Содержание

[Введение 3](#_Toc91029943)

[1 Описание предметной области 5](#_Toc91029944)

[1.1 Общие положения 5](#_Toc91029945)

[1.2 Сведения из теории 7](#_Toc91029946)

[1.3 Постановка задачи 10](#_Toc91029947)

[2 Технологии разработки приложения 11](#_Toc91029948)

[2.1 Алгоритм решения 11](#_Toc91029949)

[2.2 Макет приложения 12](#_Toc91029950)

[2.3 Описание программы 18](#_Toc91029951)

[2.4 Результаты работы программы 30](#_Toc91029952)

[3. Руководство пользователя 31](#_Toc91029953)

[Заключение 32](#_Toc91029954)

[Список использованных источников 33](#_Toc91029955)

# Введение

Автоматизация позволяет частично или полностью освободить человека от исполнения циклических процессов, или процессов, выполняющихся по строго заданному алгоритму. Развитие автоматика получила благодаря промышленно-техническому прогрессу. Даже автоматизация в быту берёт своё начало на промышленных производствах, где стремление к ускорению процесса и, соответственно, увеличение выручки способствовало внедрению новейших на тот момент наукоёмких средств автоматизации. Так же автоматизированные производства позволяли высвобождать большое количество рабочих рук, оптимизируя время и занятость персонала.

Автоматизация в наше время охватывает практически все сферы человеческой деятельности, от поддержания определённых технологических параметров до контроля работы на предприятии с помощью баз данных.

Внедрение автоматизированной системы управления позволяет:

- упростить работу рабочего персонала;

- предотвратить появление человеческой ошибки;

- повысить эффективность производства;

- увеличить прибыль предприятия;

- уменьшить время и точность обработки данных.

Системы управления металлургического завода призваны сократить издержки производства - временные и финансовые, облегчить управление бизнес процессами, включая взаимодействие с клиентами, поставщиками, автоматизировать ведение банковских счетов, заказов, платёжных требований и товаров. Компьютерная программа для металлургического завода должна быть очень гибкой и позволять вести учет в различных разрезах предприятия, как, например: учёт всех товаров на складе, учёт всех клиентов, учёт всех накладных и т. д.

Целью выполнения курсового проекта является развитие навыков реализации сетевых приложений с использованием прикладного интерфейса программирования сокетов; средой разработки прикладных сетевых приложений; получение опыта разработки сетевых программных средств. В ходе выполнения курсового проекта должно формироваться представление об этапах проектирования и создания сетевых программных средств. В ходе достижения цели решаются следующие задачи:

- развитие логического и алгоритмического мышления;

- выработка умения формулировать суждения и выводы, логически последовательно и доказательно их излагать;

- овладение навыками самостоятельной исследовательской работы;

- закрепление использования технологией сетевого программирования на стороне клиента и сервера.

1 Описание предметной области

1.1 Общие положения

Актуальность работы. Использование новых информационных технологий в разных сферах деятельности человека привело к необходимости упрощения и ускорения работы с товарами, их покупки, продажи и доставки. Такое программное обеспечение востребовано на рынке информационных технологий, поэтому постоянно изменяется, включая в себя новые возможности.

В настоящее время распространены веб-приложения по доставки различного рода товаров. Существуют многочисленные приложения, которые пользуются спросом на доставку.

Предметной областью является перевозки топлива.

Целью курсовой работы является создание веб-приложения и базы данных с помощью MS SQL Server и C#. MS SQL Server ˗ инструмент для визуального проектирования баз данных, также для администрирования и разработки. С# – платформа обладает следующими преимуществами:

- переносимость, или кросс-платформенность;

- объектная ориентированность, создана эффективная объектная модель;

- привычный синтаксис С/С++;

- встроенная и прозрачная модель безопасности;

- ориентация на интернет-задачи, сетевые распределенные приложения;

- динамичность, легкость развития и добавления новых возможностей;

- легкость в освоении;

Для автоматизации работы на металлургическом заводе, требуется выводить следующие

запросы:

- Добавление, детали, удаление и редактирование информации.

База данных должна выводить информацию:

- о заказчиках;

- о водителях;

- о топливе;

- о заказах;

- о машинах;

База данных создаётся для решения следующих задач:

- структурировать информацию о заказах;

- структурировать информацию о заказчиках.

Задачами данной курсовой работы являются:

- создание таблиц в базе данных;

- создание связей между таблицами в базе данных;

- создание запросов для в MS SQL Server;

- освоение паттерна Model View Controller

- освоение навыков программирование в среде C#;

- освоение технологии Entity Framework Core;

- освоение системы авторизации и аутентификации;

- реализация доступа клиента к серверу СУБД.

1.2 Сведения из теории

Так как сетевое приложение получает и использует большое количество данных, без систематизации данные не будет представлять какой-либо ценности. Поэтому в настоящее время системы управления базами данных (СУБД) используются для систематизации больших массивов данных, облегчения поиска по базе и внесения в них изменений. В СУБД поддерживаются различные модели данных.

Модель данных — это метод (принцип) логической организации данных, используемый СУБД. Наиболее известными являются иерархическая, сетевая и реляционная модели. Идея базы данных основана на таком привычном и простом способе записи информации - таблица. В самом деле, четкая и несложная структура (строки и столбцы) делает таблицу практически универсальным средством отображения информации любой природы.

C таблицей легко работать: добавлять новые данные; вычеркивать соответствующие строки, если информация уже не нужна; располагать строки в определенном порядке (например, по алфавиту, либо по коду) и т.д. Данные в таблицах могут иметь совершенно различную природу. Поля базы данных могут иметь один таких типов как: символьный (Char); числовые; дата (Date); логический (BOOL) и другие. В СУБД для персональных компьютеров (настольных СУБД) поддерживается преимущественно реляционная модель, которую отличает простота и единообразие представления данных простейшими двумерными таблицами.

Реляционная модель обеспечивает возможность использования в разных СУБД операций обработки данных, имеющих единую основу - алгебру отношений (реляционную алгебру), и универсального языка структурированных запросов SQL (Sequential Query Language). Реляционная база данных представляет собой множество взаимосвязанных двумерных таблиц - реляционных таблиц, называемых также отношениями, в каждой из которых содержатся сведения об одной сущности автоматизируемой предметной области.

Логическую структуру реляционной базы данных образует совокупность реляционных таблиц, между которыми установлены связи.

Реляционная таблица — это таблица, специальным образом организованная в виде файла. При переходе от таблицы к реляционной меняется терминология. Строка называется записью, Каждая запись состоит из полей (элементов столбцов). Для создания таблиц, соответствующих реляционной модели данных, используется процесс, называемый нормализацией данных, Нормализация – это удаление из таблиц повторяющихся данных путем их переноса в новые таблицы, записи которых не содержат повторяющихся значений.

Структура реляционной таблицы определяется составом полей. Каждое поле определяет определённую характеристику сущности. Таблицы состоят из строк и столбцов. Содержимое поля указывается в столбце таблицы, в которых сохраняются свойства объектов, которые отражены в таблице. Столбец содержит записи одного типа. Содержание таблицы заключается в её строках, однотипных по структуре, Каждая строка содержит данные о конкретном экземпляре сущности и называется записью.

После того, как выбран набор таблиц, необходимый для создания базы данных, определены поля таблиц, следует выбрать уникальные поля. Уникальные поля — это такие поля, в которых значения не могут повторяться, Поле, по значению которого можно идентифицировать запись в таблице, называется первичным ключом.

B нормализованной реляционной базе данных связь двух таблиц характеризуется отношениями записей типа один-к-одному, один-ко-многим, многие-к-многим. Отношение один-к-одному предполагает, что каждой записи одной таблицы соответствует одна запись в другой. Отношение один-ко-многим предполагает, что каждой записи первой таблицы соответствует много записей другой таблицы, но каждой записи второй таблицы соответствует только одна запись в первой. Соответственно, связь многие-к-многим - это связь многих записей одной таблицы ко многим записям другой.

Для двух таблиц, находящихся в отношении один-ко-многим, устанавливается связь по уникальному ключу таблицы, представляющей в отношении сторону «один», - главной таблицы в связи. Во второй таблице, представляющей в отношении сторону «многие» и называемой подчинённой, этот ключ связи может быть либо частью уникального ключа, либо не входить в состав ключа. В подчинённой таблице ключ связи называется ещё внешним ключом.

Запросы используются для выборки из базы данных информации, необходимой для пользователя. Под выборкой будем понимать динамическую таблицу с записями данных, которые удовлетворяют соответствующим условиям. Запросы обеспечивают быстрый и эффективный доступ к данным, хранящимся в таблице. Благодаря запросам можно не только выполнить сортировку или вычислить выражение, но и свести данные из связанных таблиц. Запросы не хранятся базой данных, информация храниться лишь в таблицах. Но полученная в результате запроса таблица может использоваться в качестве источника данных в формах отчётах и других запросах. Можно ввести поправки в запрос, которые отобразятся в соответствующей таблице. Целью данной базы данных является более быстрая и качественная обработка данных из таблиц.

Реляционная модель топливного гаража, описанного выше, представлена на рисунке 1.1

Diagram

Description automatically generated

Рисунок 1.1 – Реляционная модель топливного гаража

Реляционная база данных считается эффективной, если все её таблицы находятся как минимум в 3 нормальной форме. После проверки таблиц на соответствие требованиям 1НФ, 2НФ, 3НФ, можно сделать вывод, что все таблицы удовлетворяют требованиям. В них каждая запись есть отдельное независимое утверждение.

Современные веб-приложения используют системы аутентификации, авторизации, идентификации.

Идентификация — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе.

Аутентификация — процедура проверки подлинности, например проверка подлинности пользователя путем сравнения введенного им пароля с паролем, сохраненным в базе данных.

Авторизация — предоставление определенному лицу или группе лиц прав на выполнение определенных действий.

1.3 Постановка задачи

Название приложения – топливный гараж. Предоставляет услуги автомобильных перевозок топлива. Пользователи приложения делятся на три роли: администратор, водитель, заказчик. Администратор ведет учёт за заказами, имеет возможность редактировать заказы, водителей, заказчиков. Администратор может добавлять топливо, машины, водителей. Водитель отслеживает свои заказы. Заказчик может добавлять новые заказы и просматривать информацию о своих существующих заказах.

Входные данные для администратора:

- Топливо

- Машины

- Водители

- Пользователи

Выходные данные для администратора:

- Отчёт о продажах в Excel

- Таблица обо всех заказах

- Таблица водителей

- Таблица с топливом

- Таблица с машинами

Входные данные для заказчика:

- Заказ

Выходные данные для водителя:

- Таблица с заказами

Выходные данные для водителя:

- Таблица с заказами

Выбор среды и языка программирования. В качестве языка программирования был выбран C#. Для разработки веб-приложения была выбрана среда Visual Studio 2019 Community. Visual Studio Community - бесплатная полнофункциональная расширяемая среда IDE для создания современных приложений Android, iOS и Windows, а также веб-приложений и облачных служб. C# использует объектно-ориентированный подход к программированию во всем. Это означает, что надо описывать абстрактные конструкции на основе предметной области, а потом реализовывать между ними взаимодействие. Данный подход пользуется большой популярностью, потому что позволяет не держать в голове всю информацию, а работать по принципу черного ящика.

2 Технологии разработки приложения

2.1 Алгоритм решения

Алгоритм решения поставленных задач прост и для всех подзадач основан на одном и том же принципе:

– выбрать пункт из навигационного меню;

– нажать на необходимую ссылку (“Добавить”, “Удалить”, “ Изменить” или “Подробнее”);

– при выборе пунктов “Добавить”, “Изменить”, необходимо ввести нужные данные;

– нажать на кнопку “Save” или “Create”, либо нажать на ссылку “Изменить” или “Назад”;

– вывести результаты.

Например, алгоритм работы веб-страницы “Заказы”:

Добавление нового заказа:

– нажать на ссылку “Оформить новый заказ”;

– ввести необходимые данные;

– нажать кнопку “Create” для добавления топлива и возвращения на веб-страницу “Заказы” уже с обновленными данными.

Редактирование заказа:

– нажать на ссылку “Изменить”;

– изменить необходимые данные;

– нажать кнопку “Save” для возвращения на веб-страницу “Заказы” уже с измененными записями;

Удаление Заказа:

– нажать на ссылку “Удалить” для обновления веб-страницы “Заказы”;

Просмотр подробной информации о заказе:

– нажать на ссылку “Подробнее” открывается подробная информация заказа;

– нажать на кнопку “Save” для изменения необходимой информации;

– нажать на ссылку “Назад” для возвращения на веб-страницу “Заказы”;

2.2 Макет приложения

Веб-приложение на C# состоит из следующих слоёв: UI (User interface), Domain, Infrastructure.

Слой UI содержит контролеры, AutoMapper, расширения, представления.

Контроллеры представлены на рисунке 2.1

Text

Description automatically generated

Рисунок 2.1 – Контроллеры веб-приложения

AutoMapper представлен на рисунке 2.2

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 2.2 – AutoMapper веб-приложения

Представления представлены на рисунке 2.3

Text

Description automatically generated with medium confidence Graphical user interface, text

Description automatically generated with medium confidence

Рисунок 2.3 – Представления веб-приложения

Расширения представлены на рисунке 2.3

Text

Description automatically generated

Рисунок 2.3 – Представления веб-приложения

Здесь каждый класс отвечает за свою функцию:

- Контроллеры (AccountController.cs, AdminController.cs, CustomerController.cs, DriverController, HomeController.cs) служат для интерпретирования действия пользователя оповещая модель о необходимости изменения (Удаление, редактирование, подробнее, добавления). Для авторизации и аутентификации используется AccountController.cs контроллер.

- Класс AutoMapperProfile.cs используется, чтобы проецировать одну модель на другую, что позволяет сократить объемы кода и упростить программу.

- Класс DateExtensions.cs используется для преобразования даты в удобный формат.

- Класс ErrorHandlerMiddleware.cs используется для вывода исключения на представление.

- Представление Login.cshtml предназначено для вывода формы входа.

- Представление Register.cshtml предназначено для вывода формы регистрации.

- Представления, которые начинаются с Create, служат для отображения формы добавления, в которой добавляются новые записи в таблицы.

- Представления, которые начинаются с List, служат для вывода данных таблицы на html страницу.

- Представления Details служат для отображения более подробной информации, которой нет в List.

- Представления Edit служат для отображения формы изменения, в которой меняются данные полей таблицы.

Внешний вид программы на примере главной веб-страницы (Рисунок 2.4)

Graphical user interface, text, application

Description automatically generated

Рисунок 2.4 – Главная страница веб-приложения.

Страница входа в веб-приложения (Рисунок 2.5)

Graphical user interface

Description automatically generated

Рисунок 2.5 – Страница входа

Страница регистрации веб-приложения (Рисунок 2.6)

Graphical user interface

Description automatically generated

Рисунок 2.6 – Страница регистрации

Слой Domain содержит сущности и модели представления.

Сущности представлены на рисунке 2.7

A screen shot of a computer

Description automatically generated with low confidence

Рисунок 2.7 – Сущности веб-приложения

Модели представления представлены на рисунке 2.8

Text

Description automatically generated

Рисунок 2.8 – Модели представления веб-приложения

- Сущности используются для сопоставления таблиц из базы данных.

- Модели представления используются как слой абстракции от реализации UI.

Слой Infrastructure содержит контекст базы данных, сервисы.

Контекст базы данных представлен на рисунке 2.9

Text

Description automatically generated with medium confidence

Рисунок 2.9 – Контекст базы данных

Сервисы представлены на рисунке 2.10

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 2.10 – Сервисы веб-приложения

- Класс контекста базы данных позволяет создавать и выполнять запросы, отслеживать изменения в объектах и отображать эти изменения на базу данных.

- Сервисы FuelService.cs, OrderService.cs, StatusService.cs, UserService.cs, VehicleService.cs служат для обработки данных.

- Интерфейсы IFuelService.cs, IOrderService.cs, IUserService.cs, IVehicleService.cs, IService.cs служат для реализации паттерна “Репозиторий”. “Репозиторий” позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

2.3 Описание программы

Приложение реализует паттерн MVC:

Модель (model): описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных.

Представление (view): отвечают за визуальную часть или пользовательский интерфейс, нередко html-страница, через который пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных.

Контроллер (controller): представляет центральный компонент MVC, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

Имплементация сервиса OrderService представлена в листинге 1

using ClosedXML.Excel;

using FuelGarage.Domain.Entities;

using FuelGarage.Domain.ViewModels;

using FuelGarage.Infrastructure.Db;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

namespace FuelGarage.Infrastructure.Services.Orders

{

public class OrderService : IOrderService

{

private readonly GarageContext \_dbContext;

public OrderService(GarageContext dbContext)

{

\_dbContext = dbContext;

}

public void EditFuel(int id, int fuelCount)

{

var order = GetById(id);

order.FuelQuantity = fuelCount;

\_dbContext.Orders.Update(order);

\_dbContext.SaveChanges();

}

public void Create(Order order)

{

\_dbContext.Add(order);

\_dbContext.SaveChanges();

}

public void Delete(int id)

{

var order = \_dbContext.Orders.FirstOrDefault(x => x.Id == id);

\_dbContext.Remove(order);

\_dbContext.SaveChanges();

}

public void Edit(Order order)

{

\_dbContext.Orders.Update(order);

\_dbContext.SaveChanges();

}

public List<Order> GetAll()

{

var orders = \_dbContext.Orders

.Include(x => x.Status)

.Include(x => x.Fuel)

.Include(x => x.Customer)

.Include(x => x.Driver)

.AsNoTracking()

.ToList();

return orders;

}

public Order GetById(int id)

{

return \_dbContext.Orders.Where(x => x.Id == id)

.Include(x => x.Customer)

.Include(x => x.Driver)

.Include(x => x.Fuel)

.Include(x => x.Status)

.AsNoTracking()

.FirstOrDefault();

}

public List<Order> GetByCustomerId(int id)

{

return \_dbContext.Orders

.Include(x => x.Customer)

.Include(x => x.Driver)

.Include(x => x.Fuel)

.Include(x => x.Status)

.AsNoTracking()

.Where(x => x.CustomerId == id).ToList();

}

public void EditStatusById(int id, int statusId, Status status)

{

var order = GetById(id);

order.Status = status;

\_dbContext.Orders.Update(order);

\_dbContext.SaveChanges();

}

public XLWorkbook GenerateExcelReport(Report report)

{

var orders = GetAll();

if (report.ClientId != 0)

{

orders = orders

.Where(x => x.CustomerId == report.ClientId && x.LeadTime >= report.From && x.LeadTime <= report.To)

.ToList();

}

else

{

orders = orders

.Where(x => x.LeadTime >= report.From && x.LeadTime <= report.To)

.ToList();

}

var workbook = new XLWorkbook();

var worksheet = workbook.Worksheets.Add("Заказы");

var currentRow = 1;

worksheet.Cell(currentRow, 1).Value = "Имя заказчика";

worksheet.Cell(currentRow, 2).Value = "Фамилия заказчика";

worksheet.Cell(currentRow, 3).Value = "Отчество заказчика";

worksheet.Cell(currentRow, 4).Value = "Телефон заказчика";

worksheet.Cell(currentRow, 5).Value = "Адрес";

worksheet.Cell(currentRow, 6).Value = "Дата оформления";

worksheet.Cell(currentRow, 7).Value = "Дата доставки";

worksheet.Cell(currentRow, 8).Value = "Топливо";

worksheet.Cell(currentRow, 9).Value = "Количество";

worksheet.Cell(currentRow, 10).Value = "Имя водителя";

worksheet.Cell(currentRow, 11).Value = "Фамилия водителя";

worksheet.Cell(currentRow, 12).Value = "Отчество водителя";

worksheet.Cell(currentRow, 13).Value = "Телефон водителя";

foreach (var order in orders)

{

currentRow++;

worksheet.Cell(currentRow, 1).Value = order.Customer?.FirstName ?? "нет данных";

worksheet.Cell(currentRow, 2).Value = order.Customer?.LastName ?? "нет данных";

worksheet.Cell(currentRow, 3).Value = order.Customer?.MiddleName ?? "нет данных";

worksheet.Cell(currentRow, 4).Value = order.Customer?.Phone ?? "нет данных";

worksheet.Cell(currentRow, 5).Value = order.OrderAddress ?? "нет данных";

worksheet.Cell(currentRow, 6).Value = order.ApplicationTime;

worksheet.Cell(currentRow, 7).Value = order.LeadTime;

worksheet.Cell(currentRow, 8).Value = order.Fuel.Brand;

worksheet.Cell(currentRow, 9).Value = order.FuelQuantity;

worksheet.Cell(currentRow, 10).Value = order.Driver?.FirstName ?? "нет данных";

worksheet.Cell(currentRow, 11).Value = order.Driver?.LastName ?? "нет данных";

worksheet.Cell(currentRow, 12).Value = order.Driver?.MiddleName ?? "нет данных";

worksheet.Cell(currentRow, 13).Value = order.Driver?.Phone ?? "нет данных";

}

return workbook;

}

}

}

Листинг 1 – Код сервиса OrderService.cs

Пример реализации контроллера CustomerController.cs представлен в листинге 2.

using AutoMapper;

using FuelGarage.Domain.Entities;

using FuelGarage.Domain.Enums;

using FuelGarage.Domain.ViewModels;

using FuelGarage.Infrastructure.Services.Fuels;

using FuelGarage.Infrastructure.Services.Orders;

using FuelGarage.Infrastructure.Services.Users;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Security.Claims;

namespace FuelGarage.Controllers

{

[Authorize(Roles = "customer")]

public class CustomerController : Controller

{

private readonly IOrderService \_orderService;

private readonly IUserService \_userService;

private readonly IFuelService \_fuelService;

private readonly IMapper \_mapper;

public CustomerController(

IOrderService orderService,

IUserService userService,

IFuelService fuelService,

IMapper mapper)

{

\_orderService = orderService;

\_userService = userService;

\_fuelService = fuelService;

\_mapper = mapper;

}

[HttpGet]

public IActionResult Index()

{

var email = User.FindFirst(x => x.Type == ClaimsIdentity.DefaultNameClaimType).Value;

var user = \_userService.GetByEmail(email);

var orders = \_orderService.GetByCustomerId(user.Id);

var orderViewModels = \_mapper.Map<IEnumerable<CustomerOrderViewModel>>(orders);

return View(orderViewModels);

}

[HttpGet]

public IActionResult CreateOrder()

{

var fuels = \_fuelService.GetAll();

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

return View();

}

[HttpPost]

public IActionResult CreateOrder(Order model)

{

if (!\_fuelService.EraseFuel(model.FuelId, model.FuelQuantity) || model.FuelQuantity <= 0)

{

ModelState.AddModelError("", "У нас нет столько топлива :-(");

var fuels = \_fuelService.GetAll();

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

return View();

}

if (model.LeadTime < DateTime.Now)

{

ModelState.AddModelError("", "Укажите корректную дату доставки");

var fuels = \_fuelService.GetAll();

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

return View();

}

model.ApplicationTime = DateTime.Now;

var email = User.FindFirst(x => x.Type == ClaimsIdentity.DefaultNameClaimType).Value;

var user = \_userService.GetByEmail(email);

model.StatusId = (int)StatusType.Open;

model.CustomerId = user.Id;

\_orderService.Create(model);

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult DeleteOrder(int id)

{

\_orderService.Delete(id);

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult EditOrder(int id)

{

var model = \_orderService.GetById(id);

var fuels = \_fuelService.GetAll();

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

return View(model);

}

[HttpPost]

public IActionResult EditOrder(Order model)

{

var fuelCurrent = \_orderService.GetById(model.Id).FuelQuantity;

if (!\_fuelService.EditFuelFromCustomer(model.FuelId, model.FuelQuantity, fuelCurrent) || model.FuelQuantity <= 0)

{

var fuels = \_fuelService.GetAll();

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

ModelState.AddModelError("", "У нас нет столько топлива :-(");

ViewBag.Fuels = new SelectList(fuels, "Id", "Brand");

return View(model);

}

model.StatusId = (int)StatusType.Open;

\_orderService.Edit(model);

return RedirectToAction("index");

}

[HttpGet]

public IActionResult DetailsOrder(int id)

{

var model = \_orderService.GetById(id);

var orderViewModel = \_mapper.Map<CustomerOrderViewModel>(model);

return View(orderViewModel);

}

}

}

Листинг 2 – Код контроллера CustomerController.cs

Реализация модели Order представлена на листинге 3

using System;

namespace FuelGarage.Domain.Entities

{

public class Order

{

public int Id { get; set; }

public int? DriverId { get; set; }

public int? CustomerId { get; set; }

public int FuelId { get; set; }

public int FuelQuantity { get; set; }

public int StatusId { get; set; }

public string OrderAddress { get; set; }

public string OrderDescription { get; set; }

public DateTime ApplicationTime { get; set; }

public DateTime LeadTime { get; set; }

public Fuel Fuel { get; set; }

public Status Status { get; set; }

public User Driver { get; set; }

public User Customer { get; set; }

}

}

Листинг 3 – Код модели Order.cs

Реализацию представлений рассмотрим на примере добавления для модели Order на листинге 4.

@model FuelGarage.Domain.Entities.Order

@{

ViewData["Title"] = "Оформить заказ";

Layout = "\_CustomerLayout";

}

<h1>Оформить заказ</h1>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="CreateOrder">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label class="control-label">Топливо</label>

<select asp-for="FuelId" asp-items="ViewBag.Fuels"></select>

<span asp-validation-for="FuelId" class="text-danger"></span>

</div>

<div class="form-group">

<label class="control-label">Количество, л.</label>

<input asp-for="FuelQuantity" class="form-control" />

<span asp-validation-for="FuelQuantity" class="text-danger"></span>

</div>

<div class="form-group">

<label class="control-label">Адрес доставки</label>

<input asp-for="OrderAddress" class="form-control" />

<span asp-validation-for="OrderAddress" class="text-danger"></span>

</div>

<div class="form-group">

<label class="control-label">Описание</label>

<input asp-for="OrderDescription" class="form-control" />

<span asp-validation-for="OrderDescription" class="text-danger"></span>

</div>

<div class="form-group">

<label class="control-label">Дата доставки</label>

<input asp-for="LeadTime" class="form-control" />

<span asp-validation-for="LeadTime" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Назад</a>

</div>

@section Scripts {

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

}

Листинг 4 – Код представления CreateOrder.cshtml

2.4 Результаты работы программы

Входные данные формируются заказчиком при оформлении нового заказа.

Для отбора нужных заказов в форме загрузки отчёта вводятся данные о клиенте, начальная и конечная дата оформления заказов (Рисунок 2.11).

Graphical user interface, text, application

Description automatically generated

Рисунок 2.11 – Входные данные для формирования отчета

Результатом работы программы является вывод накладной на печать. Входными данными являются данные о заказе, водителе, топливе, заказчике. Исходя из этих данных формируется накладная, который выводится в Excel, в котором его можно будет распечатать. Накладная представлен на рисунке 2.12.

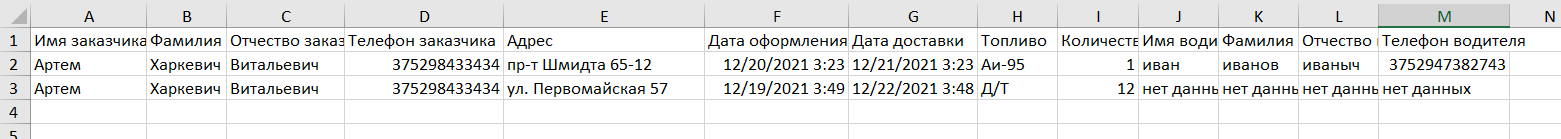


Рисунок 2.12 – Накладная на печать.

3. Руководство пользователя

Заключение

В ходе работы над курсовым проектом я отточил навыки работы с клиентской и серверной частью, а именно работой с MVC паттерном в языке программирования C#, javascript, css, технологией Entity Framework Core. Серверная часть приложения была написана на языке C# с использованием MVC.

В ходе выполнения курсового проекта были закреплены понятия объектно-ориентированного программирования, а также работой с пользовательским интерфейсом.

Результатом курсового проекта стало веб-приложение с использованием веб-интерфейса. Интерфейс созданного приложения удобен, прост и интуитивно понятен для всех пользователей. После запуска приложения пользователь может посмотреть уже существующие данные, изменить их, удалить, посмотреть подробную информацию, а также добавить новые данные.

Список использованных источников

1. Справочный материал MSDN. Microsoft [Электронный ресурс] — 2021. —Режим доступа: <https://msdn.microsoft.com/library>.
2. Справочный материал w3schools.com [Электронный ресурс] — 2021. — Режим доступа: [https://www.w3schools.com](https://www.w3schools.com/).
3. Сайт о программировании Habrahabr.ru [Электронный ресурс] — 2021 — Режим доступа: [http://habrahabr.ru](http://habrahabr.ru/).
4. Сайт о программировании Stackoverflow.com [Электронный ресурс] — 2021. — Режим доступа: [https://stackoverflow.com](https://stackoverflow.com/).
5. Сайт о программировании Metanit.com [Электронный ресурс] — 2021. — Режим доступа: <https://metanit.com>.
6. Шилдт, Г. C# 4.0. Полное руководство : пер. с англ. / Г. Шилдт. – М. : Вильямс, 2011. – 1056 с.