**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

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

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Разработка приложений баз данных для информационных систем»

на тему: **«*WEB*-ПРИЛОЖЕНИЕ баз данных «ГРУЗОПЕРЕВОЗКИ»**

Исполнитель: студент гр. ЗИТ-31

Войтишник Е.В.

Руководитель: доцент

Асенчик О.Д.

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

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

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

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

Подписи членов комиссии

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

Гомель 2021

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc66923564)

[1 Логическая и физическая структура базы данных 8](#_Toc66923565)

[1.1 Информационно-логическая модель информационной системы 8](#_Toc66923566)

[1.2 Физическая модель базы данных 11](#_Toc66923567)

[1.3 Файловая структура базы данных 13](#_Toc66923568)

[2 Аппаратное и программное обеспечение информа- ционной системы 15](#_Toc66923569)

[2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных 15](#_Toc66923570)

[2.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера 15](#_Toc66923571)

[2.3 Требования к системному и прикладному программному обеспечению на стороне клиента 15](#_Toc66923572)

[2.4 Настройка и развёртывание приложения на сервере 16](#_Toc66923573)

[3.3 Описание контроллеров 18](#_Toc66923574)

[3.4 Описание представлений 20](#_Toc66923575)

[4 Руководство пользователя 21](#_Toc66923576)

[4.2 Назначение, условие применения и функционал 21](#_Toc66923577)

[4.3 Подготовка к работе 21](#_Toc66923578)

[4.4 Описание операции по обработки данных 22](#_Toc66923579)

[5 Руководство программиста 26](#_Toc66923580)

[5.1 Назначения и условия применения программы 26](#_Toc66923581)

[5.2 Характеристики программы 26](#_Toc66923582)

[5.3 Сопровождение программного комплекса 26](#_Toc66923583)

[5.4 Входные и выходные данные 27](#_Toc66923584)

[5.5 Сообщения в ходе работы приложения 27](#_Toc66923585)

[Заключение 28](#_Toc66923586)

[Список используемых источников 29](#_Toc66923587)

[Приложение А – Код программы 30](#_Toc66923588)

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

# ВВЕДЕНИЕ

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

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

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

Предприятии по грузоперевозкам имеют очень глубокие исторические корни и постоянно совершенствовалась за всю историю существования человеческой цивилизации. Однако основные свои черты предприятии сохранили со времен античности до наших дней. История предприятий по грузоперевозкам – это постоянный процесс совершенствования технологий.

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

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

Для предприятий, занимающихся грузоперевозкой, необходимо создать приложение, позволяющее создавать и отслеживать состояние заключаемых контрактов, переченью услуг, манипулированию информацией о клиентах и сотрудниках компании.

Для реализации поставленной задачи необходимо определить список технологий и программных средств, позволяющих автоматизировать весь процесс. Программа будет состоять из источника данных, представляющего из себя базу данных, и *web*-приложения, работающего с конкретной базой данных.

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

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

Для решения поставленной задачи в качестве СУБД используется *MS SQL Server*. Данная СУБД обеспечивает поддержку баз данных очень большого объёма и обработку сложных запросов, а также имеет эффективные алгоритмы для работы с памятью и автоматизированным контролем размера файлов баз данных. В качестве технологии для разработки веб-приложения используется платформа *ASP.NET Core MVC* [3, с. 105]. Данная платформа является многофункциональной платформой для создания веб-приложений с помощью шаблона проектирования *Model-View-Controller* (модель-контроллер-представление). Структура *MVC* предполагает разделение приложения на три основных компонента: модель, представление и контроллер [6]. Каждый компонент решает свои задачи и взаимодействует с другими компонентами. Т.е. данный шаблон проектирования позволяет разделить задачи для каждого компонента, позволяет разрабатывать проект в команде, разделяя задачи между участниками и обеспечивает дальнейшую масштабируемость проекта. Благодаря такой схеме связей и распределения обязанностей между компонентами процесс масштабирования приложения становится проще, т.к. облегчается процесс написания кода, выполнения отладки и тестирования компонентов. Для доступа к данным используется технология *Entity Framework Core*. Данная технология является *ORM (object-relational mapping –* отображение данных на реальные объекты) инструментом, т.е. она позволяет работать с реляционными данными, используя классы и их иерархии. Также основным преимуществом данной технологии является использование универсального интерфейса для работы с данными, что позволяет легко и быстро сменить СУБД.

# 1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

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

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

Составление логической модели включает в себя:

– разработку требований к информационной системе,

­­­­– предварительное проектирование системы.

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

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

– «Сотрудники»;

– «Марки автомобилей»;

– «Виды грузов»;

– «Автомобили»;

– «Грузы»;

– «Рейсы».

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

Таблица 1.1 – Отношение описывающие сущность «Сотрудники»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор | Уникальный инкрементируемый идентификатор для каждого сотрудника. Является первичным ключом. | Целое число |
| ФИО | Содержит полную ФИО сотрудника предприятия. | Строка |
| Возраст | Год рождения сотрудника предприятия. | Целое число |
| Адрес | Содержит данные о месте проживания сотрудника. | Строка |

Продолжение таблицы 1.1.

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

Отношение для сущности «Грузы», описано в таблице 1.2. Отношение по условию задачи должно содержать атрибуты: «Наименование», «Вид груза», «Срок годности», «Особенности». Данное отношение следует привести к третей нормальной форме, заменив атрибут «Вид груза» на атрибут «Код вида груза» связав отношение «Грузы» с отношением «Виды грузов». Так как по условию задачи сущность «Грузы», описание сущности и атрибутов приведено в таблице 1.2, должна иметь список должностей. Тем самым будет организованна связь один ко многим, между данными отношениями.

Таблица 1.2 – Отношение описывающие сущность «Грузы»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор | Уникальный инкрементируемый идентификатор для каждого груза. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование груза. | Строка |
| Код вида груза | Содержит ссылку на идентификатор вида груза. Является внешним ключом для связи с отношением «Виды груза». | Целое число |
| Срок годности | Содержит дату окончания срока годности. | Дата |
| Особенности | Содержит описание особенностей перевозимого груза. | Строка |

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

Таблица 1.3 – Отношение описывающие сущность «Виды груза»

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

Для сущности «Рейсы» было создано отношение (таблица) с атрибутами: «Идентификатор рейса», «Заказчик», «Пункт отправления», «Пункт назначения», «Дата отправления», «Дата прибытия», «Автомобиль», «Цена», «Отметка об оплате», «Отметка о возвращении», «Груз». Данная сущность находится в ненормализованном виде и имеет связь один ко многим с отношениями «Автомобиль» и «Грузы». После приведения данной сущности к нормализованному виду появится два атрибута, такие-как «Код автомобиля» и «Код груза» соответственно. Подробное описание отношения и атрибутов приведено в таблице 1.4. Данное отношение находится в первой нормальной форме.

Таблица 1.4 – Отношение описывающие сущность «Рейсы»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор рейса | Уникальный инкрементируемый идентификатор для каждого рейса. Является первичным ключом. | Целое число |
| Код автомобиля | Содержит ссылку на идентификатор автомобиля. Является внешним ключом для связи с отношением «Автомобили». | Целое число |
| Код груза | Содержит ссылку на идентификатор груза. Является внешним ключом для связи с отношением «Груза». | Целое число |
| Заказчик | Содержит полное ФИО заказчика. | Строка |
| Пункт отправления | Содержит данные о пункте отправления. | Строка |

Продолжение таблицы 1.4.

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Пункт прибытия | Содержит данные о пункте прибытия. | Строка |
| Цена | Содержит окончательную стоимость рейса. | Число с плавающей точкой |
| Отметка об оплате | Содержит информацию об совершённой оплате заказчика. | Логическое значение |
| Отметка о возвращении | Содержит информацию о возвращении автомобиля с рейса. | Логическое значение |

Сущность «Марки автомобилей» было реализовано отношением, таблица 1.5. В данном отношении будут определены следующие атрибуты: «Идентификатор марки автомобилей», «Наименование», «Технические характеристики», «Описание», «Тип».

Таблица 1.5 – Отношение описывающие сущность «Марки автомобилей»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Идентификатор марки автомобилей | Уникальный инкрементируемый идентификатор для каждой марки автомобиля. Является первичным ключом. | Целое число |
| Наименование | Содержит наименование марки автомобилей. | Строка |
| Технические характеристики | Содержит информацию о максимальной скорости автомобиля. | Число |
| Описание | Содержит полное описание основных характеристик автомобиля для данной марки. | Строка |
| Тип | Содержит информацию о типе автомобиля. | Строка |

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

## **1.2 Физическая модель базы данных**

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

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

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

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

Для процесса преобразовании логической модели в физическую существует несколько правил:

– сущности становятся таблицами в физической базе данных;

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

– уникальные идентификаторы становятся столбцами, не допускающими значение *NULL*, т.е. первичными ключами. Также значение идентификатора делается автоинкрементным для обеспечения уникальности;

– все отношения моделируются в виде внешних ключей.

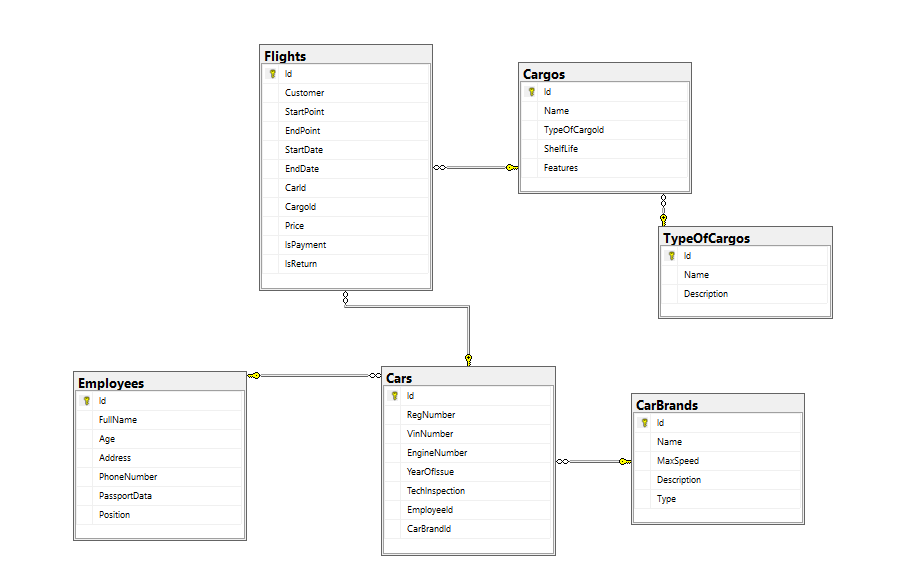


Рисунок 1.1 – Диаграмма базы данных

## **1.3 Файловая структура базы данных**

Каждая база данных *SQL Server* имеет как минимум два рабочих системных файла: файл данных и файл журнала. Файлы данных содержат данные и объекты, такие как таблицы, индексы, хранимые процедуры и представления. Файлы журнала содержат сведения, необходимые для восстановления всех транзакций в базе данных. Файлы данных могут быть объединены в файловые группы для удобства распределения и администрирования.

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

Файлы *SQL Server* имеют два типа имен файлов.

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

*os\_file\_name*: имя физического файла, включающее путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы.

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

Для каждого отношения были получены следующие таблицы: *Flights*, *Cargos*, *TypeOfCargos*, *Employees*, *Cars* и *CarBrands*.

Таблицы *Employees*, *CarBrands* и *TypeOfCargos* находятся в отношении «один» и описывают сущности «Сотрудники», «Автомобили» и «Марки автомобилей» и подобраны физические тип данных для соответствующих столбцов, установлены первичные ключи.

Таблицы *Cars, Flights* и *Cargos* находятся в отношении «многие», описывают сущности «Автомобили», «Поездки» и «Грузы». Имеют автоинкрементируемый первичный ключ и внешние ключи для связи с таблицами в отношении «Один».

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

# 2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМА- ЦИОННОЙ СИСТЕМЫ

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

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

­­– установленный *MS SQL Server*;

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

– сетевое программное обеспечение;

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

– минимальный объем оперативной памяти 1 ГБ;

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

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

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

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

­­ – процессор *x*86/*x*64 с тактовой частотой 1 ГГц;

– минимальный объем оперативной памяти 512 МБ;

– требуется как минимум 4,5 ГБ свободного места на диске;

– операционные системы *Windows* 7, 8, 10, *Linux*, *Mac OS*.

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

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

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

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

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

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

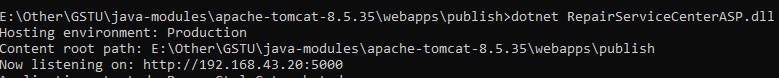


Рисунок 2.1 – Запуск веб-приложения на веб-сервере

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

**3 СТРУКТУРА ПРИЛОЖЕНИЯ**

**3.1 Описание общей структуры веб-приложения**

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

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

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

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

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

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

Классы *Car, Cargo, Employee, Flight, TypeOfCargo* и *CarBrand* описывают таблицы *Clients, Cargos, Employees, Flights, TypeOfCargos* и *CarBrands* соответственно. Код данных классов представлен в приложении Б.

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

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

## **3.3 Описание контроллеров**

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

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

Для работы с созданными моделями разработаны следующие контроллеры:

– *HomeController* – отвечает за вывод начальной страницы;

– *CarsController –* отвечает за работу с таблицей *Cars;*

– *CargosController –* отвечает за работу с таблицей *Cargos;*

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

*– FlightsController –* отвечает за работу с таблицей *Flights;*

*– CarBrandsController –* отвечает за работу с таблицей *CarBrands*

*– TypeOfCargosController –* отвечает за работу с таблицей *TypeOfCargos.*

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

– *Index;*

*– Details[GET];*

*– Create[GET];*

*– Create[POST];*

*– Edit[GET];*

*– Edit[POST];*

*– Delete[GET];*

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

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

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

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

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

Метод *Edit[POST]* вызывается при отправке результата формы редактирования записи. Данный метод в качестве входных параметров принимает идентификатор записи и объект, содержащий данные об этой записи. Если входной идентификатор и идентификатор объекта не совпадают, то метод возвращает стандартное сообщение об ошибке. Иначе метод выполняет валидацию входных данных и если данные верны, то производится обновление данных в базе. Если операция обновления прошла успешно, то происходит переход в метод *Index* текущего контроллера. В случае возникновения ошибки метод возвращает стандартное сообщение об ошибке.

## **3.4 Описание представлений**

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

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

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

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

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

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

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

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

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**4.1 Введение**

Данное *web*-приложение производит автоматизацию процесса предоставления услуг и учёта данных о внутренней экосистемы предприятий, представляющих услуг грузоперевозок.

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

*Web*-приложение предназначено для управления и учёта данных о рейсах и управление данными предприятий занимающихся грузоперевозками.

Основные функции приложения:

– просмотр списков заказов, выполненных на прошлой неделе;

– информация о всех перевозимых грузах;

– отображение информации о сотрудниках, работающих на определённой должности;

– отображение информации о заданном виде груза;

– отображение количества автомобилей, прошедших технический просмотр за определённый промежуток времени;

– добавление, просмотр и редактирования данных о марках автомобилей;

– добавление, просмотр и редактирования данных о видах грузов;

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

– добавление, просмотр и редактирования данных о проводимых рейсах;

– добавление, просмотр и редактирования данных о автомобилях.

## **4.3 Подготовка к работе**

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

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

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

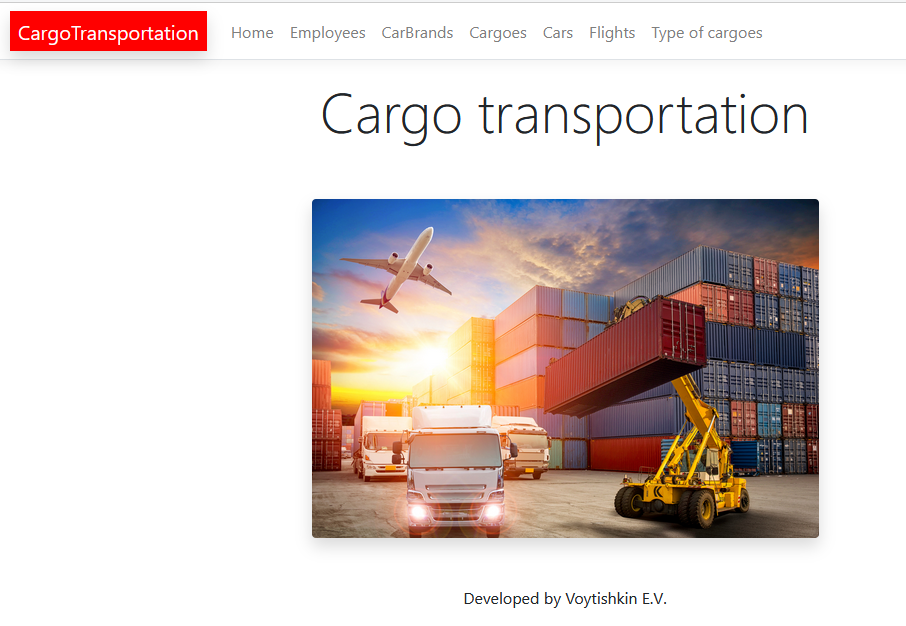


Рисунок 4.1 – Выбор сервиса

Затем загрузится новое окно со списком сотрудников, рисунок 4.2.

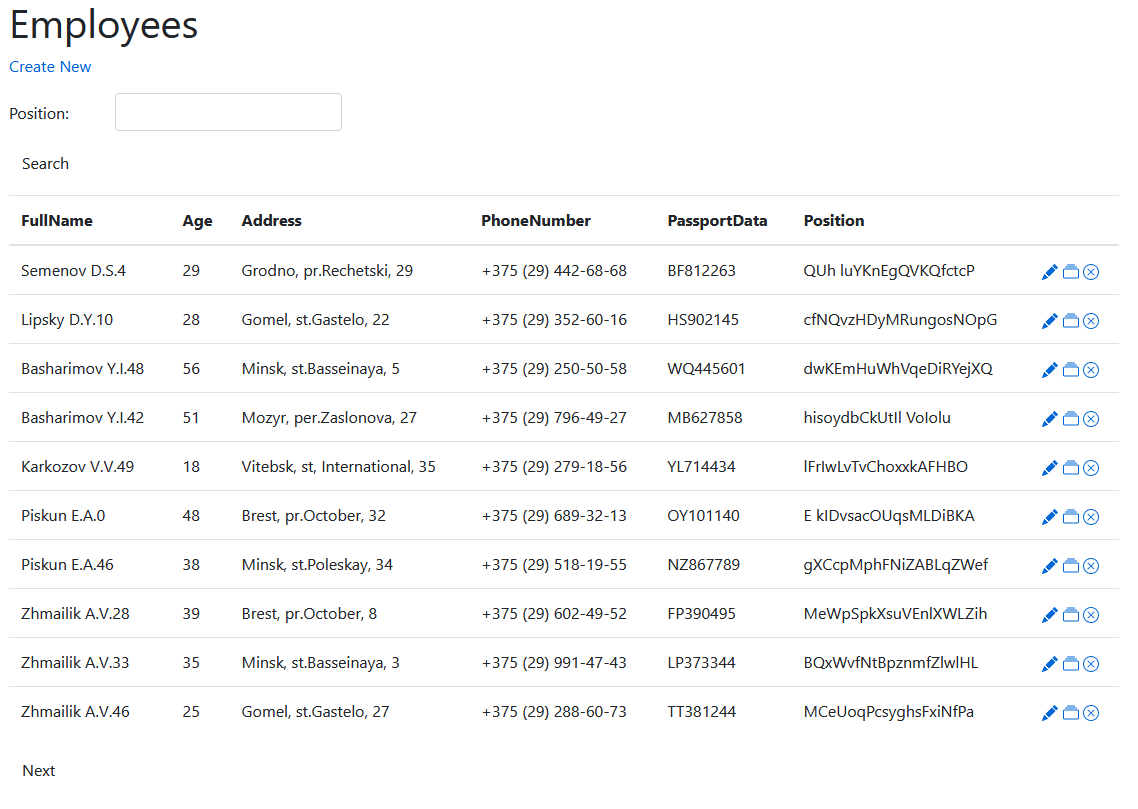


Рисунок 4.2 – Список сотрудников

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

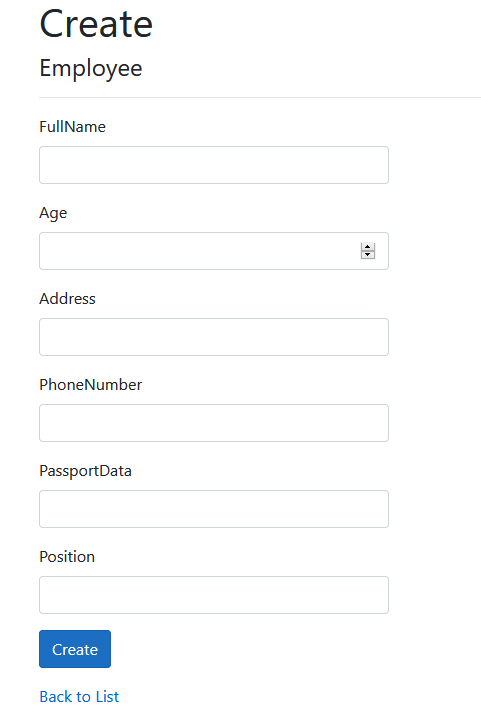


Рисунок 4.3 – Форма для добавления нового клиента

После добавления сотрудника пользователь будет перенаправлен на страницу с выборкой.

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

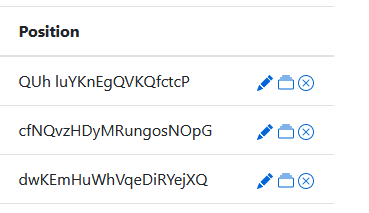


Рисунок 4.4 – Пункты для редактирования, удаления и просмотра данных о клиенте

Для редактирования требуется выбрать пункт «Редактировать», затем пользователь будет перенаправлен, на страницу с формами для редактирования данных, рисунок 4.5. Чтобы сохранить изменения, требуется нажать на кнопку сохранить.

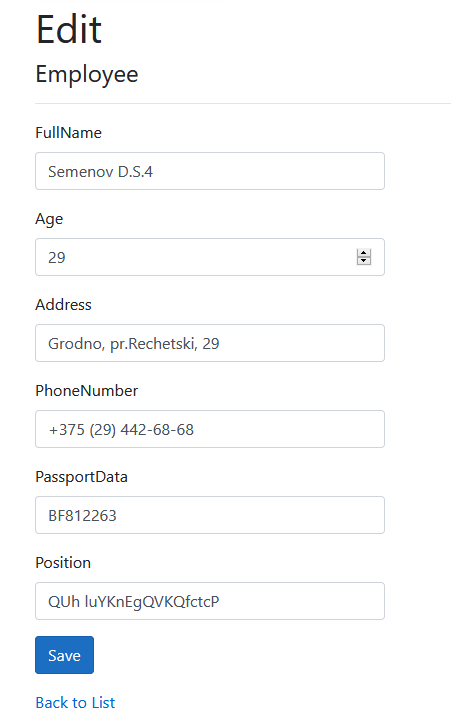


Рисунок 4.5 – Окно для редактирования данных

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

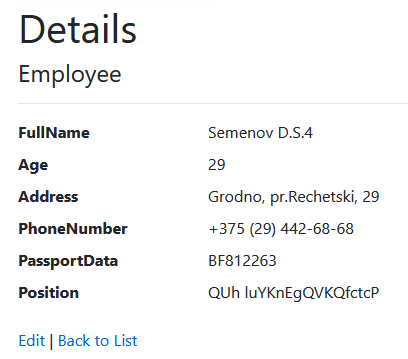


Рисунок 4.6 – Подробности о сотруднике

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

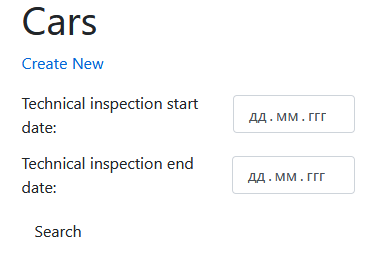


Рисунок 4.7 – Формы для фильтрации данных

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

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

# 5 РУКОВОДСТВО ПРОГРАММИСТА

## **5.1 Назначения и условия применения программы**

Приложение предназначена для предоставления информации из базы данных предприятия.

Основные функции приложения:

– производить различные манипуляции с данными из базы данных;

– предоставления данных в удобном виде пользователям для их просмотра;

– управление данными о клиентах;

– редактирования, добавление и изменения данных из базы с помощью веб-интерфейса.

Для запуска приложения на сервере должна быть установлена платформа *.NET Core*. Для соединения с базой данных, требуется предварительная конфигурация параметров для соединения с ней.

## **5.2 Характеристики программы**

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

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

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

## **5.3 Сопровождение программного комплекса**

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

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

## **5.4 Входные и выходные данные**

Входными данными для веб-приложения является:

– веб-сервер, на котором разворачивается приложение;

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

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

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

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

При работе программа может оповещать пользователя о следующих неполадках:

– некорректно введённые данных при добавлении и редактировании записей;

– некорректный *URL*-адрес, страница не найдена;

– ошибка при добавлении записей, запись с введёнными значениями уже существуют в базе.

Данные сообщения передаются в специальном виде ошибки с описанием проблемы.

# ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта было реализовано веб-приложение, которое производит автоматизацию оформления услуг, предоставляемых предприятием. Приложение является простым и удобным благодаря адаптивному и понятному интерфейсу. Критериями удобства является, в первую очередь, наличие навигационного меню, что позволяет пользователю всю необходимую информацию, а также улучшает навигацию между страницами, не производя при этом никаких лишних действий.

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

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

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

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

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

# ПРИЛОЖЕНИЕ А

(обязательное)

**Код программы**

appsettings.json

{

"ConnectionStrings": {

"DefaultConnection": "Server=.\\SQLEXPRESS;Database=CargoTransportaion;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

Car.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class Car

{

public int Id { get; set; }

public string RegNumber { get; set; }

public string VinNumber { get; set; }

public string EngineNumber { get; set; }

public int YearOfIssue { get; set; }

public DateTime TechInspection { get; set; }

public int EmployeeId { get; set; } // Driver

public int CarBrandId { get; set; }

public Employee Employee { get; set; }

public CarBrand CarBrand { get; set; }

public ICollection<Flight> Flights { get; set; }

public Car()

{

Flights = new List<Flight>();

}

}

}

Cargo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class Car

{

public int Id { get; set; }

public string RegNumber { get; set; }

public string VinNumber { get; set; }

public string EngineNumber { get; set; }

public int YearOfIssue { get; set; }

public DateTime TechInspection { get; set; }

public int EmployeeId { get; set; } // Driver

public int CarBrandId { get; set; }

public Employee Employee { get; set; }

public CarBrand CarBrand { get; set; }

public ICollection<Flight> Flights { get; set; }

public Car()

{

Flights = new List<Flight>();

}

}

}

Employee.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; }

public int Age { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string PassportData { get; set; }

public string Position { get; set; }

public ICollection<Car> Cars { get; set; }

public Employee()

{

Cars = new List<Car>();

}

}

}

Flight.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class Flight

{

public int Id { get; set; }

public string Customer { get; set; }

public string StartPoint { get; set; }

public string EndPoint { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public int CarId { get; set; }

public int CargoId { get; set; }

public double Price { get; set; }

public bool IsPayment { get; set; }

public bool IsReturn { get; set; }

public Car Car { get; set; }

public Cargo Cargo { get; set; }

}

}

TypeOfCargo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class TypeOfCargo

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

}

}

CarBrand.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Models

{

public class CarBrand

{

public int Id { get; set; }

public string Name { get; set; }

public int MaxSpeed { get; set; }

public string Description { get; set; }

public string Type { get; set; }

public ICollection<Car> Cars { get; set; }

public CarBrand()

{

Cars = new List<Car>();

}

}

}

CargoTransportationContext.cs

using CargoTransportation.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Data

{

public class CargoTransportationContext : DbContext

{

public CargoTransportationContext(DbContextOptions options) : base(options)

{

}

public DbSet<Car> Cars { get; set; }

public DbSet<Cargo> Cargos { get; set; }

public DbSet<CarBrand> CarBrands { get; set; }

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

public DbSet<Flight> Flights { get; set; }

public DbSet<TypeOfCargo> TypeOfCargos { get; set; }

}

}

DbInitializer.cs

using CargoTransportation.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Data

{

public class DbInitializer

{

private static Random randObj = new Random(1);

public static void Initialize(CargoTransportationContext db)

{

db.Database.EnsureCreated();

int employeeCount = 50;

int carBrandCount = 40;

int carCount = 100;

int cargoCount = 100;

int flightCount = 200;

TypeOfCargo(db);

EmployeeGenerate(db, employeeCount);

CarBrandGenerate(db, carBrandCount);

CarGenerate(db, carCount);

CargoGenerate(db, cargoCount);

FlightGenerat(db, flightCount);

}

private static void TypeOfCargo(CargoTransportationContext db)

{

if (db.TypeOfCargos.Any())

{

return;

}

db.TypeOfCargos.AddRange(new TypeOfCargo[]

{

new TypeOfCargo()

{

Name = "Bulk",

Description = "Some description"

},

new TypeOfCargo()

{

Name = "Powdery",

Description = "Some description"

},

new TypeOfCargo()

{

Name = "Oversized",

Description = "Some description"

},

new TypeOfCargo()

{

Name = "Piece",

Description = "Some description"

}

});

db.SaveChanges();

}

private static void EmployeeGenerate(CargoTransportationContext db, int count)

{

if (db.Employees.Any())

{

return;

}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";

string[] fullNamesVoc = { "Zhmailik A.V.", "Setko A.I.", "Semenov D.S.", "Davidchik A.E.", "Piskun E.A.",

"Drakula V.A.", "Yastrebov A.A.", "Steponenko Y.A.", "Basharimov Y.I.", "Karkozov V.V.", "Lipsky D.Y." };

string[] addressVoc = {"Mozyr, per.Zaslonova, ", "Gomel, st.Gastelo, ", "Minsk, st.Poleskay, ",

"Grodno, pr.Rechetski, ", "Vitebsk, st, International, ",

"Brest, pr.October, ", "Minsk, st.Basseinaya, ", "Mozyr, boulevard Youth, " };

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

{

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

var age = randObj.Next(18, 65);

var address = addressVoc[randObj.Next(addressVoc.GetLength(0))] + randObj.Next(count);

var phoneNumber = "+375 (29) " + randObj.Next(100, 999) + "-" + randObj.Next(10, 99) +

"-" + randObj.Next(10, 99);

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

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

var position = new string(Enumerable.Repeat(chars, 20)

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

db.Employees.Add(new Employee()

{

FullName = fullName,

Age = age,

Address = address,

PhoneNumber = phoneNumber,

PassportData = passportData,

Position = position

});

}

db.SaveChanges();

}

private static void CarBrandGenerate(CargoTransportationContext db, int count)

{

if (db.CarBrands.Any())

{

return;

}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";

string[] nameVoc = { "Mercedes-Benz W", "VW Passat B", "VW Golf ", "VW Polo ", "ВАЗ201", "Tesla Model ",

"Volvo F", "Seat ", "Toyota Camry 3.", "Hyundai "};

string[] typeVoc = { "Passenger car", "Truck ", "Сargo-passenger car" };

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

{

var name = nameVoc[randObj.Next(nameVoc.GetLength(0))] + randObj.Next(count);

int maxSpeed = randObj.Next(160, 320);

var description = new string(Enumerable.Repeat(chars, 20)

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

var type = typeVoc[randObj.Next(typeVoc.GetLength(0))] + randObj.Next(count);

db.CarBrands.Add(new CarBrand()

{

Name = name,

MaxSpeed = maxSpeed,

Description = description,

Type = type

});

}

db.SaveChanges();

}

private static void CarGenerate(CargoTransportationContext db, int count)

{

if (db.Cars.Any())

{

return;

}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int carBrandCount = db.CarBrands.Count();

int employeeCount = db.Employees.Count();

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

{

var regNumber = chars[randObj.Next(chars.Length)].ToString() + chars[randObj.Next(chars.Length)].ToString()

+ " " + randObj.Next(1000, 9999);

var vinNumber = GetRandStr(chars, 8) + randObj.Next(0, 9) + GetRandStr(chars, 2) + randObj.Next(100000, 999999);

var engineNumber = GetRandStr(chars, 2) + randObj.Next(1000, 9999) + GetRandStr(chars, 1) + randObj.Next(10, 99);

var yearOfIssue = randObj.Next(1980, 2020);

var techInspection = DateTime.Now.AddDays(-randObj.Next(1000));

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

var carBrandId = randObj.Next(1, carBrandCount + 1);

db.Cars.Add(new Car()

{

RegNumber = regNumber,

VinNumber = vinNumber,

EngineNumber = engineNumber,

YearOfIssue = yearOfIssue,

TechInspection = techInspection,

EmployeeId = employeeId,

CarBrandId = carBrandId

});

}

db.SaveChanges();

}

private static void CargoGenerate(CargoTransportationContext db, int count)

{

if (db.Cargos.Any())

{

return;

}

string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";

var typeOfCargoCount = db.TypeOfCargos.Count();

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

{

var name = GetRandStr(chars, 20);

var typeOfCargoId = randObj.Next(1, typeOfCargoCount + 1);

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

var features = GetRandStr(chars, 40);

db.Cargos.Add(new Cargo()

{

Name = name,

TypeOfCargoId = typeOfCargoId,

ShelfLife = shelfLife,

Features = features

});

}

db.SaveChanges();

}

private static void FlightGenerat(CargoTransportationContext db, int count)

{

if (db.Flights.Any())

{

return;

}

string[] customerVoc =

{

"Lipsky D.Y.", "Stolny S.D.", "Semenov D.S.", "Deker M.A.",

"Ropot I.V.", "Butkovski Y.V.",

"Stepanenko Y.V.", "Moiseikov R.A.", "Rogolevich N.V.", "Gerosimenko M.A.",

"Galetskiy A.A.", "Zankevich K.A."

};

string[] pointVoc =

{

"Gomel", "Mozyr", "Minsk", "Grodno", "Brest", "Moscow", "Mogilev"

};

int carCount = db.Cargos.Count();

int cargoCount = db.Cargos.Count();

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

{

var customer = customerVoc[randObj.Next(customerVoc.GetLength(0))] + randObj.Next(count);

var startPoint = pointVoc[randObj.Next(pointVoc.GetLength(0))] + randObj.Next(count);

var endPoint = pointVoc[randObj.Next(pointVoc.GetLength(0))] + randObj.Next(count);

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

var endDate = startDate.AddDays(randObj.Next(500));

var carId = randObj.Next(1, carCount + 1);

var cargoId = randObj.Next(1, cargoCount + 1);

var price = randObj.NextDouble() \* 100;

var isPayment = randObj.Next(0, 1000) > 500 ? true : false;

var isReturn = randObj.Next(0, 1000) > 500 ? true : false;

db.Flights.Add(new Flight()

{

Customer = customer,

StartPoint = startPoint,

EndPoint = endPoint,

StartDate = startDate,

EndDate = endDate,

CarId = carId,

CargoId = cargoId,

Price = price,

IsPayment = isPayment,

IsReturn = isReturn

});

}

db.SaveChanges();

}

private static string GetRandStr(string chars, int maxChar)

{

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

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

}

}

}

ErrorViewModel.cs

using System;

namespace RepairServiceCenterASP.Models

{

public class ErrorViewModel

{

public string RequestId { get; set; }

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

}

}

CarBrandsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

using CargoTransportation.ViewModels;

namespace CargoTransportation.Controllers

{

public class CarBrandsController : Controller

{

private readonly CargoTransportationContext \_context;

public CarBrandsController(CargoTransportationContext context)

{

\_context = context;

}

// GET: CarBrands

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

{

var pageSize = 10;

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

IQueryable<CarBrand> carBrands = \_context.CarBrands

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

.Take(pageSize); ;

return View(new CarBrandViewModel()

{

CarBrands = await carBrands.ToListAsync(),

PageViewModel = new PageViewModel(itemCount, page, pageSize)

});

}

// GET: CarBrands/Details/5

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

{

if (id == null)

{

return NotFound();

}

var carBrand = await \_context.CarBrands

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

if (carBrand == null)

{

return NotFound();

}

return View(carBrand);

}

// GET: CarBrands/Create

public IActionResult Create()

{

return View();

}

// POST: CarBrands/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,MaxSpeed,Description,Type")] CarBrand carBrand)

{

if (ModelState.IsValid)

{

\_context.Add(carBrand);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(carBrand);

}

// GET: CarBrands/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var carBrand = await \_context.CarBrands.FindAsync(id);

if (carBrand == null)

{

return NotFound();

}

return View(carBrand);

}

// POST: CarBrands/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,MaxSpeed,Description,Type")] CarBrand carBrand)

{

if (id != carBrand.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(carBrand);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CarBrandExists(carBrand.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(carBrand);

}

// GET: CarBrands/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var carBrand = await \_context.CarBrands

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

if (carBrand == null)

{

return NotFound();

}

return View(carBrand);

}

// POST: CarBrands/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var carBrand = await \_context.CarBrands.FindAsync(id);

\_context.CarBrands.Remove(carBrand);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CarBrandExists(int id)

{

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

}

}

}

CargoesController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

using CargoTransportation.ViewModels;

using CargoTransportation.ViewModels.Filters;

namespace CargoTransportation.Controllers

{

public class CargoesController : Controller

{

private readonly CargoTransportationContext \_context;

public CargoesController(CargoTransportationContext context)

{

\_context = context;

}

// GET: Cargoes

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

{

var pageSize = 10;

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

IQueryable<Cargo> cargos = \_context.Cargos;

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

{

cargos = cargos.Where(c => c.TypeOfCargoId == selectedTypeOfCargoId);

}

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

.Take(pageSize)

.Include(c => c.TypeOfCargo);

return View(new CargoViewModel()

{

Cargos = await cargos.ToListAsync(),

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

CargoFilter = new CargoFilter(selectedTypeOfCargoId, await \_context.TypeOfCargos.ToListAsync())

});

}

// GET: Cargoes/Details/5

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

{

if (id == null)

{

return NotFound();

}

var cargo = await \_context.Cargos.Include(c => c.TypeOfCargo)

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

if (cargo == null)

{

return NotFound();

}

return View(cargo);

}

// GET: Cargoes/Create

public IActionResult Create()

{

ViewData["TypeOfCargoId"] = new SelectList(\_context.TypeOfCargos, "Id", "Name");

return View();

}

// POST: Cargoes/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,TypeOfCargoId,ShelfLife,Features")] Cargo cargo)

{

if (ModelState.IsValid)

{

\_context.Add(cargo);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["TypeOfCargoId"] = new SelectList(\_context.TypeOfCargos, "Id", "Name", cargo.TypeOfCargoId);

return View(cargo);

}

// GET: Cargoes/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var cargo = await \_context.Cargos.FindAsync(id);

if (cargo == null)

{

return NotFound();

}

ViewData["TypeOfCargoId"] = new SelectList(\_context.TypeOfCargos, "Id", "Name", cargo.TypeOfCargoId);

return View(cargo);

}

// POST: Cargoes/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,TypeOfCargoId,ShelfLife,Features")] Cargo cargo)

{

if (id != cargo.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(cargo);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CargoExists(cargo.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["TypeOfCargoId"] = new SelectList(\_context.TypeOfCargos, "Id", "Name", cargo.TypeOfCargoId);

return View(cargo);

}

// GET: Cargoes/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var cargo = await \_context.Cargos.Include(c => c.TypeOfCargo)

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

if (cargo == null)

{

return NotFound();

}

return View(cargo);

}

// POST: Cargoes/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var cargo = await \_context.Cargos.FindAsync(id);

\_context.Cargos.Remove(cargo);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CargoExists(int id)

{

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

}

}

}

EmployeesController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

using CargoTransportation.ViewModels;

using CargoTransportation.ViewModels.Filters;

namespace CargoTransportation.Controllers

{

public class EmployeesController : Controller

{

private readonly CargoTransportationContext \_context;

public EmployeesController(CargoTransportationContext context)

{

\_context = context;

}

// GET: Employees

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

{

var pageSize = 10;

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

IQueryable<Employee> employees = \_context.Employees;

if (!string.IsNullOrEmpty(position))

{

employees = employees.Where(e => e.Position.StartsWith(position));

}

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

.Take(pageSize);

return View(new EmployeeViewModel()

{

Employees = await employees.ToListAsync(),

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

EmployeeFilter = new EmployeeFilter(position)

});

}

// GET: Employees/Details/5

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

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees

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

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// GET: Employees/Create

public IActionResult Create()

{

return View();

}

// POST: Employees/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,FullName,Age,Address,PhoneNumber,PassportData,Position")] Employee employee)

{

if (ModelState.IsValid)

{

\_context.Add(employee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(employee);

}

// GET: Employees/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees.FindAsync(id);

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// POST: Employees/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,Age,Address,PhoneNumber,PassportData,Position")] Employee employee)

{

if (id != employee.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employee);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeExists(employee.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(employee);

}

// GET: Employees/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var employee = await \_context.Employees

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

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// POST: Employees/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var employee = await \_context.Employees.FindAsync(id);

\_context.Employees.Remove(employee);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EmployeeExists(int id)

{

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

}

}

}

CarsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

using CargoTransportation.ViewModels;

using CargoTransportation.ViewModels.Filters;

namespace CargoTransportation.Controllers

{

public class CarsController : Controller

{

private readonly CargoTransportationContext \_context;

public CarsController(CargoTransportationContext context)

{

\_context = context;

}

// GET: Cars

public async Task<IActionResult> Index(DateTime? techStartDate, DateTime? techEndDate, int page = 1)

{

var pageSize = 10;

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

IQueryable<Car> cars = \_context.Cars;

if (techStartDate.HasValue)

{

cars = cars.Where(c => c.TechInspection >= techStartDate.Value);

}

if (techEndDate.HasValue)

{

cars = cars.Where(c => c.TechInspection <= techEndDate.Value);

}

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

.Take(pageSize)

.Include(c => c.CarBrand)

.Include(c => c.Employee);

return View(new CarViewModel()

{

Cars = await cars.ToListAsync(),

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

CarFilter = new CarFilter(techStartDate, techEndDate)

});

}

// GET: Cars/Details/5

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars

.Include(c => c.CarBrand)

.Include(c => c.Employee)

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

if (car == null)

{

return NotFound();

}

return View(car);

}

// GET: Cars/Create

public IActionResult Create()

{

ViewData["CarBrandId"] = new SelectList(\_context.CarBrands, "Id", "Name");

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

return View();

}

// POST: Cars/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,RegNumber,VinNumber,EngineNumber,YearOfIssue,TechInspection,EmployeeId,CarBrandId")] Car car)

{

if (ModelState.IsValid)

{

\_context.Add(car);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["CarBrandId"] = new SelectList(\_context.CarBrands, "Id", "Name", car.CarBrandId);

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

return View(car);

}

// GET: Cars/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

ViewData["CarBrandId"] = new SelectList(\_context.CarBrands, "Id", "Name", car.CarBrandId);

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

return View(car);

}

// POST: Cars/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,RegNumber,VinNumber,EngineNumber,YearOfIssue,TechInspection,EmployeeId,CarBrandId")] Car car)

{

if (id != car.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(car);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CarExists(car.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["CarBrandId"] = new SelectList(\_context.CarBrands, "Id", "Name", car.CarBrandId);

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

return View(car);

}

// GET: Cars/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var car = await \_context.Cars

.Include(c => c.CarBrand)

.Include(c => c.Employee)

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

if (car == null)

{

return NotFound();

}

return View(car);

}

// POST: Cars/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var car = await \_context.Cars.FindAsync(id);

\_context.Cars.Remove(car);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CarExists(int id)

{

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

}

}

}

HomeController.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using BarbershopService.Models;

namespace BarbershopService.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

public IActionResult Privacy()

{

return View();

}

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

public IActionResult Error()

{

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

}

}

}

FlightsController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

using CargoTransportation.ViewModels;

using CargoTransportation.ViewModels.Filters;

namespace CargoTransportation.Controllers

{

public class FlightsController : Controller

{

private readonly CargoTransportationContext \_context;

public FlightsController(CargoTransportationContext context)

{

\_context = context;

}

// GET: Flights

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

{

var pageSize = 10;

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

IQueryable<Flight> flights = \_context.Flights;

if (flightsFromLastWeek)

{

var lastWeekDate = DateTime.Now.AddDays(-7);

flights = flights.Where(f => f.EndDate <= lastWeekDate);

}

flights = flights.Include(f => f.Car)

.Include(f => f.Cargo)

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

.Take(pageSize);

return View(new FlightViewModel()

{

Flights = await flights.ToListAsync(),

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

FlightFilter = new FlightFilter(flightsFromLastWeek)

});

}

// GET: Flights/Details/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights

.Include(f => f.Car)

.Include(f => f.Cargo)

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

if (flight == null)

{

return NotFound();

}

return View(flight);

}

// GET: Flights/Create

public IActionResult Create()

{

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "RegNumber");

ViewData["CargoId"] = new SelectList(\_context.Cargos, "Id", "Name");

return View();

}

// POST: Flights/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Customer,StartPoint,EndPoint,StartDate,EndDate,CarId,CargoId,Price,IsPayment,IsReturn")] Flight flight)

{

if (ModelState.IsValid)

{

\_context.Add(flight);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "RegNumber", flight.CarId);

ViewData["CargoId"] = new SelectList(\_context.Cargos, "Id", "Name", flight.CargoId);

return View(flight);

}

// GET: Flights/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights.FindAsync(id);

if (flight == null)

{

return NotFound();

}

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "RegNumber", flight.CarId);

ViewData["CargoId"] = new SelectList(\_context.Cargos, "Id", "Name", flight.CargoId);

return View(flight);

}

// POST: Flights/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Customer,StartPoint,EndPoint,StartDate,EndDate,CarId,CargoId,Price,IsPayment,IsReturn")] Flight flight)

{

if (id != flight.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(flight);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!FlightExists(flight.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["CarId"] = new SelectList(\_context.Cars, "Id", "RegNumber", flight.CarId);

ViewData["CargoId"] = new SelectList(\_context.Cargos, "Id", "Name", flight.CargoId);

return View(flight);

}

// GET: Flights/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights

.Include(f => f.Car)

.Include(f => f.Cargo)

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

if (flight == null)

{

return NotFound();

}

return View(flight);

}

// POST: Flights/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var flight = await \_context.Flights.FindAsync(id);

\_context.Flights.Remove(flight);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool FlightExists(int id)

{

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

}

}

}

TypeOfCargoesController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using CargoTransportation.Data;

using CargoTransportation.Models;

namespace CargoTransportation.Controllers

{

public class TypeOfCargoesController : Controller

{

private readonly CargoTransportationContext \_context;

public TypeOfCargoesController(CargoTransportationContext context)

{

\_context = context;

}

// GET: TypeOfCargoes

public async Task<IActionResult> Index()

{

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

}

// GET: TypeOfCargoes/Details/5

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

{

if (id == null)

{

return NotFound();

}

var typeOfCargo = await \_context.TypeOfCargos

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

if (typeOfCargo == null)

{

return NotFound();

}

return View(typeOfCargo);

}

// GET: TypeOfCargoes/Create

public IActionResult Create()

{

return View();

}

// POST: TypeOfCargoes/Create

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,Description")] TypeOfCargo typeOfCargo)

{

if (ModelState.IsValid)

{

\_context.Add(typeOfCargo);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(typeOfCargo);

}

// GET: TypeOfCargoes/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var typeOfCargo = await \_context.TypeOfCargos.FindAsync(id);

if (typeOfCargo == null)

{

return NotFound();

}

return View(typeOfCargo);

}

// POST: TypeOfCargoes/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description")] TypeOfCargo typeOfCargo)

{

if (id != typeOfCargo.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(typeOfCargo);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TypeOfCargoExists(typeOfCargo.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(typeOfCargo);

}

// GET: TypeOfCargoes/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var typeOfCargo = await \_context.TypeOfCargos

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

if (typeOfCargo == null)

{

return NotFound();

}

return View(typeOfCargo);

}

// POST: TypeOfCargoes/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var typeOfCargo = await \_context.TypeOfCargos.FindAsync(id);

\_context.TypeOfCargos.Remove(typeOfCargo);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool TypeOfCargoExists(int id)

{

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

}

}

}

DbInitializerExtensions.cs

using Microsoft.AspNetCore.Builder;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Middleware

{

public static class DbInitializerExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

DbInitializerMiddleware.cs

using CargoTransportation.Data;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next)

{

// инициализация базы данных

\_next = next;

}

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

{

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

{

DbInitializer.Initialize(dbContext);

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

}

// Call the next delegate/middleware in the pipeline

return \_next.Invoke(context);

}

}

}

CarBrandViewModel.cs

using CargoTransportation.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels

{

public class CarBrandViewModel

{

public ICollection<CarBrand> CarBrands { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

CargoViewModel.cs

using CargoTransportation.Models;

using CargoTransportation.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels

{

public class CargoViewModel

{

public ICollection<Cargo> Cargos { get; set; }

public PageViewModel PageViewModel { get; set; }

public CargoFilter CargoFilter { get; set; }

}

}

PageViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace NotarialOfficeRebuild.ViewModels

{

public class PageViewModel

{

public int PageNumber { get; private set; }

public int TotalPages { get; private set; }

public PageViewModel(int count, int pageNumber, int pageSize)

{

PageNumber = pageNumber;

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

}

public bool HasPreviousPage

{

get

{

return PageNumber > 1;

}

}

public bool HasNextPage

{

get

{

return PageNumber < TotalPages;

}

}

}

}

EmployeeViewModel.cs

using CargoTransportation.Models;

using CargoTransportation.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels

{

public class EmployeeViewModel

{

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

public PageViewModel PageViewModel { get; set; }

public EmployeeFilter EmployeeFilter { get; set; }

}

}

CarViewModel.cs

using CargoTransportation.Models;

using CargoTransportation.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels

{

public class CarViewModel

{

public ICollection<Car> Cars { get; set; }

public PageViewModel PageViewModel { get; set; }

public CarFilter CarFilter { get; set; }

}

}

FlightViewModel.cs

using CargoTransportation.Models;

using CargoTransportation.ViewModels.Filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels

{

public class FlightViewModel

{

public ICollection<Flight> Flights { get; set; }

public PageViewModel PageViewModel { get; set; }

public FlightFilter FlightFilter { get; set; }

}

}

CarFilter.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels.Filters

{

public class CarFilter

{

public DateTime? TechStartDate { get; set; }

public DateTime? TechEndDate { get; set; }

public CarFilter(DateTime? techStartDate, DateTime? techEndDate)

{

TechStartDate = techStartDate;

TechEndDate = techEndDate;

}

}

}

CargoFilter.cs

using CargoTransportation.Models;

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels.Filters

{

public class CargoFilter

{

public int? SelectedTypeOfCargoId { get; set; }

public SelectList TypeOfCargoes { get; set; }

public CargoFilter(int? selectedTypeOfCargoId, IList<TypeOfCargo> typeOfCargo)

{

typeOfCargo.Insert(0, new TypeOfCargo()

{

Id = 0,

Name = "All"

});

SelectedTypeOfCargoId = selectedTypeOfCargoId;

TypeOfCargoes = new SelectList(typeOfCargo, "Id", "Name", selectedTypeOfCargoId);

}

}

}

EmployeeFilter.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels.Filters

{

public class EmployeeFilter

{

public string Position { get; set; }

public EmployeeFilter(string position)

{

Position = position;

}

}

}

FlightFilter.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace CargoTransportation.ViewModels.Filters

{

public class FlightFilter

{

public bool FlightsFromLastWeek { get; set; }

public FlightFilter(bool flightsFromLastWeek)

{

FlightsFromLastWeek = flightsFromLastWeek;

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Чертёж структуры web-приложения**