**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Разработка приложений баз данных для информационных систем»

на тему: «**РАЗРАБОТКА *WEB*-ПРИЛОЖЕНИЯ БАЗ ДАННЫХ «ФАБРИКА МЕБЕЛИ**»

Исполнитель: студент гр. ИТП-21

Шеляхин М. С.

Руководитель: преподаватель

Точко В. Н.

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

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

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

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

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2020

Установа адукацыі «Гомельскі дзяржаўны

тэхнічны ўніверсітэт імя П.В. Сухога»

Факультэт аўтаматызаваных і інфармацыйных сістэм

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РЭЦЭНЗІЯ**

**на курсавы праект (работу)**

па дысцыпліне \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

выканана студэнтам \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

групы \_\_\_\_\_\_\_\_\_\_

I. Пералік заўваг па тэксту курсавога праекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

II. Агульная характарыстыка работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc58458608)

[1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ 8](#_Toc58458609)

[1.1 Логическая структура базы данных 8](#_Toc58458610)

[2.1 Физическая структура спроектированной базы данных 9](#_Toc58458611)

[2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 11](#_Toc58458612)

[2.1 Список необходимого аппаратного и программного обеспечения 11](#_Toc58458613)

[2.2 Выбор среды программирования 11](#_Toc58458614)

[2.3 Выбор источника данных 12](#_Toc58458615)

[2.4 Выбор технологии доступа к источнику данных 13](#_Toc58458616)

[2.6 Выбор технологии создания *web*-приложения 14](#_Toc58458617)

[3 СТРУКТУРА ПРОГРАММЫ 16](#_Toc58458618)

[3.1 Структура уровня моделей 16](#_Toc58458619)

[3.2 Структура уровня контроллеров 18](#_Toc58458620)

[3.3 Структура уровня представлений 19](#_Toc58458621)

[3.4 Вспомогательные файлы и классы 23](#_Toc58458622)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 24](#_Toc58458623)

[4.1 Общие сведения 24](#_Toc58458624)

[4.2 Требования к запуску приложения 24](#_Toc58458625)

[4.3 Запуск приложения 24](#_Toc58458626)

[4.4 Непредвиденные ситуации 24](#_Toc58458627)

[4.5 Описание операции по обработки данных 25](#_Toc58458628)

[5 РУКОВОДСТВО ПРОГРАММИСТА 32](#_Toc58458629)

[5.1 Назначение и условия применения программы 32](#_Toc58458630)

[5.2 Характеристики программы 32](#_Toc58458631)

[5.3 Сопровождение программного комплекса 32](#_Toc58458632)

[5.4 Обращение к программе 33](#_Toc58458633)

[5.5 Входные и выходные данные 33](#_Toc58458634)

[5.6 Сообщение в ходе работы приложения 33](#_Toc58458635)

[ЗАКЛЮЧЕНИЕ 34](#_Toc58458636)

[Список используемых источников 35](#_Toc58458637)

[ПРИЛОЖЕНИЕ А 36](#_Toc58458638)

[ПРИЛОЖЕНИЕ Б 41](#_Toc58458639)

ВВЕДЕНИЕ

В XXI веке для существования и успешного развития любого предприятия в условиях конкуренции необходимо выполнение следующих условий:

– активное взаимодействие с покупателем в режиме реального времени;

– эффективный контроль потока данных, поступающих от покупателя предприятию и обратно;

– постоянное продвижение предприятия с помощью рекламы.

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 будет создано *web*-приложение, так как оно дает пользователям возможность вводить, получать и манипулировать данными с помощью взаимодействия. Данное взаимодействие будет характеризовываться возможностью создания заказа предприятию, получения актуальной информации о предприятии и его ресурсах и т.д.

Для фабрики мебели необходимо создать приложение, позволяющее создавать и управлять заказами, изделиями, накладными, работниками и клиентами, иметь авторизацию и управление ролями, а также иметь возможность вывода информации о сотрудниках, поставках и клиентах авторизованным пользователям.

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*.

1. ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ
   1. Логическая структура базы данных

В информационной системе присутствуют следующие сущности:

– «Клиент»;

– «Работник»;

– «Мебель»;

– «Заказ»;

– «Накладная».

Сущность «Мебель» представляет из себя изделие, которое можно заказать на фабрике. Данный компонент будет иметь следующие атрибуты:

– уникальный *id*;

– наименование;

– описание;

– вид материала;

– цена;

– количество.

Для хранения списка изделий необходимо создать таблицу в БД, в которой должнен храниться список сущностей со значениями атрибутов для каждой записи.

Сущность «Работник» представляет из себя работника предприятия. Данный компонент будет иметь следующие атрибуты:

– уникальный *id*;

– ФИО работника;

– должность;

– образование.

Для хранения списка работников также необходимо создать таблицу в БД, в которой должнен храниться список сущностей со значениями атрибутов для каждой записи.

Сущность «Клиент» представляет из себя заказчика, являющегося другим предприятием с представителем и основной информацией. Данная сущность будет иметь следующие атрибуты:

– уникальный *id*;

– название;

– ФИО представителя;

– номер;

– адрес.

Так как представитель является работником другого предприятия, а данные о нем используются только в качестве описательной характеристики, нет необходимости создавать связь с таблицей работников.

Сущность «Заказ» представляет из себя заказ партии мебели, совершенный пользователем и обслуживаемый работником. Заказ должен иметь следующие атрибуты:

– уникальный *id*;

– данные о клиенте;

– данные об изделии;

– количество единиц изделия на складе;

– цена;

– специальная скидка для клиента;

– отметка о выполнении заказа;

– данные о работнике, обслуживающим заказ.

Так как в заказе присутствуют данные о клиенте, изделии и работнике, необходимо организовать связи между таблицами с клиентами, изделиями и работниками, а в качестве атрибутов создать поля с уникальными *id* записей.

Сущность «Накладная» представляет из себя накладную о поступлении материалов для изделий на склад. Накладная должна иметь следующие атрибуты:

– уникальный *id*;

– номер поставщика;

– наименование поставщика;

– дата поступления материала на склад;

– вид материала;

– цена;

– вес;

– данные об изделии, для которого получены материалы;

– данные о работнике, который обеспечивал поступление.

Так как в накладной присутствуют данные об изделии и работнике, необходимо организовать связи между таблицами с изделиями и работниками, а в качестве атрибутов создать поля с уникальными *id* записей.

* 1. Физическая структура спроектированной базы данных

Используя логическую структуру БД, можно воссоздать ее в СУБД *MS SQL Server*.

Спроектированная БД состоит из следующих таблиц:

– «*Clients*» – хранит информацию о клиентах предприятия;

– «*Employees*» – хранит информацию о работниках предприятия;

– «*Furniture*» – хранит информацию об изделиях;

– «*Orders*» – хранит информацию о заказах;

– «*Waybills*» – хранит информация о накладных;

Каждый внешний и первичный ключ представлен числом формата *int*. Номера клиентов и поставщиков представлены числом формата *int* Названия представлены строкой формата *nvarchar*, которая занимает меньший объем памяти, если длина строки меньше заданной:

– названия компаний клиентов – *nvarchar*(25);

* названия изделий – *nvarchar*(50);

– названия поставщиков – *nvarchar*(100).

Цены накладных и заказов представлены данными формата *decimal*. Поле веса представлено в виде числа формата *float*. ФИО работников и представителей представлены строкой формата *nvarchar*(100). Поле с адресом клиента представлено в виде строки формата *nvarchar(40)*. Поле с должностью работника представлено в формате *nvarchar*(50), а его образование – *nvarchar*(200). Описание мебели представлено в виде строки формата *nvarchar*(200), вид материала – *nvarchar*(60). Количество изделий представлено в виде числа *int*. Статус выполнения заказа из-за особенностей *СУБД MS SQL* представлено в виде числа *int*. Скидка для клиента представлена в формате *int*. Дата поступления материалов представлена форматом *datetime*.

Для минимализации возможности возникновения ошибки при операциях с данными всем полям запрещено хранить *null* значения.

Нормализации данных подвергаются все таблицы. Вторая нормальная форма предполагает, что каждый столбец, не являющийся ключом, должен зависеть от первичного ключа. Так как данные о работниках, клиентах и изделиях не зависят от заказа, то данные о работниках были вынесены в отдельную таблицу «*Employees*», данные об изделиях – в таблицу «*Furniture*», данные о клиентах – «*Clients*», а таблица «*Orders*» содержит внешние ключи записей, соответствующие обслуживающему данный заказ работнику, заказанному изделию и клиенту, сделавшему заказ. Так как данные об изделиях и работниках не зависят от накладной, то данные о них были вынесены в отдельные таблицы «*Employees*» и «*Furniture*», а таблица «*Waybills*» содержит внешние ключи записей, соответствующие принимавшему материалы работнику и изделию, для которого предназначены материалы.

Схема созданной базы данных представлена на рисунке 2.1.

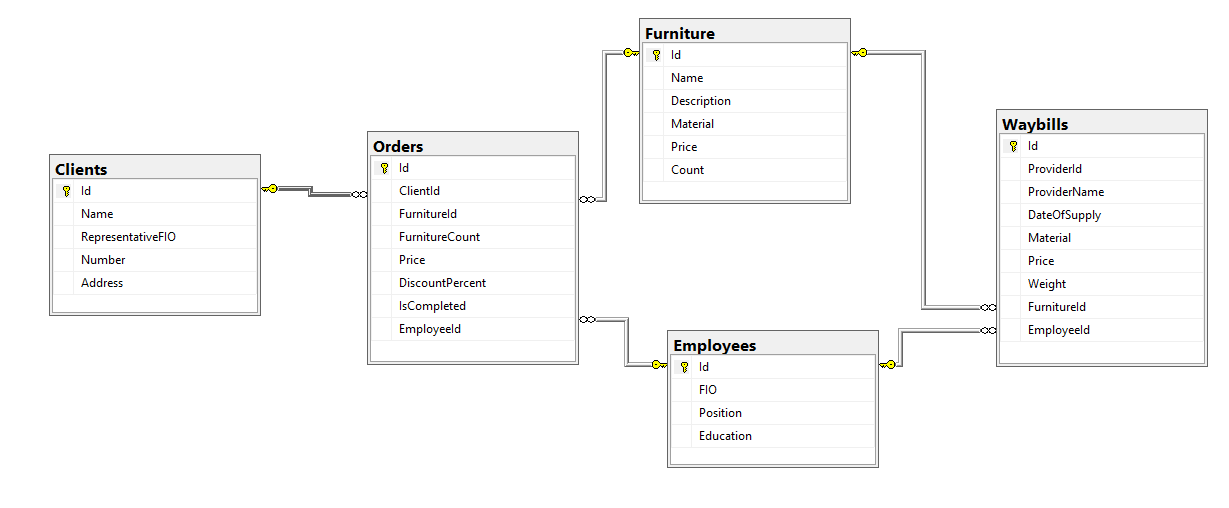


Рисунок 2.1 – Схема базы данных

1. АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## **2.1 Список необходимого аппаратного и программного обеспечения**

Для запуска и работы с приложением необходимо следующее:

– ПК с операционной системой *Windows* версии 7 или выше;

* среда выполнения, поддерживающая язык программирования *C#*;
* источник данных (база данных заданной предметной области);
* технология доступа к источнику данных;
* технология разработки приложений *ASP.NET Core*;
* браузер.

Программное обеспечение должно быть ориентировано на работу с ЯП *C#*. Аппаратное обеспечение должно быть совместимо с необходимыми программами и технологиями, а также поддерживать последние версии популярных браузеров.

2.2 Выбор среды программирования

Для правильного написания программного кода, а также тестирования, автоматизации и запуска необходимо использовать среду программирования, которая кроме редактора кода обладает компилятором, компоновщиком, отладчиком и т.д.

Среди наиболее популярных сред можно выделить:

– *Microsoft Visual Studio*;

– *PyCharm*;

– *NetBeans*;

– *Xcode*;

– *IntelliJ IDEA*;

– *Eclipse* и др.

Каждая из сред, не смотря на поддержку многих языков, в начале разрабатывался для одного определенного языка. Поэтому в качестве критериев выбора среды выступают не только ее преимущества, но и язык, используемый при написании программы.

Для ЯП *C#* выбирается СП (среда программирования) *Microsoft Visual Studio.* Как и язык, СП разработана компанией *Microsoft*, но при этом СП поддерживает множество других языков программирования, а благодаря общеязыковой среде можно создавать программу из модулей, написанных на разных языках программирования. Также *Visual Studio* имеет огромную библиотеку расширений, что позволяет создавать приложение любой спецификации, технологию *IntelliSense*, которая работает как система автодополнения и помогает не только сократить время написания кода, но и позволяет соблюсти синтаксис, вывод списка ошибок, который упрощает отладку, и многое другое.

*Microsoft Visual Studio* – среда разработки компании *Microsoft*, включающая интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. С помощью данной среды можно разработать как консольное приложение, так и графический интерфейс, веб-сайты и др.

Среди возможностей *Visual Studio* можно выделить:

– редактор кода, который поддерживает подсветку синтаксиса, вставку фрагментов кода, отображение структуры и связанных функций;

– *IntelliSense* - автозавершение кода по мере ввода;

– встроенный отладчик, который используется для поиска и исправления ошибок в исходном коде, в том числе на низком аппаратном уровне;

– дизайнер форм – помогает спроектировать внешний вид будущего приложения и работу каждого элемента интерфейса;

– комплекс инструментов для автоматизации тестирования приложений в части проверки работы интерфейсов, модульного и нагрузочного тестирования.

– поддержка групповой работы;

– система управления версиями, которую можно реализовать с помощью *Team Foundation* или *Git*.

2.3 Выбор источника данных

Для хранения большого объема данных и использования их в программе необходимо подобрать правильный источник данных и организовать эффективный доступ к нему. В качестве источника данных предпочтительно использовать СУБД (систему управления базами данных).

В данный момент наиболее предпочтительными являются СУБД:

– *Oracle*;

– *MySQL*;

– *MS SQL Server*;

– *PostgreSQL*.

*PostgreSQL* и *MySQL* являются бесплатными, но при этом эти СУБД не отличаются низкой скоростью при выполнении специфических операций, а *Oracle* требуются огромные ресурсы для работы, что подойдет далеко не для всего аппаратного обеспечения. По сравнению с ними выгодно выделяется *MS SQL Server*.

Главными преимуществами *MS SQL* являются:

– производительность;

– надежность (можно шифровать данные);

– простота.

Также эта СУБД разработано компанией *Microsoft*, что говорит о раскрытии высокого потенциала при работе с платформой *.NET Framework* и *Visual Studio* в частности.

*Microsoft SQL Server* - одна из наиболее мощных систем работы с базами данных в архитектуре "клиент-сервер". Раньше особенностью системы была работа сервера только в операционных системах ряда *Microsoft Windows*, хотя в последних версиях добавлена возможность работы с СУБД и на *Linux*.

В своем составе система имеет средства создания баз данных, работы с информацией баз данных, перенесения данных из других систем и в другие системы, резервного копирования и восстановления данных, развитую систему транзакций, систему репликации данных, реляционную подсистему для анализа, оптимизации и выполнения запросов клиентов, систему безопасности для управления правами доступа к объектам базы данных и пр.

Особенностью *MS SQL* является использование *Transact-SQL* – процедурного расширения языка *SQL*, созданного компанией *Microsoft* и *Sybase*. *SQL* был расширен такими дополнительными возможностями как:

– управляющие операторы;

– локальные и глобальные переменные;

– различные дополнительные функции для обработки строк, дат, математики и т. п.;

– поддержка аутентификации *Microsoft Windows*.

* 1. Выбор технологии доступа к источнику данных

Для получения доступа и управления базами данных из приложения необходимо воспользоваться технологией доступа к источнику данных.

Технологии доступа к данным являются прослойкой между *API* конкретного сервера и приложением пользователя, предоставляя программисту простой унифицированный механизм работы с данными. На сегодняшний день существует множество технологий доступа к данным, таких как *BDE*, *OLE*, *ODBC*, *АDО*, и до сих пор разрабатываются новые, более надежные, удобные в работе и более быстродействующие технологии. Механизмы доступа к базам данных снижают сложность обмена информацией с базами, однако интерпретация результатов их работы также достаточно трудоемка. Поэтому реализованы наборы компонентов, предназначенные для взаимодействия с механизмами обмена. Одним из инструментов для доступа к данных является технология *ADO.NET Entity Framework Core*.

*Entity Framework Core -* это современный преобразователь объектных баз данных для *.NET*. Он поддерживает запросы *LINQ*, отслеживание изменений, обновления и миграции схем. *EF Core* работает со многими базами данных, включая базу данных *SQL* (локальную и *Azure*), *SQLite*, *MySQL*, *PostgreSQL* и *Azure Cosmos DB*.

Изначально с самой первой версии *Entity Framework* поддерживал подход *Database First*, который позволял по готовой базе данных сгенерировать модель *edmx*. Затем эта модель использовалась для подключения к базе данных. Позже был добавлен подход *Model First*. Он позволял создать вручную с помощью визуального редактора модель *edmx*, и по ней создать базу данных. Начиная с 5.0 предпочтительным подходом становится *Code First*. Его суть - сначала пишется код модели на *C#*, а затем по нему генерируется база данных. При этом модель *edmx* уже не используется.

*EF Core* позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне программист оперирует таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который предлагает *Entity Framework*, происходит работа с объектами.

*Entity Framework Core* многое унаследовал от своих предшественников, в частности, *Entity Framework 6*. В тоже время надо понимать, что *EF Core* - это не новая версия по отношению к *EF 6*, а совершенно иная технология, хотя в целом принципы работы у них будут совпадать. Поэтому в рамках *EF Core* используется своя система версий. Текущая версия - 5.0 была выпущена в ноябре 2020 года.

Как технология доступа к данным *Entity Framework Core* может использоваться на различных платформах стека .*NET*. Это и стандартные платформы типа *Windows Forms*, консольные приложения, *WPF*, *UWP* и *ASP.NET Core*. При этом кроссплатформенная природа *EF Core* позволяет задействовать ее не только на ОС *Windows*, но и на *Linux* и *Mac OS X* [2, c.13].

2.6 Выбор технологии создания *web*-приложения

В настоящее время существует несколько способов создания сайтов, реализующих принципиально разные подходы и технологии, с применением разных языков и платформ программирования:

* *Active Server Pages (ASP)* – технология создания веб-приложений от *Microsoft*;
* *Active Server Pages.Net* (*ASP.NET*) – усовершенствованная версия *ASP*, являющаяся составной частью платформы *Microsoft.NET*;
* *Personal Home Page Tools* (*PHP*) – скриптовый язык программирования общего назначения;
* *Practical Extraction and Report Language* (*Perl*) – высокоуровневый интерпретируемый динамический язык программирования общего назначения;
* *Java Server Pages* (*JSP*) и многие другие.

И выбор способа создания зависит от множества факторов, начиная от целевой нагрузки сайта и заканчивая квалификацией разработчиков. Наиболее распространенными сейчас являются две технологии: *ASP.NET* и *PHP*.

*ASP.NET* – это концептуально новая технология *Microsoft*, созданная в рамках идеологии *.NET*. Ключевыми сторонами *.NET* являются масштабируемость, кроссплатформенность, межъязыковое взаимодействие и безопасное программирование. *ASP.NET* — это комплекс технологий в составе *.NET Framework*, позволяющих разработчикам создавать интернет-приложения и веб-сервисы *XML*.

В отличие от традиционных веб-страниц, являющихся сочетанием статического *HTML* и сценариев, *ASP.NET* использует скомпилированные страницы, управляемые событиями. Это позволяет разработчикам создавать интернет-приложения, отличающиеся богатством и функциональностью, обычно ассоциирующимися с приложениями, созданными на таких языках программирования, как *Visual Basic* или *Visual C++*. Однако в отличие от локальных приложений, эти скомпилированные страницы создают информацию, отправляемую на компьютеры или обозреватели клиентов с использованием языков разметки наподобие *HTML* и *XML*. Это позволяет разработчикам создавать приложения с широкой функциональностью, защищая при этом интерфейс пользователя к устройствам и системам под управлением разных операционных систем.

*ASP.NET Core* — свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом. Данная платформа разрабатывается компанией *Microsoft* совместно с сообществом и имеет большую производительность по сравнению с *ASP.NET*. Имеет модульную структуру и совместима с такими операционными системами как *Windows*, *Linux* и *Mac OS*.

Несмотря на то, что это новый фреймворк, построенный на новом веб-стеке, он обладает высокой степенью совместимости концепций с *ASP.NET*. Приложения *ASP.NET Core* поддерживают параллельное управление версиями, при котором разные приложения, работающие на одном компьютере, могут ориентироваться на разные версии *ASP.NET Core*. Это было невозможно в предыдущих версиях *ASP.NET* [3, с. 13].

1. СТРУКТУРА ПРОГРАММЫ
   1. Структура уровня моделей

Для организации работы приложения с данными из базы данных необходимо создать модели представления, отражающие сущности таблиц базы данных, а также реализовать доступ к базе данных.

На основе базы данных необходимо создать классы, отражающие все атрибуты таблиц из БД. Для этого в классы нужно добавить поля с такими же свойствами и названиями.

Соотношение базы данных и классов происходит следующим образом:

– для таблицы «Клиенты» создан класс *Client*, в котором атрибуты из таблицы представлены в виде свойств класса;

– для таблицы «Работники» создан класс *Employee*, в котором атрибуты из таблицы представлены в виде свойств класса;

– для таблицы «Мебель» создан класс *Furniture*, в котором атрибуты из таблицы представлены в виде свойств класса;

– для таблицы «Заказы» создан класс *Order*, в котором атрибуты из таблицы представлены в виде свойств класса;

– для таблицы «Накладные» создан класс Waybill, в котором атрибуты из таблицы представлены в виде свойств класса.

Для огранизации взаимодействия с базой данных *Microsoft SQL Server* используется класс контекста данных *ApplicationContext*, который принимает строку подключения к БД в качестве параметра, имеет наборы записей из таблиц, представленные в виде набора классов. Класс контекста наследует базовый класс DbContext для связи контекста с базой данных. Процесс получения данных из базы данных будет происходить автоматически.

В приложении будет использоваться постраничная навигация данных из таблиц, поэтому необходимо создать класс для хранения данных пагинации. Класс *PageViewModel* содержит свойства количества страниц и номер страницы таблицы, а также логические свойства, показывающие наличие или отсутствие предыдущей и следующей страниц.

Для представления получаемых данных необходимо создать модели представления данных, в которых информация будет храниться в понятном и удобном для пользователя виде.

Классы моделей имеют следующую структуру:

– для представления данных класса *Client* создана модель представления *ClientIndexViewModel*, в котором также присутствуют списки *id* записей и самих клиентов для отображения, объект пагинации *PageViewModel* и свойства фильтров по полям «ФИО представителя» и «Адрес». Также в модели к свойствам из класса *Client* применены аннотации данных;

– для представления данных класса *Employee* создана модель представления *EmployeeIndexViewModel*, в котором также присутствуют списки *id* записей и самих работников для отображения, объект пагинации *PageViewModel* и свойства фильтров по полям «Должность» и «Образование». Также в модели к свойствам из класса *Employee* применены аннотации данных;

– для представления данных класса *Furniture* создана модель представления *FurnitureIndexViewModel*, в котором также присутствуют списки *id* записей и самой мебели для отображения, объект пагинации *PageViewModel* и свойства фильтров по полям «Цена» и «Материал». Также в модели к свойствам из класса *Furniture* применены аннотации данных;

– так как в таблице «Заказы» присутствуют внешние ключи, создается класс представления записей таблицы *OrderViewModel*, в котором свойства внешних ключей записей из таблиц «Работники», «Клиенты» и «Мебель» заменены на свойства названия клиента, названия мебели и ФИО сотрудника. Для представления данных на странице создана модель представления *OrderIndexViewModel*, в котором также присутствуют списки *id* записей и объектов *OrderViewModel* для отображения, объект пагинации *PageViewModel*, списки значений атрибутов из других таблиц для заполнение выпадающих списков и свойства фильтров по полям «Название клиента» и «Название мебели». Также в модели к свойствам из класса *OrderViewModel* применены аннотации данных;

– так как в таблице «Накладные» присутствуют внешние ключи, создается класс представления записей таблицы *WaybillViewModel*, в котором свойства внешних ключей записей из таблиц «Работники» и «Мебель» заменены на свойства названия мебели и ФИО сотрудника. Для представления данных на странице создана модель представления *WaybillIndexViewModel*, в котором также присутствуют списки *id* записей и объектов *WaybillViewModel* для отображения, объект пагинации *PageViewModel*, списки значений атрибутов из других таблиц для заполнение выпадающих списков и свойства фильтров по полям «Название поставщика» и «Название мебели». Также в модели к свойствам из класса *WaybillViewModel* применены аннотации данных.

Так как в программе реализуются возможности аутентификации и авторизации, то необходимо создать следующие классы:

– класс *User*, который наследует класс *IdentityUser* и представляет собой сущность пользователя. Класс содержит свойства «ФИО» и «Возраст» пользователя;

– класс контекста *IdentityContext*, наследующий класс *IdentityDbContext* с сущностью *User* и представляющий собой контекст сущностей пользователей. Данный класс принимает в качестве параметров строку подключения и реализует технологию *Entity Framework*;

– модель представления формы регистрации *RegisterViewModel*, которая содержит свойства для ввода ФИО, возраста, пароля и потверждения пароля, а также поле роли пользователя, которое будет представлено в виде выпадающего списка;

– модель представления формы входа LoginViewModel, которая содержит свойства для ввода электронной почты и пароля, а также логическое значение запоминания пользователя, которое будет представлено в виде флага, и *URL* страницы, с которой был совершен переход на форму входа;

– модель представления формы создания пользователя *CreateUserViewModel*, которая будет использоваться администратором для создания нового пользователя. Модель содержит свойства электронной почты, пароля, ФИО и адреса;

– модель представления формы изменения пользователя *EditUserViewModel*, которая будет использоваться администратором для изменения профиля пользователя. Модель содержит свойства электронной почты, возраста, ФИО, а также *id* изменяемого пользователя.

Для инициализации базы данных путем заполнения БД набором начальных данных необходимо создать статический класс DbInitializer, содержащий метод проверки таблиц на наличие записей и заполнение таблиц записями в случае отсутствия записей.

Так как в *web*-приложении будет использоваться валидация данных, то необходимо создать класс для помощи в отображении ошибок приложения. *ErrorViewModel* содержит свойство *id* запроса и свойство проверки *id* на наличие значения.

Иерархическая схема классов представлена в приложении А.

* 1. **Структура уровня контроллеров**

Для организации обработки запросов и взаимодействия между моделями и представлениями необходимо создать контроллеры. Для каждой логически единой группы страниц необходимо создать отдельный контроллер, наследующий класс *Controller*.

Список контроллеров приложения:

– для группы страниц, связанной с таблицей «Клиенты», создан класс *ClientController*. Данный контроллер содержит методы обработки запросов страниц отображения данных клиентов, формы добавления и изменения, а также метода удаления записи;

– для группы страниц, связанной с таблицей «Работники», создан класс *EmployeeController*. Данный контроллер содержит методы обработки запросов страниц отображения данных работников, формы добавления и изменения, а также метода удаления записи;

– для группы страниц, связанной с таблицей «Мебель», создан класс *FurnitureController*. Данный контроллер содержит методы обработки запросов страниц отображения данных мебели, формы добавления и изменения, а также метода удаления записи;

– для группы страниц, связанной с таблицей «Заказы», создан класс *OrderController*. Данный контроллер содержит методы обработки запросов страниц отображения данных заказов, формы добавления и изменения, а также метод удаления записи и метод создания модели для представления;

– для группы страниц, связанной с таблицей «Накладные», создан класс *WaybillController*. Данный контроллер содержит методы обработки запросов страниц отображения данных накладных, формы добавления и изменения, а также метод удаления записи и метод создания модели для представления;

– для обработки запросов начальной страниц создан класс *HomeController*. Данный контроллер содержит методы обработки запросов и загрузки начальной страницы и страницы ошибки сайта;

– для реализации авторизации и аутентификации создан класс *AccountController*. Данный контроллер содержит свойства менеджера пользователей, ролей и входа, а также методы обработки запросов страниц входа, регистрации и метод выхода из профиля;

– для группы страниц, связанной с управлением пользователями, создан класс *UserManageController*. Данный контроллер содержит методы обработки запросов страниц отображения данных о пользователях, формы добавления и изменения, а также метода удаления записей пользователей. Данный контроллер доступен только администратору.

Иерархическая схема классов представлена в приложении А.

* 1. **Структура уровня представлений**

Для осуществления диалога с пользователем в сети Интернет и отображения данных БД необходимо создать представления. Представления – это страницы, написанные преимущественно на языке HTML с компиляцией страницы в класс. Для каждой модели представления, которая была создана заранее, и для стартовых страниц необходимо создать отдельные представления.

В результате получились следующие представления:

– для начальной страницы и страницы ошибок были созданы представления *Home* и *Error*. Домашняя страница не принимает модель и отображает историю фабрики, а *Error* принимает модель *ErrorViewModel* и выводит информацию об ошибке;

– для панели навигации было создано представление ­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­*\_Layout*, содержащее ссылки на главную страницу и страницы управления данными из таблиц. Также в представление встроены общие для всех стили и представление отображения ссылок на регистрацию, вход и управление пользователями. Данное представление является мастер-страницей и будет связано со всеми остальными страницами;

– для отображения ссылок на регистрацию, вход и управление пользователями было создано представление *\_LoginPartial*, в котором происходит проверка на роль пользователя: если пользователь не авторизован, то происходит вывод ссылок на формы регистрации и входа; если пользователь авторизован, происходит вывод почты пользователя и добавление кнопки выхода из аккаунта; если пользователь является администратором, вместо вывода почты происходит вывод ссылки на страницу управления пользователями;

– для управления данными из таблицы «Клиенты» были созданы представления *Index*, *\_AddClient* и *\_UpdateClient*. Представление *Index* принимает модель *ClientIndexViewModel* и отображает постраничный список данных из таблицы «Клиенты», поля и кнопка формы фильтрации, кнопки постраничного перехода по списку и кнопку вызова формы добавления записи (в том случае, если пользователь является администратором). *Id* записей и заголовки столбцов таблицы являются ссылками для вызова формы обновления и удаления записей и сортировки записей в таблицах;

– *\_AddClient* и *\_UpdateClient* являются частичными представлениями формы добавления, обновления и удаления записей. Эти представления встроены в представление *Index* и имеют разные спозобы вызова: форма *\_AddClient* появляется путем нажатия на кнопку добавления записи, а *\_UpdateClient* – путем нажатия на *id* изменяемой записи. Форма *\_AddClient* содержит поля атрибутов записи *Client* и кнопку добавления записи, при нажатии на которую происходит валидация данных и их добавление в таблицу «Клиенты» в случае положительного результата. Форма *\_UpdateClient* содержит список *id* записей с выбранным *id* изменяемой записи, ссылку на метод удаления записи, а также поля атрибутов записи *Client* и кнопку обновления записи, при нажатии на которую происходит валидация данных и их изменение в случае положительного результата. Если валидация пройдена с отрицательным успехом для любой формы, идет возвращение представления со списком ошибок;

– для управления данными из таблицы «Работники» были созданы представления *Index*, *\_AddEmployee* и *\_UpdateEmployee*. Представление *Index* принимает модель *EmployeeIndexViewModel* и отображает постраничный список данных из таблицы «Работники», поля и кнопка формы фильтрации, кнопки постраничного перехода по списку и кнопку вызова формы добавления записи (в том случае, если пользователь является администратором). *Id* записей и заголовки столбцов таблицы являются ссылками для вызова формы обновления и удаления записей и сортировки записей в таблицах;

– *\_AddEmployee* и *\_UpdateEmployee* являются частичными представлениями формы добавления, обновления и удаления записей. Эти представления встроены в представление *Index* и имеют разные спозобы вызова: форма *\_AddEmployee* появляется путем нажатия на кнопку добавления записи, а *\_UpdateEmployee* – путем нажатия на *id* изменяемой записи. Форма *\_AddEmployee* содержит поля атрибутов записи *Employee* и кнопку добавления записи, при нажатии на которую происходит валидация данных и их добавление в таблицу «Работники» в случае положительного результата. Форма *\_UpdateEmployee* содержит список *id* записей с выбранным *id* изменяемой записи, ссылку на метод удаления записи, а также поля атрибутов записи *Employee* и кнопку обновления записи, при нажатии на которую происходит валидация данных и их изменение в случае положительного результата. Если валидация пройдена с отрицательным успехом для любой формы, идет возвращение представления со списком ошибок;

– для управления данными из таблицы «Мебель» были созданы представления *Index*, *\_AddFurniture* и *\_UpdateFurniture*. Представление *Index* принимает модель *FurnitureIndexViewModel* и отображает постраничный список данных из таблицы «Мебель», поля и кнопка формы фильтрации, кнопки постраничного перехода по списку и кнопку вызова формы добавления записи (в том случае, если пользователь является администратором или работником фабрики). *Id* записей и заголовки столбцов таблицы являются ссылками для вызова формы обновления и удаления записей и сортировки записей в таблицах;

– *\_AddFurniture* и *\_UpdateFurniture* являются частичными представлениями формы добавления, обновления и удаления записей. Эти представления встроены в представление *Index* и имеют разные спозобы вызова: форма *\_AddFurniture* появляется путем нажатия на кнопку добавления записи, а *\_UpdateFurniture* – путем нажатия на *id* изменяемой записи. Форма *\_AddFurniture* содержит поля атрибутов записи *Furniture* и кнопку добавления записи, при нажатии на которую происходит валидация данных и их добавление в таблицу «Мебель» в случае положительного результата. Форма *\_UpdateFurniture* содержит список *id* записей с выбранным *id* изменяемой записи, ссылку на метод удаления записи, а также поля атрибутов записи *Client* и кнопку обновления записи, при нажатии на которую происходит валидация данных и их изменение в случае положительного результата. Если валидация пройдена с отрицательным успехом для любой формы, идет возвращение представления со списком ошибок;

– для управления данными из таблицы «Заказы» были созданы представления *Index*, *\_AddOrder* и *\_UpdateOrder*. Представление *Index* принимает модель *OrderIndexViewModel* и отображает постраничный список данных из таблицы «Заказы» в удобной для пользователя форме, поля и кнопка формы фильтрации с выпадающими списками, кнопки постраничного перехода по списку и кнопку вызова формы добавления записи (в том случае, если пользователь является администратором или работником фабрики). *Id* записей и заголовки столбцов таблицы являются ссылками для вызова формы обновления и удаления записей и сортировки записей в таблицах;

– *\_AddOrder* и *\_UpdateOrder* являются частичными представлениями формы добавления, обновления и удаления записей. Эти представления встроены в представление *Index* и имеют разные спозобы вызова: форма *\_AddOrder* появляется путем нажатия на кнопку добавления записи, а *\_UpdateOrder* – путем нажатия на *id* изменяемой записи. Форма *\_AddOrder* содержит поля и выпадающие списки атрибутов записи *OrderViewModel* и кнопку добавления записи, при нажатии на которую происходит валидация данных и их добавление в таблицу «Заказы» в случае положительного результата. Форма *\_UpdateOrder* содержит список *id* записей с выбранным *id* изменяемой записи, ссылку на метод удаления записи, а также поля и выпадающие списки атрибутов записи *OrderViewModel* и кнопку обновления записи, при нажатии на которую происходит валидация данных и их изменение в случае положительного результата. Если валидация пройдена с отрицательным успехом для любой формы, идет возвращение представления со списком ошибок;

– для управления данными из таблицы «Накладные» были созданы представления *Index*, *\_AddWaybill* и *\_UpdateWaybill*. Представление *Index* принимает модель *WaybillIndexViewModel* и отображает постраничный список данных из таблицы «Накладные» в удобной для пользователя форме, поля и кнопка формы фильтрации с выпадающими списками, кнопки постраничного перехода по списку и кнопку вызова формы добавления записи (в том случае, если пользователь является администратором). *Id* записей и заголовки столбцов таблицы являются ссылками для вызова формы обновления и удаления записей и сортировки записей в таблицах;

– *\_AddWaybill* и *\_UpdateWaybill* являются частичными представлениями формы добавления, обновления и удаления записей. Эти представления встроены в представление *Index* и имеют разные спозобы вызова: форма *\_AddWaybill* появляется путем нажатия на кнопку добавления записи, а *\_UpdateWaybill* – путем нажатия на *id* изменяемой записи. Форма *\_AddWaybill* содержит поля и выпадающие списки атрибутов записи *WaybillViewModel* и кнопку добавления записи, при нажатии на которую происходит валидация данных и их добавление в таблицу «Накладные» в случае положительного результата. Форма *\_UpdateWaybill* содержит список *id* записей с выбранным *id* изменяемой записи, ссылку на метод удаления записи, а также поля и выпадающие списки атрибутов записи *WaybillViewModel* и кнопку обновления записи, при нажатии на которую происходит валидация данных и их изменение в случае положительного результата. Если валидация пройдена с отрицательным успехом для любой формы, идет возвращение представления со списком ошибок.

Для регистрации и входа были созданы представления Register и Login. Register принимает модель RegisterViewModel, содержит поля ввода данных пользователя и пароль с подтверждением пароля, список с выбором роли пользователя и кнопку для регистрации нового пользователя с ролью и данными из формы. Login принимает модель LoginViewModel, содержит поля ввода почты и пароля, флаг запоминания пользователя и кнопку входа пользователя с данными формы;

Для управления пользователями были созданы представления *Index*, *Edit* и *Create*. Для перехода к представлению *Index* необходимо в роли администратора выбрать ссылку «Пользователи» в панели навигации. Index принимает в качестве модели список объектов класса User, имеет таблицу пользователей с кнопками удаления и изменения для каждой записи и ссылку на создание пользователя.

При нажатии на ссылку открывается форма представления *Create*, принимающая модель *CreateUserViewModel*, имеющая поля атрибутов класса *User*, список ролей и кнопку «Добавить», при нажатии на которую происходит валидация данных формы и добавление пользователя с заданной ролью.

При нажатии на кнопку «Изменить» возле записи открывается форма представления *Edit*, принимающая модель *EditUserViewModel*, имеющая поля атрибутов класса *User*, скрытый *id* изменяемого пользователя и кнопку «Сохранить» при нажатии на которую происходит валидация данных и изменение пользователя с сохраненным в модели *id*.

При нажатии на кнопку удалить происходит удаление записи пользователя.

* 1. **Вспомогательные файлы и классы**

Для стилизации приложения в папке *wwwroot* было добавлено изображение заднего фона страниц и создана каскадная таблица стилей *MyStyle*. *MyStyle* содержит стили оформления панели навигации, таблиц, форм и т.д.

Для инициализации базы данных в проекте расположена папка *MiddleWares* с файлом *DbInitializeMiddleWare*, содержащий класс *DbInitializeMiddleWare*, использующий класс *DbInitialize*. Также в файле расположен класс *DbInitializeExtensions* с методом добавления компонента *middleware* в конвейер сервисов [4].

Для встраивания таблиц, связанных с аутентификацией и авторизацией, в базу данных были созданы классы миграции контекста *IdentityContext*. Они хранятся в папке *Migrations* [5].

В класс *Startup* были добавлены вызовы методов встраивания контекстов, сущности пользователя и кэша, а также используемого компонента *middleware*.

Для правильной валидации и отображения содержимого *web*-приложения были добавлены представления: *\_ValidationScriptsPartial*, содержащее ссылки на файлы валидации; *\_ViewImports*, содержащий ссылки на используемые библиотеки; *\_ViewStart*, содержащий информацию о мастер-странице.

В файл конфигурации *appsettings* была добавлена строка подключения.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

4.1 Общие сведения

Данное руководство описывает набор необходимых знаний пользователя, особенности работы приложения и минимальные программные требования.

Целью разработки приложения является упрощение доступа к базе данных предприятия в сети Интернет. В результате получилось приложение, способное обрабатывать данные и выполнять функции, установленные в задаче к курсовой работе.

Пользователь системы должен обладать таким минимальным набором знаний:

– пользовательские навыки работы с ЭВМ;

– знание предметной области и знакомство с руководством пользователя.

Список документов, предоставляемых к обязательному ознакомлению перед началом работы:

– настоящее руководство пользователя;

– инструкциями по технике безопасности (для работы с ЭВМ).

## **4.2 Требования к запуску приложения**

Для запуска приложения требуется:

– операционная система *Windows* версией *Windows* 7 или выше;

– устнановленный *.NET Core* и наличие всех необходимых библиотек;

– установленный *MS SQL Server*, а также наличие необходимой базы данных.

## **4.3 Запуск приложения**

Приложение можно запустить через *exe*-файл напрямую или же через среду программирования. После запуске приложения пользователь может сразу же продолжить работу без использования какой-либо роли. Для завершения программы необходимо закрыть страницу сайта.

## **4.4 Непредвиденные ситуации**

Так как формы ограничивают ввод неверных данных и предупреждает пользователя об ошибках, а неавторизованный доступ к функциям пресекается, ошибки могут возникнуть с базой данных, а также с оформлением сайта и расположением элементов. Это такие ошибки, как:

– потеря соединения с базой данных;

– доступ к данному ресурсу недоступен;

* потеря соединения с сервером;
* отсутствие элементов формы в связи с неправильной загрузкой таблиц стилей или элементов страницы;
* отсутствие реакции кнопок форм на нажатие в связи с отсутствием скриптов.

При возникновении любых непредвиденных ситуаций рекомендуется проверить соединение с базой данных и наличие всех таблиц и полей в базе данных, а также наличие всех необходимых таблиц стилей и скриптов. Лучше всего будет обратится к системному программисту. Если после повторного использования программы возникают ошибки, сообщите разработчику или программисту, который в данный момент поддерживает программу для её устранения.

## **4.5 Описание операции по обработки данных**

Так как авторизованным и неавторизованным пользователям доступен только просмотр данных из базы данных, то в качестве роли необходимо выбрать администратора с полным доступом к приложению.

При запуске программы пользователю откроется окно домашней страницы с историей предприятия. Для регистрации нового администратора необходимо навести курсор мыши на верхнюю часть страницы и в появившейся панели навигации выбрать пункт «Регистрация». Домашняя страница показана на рисунке 4.1.

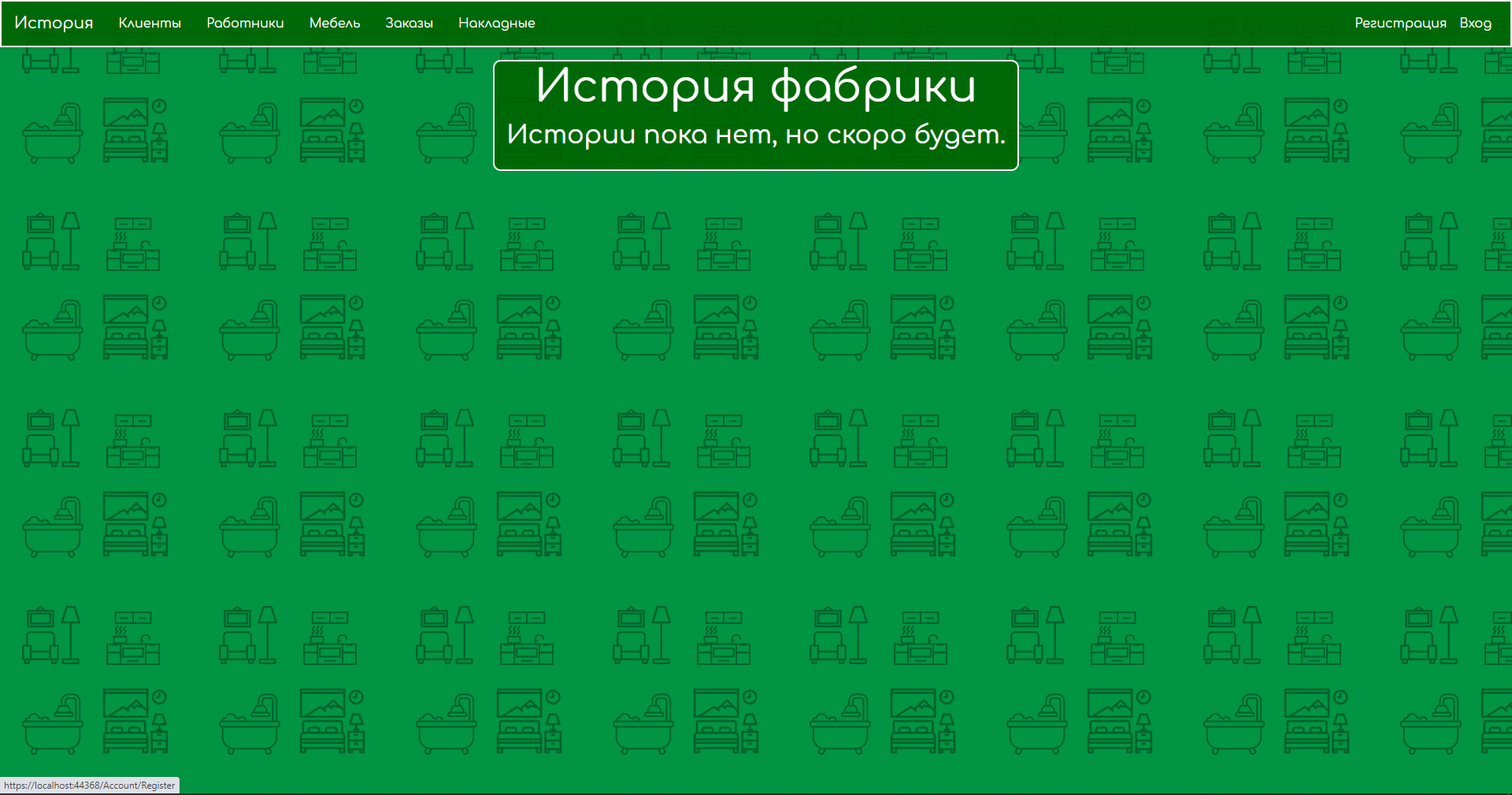


Рисунок 4.1 – домашняя страница

Далее пользователю откроется окно регистрации. Необходимо ввести данные нового пользователя и выбрать роль «Администратор», после чего нажать кнопку «Регистрация». Страница регистрации показана на рисунке 4.2.

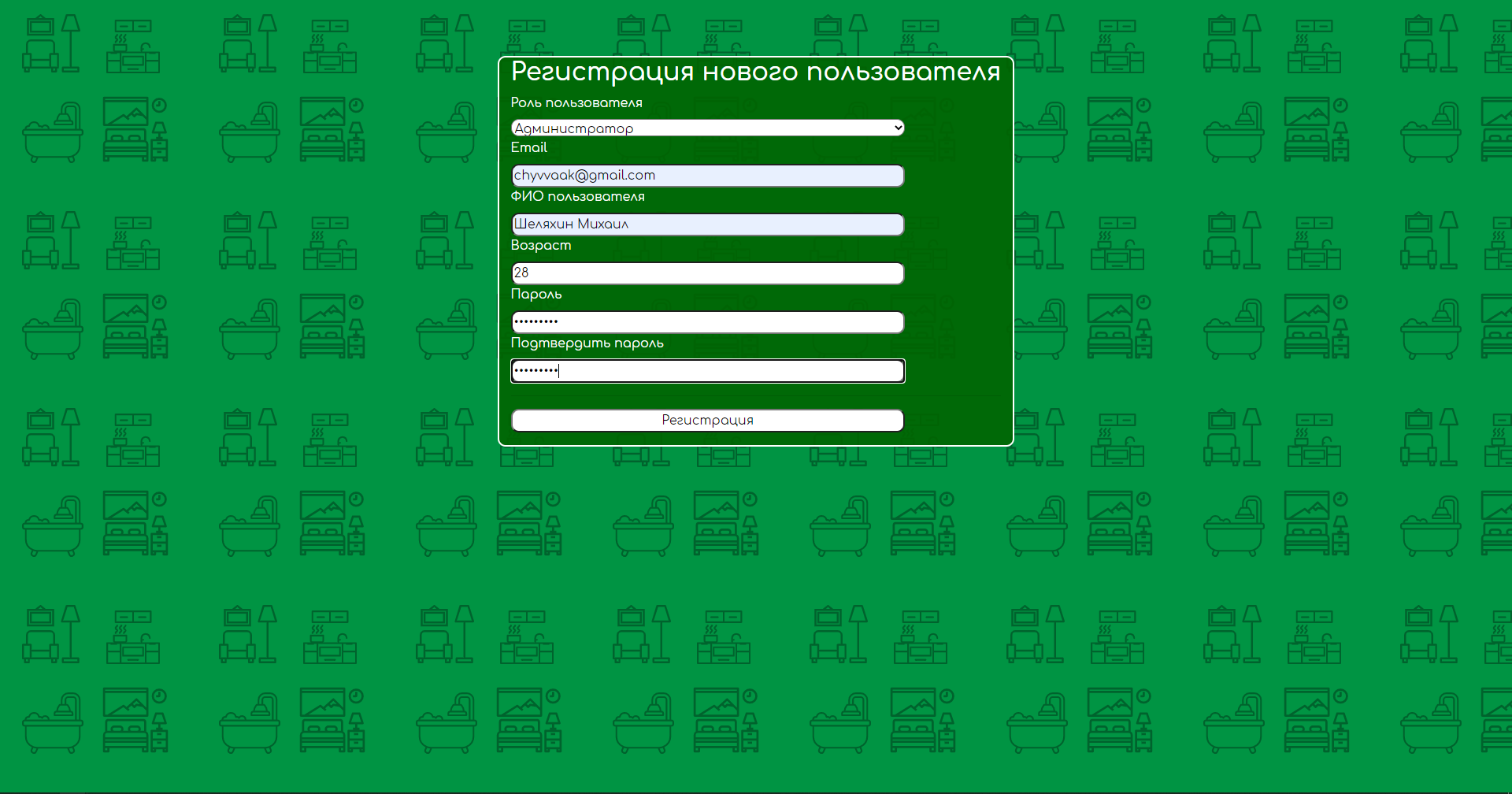


Рисунок 4.2 – страница регистрации

Если пользователь уже зарегистрирован и хочет войти под своими данными, необходимо выбрать в начальном окне пункт «Войти». После открытия окна входа необходимо корректно ввести свои данные и нажать на кнопку «Войти». Страница входа показана на рисунке 4.3.

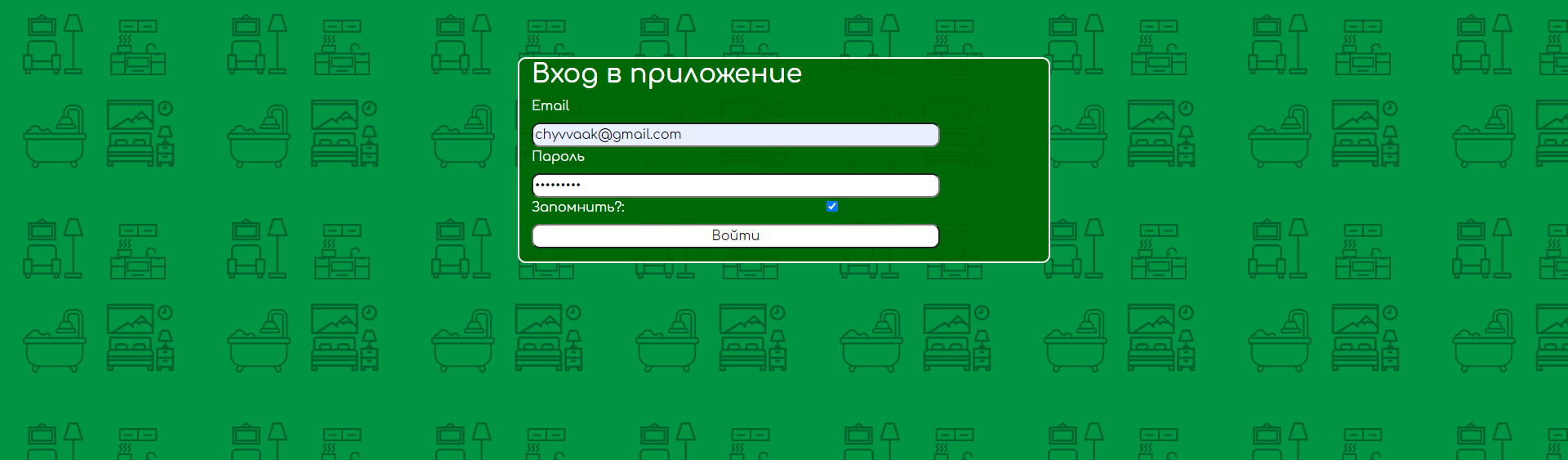


Рисунок 4.3 – страница входа

После успешного входа в панели навигации появится ссылка на страницу управления пользователями «Пользователи». Домашняя страница администратора показана на рисунке 4.4.



Рисунок 4.4 – домашняя страница администратора

Для перехода к странице управления пользователями необходимо выбрать пункт «Пользователи». В открывшемся окне будет список пользователей с возможностью добавления нового пользователя, или удаления либо изменения старого. Страница управления пользователями показана на рисунке 4.5.

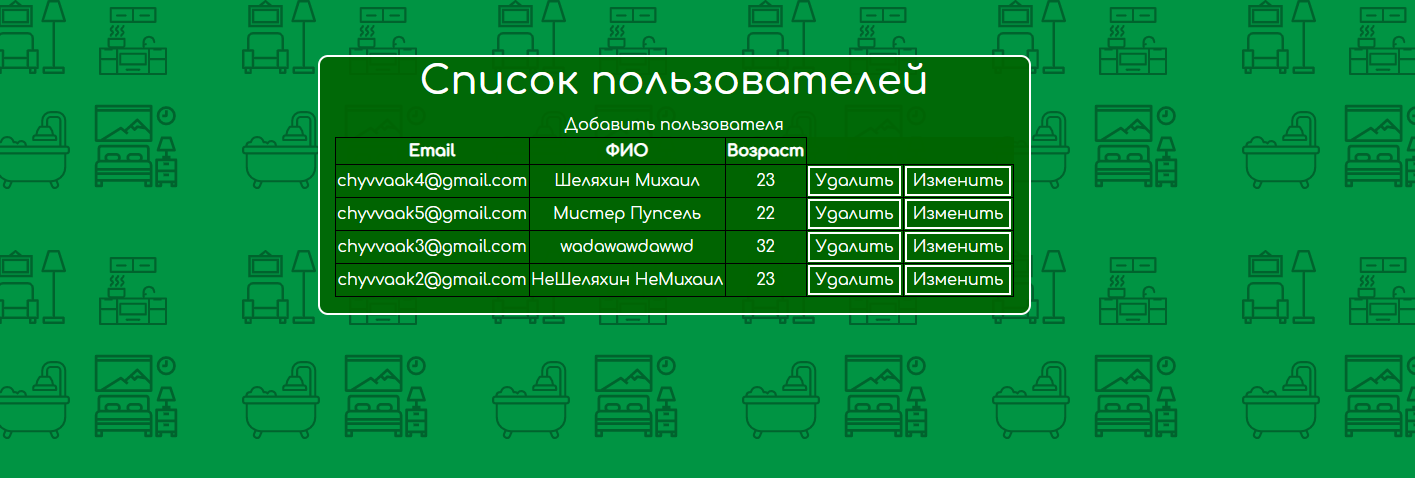


Рисунок 4.5 – страница управления пользователями

При выборе пункта «Добавить пользователя» появится страница с формой для добавления нового пользователя. Для добавления необходимо заполнить поля формы, выбрать роль пользователя и нажать на кнопку «Добавить». Страница формы показана на рисунке 4.6.

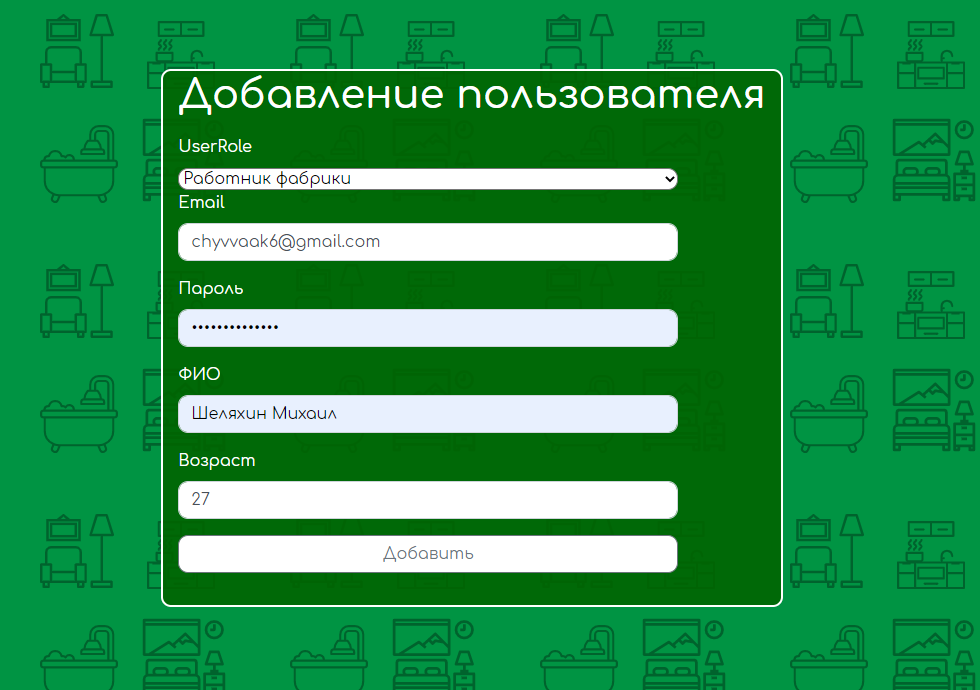


Рисунок 4.6 – страница добавления пользователя

После добавления пользователя произойдет автоматический переход на страницу со списком, где появится новый пользователь. Для его изменения необходимо нажать на кнопку «Изменить» напротив записи для изменения. Откроется страница изменения пользователя, в которую необходимо ввести в поля новые данные и нажать кнопку «Сохранить». Страница формы обновления пользователя показана на рисунке 4.7.

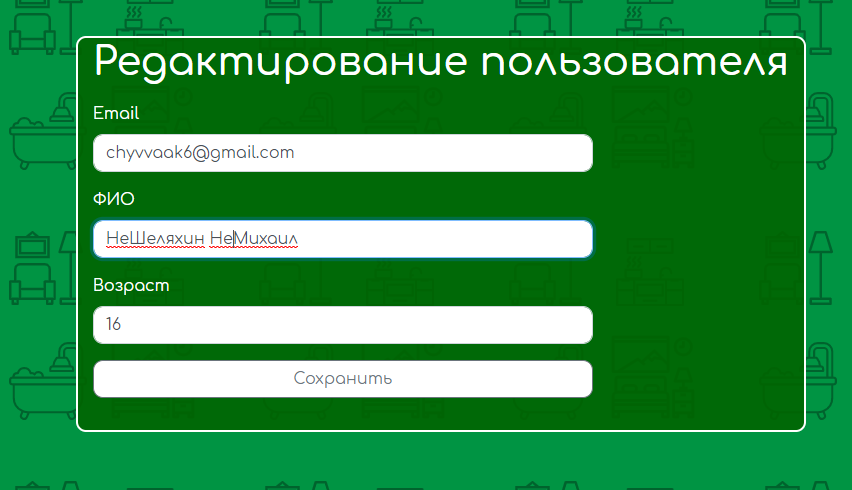


Рисунок 4.7 – окно обновления данных пользователя

После сохранения изменений произойдет автоматический переход на страницу со списком, где будут отображены новые изменения. Для удаления записи необходимо нажать на кнопку «Удалить» напротив записи для удаления. После этого произойдет удаление записи и обновление списка пользователей.

Для просмотра и изменения записей определенной таблицы необходимо выбрать пункт с названием таблицы в панели навигации. Так как структура и функционал страниц отображения записей таблиц идентичен, будет рассматриваться только одна страница с записями из таблицы «Заказы». Для этого необходимо выбрать пункт «Заказы».

После нажатия произойдет открытие страницы «Заказы» со списком заказов, полями для фильтрации, кнопками постраничного просмотра данных, а также кнопкой добавления записи. Страница «Заказы» показана на рисунке 4.8.

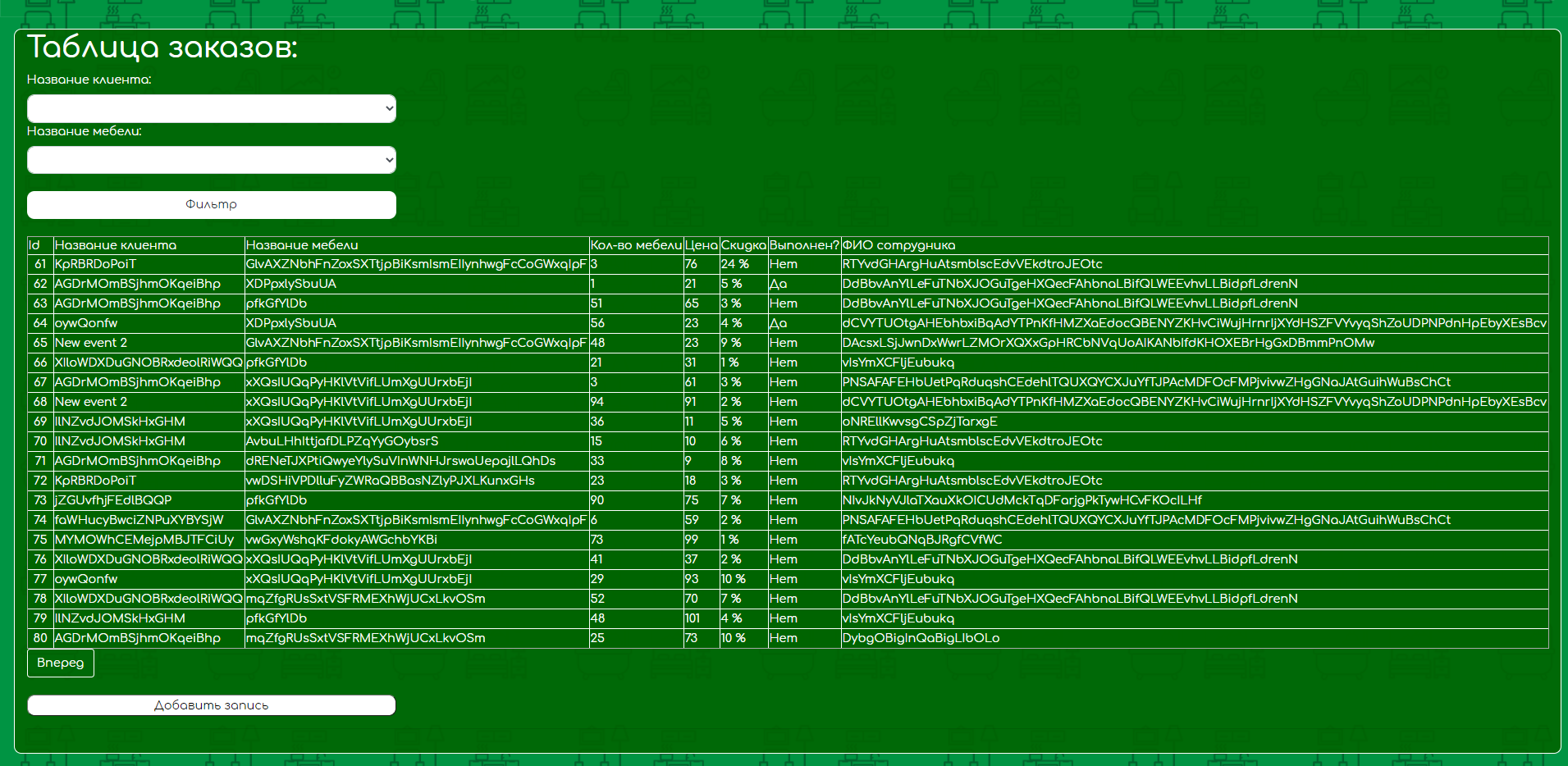
****

Рисунок 4.8 – страница «Заказы»

При вводе данных в поля фильтрации и нажатии кнопки «Фильтр» произойдет выборка данных по заданным критериям и вывод в таблицу. Последние значения полей фильтрации сохраняются в сессии при любом изменении страницы, так что в следующий раз, когда пользователь зайдет на страницу, значения полей будут автоматически заполнены. Страница отфильтрованных данных показана на рисунке 4.9.

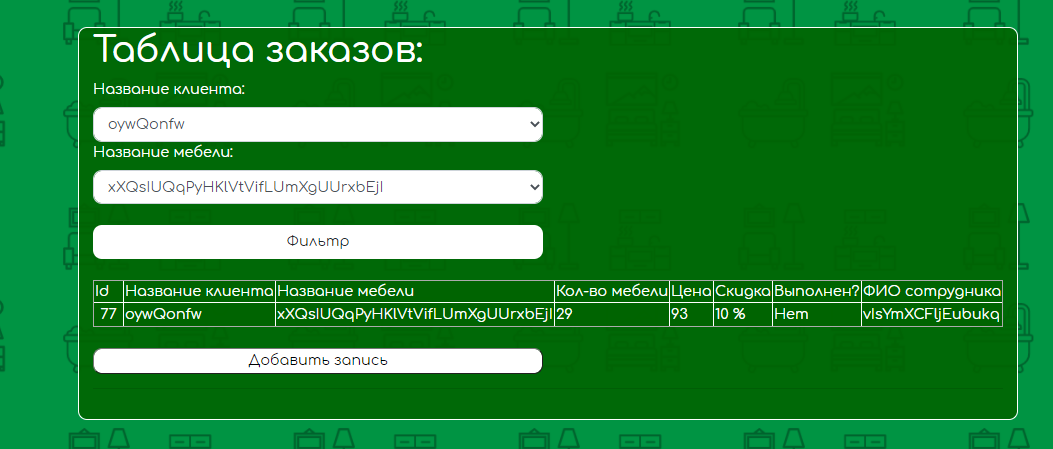
****

Рисунок 4.9 – окно с отфильтрованными данными

При нажатии кнопки вперед или назад отображение данных изменится на следующую или предыдущую группу записей. При этом если есть предыдущая или следующая страница, то автоматически появятся кнопки для перехода. Пагинация показана на рисунке 4.10.

****

Рисунок 4.10 – страница с новым набором данных

При нажатии на заголовок таблицы произойдет сортировка записей по выбранному атрибуту. Сортировка происходит в порядке возрастания.

При нажатии на кнопку «Добавить запись» откроется скрытое до этого частичное представление с формой добавления пользователя. Форма добавления показана на рисунке 4.11.

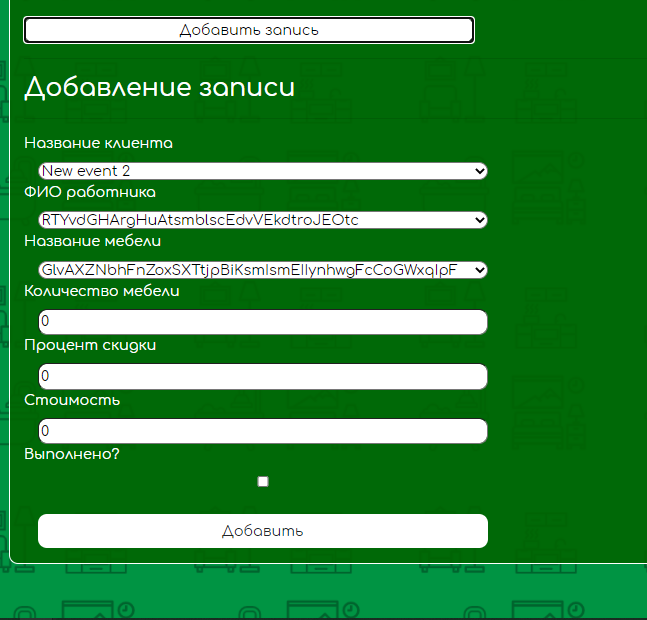
****

Рисунок 4.11 – форма добавления пользователя

При заполнении полей и нажатии на кнопку «Добавить» происходит добавление записи в БД и обновление страницы. Данные формы проходят валидацию, и если она прошла неудачно, то происходит возврат страницы со списком ошибок.

При нажатии на *id* записи открывается частичное представление с формой обновления записи и ссылкой на удаление записи. При выборе ссылки «Удалить заказ» происходит удаление записи, а при заполнении полей и выборе кнопки «Изменить» происходит валидация и изменение записи с выбранным *id*. Окно изменения данных показано на рисунке 4.12.

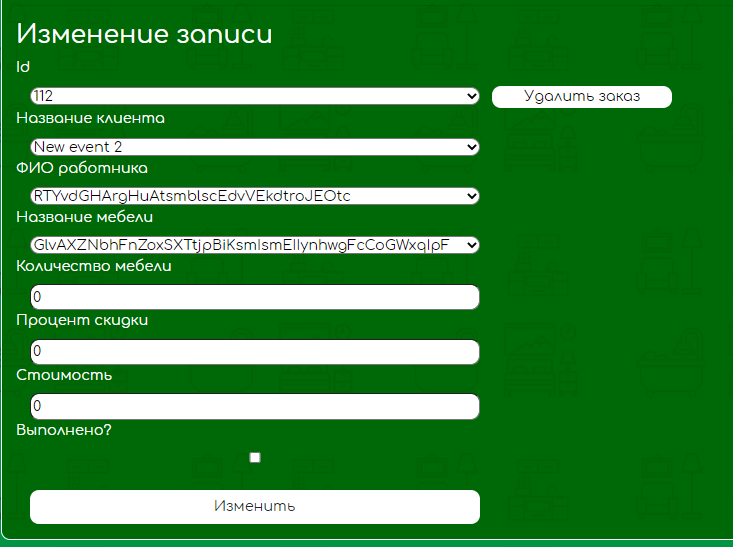
****

Рисунок 4.12 – окно изменения данных

1. **РУКОВОДСТВО ПРОГРАММИСТА**

## **5.1 Назначение и условия применения программы**

Приложение предназначено для работы с базами данных.

Основные функции приложения:

– предоставление сведений из таблиц базы данных в удобном для пользователя виде;

– осуществление авторизации и аутентификации;

– добавление данных в таблицы авторизованными пользователями;

– удаление данных из таблиц авторизованными пользователями;

* изменение данных в таблицах авторизованными пользователями;

– управление ролями.

Для запуска приложения должен быть установлен *.NET Core*. Для считывания данных устройство должно соединятся с базой данных *MS SQL Server*. Для корректного отображения информации необходимо наличие всех таблиц стилей, скриптов и изображений интерфейса.

## **5.2 Характеристики программы**

Приложение написано в среде разработки *Visual Studio 2019*, с помощью технологии *.NET Core* на языке *C#*. База данных поддерживается с помощью СУБД *MS SQL Server*, а доступ к данным осуществляется с помощью технологии *Entity Framework Core*. Графический интерфейс и работа в браузере реализована с помощью технологии *ASP.NET Core MVC*.

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

## **5.3 Сопровождение программного комплекса**

Для дополнения программного обеспечения новым функционалом можно использовать любую среду разработки на языке программирования C#. Приложения реализовано с помощью паттерна *MVC* (*Model-View-Controller*), который позволяет в свою очередь разделить модель данных, бизнес-логику приложения и представления, на три части, что позволит разрабатывать новый функционал и поддерживать приложения в команде из нескольких разработчиков. Так же использование данного паттерна сделала приложение легко масштабируемым и поддерживаемым.

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

## **5.4 Обращение к программе**

Обращение к программе происходит при запуске .*exe* файла или в среде разработки и настройки соединения с СУБД *MS SQL Server*.

Для дополнения программного обеспечения новым функционалом можно использовать любую среду разработки на языке программирования *C#*. Понятная иерархия позволяют модифицировать приложение без опасности нарушить основной функционал программы. Так как используется технология *Entity Framework Core* для связи с базой данных, не рекомендуется использовать другую СУБД.

## **5.5 Входные и выходные данные**

Для добавления входных данных необходимо через приложение использовать формы для управления данными в базе данных, расположенные на страницах таблиц БД.

Выходными данными для приложения являются отображаемые пользователю таблицы в представлении с актуальными данными.

## **5.6 Сообщение в ходе работы приложения**

При работе программа может оповещать пользователя о следующих неполадках в работы системы:

– некорректные учетные данные пользователя;

– несуществующий пользователь;

– ошибки корректности ввода данных;

– остутствие необходимых изображений;

– потеря соединения с базой данных;

– доступ к данному ресурсу недоступен;

– остутствие таблиц или полей.

Оповещения о неполадках с базой данных производится путем сообщений в специальном виде ошибки с описанием. Остальные же ошибки отображаются в приложении в виде сообщения с предупреждением и остановкой выполнения некорректной функции без завершения работы приложения.

ЗАКЛЮЧЕНИЕ

При выполнении курсовой работы создано *web*-приложение баз данных по фабрике мебели. Данное приложение обеспечивает связь с базой данных, визуализацию получаемой из базы данных информацию, ввод пользователем данных проверяемых на корректность, диалог с пользователем, а также авторизацию и аутентификацию.

Для создания приложения использовался язык программирования *C#* и платформа *.NET Core*, который взаимодействует с СУБД *Microsoft SQL Management Studio*, осуществляющий доступ к данным с помощью технологии *ADO.NET Entity Framework Core*, производя анализ и обработку данных с помощью языка запросов *LINQ to Objects* и предоставляя информацию конечному пользователю с помощью *web*-приложения, созданного с технологией *ASP.NET Core*.

Благодаря реализации шаблона проектирования *MVC* и легко разделяемой иерархии классов приложение доступно для модификаций и улучшений. Можно использовать новые классы, удалять старые или модифицировать уже созданные под свои нужды; то же самое можно сделать и с представлениями и контроллерами.

Проведенные интеграционные тесты доказывают работоспособность данной программы и функционирование всех требуемых ролей и методов. Полученное приложение готово для работы со среднестатистическим пользователем.

Данное приложение предназначено для предприятия, которое стремится вынести деятельность на просторы сети Интернет и желает получать актуальную информацию о предприятии в любое время и в любом месте. Оно позволит охватить аудиторию покупателей, активно проводящих время в сети Интернет, и привлечь работников, стремящихся максимально эффективно организовать рабочее время.

Список используемых источников

1. Практическое руководство к курсовому проектированию по курсу «Ин-форматика» для студентов технических специальностей дневной и заочной форм обучения – Гомель: ГГТУ им. П.О. Сухого, 2019. – 32 с.

2. Джон Смит. Entity Framework Core in Action– Manning Publications, 2018. — 486 p.

3. Эндрю Локк. ASP.NET Core in Action. – Manning Publications, 2018. – 712 с.

4. Создание компонентов middleware, – Электрон. данные. – Режим доступа: https://metanit.com/sharp/aspnet5/2.4.php. – Дата доступа: 12.12.2020.

5. Миграции, – Электрон. данные. – Режим доступа: https://metanit.com/sharp/entityframework/3.12.php. – Дата доступа: 12.12.2020.

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**

*Startup.cs*

using CourseProject.MiddleWares;

using CourseProject.Models;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

namespace CourseProject

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

// Добавление профилей кэширования для таблиц и остальных страниц

services.AddControllersWithViews(options =>

{

options.CacheProfiles.Add("TablesCaching",

new CacheProfile()

{

Duration = 286

});

options.CacheProfiles.Add("NoCaching",

new CacheProfile()

{

Location = ResponseCacheLocation.None,

NoStore = true

});

});

// Добавление контекста данных со строкой подключения, хранящейся в файле appsettings.json

services.AddDbContext<ApplicationContext>(options => options.UseSqlServer(Configuration.GetConnectionString("SqlServerConnection")));

services.AddDbContext<IdentityContext>(options => options.UseSqlServer(Configuration.GetConnectionString("SqlServerConnection")));

services.AddIdentity<User, IdentityRole>(options => { options.User.RequireUniqueEmail = true; }).AddEntityFrameworkStores<IdentityContext>();

services.AddMemoryCache();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

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

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseInitializeMiddleware();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

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

});

}

}

}

*Client.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Клиенты

public class Client

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string RepresentativeFIO { get; set; }

public int Number { get; set; }

public string Address { get; set; }

public Client() { }

}

}

*Employee.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Работники

public class Employee

{

[Key]

public int Id { get; set; }

public string FIO { get; set; }

public string Position { get; set; }

public string Education { get; set; }

public Employee() { }

}

}

*Furniture.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Мебель

public class Furniture

{

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public int Count { get; set; }

public Furniture() { }

}

}

*Order.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Заказы

public class Order

{

[Key]

public int Id { get; set; }

public int ClientId { get; set; }

public int FurnitureId { get; set; }

public int FurnitureCount { get; set; }

public decimal Price { get; set; }

public int DiscountPercent { get; set; }

public int IsCompleted { get; set; }

public int EmployeeId { get; set; }

public Order() { }

}

}

*Waybill.cs*

using System;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

// Модель, представляющая запись в таблице Накладные

public class Waybill

{

[Key]

public int Id { get; set; }

public int ProviderId { get; set; }

public string ProviderName { get; set; }

public DateTime DateOfSupply { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public double Weight { get; set; }

public int FurnitureId { get; set; }

public int EmployeeId { get; set; }

public Waybill() { }

}

}

*CreateUserViewModel.cs*

namespace CourseProject.Models

{

public class CreateUserViewModel

{

public string UserRole { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string FIO { get; set; }

public int Age { get; set; }

}

}

*EditUserViewModel.cs*

namespace CourseProject.Models

{

public class EditUserViewModel

{

public string Id { get; set; }

public string Email { get; set; }

public int Age { get; set; }

public string FIO { get; set; }

}

}

*IdentityContext.cs*

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace CourseProject.Models

{

public class IdentityContext : IdentityDbContext<User>

{

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

{

Database.EnsureCreated();

}

}

}

*LoginViewModel.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class LoginViewModel

{

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить?")]

public bool RememberMe { get; set; }

public string ReturnUrl { get; set; }

}

}

*RegisterViewModel.cs*

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class RegisterViewModel

{

[Required]

[Display(Name = "Роль пользователя")]

public string UserRole { get; set; }

[Required]

[DataType(DataType.EmailAddress)]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[Display(Name = "ФИО пользователя")]

public string FIO { get; set; }

[Required]

[Display(Name = "Возраст")]

public int Age { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Required]

[Compare("Password", ErrorMessage = "Пароли не совпадают")]

[DataType(DataType.Password)]

[Display(Name = "Подтвердить пароль")]

public string PasswordConfirm { get; set; }

}

}

*User.cs*

using Microsoft.AspNetCore.Identity;

namespace CourseProject.Models

{

public class User : IdentityUser

{

public string FIO { get; set; }

public int Age { get; set; }

}

}

*ClientIndexViewModel.cs*

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class ClientIndexViewModel

{

public List<Client> Clients { get; set; }

public List<int> Ids { get; set; }

public PageViewModel PageViewModel { get; set; }

public List<string> FilterRepresFIOs { get; set; }

public string FilterRepresFIO { get; set; }

public string FilterAddress { get; set; }

[Display(Name = "Id")]

public int Id { get; set; }

[Display(Name = "Название")]

public string Name { get; set; }

[Display(Name = "ФИО представителя")]

public string RepresentativeFIO { get; set; }

[Display(Name = "Номер")]

public int Number { get; set; }

[Display(Name = "Адрес")]

public string Address { get; set; }

}

}

*EmployeeIndexViewModel.cs*

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class EmployeeIndexViewModel

{

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

public PageViewModel PageViewModel { get; set; }

public List<int> Ids { get; set; }

public List<string> FilterPositions { get; set; }

public string FilterPosition { get; set; }

public string FilterEducation { get; set; }

[Display(Name = "Id")]

public int Id { get; set; }

[Display(Name = "ФИО")]

public string FIO { get; set; }

[Display(Name = "Должность")]

public string Position { get; set; }

[Display(Name = "Образование")]

public string Education { get; set; }

}

}

*FurnitureIndexViewModel.cs*

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class FurnitureIndexViewModel

{

public List<Furniture> Furniture { get; set; }

public PageViewModel PageViewModel { get; set; }

public List<string> FilterMaterials { get; set; }

public string FilterMaterial { get; set; }

public decimal FilterPrice { get; set; }

public List<int> Ids { get; set; }

[Display(Name = "Id")]

public int Id { get; set; }

[Display(Name = "Название")]

public string Name { get; set; }

[Display(Name = "Описание")]

public string Description { get; set; }

[Display(Name = "Материал")]

public string Material { get; set; }

[Display(Name = "Цена")]

public decimal Price { get; set; }

[Display(Name = "Количество")]

public int Count { get; set; }

}

}

*OrderViewModel.cs*

namespace CourseProject.Models

{

// Модель отображения записей из таблицы Заказы с заменой внешних ключей смысловыми данными

public class OrderViewModel

{

public int Id { get; set; }

public string ClientName { get; set; }

public string FurnitureName { get; set; }

public int FurnitureCount { get; set; }

public decimal Price { get; set; }

public int DiscountPercent { get; set; }

public bool IsCompleted { get; set; }

public string EmployeeFIO { get; set; }

}

}

*OrderIndexViewModel.cs*

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class OrderIndexViewModel

{

public List<OrderViewModel> OrderViewModels { get; set; }

public List<string> EmployeesFIOs { get; set; }

public List<string> ClientNames { get; set; }

public List<string> FurnitureNames { get; set; }

public PageViewModel PageViewModel { get; set; }

public List<string> FilterClients { get; set; }

public string FilterClient { get; set; }

public List<string> FilterFurnitures { get; set; }

public string FilterFurniture { get; set; }

public List<int> Ids { get; set; }

[Display(Name = "Id")]

public int Id { get; set; }

[Display(Name = "Название клиента")]

public string ClientName { get; set; }

[Display(Name = "Название мебели")]

public string FurnitureName { get; set; }

[Display(Name = "Количество мебели")]

public int FurnitureCount { get; set; }

[Display(Name = "Стоимость")]

public decimal Price { get; set; }

[Display(Name = "Процент скидки")]

public int DiscountPercent { get; set; }

[Display(Name = "Выполнено?")]

public bool IsCompleted { get; set; }

[Display(Name = "ФИО работника")]

public string EmployeeFIO { get; set; }

}

}

*WaybillViewModel.cs*

using System;

namespace CourseProject.Models

{

// Модель отображения записей из таблицы Накладные с заменой внешних ключей смысловыми данными

public class WaybillViewModel

{

public int Id { get; set; }

public int ProviderId { get; set; }

public string ProviderName { get; set; }

public DateTime DateOfSupply { get; set; }

public string Material { get; set; }

public decimal Price { get; set; }

public double Weight { get; set; }

public string FurnitureName { get; set; }

public string EmployeeFIO { get; set; }

}

}

*WaybillIndexViewModel.cs*

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseProject.Models

{

public class WaybillIndexViewModel

{

public List<WaybillViewModel> WaybillViewModels { get; set; }

public List<string> EmployeesFIOs { get; set; }

public List<string> FurnitureNames { get; set; }

public PageViewModel PageViewModel { get; set; }

public List<string> FilterFurnitureNames { get; set; }

public string FilterFurnitName { get; set; }

public string FilterProviderName { get; set; }

public List<int> Ids { get; set; }

[Display(Name = "Id")]

public int Id { get; set; }

[Display(Name = "Номер поставщика")]

public int ProviderId { get; set; }

[Display(Name = "Название мебели")]

public string FurnitureName { get; set; }

[Display(Name = "Название поставщика")]

public string ProviderName { get; set; }

[Display(Name = "Стоимость")]

public decimal Price { get; set; }

[Display(Name = "Дата поставок")]

public DateTime DateOfSupply { get; set; }

[Display(Name = "Материал")]

public string Material { get; set; }

[Display(Name = "ФИО работника")]

public string EmployeeFIO { get; set; }

[Display(Name = "Вес")]

public double Weight { get; set; }

}

}

*ApplicationContext.cs*

using Microsoft.EntityFrameworkCore;

namespace CourseProject.Models

{

// Класс контекста данных

public class ApplicationContext : DbContext

{

public DbSet<Client> Clients { get; set; }

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

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

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

public DbSet<Waybill> Waybills { get; set; }

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

{

}

}

}

*PageViewModel.cs*

using System;

namespace CourseProject.Models

{

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);

}

}

}

}

*HomeController.cs*

using System.Diagnostics;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using CourseProject.Models;

namespace CourseProject.Controllers

{

// Контроллер начальной страницы

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

// Метод получения начальной страницы.

// Начальная страница не кэшируется.

[ResponseCache(CacheProfileName = "NoCaching")]

public IActionResult Index()

{

return View();

}

// Метод получения страницы ошибки.

// Страница ошибки не кэшируется.

[ResponseCache(CacheProfileName = "NoCaching")]

public IActionResult Error()

{

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

}

}

}

*AccountController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

namespace CourseProject.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, RoleManager<IdentityRole> roleManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_roleManager = roleManager;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

User user = new User { Email = model.Email, UserName = model.Email, FIO = model.FIO, Age = model.Age };

// добавляем пользователя

var result = await \_userManager.CreateAsync(user, model.Password);

if (!await \_roleManager.RoleExistsAsync(model.UserRole))

{

await \_roleManager.CreateAsync(new IdentityRole(model.UserRole));

}

await \_userManager.AddToRoleAsync(user, model.UserRole);

if (result.Succeeded)

{

// установка куки

await \_signInManager.SignInAsync(user, false);

return RedirectToAction("Index", "Home");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

return View(new LoginViewModel { ReturnUrl = returnUrl });

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

// проверяем, принадлежит ли URL приложению

if (!string.IsNullOrEmpty(model.ReturnUrl) && Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

else

{

ModelState.AddModelError("", "Неправильный логин и (или) пароль");

}

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

// удаляем аутентификационные куки

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

}

}

*ClientController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Caching.Memory;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CourseProject.Controllers

{

public class ClientController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

private IMemoryCache cache;

public ClientController(ApplicationContext applicationContext, IMemoryCache cache)

{

db = applicationContext;

this.cache = cache;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult Index(int page = 1, string filterRepresFIO = "Все", string address = null, string type = null)

{

int pageSize = 20;

List<Client> clients;

if (!cache.TryGetValue("Clients", out clients))

{

clients = db.Clients.ToList();

cache.Set("Clients", db.Clients.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

}

List<int> Ids = clients.Select(item => item.Id).ToList();

var repFios = clients.Select(item => item.RepresentativeFIO).ToList();

repFios.Add("Все");

if (filterRepresFIO != "Все")

{

clients = clients.Where(item => item.RepresentativeFIO == filterRepresFIO).ToList();

}

if (address != null)

{

clients = clients.Where(item => item.Address.Contains(address)).ToList();

}

if (type != null)

{

clients = type switch

{

"Id" => clients.OrderBy(item => item.Id).ToList(),

"name" => clients.OrderBy(item => item.Name).ToList(),

"fio" => clients.OrderBy(item => item.RepresentativeFIO).ToList(),

"numb" => clients.OrderBy(item => item.Number).ToList(),

\_ => clients.OrderBy(item => item.Address).ToList(),

};

}

ClientIndexViewModel clientIndexViewModel = new ClientIndexViewModel()

{

Clients = clients.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),

Ids = Ids,

PageViewModel = new PageViewModel(clients.Count, page, pageSize),

FilterRepresFIOs = repFios

};

return View(clientIndexViewModel);

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult AddClient(ClientIndexViewModel model)

{

var names = db.Clients.Select(item => item.Name);

ViewData["Message"] = "";

model.Clients = db.Clients.ToList();

model.Ids = db.Employees.Select(item => item.Id).ToList();

if (model.Name == null || model.RepresentativeFIO == null || model.Address == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Client/Index.cshtml", model);

}

if (names.Contains(model.Name) || model.Name.Length == 0 || model.Name.Length > 25)

{

ViewData["Message"] += "Неправильный ввод названия";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.RepresentativeFIO.Length == 0 || model.RepresentativeFIO.Length > 100)

{

ViewData["Message"] += "Неправильный ввод ФИО представителя";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.Number.ToString().Length < 9 || model.Number.ToString().Length > 13)

{

ViewData["Message"] += "Неправильный ввод номера";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.Address.Length == 0 || model.Address.Length > 40)

{

ViewData["Message"] += "Неправильный ввод адреса";

return View("~/Views/Client/Index.cshtml", model);

}

else

{

var id = 0;

if (db.Clients.Count() != 0)

{

id = db.Clients.Select(item => item.Id).Max();

}

id++;

db.Clients.Add(new Client() { Id = id, Name = model.Name, Address = model.Address, Number = model.Number, RepresentativeFIO = model.RepresentativeFIO });

db.SaveChanges();

cache.Remove("Clients");

cache.Set("Clients", db.Clients.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Client");

}

}

[Authorize(Roles = "Администратор")]

public IActionResult DeleteClient(int id)

{

ViewData["Message"] = "";

var client = db.Clients.Where(item => item.Id == id).FirstOrDefault();

db.Clients.Remove(client);

db.SaveChanges();

cache.Remove("Clients");

cache.Set("Clients", db.Clients.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Client");

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult UpdateClient(ClientIndexViewModel model, string action = null)

{

if (action != null)

{

return DeleteClient(model.Id);

}

var names = db.Clients.Select(item => item.Name);

ViewData["Message"] = "";

model.Clients = db.Clients.ToList();

model.Ids = db.Clients.Select(item => item.Id).ToList();

if (model.Name == null || model.RepresentativeFIO == null || model.Address == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Client/Index.cshtml", model);

}

if (names.Contains(model.Name) || model.Name.Length == 0 || model.Name.Length > 25)

{

ViewData["Message"] += "Неправильный ввод названия";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.RepresentativeFIO.Length == 0 || model.RepresentativeFIO.Length > 100)

{

ViewData["Message"] += "Неправильный ввод ФИО представителя";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.Number.ToString().Length < 9 || model.Number.ToString().Length > 13)

{

ViewData["Message"] += "Неправильный ввод номера";

return View("~/Views/Client/Index.cshtml", model);

}

else if (model.Address.Length == 0 || model.Address.Length > 40)

{

ViewData["Message"] += "Неправильный ввод адреса";

return View("~/Views/Client/Index.cshtml", model);

}

else

{

var client = db.Clients.Where(item => item.Id == model.Id).FirstOrDefault();

client.Name = model.Name;

client.RepresentativeFIO = model.RepresentativeFIO;

client.Number = model.Number;

client.Address = model.Address;

db.SaveChanges();

cache.Remove("Clients");

cache.Set("Clients", db.Clients.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Client");

}

}

}

}

*EmployeeController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Caching.Memory;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CourseProject.Controllers

{

public class EmployeeController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

private IMemoryCache cache;

public EmployeeController(ApplicationContext applicationContext, IMemoryCache cache)

{

db = applicationContext;

this.cache = cache;

}

// Метод получения страницы работников.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult Index(int page = 1, string position = "Все", string education = null, string type = null)

{

int pageSize = 20;

List<Employee> employees;

if (!cache.TryGetValue("Employees", out employees))

{

employees = db.Employees.ToList();

cache.Set("Employees", db.Employees.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

}

List<int> Ids = employees.Select(item => item.Id).ToList();

List<string> positions = employees.Select(item => item.Position).ToList();

positions.Add("Все");

if (position != "Все")

{

employees = employees.Where(item => item.Position == position).ToList();

}

if (education != null)

{

employees = employees.Where(item => item.Education.Contains(education)).ToList();

}

if (type != null)

{

employees = type switch

{

"Id" => employees.OrderBy(item => item.Id).ToList(),

"pos" => employees.OrderBy(item => item.Position).ToList(),

"fio" => employees.OrderBy(item => item.FIO).ToList(),

\_ => employees.OrderBy(item => item.Education).ToList(),

};

}

EmployeeIndexViewModel employeeIndexViewModel = new EmployeeIndexViewModel()

{

Employees = employees.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),

Ids = Ids,

PageViewModel = new PageViewModel(employees.Count, page, pageSize),

FilterPositions = positions

};

return View(employeeIndexViewModel);

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult AddEmployee(EmployeeIndexViewModel model)

{

var fios = db.Employees.Select(item => item.FIO);

ViewData["Message"] = "";

model.Employees = db.Employees.ToList();

model.Ids = db.Employees.Select(item => item.Id).ToList();

if (model.FIO == null || model.Education == null || model.Position == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Employee/Index.cshtml", model);

}

if (fios.Contains(model.FIO) || model.FIO.Length == 0 || model.FIO.Length > 100)

{

ViewData["Message"] += "Неправильный ввод ФИО";

return View("~/Views/Employee/Index.cshtml", model);

}

else if (model.Position.Length == 0 || model.Position.Length > 50)

{

ViewData["Message"] += "Неправильный ввод должности";

return View("~/Views/Employee/Index.cshtml", model);

}

else if (model.Education.Length < 9 || model.Education.Length > 200)

{

ViewData["Message"] += "Неправильный ввод образования";

return View("~/Views/Employee/Index.cshtml", model);

}

else

{

var id = 0;

if (db.Employees.Count() != 0)

{

id = db.Employees.Select(item => item.Id).Max();

}

id++;

db.Employees.Add(new Employee() { Id = id, FIO = model.FIO, Position = model.Position, Education = model.Education });

db.SaveChanges();

cache.Remove("Employees");

cache.Set("Employees", db.Employees.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Employee");

}

}

[Authorize(Roles = "Администратор")]

public IActionResult DeleteEmployee(int id)

{

ViewData["Message"] = "";

var employee = db.Employees.Where(item => item.Id == id).FirstOrDefault();

db.Employees.Remove(employee);

db.SaveChanges();

cache.Remove("Employees");

cache.Set("Employees", db.Employees.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Employee");

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult UpdateEmployee(EmployeeIndexViewModel model, string action = null)

{

if (action != null)

{

return DeleteEmployee(model.Id);

}

var fios = db.Employees.Select(item => item.FIO);

ViewData["Message"] = "";

model.Employees = db.Employees.ToList();

model.Ids = db.Employees.Select(item => item.Id).ToList();

if (model.FIO == null || model.Education == null || model.Position == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Employee/Index.cshtml", model);

}

if (fios.Contains(model.FIO) || model.FIO.Length == 0 || model.FIO.Length > 100)

{

ViewData["Message"] += "Неправильный ввод ФИО";

return View("~/Views/Employee/Index.cshtml", model);

}

else if (model.Position.Length == 0 || model.Position.Length > 50)

{

ViewData["Message"] += "Неправильный ввод должности";

return View("~/Views/Employee/Index.cshtml", model);

}

else if (model.Education.Length < 0 || model.Education.Length > 200)

{

ViewData["Message"] += "Неправильный ввод образования";

return View("~/Views/Employee/Index.cshtml", model);

}

else

{

var employee = db.Employees.Where(item => item.Id == model.Id).FirstOrDefault();

employee.FIO = model.FIO;

employee.Position = model.Position;

employee.Education = model.Education;

db.SaveChanges();

cache.Remove("Employees");

cache.Set("Employees", db.Employees.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Employee");

}

}

}

}

*FurnitureController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Caching.Memory;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CourseProject.Controllers

{

public class FurnitureController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

private IMemoryCache cache;

public FurnitureController(ApplicationContext applicationContext, IMemoryCache cache)

{

db = applicationContext;

this.cache = cache;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult Index(int page = 1, string material = "Все", decimal price = 0, string type = null)

{

int pageSize = 20;

List<Furniture> furniture;

if (!cache.TryGetValue("Furniture", out furniture))

{

furniture = db.Furniture.ToList();

cache.Set("Furniture", db.Furniture.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

}

List<int> Ids = furniture.Select(item => item.Id).ToList();

List<string> materials = furniture.Select(item => item.Material).ToList();

materials.Add("Все");

if (material != "Все")

{

furniture = furniture.Where(item => item.Material == material).ToList();

}

if (price != 0)

{

furniture = furniture.Where(item => item.Price == price).ToList();

}

if (type != null)

{

furniture = type switch

{

"Id" => furniture.OrderBy(item => item.Id).ToList(),

"name" => furniture.OrderBy(item => item.Name).ToList(),

"descr" => furniture.OrderBy(item => item.Description).ToList(),

"material" => furniture.OrderBy(item => item.Material).ToList(),

"price" => furniture.OrderBy(item => item.Price).ToList(),

\_ => furniture.OrderBy(item => item.Count).ToList(),

};

}

FurnitureIndexViewModel furnitureIndexViewModel = new FurnitureIndexViewModel()

{

Furniture = furniture.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),

Ids = Ids,

PageViewModel = new PageViewModel(furniture.Count, page, pageSize),

FilterMaterials = materials

};

return View(furnitureIndexViewModel);

}

[Authorize(Roles = "Администратор, Работник фабрики")]

[HttpPost]

public IActionResult AddFurniture(FurnitureIndexViewModel model)

{

var names = db.Furniture.Select(item => item.Name);

ViewData["Message"] = "";

model.Furniture = db.Furniture.ToList();

model.Ids = db.Furniture.Select(item => item.Id).ToList();

if (model.Name == null || model.Description == null || model.Material == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Furniture/Index.cshtml", model);

}

if (names.Contains(model.Name) || model.Name.Length == 0 || model.Name.Length > 50)

{

ViewData["Message"] += "Неправильный ввод названия";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Description.Length == 0 || model.Description.Length > 200)

{

ViewData["Message"] += "Неправильный ввод описания";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Material.Length == 0 || model.Material.Length > 60)

{

ViewData["Message"] += "Неправильный ввод материала";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Count <= 0)

{

ViewData["Message"] += "Неправильное значение количества";

return View("~/Views/Furniture/Index.cshtml", model);

}

else

{

var id = 0;

if (db.Furniture.Count() != 0)

{

id = db.Furniture.Select(item => item.Id).Max();

}

id++;

db.Furniture.Add(new Furniture() { Id = id, Name = model.Name, Description = model.Description, Material = model.Material, Price = model.Price, Count = model.Count });

db.SaveChanges();

cache.Remove("Furniture");

cache.Set("Furniture", db.Furniture.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Furniture");

}

}

[Authorize(Roles = "Администратор, Работник фабрики")]

public IActionResult DeleteFurniture(int id)

{

ViewData["Message"] = "";

var furniture = db.Furniture.Where(item => item.Id == id).FirstOrDefault();

db.Furniture.Remove(furniture);

db.SaveChanges();

cache.Remove("Furniture");

cache.Set("Furniture", db.Furniture.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Furniture");

}

[Authorize(Roles = "Администратор, Работник фабрики")]

[HttpPost]

public IActionResult UpdateFurniture(FurnitureIndexViewModel model, string action = null)

{

if (action != null)

{

return DeleteFurniture(model.Id);

}

var names = db.Furniture.Select(item => item.Name);

ViewData["Message"] = "";

model.Furniture = db.Furniture.ToList();

model.Ids = db.Furniture.Select(item => item.Id).ToList();

if (model.Name == null || model.Description == null || model.Material == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Furniture/Index.cshtml", model);

}

if (names.Contains(model.Name) || model.Name.Length == 0 || model.Name.Length > 50)

{

ViewData["Message"] += "Неправильный ввод названия";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Description.Length == 0 || model.Description.Length > 200)

{

ViewData["Message"] += "Неправильный ввод описания";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Material.Length == 0 || model.Material.Length > 60)

{

ViewData["Message"] += "Неправильный ввод материала";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Furniture/Index.cshtml", model);

}

else if (model.Count <= 0)

{

ViewData["Message"] += "Неправильное значение количества";

return View("~/Views/Furniture/Index.cshtml", model);

}

else

{

var furniture = db.Furniture.Where(item => item.Id == model.Id).FirstOrDefault();

furniture.Name = model.Name;

furniture.Description = model.Description;

furniture.Material = model.Material;

furniture.Price = model.Price;

furniture.Count = model.Count;

db.SaveChanges();

cache.Remove("Furniture");

cache.Set("Furniture", db.Furniture.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

return RedirectToAction("Index", "Furniture");

}

}

}

}

*OrderController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Caching.Memory;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CourseProject.Controllers

{

public class OrderController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

private IMemoryCache cache;

public OrderController(ApplicationContext applicationContext, IMemoryCache cache)

{

db = applicationContext;

this.cache = cache;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult Index(int page = 1, string furniture = "Все", string clientName = "Все", string type = null)

{

return View(GetViewModel(page, furniture, clientName, type));

}

[Authorize(Roles = "Администратор, Работник фабрики")]

[HttpPost]

public IActionResult AddOrder(OrderIndexViewModel model)

{

ViewData["Message"] = "";

if (model.FurnitureCount <= 0)

{

ViewData["Message"] += "Неправильное значение количества мебели";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else if (model.DiscountPercent < 0 || model.DiscountPercent > 99)

{

ViewData["Message"] += "Неправильное значение скидки";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else

{

var id = 0;

if (db.Orders.Count() != 0)

{

id = db.Orders.Select(item => item.Id).Max();

}

id++;

db.Orders.Add(new Order()

{

Id = id,

ClientId = db.Clients.Where(item => item.Name == model.ClientName).First().Id,

DiscountPercent = model.DiscountPercent,

FurnitureCount = model.FurnitureCount,

Price = model.Price,

IsCompleted = model.IsCompleted ? 1 : 0,

EmployeeId = db.Employees.Where(item => item.FIO == model.EmployeeFIO).First().Id,

FurnitureId = db.Furniture.Where(item => item.Name == model.FurnitureName).First().Id

});

db.SaveChanges();

cache.Remove("Orders");

return RedirectToAction("Index", "Order");

}

}

[Authorize(Roles = "Администратор, Работник фабрики")]

[HttpPost]

public IActionResult DeleteOrder(OrderIndexViewModel model)

{

ViewData["Message"] = "";

var order = db.Orders.Where(item => item.Id == model.Id).FirstOrDefault();

db.Orders.Remove(order);

db.SaveChanges();

cache.Remove("Orders");

return RedirectToAction("Index", "Order");

}

[Authorize(Roles = "Администратор, Работник фабрики")]

[HttpPost]

public IActionResult UpdateOrder(OrderIndexViewModel model)

{

ViewData["Message"] = "";

if (model.FurnitureCount <= 0)

{

ViewData["Message"] += "Неправильное значение количества мебели";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else if (model.DiscountPercent < 0 || model.DiscountPercent > 99)

{

ViewData["Message"] += "Неправильное значение скидки";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Order/Index.cshtml", GetViewModel());

}

else

{

var order = db.Orders.Where(item => item.Id == model.Id).FirstOrDefault();

order.DiscountPercent = model.DiscountPercent;

order.ClientId = db.Clients.Where(item => item.Name == model.ClientName).First().Id;

order.EmployeeId = db.Employees.Where(item => item.FIO == model.EmployeeFIO).First().Id;

order.Price = model.Price;

order.FurnitureCount = model.FurnitureCount;

order.FurnitureId = db.Furniture.Where(item => item.Name == model.FurnitureName).First().Id;

order.IsCompleted = model.IsCompleted ? 1 : 0;

db.SaveChanges();

cache.Remove("Orders");

return RedirectToAction("Index", "Order");

}

}

private OrderIndexViewModel GetViewModel(int page = 1, string furnitureName = "Все", string clientName = "Все", string type = null)

{

int pageSize = 20;

List<Employee> employees = db.Employees.ToList();

List<Furniture> furniture = db.Furniture.ToList();

List<Client> clients = db.Clients.ToList();

List<OrderViewModel> orderViewModels;

if (!cache.TryGetValue("Orders", out orderViewModels))

{

List<Order> orders = db.Orders.ToList();

orderViewModels = new List<OrderViewModel>();

foreach (var order in orders)

{

orderViewModels.Add(new OrderViewModel()

{

Id = order.Id,

ClientName = clients.Where(item => item.Id == order.ClientId).First().Name,

DiscountPercent = order.DiscountPercent,

EmployeeFIO = employees.Where(item => item.Id == order.EmployeeId).First().FIO,

FurnitureCount = order.FurnitureCount,

FurnitureName = furniture.Where(item => item.Id == order.FurnitureId).First().Name,

IsCompleted = order.IsCompleted != 0,

Price = order.Price

});

}

cache.Set("Orders", db.Orders.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

}

List<int> Ids = orderViewModels.Select(item => item.Id).ToList();

List<string> furnituresNames = furniture.Select(item => item.Name).ToList();

List<string> clientNames = clients.Select(item => item.Name).ToList();

furnituresNames.Add("Все");

clientNames.Add("Все");

if (furnitureName != "Все")

{

orderViewModels = orderViewModels.Where(item => item.FurnitureName == furnitureName).ToList();

}

if (clientName != "Все")

{

orderViewModels = orderViewModels.Where(item => item.ClientName == clientName).ToList();

}

if (type != null)

{

orderViewModels = type switch

{

"Id" => orderViewModels.OrderBy(item => item.Id).ToList(),

"client" => orderViewModels.OrderBy(item => item.ClientName).ToList(),

"furnit" => orderViewModels.OrderBy(item => item.FurnitureName).ToList(),

"count" => orderViewModels.OrderBy(item => item.FurnitureCount).ToList(),

"price" => orderViewModels.OrderBy(item => item.Price).ToList(),

"disc" => orderViewModels.OrderBy(item => item.DiscountPercent).ToList(),

"isComp" => orderViewModels.OrderBy(item => item.IsCompleted).ToList(),

\_ => orderViewModels.OrderBy(item => item.EmployeeFIO).ToList(),

};

}

OrderIndexViewModel orderIndexViewModel = new OrderIndexViewModel()

{

OrderViewModels = orderViewModels.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),

Ids = Ids,

ClientNames = clients.Select(item => item.Name).ToList(),

EmployeesFIOs = employees.Select(item => item.FIO).ToList(),

FurnitureNames = furniture.Select(item => item.Name).ToList(),

PageViewModel = new PageViewModel(orderViewModels.Count, page, pageSize),

FilterClients = clientNames,

FilterFurnitures = furnituresNames

};

return orderIndexViewModel;

}

}

}

*WaybillController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Caching.Memory;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CourseProject.Controllers

{

public class WaybillController : Controller

{

// Объект контекста данных

private readonly ApplicationContext db;

private IMemoryCache cache;

public WaybillController(ApplicationContext applicationContext, IMemoryCache cache)

{

db = applicationContext;

this.cache = cache;

}

// Метод получения страницы клиентов.

// Данная страница кэшируется на 286 секунд.

[ResponseCache(CacheProfileName = "TablesCaching")]

public IActionResult Index(int page = 1, string providerName = null, string furnitName = "Все", string type = null)

{

return View(GetViewModel(page, providerName, furnitName, type));

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult AddWaybill(WaybillIndexViewModel model)

{

ViewData["Message"] = "";

if (model.ProviderName == null || model.Material == null || model.DateOfSupply == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.ProviderId <= 0)

{

ViewData["Message"] += "Неправильное значение номера поставщика";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Weight <= 0)

{

ViewData["Message"] += "Неправильное значение веса";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.ProviderName.Length > 100)

{

ViewData["Message"] += "Неправильный ввод названия поставщика";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Material.Length > 60)

{

ViewData["Message"] += "Неправильный ввод материала";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else

{

var id = 0;

if (db.Waybills.Count() != 0)

{

id = db.Waybills.Select(item => item.Id).Max();

}

id++;

db.Waybills.Add(new Waybill()

{

Id = id,

ProviderId = model.ProviderId,

ProviderName = model.ProviderName,

Price = model.Price,

DateOfSupply = model.DateOfSupply,

EmployeeId = db.Employees.Where(item => item.FIO == model.EmployeeFIO).First().Id,

FurnitureId = db.Furniture.Where(item => item.Name == model.FurnitureName).First().Id,

Material = model.Material,

Weight = model.Weight

});

db.SaveChanges();

cache.Remove("Waybills");

return RedirectToAction("Index", "Waybill");

}

}

[Authorize(Roles = "Администратор")]

public IActionResult DeleteWaybill(int Id)

{

ViewData["Message"] = "";

var waybill = db.Waybills.Where(item => item.Id == Id).FirstOrDefault();

db.Waybills.Remove(waybill);

db.SaveChanges();

cache.Remove("Waybills");

return RedirectToAction("Index", "Waybill");

}

[Authorize(Roles = "Администратор")]

[HttpPost]

public IActionResult UpdateWaybill(WaybillIndexViewModel model, string action = null)

{

ViewData["Message"] = "";

if (action != null)

{

return DeleteWaybill(model.Id);

}

if (model.ProviderName == null || model.Material == null || model.DateOfSupply == null)

{

ViewData["Message"] += "Отсутствие значений в строках";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.ProviderId <= 0)

{

ViewData["Message"] += "Неправильное значение номера поставщика";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Weight <= 0)

{

ViewData["Message"] += "Неправильное значение веса";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Price <= 0)

{

ViewData["Message"] += "Неправильное значение стоимости";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.ProviderName.Length > 100)

{

ViewData["Message"] += "Неправильный ввод названия поставщика";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else if (model.Material.Length > 60)

{

ViewData["Message"] += "Неправильный ввод материала";

return View("~/Views/Waybill/Index.cshtml", GetViewModel());

}

else

{

var waybill = db.Waybills.Where(item => item.Id == model.Id).FirstOrDefault();

waybill.ProviderId = model.ProviderId;

waybill.ProviderName = model.ProviderName;

waybill.Price = model.Price;

waybill.DateOfSupply = model.DateOfSupply;

waybill.EmployeeId = db.Employees.Where(item => item.FIO == model.EmployeeFIO).First().Id;

waybill.FurnitureId = db.Furniture.Where(item => item.Name == model.FurnitureName).First().Id;

waybill.Material = model.Material;

waybill.Weight = model.Weight;

db.SaveChanges();

cache.Remove("Waybills");

return RedirectToAction("Index", "Waybill");

}

}

private WaybillIndexViewModel GetViewModel(int page = 1, string providerName = null, string furnitName = "Все", string type = null)

{

int pageSize = 20;

List<Employee> employees = db.Employees.ToList();

List<Furniture> furniture = db.Furniture.ToList();

List<WaybillViewModel> waybillViewModels;

if (!cache.TryGetValue("Waybills", out waybillViewModels))

{

List<Waybill> waybills = db.Waybills.ToList();

waybillViewModels = new List<WaybillViewModel>();

foreach (var waybill in waybills)

{

waybillViewModels.Add(new WaybillViewModel()

{

Id = waybill.Id,

ProviderId = waybill.ProviderId,

ProviderName = waybill.ProviderName,

EmployeeFIO = employees.Where(item => item.Id == waybill.EmployeeId).First().FIO,

DateOfSupply = waybill.DateOfSupply,

FurnitureName = furniture.Where(item => item.Id == waybill.FurnitureId).First().Name,

Material = waybill.Material,

Price = waybill.Price,

Weight = waybill.Weight

});

}

cache.Set("Waybills", db.Waybills.ToList(), new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromMinutes(10)));

}

List<int> Ids = waybillViewModels.Select(item => item.Id).ToList();

List<string> furnitNames = furniture.Select(item => item.Name).ToList();

furnitNames.Add("Все");

if (providerName != null)

{

waybillViewModels = waybillViewModels.Where(item => item.ProviderName.Contains(providerName)).ToList();

}

if (furnitName != "Все")

{

waybillViewModels = waybillViewModels.Where(item => item.FurnitureName == furnitName).ToList();

}

if (type != null)

{

waybillViewModels = type switch

{

"Id" => waybillViewModels.OrderBy(item => item.Id).ToList(),

"numb" => waybillViewModels.OrderBy(item => item.ProviderId).ToList(),

"date" => waybillViewModels.OrderBy(item => item.DateOfSupply).ToList(),

"providerName" => waybillViewModels.OrderBy(item => item.ProviderName).ToList(),

"material" => waybillViewModels.OrderBy(item => item.Material).ToList(),

"price" => waybillViewModels.OrderBy(item => item.Price).ToList(),

"weight" => waybillViewModels.OrderBy(item => item.Weight).ToList(),

"furnit" => waybillViewModels.OrderBy(item => item.FurnitureName).ToList(),

\_ => waybillViewModels.OrderBy(item => item.EmployeeFIO).ToList(),

};

}

WaybillIndexViewModel waybillIndexViewModel = new WaybillIndexViewModel()

{

WaybillViewModels = waybillViewModels.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),

Ids = Ids,

EmployeesFIOs = employees.Select(item => item.FIO).ToList(),

FurnitureNames = furniture.Select(item => item.Name).ToList(),

PageViewModel = new PageViewModel(waybillViewModels.Count, page, pageSize),

FilterFurnitureNames = furnitNames

};

return waybillIndexViewModel;

}

}

}

*UserManageController.cs*

using CourseProject.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Linq;

using System.Threading.Tasks;

namespace CourseProject.Controllers

{

[Authorize(Roles = "Администратор")]

public class UserManageController : Controller

{

UserManager<User> \_userManager;

RoleManager<IdentityRole> \_roleManager;

public UserManageController(UserManager<User> userManager, RoleManager<IdentityRole> roleManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

}

public IActionResult Index()

{

return View(\_userManager.Users.ToList());

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(CreateUserViewModel model)

{

if (ModelState.IsValid)

{

User user = new User { Email = model.Email, UserName = model.Email, Age = model.Age, FIO = model.FIO };

var result = await \_userManager.CreateAsync(user, model.Password);

if (!await \_roleManager.RoleExistsAsync(model.UserRole))

{

await \_roleManager.CreateAsync(new IdentityRole(model.UserRole));

}

await \_userManager.AddToRoleAsync(user, model.UserRole);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

public async Task<IActionResult> Edit(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

return NotFound();

}

EditUserViewModel model = new EditUserViewModel { Id = user.Id, Email = user.Email, Age = user.Age, FIO = user.FIO };

return View(model);

}

[HttpPost]

public async Task<IActionResult> Edit(EditUserViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.FindByIdAsync(model.Id);

if (user != null)

{

user.Email = model.Email;

user.UserName = model.Email;

user.Age = model.Age;

user.FIO = model.FIO;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

}

return View(model);

}

[HttpPost]

public async Task<ActionResult> Delete(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user != null)

{

await \_userManager.DeleteAsync(user);

}

return RedirectToAction("Index");

}

}

}

*Home/Index.cshtml*

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">История фабрики</h1>

<h2>Истории пока нет, но скоро будет.</h2>

</div>

*Account/Register.cshtml*

@model CourseProject.Models.RegisterViewModel

<h2 style="margin-top:15px;">Регистрация нового пользователя</h2>

<form method="post" **asp-controller**="Account" **asp-action**="Register">

<div **asp-validation-summary**="ModelOnly"></div>

<div>

<label **asp-for**="UserRole"></label><br />

<select **asp-for**="UserRole">

<option>Базовый пользователь</option>

<option>Работник фабрики</option>

<option>Администратор</option>

</select>

</div>

<div>

<label **asp-for**="Email"></label><br />

<input **asp-for**="Email" />

<span **asp-validation-for**="Email"></span>

</div>

<div>

<label **asp-for**="FIO"></label><br />

<input **asp-for**="FIO" />

<span **asp-validation-for**="FIO"></span>

</div>

<div>

<label **asp-for**="Age"></label><br />

<input **asp-for**="Age" />

<span **asp-validation-for**="Age"></span>

</div>

<div>

<label **asp-for**="Password"></label><br />

<input **asp-for**="Password" />

<span **asp-validation-for**="Password"></span>

</div>

<div>

<label **asp-for**="PasswordConfirm"></label><br />

<input **asp-for**="PasswordConfirm" />

<span **asp-validation-for**="PasswordConfirm"></span>

</div>

<hr />

<div>

<input type="submit" value="Регистрация" />

</div>

</form>

*Account/Login.cshtml*

@model CourseProject.Models.LoginViewModel

<h2 style="margin-top:15px;">Вход в приложение</h2>

<form method="post" **asp-controller**="Account" **asp-action**="Login"

**asp-route-returnUrl**="@Model.ReturnUrl">

<div **asp-validation-summary**="ModelOnly"></div>

<div>

<label **asp-for**="Email"></label><br />

<input **asp-for**="Email" />

<span **asp-validation-for**="Email"></span>

</div>

<div>

<label **asp-for**="Password"></label><br />

<input **asp-for**="Password" />

<span **asp-validation-for**="Password"></span>

</div>

<div>

<label **asp-for**="RememberMe"></label><br />

<input **asp-for**="RememberMe" />

</div>

<div>

<input type="submit" value="Войти" />

</div>

</form>

*Client/Index.cshtml*

@model ClientIndexViewModel

@{

ViewBag.Title = "Таблица клиентов";

}

<script>

function changeAddVisibility() {

document.getElementById('UpdateClient').style.display = "none";

document.getElementById('AddClient').style.display = "block";

}

function changeUpdateVisibility(id) {

document.getElementById('UpdateClient').style.display = "block";

document.getElementById('AddClient').style.display = "none";

$("#id").val(id);

}

function SetSession() {

sessionStorage.setItem("address", $("#value1").val());

sessionStorage.setItem("filterRepresFIO", $("#value2 option:selected").val());

}

function GetSession() {

$("#value1").val(sessionStorage.getItem("address"));

$("#value2").val(sessionStorage.getItem("filterRepresFIO"));

}

</script>

<H1 style="margin-top:15px;">@ViewBag.Title:</H1>

<body onbeforeunload="SetSession()" onload="GetSession()">

<form method="get">

<div>

<label class="control-label">Адрес:&nbsp;</label>

<input name="address" id="value1" class="form-control" />

</div>

<div>

<label class="control-label">ФИО представителя:&nbsp;</label>

<select name="filterRepresFIO" id="value2" class="form-control">

@foreach (var elem in Model.FilterRepresFIOs)

{

<option>@elem</option>

}

</select>

</div>

<br />

<div>

<input type="submit" value="Фильтр" class="btn btn-default" />

</div>

</form>

<br />

<table border="1" style="left:10%;">

<thead>

<tr>

<td>

<a **asp-action**="Index" **asp-route-type**="Id">Id</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="name">Название</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="fio">ФИО представителя</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="numb">Номер</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="addr">Адрес</a>

</td>

</tr>

</thead>

@foreach (var elem in Model.Clients)

{

<tr>

<td>

@if (User.IsInRole("Администратор"))

{

<input type="button" class="btn-default" style="border:hidden; width:100%; background-color: rgba(0,100,0,0.9); color:white;" value="@elem.Id" onclick="changeUpdateVisibility(@elem.Id)" />

}

else

{

@elem.Id

}

</td>

<td>

@elem.Name

</td>

<td>

@elem.RepresentativeFIO

</td>

<td>

@elem.Number

</td>

<td>

@elem.Address

</td>

</tr>

}

</table>

<div>

@if (Model.PageViewModel.HasPreviousPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber - 1)"

class="btn" style="border:2px solid white;">

<i class="glyphicon glyphicon-chevron-left"></i>

Назад

</a>

}

@if (Model.PageViewModel.HasNextPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber + 1)"

class="btn" style="border:2px solid white;">

Вперед

<i class="glyphicon glyphicon-chevron-right"></i>

</a>

}

</div>

<br />

@if (User.IsInRole("Администратор"))

{

<div>

<input type="submit" class="btn-default" value="Добавить запись" onclick="changeAddVisibility()" />

</div>

<hr />

@ViewData["Message"]

<div>

<div id="AddClient" style="display:none;">

@{await Html.RenderPartialAsync("\_AddClient");}

</div>

<div id="UpdateClient" style="display:none;">

@{await Html.RenderPartialAsync("\_UpdateClient");}

</div>

</div>

}

</body>

*Client/\_AddClient.cshtml*

@model ClientIndexViewModel

<h3 style="margin-top:15px;">

Добавление записи

</h3>

<hr />

@using (Html.BeginForm("AddClient", "Client", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Name)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Name)

</div>

</div>

<div>

@Html.LabelFor(m => m.RepresentativeFIO)

<div class="col-md-10">

@Html.TextBoxFor(m => m.RepresentativeFIO)

</div>

</div>

<div>

@Html.LabelFor(m => m.Number)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Number)

</div>

</div>

<div>

@Html.LabelFor(m => m.Address)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Address)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Добавить" />

}

*Client/\_UpdateClient.cshtml*

@model ClientIndexViewModel

<h3 style="margin-top:15px;">

Изменение записи

</h3>

@using (Html.BeginForm("UpdateClient", "Client", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Id)

<div class="col-md-10">

<select name="Id" id="id">

@foreach (var id in Model.Ids)

{

<option>@id</option>

}

</select>

&nbsp;

<input name="action" type="submit" style="border:hidden; width:200px" value="Удалить клиента" />

</div>

</div>

<div>

@Html.LabelFor(m => m.Name)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Name)

</div>

</div>

<div>

@Html.LabelFor(m => m.RepresentativeFIO)

<div class="col-md-10">

@Html.TextBoxFor(m => m.RepresentativeFIO)

</div>

</div>

<div>

@Html.LabelFor(m => m.Number)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Number)

</div>

</div>

<div>

@Html.LabelFor(m => m.Address)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Address)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Изменить" />

}

*Employee/Index.cshtml*

@model EmployeeIndexViewModel

@{

ViewBag.Title = "Таблица работников";

}

<script>

function changeAddVisibility() {

document.getElementById('UpdateEmployee').style.display = "none";

document.getElementById('AddEmployee').style.display = "block";

}

function changeUpdateVisibility(id) {

document.getElementById('UpdateEmployee').style.display = "block";

document.getElementById('AddEmployee').style.display = "none";

$("#id").val(id);

}

function SetSession() {

sessionStorage.setItem("education", $("#value1").val());

sessionStorage.setItem("position", $("#value2 option:selected").val());

}

function GetSession() {

$("#value1").val(sessionStorage.getItem("education"));

$("#value2").val(sessionStorage.getItem("position"));

}

</script>

<body onbeforeunload="SetSession()" onload="GetSession()">

<H1 style="margin-top:15px;">@ViewBag.Title:</H1>

<form method="get">

<div>

<label class="control-label">Образование:&nbsp;</label>

<input name="education" id="value1" class="form-control" />

</div>

<div>

<label class="control-label">Должностm:&nbsp;</label>

<select name="position" id="value2" class="form-control">

@foreach (var elem in Model.FilterPositions)

{

<option>@elem</option>

}

</select>

</div>

<br />

<div>

<input type="submit" value="Фильтр" class="btn btn-default" />

</div>

</form>

<br />

<div>

<table border="1">

<thead>

<tr>

<td>

<a **asp-action**="Index" **asp-route-type**="Id">Id</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="fio">ФИО</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="pos">Должность</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="educ">Образование</a>

</td>

</tr>

</thead>

@foreach (var elem in Model.Employees)

{

<tr>

<td>

@if (User.IsInRole("Администратор"))

{

<input type="button" class="btn-default" style="border:hidden; width:100%; background-color: rgba(0,100,0,0.9); color:white;" value="@elem.Id" onclick="changeUpdateVisibility(@elem.Id)" />

}

else

{

@elem.Id

}

</td>

<td>

@elem.FIO

</td>

<td>

@elem.Position

</td>

<td>

@elem.Education

</td>

</tr>

}

</table>

</div>

<div>

@if (Model.PageViewModel.HasPreviousPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber - 1)"

class="btn" style="border:2px solid white;">

<i class="glyphicon glyphicon-chevron-left"></i>

Назад

</a>

}

@if (Model.PageViewModel.HasNextPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber + 1)"

class="btn" style="border:2px solid white;">

Вперед

<i class="glyphicon glyphicon-chevron-right"></i>

</a>

}

</div>

<br />

@if (User.IsInRole("Администратор"))

{

<div>

<input type="submit" class="btn-default" value="Добавить запись" onclick="changeAddVisibility()" />

</div>

<hr />

@ViewData["Message"]

<div>

<div id="AddEmployee" style="display:none;">

@{await Html.RenderPartialAsync("\_AddEmployee");}

</div>

<div id="UpdateEmployee" style="display:none;">

@{await Html.RenderPartialAsync("\_UpdateEmployee");}

</div>

</div>

}

</body>

*Employee/\_AddEmployee.cshtml*

@model EmployeeIndexViewModel

<h3 style="margin-top:15px;">

Добавление записи

</h3>

<hr />

@using (Html.BeginForm("AddEmployee", "Employee", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.FIO)

<div class="col-md-10">

@Html.TextBoxFor(m => m.FIO)

</div>

</div>

<div>

@Html.LabelFor(m => m.Position)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Position)

</div>

</div>

<div>

@Html.LabelFor(m => m.Education)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Education)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Добавить" />

}

*Employee/\_UpdateEmployee.cshtml*

@model EmployeeIndexViewModel

<h3 style="margin-top:15px;">

Изменение записи

</h3>

@using (Html.BeginForm("UpdateEmployee", "Employee", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Id)

<div class="col-md-10">

<select name="Id" id="id">

@foreach (var id in Model.Ids)

{

<option>@id</option>

}

</select>

&nbsp;

<input name="action" type="submit" style="border:hidden; width:200px" value="Удалить работника" />

</div>

</div>

<div>

@Html.LabelFor(m => m.FIO)

<div class="col-md-10">

@Html.TextBoxFor(m => m.FIO)

</div>

</div>

<div>

@Html.LabelFor(m => m.Position)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Position)

</div>

</div>

<div>

@Html.LabelFor(m => m.Education)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Education)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Изменить" />

}

*Furniture/Index.cshtml*

@model FurnitureIndexViewModel

@{

ViewBag.Title = "Таблица мебели";

}

<script>

function changeAddVisibility() {

document.getElementById('UpdateFurniture').style.display = "none";

document.getElementById('AddFurniture').style.display = "block";

}

function changeUpdateVisibility(id) {

document.getElementById('UpdateFurniture').style.display = "block";

document.getElementById('AddFurniture').style.display = "none";

$("#id").val(id);

}

function SetSession() {

sessionStorage.setItem("price", $("#value1").val());

sessionStorage.setItem("material", $("#value2 option:selected").val());

}

function GetSession() {

$("#value1").val(sessionStorage.getItem("price"));

$("#value2").val(sessionStorage.getItem("material"));

}

</script>

<body onbeforeunload="SetSession()" onload="GetSession()">

<H1 style="margin-top:15px;">@ViewBag.Title:</H1>

<form method="get">

<div>

<label class="control-label">Стоимость:&nbsp;</label>

<input name="price" type="number" min="0" id="value1" class="form-control" />

</div>

<div>

<label class="control-label">Материалы:&nbsp;</label>

<select name="material" id="value2" class="form-control">

@foreach (var elem in Model.FilterMaterials)

{

<option>@elem</option>

}

</select>

</div>

<br />

<div>

<input type="submit" value="Фильтр" class="btn btn-default" />

</div>

</form>

<br />

<div>

<table border="1">

<thead>

<tr>

<td>

<a **asp-action**="Index" **asp-route-type**="Id">Id</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="name">Название</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="descr">Описание</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="material">Материал</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="price">Цена</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="count">Количество</a>

</td>

</tr>

</thead>

@foreach (var elem in Model.Furniture)

{

<tr>

<td>

@if (User.IsInRole("Администратор") || User.IsInRole("Работник фабрики"))

{

<input type="button" class="btn-default" style="border:hidden; width:100%; background-color: rgba(0,100,0,0.9); color:white;" value="@elem.Id" onclick="changeUpdateVisibility(@elem.Id)" />

}

else

{

@elem.Id

}

</td>

<td>

@elem.Name

</td>

<td>

@elem.Description

</td>

<td>

@elem.Material

</td>

<td>

@elem.Price

</td>

<td>

@elem.Count

</td>

</tr>

}

</table>

</div>

<div>

@if (Model.PageViewModel.HasPreviousPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber - 1)"

class="btn" style="border:2px solid white;">

<i class="glyphicon glyphicon-chevron-left"></i>

Назад

</a>

}

@if (Model.PageViewModel.HasNextPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber + 1)"

class="btn" style="border:2px solid white;">

Вперед

<i class="glyphicon glyphicon-chevron-right"></i>

</a>

}

</div>

<br />

@if (User.IsInRole("Администратор") || User.IsInRole("Работник фабрики"))

{

<div>

<input type="submit" class="btn-default" value="Добавить запись" onclick="changeAddVisibility()" />

</div>

<hr />

@ViewData["Message"]

<div>

<div id="AddFurniture" style="display:none;">

@{await Html.RenderPartialAsync("\_AddFurniture");}

</div>

<div id="UpdateFurniture" style="display:none;">

@{await Html.RenderPartialAsync("\_UpdateFurniture");}

</div>

</div>

}

</body>

*Furniture/\_AddFurniture.cshtml*

@model FurnitureIndexViewModel

<h3 style="margin-top:15px;">

Добавление записи

</h3>

<hr />

@using (Html.BeginForm("AddFurniture", "Furniture", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Name)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Name)

</div>

</div>

<div>

@Html.LabelFor(m => m.Description)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Description)

</div>

</div>

<div>

@Html.LabelFor(m => m.Material)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Material)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.Count)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Count)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Добавить" />

}

*Furniture/\_UpdateFurniture.cshtml*

@model FurnitureIndexViewModel

<h3 style="margin-top:15px;">

Изменение записи

</h3>

@using (Html.BeginForm("UpdateFurniture", "Furniture", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Id)

<div class="col-md-10">

<select name="Id" id="id">

@foreach (var id in Model.Ids)

{

<option>@id</option>

}

</select>

&nbsp;

<input name="action" type="submit" style="border:hidden; width:200px" value="Удалить мебель" />

</div>

</div>

<div>

@Html.LabelFor(m => m.Name)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Name)

</div>

</div>

<div>

@Html.LabelFor(m => m.Description)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Description)

</div>

</div>

<div>

@Html.LabelFor(m => m.Material)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Material)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.Count)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Count)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Изменить" />

}

*Order/Index.cshtml*

@model OrderIndexViewModel

@{

ViewBag.Title = "Таблица заказов";

}

<script>

function changeAddVisibility() {

document.getElementById('UpdateOrder').style.display = "none";

document.getElementById('AddOrder').style.display = "block";

}

function changeUpdateVisibility(id) {

document.getElementById('UpdateOrder').style.display = "block";

document.getElementById('AddOrder').style.display = "none";

$("#id").val(id);

}

function SetSession() {

sessionStorage.setItem("clientName", $("#value1 option:selected").val());

sessionStorage.setItem("furniture", $("#value2 option:selected").val());

}

function GetSession() {

$("#value1").val(sessionStorage.getItem("clientName"));

$("#value2").val(sessionStorage.getItem("furniture"));

}

</script>

<body onbeforeunload="SetSession()" onload="GetSession()">

<H1 style="margin-top:15px;">@ViewBag.Title:</H1>

<form method="get">

<div>

<label class="control-label">Название клиента:&nbsp;</label>

<select name="clientName" id="value1" class="form-control">

@foreach (var elem in Model.FilterClients)

{

<option>@elem</option>

}

</select>

</div>

<div>

<label class="control-label">Название мебели:&nbsp;</label>

<select name="furniture" id="value2" class="form-control">

@foreach (var elem in Model.FilterFurnitures)

{

<option>@elem</option>

}

</select>

</div>

<br />

<div>

<input type="submit" value="Фильтр" class="btn btn-default" />

</div>

</form>

<br />

<div>

<table border="1">

<thead>

<tr>

<td>

<a **asp-action**="Index" **asp-route-type**="Id">Id</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="client">Название клиента</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="furnit">Название мебели</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="count">Кол-во мебели</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="price">Цена</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="disc">Скидка</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="isComp">Выполнен?</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="empl">ФИО сотрудника</a>

</td>

</tr>

</thead>

@foreach (var elem in Model.OrderViewModels)

{

<tr>

<td>

@if (User.IsInRole("Администратор") || User.IsInRole("Работник фабрики"))

{

<input type="button" class="btn-default" style="border:hidden; width:100%; background-color: rgba(0,100,0,0.9); color:white;" value="@elem.Id" onclick="changeUpdateVisibility(@elem.Id)" />

}

else

{

@elem.Id

}

</td>

<td>

@elem.ClientName

</td>

<td>

@elem.FurnitureName

</td>

<td>

@elem.FurnitureCount

</td>

<td>

@elem.Price

</td>

<td>

@elem.DiscountPercent

%

</td>

@if (elem.IsCompleted)

{

<td>

Да

</td>

}

else

{

<td>

Нет

</td>

}

<td>

@elem.EmployeeFIO

</td>

</tr>

}

</table>

</div>

<div>

@if (Model.PageViewModel.HasPreviousPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber - 1)"

class="btn" style="border:2px solid white;">

<i class="glyphicon glyphicon-chevron-left"></i>

Назад

</a>

}

@if (Model.PageViewModel.HasNextPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber + 1)"

class="btn" style="border:2px solid white;">

Вперед

<i class="glyphicon glyphicon-chevron-right"></i>

</a>

}

</div>

<br />

@if (User.IsInRole("Администратор") || User.IsInRole("Работник фабрики"))

{

<div>

<input type="submit" class="btn-default" value="Добавить запись" onclick="changeAddVisibility()" />

</div>

<hr />

@ViewData["Message"]

<div>

<div id="AddOrder" style="display:none;">

@{await Html.RenderPartialAsync("\_AddOrder");}

</div>

<div id="UpdateOrder" style="display:none;">

@{await Html.RenderPartialAsync("\_UpdateOrder");}

</div>

</div>

}

</body>

*Order/\_AddOrder.cshtml*

@model OrderIndexViewModel

<h3 style="margin-top:15px;">

Добавление записи

</h3>

<hr />

@using (Html.BeginForm("AddOrder", "Order", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.ClientName)

<div class="col-md-10">

<select name="ClientName">

@foreach (var name in Model.ClientNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.EmployeeFIO)

<div class="col-md-10">

<select name="EmployeeFIO">

@foreach (var fio in Model.EmployeesFIOs)

{

<option>@fio</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureName)

<div class="col-md-10">

<select name="FurnitureName">

@foreach (var name in Model.FurnitureNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureCount)

<div class="col-md-10">

@Html.TextBoxFor(m => m.FurnitureCount)

</div>

</div>

<div>

@Html.LabelFor(m => m.DiscountPercent)

<div class="col-md-10">

@Html.TextBoxFor(m => m.DiscountPercent)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.IsCompleted)

<div class="col-md-10">

@Html.CheckBoxFor(m => m.IsCompleted)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Добавить" />

}

*Order/\_UpdateOrder.cshtml*

@model OrderIndexViewModel

<h3 style="margin-top:15px;">

Изменение записи

</h3>

@using (Html.BeginForm("UpdateOrder", "Order", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Id)

<div class="col-md-10">

<select name="Id" id="id">

@foreach (var id in Model.Ids)

{

<option>@id</option>

}

</select>

&nbsp;

<input name="action" type="submit" style="border:hidden; width:200px" value="Удалить заказ" />

</div>

</div>

<div>

@Html.LabelFor(m => m.ClientName)

<div class="col-md-10">

<select name="ClientName">

@foreach (var name in Model.ClientNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.EmployeeFIO)

<div class="col-md-10">

<select name="EmployeeFIO">

@foreach (var fio in Model.EmployeesFIOs)

{

<option>@fio</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureName)

<div class="col-md-10">

<select name="FurnitureName">

@foreach (var name in Model.FurnitureNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureCount)

<div class="col-md-10">

@Html.TextBoxFor(m => m.FurnitureCount)

</div>

</div>

<div>

@Html.LabelFor(m => m.DiscountPercent)

<div class="col-md-10">

@Html.TextBoxFor(m => m.DiscountPercent)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.IsCompleted)

<div class="col-md-10">

@Html.CheckBoxFor(m => m.IsCompleted)

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Изменить" />

}

*Shared/\_Layout.cshtml*

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - CourseProject</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/site.css" />

<link rel="stylesheet" href="~/css/MyStyle.css" />

</head>

<body>

<header>

<nav class="main navbar-expand-sm navbar-toggleable-sm navbar-light box-shadow mb-3">

<div>

<a class="navbar-brand" style="color:white;" **asp-controller**="Home" **asp-action**="Index">История</a>

</div>

<a class="nav-link" style="color:white;" **asp-area**="" **asp-controller**="Client" **asp-action**="Index">Клиенты</a>

<a class="nav-link" style="color:white;" **asp-area**="" **asp-controller**="Employee" **asp-action**="Index">Работники</a>

<a class="nav-link" style="color:white;" **asp-area**="" **asp-controller**="Furniture" **asp-action**="Index">Мебель</a>

<a class="nav-link" style="color:white;" **asp-area**="" **asp-controller**="Order" **asp-action**="Index">Заказы</a>

<a class="nav-link" style="color:white;" **asp-area**="" **asp-controller**="Waybill" **asp-action**="Index">Накладные</a>

<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">

<**partial** **name**="\_LoginPartial" />

</div>

</nav>

</header>

<div class="tables">

<main class="pb-3">

@RenderBody()

</main>

</div>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script **src**="~/js/site.js" **asp-append-version**="true"></script>

@RenderSection("Scripts", required: false)

</body>

</html>

*Shared/\_LoginPartial.cshtml*

@using Microsoft.AspNetCore.Identity

@inject SignInManager<User> SignInManager

@inject UserManager<User> UserManager

<ul class="navbar-nav">

@if (SignInManager.IsSignedIn(User))

{

if (User.IsInRole("Администратор"))

{

<li class="nav-item">

<a class="nav-link" style="color:white;" **asp-controller**="UserManage" **asp-action**="Index" title="Manage">Пользователи</a>

</li>

}

else

{

<li class="nav-item">

<label class="col-form-label">Здравствуйте, @User.Identity.Name</label>

</li>

}

<li class="nav-item">

<form class="form-inline" **asp-controller**="Account" **asp-action**="Logout">

<button type="submit" class="nav-link btn btn-link" style="color:white;">Выход</button>

</form>

</li>

}

else

{

<li class="nav-item">

<a class="nav-link" style="color:white;" **asp-controller**="Account" **asp-action**="Register">Регистрация</a>

</li>

<li class="nav-item">

<a class="nav-link" style="color:white;" **asp-controller**="Account" **asp-action**="Login">Вход</a>

</li>

}

</ul>

*Shared/\_ValidationScriptsPartial.cshtml*

<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>

*Shared/Error.cshtml*

@model ErrorViewModel

@{

ViewData["Title"] = "Error";

}

<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)

{

<p>

<strong>Request ID:</strong> <code>@Model.RequestId</code>

</p>

}

<h3>Development Mode</h3>

<p>

Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.

</p>

<p>

<strong>The Development environment shouldn't be enabled for deployed applications.</strong>

It can result in displaying sensitive information from exceptions to end users.

For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE\_ENVIRONMENT</strong> environment variable to <strong>Development</strong>

and restarting the app.

</p>

*UserManage/Create.cshtml*

@model CourseProject.Models.CreateUserViewModel

@{

ViewBag.Title = "Добавление пользователя";

}

<h1 style="margin-top:15px;">@ViewBag.Title</h1>

<form **asp-action**="Create" **asp-controller**="UserManage">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div>

<label **asp-for**="UserRole"></label><br />

<select **asp-for**="UserRole">

<option>Базовый пользователь</option>

<option>Работник фабрики</option>

<option>Администратор</option>

</select>

</div>

<div class="form-group">

<label **asp-for**="Email" class="control-label">Email</label>

<input **type**="text" **asp-for**="Email" class="form-control" />

</div>

<div class="form-group">

<label **asp-for**="Password" class="control-label">Пароль</label>

<input **type**="password" **asp-for**="Password" class="form-control" />

</div>

<div class="form-group">

<label **asp-for**="FIO" class="control-label">ФИО</label>

<input **type**="text" **asp-for**="FIO" class="form-control" />

</div>

<div class="form-group">

<label **asp-for**="Age" class="control-label">Возраст</label>

<input **type**="number" **asp-for**="Age" class="form-control" min="16" max="67" **value**="16" />

</div>

<div class="form-group">

<input type="submit" value="Добавить" class="btn btn-outline-secondary" />

</div>

</form>

*UserManage/Edit.cshtml*

@model CourseProject.Models.EditUserViewModel

@{

ViewBag.Title = "Редактирование пользователя";

}

<h1 style="margin-top:15px;">@ViewBag.Title</h1>

<form **asp-action**="Edit" **asp-controller**="UserManage">

<div **asp-validation-summary**="ModelOnly"></div>

<div class="form-group">

<input **type**="hidden" **asp-for**="Id" />

</div>

<div class="form-group">

<label **asp-for**="Email" class="control-label">Email</label>

<input **type**="text" **asp-for**="Email" class="form-control" />

</div>

<div class="form-group">

<label **asp-for**="FIO" class="control-label">ФИО</label>

<input **type**="text" **asp-for**="FIO" class="form-control" />

</div>

<div class="form-group">

<label **asp-for**="Age" class="control-label">Возраст</label>

<input **type**="number" **asp-for**="Age" class="form-control" min="16" max="67" **value**="16" />

</div>

<div class="form-group">

<input type="submit" value="Сохранить" class="btn btn-outline-secondary" />

</div>

</form>

*UserManage/Index.cshtml*

@model IEnumerable<CourseProject.Models.User>

@{

ViewBag.Title = "Список пользователей";

}

<div class="text-center">

<h1 style="margin-top:15px;">

Список пользователей

</h1>

<a **asp-action**="Create">Добавить пользователя</a>

<table align="center" border="1" style="border-color: black;">

<tr><th>Email</th><th>ФИО</th><th>Возраст</th></tr>

@foreach (var user in Model)

{

<tr>

<td>@user.Email</td>

<td>@user.FIO</td>

<td>@user.Age</td>

<td>

<form **asp-action**="Delete" **asp-route-id**="@user.Id" method="post">

<button type="submit" style="color:white;">

Удалить

</button>

</form>

</td>

<td>

<form **asp-action**="Edit" **asp-route-id**="@user.Id" method="get">

<button type="submit" style="color:white;">

Изменить

</button>

</form>

</td>

</tr>

}

</table>

</div>

*Waybill/Index.cshtml*

@model WaybillIndexViewModel

@{

ViewBag.Title = "Таблица накладных";

}

<script>

function changeAddVisibility() {

document.getElementById('UpdateWaybill').style.display = "none";

document.getElementById('AddWaybill').style.display = "block";

}

function changeUpdateVisibility(id) {

document.getElementById('UpdateWaybill').style.display = "block";

document.getElementById('AddWaybill').style.display = "none";

$("#id").val(id);

}

function SetSession() {

sessionStorage.setItem("providerName", $("#value1").val());

sessionStorage.setItem("furnitName", $("#value2 option:selected").val());

}

function GetSession() {

$("#value1").val(sessionStorage.getItem("providerName"));

$("#value2").val(sessionStorage.getItem("furnitName"));

}

</script>

<body onbeforeunload="SetSession()" onload="GetSession()">

<H1 style="margin-top:15px;">@ViewBag.Title:</H1>

<form method="get">

<div>

<label class="control-label">Название поставщика:&nbsp;</label>

<input type="text" name="providerName" id="value1" class="form-control" />

</div>

<div>

<label class="control-label">Название мебели:&nbsp;</label>

<select name="furnitName" id="value2" class="form-control">

@foreach (var elem in Model.FilterFurnitureNames)

{

<option>@elem</option>

}

</select>

</div>

<br />

<div>

<input type="submit" value="Фильтр" class="btn btn-default" />

</div>

</form>

<br />

<div>

<table border="1">

<thead>

<tr>

<td>

<a **asp-action**="Index" **asp-route-type**="Id">Id</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="numb">Номер поставщика</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="providerName">Название поставщика</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="date">Дата поставок</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="material">Материал</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="price">Стоимость</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="weight">Вес</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="furnit">Название мебели</a>

</td>

<td>

<a **asp-action**="Index" **asp-route-type**="empl">ФИО сотрудника</a>

</td>

</tr>

</thead>

@foreach (var elem in Model.WaybillViewModels)

{

<tr>

<td>

@if (User.IsInRole("Администратор"))

{

<input type="button" class="btn-default" style="border:hidden; width:100%; background-color: rgba(0,100,0,0.9); color:white;" value="@elem.Id" onclick="changeUpdateVisibility(@elem.Id)" />

}

else

{

@elem.Id

}

</td>

<td>

@elem.ProviderId

</td>

<td>

@elem.ProviderName

</td>

<td>

@elem.DateOfSupply

</td>

<td>

@elem.Material

</td>

<td>

@elem.Price

</td>

<td>

@elem.Weight

</td>

<td>

@elem.FurnitureName

</td>

<td>

@elem.EmployeeFIO

</td>

</tr>

}

</table>

</div>

<div>

@if (Model.PageViewModel.HasPreviousPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber - 1)"

class="btn" style="border:2px solid white;">

<i class="glyphicon glyphicon-chevron-left"></i>

Назад

</a>

}

@if (Model.PageViewModel.HasNextPage)

{

<a **asp-action**="Index"

**asp-route-page**="@(Model.PageViewModel.PageNumber + 1)"

class="btn" style="border:2px solid white;">

Вперед

<i class="glyphicon glyphicon-chevron-right"></i>

</a>

}

</div>

<br />

@if (User.IsInRole("Администратор"))

{

<div>

<input type="submit" class="btn-default" value="Добавить запись" onclick="changeAddVisibility()" />

</div>

<hr />

@ViewData["Message"]

<div>

<div id="AddWaybill" style="display:none;">

@{await Html.RenderPartialAsync("\_AddWaybill");}

</div>

<div id="UpdateWaybill" style="display:none;">

@{await Html.RenderPartialAsync("\_UpdateWaybill");}

</div>

</div>

}

</body>

*Waybill/\_AddWaybill.cshtml*

@model WaybillIndexViewModel

<h3 style="margin-top:15px;">

Добавление записи

</h3>

<hr />

@using (Html.BeginForm("AddWaybill", "Waybill", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.EmployeeFIO)

<div class="col-md-10">

<select name="EmployeeFIO">

@foreach (var fio in Model.EmployeesFIOs)

{

<option>@fio</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureName)

<div class="col-md-10">

<select name="FurnitureName">

@foreach (var name in Model.FurnitureNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.ProviderId)

<div class="col-md-10">

@Html.TextBoxFor(m => m.ProviderId)

</div>

</div>

<div>

@Html.LabelFor(m => m.ProviderName)

<div class="col-md-10">

@Html.TextBoxFor(m => m.ProviderName)

</div>

</div>

<div>

@Html.LabelFor(m => m.Material)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Material)

</div>

</div>

<div>

@Html.LabelFor(m => m.Weight)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Weight)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.DateOfSupply)

<div class="col-md-10">

<input type="date" id="datepicker" name="DateOfSupply" min="2000-01-01" max="2022-01-01" />

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Добавить" />

}

*Waybill/\_UpdateWaybill.cshtml*

@model WaybillIndexViewModel

<h3 style="margin-top:15px;">

Изменение записи

</h3>

@using (Html.BeginForm("UpdateWaybill", "Waybill", FormMethod.Post))

{

<div>

@Html.LabelFor(m => m.Id)

<div class="col-md-10">

<select name="Id" id="id">

@foreach (var id in Model.Ids)

{

<option>@id</option>

}

</select>

&nbsp;

<input name="action" type="submit" style="border:hidden; width:200px" value="Удалить накладную" />

</div>

</div>

<div>

@Html.LabelFor(m => m.EmployeeFIO)

<div class="col-md-10">

<select name="EmployeeFIO">

@foreach (var fio in Model.EmployeesFIOs)

{

<option>@fio</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.FurnitureName)

<div class="col-md-10">

<select name="FurnitureName">

@foreach (var name in Model.FurnitureNames)

{

<option>@name</option>

}

</select>

</div>

</div>

<div>

@Html.LabelFor(m => m.ProviderId)

<div class="col-md-10">

@Html.TextBoxFor(m => m.ProviderId)

</div>

</div>

<div>

@Html.LabelFor(m => m.ProviderName)

<div class="col-md-10">

@Html.TextBoxFor(m => m.ProviderName)

</div>

</div>

<div>

@Html.LabelFor(m => m.Material)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Material)

</div>

</div>

<div>

@Html.LabelFor(m => m.Weight)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Weight)

</div>

</div>

<div>

@Html.LabelFor(m => m.Price)

<div class="col-md-10">

@Html.TextBoxFor(m => m.Price)

</div>

</div>

<div>

@Html.LabelFor(m => m.DateOfSupply)

<div class="col-md-10">

<input type="date" id="datepicker" name="DateOfSupply" min="2000-01-01" max="2022-01-01" />

</div>

</div>

<br />

<input type="submit" class="btn btn-default" style="left:15px; position:relative" value="Изменить" />

}

*Views/\_ViewImports.cshtml*

@using CourseProject

@using CourseProject.Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

*Views/\_ViewStart.cshtml*

@{

Layout = "\_Layout";

}

*appsettings.json*

{

"ConnectionStrings": {

"SqlServerConnection": "Data Source = DESKTOP-G8EVE4H; Database=FurnitureFabric; Integrated Security = True; Connect Timeout = 60; Encrypt = False; TrustServerCertificate = False; ApplicationIntent = ReadWrite; MultiSubnetFailover = False"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

*Program.cs*

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Hosting;

namespace CourseProject

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

*MyStyle.css*

@import url('https://fonts.googleapis.com/css2?family=Comfortaa:wght@400;500&display=swap');

body {

background-image: url(../background.png);

font-family: 'Comfortaa', cursive;

color: white;

}

.main {

width: 100%;

padding-right: 15px;

padding-left: 15px;

margin-right: auto;

margin-left: auto;

opacity: 0;

position: relative;

display: -ms-flexbox;

display: flex;

-ms-flex-wrap: wrap;

flex-wrap: wrap;

-ms-flex-align: center;

align-items: center;

-ms-flex-pack: justify;

justify-content: space-between;

padding: 0.5rem 1rem;

color: white;

}

.main:hover {

opacity: 1;

transition: 1s;

background-color: rgba(0,100,0,0.9);

border: 2px solid white;

color: white;

}

main a:hover {

color: white;

}

.tables {

background-color: rgba(0,100,0,0.9);

width: max-content;

border-radius: 10px;

padding-right: 15px;

padding-left: 15px;

margin-right: auto;

margin-left: auto;

left:300px;

border: 2px solid white;

}

.tables table {

background-color: rgba(0,100,0,0.9);

border-color:white;

}

.tables input,.tables select {

background-color: white;

border-radius: 10px;

width: 500px;

}

.tables a {

color: white;

}

.tables button{

background-color: rgba(0,100,0,0.9);

border: 2px solid white;

color:white;

}

ПРИЛОЖЕНИЕ Б

(обязательное)

**Схема классов**