

**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 «НОТАРИАЛЬНАЯ  
КОНТОРА»**»

Исполнитель: студент гр. ЗИТ-31  
Алампиев М.Н.

Руководитель: доцент  
Асенчик О.Д.

Дата проверки: \_\_\_\_\_  
Дата допуска к защите: \_\_\_\_\_  
Дата защиты: \_\_\_\_\_  
Оценка работы: \_\_\_\_\_

Подписи членов комиссии  
по защите курсовой работы: \_\_\_\_\_

Гомель 2021



## СОДЕРЖАНИЕ

Введение .....	5
1 Логическая и физическая структура базы данных .....	8
1.1 Информационно-логическая модель информационной системы .....	8
1.2 Физическая модель базы данных .....	11
1.3 Файловая структура базы данных .....	13
2 Аппаратное и программное обеспечение информационной системы.....	15
2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных .....	15
2.2 Требования к системному и прикладному программному обеспечению на стороне <i>web</i> -сервера .....	15
2.3 Требования к системному и прикладному программному обеспечению на стороне клиента.....	15
2.4 Настройка и развёртывание приложения на сервере .....	16
2.5 Описание контроллеров .....	18
2.6 Описание представлений.....	20
3 Руководство пользователя.....	21
3.1 Назначение, условие применения и функционал .....	21
3.2 Подготовка к работе .....	21
3.3 Описание операции по обработки данных .....	22
4 Руководство программиста .....	26
4.1 Назначения и условия применения программы.....	26
4.2 Характеристики программы .....	26
4.3 Сопровождение программного комплекса.....	26
4.4 Входные и выходные данные .....	27
4.5 Сообщения в ходе работы приложения .....	27
Заключение .....	28
Список используемых источников.....	29
Приложение А – Код программы .....	30
Приложение Б – Чертёж структуры web-приложения	<b>Ошибка! Закладка не определена.</b>

## ВВЕДЕНИЕ

Данная курсовой проект посвящён разработке *web*-приложения баз данных нотариального предприятия, 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»).

Нотариальное предприятие имеет очень глубокие исторические корни и постоянно совершенствовалась за всю историю существования человеческой цивилизации. Однако основные свои черты предприятия сохранили со времен античности до наших дней. История нотариальных контор – это постоянный процесс совершенствования технологий.

В качестве современного подхода к реализации всех этих условий применяется создание сайта или клиент-серверного приложения для работы в сети Интернет. Сайты представляют из себя набора страниц, объединенных в единый ресурс, и имеют простую архитектуру и небольшой размер. Приложение представляет из себя компьютерное приложение, разработанное для сети Интернет, и одной из его особенностей является работа с контентом и личными данными пользователя.

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

Для нотариальной конторы необходимо создать приложение, позволяющее создавать и отслеживать состояние заключаемых контрактов, переченню услуг, манипулированию информацией о клиентах и работников компании.

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 объема и обработку сложных запросов, а также имеет эффективные алгоритмы для работы с памятью и автоматизированным контролем размера файлов баз данных. В качестве технологии для разработки веб-приложения используется платформа *ASP.NET Core MVC* [3, с. 105]. Данная платформа является многофункциональной платформой для создания веб-приложений с помощью шаблона проектирования *Model-View-Controller* (модель-контроллер-представление). Структура *MVC* предполагает разделение приложения на три основных компонента: модель, представление и контроллер [6]. Каждый компонент решает свои задачи и взаимодействует с другими компонентами. Т.е. данный шаблон проектирования позволяет разделить задачи для каждого компонента, позволяет разрабатывать проект в команде, разделяя задачи между участниками и обеспечивает дальнейшую масштабируемость проекта. Благодаря такой схеме связей и распределения обязанностей между компонентами процесс масштабирования приложения становится проще, т.к. облегчается процесс написания кода, выполнения отладки и

тестирования компонентов. Для доступа к данным используется технология *Entity Framework Core*. Данная технология является *ORM (object-relational mapping* – отображение данных на реальные объекты) инструментом, т.е. она позволяет работать с реляционными данными, используя классы и их иерархии. Также основным преимуществом данной технологии является использование универсального интерфейса для работы с данными, что позволяет легко и быстро сменить СУБД.

# 1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

## 1.1 Информационно-логическая модель информационной системы

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:

- «Сотрудники»;
- «Должности»;
- «Услуги»;
- «Клиенты»;
- «Контракт».

Для сущности «Должности» было создано отношение (таблица) с атрибутами: «Идентификатор», «Наименование», «Оклад», «Обязанности», «Требования». Подробное описание отношения и атрибутов приведено в таблице 1.1. Данное отношение находится в первой нормальной форме.

Таблица 1.1 – Отношение описывающие сущность «Виды автомобилей»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого вида автомобиля. Является первичным ключом.	Целое число
Наименование	Содержит наименование должности сотрудника.	Строка
Оклад	Минимальный оклад соответствующей должности сотрудника.	Число с плавающей точкой
Обязанности	Описание обязанностей соответствующей должности сотрудника.	Строка

Продолжение таблицы 1.1.

Атрибуты	Описание домена	Тип данных
Требования	Список требований к сотруднику занимающую данную должность	Строка

Отношение для сущности «Сотрудники», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «ФИО», «Дата рождения», «Адрес», «Телефон», «Занимаемая должность». Данное отношение следует привести к третьей нормальной форме, заменив атрибут «Занимаемая должность» на атрибут «Код должности» связав отношение «Сотрудники» с отношением «Должности». Так как по условию задачи сущность «Сотрудники», описание сущности и атрибутов приведено в таблице 1.1, должна иметь список должностей. Тем самым будет организована связь один ко многим, между данными отношениями.

Таблица 1.2 – Отношение описывающие сущность «Виды грузов»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждого отзыва. Является первичным ключом.	Целое число
ФИО	Содержит ФИО сотрудника.	Строка
Идентификатор должности	Содержит ссылку на идентификатор должности. Является внешним ключом для связи с отношением «Должности».	Целое число
Дата рождения	Содержит дату рождения сотрудника.	Дата
Адрес	Содержит полный адрес, по которому проживает сотрудник.	Строка
Телефон	Содержит мобильный телефон сотрудника.	Строка

Отношение для «Услуги» состоит из атрибутов: «Идентификатор», «Наименование», «Стоимость». Подробное описание отношения и атрибутов приведено в таблице 1.3. Данное отношение находится в первой нормальной форме.



Таблица 1.3 – Отношение описывающие сущность «Услуги»

Атрибуты	Описание домена	Тип данных
Идентификатор	Уникальный инкрементируемый идентификатор для каждой марки. Является первичным ключом.	Целое число
Наименование	Содержит наименование услуги.	Строка
Стоимость	Содержит стоимость услуги.	Число с плавающей точкой

Для сущности «Контракт» было создано отношение (таблица) с атрибутами: «Идентификатор контракта», «Дата подписания», «Дата окончания», «Стоимость услуги», «Клиент», «Сотрудник». Данная сущность находится в ненормализованном виде и имеет связь один ко многим с отношениями «Клиент», «Сотрудник» и «Услуга». После приведения данной сущности к нормализованному виду появится два атрибута, такие как «Код услуги», «Код клиента» и «Код сотрудника» соответственно. Подробное описание отношения и атрибутов приведено в таблице 1.4. Данное отношение находится в первой нормальной форме.

Таблица 1.4 – Отношение описывающие сущность «Услуги»

Атрибуты	Описание домена	Тип данных
Идентификатор контракта	Уникальный инкрементируемый идентификатор для каждого контракта. Является первичным ключом.	Целое число
Код сотрудника	Содержит ссылку на идентификатор сотрудника предприятия. Является внешним ключом для связи с отношением «Сотрудник».	Целое число
Код клиента	Содержит ссылку на идентификатор клиента. Является внешним ключом для связи с отношением «Клиент».	Целое число
Код услуги	Содержит ссылку на идентификатор услуги. Является внешним ключом для связи с отношением «Услуга».	Целое число
Дата подписания	Содержит дату подписания контракта.	Дата

Продолжение таблицы 1.4.

Атрибуты	Описание домена	Тип данных
Дата окончания	Содержит дату окончания действия контракта.	Дата

Сущность «Клиенты» было реализовано отношением, таблица 1.5. В данном отношении будут определены следующие атрибуты: «Идентификатор клиента», «ФИО», «Серия паспорта», «Номер паспорта», «Дата рождения», «Адрес», «Телефон».

Таблица 1.5 – Отношение описывающие сущность «Клиенты»

Атрибуты	Описание домена	Тип данных
Идентификатор клиента	Уникальный инкрементируемый идентификатор для каждой груза. Является первичным ключом.	Целое число
ФИО	Содержит ФИО сотрудника.	Строка
Серия паспорта	Содержит информацию о серии паспорта	Строка
Номер паспорта	Содержит информацию о номере паспорта.	Целое число
Дата рождения	Содержит информацию о дате рождения клиента.	Дата
Адрес	Содержит полный адрес проживания клиента.	Строка
Телефон	Содержит мобильный телефон клиента.	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 уста-

новки 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 – экстенды (*extent* – непрерывная область). Каждый экстенд содержит 8 страниц и занимает 64 Кбайт. Для управления экстендами используются страницы специального типа *GAM* – *Global Allocation Map*, каждая из которых может хранить информацию о заполнении 64 000 экстендов. 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 экстенды. 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 преобразования логической модели в физическую существует несколько правил:

- сущности становятся таблицами в физической базе данных;
- атрибуты становятся столбцами в физической базе данных. Также для каждого столбца необходимо определить подходящий тип данных;

- уникальные идентификаторы становятся столбцами, не допускающими значение *NULL*, т.е. первичными ключами. Также значение идентификатора делается автоинкрементным для обеспечения уникальности;
- все отношения моделируются в виде внешних ключей.

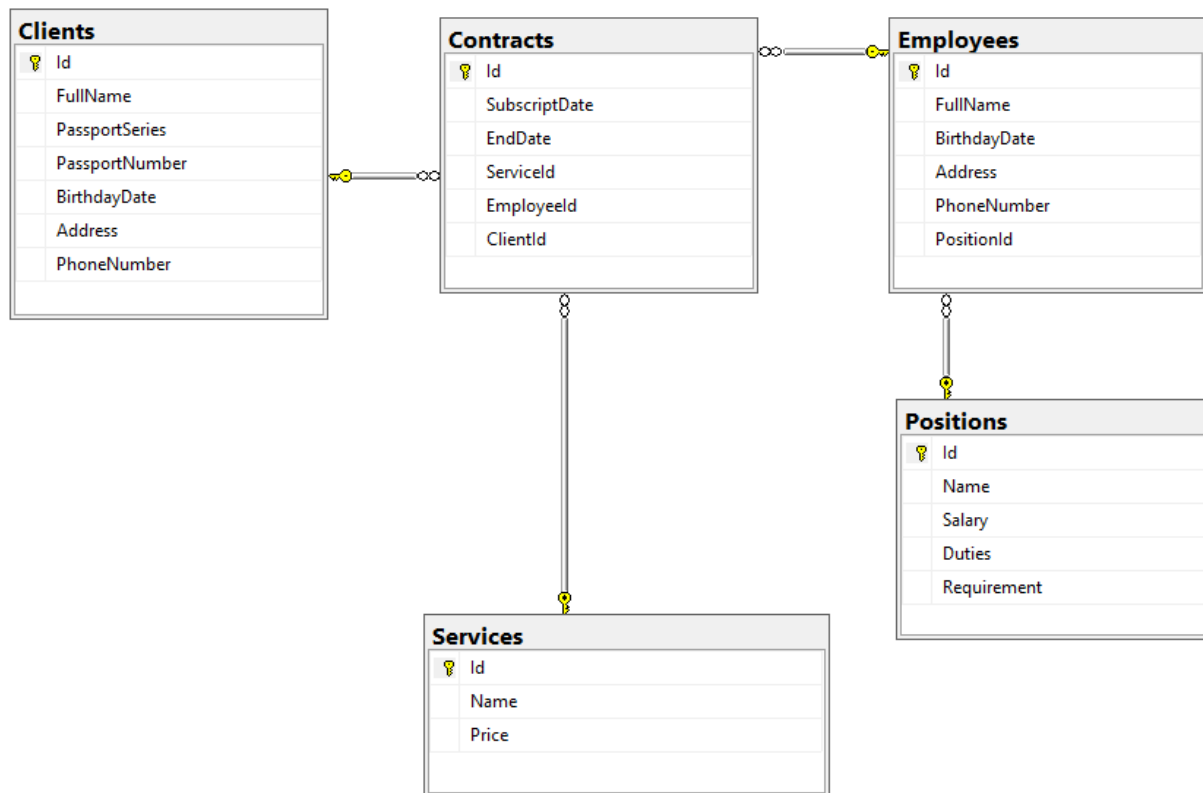


Рисунок 1.1 – Диаграмма базы данных

### 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: *Clients*, *Contracts*, *Employees*, *Positions*, *Services*.

Таблицы *Clients*, *Services* и *Positions* находятся в отношении «один» и описывают сущности «Клиенты», «Сотрудники» и «Услуги» и подобраны физические тип данных для соответствующих столбцов, установлены первичные ключи.

Таблицы *Employees* и *Contracts* находятся в отношении «многие», описывают сущности «Сотрудников» и «Контракты». Имеют автоинкрементируемый первичный ключ и внешние ключи для связи с таблицами в отношении «Один».

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

## **2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

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

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

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

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

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

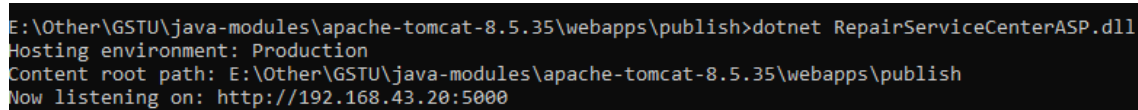
### **2.3 Требования к системному и прикладному программному обеспечению на стороне клиента**

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

## 2.4 Настройка и развёртывание приложения на сервере

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

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



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

Рисунок 2.1 – Запуск веб-приложения на веб-сервере

Чтобы подключиться к базе данных, требуется сконфигурировать подключение к ней. Для этого требуется отредактировать конфигурационный файл приложения «*appsetting.json*» и изменить строку подключение. Для того чтобы веб-приложению удалось установить соединение с базой данных, СУБД и веб-приложение должны 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 введенных пользователем данных и логику формирования ответа пользователю. Контроллер является начальной отправной точкой в приложении и отвечает за выбор рабочих типов моделей и отображаемых представлений.

### 3.2 Описание классов для доступа к данным

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

Классы *Client*, *Contract*, *Employee*, *Position* и *Service* описывают таблицы *Client*, *Contract*, *Employee*, *Position* и *Service* соответственно. Код данных классов представлен в приложении Б.

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

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



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

### 3.3 Описание контроллеров

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

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

Для работы с созданными моделями разработаны следующие контроллеры:

- *HomeController* – отвечает за вывод начальной страницы;
- *ClientsController* – отвечает за работу с таблицей *Clients*;
- *ContractsController* – отвечает за работу с таблицей *Contracts*;
- *EmployeesControllers* – отвечает за работу с таблицей *Employees*;
- *PositionsController* – отвечает за работу с таблицей *Positions*;
- *ServicesController* – отвечает за работу с таблицей *Services*.

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:

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

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

- *Services* – содержит представления для работы с данными о услугах, 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.

Для каждого представления с выборкой данных был разработан класс «модель-представление» (*ViewModel*), данный класс нужен для создание постраничной навигации. Так же эти классы содержат объекты для дополнительной манипуляции с данными (фильтрации и сортировки). Так же некоторые данные выборки, которые редко редактируются и добавляются, кешируется в кеше браузера с помощью атрибута *ResponseCache* (кешируются *css* стили, *html* страничка).

## 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 4.1 Введение

Данное *web*-приложение производит автоматизацию процесса предоставления услуг и учёта данных о внутренней экосистеме предприятий, представляющих услуг нотариальной конторы.

### 4.2 Назначение, условие применения и функционал

*Web*-приложение предназначено для управления и учёта данных о рейсах и управление данными о данных в нотариальной конторе.

Основные функции приложения:

- просмотр списков договоров, выполненных в прошлом году;
- просмотр списка клиентов, проживающих в определённом городе;
- отображение информации о сотрудниках, работающих на определённой должности;
- отображение контрактов, сделанных в определённые даты;
- отображение стоимости выполненных заказов для определённого клиента;
- добавление, просмотр и редактирования данных о должностях;
- добавление, просмотр и редактирования данных о заключаемых контрактах;
- фильтрация, сортировка и выборка данных о сотрудниках по заданным критериям;
- добавление, просмотр и редактирования данных о предоставляемых услугах;
- добавление, просмотр и редактирования данных о клиентах.

### 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 клиентах, требуется выбрать вкладку «*Clients*» вверху окна браузера, рисунок 4.1.

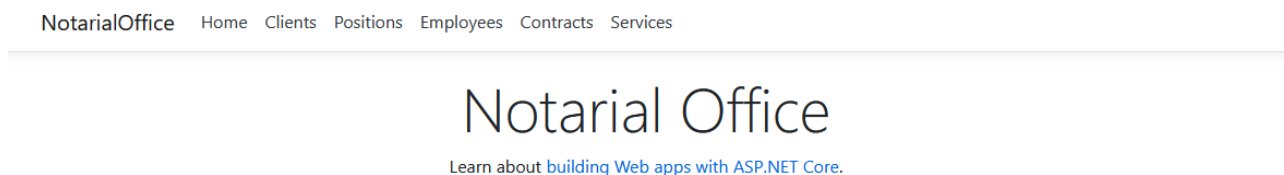


Рисунок 4.1 – Выбор сервиса

Затем загрузится новое окно со списком клиентов, рисунок 4.2.

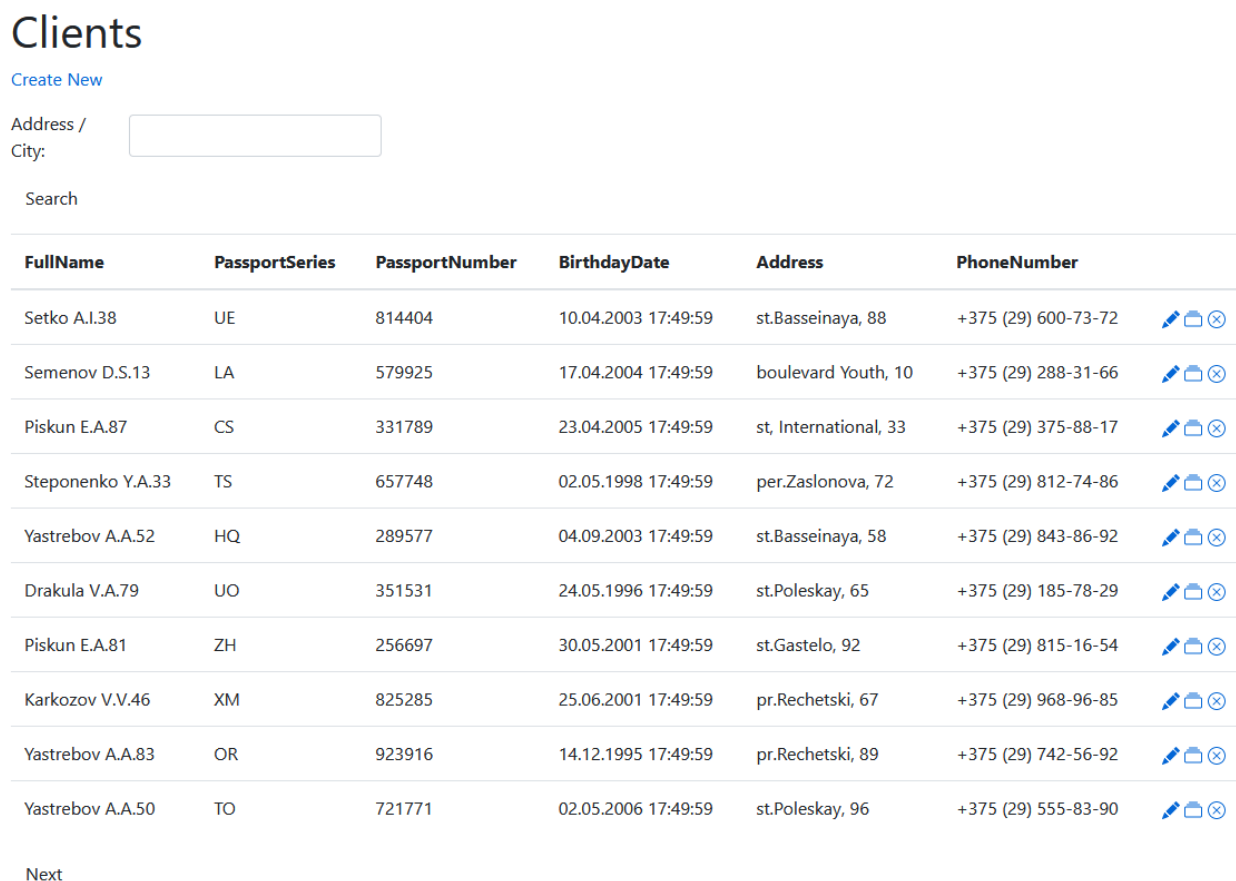


Рисунок 4.2 – Список клиентов

Чтобы добавить нового клиента, нужно нажать на ссылку, выделенную голубым цветом, под названием вкладки (в данном случае – «*Clients*»), затем загрузится новая страница, рисунок 4.3 с формами для оформления нового рейса

(все поля формы имеют проверку на корректность введенных данных). Чтобы окончательно оформить заказ, нужно нажать на кнопку «Create».

Create

Trip

Car

Tesla Minibus 59

Customer

StartLocation

FinishLocation

Cargo

Cargo 100

StartDate

ДД . ММ . ГГГГ

FinishDate

ДД . ММ . ГГГГ

Price

☐ IsPayment

☐ IsReturn

Create

[Back to List](#)

Рисунок 4.3 – Форма для добавления нового клиента

После добавления клиента пользователь будет перенаправлен на страницу с выборкой.

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













PhoneNumber	
+375 (29) 600-73-72	  
+375 (29) 288-31-66	  
+375 (29) 375-88-17	  
+375 (29) 812-74-86	  

Рисунок 4.4 – Пункты для редактирования, удаления и просмотра данных о клиенте

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.

**Edit**  
Client

FullName  
Setko A.I.38

PassportSeries  
UE

PassportNumber  
814404

BirthdayDate  
2003-04-10

Address  
st.Basseinaya, 88

PhoneNumber  
+375 (29) 600-73-72

[Save](#)

[Back to List](#)

Рисунок 4.5 – Окно для редактирования данных о рейсе

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

**Details**  
Client

<b>FullName</b>	Setko A.I.38
<b>PassportSeries</b>	UE
<b>PassportNumber</b>	814404
<b>BirthdayDate</b>	10.04.2003
<b>Address</b>	st.Basseinaya, 88
<b>PhoneNumber</b>	+375 (29) 600-73-72

[Edit](#) | [Back to List](#)

Рисунок 4.6 – Подробности рейса

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

The image shows a web interface titled "Contacts". Below the title is a link "Create New". There are three filter fields: "Subscript date:" with a date input field showing "ДД . ММ . ГГГГ", "End date:" with a date input field showing "ДД . ММ . ГГГГ", and "Client:" with a dropdown menu showing "All". Below these fields is a "Search" button.

Рисунок 4.7 – Формы для фильтрации данных

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

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, чтобы автоматизировать учёт деятельности нотариальной конторы.

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#*. Приложение реализовано с помощью паттерна *MVC (Model-View-Controller)*, который позволяет в свою очередь разделить модель данных, бизнес-логику приложения и представления, на три части, что позволит разрабатывать новый функционал и поддерживать приложения в команде из нескольких разработчиков. Так же использование данного паттерна сделала приложение легко масштабируемым и поддерживаемым.

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

## 5.4 Входные и выходные данные

Входными данными для веб-приложения является:

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

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

## 5.5 Сообщения в ходе работы приложения

При работе программа может оповещать пользователя о следующих неполадках:

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

Данные сообщения передаются в специальном виде ошибки с описанием проблемы.

## ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта было реализовано веб-приложение, которое производит автоматизацию оформления услуг предоставляемых предприятием. 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 нотариальным конторам автоматизировать свой производственный процесс и учёт заказов.

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

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

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

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

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Код программы

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=.;\\SQLEXPRESS;Database=NotarialOffice;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Client.cs

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

namespace NotarialOfficeRebuild.Models
{
    public class Client
    {
        public int Id { get; set; }
        public string FullName { get; set; }
        public string PassportSeries { get; set; }
        public int PassportNumber { get; set; }
        public DateTime BirthdayDate { get; set; }
        public string Address { get; set; }
        public string PhoneNumber { get; set; }

        public ICollection<Contract> Contracts { get; set; }

        public Client()
        {
            Contracts = new List<Contract>();
        }
    }
}
```

Contract.cs

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

namespace NotarialOfficeRebuild.Models
```

```

{
    public class Contract
    {
        public int Id { get; set; }
        public DateTime SubscriptDate { get; set; }
        public DateTime EndDate { get; set; }
        public int ServiceId { get; set; }
        public int EmployeeId { get; set; }
        public int ClientId { get; set; }

        public Employee Employee { get; set; }
        public Client Client { get; set; }
        public Service Service { get; set; }
    }
}

```

Employee.cs

```

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

namespace NotarialOfficeRebuild.Models
{
    public class Employee
    {
        public int Id { get; set; }
        public string FullName { get; set; }
        public DateTime BirthdayDate { get; set; }
        public string Address { get; set; }
        public string PhoneNumber { get; set; }
        public int PositionId { get; set; }

        public Position Position { get; set; }

        public ICollection<Contract> Contracts { get; set; }

        public Employee()
        {
            Contracts = new List<Contract>();
        }
    }
}

```

Position.cs

```

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

namespace NotarialOfficeRebuild.Models
{
    public class Position
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public double Salary { get; set; }
        public string Duties { get; set; }
        public string Requirement { get; set; }
    }
}

```

```

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

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

```

Service.cs

```

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

namespace NotarialOfficeRebuild.Models
{
    public class Service
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public double Price { get; set; }

        public ICollection<Contract> Contracts { get; set; }

        public Service()
        {
            Contracts = new List<Contract>();
        }
    }
}

```

NotarialOfficeContext.cs

```

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

namespace NotarialOfficeRebuild.Data
{
    public class NotarialOfficeContext : DbContext
    {
        public NotarialOfficeContext(DbContextOptions options) : base(options)
        {
        }

        public DbSet<Client> Clients { get; set; }
        public DbSet<Contract> Contracts { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<Service> Services { get; set; }
    }
}

```

DbInitializer.cs

```

using NotarialOfficeRebuild.Models;
using System;

```

```

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

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

            int clientCount = 100;
            int employeeCount = 50;
            int serviceCount = 20;
            int contractCount = 100;

            PositionGenerate(db);
            ClientGenerate(db, clientCount);
            EmployeeGenerate(db, employeeCount);
            ServiceGenerate(db, serviceCount);
            ContractGenerate(db, contractCount);
        }

        private static void PositionGenerate(NotarialOfficeContext db)
        {
            if (db.Positions.Any())
            {
                return;
            }

            db.Positions.AddRange(new Position[]
            {
                new Position()
                {
                    Name = "Notary",
                    Salary = 1000,
                    Duties = "Communication with clients",
                    Requirement = "Communication, diploma"
                },
                new Position()
                {
                    Name = "Assistant notary",
                    Salary = 600,
                    Duties = "Paperwork",
                    Requirement = "Communication, incomplete higher education"
                },
                new Position()
                {
                    Name = "Secretary ",
                    Salary = 400,
                    Duties = "Paperwork",
                    Requirement = "Communication, secondary education"
                },
            });

            db.SaveChanges();
        }

        private static void ClientGenerate(NotarialOfficeContext db, int count)
        {

```



```

if (db.Clients.Any())
{
    return;
}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string[] fullNamesVoc = { "Zhmailik A.V.", "Setko A.I.", "Semenov D.S.", "Davidchik A.E.", "Piskun E.A.",
    "Drakula V.A.", "Yastrebov A.A.", "Steponenko Y.A.", "Basharimov Y.I.", "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 (int i = 0; i < count; i++)
{
    var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(count);
    var passportSeries = chars[randObj.Next(chars.Length)].ToString() + chars[ran-
dObj.Next(chars.Length)].ToString();
    var passportNum = randObj.Next(100000, 999999);
    var birthdayDate = DateTime.Now.AddDays(-randObj.Next(5000, 10000));
    var address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(count);
    var phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) +
        "-" + randObj.Next(10, 99);

    db.Clients.Add(new Client()
    {
        FullName = fullName,
        PassportSeries = passportSeries,
        PassportNumber = passportNum,
        BirthdayDate = birthdayDate,
        Address = address,
        PhoneNumber = phoneNumber
    });
}
db.SaveChanges();
}

private static void EmployeeGenerate(NotarialOfficeContext db, int count)
{
    if (db.Employees.Any())
    {
        return;
    }

    int positionCount = db.Positions.Count();

    string[] fullNamesVoc =
    {
        "Lipsky D.Y.", "Stolny S.D.", "Semenov D.S.", "Deker M.A.",
        "Ropot I.V.", "Butkovski Y.V.",
        "Stepanenko Y.V.", "Moiseikov R.A.", "Rogolevich N.V.", "Gerosimenko M.A.",
        "Galetskiy A.A.", "Zankevich K.A."
    };

    string[] addressVoc = { "per.Zaslonova, ", "st.Gastelo, ", "st.Poleskay, ", "pr.Rechetski, ", "st, International, ",
        "pr.October, ", "st.Basseinaya, ", "boulevard Youth, " };

    for (int i = 0; i < count; i++)
    {
        var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(count);
        var birthdayDate = DateTime.Now.AddDays(-randObj.Next(5000, 10000));
    }
}

```

```

var address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(count);
var phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) +
    "-" + randObj.Next(10, 99);
var positionId = randObj.Next(1, positionCount + 1);

db.Employees.Add(new Employee()
{
    FullName = fullName,
    BirthdayDate = birthdayDate,
    Address = address,
    PhoneNumber = phoneNumber,
    PositionId = positionId
});
}
db.SaveChanges();
}

private static void ServiceGenerate(NotarialOfficeContext db, int count)
{
    if (db.Services.Any())
    {
        return;
    }

    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";

    for (int i = 0; i < count; i++)
    {
        var name = new string(Enumerable.Repeat(chars, 20)
            .Select(s => s[randObj.Next(s.Length)]).ToArray());
        var price = randObj.NextDouble() * 100;
        db.Services.Add(new Service()
        {
            Name = name,
            Price = price
        });
    }

    db.SaveChanges();
}

private static void ContractGenerate(NotarialOfficeContext db, int count)
{
    if (db.Contracts.Any())
    {
        return;
    }

    int serviceCount = db.Services.Count();
    int employeeCount = db.Employees.Count();
    int clientCount = db.Clients.Count();

    for (int i = 0; i < count; i++)
    {
        var subDate = DateTime.Now.AddDays(-randObj.Next(1000));
        var endDate = subDate.AddDays(randObj.Next(500));
        var serviceId = randObj.Next(1, serviceCount + 1);
        var employeeId = randObj.Next(1, employeeCount + 1);
        var clientId = randObj.Next(1, clientCount + 1);

        db.Contracts.Add(new Contract()
        {

```

```

        SubscriptDate = subDate,
        EndDate = endDate,
        ServiceId = serviceId,
        EmployeeId = employeeId,
        ClientId = clientId
    });
}
db.SaveChanges();
}
}
}

```

ErrorViewModel.cs

```

using System;

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

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

```

ClientsController.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 NotarialOfficeRebuild.Data;
using NotarialOfficeRebuild.Models;
using NotarialOfficeRebuild.ViewModels;
using NotarialOfficeRebuild.ViewModels.Filters;

namespace NotarialOfficeRebuild.Controllers
{
    public class ClientsController : Controller
    {
        private readonly NotarialOfficeContext _context;

        public ClientsController(NotarialOfficeContext context)
        {
            _context = context;
        }

        // GET: Clients
        public async Task<IActionResult> Index(string address, int page = 1)
        {
            var pageSize = 10;
            var itemCount = _context.Clients.Count();

            IQueryable<Client> clients = _context.Clients;

```

```

        if (!string.IsNullOrEmpty(address))
        {
            clients = clients.Where(client => client.Address.Contains(address));
        }

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

        return View(new ClientViewModel()
        {
            Clients = await clients.ToListAsync(),
            PageViewModel = new PageViewModel(itemCount, page, pageSize),
            ClientFilter = new ClientFilter(address)
        });
    }

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

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

        return View(client);
    }

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

    // POST: Clients/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,PassportSeries,PassportNumber,Birth-
dayDate,Address,PhoneNumber")] Client client)
    {
        if (ModelState.IsValid)
        {
            _context.Add(client);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(client);
    }
}

```

```

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

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

// POST: Clients/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,PassportSeries,PassportNumber,Birth-
dayDate,Address,PhoneNumber")] Client client)
{
    if (id != client.Id)
    {
        return NotFound();
    }

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

// GET: Clients/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {

```

```

        return NotFound();
    }

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

    return View(client);
}

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

private bool ClientExists(int id)
{
    return _context.Clients.Any(e => e.Id == id);
}
}

```

ContractsController.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 NotarialOfficeRebuild.Data;
using NotarialOfficeRebuild.Models;
using NotarialOfficeRebuild.ViewModels;
using NotarialOfficeRebuild.ViewModels.Filters;

namespace NotarialOfficeRebuild.Controllers
{
    public class ContractsController : Controller
    {
        private readonly NotarialOfficeContext _context;

        public ContractsController(NotarialOfficeContext context)
        {
            _context = context;
        }

        // GET: Contracts
        public async Task<IActionResult> Index(DateTime? subscriptionDate, DateTime? endDate, int? selectedClientId,
            int page = 1)

```

```

{
    var pageSize = 10;
    var itemCount = _context.Contracts.Count();

    IQueryable<Contract> notarialOfficeContext = _context.Contracts;

    if (selectedClientId.HasValue && selectedClientId.Value != 0)
    {
        notarialOfficeContext = notarialOfficeContext.Where(c => c.ClientId == selectedClientId);
    }

    if (subscriptionDate.HasValue)
    {
        notarialOfficeContext = notarialOfficeContext.Where(c => c.SubscriptDate >= subscriptionDate.Value);
    }

    if (endDate.HasValue)
    {
        notarialOfficeContext = notarialOfficeContext.Where(c => c.EndDate <= endDate.Value);
    }

    notarialOfficeContext = notarialOfficeContext
        .Include(c => c.Client)
        .Include(c => c.Employee)
        .Include(c => c.Service)
        .Skip((page - 1) * pageSize)
        .Take(pageSize);

    return View(new ContractViewModel()
    {
        Contracts = await notarialOfficeContext.ToListAsync(),
        PageViewModel = new PageViewModel(itemCount, page, pageSize),
        ContractFilter = new ContractFilter(subscriptionDate, endDate, selectedClientId, await _context.Cli-
ents.ToListAsync())
    });
}

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

    var contract = await _context.Contracts
        .Include(c => c.Client)
        .Include(c => c.Employee)
        .Include(c => c.Service)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (contract == null)
    {
        return NotFound();
    }

    return View(contract);
}

// GET: Contracts/Create
public IActionResult Create()
{
    ViewData["ClientId"] = new SelectList(_context.Clients, "Id", "FullName");
}

```

```

        ViewData["EmployeeId"] = new SelectList(_context.Employees, "Id", "FullName");
        ViewData["ServiceId"] = new SelectList(_context.Services, "Id", "Name");
        return View();
    }

    // POST: Contracts/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,SubscriptDate,EndDate,ServiceId,EmployeeId,ClientId")]
Contract contract)
    {
        if (ModelState.IsValid)
        {
            _context.Add(contract);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        ViewData["ClientId"] = new SelectList(_context.Clients, "Id", "FullName", contract.ClientId);
        ViewData["EmployeeId"] = new SelectList(_context.Employees, "Id", "FullName", contract.EmployeeId);
        ViewData["ServiceId"] = new SelectList(_context.Services, "Id", "Name", contract.ServiceId);
        return View(contract);
    }

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

        var contract = await _context.Contracts.FindAsync(id);
        if (contract == null)
        {
            return NotFound();
        }
        ViewData["ClientId"] = new SelectList(_context.Clients, "Id", "FullName", contract.ClientId);
        ViewData["EmployeeId"] = new SelectList(_context.Employees, "Id", "FullName", contract.EmployeeId);
        ViewData["ServiceId"] = new SelectList(_context.Services, "Id", "Name", contract.ServiceId);
        return View(contract);
    }

    // POST: Contracts/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,SubscriptDate,EndDate,ServiceId,EmployeeId,ClientId")]
Contract contract)
    {
        if (id != contract.Id)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(contract);
            }

```



```

        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ContractExists(contract.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
ViewData["ClientId"] = new SelectList(_context.Clients, "Id", "FullName", contract.ClientId);
ViewData["EmployeeId"] = new SelectList(_context.Employees, "Id", "FullName", contract.EmployeeId);
ViewData["ServiceId"] = new SelectList(_context.Services, "Id", "Name", contract.ServiceId);
return View(contract);
}

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

    var contract = await _context.Contracts
        .Include(c => c.Client)
        .Include(c => c.Employee)
        .Include(c => c.Service)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (contract == null)
    {
        return NotFound();
    }

    return View(contract);
}

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

private bool ContractExists(int id)
{
    return _context.Contracts.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 NotarialOfficeRebuild.Data;
using NotarialOfficeRebuild.Models;
using NotarialOfficeRebuild.ViewModels;
using NotarialOfficeRebuild.ViewModels.Filters;

namespace NotarialOfficeRebuild.Controllers
{
    public class EmployeesController : Controller
    {
        private readonly NotarialOfficeContext _context;

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

        // GET: Employees
        public async Task<IActionResult> Index(int? selectedPositionId, int page = 1)
        {
            var pageSize = 10;
            var itemCount = _context.Employees.Count();

            IQueryable<Employee> notarialOfficeContext = _context.Employees;

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

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

            return View(new EmployeeViewModel()
            {
                Employees = await notarialOfficeContext.ToListAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize),
                EmployeeFilter = new EmployeeFilter(selectedPositionId, 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.Position)
```

```

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

    return View(employee);
}

// GET: Employees/Create
public IActionResult Create()
{
    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<ActionResult> Create([Bind("Id,FullName,BirthdayDate,Address,PhoneNumber,PositionId")]
Employee employee)
{
    if (ModelState.IsValid)
    {
        _context.Add(employee);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["PositionId"] = new SelectList(_context.Positions, "Id", "Name", employee.PositionId);
    return View(employee);
}

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

    var employee = await _context.Employees.FindAsync(id);
    if (employee == null)
    {
        return NotFound();
    }
    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<ActionResult> Edit(int id, [Bind("Id,FullName,BirthdayDate,Address,PhoneNumber,PositionId")] 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["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.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);
}
}

```

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 });
        }
    }
}
```

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 NotarialOfficeRebuild.Data;
using NotarialOfficeRebuild.Models;

namespace NotarialOfficeRebuild.Controllers
{
    public class PositionsController : Controller
    {
        private readonly NotarialOfficeContext _context;
```

```

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

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

// 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,Requirement")] 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,Requirement")] 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);
        }
    }
}

```

ServicesController.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 NotarialOfficeRebuild.Data;
using NotarialOfficeRebuild.Models;
using NotarialOfficeRebuild.ViewModels;

namespace NotarialOfficeRebuild.Controllers
{
    public class ServicesController : Controller
    {
        private readonly NotarialOfficeContext _context;

        public ServicesController(NotarialOfficeContext context)
        {
            _context = context;
        }

        // GET: Services
        public async Task<IActionResult> Index(int page = 1)
        {
            var pageSize = 10;
            var itemCount = _context.Services.Count();

            IQueryable<Service> serviceContext = _context.Services
                .Skip((page - 1) * pageSize)
                .Take(pageSize);

            return View(new ServiceViewModel()
            {
                Services = await serviceContext.ToArrayAsync(),
                PageViewModel = new PageViewModel(itemCount, page, pageSize)
            });
        }

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

```



```

    }

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

    return View(service);
}

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

// POST: Services/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,Price")] Service service)
{
    if (ModelState.IsValid)
    {
        _context.Add(service);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(service);
}

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

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

// POST: Services/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,Price")] Service service)
{
    if (id != service.Id)
    {
        return NotFound();
    }
}

```

```

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

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

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

        return View(service);
    }

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

    private bool ServiceExists(int id)
    {
        return _context.Services.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 NotarialOfficeRebuild.Middleware
{
    public static class DbInitializerExtensions
    {
        public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<DbInitializerMiddleware>();
        }
    }
}

```

DbInitializerMiddleware.cs

```

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

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

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

        public Task Invoke(HttpContext context, IServiceProvider serviceProvider, NotarialOfficeContext 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);
        }
    }
}

```

ClientViewModel.cs

```

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

namespace NotarialOfficeRebuild.ViewModels

```

```

{
    public class ClientViewModel
    {
        public IEnumerable<Client> Clients { get; set; }
        public ClientFilter ClientFilter { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

ContractViewModel.cs

```

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

namespace NotarialOfficeRebuild.ViewModels
{
    public class ContractViewModel
    {
        public IEnumerable<Contract> Contracts { get; set; }
        public ContractFilter ContractFilter { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

PageViewModel.cs

```

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

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

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

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

        public bool HasNextPage
        {

```

```

        get
        {
            return PageNumber < TotalPages;
        }
    }
}

```

#### EmployeeViewModel.cs

```

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

namespace NotarialOfficeRebuild.ViewModels
{
    public class EmployeeViewModel
    {
        public IEnumerable<Employee> Employees { get; set; }
        public EmployeeFilter EmployeeFilter { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

#### ServiceViewModel.cs

```

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

namespace NotarialOfficeRebuild.ViewModels
{
    public class ServiceViewModel
    {
        public IEnumerable<Service> Services { get; set; }
        public PageViewModel PageViewModel { get; set; }
    }
}

```

#### ClientFilter.cs

```

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

namespace NotarialOfficeRebuild.ViewModels.Filters
{
    public class ClientFilter
    {
        public string Address { get; set; }

        public ClientFilter(string address)
        {

```

```

        Address = address;
    }
}

```

#### ContractFilter.cs

```

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

namespace NotarialOfficeRebuild.ViewModels.Filters
{
    public class ContractFilter
    {
        public DateTime? SubscriptionDate { get; set; }
        public DateTime? EndDate { get; set; }
        public int? SelectedClientId { get; set; }
        public SelectList Clients { get; set; }

        public ContractFilter(DateTime? subscriptionDate, DateTime? endDate, int? selectedClientId, IList<Client> clients)
        {
            clients.Insert(0, new Client()
            {
                Id = 0,
                FullName = "All"
            });

            SubscriptionDate = subscriptionDate;
            EndDate = endDate;
            SelectedClientId = selectedClientId;
            Clients = new SelectList(clients, "Id", "FullName", selectedClientId);
        }
    }
}

```

#### EmployeeFilter.cs

```

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

namespace NotarialOfficeRebuild.ViewModels.Filters
{
    public class EmployeeFilter
    {
        public int? SelectedPositionId { get; set; }
        public SelectList Positions { get; set; }

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

```

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