microsoft.Extensions.Logging;
using BarbershopService.Models;

namespace BarbershopService.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            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 });
        }
    }
}
```

OrdersController.cs

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

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;
using ConstructionCompany.ViewModel.Filter;

namespace ConstructionCompany.Controllers
{
   public class OrdersController : Controller
   {
      private readonly ConstructionCompanyContext _context;

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

      // GET: Orders
      public async Task<IActionResult> Index(int? selectedCustomerId, int? selectedBrigadeId, int page = 1)
      {
         var pageSize = 5;
         var itemCount = _context.Orders.Count();
         var count = 0;

         IQueryable<Order> orders = _context.Orders;

         if (selectedBrigadeId.HasValue && selectedBrigadeId != 0)
         {
            count = orders.Where(order => order.BrigadeId == selectedBrigadeId && order.CompletionMark ==
true).Count();
         }

         if (selectedCustomerId.HasValue && selectedCustomerId != 0)
         {
            orders = orders.Where(order => order.CustomerId == selectedCustomerId);
         }

         orders = orders.Skip((page - 1) * pageSize)
            .Include(o => o.Brigade)
            .Include(o => o.Customer)
            .Include(o => o.TypeOfJobs)
            .Take(pageSize);

         return View(new OrderViewModel()
         {
            Orders = await orders.ToListAsync(),
            PageViewModel = new PageViewModel(itemCount, page, pageSize),
            OrderFilter = new OrderFilter(selectedCustomerId, selectedBrigadeId,
                                 await _context.Customers.ToListAsync(),
                                 await _context.Brigades.ToListAsync(), count)
         });
      }

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

```
        }

        var order = await _context.Orders
            .Include(o => o.Brigade)
            .Include(o => o.Customer)
            .Include(o => o.TypeOfJobs)
            .FirstOrDefaultAsync(m => m.Id == id);
        if (order == null)
        {
            return NotFound();
        }

        return View(order);
    }

    // GET: Orders/Create
    public IActionResult Create()
    {
        ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name");
        ViewData["CustomerId"] = new SelectList(_context.Customers, "Id", "FullName");
        ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name");
        return View();
    }

    // POST: Orders/Create
    // To protect from overposting attacks, enable the specific properties you want to bind to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("Id,Cost,StartDate,EndDate,CompletionMark,Payment-
Mark,CustomerId,TypeOfJobId,BrigadeId")] Order order)
    {
        if (ModelState.IsValid)
        {
            _context.Add(order);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", order.BrigadeId);
        ViewData["CustomerId"] = new SelectList(_context.Customers, "Id", "FullName", order.CustomerId);
        ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", order.TypeOfJobId);
        return View(order);
    }

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

        var order = await _context.Orders.FindAsync(id);
        if (order == null)
        {
            return NotFound();
        }
```

```
            ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", order.BrigadeId);
            ViewData["CustomerId"] = new SelectList(_context.Customers, "Id", "FullName", order.CustomerId);
            ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", order.TypeOfJobId);
            return View(order);
        }

        // POST: Orders/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,Cost,StartDate,EndDate,CompletionMark,Payment-
Mark,CustomerId,TypeOfJobId,BrigadeId")] Order order)
        {
            if (id != order.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(order);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!OrderExists(order.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["BrigadeId"] = new SelectList(_context.Brigades, "Id", "Name", order.BrigadeId);
            ViewData["CustomerId"] = new SelectList(_context.Customers, "Id", "FullName", order.CustomerId);
            ViewData["TypeOfJobId"] = new SelectList(_context.TypeOfJobs, "Id", "Name", order.TypeOfJobId);
            return View(order);
        }

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

            var order = await _context.Orders
                .Include(o => o.Brigade)
                .Include(o => o.Customer)
                .Include(o => o.TypeOfJobs)
```

```
            .FirstOrDefaultAsync(m => m.Id == id);
        if (order == null)
        {
            return NotFound();
        }

        return View(order);
    }

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

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

PositionsController.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 ConstructionCompany.Data;
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;

namespace ConstructionCompany.Controllers
{
  public class PositionsController : Controller
  {
    private readonly ConstructionCompanyContext _context;

    public PositionsController(ConstructionCompanyContext context)
    {
        _context = context;
    }

    // GET: Positions
    public async Task<IActionResult> Index(int page = 1)
    {
        var pageSize = 5;
        var itemCount = _context.Positions.Count();

        IQueryable<Position> positions = _context.Positions;
```

```
    positions = positions.Skip((page - 1) * pageSize)
        .Take(pageSize);

    return View(new PositionViewModel()
    {
        Positions = await positions.ToListAsync(),
        PageViewModel = new PageViewModel(itemCount, page, pageSize)
    });
}

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

    var position = await _context.Positions
        .FirstOrDefaultAsync(m => m.Id == id);
    if (position == null)
    {
        return NotFound();
    }

    return View(position);
}

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

// POST: Positions/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Name,Salary,Duties,Requirements")] Position position)
{
    if (ModelState.IsValid)
    {
        _context.Add(position);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(position);
}

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

    var position = await _context.Positions.FindAsync(id);
    if (position == null)
    {
        return NotFound();
```

```
    }
    return View(position);
}

// POST: Positions/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Salary,Duties,Requirements")] Position position)
{
    if (id != position.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(position);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!PositionExists(position.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(position);
}

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

    var position = await _context.Positions
        .FirstOrDefaultAsync(m => m.Id == id);
    if (position == null)
    {
        return NotFound();
    }

    return View(position);
}

// POST: Positions/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
```

```
            var position = await _context.Positions.FindAsync(id);
            _context.Positions.Remove(position);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool PositionExists(int id)
        {
            return _context.Positions.Any(e => e.Id == id);
        }
    }
}
```

TypeOfJobsController.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 ConstructionCompany.Data;
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel;
using ConstructionCompany.ViewModel.Filter;

namespace ConstructionCompany.Controllers
{
    public class TypeOfJobsController : Controller
    {
        private readonly ConstructionCompanyContext _context;

        public TypeOfJobsController(ConstructionCompanyContext context)
        {
            _context = context;
        }

        // GET: TypeOfJobs
        public async Task<IActionResult> Index(bool sortTypeByMaterial, int page = 1)
        {
            var pageSize = 5;
            var itemCount = _context.TypeOfJobs.Count();

            IQueryable<TypeOfJob> typeOfJobs = _context.TypeOfJobs;

            if (sortTypeByMaterial)
            {
                typeOfJobs = typeOfJobs.Where(typeOfJobs => typeOfJobs.Materials.Count() >=
3);
            }

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

            return View(new TypeOfJobViewModel()
            {
                TypeOfJobs = await typeOfJobs.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize),
                TypeOfJobFilter = new TypeOfJobFilter(sortTypeByMaterial)
            });
        }

        // GET: TypeOfJobs/Details/5
```

```
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var typeOfJob = await _context.TypeOfJobs
                .FirstOrDefaultAsync(m => m.Id == id);
            if (typeOfJob == null)
            {
                return NotFound();
            }

            return View(typeOfJob);
        }

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

        // POST: TypeOfJobs/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("Id,Name,Description,Cost")] TypeOfJob
typeOfJob)
        {
            if (ModelState.IsValid)
            {
                _context.Add(typeOfJob);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(typeOfJob);
        }

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

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

        // POST: TypeOfJobs/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description,Cost")]
TypeOfJob typeOfJob)
        {
            if (id != typeOfJob.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(typeOfJob);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!TypeOfJobExists(typeOfJob.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(typeOfJob);
        }

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

            var typeOfJob = await _context.TypeOfJobs
                .FirstOrDefaultAsync(m => m.Id == id);
            if (typeOfJob == null)
            {
                return NotFound();
            }

            return View(typeOfJob);
        }

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

        private bool TypeOfJobExists(int id)
        {
            return _context.TypeOfJobs.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 AdvertisingAgency.Middleware
{
    public static class DbInitializerExtensions
    {
        public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<DbInitializerMiddleware>();
        }
    }
}
```

DbInitializerMiddleware.cs

```
using ConstructionCompany.Data;
using Microsoft.AspNetCore.Http;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

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

        public DbInitializerMiddleware(RequestDelegate next)
        {
            // инициализация базы данных
            _next = next;
        }

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

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

BrigadeViewModel.cs

```
using ConstructionCompany.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class BrigadeViewModel
   {
      public IEnumerable<Brigade> Brigades { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

PageViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AdvertisingAgency.ViewModel
{
   public class PageViewModel
   {
      public int PageNumber { get; private set; }
      public int TotalPages { get; private set; }

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

      public bool HasPreviousPage
      {
         get
         {
            return PageNumber > 1;
         }
      }

      public bool HasNextPage
      {
         get
         {
            return PageNumber < TotalPages;
         }
      }
   }
}
```

CustomerViewModel.cs

```
using ConstructionCompany.Models;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class CustomerViewModel
   {
      public IEnumerable<Customer> Customers { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

EmployeeViewModel.cs

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel.Filter;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class EmployeeViewModel
   {
      public IEnumerable<Employee> Employees { get; set; }
      public EmployeeFilter EmployeeFilter { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

MaterialViewModel.cs

```
using ConstructionCompany.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class MaterialViewModel
   {
      public IEnumerable<Materials> Materials { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

OrderViewModel.cs

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel.Filter;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace ConstructionCompany.ViewModel
{
   public class OrderViewModel
   {
      public IEnumerable<Order> Orders { get; set; }
      public OrderFilter OrderFilter { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

PositionViewModel.cs

```
using ConstructionCompany.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class PositionViewModel
   {
      public IEnumerable<Position> Positions { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

TypeOfJobsViewModel.cs

```
using ConstructionCompany.Models;
using ConstructionCompany.ViewModel.Filter;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel
{
   public class TypeOfJobViewModel
   {
      public IEnumerable<TypeOfJob> TypeOfJobs { get; set; }
      public TypeOfJobFilter TypeOfJobFilter { get; set; }
      public PageViewModel PageViewModel { get; set; }
   }
}
```

OrderFilter.cs

```
using ConstructionCompany.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel.Filter
{
```

```
    public class OrderFilter
    {
        public int? SelectedCustomerId { get; set; }
        public SelectList Customers { get; set; }
        public int? SelectedBrigadeId { get; set; }
        public SelectList Brigades { get; set; }

        public int Count { get; set; }

        public OrderFilter(int? selelectedCustomerId, int? selectedBrigadeId, IList<Customer> customers, IList<Brigade>
brigades, int count)
        {
            brigades.Insert(0, new Brigade()
            {
                Id = 0,
                Name = "All"
            });
            customers.Insert(0, new Customer()
            {
                Id = 0,
                FullName = "All"
            });

            SelectedCustomerId = selelectedCustomerId;
            SelectedBrigadeId = selectedBrigadeId;
            Brigades = new SelectList(brigades, "Id", "Name", selectedBrigadeId);
            Customers = new SelectList(customers, "Id", "FullName", selelectedCustomerId);
            Count = count;
        }
    }
}
```

EmployeeFilter.cs

```
using ConstructionCompany.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel.Filter
{
    public class EmployeeFilter
    {
        public int? SelectedBrigadeId { get; set; }
        public int? SelectedPositionId { get; set; }
        public SelectList Brigades { get; set; }
        public SelectList Positions { get; set; }

        public EmployeeFilter(int? selelectedBrigadeId, int? selectedPositionId, IList<Brigade> brigades, IList<Position>
positions)
        {
            positions.Insert(0, new Position()
            {
                Id = 0,
                Name = "All"
            });
            brigades.Insert(0, new Brigade()
```

```
            {
                Id = 0,
                Name = "All"
            });

            SelectedBrigadeId = selelectedBrigadeId;
            SelectedPositionId = SelectedPositionId;
            Positions = new SelectList(positions, "Id", "Name", selectedPositionId);
            Brigades = new SelectList(brigades, "Id", "Name", selelectedBrigadeId);
        }
    }
}
```

TypeOfJobFilter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ConstructionCompany.ViewModel.Filter
{
    public class TypeOfJobFilter
    {
        public bool TypeFilter { get; set; }

        public TypeOfJobFilter(bool typeFilter)
        {
            TypeFilter = typeFilter;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Чертёж структуры web-приложения