**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 «Брачное агенство»**

Исполнитель: студент гр. ЗИТ-31

Соколова Т.Г.

Руководитель: доцент

Асенчик О.Д.

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

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

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

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

Подписи членов комиссии

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

Гомель 2021

Рецензия

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc66923564)

[1 Логическая и физическая структура базы данных 6](#_Toc66923565)

[1.1 Информационно-логическая модель информационной системы 6](#_Toc66923566)

[1.2 Физическая модель базы данных 10](#_Toc66923567)

[1.3 Файловая структура базы данных 11](#_Toc66923568)

[2 Аппаратное и программное обеспечение информа- ционной системы 13](#_Toc66923569)

[2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных 13](#_Toc66923570)

[2.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера 13](#_Toc66923571)

[2.3 Требования к системному и прикладному программному обеспечению на стороне клиента 13](#_Toc66923572)

[2.4 Настройка и развёртывание приложения на сервере 14](#_Toc66923573)

[3.3 Описание контроллеров 16](#_Toc66923574)

[3.4 Описание представлений 18](#_Toc66923575)

[4 Руководство пользователя 19](#_Toc66923576)

[4.2 Назначение, условие применения и функционал 19](#_Toc66923577)

[4.3 Подготовка к работе 19](#_Toc66923578)

[4.4 Описание операции по обработки данных 20](#_Toc66923579)

[5 Руководство программиста 24](#_Toc66923580)

[5.1 Назначения и условия применения программы 24](#_Toc66923581)

[5.2 Характеристики программы 24](#_Toc66923582)

[5.3 Сопровождение программного комплекса 24](#_Toc66923583)

[5.4 Входные и выходные данные 25](#_Toc66923584)

[5.5 Сообщения в ходе работы приложения 25](#_Toc66923585)

[Заключение 26](#_Toc66923586)

[Список используемых источников 27](#_Toc66923587)

[Приложение А – Код программы 28](#_Toc66923588)

[Приложение Б – Чертёж структуры web-приложения 57](#_Toc66923589)

# ВВЕДЕНИЕ

В современном мире каждый день, каждый час, даже каждую минуту мы не только получаем, но и обрабатываем невероятное множество единиц информации. Но ресурсы памяти не безграничны, а информации становится все больше и больше. И если для комфортной жизни человек научился отфильтровывать ненужное, то для решения бизнес задач потребуется работать каждый день с большими объемами информации: базами данных, каталогами, документооборотом и другими.

Представим, что нужно оперативно обновлять большой объем информации, и делать это приходится с определенной периодичностью (ассортимент, стоимость, наличие на складе или производстве и т.п.). Если заниматься этим вручную, то потребуется не только большой штат сотрудников, но и внушительное количество времени. Такую задачу легко значительно упростить, автоматизировав работу своего предприятия. С развитием технологий любое предприятие получило возможность автоматизировать свою работу.

После внедрения автоматизации работы предприятия компания получает следующие плюсы:

Легкий и быстрый сбор информации и ее обработка.

Оперативное получение всех актуальных показателей, связанных с работой.

Автоматизация документооборота, складского учета и бухгалтерского учета.

Сбор, хранение и учет всей необходимой информации о клиентах, истории их заказов.

Автоматический расчет цен.

Оптимизация работы персонала и прочее.

Кроме всего, руководитель или собственник после внедрения системы автоматизации получает возможность объективно оценить работу своих сотрудников.

Задача данного курсового проекта спроектировать и создать базу данных клиент-серверной СУБД и соответствующее web-приложение в заданной предметной области. Web приложение должно содержать классы для доступа к данным, контроллеры и представления. Созданные с использованием технологии ASP.NET MVC (ASP.NET Razor Pages), отображать, добавлять, удалять и редактировать данные из таблиц базы данных.

# 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»;

– «Знаки зодиака»;

– «Отношения»;

– «Национальности»;

– «Клиенты»;

– «Услуги».

Для сущности «Сотрудники» было создано отношение (таблица) с атрибутами: «Идентификатор», «ФИО», «Возраст», «Пол», «Адрес», «Паспортные данные», «Должность». Подробное описание отношения и атрибутов приведено в таблице 1.1. Данное отношение находится в первой нормальной форме.

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 предприятия. | Строка |
| Возраст | Год рождения сотрудника предприятия. | Целое число |
| Адрес | Содержит данные о месте проживания сотрудника. | Строка |

Продолжение таблицы 1.1.

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Паспортные данные | Содержит серию и номер паспорта сотрудника предприятия. | Строка |
| Пол | Содержит пол сотрудника. | Строка |

Отношение для сущности «Знаки зодиака», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «Наименование», «Описание». Так как по условию задачи сущность «Знаки зодиака», описание сущности и атрибутов приведено в таблице 1.2, должна иметь список должностей.

Таблица 1.2 – Отношение описывающие сущность «Знаки зодиака»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор | Уникальный инкрементируемый идентификатор для каждого знака зодиака. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование знака зодиака. | Строка |
| Описание | Содержит описание знака зодиака. | Строка |

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

Таблица 1.3 – Отношение описывающие сущность «Отношения»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор | Уникальный инкрементируемый идентификатор для каждого вида отношения. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование. | Строка |
| Описание | Содержит полное описание вида отношений. | Строка |

Для сущности «Национальности» было создано отношение (таблица) с атрибутами: «Идентификатор национальности», «Наименование», «Описание». Подробное описание отношения и атрибутов приведено в таблице 1.4. Данное отношение находится в первой нормальной форме.

Таблица 1.4 – Отношение описывающие сущность «Клиенты»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор национальности | Уникальный инкрементируемый идентификатор для каждого клиента. Является первичным ключом. | Целое число |
| ФИО | Содержит ФИО клиента. | Строка |
| Пол | Содержит пол клиента. | Строка |
| Дата рождения | Содержит дату рождения клиента. | Дата |
| Возраст | Содержит возраст клиента. | Целое число |
| Рост | Содержит рост клиента. | Целое число |
| Вес | Содержит вес клиента. | Целое число |
| Количество детей | Содержит количество детей. | Целое число |
| Семейное положение | Содержит информацию о семейном положении клиента. | Строка |
| Вредные привычки | Содержит информацию о вредных привычках клиента. | Строка |
| Хобби | Содержит информацию о хобби клиентов. | Строка |
| Описание | Содержит описание клиента. | Строка |
| Код знака зодиака | Содержит ссылку на знак зодиака клиента. | Целое число |
| Код отношения | Содержит ссылку на отношения клиента. | Целое число |

Продолжение таблица 1.4.

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код национальности | Содержит ссылку на национальность. | Целое число |
| Адрес | Содержит адрес клиента. | Строка |
| Телефон | Содержит номер телефона клиента. | Строка |
| Паспортные данные | Содержит сведенья о паспортных данных клиента. | Строка |

Сущность «Услуги» было реализовано отношением, таблица 1.5. В данном отношении будут определены следующие атрибуты: «Идентификатор услуги», «Дата», «Стоимость», «Клиент», «Сотрудник».

Таблица 1.5 – Отношение описывающие сущность «Услуги»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор услуги | Уникальный инкрементируемый идентификатор для каждой оказанной услуги. Является первичным ключом. | Целое число |
| Дата | Содержит дату оказания услуги. | Дата |
| Стоимость | Содержит значения стоимости услуги. | Число с плавающей точкой |
| Код клиента | Содержит ссылку на клиента. | Целое число |
| Код сотрудника | Содержит ссылку на сотрудника оказывающий услугу. | Целое число |

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.

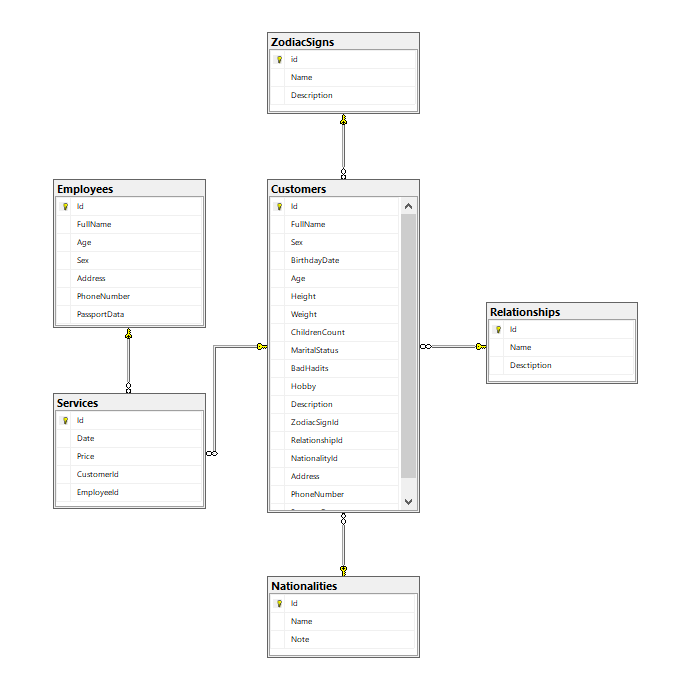


Рисунок 1.1 – Диаграмма базы данных

## **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: *Customers*, *Employees*, *Services*, *Nationalities*, *ZodiacSign* и *Relationships*.

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

Таблицы *Employees, Nationalities, ZodiacSign* и *Relationships* находятся в отношении «многие», описывают сущности «Сотрудники», «Знаки зодиака» и «Отношения». 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 ИНФОРМА– ЦИОННОЙ 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 MarriageAgency -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 MarriageAgency.dll*» с командной строки, рисунок 2.1, после этого веб-приложение будет запущенно на сервере. Чтобы пользователь мог использовать веб-приложение, он должен находится в одной сети с веб-сервером.

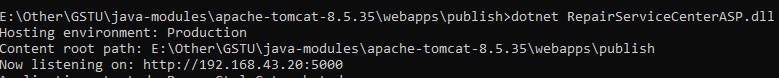


Рисунок 2.1 – Запуск веб-приложения на веб-сервере

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

**3 СТРУКТУРА ПРИЛОЖЕНИЯ**

**3.1 Описание общей структуры веб-приложения**

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

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

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

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

**3.2** **Описание классов для доступа к данным**

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

Классы *ZodiacSign, Customer, Employee, Relationship, Nationality* и *Service* описывают таблицы *ZodiacSigns, Customers, Employees, Relationships, Nationalitys* и *Services* соответственно. Код данных классов представлен в приложении Б.

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

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

## **3.3 Описание контроллеров**

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

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

Для работы с созданными моделями разработаны следующие контроллеры:

– *HomeController* – отвечает за вывод начальной страницы;

– *ZodiacSignsController –* отвечает за работу с таблицей *ZodiacSign;*

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

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

*– RelationshipsController –* отвечает за работу с таблицей *Relationships;*

*– NationalitysController –* отвечает за работу с таблицей *Nationalitys;*

*– 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:

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

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

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

– *Nationalities–* содержит представления для работы с данными о национальностях, 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, 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 услуг и учёта данных о внутренней экосистемы предприятий.

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

*Web*-приложение предназначено для управления и учёта.

Основные функции приложения:

– просмотр полной информации обо всех клиентах;

– информация о предоставляемых услуг;

– отображение информации о списках услуг, стоимость которых составляет менее 30 рублей или более 100 рублей;

– отображение стоимости оказанных услуг по каждому сотруднику;

– добавление, просмотр и редактирования данных о знаках зодиака;

– добавление, просмотр и редактирования данных о отношениях;

– фильтрация, сортировка и выборка данных о сотрудниках по заданным критериям;

– добавление, просмотр и редактирования данных о национальностях;

– добавление, просмотр и редактирования данных о 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 сотрудниках, требуется выбрать вкладку «*Employees*» в вверху окна браузера, рисунок 4.1.

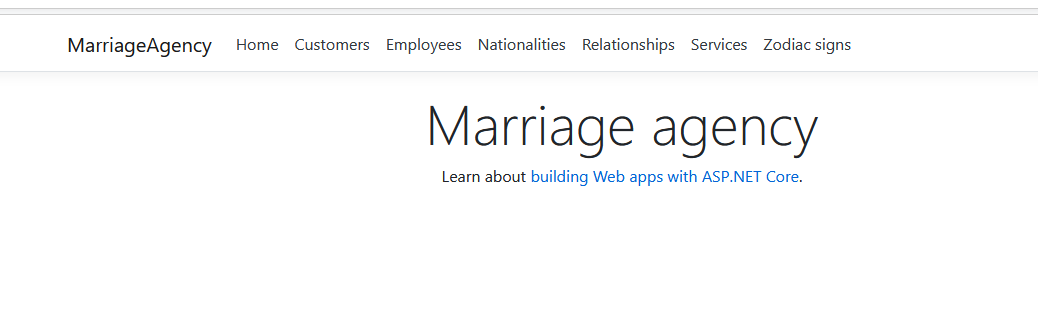


Рисунок 4.1 – Выбор сервиса

Затем загрузится новое окно со списком сотрудников, рисунок 4.2.

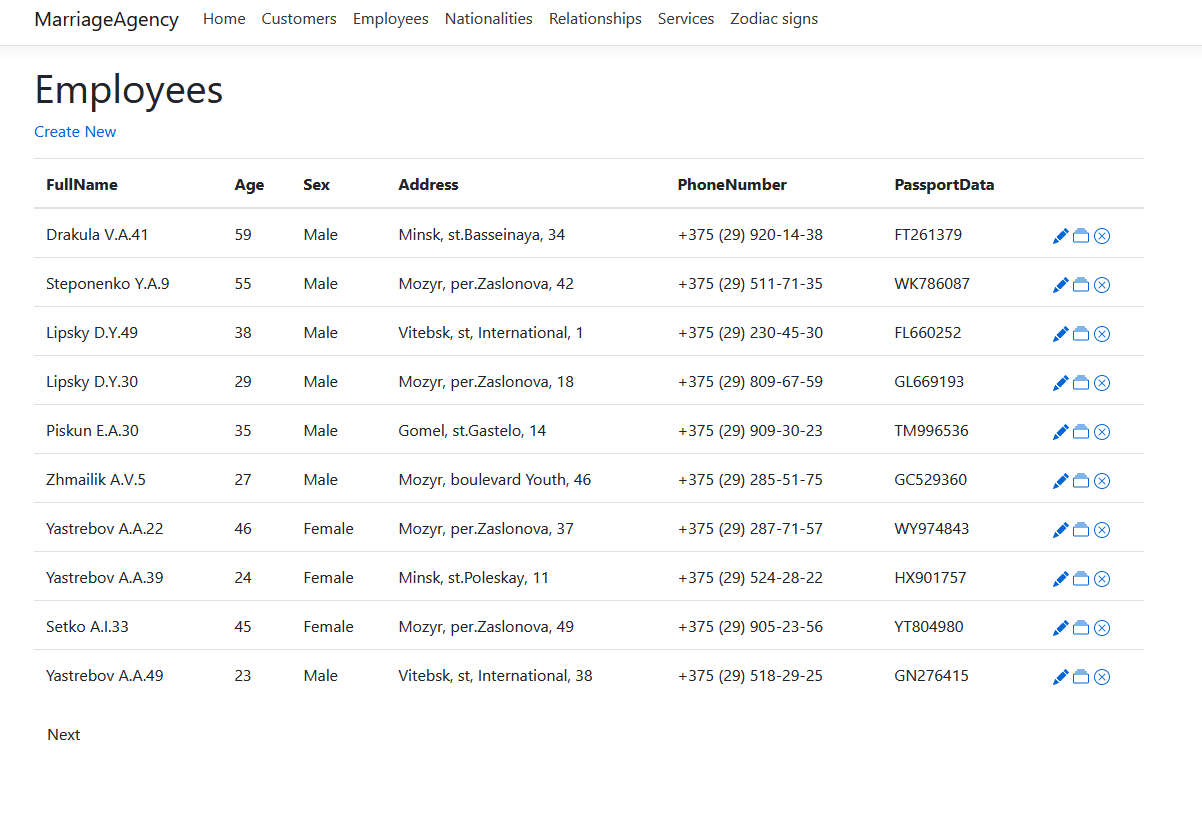


Рисунок 4.2 – Список сотрудников

Чтобы добавить нового сотрудника, нужно нажать на ссылку, выделенную голубым цветом, под названием вкладки (в данном случаи – «*Employees*»), затем загрузится новая страница, рисунок 4.3 с формами для создания нового сотрудника (все поля формы имеют проверку на корректность введённых данных). Чтобы окончательно оформить заказ, нужно нажать на кнопку «*Create*».

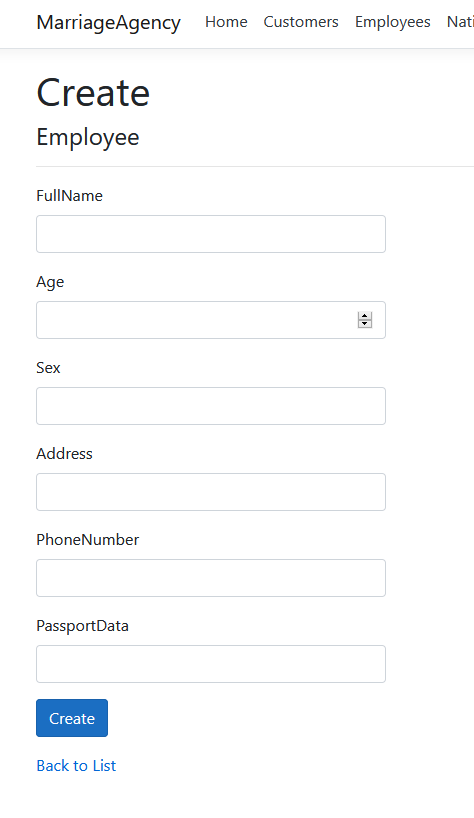


Рисунок 4.3 – Форма для добавления нового клиента

После добавления сотрудника пользователь будет перенаправлен на страницу с выборкой.

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

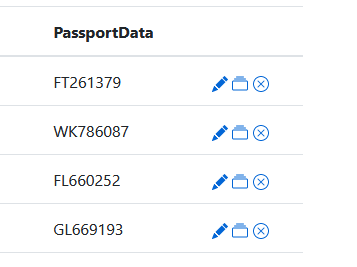


Рисунок 4.4 – Пункты для редактирования, удаления и просмотра данных о клиенте

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.

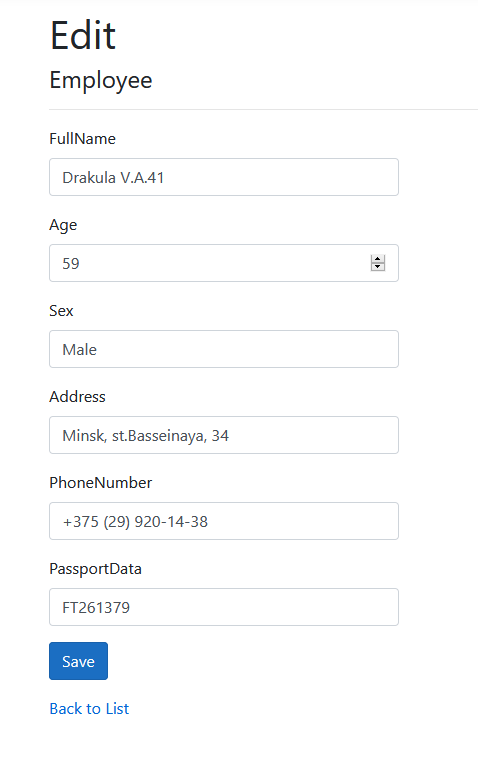


Рисунок 4.5 – Окно для редактирования данных

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

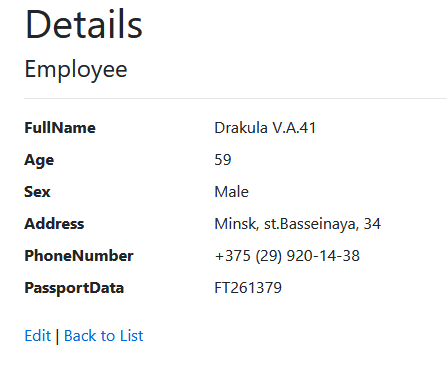


Рисунок 4.6 – Подробности о сотруднике

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

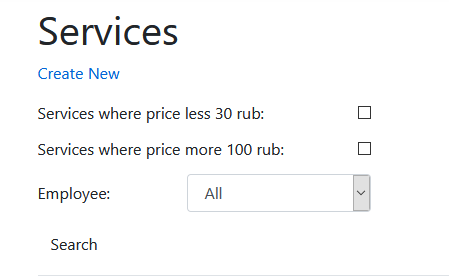


Рисунок 4.7 – Формы для фильтрации данных

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 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 предприятиям автоматизировать свой производственный процесс и учёт предоставляемых услуг.

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=MarriageAgency;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

Customer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Models

{

public class Customer

{

public int Id { get; set; }

public string FullName { get; set; }

public string Sex { get; set; }

public DateTime BirthdayDate { get; set; }

public int Age { get; set; }

public int Height { get; set; }

public int Weight { get; set; }

public int ChildrenCount { get; set; }

public string MaritalStatus { get; set; }

public string BadHadits { get; set; }

public string Hobby { get; set; }

public string Description { get; set; }

public int ZodiacSignId { get; set; }

public int RelationshipId { get; set; }

public int NationalityId { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string PassportData { get; set; }

public ZodiacSign ZodiacSign { get; set; }

public Relationship Relationship { get; set; }

public Nationality Nationality { get; set; }

public ICollection<Service> Services { get; set; }

public Customer()

{

Services = new List<Service>();

}

}

}

Employee.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.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 PhoneNumber { get; set; }

public string PassportData { get; set; }

public ICollection<Service> Services { get; set; }

public Employee()

{

Services = new List<Service>();

}

}

}

Nationality.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Models

{

public class Nationality

{

public int Id { get; set; }

public string Name { get; set; }

public string Note { get; set; }

public ICollection<Customer> Customers { get; set; }

public Nationality()

{

Customers = new List<Customer>();

}

}

}

Relationship.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Models

{

public class Relationship

{

public int Id { get; set; }

public string Name { get; set; }

public string Desctiption { get; set; }

public ICollection<Customer> Customers { get; set; }

public Relationship()

{

Customers = new List<Customer>();

}

}

}

Service.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Models

{

public class Service

{

public int Id { get; set; }

public DateTime Date { get; set; }

public double Price { get; set; }

public int CustomerId { get; set; }

public int EmployeeId { get; set; }

public Customer Customer { get; set; }

public Employee Employee { get; set; }

}

}

ZodiacSign.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Models

{

public class ZodiacSign

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public ICollection<Customer> Customers { get; set; }

public ZodiacSign()

{

Customers = new List<Customer>();

}

}

}

MarriageAgencyContext.cs

using MarriageAgency.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Data

{

public class MarriageAgencyContext : DbContext

{

public MarriageAgencyContext(DbContextOptions options) : base(options)

{

}

public DbSet<Customer> Customers { get; set; }

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

public DbSet<Nationality> Nationalities { get; set; }

public DbSet<Relationship> Relationships { get; set; }

public DbSet<Service> Services { get; set; }

public DbSet<ZodiacSign> ZodiacSigns { get; set; }

}

}

DbInitializer.cs

using MarriageAgency.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Data

{

public class DbInitializer

{

private static Random randObj = new Random(1);

public static void Initialize(MarriageAgencyContext db)

{

db.Database.EnsureCreated();

int employeeCount = 50;

int clientCount = 50;

int serviceCount = 100;

ZodiacSignGenerate(db);

RelationshipGenerator(db);

NationalityGenerate(db);

EmployeesGenerate(db, employeeCount);

CustomersGenerate(db, clientCount);

ServicesGenerate(db, serviceCount);

}

public static void ZodiacSignGenerate(MarriageAgencyContext db)

{

if (db.ZodiacSigns.Any())

{

return;

}

db.ZodiacSigns.AddRange(new ZodiacSign[]

{

new ZodiacSign()

{

Name = "Aries",

Description = "Aries loves to be number one," +

" so it’s no surprise that these audacious rams are the first " +

"sign of the zodiac."

},

new ZodiacSign()

{

Name = "Taurus",

Description = "Taurus is an earth sign represented by the bull." +

" Like their celestial spirit animal, Taureans enjoy relaxing in" +

" serene, bucolic environments surrounded by soft sounds, soothing" +

" aromas, and succulent flavorsc."

},

new ZodiacSign()

{

Name = "Gemini",

Description = "Have you ever been so busy that you wished you could " +

"clone yourself just to get everything done? That’s the Gemini experience " +

"in a nutshell."

},

new ZodiacSign()

{

Name = "Cancer",

Description = "Cancer is a cardinal water sign. Represented by the crab, this crustacean seamlessly weaves between the sea and shore representing Cancer’s ability to exist in both emotional and material realms."

},

new ZodiacSign()

{

Name = "Leo",

Description = "Roll out the red carpet because Leo has arrived. Leo is represented by the lion and these spirited fire signs are the kings and queens of the celestial jungle."

},

new ZodiacSign()

{

Name = "Virgo",

Description = "Virgo is an earth sign historically represented by the goddess of wheat and agriculture, an association that speaks to Virgo’s deep-rooted presence in the material world."

},

new ZodiacSign()

{

Name = "Libra",

Description = "Libra is an air sign represented by the scales (interestingly, the only inanimate object of the zodiac), an association that reflects Libra's fixation on balance and harmony."

},

new ZodiacSign()

{

Name = "Scorpio",

Description = "Scorpio is one of the most misunderstood signs of the zodiac. Because of its incredible passion and power, Scorpio is often mistaken for a fire sign. "

},

new ZodiacSign()

{

Name = "Sagittarius",

Description = "Represented by the archer, Sagittarians are always on a quest for knowledge."

},

new ZodiacSign()

{

Name = "Capricorn",

Description = "The last earth sign of the zodiac, Capricorn is represented by the sea goat, a mythological creature with the body of a goat and the tail of a fish."

},

new ZodiacSign()

{

Name = "Aquarius",

Description = "Despite the “aqua” in its name, Aquarius is actually the last air sign of the zodiac. "

},

new ZodiacSign()

{

Name = "Pisces",

Description = "Pisces, a water sign, is the last constellation of the zodiac. It's symbolized by two fish swimming in opposite directions, representing the constant division of Pisces' attention between fantasy and reality. "

},

});

db.SaveChanges();

}

public static void RelationshipGenerator(MarriageAgencyContext db)

{

if (db.Relationships.Any())

{

return;

}

db.Relationships.AddRange(new Relationship[]

{

new Relationship()

{

Name = "Codependent Relationships",

Desctiption = "A codependent relationship means that one (or more likely both) of you are reliant on the other to function."

},

new Relationship()

{

Name = "Independent Relationships",

Desctiption = "The flip side of the codependent relationship is the independent relationship."

},

new Relationship()

{

Name = "Dominant/Submissive Relationships",

Desctiption = "The unhealthy version of our first two entries takes the negative sides of both to an extreme. One person in the relationship exerts total control over the other."

},

new Relationship()

{

Name = "Open Relationships",

Desctiption = "An open relationship is another version of an independent relationship."

},

new Relationship()

{

Name = "Long Distance Relationships ",

Desctiption = "This one stands out as unique on the list."

},

new Relationship()

{

Name = "Toxic Relationships ",

Desctiption = "The complete opposite of a healthy relationship is a toxic one."

}

});

db.SaveChanges();

}

public static void NationalityGenerate(MarriageAgencyContext db)

{

if (db.Nationalities.Any())

{

return;

}

db.Nationalities.AddRange(new Nationality[]

{

new Nationality()

{

Name = "Ukrainians",

Note = "Love lard"

},

new Nationality()

{

Name = "Belarusians",

Note = "Love potatoes"

},

new Nationality()

{

Name = "Moldovans",

Note = "Roman-speaking people in Southeast Europe."

},

new Nationality()

{

Name = "Jews",

Note = "People of semitic origin."

},

new Nationality()

{

Name = "Gypsies",

Note = "One of the largest ethnic minorities in Europe"

},

});

db.SaveChanges();

}

public static void EmployeesGenerate(MarriageAgencyContext db, int count)

{

if (db.Customers.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.", "Lipsky D.Y." };

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[] sexVoc = { "Male", "Female" };

for (int i = 0; i < count; i++)

{

var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(count);

var age = randObj.Next(18, 60);

var sex = sexVoc[randObj.Next(sexVoc.GetLength(0))];

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 passportData = new string(Enumerable.Repeat(chars, 2)

.Select(s => s[randObj.Next(s.Length)]).ToArray()).ToUpper() + randObj.Next(100000, 999999);

db.Employees.Add(new Employee()

{

FullName = fullName,

Age = age,

Sex = sex,

Address = address,

PhoneNumber = phoneNumber,

PassportData = passportData

});

}

db.SaveChanges();

}

public static void CustomersGenerate(MarriageAgencyContext db, int count)

{

if (db.Customers.Any())

{

return;

}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";

string upperChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

var zodiacSignCount = db.ZodiacSigns.Count();

var relationshipCount = db.Relationships.Count();

var nationalityCount = db.Nationalities.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 = {"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[] sexVoc = { "Male", "Female" };

string[] maritalStatusVoc = { "Married", "Single" };

string[] badHabitsVoc = { "Smoking", "Binge" };

for (int i = 0; i < count; i++)

{

var fullName = fullNamesVoc[randObj.Next(fullNamesVoc.GetLength(0))] + randObj.Next(count);

var sex = sexVoc[randObj.Next(sexVoc.GetLength(0))];

var birthdayDate = DateTime.Now.AddDays(-randObj.Next(5000));

var age = (DateTime.Now - birthdayDate).Days;

var height = randObj.Next(160, 210);

var weight = randObj.Next(50, 120);

var childrenCount = randObj.Next(3);

var martialStatus = maritalStatusVoc[randObj.Next(maritalStatusVoc.GetLength(0))];

var badHabits = badHabitsVoc[randObj.Next(badHabitsVoc.GetLength(0))];

var hobby = GetRandStr(chars, 15);

var description = GetRandStr(chars, 25);

var zodicaSignId = randObj.Next(1, zodiacSignCount + 1);

var relationshipId = randObj.Next(1, relationshipCount + 1);

var nationalityId = randObj.Next(1, nationalityCount + 1);

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 passportData = new string(Enumerable.Repeat(upperChars, 2)

.Select(s => s[randObj.Next(s.Length)]).ToArray()).ToUpper() + randObj.Next(100000, 999999);

db.Customers.Add(new Customer()

{

FullName = fullName,

Sex = sex,

BirthdayDate = birthdayDate,

Age = age,

Height = height,

Weight = weight,

ChildrenCount = childrenCount,

MaritalStatus = martialStatus,

BadHadits = badHabits,

Hobby = hobby,

Description = description,

ZodiacSignId = zodicaSignId,

RelationshipId = relationshipId,

NationalityId = nationalityId,

Address = address,

PhoneNumber = phoneNumber,

PassportData = passportData

});

}

db.SaveChanges();

}

public static void ServicesGenerate(MarriageAgencyContext db, int count)

{

if (db.Services.Any())

{

return;

}

var customersCount = db.Customers.Count();

var employeesCount = db.Employees.Count();

for (int i = 0; i < count; i++)

{

var date = DateTime.Now.AddDays(-randObj.Next(500));

var price = randObj.Next(1, 250);

var customerId = randObj.Next(1, customersCount + 1);

var employeeId = randObj.Next(1, employeesCount + 1);

db.Services.Add(new Service()

{

Date = date,

Price = price,

CustomerId = customerId,

EmployeeId = employeeId

});

}

db.SaveChanges();

}

private static string GetRandStr(string chars, int maxChar)

{

return new string(Enumerable.Repeat(chars, maxChar)

.Select(s => s[randObj.Next(s.Length)]).ToArray());

}

}

}

ErrorViewModel.cs

using System;

namespace RepairServiceCenterASP.Models

{

public class ErrorViewModel

{

public string RequestId { get; set; }

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

}

}

ZodiacSignsController.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 MarriageAgency.Data;

using MarriageAgency.Models;

namespace MarriageAgency.Controllers

{

public class ZodiacSignsController : Controller

{

private readonly MarriageAgencyContext \_context;

public ZodiacSignsController(MarriageAgencyContext context)

{

\_context = context;

}

// GET: ZodiacSigns

public async Task<IActionResult> Index()

{

return View(await \_context.ZodiacSigns.ToListAsync());

}

// GET: ZodiacSigns/Details/5

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

{

if (id == null)

{

return NotFound();

}

var zodiacSign = await \_context.ZodiacSigns

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

if (zodiacSign == null)

{

return NotFound();

}

return View(zodiacSign);

}

// GET: ZodiacSigns/Create

public IActionResult Create()

{

return View();

}

// POST: ZodiacSigns/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")] ZodiacSign zodiacSign)

{

if (ModelState.IsValid)

{

\_context.Add(zodiacSign);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(zodiacSign);

}

// GET: ZodiacSigns/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var zodiacSign = await \_context.ZodiacSigns.FindAsync(id);

if (zodiacSign == null)

{

return NotFound();

}

return View(zodiacSign);

}

// POST: ZodiacSigns/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")] ZodiacSign zodiacSign)

{

if (id != zodiacSign.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(zodiacSign);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ZodiacSignExists(zodiacSign.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(zodiacSign);

}

// GET: ZodiacSigns/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var zodiacSign = await \_context.ZodiacSigns

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

if (zodiacSign == null)

{

return NotFound();

}

return View(zodiacSign);

}

// POST: ZodiacSigns/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var zodiacSign = await \_context.ZodiacSigns.FindAsync(id);

\_context.ZodiacSigns.Remove(zodiacSign);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ZodiacSignExists(int id)

{

return \_context.ZodiacSigns.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 MarriageAgency.Data;

using MarriageAgency.Models;

using MarriageAgency.ViewModels;

using MarriageAgency.ViewModels.Filters;

namespace MarriageAgency.Controllers

{

public class ServicesController : Controller

{

private readonly MarriageAgencyContext \_context;

public ServicesController(MarriageAgencyContext context)

{

\_context = context;

}

// GET: Services

public async Task<IActionResult> Index(int? employeeId, bool less30Rub = false, bool more100Rub = false, int page = 1)

{

var pageSize = 10;

var itemCount = \_context.Employees.Count();

IQueryable<Service> services = \_context.Services;

if (employeeId.HasValue && employeeId.Value != 0)

{

services = services.Where(s => s.EmployeeId == employeeId);

}

if (less30Rub)

{

services = services.Where(s => s.Price < 30);

}

if (more100Rub)

{

services = services.Where(s => s.Price > 100);

}

services = services.Skip((page - 1) \* pageSize)

.Take(pageSize)

.Include(s => s.Customer)

.Include(s => s.Employee);

return View(new ServiceViewModel()

{

Services = await services.ToListAsync(),

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

ServiceFilter = new ServiceFilter(less30Rub, more100Rub, employeeId, await \_context.Employees.ToListAsync())

});

}

// GET: Services/Details/5

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

{

if (id == null)

{

return NotFound();

}

var service = await \_context.Services

.Include(s => s.Customer)

.Include(s => s.Employee)

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

if (service == null)

{

return NotFound();

}

return View(service);

}

// GET: Services/Create

public IActionResult Create()

{

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

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

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,Date,Price,CustomerId,EmployeeId")] Service service)

{

if (ModelState.IsValid)

{

\_context.Add(service);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["CustomerId"] = new SelectList(\_context.Customers, "Id", "FullName", service.CustomerId);

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

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();

}

ViewData["CustomerId"] = new SelectList(\_context.Customers, "Id", "FullName", service.CustomerId);

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

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,Date,Price,CustomerId,EmployeeId")] 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));

}

ViewData["CustomerId"] = new SelectList(\_context.Customers, "Id", "FullName", service.CustomerId);

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

return View(service);

}

// GET: Services/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var service = await \_context.Services

.Include(s => s.Customer)

.Include(s => s.Employee)

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

if (service == null)

{

return NotFound();

}

return View(service);

}

// POST: Services/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> 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);

}

}

}

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 MarriageAgency.Data;

using MarriageAgency.Models;

using MarriageAgency.ViewModels;

namespace MarriageAgency.Controllers

{

public class EmployeesController : Controller

{

private readonly MarriageAgencyContext \_context;

public EmployeesController(MarriageAgencyContext context)

{

\_context = context;

}

// GET: Employees

public async Task<IActionResult> Index(int page = 1)

{

var pageSize = 10;

var itemCount = \_context.Employees.Count();

IQueryable<Employee> employees = \_context.Employees

.Skip((page - 1) \* pageSize)

.Take(pageSize);

return View(new EmployeeViewModel()

{

Employees = await employees.ToListAsync(),

PageViewModel = new PageViewModel(itemCount, page, pageSize)

});

}

// GET: Employees/Details/5

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

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees

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

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// GET: Employees/Create

public IActionResult Create()

{

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,PhoneNumber,PassportData")] Employee employee)

{

if (ModelState.IsValid)

{

\_context.Add(employee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

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();

}

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,PhoneNumber,PassportData")] 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));

}

return View(employee);

}

// GET: Employees/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees

.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);

}

}

}

RelationshipsController.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 MarriageAgency.Data;

using MarriageAgency.Models;

namespace MarriageAgency.Controllers

{

public class RelationshipsController : Controller

{

private readonly MarriageAgencyContext \_context;

public RelationshipsController(MarriageAgencyContext context)

{

\_context = context;

}

// GET: Relationships

public async Task<IActionResult> Index()

{

return View(await \_context.Relationships.ToListAsync());

}

// GET: Relationships/Details/5

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

{

if (id == null)

{

return NotFound();

}

var relationship = await \_context.Relationships

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

if (relationship == null)

{

return NotFound();

}

return View(relationship);

}

// GET: Relationships/Create

public IActionResult Create()

{

return View();

}

// POST: Relationships/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,Desctiption")] Relationship relationship)

{

if (ModelState.IsValid)

{

\_context.Add(relationship);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(relationship);

}

// GET: Relationships/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var relationship = await \_context.Relationships.FindAsync(id);

if (relationship == null)

{

return NotFound();

}

return View(relationship);

}

// POST: Relationships/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,Desctiption")] Relationship relationship)

{

if (id != relationship.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(relationship);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!RelationshipExists(relationship.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(relationship);

}

// GET: Relationships/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var relationship = await \_context.Relationships

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

if (relationship == null)

{

return NotFound();

}

return View(relationship);

}

// POST: Relationships/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var relationship = await \_context.Relationships.FindAsync(id);

\_context.Relationships.Remove(relationship);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool RelationshipExists(int id)

{

return \_context.Relationships.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 });

}

}

}

NationalitiesController.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 MarriageAgency.Data;

using MarriageAgency.Models;

namespace MarriageAgency.Controllers

{

public class NationalitiesController : Controller

{

private readonly MarriageAgencyContext \_context;

public NationalitiesController(MarriageAgencyContext context)

{

\_context = context;

}

// GET: Nationalities

public async Task<IActionResult> Index()

{

return View(await \_context.Nationalities.ToListAsync());

}

// GET: Nationalities/Details/5

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

{

if (id == null)

{

return NotFound();

}

var nationality = await \_context.Nationalities

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

if (nationality == null)

{

return NotFound();

}

return View(nationality);

}

// GET: Nationalities/Create

public IActionResult Create()

{

return View();

}

// POST: Nationalities/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,Note")] Nationality nationality)

{

if (ModelState.IsValid)

{

\_context.Add(nationality);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(nationality);

}

// GET: Nationalities/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var nationality = await \_context.Nationalities.FindAsync(id);

if (nationality == null)

{

return NotFound();

}

return View(nationality);

}

// POST: Nationalities/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,Note")] Nationality nationality)

{

if (id != nationality.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(nationality);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!NationalityExists(nationality.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(nationality);

}

// GET: Nationalities/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var nationality = await \_context.Nationalities

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

if (nationality == null)

{

return NotFound();

}

return View(nationality);

}

// POST: Nationalities/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var nationality = await \_context.Nationalities.FindAsync(id);

\_context.Nationalities.Remove(nationality);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool NationalityExists(int id)

{

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

}

}

}

HomeController.cs

using MarriageAgency.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.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 });

}

}

}

DbInitializerExtensions.cs

using Microsoft.AspNetCore.Builder;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Middleware

{

public static class DbInitializerExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

DbInitializerMiddleware.cs

using MarriageAgency.Data;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next)

{

// инициализация базы данных

\_next = next;

}

public Task Invoke(HttpContext context, IServiceProvider serviceProvider, MarriageAgencyContext 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);

}

}

}

CustomerViewModel.cs

using MarriageAgency.Models;

using MarriageAgency.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.ViewModels

{

public class CustomerViewModel

{

public ICollection<Customer> Customers { get; set; }

public PageViewModel PageViewModel { get; set; }

public CustomerFilter CustomerFilter { get; set; }

}

}

EmployeeViewModel.cs

using MarriageAgency.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.ViewModels

{

public class EmployeeViewModel

{

public ICollection<Employee> Employees { 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;

}

}

}

}

ServiceViewModel.cs

using MarriageAgency.Models;

using MarriageAgency.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.ViewModels

{

public class ServiceViewModel

{

public ICollection<Service> Services { get; set; }

public PageViewModel PageViewModel { get; set; }

public ServiceFilter ServiceFilter { get; set; }

}

}

CustomerFilter.cs

using MarriageAgency.Models;

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.ViewModels.Filters

{

public class CustomerFilter

{

public int? ZodiacSignId { get; set; }

public SelectList ZodiacSigns { get; set; }

public CustomerFilter(int? zodiacSignId, IList<ZodiacSign> zodiacSigns)

{

zodiacSigns.Insert(0, new ZodiacSign()

{

Id = 0,

Name = "All"

});

ZodiacSignId = zodiacSignId;

ZodiacSigns = new SelectList(zodiacSigns, "Id", "Name", zodiacSignId);

}

}

}

ServiceFilter.cs

using MarriageAgency.Models;

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MarriageAgency.ViewModels.Filters

{

public class ServiceFilter

{

public bool Less30Rub { get; set; }

public bool More100Rub { get; set; }

public int? EmployeeId { get; set; }

public SelectList Employees { get; set; }

public ServiceFilter(bool less30Rub, bool more100Rub, int? employeeId, IList<Employee> employees)

{

employees.Insert(0, new Employee()

{

Id = 0,

FullName = "All"

});

Less30Rub = less30Rub;

More100Rub = more100Rub;

EmployeeId = employeeId;

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

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Чертёж структуры web-приложения**